The background of the entire cover is a close-up, high-resolution image of a wood grain. The grain lines are curved and flow from the top left towards the bottom right, creating a sense of movement and depth. The colors range from light tan to deep, rich browns, with some darker, almost black, lines interspersed.

# WPROWADZENIE DO SYSTEMÓW BAZ DANYCH

Wydanie VII

ELMASRI • NAVATHE

**Helion** 

Tytuł oryginału: Fundamentals of Database Systems (7th Edition)

Tłumaczenie: Tomasz Walczak

z wykorzystaniem fragmentów „Wprowadzenie do systemów baz danych”

w przekładzie Bartłomieja Garbacza, Bartłomieja Moczulskiego i Mikołaja Szczepaniaka

ISBN: 978-83-283-4696-3

Authorized translation from the English language edition, entitled:

FUNDAMENTALS OF DATABASE SYSTEMS, Seventh Edition, ISBN 0133970779;

by Ramez Elmasri; and by Shamkant B. Navathe; published by Pearson Education, Inc.

Copyright © 2016, 2011, 2007 by Ramez Elmasri and Shamkant B. Navathe.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc. Polish language edition published by HELION SA, Copyright © 2019.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione.

Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Helion SA dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Helion SA nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion SA

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 231 22 19, 32 230 98 63

e-mail: [helion@helion.pl](mailto:helion@helion.pl)

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

[http://helion.pl/user/opinie/wpisy7\\_ebook](http://helion.pl/user/opinie/wpisy7_ebook)

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

- [Poleć książkę na Facebook.com](#)
- [Kup w wersji papierowej](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to!](#) » [Nasza społeczność](#)

*Dla Amelii  
oraz  
Ramy, Riada, Katriny i Thomasa  
R. E.*

*Dla mojej żony Aruny za jej miłość, wsparcie i zrozumienie  
oraz  
dla Rohana, Mai i Ayusha za to, że wnieśli tyle radości w nasze życie  
S. B. N.*





---

# Spis treści

<b>Przedmowa</b>	<b>25</b>
<b>O autorach</b>	<b>33</b>
<b>I. Wprowadzenie do baz danych</b>	<b>35</b>
<b>1. Bazy danych i ich użytkownicy</b>	<b>37</b>
1.1. Wprowadzenie	38
1.2. Przykład	41
1.3. Właściwości rozwiązań opartych na bazach danych	44
1.3.1. Samoopisująca natura systemów baz danych	45
1.3.2. Oddzielenie programów od danych oraz abstrakcja danych	46
1.3.3. Obsługa wielu perspektyw dla tych samych danych	48
1.3.4. Współdzielenie danych oraz wielodostępne przetwarzanie transakcji	49
1.4. Aktorzy na scenie	50
1.4.1. Administratorzy bazy danych	50
1.4.2. Projektanci bazy danych	50
1.4.3. Użytkownicy końcowi	51
1.4.4. Analitycy systemowi i programiści aplikacji (inżynierowie oprogramowania)	52
1.5. Pracownicy poza sceną	52
1.6. Zalety stosowania rozwiązań opartych na systemach zarządzania bazami danych	53
1.6.1. Kontrola nadmiarowości	53
1.6.2. Ograniczanie możliwości uzyskania nieautoryzowanego dostępu	55
1.6.3. Zapewnianie miejsca trwałego przechowywania dla obiektów stosowanych w programach	56
1.6.4. Zapewnianie struktur przechowywania dla efektywnego przetwarzania zapytań	56
1.6.5. Zapewnianie możliwości tworzenia kopii bezpieczeństwa i odzyskiwania danych	57
1.6.6. Zapewnianie interfejsów dla wielu użytkowników	57
1.6.7. Reprezentowanie skomplikowanych relacji pomiędzy danymi	57
1.6.8. Wymuszanie więzów integralności	58
1.6.9. Zezwalanie na wnioskowanie i podejmowanie działań w oparciu o zdefiniowane reguły	59

1.6.10. Dodatkowe własności wynikające ze stosowania rozwiązań opartych na bazach danych	59
1.7. Krótka historia praktycznych zastosowań baz danych	60
1.7.1. Wczesne zastosowania baz danych oparte na systemach hierarchicznych i sieciowych	60
1.7.2. Zapewnianie elastyczności w rozwiązaniach opartych na relacyjnych bazach danych	61
1.7.3. Aplikacje obiektowe i konieczność wprowadzenia bardziej skomplikowanych baz danych	62
1.7.4. Wykorzystywana w handlu elektronicznym wymiana danych za pośrednictwem internetu z użyciem XML-a	62
1.7.5. Rozszerzanie możliwości współczesnych systemów baz danych z myślą o nowych zastosowaniach	63
1.7.6. Powstanie systemów przechowywania big data i baz NOSQL	64
1.8. Kiedy nie należy używać systemów zarządzania bazami danych	64
1.9. Podsumowanie	65
Pytania powtórkowe	66
Ćwiczenia	66
Wybrane publikacje	67
<b>2. Architektura systemów baz danych i związane z nimi pojęcia</b>	<b>69</b>
2.1. Modele danych, schematy i egzemplarze	70
2.1.1. Kategorie modeli danych	71
2.1.2. Schematy, egzemplarze i stany baz danych	72
2.2. Trójwarstwowa architektura i niezależność danych	74
2.2.1. Architektura trójwarstwowa	74
2.2.2. Niezależność danych	76
2.3. Języki i interfejsy baz danych	77
2.3.1. Języki systemów zarządzania bazami danych	77
2.3.2. Interfejsy systemów zarządzania bazami danych	80
2.4. Środowisko systemu bazy danych	82
2.4.1. Moduły składające się na system zarządzania bazą danych	82
2.4.2. Narzędzia systemu bazy danych	85
2.4.3. Narzędzia, środowiska aplikacji oraz mechanizmy komunikacji	86
2.5. Architektury systemów zarządzania bazami danych	
— scentralizowane i typu klient-serwer	87
2.5.1. Scentralizowane architektury systemów zarządzania bazami danych	87
2.5.2. Podstawowe architektury typu klient-serwer	87
2.5.3. Dwuwarstwowe architektury typu klient-serwer dla systemów zarządzania bazami danych	90
2.5.4. Trójwarstwowe i n-warstwowe architektury typu klient-serwer dla aplikacji internetowych	90
2.6. Klasyfikacja systemów zarządzania bazami danych	92
2.7. Podsumowanie	95
Pytania powtórkowe	96
Ćwiczenia	97
Wybrane publikacje	97

## II. Konceptyjne modelowanie danych i projektowanie baz danych

99

<b>3. Modelowanie danych zgodnie z modelem związków encji</b>	<b>101</b>
3.1. Stosowanie wysokopoziomowych, konceptyjnych modeli danych podczas projektowania bazy danych	103
3.2. Przykładowa aplikacja bazy danych	105
3.3. Typy encji, zbiory encji, atrybuty i klucze	106
3.3.1. Encje i atrybuty	107
3.3.2. Typy encji, zbiory encji, klucze i zbiory wartości	109
3.3.3. Początkowy projekt konceptyjny bazy danych FIRMA	113
3.4. Typy związków, zbiory związków, role i ograniczenia strukturalne	114
3.4.1. Typy, zbiory i egzemplarze związków	115
3.4.2. Stopień związku, nazwy ról oraz związki rekurencyjne	115
3.4.3. Ograniczenia dla typów związków	118
3.4.4. Atrybuty typów związków	121
3.5. Słabe typy encji	122
3.6. Udoskonalanie projektu ER dla bazy danych FIRMA	123
3.7. Diagramy ER, konwencje nazewnictwa oraz zagadnienia związane z projektowaniem	124
3.7.1. Podsumowanie notacji diagramów związków encji (ER)	124
3.7.2. Prawidłowe nazewnictwo konstrukcji schematu	125
3.7.3. Decyzje projektowe związane z tworzeniem schematu konceptyjnego ER	127
3.7.4. Notacje alternatywne względem tradycyjnych diagramów związków encji (ER)	128
3.8. Przykładowa inna notacja: diagramy klas UML	129
3.9. Typy związków stopnia wyższego niż drugi	132
3.9.1. Wybór pomiędzy związkami binarnymi a trójskładnikowymi (lub wyższych stopni)	132
3.9.2. Ograniczenia związków trójskładnikowych (i wyższych stopni)	136
3.10. Inny przykład — baza danych UNIWERSYTET	137
3.11. Podsumowanie	139
Pytania powtórkowe	140
Ćwiczenia	141
Ćwiczenia laboratoryjne	147
Wybrane publikacje	149
<b>4. Rozszerzony model związków encji</b>	<b>151</b>
4.1. Podklasy, nadklasy i dziedziczenie	152
4.2. Specjalizacja i generalizacja	154
4.2.1. Specjalizacja	154
4.2.2. Generalizacja	155
4.3. Ograniczenia i właściwości hierarchii specjalizacji i generalizacji	157
4.3.1. Ograniczenia dotyczące specjalizacji i generalizacji	157
4.3.2. Hierarchie i kraty specjalizacji i generalizacji	160
4.3.3. Stosowanie procesów specjalizacji i generalizacji podczas udoskonalania schematów konceptyjnych	163
4.4. Modelowanie typów UNII w oparciu o kategorie	164
4.5. Przykład schematu EER dla bazy danych UNIWERSYTET oraz formalne definicje dla modelu EER	166
4.5.1. Inny przykład bazy danych UNIWERSYTET	166
4.5.2. Wybory projektowe związane ze specjalizacją i generalizacją	169
4.5.3. Formalne definicje pojęć stosowanych w modelu EER	170

4.6. Przykładowa inna notacja: reprezentowanie specjalizacji-generalizacji na diagramach klas języka UML	171
4.7. Abstrakcja danych, reprezentacja wiedzy oraz zagadnienia związane z ontologią	173
4.7.1. Klasyfikacja i tworzenie egzemplarzy	174
4.7.2. Identyfikacja	175
4.7.3. Specjalizacja i generalizacja	176
4.7.4. Agregacja i asocjacja	176
4.7.5. Ontologia i sieć semantyczna	178
4.8. Podsumowanie	179
Pytania powtórkowe	180
Ćwiczenia	181
Ćwiczenia laboratoryjne	188
Wybrane publikacje	190

### III. Relacyjny model danych i SQL 193

<b>5. Relacyjny model danych i ograniczenia relacyjnych baz danych</b>	<b>195</b>
5.1. Pojęcia z modelu relacyjnego	196
5.1.1. Dziedziny, atrybuty, krotki i relacje	197
5.1.2. Właściwości relacji	199
5.1.3. Notacja modelu relacyjnego	203
5.2. Ograniczenia modelu relacyjnego i schematy relacyjnych baz danych	203
5.2.1. Ograniczenia dziedziny	204
5.2.2. Ograniczenia klucza i ograniczenia wartości pustych	205
5.2.3. Relacyjne bazy danych i schematy relacyjnych baz danych	206
5.2.4. Integralność encji, integralność odwołań i klucze obce	209
5.2.5. Pozostałe typy ograniczeń	210
5.3. Operacje aktualizacji, transakcje i obsługa naruszeń więzów integralności	212
5.3.1. Operacja wstawiania	212
5.3.2. Operacja usuwania	214
5.3.3. Operacja aktualizacji	215
5.3.4. Transakcje	216
5.4. Podsumowanie	216
Pytania powtórkowe	217
Ćwiczenia	218
Wybrane publikacje	222
<b>6. Podstawy języka SQL</b>	<b>223</b>
6.1. Definicje danych i typy danych języka SQL	225
6.1.1. Stosowane w języku SQL pojęcia schematu i katalogu	226
6.1.2. Polecenie CREATE TABLE języka SQL	226
6.1.3. Typy danych atrybutów oraz dziedziny wartości w standardzie SQL	229
6.2. Określanie ograniczeń w języku SQL	231
6.2.1. Definiowanie ograniczeń i wartości domyślnych dla atrybutów	232
6.2.2. Definiowanie ograniczeń klucza i więzów integralności odwołań	233
6.2.3. Nadawanie nazw definiowanym ograniczeniom	235
6.2.4. Stosowanie klauzuli CHECK do określania ograniczeń dla krotek	235
6.3. Podstawowe zapytania języka SQL	236
6.3.1. Struktura podstawowych zapytań języka SQL: SELECT-FROM-WHERE	236
6.3.2. Niejednoznaczne nazwy atrybutów, mechanizm nazw zastępczych (aliasów) oraz zmienne krotek	239
6.3.3. Nieokreślona klauzula WHERE i zastosowania symbolu gwiazdki	241



6.3.4. Tabele i zbiory w języku SQL	242
6.3.5. Dopasowywanie podciągów znaków do wzorca oraz operacje arytmetyczne	244
6.3.6. Sortowanie wyników zapytań	246
6.3.7. Omówienie i podsumowanie prostych zapytań języka SQL	246
6.4. Dostępne w języku SQL polecenia INSERT, DELETE i UPDATE	247
6.4.1. Polecenie INSERT	247
6.4.2. Polecenie DELETE	249
6.4.3. Polecenie UPDATE	250
6.5. Dodatkowe własności języka SQL	250
6.6. Podsumowanie	252
Pytania powtórkowe	252
Ćwiczenia	253
Wybrane publikacje	255

## **7. Jeszcze o języku SQL — złożone zapytania, wyzwalacze, perspektywy i modyfikowanie schematów** **257**

7.1. Bardziej skomplikowane zapytania języka SQL pobierające dane	257
7.1.1. Operacje porównania z wartością pustą (NULL) oraz logika trójwartościowa	258
7.1.2. Zapytania zagnieżdżone, krotki oraz porównywanie zbiorów i wielozbiorów	260
7.1.3. Zagnieżdżone zapytania skorelowane	262
7.1.4. Dostępne w języku SQL funkcje EXISTS i UNIQUE	263
7.1.5. Jawne deklarowanie zbiorów i zmienianie nazw atrybutów w języku SQL	265
7.1.6. Tabele połączone w języku SQL	266
7.1.7. Funkcje agregujące w języku SQL	268
7.1.8. Grupowanie: klauzule GROUP BY i HAVING	270
7.1.9. Inne konstrukcje języka SQL: WITH i CASE	273
7.1.10. Zapytania rekurencyjne w języku SQL	274
7.1.11. Omówienie i podsumowanie zapytań języka SQL	275
7.2. Definiowanie ograniczeń w postaci asercji i działań w postaci wyzwalaczy	277
7.2.1. Definiowanie ogólnych ograniczeń w postaci asercji w języku SQL	277
7.2.2. Wprowadzenie do wyzwalaczy w języku SQL	278
7.3. Perspektywy (tabele wirtualne) w języku SQL	280
7.3.1. Pojęcie perspektywy w języku SQL	280
7.3.2. Definiowanie perspektyw w języku SQL	280
7.3.3. Implementacja perspektyw i mechanizm ich aktualizowania	282
7.3.4. Perspektywy jako mechanizm uwierzytelniania	284
7.4. Dostępne w języku SQL polecenia zmiany schematu	285
7.4.1. Polecenie DROP	285
7.4.2. Polecenie ALTER	286
7.5. Podsumowanie	287
Pytania powtórkowe	289
Ćwiczenia	289
Wybrane publikacje	291

## **8. Algebra relacyjna i rachunek relacji** **293**

8.1. Relacyjne operacje unarne: selekcja i projekcja	295
8.1.1. Operacja selekcji	295
8.1.2. Operacja projekcji	297
8.1.3. Sekwencje operacji i operacja ZMIANA NAZWY	299
8.2. Operacje algebry relacyjnej pochodzące z teorii zbiorów	301
8.2.1. Operacje sumy, części wspólnej i różnicy	301
8.2.2. Operacja iloczynu (produktu) kartezjańskiego	303

8.3.	Binarne operacje na relacjach: złączenie i dzielenie	305
8.3.1.	Operacja złączenia	305
8.3.2.	Odmiany operacji złączenia: operacje równo-złączenia i złączenia naturalnego	307
8.3.3.	Kompletny zbiór operacji algebry relacyjnej	309
8.3.4.	Operacja dzielenia	310
8.3.5.	Notacja drzew zapytań	313
8.4.	Dodatkowe operacje relacyjne	314
8.4.1.	Uogólniona projekcja	314
8.4.2.	Funkcje agregujące i mechanizm grupowania	314
8.4.3.	Rekurencyjne operacje domknięcia	316
8.4.4.	Operacje złączenia zewnętrznego	317
8.4.5.	Operacja sumy zewnętrznej	319
8.5.	Przykłady zapytań w algebrze relacyjnej	320
8.6.	Relacyjny rachunek krotek	323
8.6.1.	Zmienne krotek i relacje zakresowe	324
8.6.2.	Wyrażenia i wzory w relacyjnym rachunku krotek	325
8.6.3.	Kwantyfikatory uniwersalne i egzystencjalne	327
8.6.4.	Przykładowe zapytania w relacyjnym rachunku krotek	328
8.6.5.	Notacja używana dla grafów zapytań	329
8.6.6.	Wzajemne przekształcanie kwantyfikatorów uniwersalnych i egzystencjalnych	330
8.6.7.	Stosowanie kwantyfikatorów uniwersalnych w zapytaniach	331
8.6.8.	Bezpieczne wyrażenia	333
8.7.	Relacyjny rachunek dziedzin	334
8.8.	Podsumowanie	336
	Pytania powtórkowe	338
	Ćwiczenia	338
	Ćwiczenia laboratoryjne	343
	Wybrane publikacje	346

## **9. Projektowanie relacyjnych baz danych przez odwzorowywanie modelu ER i EER w model relacyjny**

**347**

9.1.	Projektowanie relacyjnych baz danych w oparciu o odwzorowywanie modelu ER w model relacyjny	348
9.1.1.	Algorytm odwzorowujący model ER w model relacyjny	348
9.1.2.	Omówienie i podsumowanie odwzorowania konstrukcji modelu ER w odpowiednie konstrukcje modelu relacyjnego	355
9.2.	Odwzorowania konstrukcji modelu EER w relacje	357
9.2.1.	Odwzorowywanie specjalizacji i generalizacji	357
9.2.2.	Odwzorowywanie współdzielonych podklas (konstrukcji dziedziczenia wielokrotnego)	360
9.2.3.	Odwzorowywanie kategorii (typów unii)	361
9.3.	Podsumowanie	361
	Pytania powtórkowe	362
	Ćwiczenia	362
	Ćwiczenia laboratoryjne	364
	Wybrane publikacje	365

## IV. Techniki programowania baz danych

367

<b>10. Wprowadzenie do technik programowania w języku SQL</b>	<b>369</b>
10.1. Przegląd technik i zagadnień z obszaru programowania baz danych	370
10.1.1. Strategie programowania baz danych	371
10.1.2. Niezgodność impedancji	372
10.1.3. Typowa sekwencja operacji składających się na interakcję w programowaniu baz danych	373
10.2. Osadzony język SQL, dynamiczny język SQL oraz język SQLJ	374
10.2.1. Wyszukiwanie pojedynczych krotek za pomocą poleceń osadzonego języka SQL	375
10.2.2. Przetwarzanie wyników zapytań za pomocą kursorów	379
10.2.3. Określanie zapytań w czasie wykonywania programu — stosowanie dynamicznego języka SQL	381
10.2.4. SQLJ: osadzanie poleceń języka SQL w języku Java	383
10.2.5. Używanie iteratorów do przetwarzania wyników zapytań w standardzie SQLJ	385
10.3. Programowanie baz danych z wywołaniami funkcji i bibliotekami klas: SQL/CLI oraz JDBC	388
10.3.1. Programowanie baz danych z wykorzystaniem interfejsu SQL/CLI oraz języka C w roli nadrzędnego języka programowania	389
10.3.2. JDBC: biblioteka klas języka SQL służąca do programowania w języku Java	393
10.4. Procedury składowane w bazie danych i technika SQL/PSM	398
10.4.1. Procedury i funkcje składowane w bazie danych	399
10.4.2. SQL/PSM: Rozszerzenie standardu SQL o możliwość określania trwale składowanych modułów	400
10.5. Porównanie trzech opisanych podejść	402
10.6. Podsumowanie	403
Pytania powtórkowe	403
Ćwiczenia	404
Wybrane publikacje	404
<b>11. Programowanie internetowych baz danych z użyciem języka PHP</b>	<b>405</b>
11.1. Prosty przykład zastosowania PHP	406
11.2. Przegląd podstawowych mechanizmów języka PHP	408
11.2.1. Zmienne, typy danych i konstrukcje programistyczne języka PHP	409
11.2.2. Tablice w PHP	410
11.2.3. Funkcje w języku PHP	412
11.2.4. Zmienne i formularze serwera PHP	414
11.3. Przegląd programowania baz danych za pomocą PHP	415
11.3.1. Nawiązywanie połączenia z bazą danych	415
11.3.2. Pobieranie danych z formularzy i wstawianie rekordów	417
11.3.3. Zapytania pobierające dane z tabel bazy	418
11.4. Krótki przegląd technologii programowania internetowych baz danych w Javie	419
11.5. Podsumowanie	420
Pytania powtórkowe	420
Ćwiczenia	421
Wybrane publikacje	421

## V. Podejścia obiektowe, obiektowo-relacyjne i XML: zagadnienia, modele, języki i standardy

423

<b>12. Bazy obiektowe i obiektowo-relacyjne</b>	<b>425</b>
12.1. Przegląd pojęć obiektowych	427
12.1.1. Wprowadzenie do pojęć i cech obiektowych	427
12.1.2. Tożsamość obiektów i porównanie obiektów z literałami	429
12.1.3. Złożone struktury typów obiektów i literałów	430
12.1.4. Enkapsulacja operacji i trwałość obiektów	432
12.1.5. Hierarchia typów i dziedziczenie	436
12.1.6. Inne pojęcia obiektowe	438
12.1.7. Podsumowanie zagadnień dotyczących obiektowych baz danych	440
12.2. Rozszerzenia obiektowe w standardzie SQL	440
12.2.1. Typy definiowane przez użytkownika za pomocą polecenia CREATE TYPE i obiekty złożone	443
12.2.2. Identyfikatory obiektów oparte na odwołaniach	444
12.2.3. Tworzenie tabel z wykorzystaniem UDT	444
12.2.4. Enkapsulacja operacji	445
12.2.5. Dziedziczenie i przeciążanie funkcji	446
12.2.6. Określanie związków za pomocą odwołań	446
12.3. Model obiektowy ODMG i język definiowania obiektów ODL	447
12.3.1. Przegląd modelu obiektowego ODMG	448
12.3.2. Dziedziczenie w modelu obiektowym ODMG	453
12.3.3. Wbudowane interfejsy i klasy w modelu obiektowym	454
12.3.4. Obiekty atomowe (definiowane przez użytkownika)	456
12.3.5. Ekstensje, klucze i obiekty-fabryki	458
12.3.6. Język definicji obiektów ODL	460
12.4. Projektowanie koncepcyjne obiektowej bazy danych	465
12.4.1. Różnice pomiędzy koncepcyjnym projektowaniem obiektowych i relacyjnych baz danych	465
12.4.2. Odzworowywanie schematu EER na schemat obiektowy	466
12.5. Obiektowy język zapytań OQL	468
12.5.1. Proste zapytania OQL, punkty wejścia bazy danych i zmienne iterujące	469
12.5.2. Wyniki zapytań i wyrażenia ścieżkowe	470
12.5.3. Inne cechy OQL	472
12.6. Przegląd wiązania z językiem C++ w standardzie ODMG	476
12.7. Podsumowanie	478
Pytania powtórkowe	479
Ćwiczenia	480
Wybrane publikacje	481
<b>13. XML — rozszerzalny język znaczników</b>	<b>483</b>
13.1. Dane strukturalne, półstrukturalne i niestrukturalne	484
13.2. Hierarchiczny (drzewiasty) model danych w dokumentach XML	488
13.3. Dokumenty XML, DTD i schematy	491
13.3.1. Dobrze uformowane i prawidłowe dokumenty XML oraz XML DTD	491
13.3.2. Schematy XML	494
13.4. Zapisywanie dokumentów XML w bazach i ich pobieranie	499
13.5. Języki związane ze standardem XML	500
13.5.1. XPath, czyli określanie ścieżek w dokumentach XML	500
13.5.2. XQuery: definiowanie zapytań w XML	502



13.5.3. Inne języki i protokoły związane ze standardem XML	504
13.6. Pobieranie dokumentów XML z relacyjnych baz danych	505
13.6.1. Tworzenie hierarchicznych perspektyw w formacie XML dla danych płaskich lub zapisanych w grafie	505
13.6.2. Przerywanie cykli w celu zamiany grafów w drzewa	509
13.6.3. Dodatkowe kroki związane z tworzeniem dokumentu XML na podstawie bazy danych	510
13.7. XML/SQL: funkcje języka SQL generujące dane w formacie XML	511
13.8. Podsumowanie	512
Pytania powtórkowe	513
Ćwiczenia	513
Wybrane publikacje	514

## **VI. Teoria projektowania baz danych i normalizacja** **515**

<b>14. Podstawy zależności funkcyjnych i normalizacji w relacyjnych bazach danych</b>	<b>517</b>
14.1. Nieformalne wskazówki dotyczące projektowania schematów relacji	519
14.1.1. Wymuszanie jednoznacznej semantyki atrybutów relacji	519
14.1.2. Nadmiarowe informacje w krotkach oraz anomalie aktualizacji	521
14.1.3. Wartości null w krotkach	525
14.1.4. Generowanie fałszywych krotek	526
14.1.5. Podsumowanie i omówienie wskazówek projektowych	528
14.2. Zależności funkcyjne	528
14.2.1. Definicja zależności funkcyjnej	529
14.3. Postaci normalne oparte na kluczach głównych	532
14.3.1. Normalizacja relacji	532
14.3.2. Praktyczne zastosowania postaci normalnych	533
14.3.3. Definicje kluczy i atrybutów należących do kluczy	534
14.3.4. Pierwsza postać normalna	535
14.3.5. Druga postać normalna	539
14.3.6. Trzecia postać normalna	540
14.4. Definicje ogólne drugiej i trzeciej postaci normalnej	542
14.4.1. Definicja ogólna drugiej postaci normalnej	542
14.4.2. Definicja ogólna trzeciej postaci normalnej	544
14.4.3. Interpretacja definicji ogólnej trzeciej postaci normalnej	544
14.5. Postać normalna Boyce'a-Codda	545
14.5.1. Dekompozycja relacji niebędących w BCNF	547
14.6. Zależności wielowartościowe i czwarta postać normalna	549
14.6.1. Formalna definicja zależności wielowartościowej	549
14.7. Zależności złączeniowe i piąta postać normalna	552
14.8. Podsumowanie	553
Pytania powtórkowe	555
Ćwiczenia	556
Ćwiczenia laboratoryjne	560
Wybrane publikacje	560
<b>15. Algorytmy projektowania relacyjnych baz danych i dodatkowe zależności</b>	<b>563</b>
15.1. Inne zagadnienia z obszaru zależności funkcyjnych: reguły wnioskowania, równoważności i pokrycie minimalne	564
15.1.1. Reguły wnioskowania dla zależności funkcyjnych	565
15.1.2. Równoważność zbiorów zależności funkcyjnych	569
15.1.3. Zbiory minimalne zależności funkcyjnych	570

15.2. Właściwości dekompozycji relacyjnych	573
15.2.1. Dekompozycja relacji i niewystarczalność postaci normalnych	573
15.2.2. Właściwość zachowania zależności dekompozycji	574
15.2.3. Właściwość złączenia bezstratnego (nieaddytywnego) dekompozycji	575
15.2.4. Testowanie dekompozycji binarnych pod względem występowania właściwości złączenia nieaddytywnego	577
15.2.5. Kolejne dekompozycje o złączeniach nieaddytywnych	578
15.3. Algorytmy projektowania schematów relacyjnych baz danych	579
15.3.1. Dekompozycja na schematy w trzeciej postaci normalnej z zachowaniem zależności i właściwością złączenia nieaddytywnego (bezstratnego)	579
15.3.2. Dekompozycja ze złączeniem nieaddytywnym na schematy w postaci normalnej Boyce'a-Codda	582
15.4. Problemy związane z wartościami pustymi i krotkami zawieszonymi oraz inne projekty relacyjne	584
15.4.1. Problemy związane z wartościami pustymi i krotkami zawieszonymi	584
15.4.2. Omówienie algorytmów normalizacyjnych i innych projektów relacyjnych	586
15.5. Dalsze omówienie zależności wielowartościowych i 4NF	588
15.5.1. Reguły wnioskowania dla zależności funkcyjnych i wielowartościowych	588
15.5.2. Jeszcze o czwartej postaci normalnej	589
15.5.3. Dekompozycja ze złączeniem nieaddytywnym na relacje w czwartej postaci normalnej	591
15.6. Inne zależności i postaci normalne	592
15.6.1. Zależności złączeniowe i piąta postać normalna	592
15.6.2. Zależności zawierania	592
15.6.3. Zależności funkcyjne oparte na funkcjach i procedurach arytmetycznych	593
15.6.4. Postać normalna klucza dziedziny	594
15.7. Podsumowanie	595
Pytania powtórkowe	595
Ćwiczenia	596
Ćwiczenia laboratoryjne	598
Wybrane publikacje	598

## **VII. Struktury plikowe, funkcje mieszające, indeksowanie i projekty fizyczne baz danych** **601**

<b>16. Składowanie danych na dysku, podstawowe struktury plikowe, funkcje mieszające i nowoczesne struktury składowania</b>	<b>603</b>
16.1. Wprowadzenie	604
16.1.1. Hierarchie pamięciowe i urządzenia składowania danych	605
16.1.2. Przechowywanie baz danych	607
16.2. Drugorzędne urządzenia pamięciowe	609
16.2.1. Opis sprzętowy napędów dyskowych	609
16.2.2. Zwiększanie wydajności dostępu do danych na dysku	615
16.2.3. Pamięć masowa SSD	616
16.2.4. Taśmowe urządzenia pamięciowe	617
16.3. Buforowanie bloków	619
16.3.1. Zarządzanie buforem	620
16.3.2. Strategie zastępowania danych w buforze	622
16.4. Rozmieszczanie rekordów plików na dysku	623
16.4.1. Rekordy i typy rekordów	623
16.4.2. Pliki oraz rekordy o stałej i zmiennej długości	624

16.4.3. Rozmieszczenie rekordów w blokach i rekordy segmentowane oraz niesegmentowane	626
16.4.4. Alokowanie bloków pliku na dysku	627
16.4.5. Nagłówki plików	627
16.5. Operacje wykonywane na plikach	627
16.6. Pliki nieuporządkowanych rekordów (pliki stertowe)	631
16.7. Pliki uporządkowanych rekordów (pliki posortowane)	632
16.8. Techniki mieszania	636
16.8.1. Mieszanie wewnętrzne	636
16.8.2. Mieszanie zewnętrzne dla plików na dysku	639
16.8.3. Techniki mieszania umożliwiające dynamiczne rozszerzanie plików	642
16.9. Inne podstawowe metody organizacji plików	647
16.9.1. Pliki rekordów mieszanych	647
16.9.2. B-drzewa i inne struktury danych służące jako podstawowe metody organizacji	648
16.10. Zapewnianie równoległego dostępu do dysku przy użyciu architektury RAID	648
16.10.1. Zwiększanie niezawodności przy użyciu architektury RAID	650
16.10.2. Poprawianie wydajności przy użyciu architektury RAID	651
16.10.3. Metody organizacji i poziomy architektury RAID	651
16.11. Nowoczesne architektury składowania danych	653
16.11.1. Sieci obszarów składowania danych	653
16.11.2. Technologia NAS	654
16.11.3. iSCSI i inne sieciowe protokoły składowania danych	655
16.11.4. Technologia Automated Storage Tiering	656
16.11.5. Obiektowa pamięć masowa	656
16.12. Podsumowanie	657
Pytania powtórkowe	658
Ćwiczenia	660
Wybrane publikacje	663
<b>17. Struktury indeksowe dla plików i fizyczne projekty baz danych</b>	<b>665</b>
17.1. Rodzaje jednopoziomowych indeksów uporządkowanych	666
17.1.1. Indeksy główne	667
17.1.2. Indeksy klastrowania	670
17.1.3. Indeksy drugorzędne	673
17.1.4. Podsumowanie	677
17.2. Indeksy wielopoziomowe	677
17.3. Dynamiczne indeksy wielopoziomowe z użyciem B-drzew i B <sup>+</sup> -drzew	681
17.3.1. Drzewa wyszukiwania i B-drzewa	682
17.3.2. B <sup>+</sup> -drzewa	687
17.4. Indeksy na wielu kluczach	696
17.4.1. Indeks uporządkowany na wielu atrybutach	697
17.4.2. Mieszanie partycjonowane	697
17.4.3. Pliki matrycowe	697
17.5. Inne rodzaje indeksów	699
17.5.1. Indeksy oparte na mieszanii	699
17.5.2. Indeksy bitmapowe	699
17.5.3. Indeksowanie oparte na funkcji	702
17.6. Ogólne zagadnienia związane z indeksami	703
17.6.1. Indeksy logiczne a fizyczne	703
17.6.2. Tworzenie indeksu	704
17.6.3. Dostrajanie indeksów	705
17.6.4. Dodatkowe kwestie związane ze składowaniem relacji i indeksów	706

17.7. Fizyczne projektowanie baz danych w przypadku baz relacyjnych	708
17.7.1. Czynniki wpływające na fizyczny projekt bazy danych	708
17.7.2. Decyzje dotyczące fizycznego projektu bazy danych	710
17.8. Podsumowanie	711
Pytania powtórkowe	713
Ćwiczenia	714
Wybrane publikacje	716

## **VIII. Przetwarzanie i optymalizacja zapytań** **719**

<b>18. Strategie przetwarzania zapytań</b>	<b>721</b>
18.1. Translacja zapytań języka SQL do postaci wyrażeń algebry relacji i innych operacji	724
18.1.1. Dodatkowe operatory złączeń częściowych i antyzłączeń	725
18.2. Algorytmy sortowania zewnętrznego	727
18.3. Algorytmy operacji selekcji	729
18.3.1. Możliwości implementacji operacji SELECT	729
18.3.2. Metody wyszukiwania dla selekcji na podstawie warunku koniunktywnego	731
18.3.3. Metody wyszukiwania dla selekcji na podstawie alternatywy logicznej	733
18.3.4. Szacowanie selektywności warunku	733
18.4. Implementacja operacji JOIN	734
18.4.1. Metody implementacji złączeń	735
18.4.2. Wpływ dostępnej przestrzeni bufora i pliku używanego w pętli zewnętrznej na wydajność operacji złączenia w pętli zagnieżdżonej	738
18.4.3. Wpływ współczynnika selekcji złączenia na wydajność tej operacji	739
18.4.4. Ogólna postać partycjonowanego złączenia mieszającego	741
18.4.5. Hybrydowe złączanie mieszające	742
18.5. Algorytmy operacji projekcji i teoriomnogościowych	743
18.5.1. Stosowanie antyzłączeń w operacji SET DIFFERENCE (EXCEPT lub MINUS w języku SQL)	744
18.6. Implementacja operacji agregujących oraz złączeń różnego rodzaju	745
18.6.1. Implementacja operacji agregujących	745
18.6.2. Implementacja różnego rodzaju złączeń	746
18.7. Łączenie operacji poprzez mechanizm potokowy	748
18.7.1. Iteratory używane do implementowania operacji fizycznych	749
18.8. Algorytmy równoległego przetwarzania zapytań	750
18.8.1. Równoległość na poziomie operatorów	751
18.8.2. Równoległość w jednym zapytaniu	754
18.8.3. Równoległość w wielu zapytaniach	754
18.9. Podsumowanie	755
Pytania powtórkowe	755
Ćwiczenia	756
Wybrane publikacje	756
<b>19. Optymalizacja zapytań</b>	<b>757</b>
19.1. Drzewa zapytań i heurystyki optymalizacji zapytań	758
19.1.1. Notacja drzew zapytań i grafów zapytań	758
19.1.2. Heurystyczna optymalizacja drzew zapytań	760
19.2. Wybór planów wykonania zapytań	767
19.2.1. Różne sposoby wykonywania zapytań	767
19.2.2. Optymalizacja podzapytań zagnieżdżonych	768
19.2.3. Scalanie podzapytań (perspektyw)	770
19.2.4. Perspektywy zmateriałizowane	772



19.3. Wykorzystanie selektywności w optymalizacji kosztowej	775
19.3.1. Składowe kosztu wykonywania zapytań	776
19.3.2. Informacje z katalogu używane w funkcjach kosztu	777
19.3.3. Histogramy	778
19.4. Funkcje kosztu dla operacji SELECT	778
19.4.1. Przykład optymalizacji selekcji na podstawie wzorów szacowania kosztów	781
19.5. Przykłady funkcji kosztu dla operacji JOIN	783
19.5.1. Selektywność i liczność złączeń częściowych i antyzłączeń	785
19.5.2. Przykład optymalizacji złączenia na podstawie funkcji kosztu	786
19.5.3. Zapytania dotyczące wielu relacji i porządkowanie złączeń	787
19.5.4. Optymalizacja fizyczna	789
19.5.5. Określanie kolejności złączeń za pomocą programowania dynamicznego	790
19.6. Przykład ilustrujący kosztową optymalizację zapytań	792
19.7. Dodatkowe zagadnienia związane z optymalizacją zapytań	794
19.7.1. Wyświetlanie planu wykonania zapytania uzyskanego przez system	794
19.7.2. Szacowanie wielkości wyników dla innych operacji	795
19.7.3. Zapis planu w pamięci podręcznej	796
19.7.4. Optymalizacja z wykorzystaniem pierwszych k wyników	796
19.8. Przykład optymalizacji zapytań w hurtowniach danych	796
19.9. Optymalizacja zapytań w bazach Oracle	799
19.9.1. Optymalizator fizyczny	799
19.9.2. Globalny optymalizator zapytań	799
19.9.3. Optymalizacja adaptacyjna	800
19.9.4. Przetwarzanie tablicowe	801
19.9.5. Wskazówki	801
19.9.6. Zarysy	802
19.9.7. Zarządzanie planami wykonywania instrukcji języka SQL	802
19.10. Semantyczna optymalizacja zapytań	803
19.11. Podsumowanie	804
Pytania powtórkowe	804
Ćwiczenia	805
Wybrane publikacje	806

## **IX. Przetwarzanie transakcji, sterowanie współbieżne i odtwarzanie baz danych**

**809**

<b>20. Wprowadzenie do problematyki i teorii przetwarzania transakcji</b>	<b>811</b>
20.1. Wprowadzenie do problematyki przetwarzania transakcji	812
20.1.1. Systemy jedno- i wieloużytkownikowe	812
20.1.2. Transakcje, elementy baz danych, operacje odczytu i zapisu oraz buforu SZBD	813
20.1.3. Uzasadnienie potrzeby stosowania sterowania współbieżnego	815
20.1.4. Uzasadnienie potrzeby odtwarzania	818
20.2. Pojęcia dotyczące transakcji i systemu	819
20.2.1. Stany transakcji i dodatkowe operacje	819
20.2.2. Dziennik systemowy	821
20.2.3. Punkt zatwierdzenia transakcji	822
20.2.4. Strategie zastępowania bufora specyficzne dla SZBD	823
20.3. Pożądane właściwości transakcji	824
20.4. Charakteryzowanie harmonogramów na podstawie możliwości odtwarzania	825
20.4.1. Harmonogramy (historie) transakcji	825
20.4.2. Charakterystyka harmonogramów według możliwości odtwarzania	827

20.5. Charakterystyka harmonogramów według ich szeregowalności	829
20.5.1. Harmonogramy szeregowo, nieszeregowo oraz konfliktowo-szeregowalne	830
20.5.2. Sprawdzanie występowania szeregowalności konfliktowej harmonogramu	834
20.5.3. Wykorzystywanie szeregowalności do sterowania współbieżnego	837
20.5.4. Równoważność perspektywiczna i szeregowalność perspektywiczna	839
20.5.5. Inne rodzaje równoważności harmonogramów	840
20.6. Obsługa transakcji w języku SQL	841
20.7. Podsumowanie	843
Pytania powtórkowe	844
Ćwiczenia	845
Wybrane publikacje	846
<b>21. Techniki sterowania współbieżnego</b>	<b>847</b>
21.1. Techniki blokowania dwufazowego dla celów sterowania współbieżnego	848
21.1.1. Rodzaje blokad i systemowe tabele blokad	848
21.1.2. Gwarantowanie szeregowalności blokowania dwufazowego	853
21.1.3. Problem zakleszczenia i zagłodzenia	856
21.2. Sterowanie współbieżne w oparciu o uporządkowanie według znaczników czasu	860
21.2.1. Znaczniki czasu	860
21.2.2. Algorytm uporządkowania według znaczników czasu	860
21.3. Techniki wielowersyjnego sterowania współbieżnego	862
21.3.1. Technika wielowersyjna oparta na porządkowaniu według znaczników czasu	863
21.3.2. Wielowersyjne blokowanie dwufazowe z użyciem blokad certyfikujących	864
21.4. Sterowanie współbieżne z użyciem technik walidacyjnych (optymistycznych) i izolacji snapshotów	866
21.4.1. Walidacyjne (optymistyczne) sterowanie współbieżne	866
21.4.2. Sterowanie współbieżne oparte na izolacji snapshotów	868
21.5. Ziarnistość elementów danych i blokowanie z wieloma poziomami ziarnistości	869
21.5.1. Kwestie dotyczące poziomu ziarnistości w przypadku blokowania	869
21.5.2. Blokowanie z wieloma poziomami ziarnistości	870
21.6. Użycie blokad dla celów sterowania współbieżnego w przypadku indeksów	872
21.7. Inne kwestie związane ze sterowaniem współbieżnym	874
21.7.1. Wstawianie, usuwanie i rekordy fantomowe	875
21.7.2. Transakcje interaktywne	876
21.7.3. Zatraski	876
21.8. Podsumowanie	876
Pytania powtórkowe	877
Ćwiczenia	878
Wybrane publikacje	879
<b>22. Techniki odtwarzania baz danych</b>	<b>881</b>
22.1. Pojęcia związane z odtwarzaniem	882
22.1.1. Zarys problematyki odtwarzania i podział algorytmów odtwarzania na odrębne kategorie	882
22.1.2. Zapisywanie w pamięci podręcznej (buforowanie) bloków dyskowych	883
22.1.3. Rejestrowanie zapisów z wyprzedzeniem, technika zabierania oraz wymuszania	885
22.1.4. Punkty kontrolne w dzienniku systemowym oraz tworzenie przybliżonych punktów kontrolnych	887
22.1.5. Wycofywanie transakcji i wycofywanie kaskadowe	888
22.1.6. Działania transakcji niewpływające na bazy danych	890
22.2. Techniki odtwarzania NO-UNDO/REDO oparte na aktualizacjach odroczonech	890
22.3. Techniki odtwarzania oparte na aktualizacjach natychmiastowych	893

22.4. Stronicowanie z przesłaniem	895
22.5. Algorytm odtwarzania ARIES	897
22.6. Odtwarzanie w systemach wielu baz danych	901
22.7. Tworzenie kopii bezpieczeństwa bazy danych i odtwarzanie po awariach katastroficznych	902
22.8. Podsumowanie	903
Pytania powtórkowe	904
Ćwiczenia	905
Wybrane publikacje	908

## **X. Rozproszone bazy danych, systemy NOSQL i big data** **911**

---

<b>23. Zagadnienia z obszaru rozproszonych baz danych</b>	<b>913</b>
23.1. Zagadnienia z obszaru rozproszonych baz danych	914
23.1.1. Co sprawia, że baza danych jest rozproszona?	914
23.1.2. Przezroczystość	915
23.1.3. Stabilność i dostępność	916
23.1.4. Skalowalność i odporność na podział	917
23.1.5. Autonomia	917
23.1.6. Zalety rozproszonych baz danych	917
23.2. Techniki fragmentacji, replikacji i alokacji danych	
w projekcie rozproszonej bazy danych	918
23.2.1. Fragmentacja danych i sharding	918
23.2.2. Replikacja i alokacja danych	921
23.2.3. Przykłady fragmentacji, alokacji i replikacji danych	922
23.3. Techniki sterowania współbieżnego i odtwarzania danych	
w rozproszonych bazach danych	925
23.3.1. Rozproszone sterowanie współbieżne oparte na wyróżnionej kopii danych	925
23.3.2. Rozproszone sterowanie współbieżne oparte na głosowaniu	927
23.3.3. Rozproszone odtwarzanie danych	927
23.4. Przegląd zarządzania transakcjami w rozproszonych bazach danych	928
23.4.1. Protokół zatwierdzania dwufazowego	929
23.4.2. Protokół zatwierdzania trójfazowego	929
23.4.3. Obsługa zarządzania transakcjami w systemie operacyjnym	930
23.5. Przetwarzanie zapytań i optymalizacja w rozproszonych bazach danych	930
23.5.1. Przetwarzanie zapytań rozproszonych	931
23.5.2. Koszty przesyłu danych w przetwarzaniu zapytań rozproszonych	931
23.5.3. Rozproszone przetwarzanie zapytań z użyciem złączeń częściowych	933
23.5.4. Dekompozycja zapytań i aktualizacji	934
23.6. Rodzaje rozproszonych systemów baz danych	937
23.6.1. Zarządzanie federacyjnymi systemami baz danych	938
23.7. Architektury rozproszonych baz danych	940
23.7.1. Architektura równoległa a rozproszona	940
23.7.2. Ogólna architektura czystych baz rozproszonych	942
23.7.3. Architektura federacyjnych baz danych	942
23.7.4. Przegląd trójwarstwowej architektury klient-serwer	944
23.8. Zarządzanie rozproszonym katalogiem	946
23.9. Podsumowanie	947
Pytania powtórkowe	948
Ćwiczenia	949
Wybrane publikacje	951

<b>24. Bazy danych NOSQL i systemy składowania big data</b>	<b>955</b>
24.1. Wprowadzenie do systemów NOSQL	955
24.1.1. Powstanie systemów NOSQL	955
24.1.2. Cechy systemów NOSQL	957
24.1.3. Kategorie systemów NOSQL	959
24.2. Twierdzenie CAP	960
24.3. Dokumentowe systemy NOSQL i baza MongoDB	961
24.3.1. Model danych z systemu MongoDB	962
24.3.2. Operacje CRUD w systemie MongoDB	965
24.3.3. Cechy systemu rozproszonego MongoDB	965
24.4. Magazyny NOSQL z parami klucz-wartość	967
24.4.1. Przegląd systemu DynamoDB	967
24.4.2. Rozproszony magazyn danych z parami klucz-wartość — Voldemort	968
24.4.3. Przykładowe inne magazyny z parami klucz-wartość	971
24.5. Kolumnowe systemy NOSQL	972
24.5.1. Model danych i wersjonowanie w systemie HBase	972
24.5.2. Operacje CRUD w systemie HBase	974
24.5.3. Zagadnienia związane ze składowaniem danych i systemem rozproszonym w HBase	974
24.6. Grafowe bazy NOSQL i system Neo4j	975
24.6.1. Model danych w systemie Neo4j	975
24.6.2. Język zapytań Cypher w systemie Neo4j	977
24.6.3. Cechy interfejsów i systemu rozproszonego w Neo4j	979
24.7. Podsumowanie	980
Pytania powtórkowe	981
Wybrane publikacje	982
<b>25. Technologie z obszaru big data oparte na modelu MapReduce i systemie Hadoop</b>	<b>983</b>
25.1. Czym jest big data?	986
25.2. Wprowadzenie do technologii MapReduce i Hadoop	988
25.2.1. Tło historyczne	988
25.2.2. Model MapReduce	989
25.2.3. Wersje Hadoopa	993
25.3. System HDFS	993
25.3.1. Wymagania wstępne związane z systemem HDFS	994
25.3.2. Architektura systemu HDFS	994
25.3.3. Operacje wejścia-wyjścia na plikach i zarządzanie replikami w systemie HDFS	996
25.3.4. Skalowalność systemu HDFS	997
25.3.5. Ekosystem Hadoopa	998
25.4. Model MapReduce: dodatkowe szczegóły	999
25.4.1. Środowisko uruchomieniowe MapReduce	999
25.4.2. Przykład: złączenia w modelu MapReduce	1002
25.4.3. Apache Hive	1006
25.4.4. Zalety technologii Hadoop i MapReduce	1008
25.5. Hadoop 2 (nazywany też YARN)	1009
25.5.1. Uzasadnienie powstania platformy YARN	1009
25.5.2. Architektura platformy YARN	1012
25.5.3. Inne platformy w YARN	1015
25.6. Ogólne omówienie	1017
25.6.1. Hadoop i MapReduce a równoległe relacyjne SZBD	1017
25.6.2. Big data w chmurach obliczeniowych	1019



25.6.3. Problemy z lokalnością danych i optymalizacja zasobów w aplikacjach z obszaru big data działających w chmurze	1021
25.6.4. YARN jako platforma usług z obszaru danych	1022
25.6.5. Wyzwania związane z technologiami z obszaru big data	1023
25.6.6. Przyszłość	1024
25.7. Podsumowanie	1026
Pytania powtórkowe	1027
Wybrane publikacje	1028

## **XI. Zaawansowane modele, systemy i zastosowania baz danych**

**1031**

<b>26. Rozszerzone modele danych: wprowadzenie do aktywnych, czasowych, przestrzennych, multimedialnych i dedukcyjnych baz danych</b>	<b>1033</b>
26.1. Wyzwalacze i inne pojęcia związane z aktywnymi bazami danych	1035
26.1.1. Uogólniony model aktywnych baz danych i wyzwalacze w Oracle	1035
26.1.2. Projektowanie i implementacja aktywnych baz danych	1039
26.1.3. Przykładowe aktywne reguły poziomu wyrażenia w systemie STARBURST	1042
26.1.4. Możliwe zastosowania aktywnych baz danych	1044
26.1.5. Wyzwalacze w SQL-99	1044
26.2. Koncepcja czasowych baz danych	1045
26.2.1. Reprezentacja czasu, kalendarze i wymiary czasu	1046
26.2.2. Wprowadzenie czasu do relacyjnych baz danych z obsługą wersji krotek	1048
26.2.3. Czas w obiektowych bazach danych z obsługą wersji atrybutów	1053
26.2.4. Konstruowanie zapytań czasowych i język TSQL2	1055
26.2.5. Szeregi czasowe	1057
26.3. Zagadnienia z obszaru przestrzennych baz danych	1058
26.3.1. Wprowadzenie do przestrzennych baz danych	1058
26.3.2. Typy i modele danych przestrzennych	1059
26.3.3. Operatory i zapytania przestrzenne	1060
26.3.4. Indeksowanie danych przestrzennych	1062
26.3.5. Eksploracja danych przestrzennych	1063
26.3.6. Zastosowania danych przestrzennych	1065
26.4. Zagadnienia z obszaru multimedialnych baz danych	1065
26.4.1. Automatyczna analiza obrazów	1067
26.4.2. Wykrywanie obiektów na obrazach	1068
26.4.3. Semantyczne opisywanie obrazów	1069
26.4.4. Analizy danych audio	1069
26.5. Wprowadzenie do dedukcyjnych baz danych	1070
26.5.1. Przegląd dedukcyjnych baz danych	1070
26.5.2. Notacja języków Prolog i Datalog	1071
26.5.3. Notacja języka Datalog	1073
26.5.4. Forma klauzulowa i klauzule Horna	1074
26.5.5. Interpretacja reguł	1075
26.5.6. Programy w języku Datalog i ich bezpieczeństwo	1077
26.5.7. Zastosowanie operacji relacyjnych	1081
26.5.8. Wykonywanie zapytań nierekurencyjnych	1081
26.6. Podsumowanie	1083
Pytania powtórkowe	1085
Ćwiczenia	1086
Wybrane publikacje	1089

<b>27. Wprowadzenie do wyszukiwania informacji i danych w internecie</b>	<b>1093</b>
27.1. Zagadnienia z obszaru wyszukiwania informacji (WI)	1093
27.1.1. Wprowadzenie do wyszukiwania informacji	1094
27.1.2. Porównanie baz danych i systemów WI	1097
27.1.3. Krótka historia WI	1098
27.1.4. Tryby interakcji w systemach WI	1099
27.1.5. Ogólny proces WI	1100
27.2. Modele wyszukiwania	1101
27.2.1. Model logiczny	1101
27.2.2. Model oparty na przestrzeni wektorowej	1102
27.2.3. Model probabilistyczny	1104
27.2.4. Model semantyczny	1106
27.3. Typy zapytań w systemach WI	1107
27.3.1. Zapytania oparte na słowach kluczowych	1107
27.3.2. Zapytania logiczne	1107
27.3.3. Zapytania do wyszukiwania fraz	1108
27.3.4. Zapytania z określaniem odległości słów	1108
27.3.5. Zapytania z symbolami wieloznacznymi	1108
27.3.6. Zapytania w języku naturalnym	1109
27.4. Wstępne przetwarzanie tekstu	1109
27.4.1. Usuwanie słów pomijanych	1109
27.4.2. Stemming	1109
27.4.3. Korzystanie z tezaury	1110
27.4.4. Inne etapy przetwarzania: cyfry, myślniki, znaki przestankowe, wielkość znaków	1111
27.4.5. Wydobywanie informacji	1112
27.5. Indeksy odwrócone	1112
27.5.1. Wprowadzenie do systemu Lucene	1114
27.6. Miary oceny adekwatności wyników wyszukiwania	1115
27.6.1. Czułość i precyzja	1116
27.6.2. Średnia precyzja	1118
27.6.3. Krzywa czułość/precyzja	1118
27.6.4. Miara F	1118
27.7. Wyszukiwanie i analizy w sieci WWW	1119
27.7.1. Analizy danych internetowych i ich związki z WI	1119
27.7.2. Analizy struktury sieci WWW	1121
27.7.3. Analizowanie struktury odsyłaczy na stronach internetowych	1122
27.7.4. Analizy treści w sieci WWW	1123
27.7.5. Podejścia analizowania treści w sieci WWW	1125
27.7.6. Analizy użytkowania witryn	1126
27.7.7. Praktyczne zastosowania analiz użytkowania witryn	1128
27.8. Trendy w wyszukiwaniu informacji	1129
27.8.1. Wyszukiwanie fasetowe	1129
27.8.2. Wyszukiwanie społecznościowe	1130
27.8.3. Wyszukiwanie informacji w dialogach	1131
27.8.4. Probabilistyczne modelowanie tematu	1131
27.8.5. Systemy odpowiadania na pytania	1132
27.9. Podsumowanie	1135
Pytania powtórkowe	1136
Wybrane publikacje	1138
<b>28. Elementy eksploracji danych</b>	<b>1141</b>
28.1. Przegląd technologii eksploracji danych	1142
28.1.1. Eksploracja danych kontra hurtownie danych	1142
28.1.2. Eksploracja danych jako część procesu odkrywania wiedzy	1142

28.1.3. Cele eksploracji danych i odkrywania wiedzy	1143
28.1.4. Rodzaje wiedzy odkrywanej w procesie eksploracji danych	1145
28.2. Reguły asocjacyjne	1146
28.2.1. Model koszyka klienta supermarketu, poziom wsparcia i poziom ufności	1146
28.2.2. Algorytm Apriori	1148
28.2.3. Algorytm próbkujący	1150
28.2.4. Drzewa częstych wzorców i algorytm ich tworzenia	1151
28.2.5. Algorytm partycjonujący	1155
28.2.6. Pozostałe typy reguł asocjacyjnych	1156
28.2.7. Dodatkowe problemy związane z regułami asocjacyjnymi	1159
28.3. Klasyfikacja	1160
28.4. Grupowanie	1163
28.5. Strategie rozwiązywania pozostałych problemów związanych z eksploracją danych	1166
28.5.1. Odkrywanie wzorców sekwencyjnych	1166
28.5.2. Odkrywanie wzorców w szeregach czasowych	1167
28.5.3. Regresja	1167
28.5.4. Sieci neuronowe	1168
28.5.5. Algorytmy genetyczne	1169
28.6. Zastosowania technik eksploracji danych	1170
28.7. Komercyjne narzędzia eksploracji danych	1170
28.7.1. Interfejs użytkownika	1171
28.7.2. Interfejs programowy aplikacji	1171
28.7.3. Kierunki przyszłego rozwoju	1171
28.8. Podsumowanie	1173
Pytania powtórkowe	1173
Ćwiczenia	1174
Wybrane publikacje	1176
<b>29. Przegląd hurtowni danych i rozwiązań OLAP</b>	<b>1177</b>
29.1. Wprowadzenie, definicje i terminologia	1177
29.2. Właściwości hurtowni danych	1179
29.3. Modelowanie danych dla hurtowni danych	1181
29.4. Budowanie hurtowni danych	1187
29.5. Typowe funkcje hurtowni danych	1191
29.6. Hurtownie danych kontra perspektywy	1192
29.7. Trudności z implementowaniem hurtowni danych	1193
29.8. Podsumowanie	1194
Pytania powtórkowe	1195
Wybrane publikacje	1195

## **XII. Dodatkowe zagadnienia z obszaru baz danych: bezpieczeństwo**

**1197**


---

<b>30. Bezpieczeństwo w bazach danych</b>	<b>1199</b>
30.1. Wprowadzenie do bezpieczeństwa baz danych	1200
30.1.1. Rodzaje zabezpieczeń	1200
30.1.2. Środki kontroli	1201
30.1.3. Bezpieczeństwo a administrator bazy danych	1203
30.1.4. Ochrona dostępu, konta użytkowników i audyty bazy danych	1204
30.1.5. Dane wrażliwe i typy ujawnień	1205
30.1.6. Związki między bezpieczeństwem a prywatnością informacji	1206

30.2. Dyspozycyjna kontrola dostępu polegająca na nadawaniu i odbieraniu uprawnień	1207
30.2.1. Typy uprawnień dyspozycyjnych	1207
30.2.2. Określanie uprawnień przy użyciu perspektyw	1208
30.2.3. Cofanie uprawnień	1209
30.2.4. Propagacja uprawnień poprzez opcję GRANT	1209
30.2.5. Przykład	1209
30.2.6. Określanie ograniczeń propagacji uprawnień	1211
30.3. Realizacja zabezpieczeń wielopoziomowych	
za pomocą obowiązkowej kontroli dostępu i zabezpieczeń opartych na rolach	1212
30.3.1. Porównanie dyspozycyjnego i obowiązkowego modelu bezpieczeństwa	1215
30.3.2. Kontrola dostępu oparta na rolach	1215
30.3.3. Zabezpieczenia oparte na etykietach i kontrola dostępu na poziomie wierszy	1217
30.3.4. Kontrola dostępu dla danych w formacie XML	1218
30.3.5. Polityki kontroli dostępu dla aplikacji sieciowych i mobilnych	1219
30.4. Wstrzykiwanie kodu w języku SQL	1220
30.4.1. Metody wstrzykiwania kodu w języku SQL	1221
30.4.2. Zagrożenia związane ze wstrzykiwaniem kodu w języku SQL	1222
30.4.3. Techniki ochrony przed wstrzykiwaniem kodu w języku SQL	1223
30.5. Wprowadzenie do bezpieczeństwa statystycznych baz danych	1224
30.6. Wprowadzenie do kontroli przepływu	1225
30.6.1. Ukryte kanały	1226
30.7. Szyfrowanie i infrastruktura klucza publicznego	1227
30.7.1. Szyfry DES i AES	1228
30.7.2. Algorytmy z kluczem symetrycznym	1228
30.7.3. Szyfrowanie kluczem publicznym (asymetrycznym)	1228
30.7.4. Podpis cyfrowy	1230
30.7.5. Certyfikaty cyfrowe	1230
30.8. Problemy z prywatnością i jej zachowywanie	1231
30.9. Wyzwania związane z utrzymaniem bezpieczeństwa baz danych	1232
30.9.1. Jakość danych	1232
30.9.2. Prawa własności intelektualnej	1232
30.9.3. Odporność baz danych	1233
30.10. Zabezpieczenia oparte na etykietach w bazach Oracle	1233
30.10.1. Technologia wirtualnych prywatnych baz danych	1234
30.10.2. Architektura zabezpieczeń opartych na etykietach	1234
30.10.3. Współdziałanie etykiet danych i etykiet użytkowników	1235
30.11. Podsumowanie	1236
Pytania powtórkowe	1237
Ćwiczenia	1239
Wybrane publikacje	1239

## **Dodatki** **1241**

### **A. Alternatywne notacje modeli związków encji** **1243**

### **B. Parametry dysków** **1247**

### **C. Omówienie języka QBE** **1251**

## **Bibliografia** **1259**

## **Skorowidz** **1319**

---

# Przedmowa

Ta książka wprowadza podstawowe pojęcia niezbędne do projektowania, wykorzystywania i implementowania systemów baz danych i opartych na nich aplikacji. W niniejszej prezentacji położono szczególny nacisk na podstawy modelowania i projektowania baz danych, języki i modele udostępniane przez systemy zarządzania bazami danych, a także techniki implementacji samych systemów. Książka została stworzona nie tylko z myślą o roli podręcznika wykorzystywanego podczas jedno- lub dwusemestralnego kursu poświęconemu systemom baz danych na poziomie licealnym, studiów licencjackich lub studiów magisterskich, ale także w celu stworzenia praktycznego źródła informacji. Naszym zamiarem jest zapewnienie szczegółowej i aktualnej prezentacji najważniejszych aspektów systemów i aplikacji bazodanowych oraz powiązanych technologii. Zakładamy, że czytelnicy tej książki mają elementarną wiedzę z zakresu programowania i struktur danych, oraz że dysponują przynajmniej minimalnym doświadczeniem w kwestii architektury komputerów.

## Nowości w tym wydaniu

W niniejszym, siódmym wydaniu wprowadzono następujące ważne elementy:

- Zmieniono kolejność rozdziałów (na podstawie ankiety przeprowadzonej wśród wykładowców posługujących się tym podręcznikiem). Jednak książka nadal jest tak uporządkowana, aby poszczególni wykładowcy mogli albo realizować materiał zgodnie z nową kolejnością, albo *zdecydować się na inny układ* (np. z wydania szóstego).
- Dodano dwa nowe rozdziały dotyczące ostatnich osiągnięć z obszaru systemów bazodanowych i big data. W jednym z tych rozdziałów, 24., znajdziesz wprowadzenie do nowszego rodzaju systemów bazodanowych, tzw. **baz NOSQL**. W drugim, 25., opisano technologie przetwarzania **big data**, w tym model **MapReduce** i system **Hadoop**.

- Rozdział poświęcony przetwarzaniu i optymalizacji zapytań został wzbogacony i podzielony na dwa rozdziały. Rozdział 18. dotyczy głównie strategii i algorytmów przetwarzania zapytań, a rozdział 19. jest poświęcony technikom optymalizowania zapytań.
- Obok przykładowej bazy z poprzednich wydań, FIRMA, w początkowych rozdziałach (od 3. do 8.) używana jest druga przykładowa baza, UNIWERSYTET.
- Liczne rozdziały zostały w mniejszym lub większym stopniu zaktualizowane o nowe techniki i metody. Zamiast opisywać te zmiany w tym miejscu, przedstawiamy je w dalszej części przedmowy, w omówieniu układu siódmego wydania.

Oto najważniejsze cechy tej książki:

- Elastyczny układ, który można dostosować do indywidualnych potrzeb. *Rozdziały można przerabiać w różnej kolejności w zależności od preferencji wykładowcy.*
- Diagram powiązań (przedstawiony w dalszej części przedmowy), ilustrujący, które rozdziały zależą od innych wcześniejszych. Jest to pomoc dla wykładowców, którzy chcą dobrać *kolejność prezentacji rozdziałów*.
- Zestaw dodatków, w tym rozbudowane materiały dla wykładowców i studentów — prezentacje w programie PowerPoint, rysunki z tekstu i wskazówki dla nauczycieli wraz z rozwiązaniami.

## Układ i zawartość siódmego wydania

W siódmym wydaniu wprowadzono pewne zmiany w układzie, a także poprawki w poszczególnych rozdziałach. Obecnie książka jest podzielona na 12 części:

- W części I. (rozdziały 1. i 2.) opisano podstawowe, wprowadzające zagadnienia potrzebne do zrozumienia modeli, systemów i języków z obszaru baz danych. W rozdziałach 1. i 2. znajdziesz wprowadzenie do baz danych oraz omówienie typowych użytkowników, a także zagadnień, terminologii i architektury z dziedziny systemów zarządzania bazami danych (SZBD). Opisano tu też postępy w technologiach bazodanowych i krótką historię modeli danych. Te rozdziały zostały zaktualizowane o nowsze technologie, takie jak systemy NOSQL.
- Część II. (rozdziały 3. i 4.) obejmuje prezentację modelu związków encji (ang. *Entity-Relationship* — ER) i projektowania baz danych. Jednak *należy zauważyć*, że wykładowcy mogą omówić rozdziały dotyczące modelu relacyjnego (od 5. do 8.) *przed rozdziałami 3. i 4.*, jeśli preferują taką kolejność. W rozdziale 3. zagadnienia dotyczące modelu ER i diagramów ER są wykorzystane do zilustrowania koncepcyjnego projektowania baz danych. W rozdziale 4. pokazano, jak wzbogacić podstawowy model ER o dodatkowe aspekty, takie jak podklasy, specjalizacja, generalizacja, typy unii (kategorie) i dziedziczenie. W ten sposób dojdziemy do wzbogaconego modelu ER (ang. *enhanced ER* — EER) i diagramów EER. W rozdziałach 3. i 4. przedstawiono diagramy UML, będące alternatywą dla modeli i diagramów ER/EER.



- Część III. (rozdziały od 5. do 9.) obejmuje szczegółową prezentację baz relacyjnych i języka SQL. W rozdziałach dotyczących SQL-a znajdziesz nowe, dodatkowe materiały na temat kilku konstruktów, których nie opisano w poprzednim wydaniu. Rozdział 5. to opis podstawowego modelu relacyjnego, ograniczeń integralności i operacji aktualizacji. W rozdziale 6. przedstawiono inne podstawowe elementy standardu SQL w kontekście baz relacyjnych, w tym definicje danych, operacje modyfikowania danych i proste zapytania SQL-owe. Rozdział 7. zawiera omówienie bardziej skomplikowanych zapytań SQL-owych, a także elementów tego języka takich jak wyzwalacze, asercje, perspektywy, modyfikowanie schematu. Rozdział 8. obejmuje opis formalnych operacji z algebry relacyjnej i rachunku relacyjnego. Pozwala to wykładowcom rozpocząć projekty z użyciem SQL-a na wczesnych etapach kursu (możliwe jest omówienie rozdziału 8. przed rozdziałami 6. i 7.). Ostatni fragment części 2., rozdział 9., zawiera omówienie odwzorowań z modeli ER i EER na model relacyjny. Służące do tego algorytmy można wykorzystać do zaprojektowania schematu bazy relacyjnej na podstawie koncepcyjnego schematu ER lub EER.
- Część IV. (rozdziały 10. i 11.) dotyczy technik programowania baz danych. Te rozdziały można zadać studentom jako lekturę wraz z materiałami dotyczącymi konkretnego języka używanego na kursie w projektach programistycznych (dokumentacja takich języków jest łatwo dostępna w internecie). W rozdziale 10. omówiono tradycyjne zagadnienia programowania w SQL-u, np.: osadzony SQL, dynamiczny SQL, ODBC, SQLJ, JDBC i SQL/CLI. Rozdział 11. to wprowadzenie do programowania baz internetowych (w przykładach używany jest język skryptowy PHP). Ten rozdział obejmuje nowy materiał poświęcony wykorzystaniu technologii Javy do programowania baz internetowych.
- Część V. (rozdziały 12. i 13.) obejmuje zaktualizowany materiał dotyczący baz obiektowo-relacyjnych i obiektowych (rozdział 12.) oraz XML-a (rozdział 13.). Oba te rozdziały pokazują obecnie, jak w nowsze wersje standardu SQL wbudowano aspekty obiektowe i związane z XML-em. W rozdziale 12. najpierw przedstawione są bazy obiektowe, a dalej wyjaśniono, jak włączono je w standard SQL, aby dodać możliwości obiektowe do systemów relacyjnych. Następnie opisano model obiektowy ODMG oraz używane w nim definicje obiektów i języki zapytań. Rozdział 13. to omówienie modelu i języków z obszaru XML-a (ang. *eXtensible Markup Language*). Wyjaśniono tu, jak XML jest powiązany z systemami bazodanowymi. Przedstawiono zagadnienia i języki związane z XML-em i porównano model XML-owy z modelami tradycyjnymi. Pokazano też, jak przekształcać dane między XML-em a reprezentacją relacyjną. Podano również polecenia SQL-owe służące do pobierania dokumentów XML-owych z tabel relacyjnych.
- Część VI. (rozdziały 14. i 15.) jest poświęcona normalizacji i teorii projektowania relacyjnego. Wszystkie formalne aspekty algorytmów normalizacji zostały przeniesione do rozdziału 15. W rozdziale 14. zdefiniowane są zależności funkcyjne i oparte na nich postaci normalne. W rozdziale 14. opisano też intuicyjną, realizowaną krok po kroku metodę normalizacji, a także zależnościowe wielowartościowe i zależności złączeń. Rozdział 15. jest poświęcony teorii normalizacji, a także formalnym aspektom, teoriom i algorytmom rozwijanym na potrzeby projektowania baz relacyjnych z wykorzystaniem normalizacji. Znajdziesz tu też algorytmy dekompozycji i syntezy relacyjnej.

- Część VII. (rozdziały 16. i 17.) zawiera informacje na temat uporządkowania plików na dysku (rozdział 16.) i indeksowania plików bazodanowych (rozdział 17.). W rozdziale 16. opisano główne metody porządkowania plików z rekordami na dysku, w tym pliki uporządkowane (posortowane), nieuporządkowane (stertowe) i tworzone za pomocą techniki mieszania. Omówione są tam techniki mieszania statycznego i dynamicznego. Rozdział 16. został zaktualizowany o materiały dotyczące strategii zarządzania buforami w SZBD, a także o omówienie nowych urządzeń pamięci masowej oraz standardów związanych z plikami i nowoczesnymi architekturami składowania danych. W rozdziale 17. opisano techniki indeksowania plików, w tym struktury danych takie jak B-drzewa i B<sup>+</sup>-drzewa oraz pliki matrycowe. Rozdział ten został wzbogacony o nowe przykłady i rozszerzone omówienie indeksowania z uwzględnieniem wyboru odpowiednich indeksów i tworzenia indeksów na etapie rozwijania projektu fizycznego.
- Część VIII. (rozdziały 18. i 19.) obejmuje materiały na temat algorytmów przetwarzania zapytań (rozdział 18.) i technik optymalizacji (rozdział 19.). Te dwa rozdziały zostały zaktualizowane i rozdzielone z jednego, w którym we wcześniejszych wydaniach omówione były oba te zagadnienia. Znajdziesz tu też nowsze techniki stosowane w komercyjnych SZBD. W rozdziale 18. zaprezentowane są algorytmy do wyszukiwania rekordów w plikach na dysku, do łączenia rekordów z dwóch plików (tabel) oraz do wykonywania innych operacji relacyjnych. Rozdział ten zawiera nowy materiał, w tym omówienie złączeń częściowych i antyzłączeń wraz z przykładami ich stosowania w przetwarzaniu zapytań, a także opis technik szacowania selektywności. W rozdziale 19. opisano techniki optymalizowania zapytań z użyciem reguł szacowania kosztów i heurystyk. Znajduje się tu nowy materiał poświęcony optymalizacji zagnieżdżonych podzapytań, wykorzystaniu histogramów, optymalizacji fizycznej, określaniu kolejności złączeń i optymalizowaniu typowych zapytań w hurtowniach danych.
- W części IX. (rozdziały 20., 21. i 22.) opisano zagadnienia związane z przetwarzaniem transakcji, kontrolą współbieżności i odtwarzaniem baz danych po awariach. Te rozdziały zostały zaktualizowane o nowsze techniki stosowane w niektórych komercyjnych i otwartych SZBD. Rozdział 20. zawiera wprowadzenie do technik potrzebnych w systemach przetwarzania transakcji. Zdefiniowane są tu możliwość odtwarzania i szeregowalność harmonogramów. Ponadto dodano nowe fragmenty na temat strategii zastępowania zawartości bufora w SZBD i izolacji snapshotów. W rozdziale 21. znajdziesz przegląd różnych typów protokołów sterowania współbieżnego z naciskiem na blokowanie dwufazowe. Opisano też porządkowanie według znaczników czasu, techniki optymistycznego sterowania współbieżnego i blokowanie z wieloma poziomami ziarnistości. Rozdział 21. zawiera również nowe omówienie metod sterowania współbieżnego opartych na izolacji snapshotów. W rozdziale 22. skoncentrowano się na protokołach odtwarzania baz danych. Znajdziesz tam przegląd zagadnień i technik związanych z tym obszarem.
- Część X. (rozdziały 23., 24. i 25.) obejmuje rozdział poświęcony rozproszonym bazom danych (23.) i dwa nowe rozdziały: jeden na temat systemów NOSQL służących do składowania big data (24.) i drugi dotyczący technologii z obszaru big data opartych na systemie Hadoop i modelu MapReduce (25.). Rozdział 23. zawiera wprowadzenie do zagadnień z dziedziny rozproszonych baz danych, w tym do dostępności, skalowalności, replikacji i fragmentacji danych, utrzymywania

spójności danych między replikami, a także wielu innych kwestii i technik. W rozdziale 24. systemy typu NOSQL są podzielone na cztery ogólne kategorie (dla każdej z nich przedstawiono przykładowy system). Opisano tu i porównano modele danych, operacje, a także strategie replikacji, udostępniania i skalowania typowe dla każdej kategorii. Rozdział 25. zawiera wprowadzenie do modelu MapReduce służącego do rozproszonego przetwarzania big data. Dalej przedstawiono systemy Hadoop i HDFS (ang. *Hadoop Distributed File System*), wysokopoziomowe interfejsy Pig i Hive, a także architekturę YARN.

- Część XI. (rozdziały od 26. do 29.) jest poświęcona zaawansowanym modelom, systemom i aplikacjom bazodanowym. Obejmuje ona następujący materiał: rozdział 26. to wprowadzenie do kilku zaawansowanych modeli obejmujących aktywne bazy i wyzwalacze (podrozdział 26.1), czasowe bazy danych (podrozdział 26.2), przestrzenne bazy danych (podrozdział 26.3), multimedialne bazy danych (podrozdział 26.4) i dedukcyjne bazy danych (podrozdział 26.5). W rozdziale 27. omówiono wyszukiwanie informacji (ang. *Information Retrieval* — IR) i wyszukiwarki internetowe. Opisano tu zagadnienia takie jak IR, wyszukiwanie z użyciem słów kluczowych, porównanie baz danych z modelem IR, modele wyszukiwania, ocenę wyszukiwania i algorytmy rankingowe. Rozdział 28. to wprowadzenie do drażenia danych. Znajdziesz tu przegląd różnych metod z tego obszaru, w tym wykorzystanie reguł asocjacyjnych, grupowanie, klasyfikację i wykrywanie wzorców sekwencyjnych. Rozdział 29. zawiera przegląd zagadnień z obszaru hurtowni danych, w tym modeli i operacji oraz procesu budowania hurtowni.
- Część XII. (rozdział 30.) zawiera jeden rozdział poświęcony bezpieczeństwu baz danych. Znajdziesz tu omówienie poleceń SQL-a związanych z dyspozycyjnym systemem kontroli dostępu (GRANT, REVOKE), a także poziomów zabezpieczeń, modeli wbudowywania obowiązkowej kontroli dostępu w bazy relacyjne, zagrożeń takich jak ataki przez wstrzykiwanie kodu w SQL-u, a także technik i metod związanych z bezpieczeństwem i prywatnością danych.
- W dodatku A opisanych jest kilka różnych notacji do tworzenia diagramów przedstawiających koncepcyjne schematy ER lub EER. Wykładowca może stosować te notacje zamiast zapisu używanego przez nas. W dodatku B przedstawione są ważne fizyczne parametry dysków. Dodatek C zawiera przegląd graficznego języka zapytań QBE.

## Wskazówki dotyczące jak najlepszego wykorzystywania tej książki

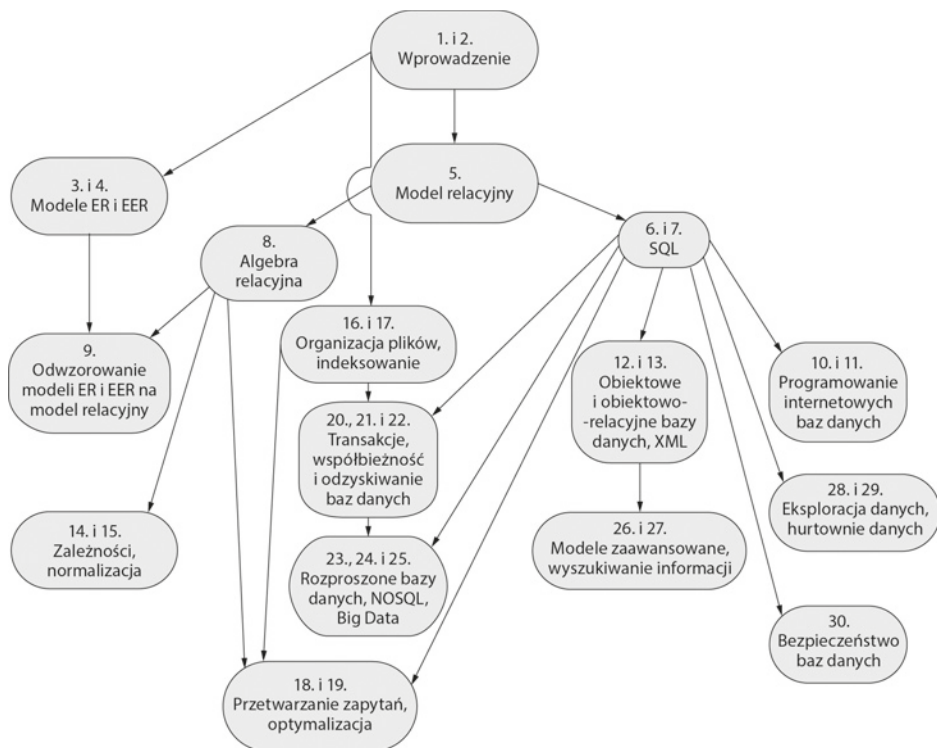
Istnieje oczywiście wiele różnych sposobów przeprowadzania kursów poświęconych bazom danych. Rozdziały pogrupowane w częściach od 1. do 7. mogą być wykorzystywane podczas zajęć wprowadzających do przedmiotu poświęconego systemom baz danych — kolejność ich omawiania zależy od strategii przyjętej przez prowadzącego. Wybrane rozdziały i podrozdziały można oczywiście pominąć, prowadzący zajęcia może także podjąć decyzję o rozszerzeniu materiału o pozostałe rozdziały (w zależności od tego, na które zagadnienia chce położyć szczególny nacisk). Na końcu każdego podrozdziału otwierającego (taki

podrozdział znajduje się na początku każdego rozdziału) umieszczono listę tych podrozdziałów, które można pominąć w sytuacji, gdy omówienie danego materiału nie wymaga szczegółowej analizy pewnych zagadnień. Podczas zajęć wprowadzających w świat baz danych zalecamy wykorzystanie rozdziałów 1. – 15. i dołączenie kilku wybranych fragmentów pozostałych rozdziałów (oczywiście w zależności od wiedzy studentów oraz rzeczywistego celu całego kursu). Przykładowo, kurs, w którym chcemy położyć szczególny nacisk na techniki implementacji systemów, powinien obejmować rozdziały z części 7., 8. i 9.

Treść rozdziałów 3. i 4., w których omówiliśmy modelowanie koncepcyjne w oparciu o modele ER i EER, może być szczególnie przydatna podczas nabywania gruntownej wiedzy na temat koncepcji z obszaru baz danych. Jeśli jednak prowadzący kurs będzie się chciał w większym stopniu skupić np. na implementacji systemów zarządzania bazami danych (SZBD), zawartość tych rozdziałów może zostać częściowo pominięta, przeniesiona na dalsze zajęcia lub w skrajnych przypadkach nawet w całości pominięta. Także materiał prezentowany w rozdziałach 16. i 17. (dotyczący organizacji plików i indeksowania) może zostać omówiony nieco wcześniej, nieco później lub wcale (jeśli celem zajęć jest np. prezentacja jedynie modeli i języków baz danych). W przypadku studentów, którzy uczestniczyli już w zajęciach z przedmiotu poświęconego organizacji plików, wybrane fragmenty tych rozdziałów mogą stanowić jedynie materiał do niezbędnych powtórek oraz cenne źródło ćwiczeń ułatwiających utrwalanie zdobytej wcześniej wiedzy.

Jeśli kurs dotyczy przede wszystkim projektowania baz danych, wykładowca powinien szybko omówić rozdziały 3. i 4., a następnie przejść do baz relacyjnych. Pełny cykl życia projektowania i implementowania bazy danych obejmuje przygotowanie projektu koncepcyjnego (patrz rozdziały 3. i 4.), bazy relacyjne (patrz rozdziały 5., 6. i 7.), odwzorowanie modelu danych (patrz rozdział 9.), normalizację (patrz rozdział 14.) oraz implementację w języku SQL (patrz rozdział 10.). Jeśli nacisk położony jest na programowanie baz internetowych i aplikacje internetowe, należy uwzględnić także rozdział 11. W większości przypadków wymagana jest także dodatkowa dokumentacja systemu zarządzania relacyjną bazą danych (ang. *relational database management system*, w skrócie *RDBMS*). Ta książka została napisana w taki sposób, aby prowadzący kurs mógł sam ustalać najlepszą jego zdaniem kolejność omawiania poszczególnych zagadnień. Przedstawiony poniżej diagram ilustruje główne zależności pomiędzy poszczególnymi rozdziałami. Zgodnie ze wskazaniem tego schematu, możliwe jest przejście do omawiania wielu różnych zagadnień już po omówieniu materiału wprowadzającego z pierwszych dwóch rozdziałów. Poniższy diagram może się co prawda wydać skomplikowany, jednak zasadnicze znaczenie ma jedynie omawianie zagadnień poruszanych w poszczególnych rozdziałach we właściwej kolejności (a więc bez naruszania zilustrowanych poniżej zależności). Diagram na następnej stronie może być pomocny dla wykładowców prowadzących równoległe zajęcia i szukających alternatywnych kolejności prezentacji.

Podczas półrocznego kursu opartego na tej książce niektóre rozdziały mogą zostać wyznaczone i przydzielone studentom jedynie jako materiał do przeczytania. Niniejsza książka może także być podstawą kursu całorocznego (dwusemestralnego). Zajęcia składające się na pierwszy semestr (taki przedmiot może nosić tytuł „Wprowadzenie do projektowania baz danych i systemów baz danych”) mogą wówczas obejmować większość materiału z rozdziałów od 1. do 15. Zajęcia składające się na drugi semestr takiego kursu (odpowiedni przedmiot może nosić tytuł „Modele i techniki implementowania baz danych”) mogą obejmować rozdziały od 16. do 30. W zależności od preferencji wykładowcy kurs całoroczny można też zaprojektować na wiele innych sposobów.



## Podziękowania

Jest nam ogromnie miło podziękować za wsparcie i udział w pracach nad tą książką licznej grupie osób. Po pierwsze, chcielibyśmy podziękować naszemu redaktorowi, Mattowi Goldsteinowi, za jego wskazówki, zachęty i wsparcie. Dziękujemy też Rose Kernan (za doskonałe zarządzanie pracami produkcyjnymi), Patricii Daly (za szczegółową redakcję tekstu), Marcie McMaster (za staranną korektę książki) i Scottowi Disanno (redaktorowi naczelnemu w zespole produkcyjnym). Chcemy też podziękować Kelsey Loanes z wydawnictwa Pearson za nieustanną pomoc przy tym projekcie, jak również recenzentom. Oto oni: Michael Doherty, Deborah Dunn, Imad Rahal, Karen Davis, Gilliean Lee, Leo Mark, Monisha Pulimood, Hassan Reza, Susan Vrbsky, Li Da Xu, Weining Zhang i Vincent Oria.

Ramez Elmasri dziękuje Kulsawasdowi Jitkajornwanichowi, Vivekowi Sharmie i Suriemu Swaminathanowi za pomoc w przygotowaniu fragmentów materiału z rozdziału 24. Sham Navathe jest wdzięczny wymienionym dalej osobom, które dokonały krytycznej oceny i poprawek materiałów dotyczących różnych zagadnień. Oto ci ludzie: Dan Forsythe i Satish Damle (za dyskusje na temat systemów składowania danych), Rafi Ahmed (za szczegółową zmianę uporządkowania materiałów dotyczących przetwarzania zapytań i optymalizacji), Harish Butani, Balaji Palanisamy i Prajakta Kalmegh (za pomoc w przygotowaniu materiałów na temat technologii Hadoop i MR), Vic Ghorpadey i Nenad Jukic (za poprawki w materiałach dotyczących hurtowni danych), Frank Rietta (za nowsze techniki z obszaru bezpieczeństwa baz danych), Kunal Malhotra (za różne dyskusje) i Saurav Sahay (za nowinki w dziedzinie systemów wyszukiwania danych).



Jeszcze raz chcemy podziękować osobom, które recenzowały wcześniejsze wydania *Wprowadzenia do systemów baz danych* i wniosły w nie swój wkład. Oto one:

- **Wydanie pierwsze.** Alan Apt (redaktor), Don Batory, Scott Downing, Dennis Heimbinger, Julia Hodges, Yannis Ioannidis, Jim Larson, Per-Ake Larson, Dennis McLeod, Rahul Patel, Nicholas Roussopoulos, David Stemple, Michael Stonebraker, Frank Tompa i Kyu-Young Whang.
- **Wydanie drugie.** Dan Joraanstad (redaktor), Rafi Ahmed, Antonio Albano, David Beech, Jose Blakeley, Panos Chrysanthis, Suzanne Dietrich, Vic Ghorpadey, Goetz Graefe, Eric Hanson, Junguk L. Kim, Roger King, Vram Kouramajian, Vijay Kumar, John Lowther, Sanjay Manchanda, Toshimi Minoura, Inderpal Mumick, Ed Omiecinski, Girish Pathak, Raghu Ramakrishnan, Ed Robertson, Eugene Sheng, David Stotts, Marianne Winslett i Stan Zdonick.
- **Wydanie trzecie.** Maite Suarez-Rivas i Katherine Harutunian (redaktorki), Suzanne Dietrich, Ed Omiecinski, Rafi Ahmed, Francois Bancilhon, Jose Blakeley, Rick Cattell, Ann Chervenak, David W. Embley, Henry A. Etlinger, Leonidas Fegaras, Dan Forsyth, Farshad Fotouhi, Michael Franklin, Sreejith Gopinath, Goetz Craefe, Richard Hull, Sushil Jajodia, Ramesh K. Karne, Harish Kotbagi, Vijay Kumar, Tarcisio Lima, Ramon A. Mata-Toledo, Jack McCaw, Dennis McLeod, Rokia Missaoui, Magdi Morsi, M. Narayanaswamy, Carlos Ordonez, Joan Peckham, Betty Salzberg, Ming-Chien Shan, Junping Sun, Rajshekhar Sunderraman, Aravindan Veerasamy i Emilia E. Villareal.
- **Wydanie czwarte.** Maite Suarez-Rivas, Katherine Harutunian, Daniel Rausch i Juliet Silveri (redaktorzy), Phil Bernhard, Zhengxin Chen, Jan Chomicki, Hakan Ferhatosmanoglu, Len Fisk, William Hankley, Ali R. Hurson, Vijay Kumar, Peretz Shoval i Jason T.L. Wang (recenzenci), Ed Omiecinski (wniósł wkład w rozdział 27.). Pomocne osoby z Uniwersytetu Teksańskiego w Arlington: Jack Fu, Hyoil Han, Babak Hojabri, Charley Li, Ande Swathi i Steven Wu. Pomocne osoby z uczelni Georgia Tech: Weimin Feng, Dan Forsythe, Angshuman Guin, Abrar Ul-Haque, Bin Liu, Ying Liu, Wanxia Xie i Waigen Yee.
- **Wydanie piąte.** Matt Goldstein i Katherine Harutunian (redaktorzy), Michelle Brown, Gillian Hall, Patty Mahtani, Maite Suarez-Rivas, Bethany Tidd, Joyce Cosentino Wells (z wydawnictwa Addison-Wesley), Hani Abu-Salem, Jamal R. Alsabbagh, Ramzi Bualuan, Soon Chung, Sumali Conlon, Hasan Davulcu, James Geller, Le Gruenwald, Latifur Khan, Herman Lam, Byung S. Lee, Donald Sanderson, Jamil Saquer, Costas Tsatsoulis i Jack C. Wileden (recenzenci), Raj Sunderraman (opracował projekty praktyczne), Salman Azar (przygotował zestaw nowych ćwiczeń), Gaurav Bhatia, Fariborz Farahmand, Ying Liu, Ed Omiecinski, Nalini Polavarapu, Liora Sahar, Saurav Sahay i Wanxia Xie (z uczelni Georgia Tech).
- **Wydanie szóste.** Matt Goldstein (redaktor), Gillian Hall (zarządzanie pracami produkcyjnymi), Rebecca Greenberg (redakcja), Jeff Holcomb, Marilyn Lloyd, Margaret Waples, Chelsea Bell (z wydawnictwa Pearson), Rafi Ahmed, Venu Dasigi, Neha Deodhar, Fariborz Farahmand, Hariprasad Kumar, Leo Mark, Ed Omiecinski, Balaji Palanisamy, Nalini Polavarapu, Parimala R. Pranesh, Bharath Rengarajan, Liora Sahar, Saurav Sahay, Narsi Srinivasan i Wanxia Xie.

Na końcu chcielibyśmy gorąco podziękować naszym rodzinom za ich wsparcie, zachęty i cierpliwość.

R.E.

S.B.N.

---

## O autorach

**Ramez Elmasri** jest profesorem i zastępcą dyrektora na wydziale Informatyki i Inżynierii Uniwersytetu Teksańskiego w Arlington. Opublikował ponad 140 recenzowanych prac naukowych oraz był promotorem 16 uczestników studiów doktoranckich i ponad 100 uczestników studiów magisterskich. Jego badania dotyczą wielu obszarów zarządzania bazami danych i big data, w tym modelowania koncepcyjnego, integracji danych, języków zapytań, technik indeksowania, baz czasowych i czasowo-przestrzennych, baz bioinformatycznych, rejestrowania danych z sieci czujników, a także drążenia i analizy danych przestrzennych oraz czasowoprzestrzennych. Elmasri pracował jako konsultant dla różnych firm, takich jak Digital, Honeywell, Hewlett Packard i Action Technologies, a także doradzał kancelariom prawnym w zakresie patentów. Był dyrektorem programowym konferencji 1993 International Conference on Conceptual Modeling (ER) i wicedyrektorem programowym konferencji 1994 IEEE International Conference on Data Engineering. Zasiadał w Komitecie Sterującym konferencji ER, a także w radach programowych wielu innych konferencji. Prowadził kilka wykładów na konferencjach VLDB, ICDE i ER. Jest współautorem książki *Operating Systems: A Spiral Approach* (McGraw-Hill, 2009), napisanej razem z Gilem Carrickiem i Davidem Levinem. W 1999 r. Elsmari otrzymał nagrodę UTA College of Engineering Outstanding Teaching Award. Uzyskał dyplomy licencjata inżynierii na Uniwersytecie Aleksandryjskim oraz magistra i doktora informatyki na Uniwersytecie Stanforda.

**Shamkant B. Navathe** jest profesorem i założycielem grupy badawczej ds. baz danych na Wydziale Informatyki Georgia Institute of Technology w Atlancie. Pracował w działach badawczych firm IBM i Siemens. Doradzał też różnym firmom, takim jak Digital, Computer Corporation of America, Hewlett Packard, Equifax i Persistent Systems. Był współprzewodniczącym konferencji 1996 International VLDB (Very Large Data Base) w Bombaju. Był też współprzewodniczącym ds. programowych konferencji ACM SIGMOD 1985 International Conference i współprzewodniczącym warsztatów IFIP WG 2.6 Data Semantics Workshop w 1995 r. Działał w fundacji VLDB i zasiadał w komitetach sterujących kilku konferencji. Był redaktorem współpracującym w różnych magazynach naukowych, w tym „ACM Computing Surveys” i „IEEE Transactions on Knowledge and Data Engineering”. Jest też współautorem książki *Conceptual Design: An Entity Relationship Approach* (Addison Wesley, 1992; pozostali autorzy: Carlo Batini i Stefano Ceri). Navathe jest członkiem organizacji Association for Computing Machinery (ACM), a w 2015 r. otrzymał nagrodę IEEE TCDE Computer Science, Engineering and Education Impact. Uzyskał dyplom doktora na Uniwersytecie Michigan i opublikował ponad 150 recenzowanych prac w magazynach i materiałach konferencyjnych.





I

---

# Wprowadzenie do baz danych



# Bazy danych i ich użytkownicy

Bazy danych i systemy baz danych stały się kluczowym narzędziem w życiu codziennym współczesnego społeczeństwa. Niemal codziennie każdy z nas wykonuje wiele czynności, które w taki czy inny sposób wiążą się z wykorzystywaniem bazy danych. Przykładowo, kiedy idziemy do banku celem wpłacenia lub wypłacenia pieniędzy z naszego konta, kiedy rezerwujemy pokój w hotelu lub bilet lotniczy, kiedy szukamy interesujących nas pozycji w skomputeryzowanym katalogu biblioteki lub kiedy kupujemy cokolwiek (książkę, zabawkę czy choćby komputer) w sklepie internetowym obsługiwanym za pośrednictwem strony WWW, jest bardzo prawdopodobne, że tego typu czynności wymagają od kogoś lub od jakiegoś programu komputerowego właśnie dostępu do bazy danych. Nawet zwykłe zakupy w supermarkecie w wielu przypadkach wiążą się obecnie z automatyczną aktualizacją bazy danych reprezentującej stan zapasów poszczególnych artykułów.

Wymienione powyżej działania są przykładami tego, co możemy nazwać **tradycyjnymi zastosowaniami baz danych**, w których większość przechowywanych i uzyskiwanych informacji ma postać albo danych tekstowych, albo danych numerycznych. Przez ostatnie kilka lat stały postęp technologiczny doprowadził do wynalezienia zupełnie nowych, pasjonujących zastosowań dla systemów baz danych. Rozpowszechnienie się serwisów społecznościowych takich jak Facebook, Twitter i Flickr wymagało opracowania bardzo dużych baz danych przechowujących nietradycyjne dane, np.: posty, tweety, zdjęcia i filmy wideo. Stworzono nowe typy systemów baz danych, często nazywane **systemami składowania big data** lub **systemami typu NOSQL**, przeznaczone do zarządzania danymi w serwisach społecznościowych. Systemy tego rodzaju są też używane przez firmy takie jak Google, Amazon i Yahoo! do zarządzania danymi potrzebnymi w wyszukiwarkach, a także w **pamięci masowej w chmurze**, gdzie użytkownicy otrzymują możliwość składowania danych w sieci WWW w celu zarządzania wszelkimi typami danych, w tym dokumentami, programami, zdjęciami, filmami i e-mailami. Przegląd nowych typów systemów baz danych zawiera rozdział 24.

W tym miejscu wymienimy kilka innych zastosowań baz danych. Powszechna obsługa rejestrowania zdjęć i filmów w telefonach komórkowych i innych urządzeniach umożliwiła cyfrowe zapisywanie fotografii oraz nagrań audio i wideo. Pliki tego rodzaju stają się ważnym elementem **multimedialnych baz danych**. **Geograficzne systemy informacyjne** (ang. *Geographic Information Systems — GIS*) mogą przechowywać i analizować mapy, dane o pogodzie, a nawet zdjęcia satelitarne. **Hurtownie danych i systemy OLAP** (od ang. *Online Analytical Processing*) są wykorzystywane w wielu firmach do wydzielania z ogromnych baz danych przydatnych informacji i ich analizy, która ma ostatecznie ułatwić podjęcie właściwych decyzji. Technologie **baz danych czasu rzeczywistego i aktywnych baz danych** są przydatne podczas sterowania kontrolą i procesami w produkcji

przemysłowej. Okazuje się także, że **techniki przeszukiwania** baz danych znalazły zastosowanie w świecie witryn internetowych, gdzie usprawniono mechanizmy wyszukiwania przez użytkowników potrzebnych informacji w internecie.

Aby zrozumieć podstawy tych technologii, musimy jednak rozpocząć nasze rozważania od analizy bardziej tradycyjnych zastosowań baz danych. W podrozdziale 1.1 zdefiniujemy więc to, czym faktycznie jest baza danych, i dopiero potem przystąpimy do omawiania innych podstawowych pojęć. W podrozdziale 1.2 przedstawimy prosty przykład bazy danych *UNIwersytet*, który będzie stanowił dobrą ilustrację dla naszych rozważań. W podrozdziale 1.3 opiszemy niektóre spośród najważniejszych własności systemów baz danych, natomiast w podrozdziałach 1.4 i 1.5 spróbujemy odpowiednio skategoryzować typy pracowników, których praca wiąże się z wykorzystywaniem systemów baz danych. Podrozdziały 1.6, 1.7 oraz 1.8 zawierają bardziej szczegółową analizę możliwości oferowanych przez systemy baz danych oraz ich typowych zastosowań. Treść podrozdziału 1.9 będzie podsumowaniem całego rozdziału.

Czytelnicy, którzy chcieliby bardzo szybko przebrnąć przez teoretyczne wprowadzenie do systemów baz danych, mogą przeczytać jedynie podrozdziały od 1.1 do 1.5, pomijając podrozdziały od 1.6 do 1.8 i od razu przejść do rozdziału 2.

## 1.1. Wprowadzenie

Bazy danych i technologie baz danych w ogromnym stopniu przyczyniają się do wzrostu liczby zastosowań komputerów. Możemy z pełnym przekonaniem stwierdzić, że właśnie bazy danych odgrywają główną rolę niemal we wszystkich obszarach, w których w ogóle wykorzystuje się komputery — wystarczy wymienić takie gałęzie jak biznes, handel elektroniczny, media społecznościowe, inżynieria, medycyna, genetyka, prawo, edukacja czy bibliotekarstwo. Określenie *baza danych* jest dziś na tyle popularne, że powinniśmy rozpocząć nasze rozważania od zdefiniowania, czym faktycznie jest baza danych. Przedstawiona poniżej definicja będzie na razie dosyć ogólna.

**Baza danych** jest zbiorem powiązanych ze sobą danych. Przez **dane** rozumiemy w tym przypadku znane fakty, które można jakoś zarejestrować, i które mają konkretne znaczenie. Przykładowo, rozważmy nazwiska, numery telefonów oraz adresy osób, które znamy. Obecnie takie dane są zwykle przechowywane w telefonach komórkowych, gdzie działa proste oprogramowanie bazodanowe. Ponadto takie dane mogą być zarejestrowane w odpowiednio indeksowanej książce adresowej lub przechowywane na twardym dysku komputera z wykorzystaniem oprogramowania (choćby aplikacji Microsoft Access lub Microsoft Excel). Mamy więc zbiór powiązanych ze sobą danych o konkretnym znaczeniu, co oznacza, że dysponujemy bazą danych.

Powyższa definicja bazy danych jest dosyć ogólna; przykładowo, możemy traktować zbiór słów składających się na tę stronę tekstu jako zbiór wzajemnie powiązanych danych, a więc jako bazę danych. Pojęcie *bazy danych* w powszechnym rozumieniu jest jednak znacznie węższe. Każda taka baza danych musi mieć następujące własności:

- Baza danych reprezentuje jakiś wybrany aspekt świata rzeczywistego, nazywany niekiedy **mini-światem** lub **dziedziną problemu** (ang. *Universe of Discourse* — *UoD*). Zmiany w takim mini-świecie muszą być uwzględniane w bazie danych.
- Baza danych jest logicznie koherentnym zbiorem danych z jakimś spójnym znaczeniem. Przypadkowy zbiór danych nie jest prawidłową bazą danych i nie należy dla niego stosować tego określenia.
- Baza danych jest projektowana, konstruowana i wypełniana danymi w określonym celu. Do bazy danych powinna być przypisana grupa docelowych użytkowników oraz z góry przyjęte zastosowania, które będą realizowane przez tych użytkowników.

Innymi słowy, każda baza danych powinna mieć pewne źródło, z którego będą pochodzić przechowywane dane, pewien stopień interakcji ze zdarzeniami mającymi miejsce w reprezentowanym przez nią wycinku świata rzeczywistego oraz użytkowników, którzy są aktywnie zainteresowani jej zawartością. Użytkownicy bazy mogą przeprowadzać transakcje biznesowe (np. klient może kupić aparat). Mogą też nastąpić zdarzenia (takie jak urodzenie dziecka przez pracownika) powodujące zmianę informacji w bazie. Aby baza zawsze była precyzyjna i wiarygodna, musi stanowić wierne odzwierciedlenie reprezentowanego przez nią mini-świata. Dlatego zmiany trzeba odzwierciedlać w bazie tak szybko, jak tylko to możliwe.

Bazy danych mogą mieć dowolne rozmiary i zróżnicowane poziomy złożoności. Przykładowo, lista nazwisk i adresów może się składać tylko z kilkuset rekordów, z których każdy ma stosunkowo prostą strukturę. Z drugiej strony, skomputeryzowany katalog zbiorów wielkiej biblioteki może się składać z pół miliona wpisów zorganizowanych w różnych kategoriach (według nazwiska autora, tematu, tytułu itp.), które są z kolei uporządkowane w kolejności alfabetycznej. Jeszcze większa i bardziej skomplikowana baza danych jest utrzymywana przez serwisy społecznościowe takie jak Facebook, który ma ponad miliard użytkowników. W takiej bazie trzeba przechowywać informacje o tym, którzy użytkownicy są powiązani z innymi jako *przyjaciele*, posty każdego użytkownika, dane o użytkownikach mogących zobaczyć poszczególne posty i rozbudowane informacje innego rodzaju potrzebne do prawidłowego działania witryny. W witrynach tego rodzaju potrzebnych jest wiele baz, aby śledzić stale zmieniające się informacje niezbędne w serwisach społecznościowych.

Przykładową dużą bazę z obszaru handlu elektronicznego przechowuje Amazon.com. Zawiera ona dane na temat ponad 60 milionów aktywnych użytkowników oraz milionów książek, płyt CD, filmów, płyt DVD, gier, urządzeń elektronicznych, odzieży itd. Baza ta zajmuje ponad 42 terabajty (TB; terabajt to  $10^{12}$  bajtów) i jest przechowywana na setkach komputerów (zwanych serwerami). Miliony osób każdego dnia odwiedzają witrynę Amazon.com i korzystają z opisanej bazy w trakcie zakupów. Ta baza jest stale aktualizowana, gdy do oferty dodawane są nowe książki i inne produkty. Po dokonaniu transakcji aktualizowane są stany magazynu.

Baza danych może być generowana i utrzymywana ręcznie lub może zostać całkowicie skomputeryzowana. Przykładowo, katalog kart bibliotecznych jest tym rodzajem bazy danych, który można stworzyć i utrzymywać bez pomocy komputerów. Skomputeryzowane bazy danych mogą być tworzone i utrzymywane zarówno za pomocą grupy programistów napisanych specjalnie z myślą o tym zadaniu, jak i przez system zarządzania bazą danych. W tej książce będziemy się oczywiście skupiali wyłącznie na skomputeryzowanych bazach danych.

**System zarządzania bazą danych** (w skrócie **SZBD**, ang. *Database Management System* — *DBMS*) jest zbiorem programów, które umożliwiają tworzenie i utrzymywanie bazy danych. SZBD jest więc *uniwersalnym systemem programowym*, który ułatwia *definiowanie, konstruowanie, manipulowanie i udostępnianie* baz danych różnym użytkownikom i aplikacjom. **Definiowanie** bazy danych wiąże się z określaniem typów i struktur danych oraz ograniczeń dla przechowywanych informacji. W SZBD przechowywane są też definicje bazy lub informacje opisowe w postaci katalogu lub słownika bazy danych. Są to tzw. **metadane**. **Konstruowanie** bazy danych jest kontrolowanym przez SZBD procesem umieszczania w niej właściwych informacji na jakimś medium przechowywania. **Manipulowanie** bazą danych obejmuje takie działania jak wykonywanie zapytań wyciągających z bazy określone informacje, aktualizowanie bazy danych w taki sposób, aby zawarte w niej informacje odzwierciedlały rzeczywisty stan reprezentowanego miniświata, oraz generowanie raportów na podstawie informacji zawartych w bazie danych. **Udostępnianie** bazy danych umożliwia wielu użytkownikom i programom jednocześnie operowanie na zawartych w niej informacjach.

**Aplikacja** uzyskuje dostęp do bazy, wysyłając zapytania lub żądania do SZBD. **Zapytanie**<sup>1</sup> zwykle powoduje pobranie jakichś danych. **Transakcja** może powodować wczytanie jakichś danych i zapisanie pewnych informacji w bazie.

Do pozostałych funkcji oferowanych przez systemy zarządzania bazami danych należy *ochrona* bazy danych oraz *konserwowanie* jej przez długi okres czasu. Proces **ochrony** bazy danych obejmuje zarówno *ochronę systemową* przed niewłaściwym działaniem (lub awariami) sprzętu oraz oprogramowania, jak i *zabezpieczanie* przed nieuprawnionym dostępem. Cykl życia wielkich baz danych może trwać wiele lat, zatem systemy zarządzania takimi bazami danych muszą umożliwiać ich **konserwowanie**, tak aby możliwe było dostosowywanie ich do ewoluujących wymagań.

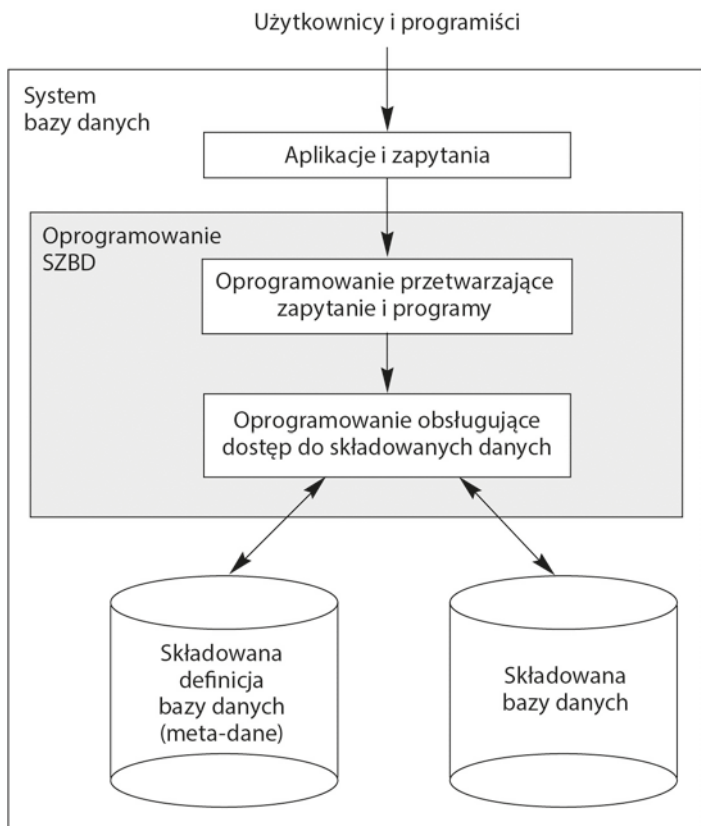
Implementowanie skomputeryzowanych baz danych wcale nie wymaga stosowania uniwersalnego oprogramowania typu SZBD. Możemy przecież napisać własny zbiór programów tworzących i utrzymujących naszą bazę danych, tworząc tym samym własne, *specyficzne* oprogramowanie typu SZBD do konkretnych zastosowań, np. do obsługi rezerwacji w liniach lotniczych. W obu przypadkach (a więc niezależnie od tego, czy zdecydowaliśmy się wykorzystać uniwersalny SZBD, czy nie) musimy zwykle wdrożyć w docelowym systemie komputerowym dość skomplikowany pakiet oprogramowania. W rzeczywistości większość systemów zarządzania bazami danych jest bardzo skomplikowanymi systemami oprogramowania.

Aby prezentowany w tym podrozdziale zbiór definicji wprowadzających był kompletny, musimy jeszcze określić, czym jest **system bazy danych** — otóż jest to połączenie samej bazy danych z oprogramowaniem SZBD. Rysunek 1.1 ilustruje niektóre spośród omówionych do tej pory zagadnień.

---

<sup>1</sup> Pojęcie *zapytanie*, pierwotnie oznaczające pytanie, jest czasem nieformalnie używane do określania wszelkiego rodzaju interakcji z bazami, w tym do modyfikowania danych.





RYSUNEK 1.1. Uprozczone środowisko systemu bazy danych

## 1.2. Przykład

Przeanalizujemy teraz prosty przykład, który większości czytelników tej książki może wydać się znajomy — chodzi o bazę danych `UNIWERSYTET` reprezentującą informacje dotyczące studentów, przedmiotów i ocen w środowisku wyższej uczelni. Na rysunku 1.2 przedstawiono strukturę bazy danych wraz z kilkoma przykładowymi informacjami wypełniającymi poszczególne tabele. Prezentowana baza danych składa się z pięciu plików, z których każdy zawiera **rekordy danych** tego samego typu<sup>2</sup>. Plik `STUDENT` zawiera dane na temat poszczególnych studentów, plik `PRZEDMIOT` — na temat poszczególnych przedmiotów, plik `KURS` — na temat poszczególnych kursów z odpowiednich przedmiotów, plik `RAPORT_OCEN` zawiera dane na temat ocen uzyskanych przez studentów podczas zaliczania poszczególnych kursów, natomiast plik `WYMAGANIE_WSTEPNE` — informacje o wymaganiach odnośnie do zaliczonych zajęć przed uzyskaniem możliwości uczestnictwa w zajęciach z poszczególnych przedmiotów.

<sup>2</sup> Wykorzystujemy w tym miejscu pojęcie *pliku* w sposób nieformalny. Na poziomie koncepcyjnym *plik* jest *zbiorem* rekordów, które mogą (choć nie muszą) być uporządkowane.

Aby *zdefiniować* taką bazę danych, musimy określić strukturę rekordów przechowywanych w każdym z plików przez sprecyzowanie różnych typów **elementów danych**, które mają być przechowywane w kolejnych rekordach. Na rysunku 1.2 każdy rekord pliku STUDENT zawiera dane reprezentujące nazwisko studenta (element danych *Nazwisko*), numer indeksu (element danych *NumerIndeksu*), rok studiów (element danych *RokSt*, który może zawierać takie wartości jak *pierwszy* lub *1*, *Drugi* lub *2*, ...) oraz kierunek (element danych *Kierunek*, który może zawierać takie wartości jak *MAT*, *Informatyka* lub *INF*, ...). Każdy rekord pliku PRZEDMIOT zawiera dane reprezentujące nazwę przedmiotu (element danych *NazwaPrzedmiotu*), identyfikator przedmiotu (element danych *Numer* → *Przedmiotu*), liczbę godzin tygodniowo (element danych *Godziny*) oraz wydział, na którym dany przedmiot jest wykładany (element danych *Wydział*), itd. Dla każdego takiego elementu danych wewnątrz rekordu musimy także określić **typ danych**. Przykładowo, możemy określić, że element danych *Nazwisko* w pliku STUDENT jest ciągiem znaków alfabety, element danych *NumerIndeksu* w tym samym pliku jest liczbą całkowitą, natomiast element danych *Ocena* w pliku RAPORT\_OCEN jest pojedynczą wartością ze zbioru {5, 4, 3, 2}. Do reprezentowania wartości poszczególnych elementów danych możemy także użyć odpowiedniego schematu kodowania. Przykładowo, w widocznym na rysunku 1.2 elemencie danych *RokSt* w pliku STUDENT wartość *1* lub *Pierwszy* reprezentuje studenta pierwszego roku, wartość *2* lub *Drugi* studenta drugiego roku, wartość *3* lub *Trzeci* studenta trzeciego roku, wartość *4* lub *Czwarty* studenta czwartego roku, a wartość *5* lub *Piąty* — studenta piątego roku. Aby *skonstruować* bazę danych UNIWERSYTET, zapisujemy dane, które mają reprezentować poszczególnych studentów, przedmioty, kursy, oceny i wymagania w postaci rekordu w odpowiednim pliku. Łatwo zauważyć, że rekordy w różnych plikach mogą być ze sobą powiązane. Przykładowo, rekord *Nowak* w pliku STUDENT może być powiązany z dwoma rekordami w pliku RAPORT\_OCEN, który przechowuje oceny, jakie student o takim nazwisku uzyskał z dwóch zaliczanych przezeń kursów. Podobnie, każdy rekord w pliku WYMAGANIE\_WSTEPNE jest związany z dwoma rekordami reprezentującymi dwa różne przedmioty: przedmiot, dla którego zdefiniowano dane wymaganie, oraz przedmiot, którego zaliczenie stanowi warunek uczestnictwa w zajęciach z pierwszego przedmiotu. Większość średnich i wielkich baz danych składa się z wielu typów rekordów i zawiera *wiele związków* pomiędzy tymi rekordami.

*Manipulowanie* bazą danych wiąże się z wykonywaniem zapytań i aktualizacją zawartych w niej informacji. Przykładowo, zapytania mogą mieć następującą postać:

- „wygeneruj listę wszystkich przedmiotów, w których uczestniczył student o nazwisku *Nowak*, wraz z uzyskanymi ocenami”,
- „wygeneruj listę nazwisk studentów, którzy brali udział w zajęciach z danego kursu w ramach przedmiotu *Bazy Danych* w semestrze jesiennym 2008 roku wraz z uzyskanymi przez nich ocenami z tego kursu”,
- „jakie wymagania należy spełnić przed przystąpieniem do zajęć z przedmiotu *Bazy danych*?”.

Przykładowe instrukcje aktualizujące mogą natomiast mieć postać:

- „zmień rok studiów studenta *Nowak* na drugi (*Drugi*)”,
- „stwórz nowy kurs dla przedmiotu *Bazy danych* prowadzonego w bieżącym semestrze”,
- „zapisz ocenę 5 dla studenta *Nowak* za zaliczenie w ostatnim semestrze wskazanego kursu z przedmiotu *Bazy danych*”.

**STUDENT**

NazwaPrzedmiotu	NumerIndeksu	Rok	Kierunek
Nowak	17	1	INF
Kowalski	8	2	INF

**PRZEDMIOT**

NazwaPrzedmiotu	NumerPrzedmiotu	Godziny	Wydział
Wprowadzenie do informatyki	INF1310	4	INF
Struktury danych	INF3320	4	INF
Matematyka dyskretna	MAT2410	3	MAT
Bazy danych	INF3380	3	INF

**DZIAŁ**

IdentyfikatorDziału	NumerPrzedmiotu	Semestr	Rok	Prowadzący
85	MAT2410	Jesienny	07	Kaczmarek
92	INF1310	Jesienny	07	Asnyk
102	INF3320	Wiosenny	08	Kowalik
112	MAT2410	Jesienny	08	Chłopicki
119	INF1310	Jesienny	08	Asnyk
135	INF3380	Jesienny	08	Szamotulski

**RAPORT\_OCEN**

NumerIndeksu	IdentyfikatorDziału	Ocena
17	112	4
17	119	3
8	85	5
8	92	5
8	102	4
8	135	5

**WYMAGANIA\_WSTĘPNE**

NumerPrzedmiotu	NumerPWymagania
INF3380	INF3320
INF3380	MAT2410
INF3320	INF1310

RYSUNEK 1.2. Baza danych zawierająca informacje o studentach i przedmiotach

Przetworzenie wymienionych przed chwilą nieformalnych zapytań i instrukcji aktualizujących wymaga oczywiście nadania im postaci precyzyjnych poleceń zapisanych w języku zapytań, który jest obsługiwany w wykorzystywanym systemie zarządzania bazą danych.

W tym miejscu warto opisać bazy danych w ramach większego przedsięwzięcia, jakim jest budowanie systemu informatycznego organizacji. Dział informatyczny (IT) firmy projektuje i konserwuje system informatyczny obejmujący różne komputery, systemy składo-

wania danych, oprogramowanie i bazy danych. Projektowanie nowej aplikacji używającej istniejącej bazy lub projektowanie zupełnie nowych baz rozpoczyna się od etapu **określenia specyfikacji i analizowania wymagań**. Wymagania są szczegółowo dokumentowane i przekształcane w projekt koncepcyjny, który można przedstawiać i modyfikować za pomocą różnych narzędzi komputerowych. Dzięki nim taki projekt można łatwo konserwować, modyfikować i przekształcać w implementację bazy danych (używany do tego model ER przedstawiamy w rozdziale 3.). Projekt koncepcyjny jest następnie przekształcany w **projekt logiczny**, który można zaprezentować w formie modelu danych zaimplementowanego w komercyjnym systemie SZBD. W książce omawiamy różne rodzaje systemów SZBD z naciskiem na systemy relacyjne opisywane w rozdziałach od 5. do 9.

Na ostatnim etapie powstaje **projekt fizyczny**. Wtedy to dokładnie określana jest specyfikacja składowania danych i dostępu do nich. Projekt bazy zostaje zaimplementowany, baza jest zapełniana danymi, a następnie stale konserwowana, aby odzwierciedlała stan mini-świata.

### 1.3. Właściwości rozwiązań opartych na bazach danych

Wiele unikatowych właściwości odróżnia rozwiązania oparte na bazach danych od starszych metod programowania aplikacji wykorzystujących pliki z danymi. W tradycyjnym **przetwarzaniu plików** każdy użytkownik definiuje i implementuje pliki, które są mu potrzebne do konkretnego zastosowania danego programu, i czynność ta stanowi jeden z elementów programowania tego zastosowania. Przykładowo, jeden użytkownik — przyjmijmy, że jest to *dziekanat* — może utrzymywać plik z informacjami o studentach i uzyskanych przez nich ocenach. Programy drukujące listy studentów i umożliwiające wpisywanie do pliku nowych ocen są implementowane w postaci jednego z elementów tego zastosowania. Drugi użytkownik — niech to będzie tym razem *dział księgowości* — może zarządzać przyznawaniem i wypłacaniem stypendiów. Chociaż oba podmioty są zainteresowane danymi na temat studentów, każdy utrzymuje własne pliki (wraz z programami operującymi na tych plikach), ponieważ każdy z tych podmiotów wymaga pewnych danych, które nie są dostępne w plikach utrzymywanych przez pozostałe podmioty. Efektem tej nadmiarowości w definiowaniu i przechowywaniu danych jest niepotrzebna strata wykorzystywanej przestrzeni oraz zwiększony koszt utrzymywania aktualnych informacji.

Typowe rozwiązanie oparte na bazie danych przewiduje, że utrzymywane jest tylko jedno repozytorium danych, które, raz utworzone, jest stosowane przez wielu różnych użytkowników za pomocą zapytań, transakcji i aplikacji. Poniżej przedstawiono główne cechy odróżniające rozwiązania oparte na bazach danych od rozwiązań opartych na przetwarzaniu plików:

- samoopisująca natura systemów baz danych,
- oddzielenie programów od danych oraz abstrakcja danych,
- obsługa wielu perspektyw dla tych samych danych,
- współdzielenie danych oraz przetwarzanie transakcji.

W poniższych punktach opiszemy oddzielnie każdą z tych właściwości. Dodatkowe omówienie własności systemów baz danych można znaleźć w podrozdziałach 1.6 i 1.8.

### 1.3.1. Samoopisująca natura systemów baz danych

Podstawowa właściwość rozwiązań opartych na stosowaniu baz danych określa, że system bazy danych zawiera nie tylko samą bazę danych, ale także kompletną definicję lub opis jej struktury i ograniczeń. Wspomniana definicja jest przechowywana w katalogu systemu zarządzania bazą danych (SZBD), który zawiera informacje odnośnie do struktury poszczególnych plików, typu i formatu przechowywania poszczególnych elementów danych oraz rozmaitych ograniczeń nałożonych na te dane. Informacje przechowywane w katalogu są często nazywane **metadanymi**. Metadane zawsze opisują strukturę głównej bazy danych (patrz rysunek 1.1). Warto zauważyć, że niektóre nowsze systemy baz danych, tzw. systemy NOSQL, nie wymagają metadanych. Zamiast tego dane są przechowywane w formie **danych samoopisujących**, które obejmują w jednej strukturze nazwy i wartości elementów danych (patrz rozdział 24.).

Katalog jest wykorzystywany nie tylko przez oprogramowanie systemu zarządzania bazą danych, ale także przez jej użytkowników, którzy potrzebują informacji na temat struktury bazy danych. Uniwersalne pakiety programowe SZBD nie są tworzone z myślą o konkretnych zastosowaniach baz danych, a więc muszą się odwoływać do katalogów celem uzyskania informacji o strukturze plików w konkretnych bazach danych (np. aby dysponować niezbędną wiedzą na temat typu i formatu danych, które mają zostać odczytane lub zapisane). Oprogramowanie systemu zarządzania bazą danych, które wykorzystuje katalog do przechowywania definicji baz danych, musi się sprawdzać tak samo dobrze *we wszystkich możliwych zastosowaniach obsługiwanych baz danych* — niezależnie od tego, czy jest to baza danych uniwersytetu, banku, czy jakiegokolwiek innego podmiotu.

W tradycyjnym modelu przetwarzania plików definicja danych jest zwykle częścią samych aplikacji wykorzystywanych do ich obsługi. Oznacza to, że możliwości tych programów ograniczają się do pracy *tylko z jedną bazą danych*, której struktura została w nich z góry zadeklarowana. Przykładowo, aplikacja napisana w języku programowania C++ może zawierać deklaracje struktur lub klas. O ile oprogramowanie przetwarzające pliki może uzyskiwać dostęp wyłącznie do określonych baz danych, oprogramowanie systemów zarządzania bazami danych może obsługiwać rozmaite bazy danych przez odczytywanie i odpowiednie wykorzystywanie ich definicji zawartych w katalogu.

W przykładzie zaprezentowanym na rysunku 1.2 katalog wykorzystywanego systemu SZBD będzie przechowywał definicje wszystkich przedstawionych plików. Rysunek 1.3 przedstawia wybrane wpisy z katalogu bazy danych. Za każdym razem, gdy do systemu zarządzania bazą danych dociera żądanie dostępu, np. wartości elementu danych *Nazwisko* rekordu w pliku *STUDENT*, oprogramowanie tego systemu odwołuje się do katalogu, aby określić strukturę pliku *STUDENT* oraz pozycję i rozmiar elementu danych *Nazwisko* wewnątrz danego rekordu. Zupełnie inaczej byłoby w przypadku typowej aplikacji przetwarzającej pliki, gdzie struktura pliku i — w ekstremalnej sytuacji — także dokładne położenie elementu danych *Nazwisko* wewnątrz rekordu *STUDENT* są na stałe zakodowane wewnątrz każdego programu, który może uzyskać dostęp do składowanych w ten sposób informacji.

RELACJE

Nazwa_relacji	Liczba_kolumn
STUDENT	4
PRZEDMIOT	4
DZIAŁ	5
RAPORT_OCEN	3
WYMAGANIE_WSTĘPNE	2

KOLUMNY

Nazwa_kolumny	Typ_danych	Należy_do_relacji
Nazwisko	Character (30)	STUDENT
NumerIndeksu	Character (4)	STUDENT
Rok	Integer (1)	STUDENT
Kierunek	TypKierunku	STUDENT
Nazwa_przedmiotu	Character (10)	PRZEDMIOT
Numer_przedmiotu	XXXXNNNN	PRZEDMIOT
...	...	...
...	...	...
...	...	...
NumerWymagania	XXXXNNNN	WYMAGANIE_WSTĘPNE

RYSUNEK 1.3. Przykładowy katalog bazy danych z rysunku 1.2

*Uwaga:* TypKierunku jest zdefiniowany jako typ wyliczeniowy obejmujący wszystkie znane kierunki. XXXXNNNN służy do definiowania typu za pomocą czterech liter, po których następują cztery cyfry.

1.3.2. Oddzielenie programów od danych oraz abstrakcja danych

W tradycyjnej metodzie przetwarzania plików struktura plików z danymi jest umieszczana w wykorzystywanych programach, zatem wszystkie ewentualne zmiany tej struktury mogą wymagać odpowiedniego *zmodyfikowania wszystkich programów*, które uzyskują dostęp do danego pliku. Zupełnie inaczej jest w przypadku programów wykorzystujących pośrednictwo systemów zarządzania bazami danymi, które w większości podobnych sytuacji nie wymagają tego typu dodatkowych zmian. Struktura plików z danymi jest przechowywana w specjalnym katalogu systemu SZBD, dzięki czemu jest oddzielona od programów dostępowych. Taki model nazywamy **niezależnością programu od danych**.

Przykładowo, pewien program uzyskujący dostęp do pliku mógłby zostać napisany w taki sposób, że będzie mógł pracować wyłącznie z rekordami pliku STUDENT, których struktura jest identyczna jak ta przedstawiona na rysunku 1.4. Jeśli uznamy za konieczne dodanie kolejnego elementu danych do każdego rekordu w tym pliku, np. reprezentujący datę urodzenia element *DataUrodzenia*, tak skonstruowany program nie będzie mógł dłużej działać i najprawdopodobniej konieczna będzie ingerencja w jego kod źródłowy.



Nazwa elementu danych	Pozycja początkowa w rekordzie	Długość w znakach (bajtach)
Nazwisko	1	30
Numer indeksu	31	4
Rok	35	4
Kierunek	39	4

RYSUNEK 1.4. Wewnętrzny format przechowywania dla rekordu z pliku STUDENT oparty na katalogu bazy danych z rysunku 1.3

Gdybyśmy zamiast tego programu użyli środowiska SZBD, musielibyśmy jedynie zmienić w katalogu opis rekordów pliku STUDENT w taki sposób, aby uwzględniły dodanie nowego elementu danych *DataUrodzenia* — zmiana struktury nie pociągałaby więc za sobą konieczności dodatkowych modyfikacji żadnego programu. Już podczas następnego odwołania programu SZBD do zmienionego katalogu zostałaby odczytana i wykorzystana nowa struktura rekordów pliku STUDENT.

W niektórych typach systemów baz danych, np. w systemach obiektowych i obiektowo-relacyjnych (patrz rozdział 12.), to użytkownicy mogą (w ramach definiowania baz danych) określać możliwe operacje na danych. Taka **operacja** (nazywana także *funkcją* lub *metodą*) składa się z dwóch elementów. Pierwszym z nich jest *interfejs* (nazywany często *sygnaturą*), który zawiera nazwę operacji oraz typy danych jej argumentów (nazywanych także parametrami). Drugim jest *implementacja* (*metoda*), która jest definiowana osobno i może być zmieniana bez konieczności modyfikowania odpowiadającego jej interfejsu. Aplikacje użytkownika mogą operować na danych przez wywoływanie tak zdefiniowanych operacji poprzez ich nazwy i argumenty, niezależnie od tego, w jaki sposób te operacje zostały zaimplementowane. Można taki model nazwać **niezależnością programu od operacji**.

Właściwość, która umożliwia zapewnienie niezależności programu od danych oraz niezależności programu od operacji, jest nazywana **abstrakcją danych**. Systemy zarządzania bazami danych udostępniają użytkownikom **konceptyjną reprezentację** danych, która nie zawiera wielu szczegółów związanych z wykorzystywanymi technikami przechowywania danych oraz implementacji operacji. Nieformalnie można przyjąć, że jednym z typów abstrakcji danych jest **model danych**, stosowany do zapewniania właśnie reprezentacji konceptyjnej. Model danych wykorzystuje takie logiczne pojęcia jak obiekty, ich własności oraz ich wewnętrzne relacje, które dla wielu użytkowników są bardziej zrozumiałe od pojęć związanych z przechowywaniem danych na współczesnych komputerach. Oznacza to, że model danych *ukrywa* szczegóły przechowywania i implementacji danych, które nie są przedmiotem zainteresowania dla większości użytkowników baz danych.

Przykładowo, przeanalizujmy ponownie strukturę bazy danych przedstawioną na rysunkach 1.2 i 1.3. Wewnętrzna implementacja pliku może być zdefiniowana za pomocą długości jego rekordów — liczby znaków (bajtów) w każdym rekordzie — natomiast każdy z wykorzystywanych elementów danych może zostać określony za pomocą jego bajtu początkowego wewnątrz odpowiedniego rekordu oraz długości (także wyrażonej w bajtach). Rekord z pliku STUDENT byłby w takim przypadku reprezentowany w taki sam sposób, jak to przedstawiono na rysunku 1.4. Typowego użytkownika bazy danych nie interesują jednak ani dokładne miejsce przechowywania poszczególnych elementów danych wewnątrz rekordu, ani jego rzeczywista długość; zamiast tego, taki użytkownik oczekuje



przede wszystkim tego, aby po odwołaniu się do elementu *Nazwisko* wybranego rekordu w pliku *STUDENT* została zwrócona prawidłowa wartość. Konceptyjną reprezentację rekordów w pliku *STUDENT* przedstawiono na rysunku 1.2. Wiele innych szczegółów organizacji plików z danymi (w tym zdefiniowanych dla niego ścieżek dostępu) można ukryć przed użytkownikami baz danych dzięki odpowiednim mechanizmom systemu zarządzania bazami danych (szczegóły związane z przechowywaniem danych omówimy w rozdziałach 16. i 17.).

W typowym rozwiązaniu opartym na bazie danych szczegółowa struktura i organizacja poszczególnych plików jest przechowywana w katalogu. Użytkownicy bazy danych i obsługujących ją aplikacje odwołują się jedynie do konceptyjnej reprezentacji tych plików, a za wydobywanie z katalogu szczegółów związanych z przechowywaniem plików odpowiadają stosowne moduły dostępu używanych systemów zarządzania bazami danych. Do zapewniania abstrakcji danych użytkownikom baz danych można wykorzystywać rozmaite modele danych. Znaczna część tej książki jest poświęcona prezentacji różnych modeli danych i stosowanym w nich mechanizmom abstrahowania reprezentacji danych.

W obiektowych i obiektowo-relacyjnych bazach danych proces abstrahowania obejmuje nie tylko struktury danych, ale także zdefiniowane dla nich operacje. Takie operacje stanowią abstrakcję dla czynności w reprezentowanym przez bazę danych mini-świecie, które zwykle są łatwiejsze do zrozumienia dla zwykłych użytkowników. Przykładowo, do obliczania średniej ocen wskazanego obiektu z pliku *STUDENT* może służyć operacja *OBLICZ\_ŚROC*. Operacje takie jak ta mogą być wywoływane przez użytkownika za pomocą odpowiednich zapytań (lub za pośrednictwem aplikacji obsługujących bazę danych) bez znajomości szczegółów dotyczących sposobu zaimplementowania wykonywanych działań.

### 1.3.3. Obsługa wielu perspektyw dla tych samych danych

Typowa baza danych ma wielu użytkowników, z których każdy może potrzebować dostępu do innej **perspektywy** (ang. *view*) dla tej bazy danych. Perspektywa może być albo podzbiorem bazy danych, albo może zawierać **dane wirtualne**, które zostały wywiedzione z plików bazy danych (dane wirtualne same w sobie nie są więc trwale składowane w bazie danych). Niektórzy użytkownicy w ogóle nie muszą wiedzieć, czy dane, do których się odwołują, faktycznie są przechowywane w takiej postaci w bazie danych, czy są z niej jedynie w odpowiedni sposób wywiedzione. Wielodostępny system zarządzania bazą danych, który jest wykorzystywany przez użytkowników do odmiennych celów, musi zapewniać mechanizmy ułatwiające definiowanie wielu różnych perspektyw. Przykładowo, jeden z użytkowników bazy danych przedstawionej na rysunku 1.2 może być zainteresowany wyłącznie dostępem z możliwością drukowania do listy studentów wraz z opisującymi ich informacjami; perspektywę przygotowaną dla takiego użytkownika przedstawiono na rysunku 1.5(a). Drugi użytkownik, którego interesuje wyłącznie sprawdzanie, czy poszczególni studenci zaliczyli wszystkie przedmioty wymagane do ich udziału w zajęciach, na które się zapisali, może potrzebować perspektywy przedstawionej na rysunku 1.5(b).

## LISTA

NazwiskoStudenta	Lista studentów				
	NumerPrzedmiotu	Ocena	Semestr	Rok	IdentyfikatorDziału
Nowak	INF1310	3	Jesienny	08	119
	MAT2410	4	Jesienny	08	112
Kowalski	MAT2410	5	Jesienny	07	85
	INF1310	5	Jesienny	07	92
	INF3320	4	Wiosenny	08	102
	INF3380	5	Jesienny	08	135

(a)

## WYMAGANIA WSTĘPNE

NazwaPrzedmiotu	NumerPrzedmiotu	WymaganiaWstępne
Bazy danych	INF3380	INF3320
		MAT2410
(b) Struktury danych	INF3320	INF1310

(b)

RYSUNEK 1.5. Dwie perspektywy dla bazy danych z rysunku 1.2. (a) Perspektywa REJESTR\_OCEN\_STUDENT. (b) Perspektywa PRZEDMIOT\_WYMAGANIA\_WSTĘPNE

### 1.3.4. Współdzielenie danych oraz wielodostępne przetwarzanie transakcji

Wielodostępny system zarządzania bazą danych, jak sama nazwa wskazuje, musi umożliwiać wielu użytkownikom jednoczesny dostęp do bazy danych. Ta właściwość ma zasadnicze znaczenie, jeśli z pojedynczą bazą danych chcemy zintegrować wiele aplikacji. System zarządzania bazą danych musi wówczas zawierać oprogramowanie **sterowania współbieżnego** (ang. *concurrency control*), które zapewni — w sposób kontrolowany — wielu użytkownikom możliwość podejmowania prób aktualizacji tych samych danych i zagwarantuje, że efekt tych działań będzie poprawny. Przykładowo, kiedy wielu pracowników linii lotniczych spróbuje jednocześnie zarezerwować miejsce w samolocie, system zarządzania bazą danych powinien zagwarantować dostępność każdego miejsca tylko dla jednego pasażera obsługiwanego przez jednego pracownika. Tego typu zastosowania są określane ogólnym terminem rozwiązań z **przetwarzaniem transakcji na bieżąco** (ang. *Online Transaction Processing — OLTP*). Zasadniczą rolą oprogramowania wielodostępnego systemu zarządzania bazą danych jest wówczas zapewnienie właściwego przetwarzania współbieżnych transakcji.

Pojęcie **transakcji** stało się centralnym elementem w wielu współczesnych zastosowaniach baz danych. Transakcja jest *wykonywanym programem* lub *procesem*, na który składa się jeden lub więcej operacji dostępu do bazy danych, takich jak odczytywanie czy aktualizacja jej rekordów. Każda transakcja, która zostanie zrealizowana w całości, ma w założeniu wykonywać logicznie poprawne operacje dostępu do bazy danych i nie może wpływać na przebieg pozostałych transakcji. System zarządzania bazą danych musi wymuszać wiele właściwości transakcji. Własność **izolacji** daje nam gwarancję, że każda transakcja będzie wykonywana w warunkach separacji od pozostałych transakcji, nawet

jeśli jednocześnie będą wykonywane setki transakcji. Własność **niepodzielności** gwarantuje, że albo zostaną wykonane wszystkie operacje na bazie danych składające się na daną transakcję, albo nie zostanie wykonana żadna z tych operacji. Szczegółowe omówienie transakcji można znaleźć w części 9. tej książki.

Wymienione w tym podrozdziale właściwości są głównymi elementami odróżniającymi systemy zarządzania bazami danych od tradycyjnego oprogramowania przetwarzającego pliki. W podrozdziale 1.6 omówimy dodatkowe cechy charakterystyczne dla systemów zarządzania bazami danych; najpierw jednak spróbujemy skategoryzować różne typy osób pracujących w środowisku systemów baz danych.

## 1.4. Aktorzy na scenie

W przypadku niedużej, osobistej bazy danych (np. wspomianej w podrozdziale 1.1 listy adresów) zwykle jedna osoba definiuje, konstruuje i operuje na bazie danych, zatem nie ma potrzeby zapewniania mechanizmów jej współdzielenia pomiędzy wielu użytkowników. Jednak w dużych organizacjach wiele osób jest zaangażowanych w projektowanie, wykorzystywanie i konserwację ogromnych baz danych wykorzystywanych przez setki lub tysiące użytkowników. W tym podrozdziale spróbujemy zidentyfikować osoby, których praca wiąże się z codziennym wykorzystywaniem ogromnych baz danych — nazwiemy ich *aktorami na scenie*. W podrozdziale 1.5 przeanalizujemy role osób, które można nazwać *pracownikami poza sceną* — czyli osób, które co prawda pracują nad utrzymaniem środowiska systemu bazy danych, ale nie są aktywnie zainteresowane samą zawartością tej bazy danych.

### 1.4.1. Administratorzy bazy danych

W każdej organizacji, gdzie wiele osób wykorzystuje te same zasoby, istnieje konieczność zatrudnienia głównego administratora, który będzie te zasoby nadzorował i nimi zarządzał. W środowisku bazy danych głównym zasobem jest oczywiście sama baza danych, drugim zasobem jest natomiast system zarządzania bazą danych lub oprogramowanie pokrewne. Obowiązki związane z administrowaniem tymi zasobami spadają na **administratora bazy danych** (ang. *database administrator — DBA*). Osoba na tym stanowisku odpowiada za uwierzytelnianie dostępu do bazy danych, koordynowanie i monitorowanie jej wykorzystania oraz za pozyskiwanie niezbędnych zasobów programowych i sprzętowych. Administrator bazy danych jest także odpowiedzialny za wszystkie problemy, w tym naruszenia bezpieczeństwa lub zbyt długi czas odpowiedzi. W wielkich organizacjach administratorzy baz danych dysponują zwykle całym sztabem ludzi, którzy pomagają im w realizacji wymienionych zadań.

### 1.4.2. Projektanci bazy danych

**Projektanci bazy danych** odpowiadają za identyfikację danych, które docelowo mają być przechowywane w bazie danych, oraz za wybór właściwych struktur, które będą te dane reprezentowały i przechowywały. Większość tych zadań musi być wykonana zanim jeszcze baza danych zostanie faktycznie zaimplementowana i wypełniona danymi. Projektanci baz

danych odpowiadają za zapewnienie odpowiedniej komunikacji ze wszystkimi potencjalnymi użytkownikami projektowanych rozwiązań, aby dobrze zrozumieć definiowane przez nich wymagania — tylko w ten sposób mogą zapewnić zgodność efektów swojej pracy z tymi wymaganiami. W wielu przypadkach projektanci należą do zespołu kierowanego przez administratora bazy danych i mogą mieć przydzielane zupełnie inne obszary odpowiedzialności już po zakończeniu procesu projektowania bazy danych. Projektanci bazy danych zazwyczaj uzgadniają oczekiwany kształt opracowywanego rozwiązania z każdą grupą potencjalnych użytkowników i tworzą takie **perspektywy** bazy danych, które w jak największym stopniu odpowiadają zdefiniowanym przez te grupy wymaganiom dotyczącym dostępnych danych i funkcji przetwarzających. Każda perspektywa jest następnie analizowana i *integrowana* z perspektywami opracowanymi dla pozostałych grup przyszłych użytkowników. Ostateczna wersja projektu bazy danych musi być zgodna z wymaganiami zgłoszonymi przez wszystkie takie grupy.

### 1.4.3. Użytkownicy końcowi

**Użytkownicy końcowi** to ci pracownicy organizacji, których codzienna praca wymaga dostępu do takich operacji na bazie danych jak przetwarzanie zapytań, aktualizowanie informacji oraz generowanie raportów — baza danych jest więc tworzona przede wszystkim dla nich. Istnieje kilka kategorii, do których można przypisać użytkowników końcowych:

- **Użytkownicy dorywczy** korzystają z bazy danych tylko od czasu do czasu, ale mogą za każdym razem potrzebować innych informacji. Tacy użytkownicy końcowi często używają do definiowania swoich żądań wyszukanych języków zapytań do bazy danych i zwykle pełnią stanowiska menadżerów średniego lub wysokiego szczebla.
- **Naiwni** lub **parametryczni użytkownicy końcowi** stanowią znaczącą część wszystkich użytkowników bazy danych. Ich praca polega przeważnie na wielokrotnym wykonywaniu zapytań i aktualizowaniu informacji zawartych w bazie danych. Użytkownicy należący do tej grupy wykorzystują zwykle standardowe typy zapytań i operacji aktualizacji, które zostały bardzo uważnie zaprogramowane i przetestowane (są to tzw. **transakcje zapuszkowane** — ang. *canned transactions*). Obecnie wiele takich zadań może być wykonywanych za pomocą **aplikacji mobilnych** stosowanych w urządzeniach mobilnych. Zadania realizowane przez tego typu użytkowników mogą się od siebie bardzo różnić:
  - Kasjerzy bankowi mogą sprawdzać salda kont i wykonywać operacje wpłat oraz wypłat.
  - Pracownicy biur rezerwacji linii lotniczych, hoteli oraz firm wynajmujących samochody sprawdzają dostępność żadanego zasobu i dokonują rezerwacji.
  - Pracownicy magazynu firmy kurierskiej rejestrują identyfikatory i opisy przyjętych paczek za pomocą kodów kreskowych, aktualizując tym samym centralną bazę danych otrzymanych i transportowanych przesyłek.
  - Użytkownicy mediów społecznościowych zamieszczają posty i czytają informacje w witrynach serwisów społecznościowych.

- Do grupy **doświadczonych użytkowników końcowych** należą najczęściej inżynierowie, naukowcy, analitycy biznesowi i wszyscy inni pracownicy organizacji, którzy mieli już do czynienia z mechanizmami systemów zarządzania bazami danych oraz stosowali już rozmaite rozwiązania odpowiadające ich skomplikowanym wymaganiom.
- **Samodzielni użytkownicy** utrzymują zwykle osobiste bazy danych w oparciu o gotowe pakiety oprogramowania, które oferują łatwe w użyciu interfejsy, często oparte na wygodnych menu lub na komponentach graficznych. Przykładem może być użytkownik pakietu ułatwiającego zarządzanie podatkami, który przechowuje w swoim oprogramowaniu osobiste dane finansowe.

Typowe systemy zarządzania bazami danych oferują wiele mechanizmów ułatwiających dostęp do obsługiwanych baz danych. Niedoświadczeni użytkownicy końcowi nie muszą wówczas poświęcać zbyt dużo czasu na naukę obsługi udostępnianej w ten sposób funkcjonalności — muszą jedynie zrozumieć, jak należy korzystać z interfejsu użytkownika aplikacji mobilnych lub standardowych transakcji, które zostały zaprojektowane i zaimplementowane specjalnie z myślą o nich. Użytkownicy korzystający z bazy danych doręwczo muszą się nauczyć tylko kilku funkcji, które będą być może wielokrotnie stosowali w swojej pracy. Doświadczeni użytkownicy zapewne spróbują poznać większość mechanizmów wykorzystywanego systemu zarządzania bazą danych, aby móc w przyszłości realizować funkcje zdefiniowane w ich zawiłych wymaganiach. Samodzielni użytkownicy zwykle nabierają ogromnej wprawy w wykorzystywaniu konkretnych pakietów oprogramowania.

#### 1.4.4. Analitycy systemowi i programiści aplikacji (inżynierowie oprogramowania)

**Analitycy systemowi** określają wymagania użytkowników końcowych, szczególnie tych należących do grupy naiwnych lub parametrycznych, i opracowują specyfikacje wielokrotnie wykonywanych transakcji, które powinny w jak największym stopniu odpowiadać tym wymaganiom. **Programiści aplikacji** w pierwszej kolejności implementują w postaci programów specyfikacje otrzymane od analityków systemowych, po czym poddają gotowe rozwiązania testom, debugują je, przygotowują niezbędną dokumentację i konserwują przygotowane w ten sposób transakcje. Analitycy systemowi i programiści aplikacji (nazywani często **inżynierami oprogramowania**) powinni mieć odpowiednią wiedzę na temat wszystkich tych możliwości oferowanych przez wykorzystywany system zarządzania bazą danych, które mogą się przydać podczas realizacji tego zadania.

### 1.5. Pracownicy poza sceną

Poza osobami, które projektują bazy danych, wykorzystują je i administrują nimi, działania związane z projektowaniem, tworzeniem i operowaniem na *oprogramowaniu i środowisku systemowym SZBD* wymagają udziału także innych osób. Pracownicy należący do tej grupy zwykle nie są zainteresowani samą bazą danych ani zawartymi w niej informacjami — nazywamy ich *pracownikami poza sceną*. Takie osoby można podzielić na następujące grupy:

- **Projektanci i programiści systemu zarządzania bazą danych** są autorami projektów i (mających postać pakietów oprogramowania) implementacji modułów i interfejsów systemu zarządzania bazą danych. Systemy zarządzania bazami danych są bardzo skomplikowanymi pakietami oprogramowania, które składają się z wielu **modułów** (komponentów), włącznie z modułami implementującymi katalog, język przetwarzania zapytań, interfejs przetwarzania, mechanizmy dostępu i buforowania, sterowanie współbieżne oraz obsługę odzyskiwania danych i bezpieczeństwa. Każdy system zarządzania bazą danych musi oferować interfejsy dla innych systemów oprogramowania, takich jak systemy operacyjne czy kompilatory różnych języków programowania.
- Do grupy **programistów narzędzi** należą wszystkie osoby projektujące i implementujące **narzędzia**, czyli pakiety oprogramowania, które nie tylko ułatwiają projektowanie i wykorzystywanie systemu bazy danych, ale także pomagają poprawić jego wydajność. Narzędzia są opcjonalnymi pakietami, które w wielu przypadkach można dokupić osobno do już posiadanego systemu zarządzania bazą danych. Takie pakiety mogą służyć projektowaniu baz danych, monitorowaniu wydajności, obsłudze interfejsów języka naturalnego lub interfejsów graficznych, modelowaniu, przeprowadzaniu symulacji oraz testowemu generowaniu danych. W wielu przypadkach narzędzia tego typu są opracowywane i wprowadzane na rynek przez niezależnych producentów.
- Kategoria **operatorów i personelu pomocniczego** obejmuje administratorów systemów, którzy odpowiadają za bieżące funkcjonowanie i konserwację środowiska sprzętowego i programowego dla wykorzystywanego systemu bazy danych.

Wymienione powyżej kategorie pracowników poza sceną są co prawda bardzo pomocne podczas udostępniania systemów baz danych użytkownikom końcowym, jednak pracownicy zaliczani do tej grupy raczej nie wykorzystują tych samych baz danych do własnych celów.

## 1.6. Zalety stosowania rozwiązań opartych na systemach zarządzania bazami danych

W tym podrozdziale omówimy niektóre z zalet stosowania systemów zarządzania bazami danych oraz cechy, które powinien posiadać dobry system tego typu. Prezentowane tutaj możliwości systemów zarządzania bazami danych należy traktować jako elementy uzupełniające cztery podstawowe właściwości omówione w podrozdziale 1.3. Administratorzy baz danych wykorzystują taką dodatkową funkcjonalność do osiągania rozmaitych celów związanych z projektowaniem, administrowaniem i wykorzystywaniem ogromnych, wielodostępnych baz danych.

### 1.6.1. Kontrola nadmiarowości

W tradycyjnych pakietach oprogramowania tworzonych z myślą o przetwarzaniu plików z danymi, każda grupa użytkowników utrzymuje własne pliki obsługiwane przez aplikacje operujące na zawartych w nich danych. Przykładowo, przeanalizujemy raz jeszcze przykład



bazy danych UNIWERSYTET z rysunku 1.2; przyjmijmy, że dwie grupy użytkowników mogą stanowić personel dziekanatu oraz działu księgowości. W podejściu tradycyjnym każda z tych grup niezależnie od siebie utrzymywałaby pliki reprezentujące informacje o studentach. Dział księgowości obsługiwałby dane o rejestracji i płatnościach; natomiast dziekanat śledziłby przedmioty, w których poszczególni studenci uczestniczą, oraz dane o uzyskanych ocenach. Kolejne grupy użytkowników mogłyby dodatkowo powielać niektóre lub wszystkie te dane we własnych plikach.

Taka **nadmiarowość** (redundancja), polegająca na wielokrotnym przechowywaniu tych samych danych, prowadzi do wielu niepotrzebnych problemów. Po pierwsze, każda logiczna aktualizacja reprezentowanych w ten sposób informacji (jak choćby wpisanie danych nowego studenta) musi być przeprowadzana wielokrotnie — przynajmniej raz w każdym pliku, w którym rejestrowane są dane studentów. To zaś prowadzi do *zwielokrotnienia wysiłku*. Po drugie, w sytuacji, gdy te same dane są przechowywane w wielu miejscach, mamy do czynienia z *marnotrawstwem przestrzeni przechowywania*. Po trzecie, pliki reprezentujące te same dane mogą się stać *niespójne*. Może to być efekt np. zastosowania operacji aktualizacji tylko dla niektórych danych, z pominięciem pozostałych. Okazuje się, że nawet jeśli aktualizacja (np. wspomniana operacja dodania nowego studenta) zostanie zastosowana dla wszystkich wymagających tego plików, dane dotyczące danego studenta nadal mogą być narażone na *niespójność* spowodowaną niezależnymi aktualizacjami przeprowadzanymi przez poszczególne grupy użytkowników. Przykładowo, jedna z grup użytkowników może błędnie podać datę urodzenia studenta ('19-STY-1988'), natomiast pozostałe grupy mogą wpisać poprawną datę ('19-STY-1988').

W technice opartej na bazie danych perspektywy dla różnych grup użytkowników bazy danych są tworzone i integrowane już w fazie jej projektowania. Idealnym rozwiązaniem jest oczywiście posiadanie jednego projektu bazy danych, który uwzględni wszystkie logiczne elementy danych (np. nazwisko i datę urodzenia studenta) umieszczone w *jednym miejscu*. Na tym polega **normalizacja danych**, zapewniająca spójność danych oraz oszczędność przestrzeni przechowywania (normalizacja danych jest opisana w części 6.).

W praktyce jednak często konieczne jest zastosowanie techniki **kontrolowanej nadmiarowości**, która poprawi wydajność przetwarzania zapytań. Przykładowo, możemy nadmiarowo przechowywać elementy danych *NazwiskoStudenta* i *NumerPrzedmiotu* w pliku RAPORT\_OCEN (patrz rysunek 1.5(a)), ponieważ za każdym razem, gdy wskutek wykonania zapytania uzyskamy rekord z tego pliku, będziemy chcieli otrzymać jednocześnie nazwisko studenta i numer (identyfikator) przedmiotu wraz z wystawioną oceną, numerem studenta oraz identyfikatorem kursu. Jeśli umieścimy wszystkie te dane w jednym miejscu, skompletowanie potrzebnych informacji nie będzie wymagało przeszukiwania wielu plików. Na tym polega **denormalizacja**. W takich przypadkach system zarządzania bazą danych powinien jednak zapewniać możliwość *kontrolowania* nadmiarowości, aby zapobiec ewentualnym niespójnościom pomiędzy różnymi plikami. Taki mechanizm może to robić automatycznie przez sprawdzanie, czy pary wartości *NazwiskoStudenta* → *NumerIndeksu* we wszystkich rekordach pliku RAPORT\_OCEN (patrz rysunek 1.6(a)) odpowiadają parom wartości *Nazwisko-NumerIndeksu* w odpowiednich rekordach pliku STUDENT (patrz rysunek 1.2). Podobnie, pary wartości *IdentyfikatorKursu-NumerPrzedmiotu* w poszczególnych rekordach pliku RAPORT\_OCEN muszą odpowiadać odpowiednim parom wartości w rekordach pliku KURS. Takie testy można zdefiniować w systemie zarządzania bazą



danych już podczas projektowania bazy danych — SZBD będzie wówczas wymuszał odpowiednie działania kontrolne za każdym razem, gdy zostanie podjęta próba zaktualizowania pliku RAPORT\_OCEN. Na rysunku 1.6(b) przedstawiono rekord pliku RAPORT\_OCEN, który jest niespójny z przedstawioną na rysunku 1.2 zawartością pliku STUDENT — jest to więc przykład sytuacji, w której *nie zastosowano kontroli nadmiarowości* i zapisano w pliku RAPORT\_OCEN błędne dane.

GRADE\_REPORT

NumerIndeksu	NazwiskoStudenta	IdentyfikatorDziału	NumerPrzedmiotu	Ocena
17	Nowak	112	MAT2410	4
17	Nowak	119	INF1310	3
8	Kowalski	85	MAT2410	5
8	Kowalski	92	INF1310	5
8	Kowalski	102	INF3320	4
8	Kowalski	135	INF3380	5

(a)

GRADE\_REPORT

NumerIndeksu	NazwiskoStudenta	IdentyfikatorDziału	NumerPrzedmiotu	Ocena
17	Kowalski	112	MAT2410	4

(b)

RYSUNEK 1.6. Nadmiarowe przechowywanie elementów danych NazwiskoStudenta i NumerPrzedmiotu w pliku RAPORT\_OCEN. (a) Spójne dane. (b) Niespójny rekord

## 1.6.2. Ograniczanie możliwości uzyskania nieautoryzowanego dostępu

Kiedy jedną wielką bazę danych współdzielili wielu użytkowników, jest mało prawdopodobne, by większość użytkowników mogła uzyskać autoryzowany dostęp do wszystkich informacji przechowywanych w tej bazie danych. Przykładowo, dane finansowe są często uznawane za poufne i powinny w związku z tym być dostępne tylko dla niektórych użytkowników. Niektórzy użytkownicy mogą dodatkowo uzyskać prawo wyłącznie do odczytywania pewnych danych, inni natomiast mogą dysponować zarówno uprawnieniami odczytu, jak i możliwością aktualizacji. Oznacza to, że typ operacji dostępu (odczytu lub aktualizacji) także musi być odpowiednio kontrolowany. Użytkownicy lub grupy użytkowników mają zwykle przydzielane numery (identyfikatory) kont zabezpieczonych hasłem — dostęp do bazy danych jest wówczas możliwy wyłącznie po podaniu tych informacji uwierzytelniających. System zarządzania bazą danych powinien oferować **podsystem bezpieczeństwa i autoryzacji**, za pomocą którego administrator bazy danych będzie mógł tworzyć konta użytkowników (lub grup użytkowników) i nakładać na nie odpowiednie ograniczenia. System zarządzania bazą danych powinien następnie automatycznie wymuszać przestrzeganie tych ograniczeń. Warto pamiętać, że podobne elementy zabezpieczające można spotkać w samym oprogramowaniu systemu zarządzania bazą danych. Przykładowo, tylko osoby z uprawnieniami administratora bazy danych mogą uruchamiać pewne **uprzywilejowane elementy tego oprogramowania**, np. moduły przeznaczone do tworzenia nowych kont użytkowników. Podobnie, użytkownicy parametryczni mogą mieć możliwość uzyskiwania dostępu do bazy danych wyłącznie za pośrednictwem transakcji

zaprojektowanych specjalnie z myślą o realizowanych przez nich zadaniach. Bezpieczeństwo i autoryzację opisujemy w rozdziale 30.

### 1.6.3. Zapewnianie miejsca trwałego przechowywania dla obiektów stosowanych w programach

Bazy danych mogą być wykorzystywane do zapewniania **miejsca trwałego przechowywania** dla obiektów i struktur danych stosowanych w programach. Jest to jedna z przyczyn tworzenia **obiektowych systemów baz danych** (patrz rozdział 12.). Języki programowania wykorzystują zwykle skomplikowane struktury danych, jak struktury czy definicje klas w językach C++ i Java. Wartości reprezentowane przez zmienne wykorzystywane w tak napisanych programach są usuwane z chwilą zakończenia ich pracy, chyba że programista napisze kod, który wprost będzie je umieszczał w trwałych plikach dyskowych (co zwykle wiąże się z koniecznością konwertowania tych skomplikowanych struktur do odpowiedniego formatu). Kiedy zaistnieje potrzeba ponownego odczytania danych, programista musi je przekonwertować z formatu pliku do zmiennej lub struktury obiektowej, która może być przetwarzana w programie. Obiektywne systemy baz danych są zgodne z takimi językami programowania jak C++ czy Java, a oprogramowanie systemów zarządzania bazami danych może automatycznie wykonywać niezbędne konwersje. Oznacza to, że skomplikowany obiekt w C++ może być trwale przechowywany w obiektowym systemie zarządzania bazą danych. O takiej strukturze mówimy, że jest obiektem **trwałym**, ponieważ zakończenie pracy jego programu nie powoduje jego usunięcia (taka utrwalona struktura może zostać ponownie odczytana przez inny program).

Trwałe przechowywanie obiektów i struktur danych stosowanych w programach jest ważnym elementem funkcjonalności systemów baz danych. Tradycyjne systemy tego typu często były narażone na **problem niezgodności impedancji**, ponieważ struktury danych oferowane przez systemy zarządzania bazami danych były niezgodne ze strukturami danych wykorzystywanymi w językach programowania. Obiektywne systemy baz danych oferują zwykle **zgodność** obsługiwanych struktur danych ze strukturami stosowanymi w jednym lub większej ilości obiektowych języków programowania.

### 1.6.4. Zapewnianie struktur przechowywania dla efektywnego przetwarzania zapytań

Systemy baz danych muszą zapewniać możliwość *efektywnego wykonywania zapytań i operacji aktualizacji danych*. Ponieważ baza danych jest zwykle przechowywana na dysku, system zarządzania bazą danych musi udostępniać wyspecjalizowane struktury danych i techniki wyszukiwania, które przyspieszą proces przeglądania dysku w poszukiwaniu żądanych rekordów. Do tego celu wykorzystywane są pliki zewnętrzne nazywane **indeksami**. Indeksy opierają się zwykle na drzewiastych strukturach danych lub strukturach mieszających odpowiednio przystosowanych do przeszukiwania danych dyskowych. Aby przetworzyć rekordy bazy danych żądane przez konkretne zapytanie, należy je najpierw skopiować z dysku do pamięci operacyjnej. Oznacza to, że systemy zarządzania bazami danych często zawierają specjalne moduły **buforujące**, które utrzymują fragmenty baz danych w wyasygnowanych do tego celu buforach w pamięci operacyjnej. Zwykle za buforowanie danych dysko-

wych w pamięci odpowiada system operacyjny. Jednak ponieważ buforowanie danych ma bardzo istotny wpływ na wydajność systemów SZBD, większość takich systemów stosuje własne mechanizmy buforowania.

Należący do systemu zarządzania bazą danych moduł **przetwarzania zapytań i optymalizacji** odpowiada za wybór (w oparciu o istniejące struktury przechowywania danych) efektywnego planu wykonania dla każdego zapytania. Wybór rodzaju tworzonych i utrzymywanych indeksów jest częścią procesów *fizycznego projektowania i strojenia bazy danych*, za które zawsze odpowiedzialny jest zespół administratorów bazy danych. Przetwarzanie i optymalizowanie zapytań jest opisane w części 8. tej książki.

### 1.6.5. Zapewnianie możliwości tworzenia kopii bezpieczeństwa i odzyskiwania danych

System zarządzania bazą danych musi także zapewniać mechanizmy ułatwiające przywracanie pierwotnego stanu po awariach sprzętowych lub programowych. Służy do tego należący do systemu zarządzania bazą danych specjalny **podsystem tworzenia kopii bezpieczeństwa i odzyskiwania danych**. Przykładowo, jeśli wykorzystywany system komputerowy ulegnie awarii w czasie wykonywania skomplikowanej transakcji aktualizującej dane, podsystem odzyskiwania musi się upewnić, że baza danych zostanie przywrócona do stanu sprzed momentu, w którym rozpoczęło się wykonywanie przerwanej transakcji. Kopie bezpieczeństwa są też niezbędne po wystąpieniu nieodwracalnej awarii dysku. Odzyskiwanie danych i tworzenie kopii bezpieczeństwa jest opisane w rozdziale 22.

### 1.6.6. Zapewnianie interfejsów dla wielu użytkowników

Ponieważ bazy danych mogą być wykorzystywane przez wiele typów użytkowników, którzy dysponują wiedzą techniczną na bardzo zróżnicowanych poziomach, systemy zarządzania bazami danych powinny udostępniać różne interfejsy użytkownika. Takimi interfejsami są aplikacje mobilne, język zapytań wykorzystywany przez użytkowników doraźnych, interfejs języka programowania dla programistów aplikacji, formularze i kody poleceń dla użytkowników parametrycznych oraz interfejsy oparte na menu i języku naturalnym dla samodzielnych użytkowników. Zarówno interfejsy oparte na formularzach, jak i te oparte na menu są często nazywane **graficznymi interfejsami użytkownika** (ang. *Graphical User Interface* — *GUI*). Istnieje wiele wyspecjalizowanych języków i środowisk do definiowania takich graficznych interfejsów użytkownika. Dosyć popularne są także narzędzia tworzące interfejsy GUI dla baz danych oparte na stronach WWW.

### 1.6.7. Reprezentowanie skomplikowanych relacji pomiędzy danymi

Baza danych może zawierać wiele zróżnicowanych danych, które są wzajemnie powiązane na różne sposoby. Przykładowo, przeanalizujmy raz jeszcze bazę danych przedstawioną na rysunku 1.2. Rekord dla studenta o nazwisku *Kowalski* w pliku *STUDENT* jest związany z czterema rekordami w pliku *RAPORT\_OCEN*. Podobnie, każdy rekord reprezentujący pojedynczy kurs jest związany nie tylko z pojedynczym rekordem reprezentującym przedmiot, ale tak-

że z kilkoma rekordami z pliku `RAPORT_OCEN` — po jednym dla każdego studenta, który zaliczył dany kurs. System zarządzania bazą danych musi oferować możliwość nie tylko samego reprezentowania wraz z danymi rozmaitych skomplikowanych związków, ale także łatwego i efektywnego generowania i aktualizowania powiązanych ze sobą informacji.

## 1.6.8. Wymuszanie więzów integralności

Większość baz danych ma zdefiniowane pewne **więzy integralności**, które muszą być spełnione przez przechowywane dane. System zarządzania bazą danych powinien oferować możliwość zarówno definiowania takich ograniczeń, jak i wymuszania ich przestrzegania. Najprostszym typem więzów integralności jest wymuszenie określonego typu danych dla poszczególnych elementów danych. Przykładowo, w zaprezentowanej na rysunku 1.3 bazie danych możemy określić, że element danych *RokSt* w każdym z rekordów pliku `STUDENT` musi być liczbą całkowitą z przedziału od 1 do 5, natomiast wartości elementu danych *Nazwisko* w rekordach tego samego pliku muszą być maksymalnie 30-znakowymi ciągami liter alfabetu. Bardziej skomplikowane, popularne typy więzów integralności polegają na określaniu wymaganych związków pomiędzy rekordami w jednym pliku a rekordami w innych plikach. Przykładowo, w bazie danych przedstawionej na rysunku 1.2 możemy określić, że *każdy rekord kursu musi być powiązany z odpowiednim rekordem przedmiotu*. Są to więzy **integralności odwołań**. Jeszcze inny rodzaj więzów integralności określa unikatowość wartości we wskazanym elemencie danych — takie ograniczenie może mieć postać: *każdy rekord przedmiotu musi mieć unikatową wartość elementu danych NumerPrzedmiotu*. Jest to ograniczenie **klucza** lub **unikatowości**. Zaprezentowane przed chwilą przykładowe ograniczenia wynikają z **semantyki** (znaczenia) danych oraz reprezentowanego za ich pomocą miniświata. Za identyfikację więzów integralności w fazie projektowania bazy danych zawsze odpowiada jej projektant. Niektóre ograniczenia mogą być definiowane w systemie zarządzania bazą danych i automatycznie przez ten system wymuszane; inne mogą wymagać przeprowadzania odpowiednich testów na poziomie programów aktualizujących dane lub już w momencie wpisywania informacji. W typowych dużych aplikacjach ograniczenia są powszechnie nazywane **regułami biznesowymi**.

Element danych może być błędnie wpisany i nadal spełniać warunki wynikające z określonych więzów integralności. Przykładowo, jeśli na skutek błędu osoby wpisującej dane, dla studenta, który uzyskał w rzeczywistości ocenę 5, zostanie zarejestrowana w bazie danych ocena 3, system zarządzania bazą danych *nie będzie mógł* automatycznie wykryć tego błędu, ponieważ ocena 3 jest poprawną wartością dla elementu danych *Ocena*. Podobne błędy wynikające z nieuwważnego wpisywania danych do systemu można wykrywać wyłącznie „ręcznie” (np. kiedy student poskarży się, że zarejestrowana ocena jest niezgodna z rzeczywistością) i poprawiać przez aktualizację bazy danych. Okazuje się jednak, że próba wpisania oceny 7 może być automatycznie odrzucana przez system zarządzania bazą danych, ponieważ 7 nie jest poprawną wartością dla elementu danych *Ocena*. W trakcie omawiania w dalszych rozdziałach modeli danych przedstawione zostaną dotyczące ich zasady. Na przykład w opisanym w rozdziale 3. modelu ER relacja musi obejmować przynajmniej dwie encje. Reguły dotyczące określonych modeli danych są nazywane ich **regułami naturalnymi**.

### 1.6.9. Zezwalanie na wnioskowanie i podejmowanie działań w oparciu o zdefiniowane reguły

Niektóre systemy baz danych oferują możliwość definiowania *reguł wnioskowania* (reguł dedukcyjnych), w oparciu o które można *wywodzić* nowe informacje na podstawie faktów reprezentowanych w bazie danych. Takie systemy są nazywane **dedukcyjnymi systemami baz danych**. Przykładowo, w mini-świecie mogą istnieć skomplikowane zasady określające, czy dany student odbywa staż. Można te zasady zdefiniować *deklaratywnie* w postaci **reguł**, które będą kompilowane i utrzymywane przez system zarządzania bazą danych, i które będą umożliwiały wyszukiwanie wszystkich studentów odbywających staż. W tradycyjnych systemach zarządzania bazami danych do obsługi tego typu zastosowań konieczny byłby odpowiedni *kod programu proceduralnego*. Jeśli jednak reguły obowiązujące w reprezentowanym mini-świecie ulegną zmianie, generalnie znacznie łatwiejszym zadaniem (od ponownego kodowania programów proceduralnych) jest zmiana zadeklarowanych wcześniej reguł wnioskowania. W stosowanych obecnie systemach relacyjnych baz danych z tabelami można powiązać **wyzwalacze**. Wyzwalacz to rodzaj reguły aktywowanej przez aktualizację tabeli, który powoduje wykonanie dodatkowych operacji na innych tabelach, przesłanie komunikatów itd. Bardziej złożone procedury wymuszania reguł to **procedury składowane**. Stanowią one część definicji bazy danych i są wywoływane po spełnieniu określonych warunków. Mechanizmy zapewniające jeszcze większe możliwości są oferowane przez **aktywne systemy baz danych**, gdzie w przypadku wystąpienia określonych zdarzeń lub spełnienia określonych warunków aktywne reguły mogą automatycznie inicjować zdefiniowane wcześniej działania. W rozdziale 26. znajdziesz wprowadzenie do aktywnych baz danych (podrozdział 26.1) i dedukcyjnych baz danych (podrozdział 26.5).

### 1.6.10. Dodatkowe własności wynikające ze stosowania rozwiązań opartych na bazach danych

W tym punkcie omówimy niektóre dodatkowe korzyści wynikające ze stosowania rozwiązań opartych na bazach danych i dotyczące większości organizacji.

**Możliwość wymuszania stosowania standardów.** Rozwiązania oparte na bazach danych umożliwiają administratorom baz danych wymuszanie na użytkownikach stosowania standardów przyjętych w danej organizacji. Standardy ułatwiają wzajemną komunikację i współpracę pomiędzy różnymi działami organizacji oraz pomiędzy użytkownikami przydzielonymi do różnych projektów. Standardy można definiować zarówno dla nazw i formatów elementów danych, jak i dla formatów wyświetlania, struktury raportów, terminologii itp. Administrator bazy danych może łatwiej wymuszać stosowanie standardów w środowisku scentralizowanej bazy danych niż w środowisku, w którym każda grupa użytkowników sama kontroluje swoje oprogramowanie i przetwarzane pliki.

**Skrócony czas wytwarzania aplikacji.** Szybkość wytwarzania nowych aplikacji (choćby takich, które pobierają z bazy danych pewne dane i drukują na ich podstawie nowy raport) jest zwykle wykorzystywana w roli kluczowego hasła marketingowego promującego wszelkie rozwiązania oparte na bazach danych. Projektowanie i implementowanie od zera nowych baz danych może w wielu przypadkach wymagać więcej czasu niż pisanie wyspecjalizowanych aplikacji operujących na plikach. Kiedy jednak taka baza danych będzie

już gotowa, tworzenie nowych aplikacji w oparciu o odpowiednie mechanizmy systemu zarządzania bazą danych wymaga generalnie minimalnych nakładów czasowych. Tworzenie aplikacji z wykorzystaniem tych mechanizmów wymaga w przybliżeniu od jednej szóstej do jednej czwartej czasu, który zużylibyśmy na opracowanie odpowiedniego programu pracującego w tradycyjnym systemie plików.

**Elastyczność.** Każda ewentualna zmiana wymagań może się wiązać z koniecznością zmodyfikowania struktury bazy danych. Przykładowo, nowa grupa użytkowników może zgłosić konieczność przetwarzania informacji, które nie są przechowywane w bazie danych w obecnym kształcie. W efekcie, może zaistnieć potrzeba dodania do bazy danych nowego pliku lub rozszerzenia elementów danych w pliku już istniejącym. Współczesne systemy zarządzania bazami danych oferują na szczęście pewne możliwości w zakresie wprowadzania ewolucyjnych zmian do struktury bazy danych bez konieczności modyfikowania już przechowywanych danych ani istniejących aplikacji.

**Dostępność aktualnych informacji.** System zarządzania bazą danych udostępnia tę bazę wszystkim użytkownikom. Oznacza to, że w momencie zastosowania w udostępnianej bazie danych operacji aktualizacji wykonanej przez jednego użytkownika, wszyscy pozostali użytkownicy mogą natychmiast zobaczyć efekt wykonania tej operacji. Dostępność aktualnych informacji ma zasadnicze znaczenie w wielu aplikacjach przetwarzających transakcje (takich jak systemy rezerwacji czy bankowe bazy danych) — jest to możliwe dzięki mechanizmom sterowania współbieżnego i podsystemowi odzyskiwania danych systemu zarządzania bazą danych.

**Korzyści ekonomiczne.** Rozwiązania oparte na systemach sterowania bazami danych umożliwiają konsolidowanie danych i aplikacji celem ograniczenia niepotrzebnego pokrywania się działań personelu przetwarzającego dane przydzielonego do różnych projektów lub pracującego w różnych działach organizacji. Dzięki temu cała organizacja może inwestować większe środki w nowoczesne procesory, układy pamięci czy sprzęt sieciowy, zamiast pozwalać poszczególnym działom na dokonywanie samodzielnych zakupów (często gorszego) wyposażenia. Można w ten sposób zredukować łączne koszty operacyjne i zarządzania.

## 1.7. Krótka historia praktycznych zastosowań baz danych

Przedstawimy teraz krótki przegląd historii zastosowań systemów zarządzania bazami danych oraz ich wpływu na dynamikę rozwoju nowych rodzajów systemów baz danych.

### 1.7.1. Wczesne zastosowania baz danych oparte na systemach hierarchicznych i sieciowych

Wiele wczesnych rozwiązań opartych na bazach danych polegało na przechowywaniu rekordów utrzymywanych przez ogromne organizacje, takie jak korporacje, uniwersytety, szpitale czy banki. W wielu z tych rozwiązań trzeba było reprezentować wielkie liczby rekor-



dów o podobnej strukturze. Przykładowo, w oprogramowaniu stosowanym przez uniwersytety podobne informacje były przechowywane dla każdego studenta, każdego przedmiotu, każdej oceny itp. Takie rozwiązania wiązały się z koniecznością utrzymywania wielu typów rekordów oraz obsługi wielu istniejących pomiędzy nimi relacji.

Jednym z głównych problemów występujących we wczesnych systemach baz danych było mieszanie relacji koncepcyjnych z fizycznym modelem przechowywania i rozmieszczania rekordów na dysku. Dlatego takie systemy nie zapewniały wystarczającej *abstrakcji danych* i *niezależności danych od programu*. Przykładowo, rekordy reprezentujące oceny uzyskane przez konkretnego studenta mogły być fizycznie przechowywane bezpośrednio obok odpowiedniego rekordu reprezentującego samego studenta. Takie rozwiązanie zapewniałoby co prawda bardzo efektywny dostęp do oryginalnych zapytań i transakcji, których obsługa stanowiła jeden z celów projektowania całej bazy danych, jednak nie gwarantowałoby wystarczającej elastyczności w zakresie efektywności dostępu do danych za pomocą nowych zapytań i transakcji. Szczególnie trudna byłaby efektywna implementacja nowych zapytań, które wymagałyby do sprawnego przetwarzania danych zupełnie innej organizacji fizycznego przechowywania informacji. Okazuje się, że dosyć trudne byłoby także zreorganizowanie bazy danych w sytuacji, gdyby pojawiły się jakieś zmiany w oryginalnych wymaganiach dotyczących danego rozwiązania.

Innym niedociągnięciem wczesnych systemów baz danych było udostępnianie przez nie wyłącznie interfejsów dla języków programowania. Takie rozwiązanie niepotrzebnie wydłużało czas i zwiększało koszty implementowania nowych zapytań i transakcji, ponieważ wymagało pisania, testowania i debugowania nowych programów. Większość tego typu systemów baz danych była implementowana na dużych i drogich komputerach centralnych (działo się tak począwszy od połowy lat 60-tych aż do końca lat 80-tych). Najbardziej popularne odmiany wczesnych systemów baz danych były konstruowane zgodnie z trzema paradygmatami — były to odpowiednio systemy hierarchiczne, systemy oparte na modelu sieciowym oraz odwrócone systemy plików.

### 1.7.2. Zapewnianie elastyczności w rozwiązaniach opartych na relacyjnych bazach danych

Początkowo, relacyjne bazy danych miały stanowić rozwiązanie, które pozwoli oddzielić mechanizmy fizycznego przechowywania danych od ich koncepcyjnej reprezentacji i jednocześnie wniesie odpowiedni model matematyczny na potrzeby reprezentacji danych i zapytań. Relacyjny model danych wprowadził także wysokopoziomowe języki zapytań, które stanowiły pierwszą ciekawą i efektywną alternatywę dla interfejsów języków programowania — ich wprowadzenie pozwoliło na znaczne przyspieszenie tworzenia nowych zapytań. Przykładem relacyjnej reprezentacji danych jest struktura z rysunku 1.2. Systemy relacyjne były początkowo przeznaczone do tych samych zastosowań co wspomniane w poprzednim podrozdziale wczesne rozwiązania oparte na bazach danych, jednak szybko okazało się, że oferują nieporównanie większą elastyczność w obszarze szybkiego tworzenia nowych zapytań oraz reorganizacji struktury bazy danych (np. w odpowiedzi na zmiany pierwotnych wymagań). Oznaczało to znaczne zwiększenie *abstrakcji danych* i *niezależności danych od programu* w porównaniu z wcześniejszymi rozwiązaniami.



Wczesne eksperymentalne systemy relacyjne były tworzone i rozwijane już w późnych latach 70-tych, natomiast komercyjne relacyjne systemy zarządzania bazami danych (ang. *Relational Database Management System* — *RDBMS*) zostały wprowadzone we wczesnych latach 80-tych — pierwsze rozwiązania tego typu były mało wydajne, ponieważ podczas operacji dostępu do rekordów wzajemnie powiązanych za pomocą relacji nie wykorzystywano jeszcze wskaźników do fizycznych obszarów pamięci. Wraz z rozwojem nowych technologii przechowywania danych, nowych technik indeksowania informacji, a także lepszych metod przetwarzania i optymalizacji zapytań, wydajność relacyjnych systemów zarządzania bazami danych stopniowo wzrastała. Relacyjne bazy danych stały się ostatecznie dominującym rodzajem systemów baz danych, przynajmniej w tradycyjnych zastosowaniach. Relacyjne bazy danych są obecnie dostępne dla niemal wszystkich typów komputerów, od niewielkich komputerów osobistych do dużych serwerów.

### 1.7.3. Aplikacje obiektowe i konieczność wprowadzenia bardziej skomplikowanych baz danych

Rozwój obiektowych języków programowania, który nastąpił w latach 80-tych, oraz potrzeba przechowywania i udostępniania obiektów o skomplikowanej strukturze doprowadziły ostatecznie do rozwoju obiektowych baz danych. Obiektowe bazy danych były początkowo uważane za rozwiązanie konkurencyjne względem relacyjnych baz danych, ponieważ oferowały bardziej ogólne struktury danych. Twórcy tego typu baz danych zastosowali także wiele przydatnych reguł znanych ze świata obiektowych języków programowania, a więc abstrakcyjne typy danych, hermetyzację operacji, dziedziczenie czy mechanizm identyfikacji obiektów. Poziom złożoności tego modelu i brak wcześniejszych standardów przyczynił się jednak do bardzo ograniczonej popularności tego typu rozwiązań. Obiektowe bazy danych są obecnie wykorzystywane do bardzo wyspecjalizowanych celów, takich jak projektowanie inżynierskie, publikowanie multimediiów czy obsługa systemów produkcji przemysłowej. Mimo oczekiwań, że rozwiązania te zyskają dużą popularność, ich ogólny udział na rynku produktów bazodanowych pozostaje niski. Ponadto wiele mechanizmów obiektowych wprowadzono w nowszych wersjach relacyjnych SZBD, co doprowadziło do powstania obiektowo-relacyjnych systemów zarządzania bazami danych (ORSZBD).

### 1.7.4. Wykorzystywana w handlu elektronicznym wymiana danych za pośrednictwem internetu z użyciem XML-a

Internet jest ogromną siecią połączonych ze sobą komputerów. Użytkownicy internetu mogą tworzyć dokumenty za pomocą specjalnych języków publikacji w sieci WWW, takich jak HTML (od ang. *HyperText Markup Language*), i umieszczać tak przygotowane materiały na serwerach WWW, gdzie pozostali użytkownicy (klienci) mogą uzyskiwać do nich dostęp i przeglądać je za pomocą przeglądarek. Dokumenty można ze sobą łączyć za pomocą tzw. **hiperłączy**, które są wskaźnikami do innych dokumentów. W latach 90-tych ogromną popularność zdobyło zupełnie nowe zastosowanie internetu — handel elektroniczny (ang. *electronic commerce* lub *e-commerce*). Szybko okazało się, że część informacji (na przykład o lotach, cenach towarów lub dostępności produktów) wyświetlanych na stronach WWW sklepów internetowych może być dynamicznie pobierana z działającego w tle systemu za-

rzządzania bazą danych. W odpowiedzi na rosnące zapotrzebowanie na rozwiązania tego typu opracowano wiele technik wymiany danych za pośrednictwem stron WWW. Jednym ze standardów wymiany danych pomiędzy różnymi typami baz danych i stron internetowych jest *XML* (ang. *eXtended Markup Language*). Język XML łączy w sobie mechanizmy typowe dla systemów opartych na dokumentach z elementami charakterystycznymi dla modelowania baz danych. Przegląd XML-a znajdziesz w rozdziale 13.

### 1.7.5. Rozszerzanie możliwości współczesnych systemów baz danych z myślą o nowych zastosowaniach

Sukces systemów baz danych w tradycyjnych zastosowaniach zachęcił programistów do opracowania zupełnie innych rodzajów aplikacji, w których z powodzeniem będzie można wykorzystywać podobne technologie. Nowe rozwiązania miały być wykorzystywane w obszarach, w których do tej pory dominowały tradycyjne aplikacje operujące na wyspecjalizowanych plikach i strukturach danych. Poniżej przedstawiono przykłady takich obszarów:

- Rozwiązania **naukowe**, które wymagają przechowywania ogromnych ilości danych generowanych w trakcie eksperymentów naukowych w takich obszarach jak fizyka cząstek elementarnych, odwzorowywanie ludzkiego materiału genetycznego czy odkrywanie struktury białek.
- Przechowywanie i udostępnianie **obrazów**, począwszy od zeskanowanych artykułów prasowych i rodzinnych fotografii, a skończywszy na zdjęciach satelitarnych i obrazach powstających podczas takich procedur medycznych jak prześwietlenia rentgenowskie czy rezonans magnetyczny.
- Przechowywanie i udostępnianie obrazu **wideo**, np. filmów czy **klipów wideo** z wiadomościami telewizyjnymi lub zapisem z osobistych kamer cyfrowych.
- Rozwiązania w zakresie **drażenia danych**, w których ogromne ilości danych są poddawane analizie pod kątem wystąpień konkretnych wzorców lub relacji, a także w celu wyszukiwania nietypowych wzorców w związku z wykrywaniem oszustw z użyciem kart kredytowych.
- Techniki **przestrzenne**, w których przechowuje i analizuje się przestrzenne lokalizacje takich danych jak informacje o pogodzie czy mapy wykorzystywane w systemach informacji geograficznej i do nawigacji samochodowej.
- Zastosowania związane z przechowywaniem **szeregów czasowych**, w tym danych ekonomicznych odpowiadających równomiernie rozłożonym punktom w czasie; mogą to być np. wykresy dziennej sprzedaży lub miesięcznego produktu krajowego brutto.

Bardzo szybko okazało się, że podstawowe relacyjne systemy baz danych nie nadają się do obsługi wielu z wymienionych przed chwilą zastosowań — zwykle powodem takiego stanu rzeczy jest jeden z poniższych problemów:

- Do modelowania istniejącego stanu mini-świata potrzebne są bardziej skomplikowane struktury danych niż prosta reprezentacja relacyjna.
- Poza podstawowymi typami numerycznymi i znakowymi niezbędne są nowe typy danych.

- Do operowania na nowych typach danych potrzebne są zupełnie nowe operacje i konstrukcje języka zapytań.
- Niezbędne są nowe struktury przechowywania i indeksowania danych.

Wymienione powyżej ograniczenia spowodowały, że twórcy systemów zarządzania bazami danych zaczęli wprowadzać nowe funkcje. Zastosowania niektórych z nich są uniwersalne — dotyczy to np. przenoszenia mechanizmów z obiektowych baz danych do systemów relacyjnych. Inne elementy nowej funkcjonalności mają konkretne przeznaczenie i są zwykle oferowane w postaci opcjonalnych modułów, które można stosować do realizacji specyficznych zadań. Przykładowo, użytkownicy mogą zakupić moduł obsługujący szeregi czasowe, który będą następnie wykorzystywali w swoim relacyjnym systemie zarządzania bazą danych do obsługi takich szeregów.

### 1.7.6. Powstanie systemów przechowywania big data i baz NOSQL

W pierwszej dekadzie naszego wieku rozpowszechnienie się rozwiązań i platform takich jak serwisy społecznościowe, witryny dużych sklepów elektronicznych, indeksy wyszukiwarek oraz pamięci masowe i kopie bezpieczeństwa przechowywane w chmurze doprowadziło do znacznego wzrostu ilości danych przechowywanych w dużych bazach i na pojemnych serwerach. Do zarządzania tymi ogromnymi bazami potrzebne były nowe rodzaje systemów baz danych. Systemy te musiały zapewniać szybsze wyszukiwanie i pobieranie danych, a także niezawodne i bezpieczne przechowywanie nietradycyjnych rodzajów danych, np. postów i tweetów w mediach społecznościowych. Niektóre z wymagań stawianych nowym systemom nie były zgodne z relacyjnymi SZBD opartymi na SQL-u (SQL to standardowy model danych i język baz relacyjnych). Nazwa *NOSQL* jest zwykle interpretowana jako *Not Only SQL*, co oznacza, że w systemach zarządzających dużymi ilościami danych niektóre informacje są przechowywane z użyciem systemów SQL-owych, natomiast inne — za pomocą baz NOSQL. Zależy to od wymogów stawianych danej aplikacji.

## 1.8. Kiedy nie należy używać systemów zarządzania bazami danych

Pomimo wymienionych w tym rozdziale zalet systemów zarządzania bazami danych, istnieje kilka sytuacji, w których stosowanie tego typu rozwiązań wiąże się z niepotrzebnymi kosztami, których z kolei można uniknąć stosując tradycyjne rozwiązania przetwarzające pliki. Nadmierne koszty stosowania systemów zarządzania bazami danych mogą wynikać z następujących uwarunkowań:

- Zbyt wysoki początkowy koszt inwestycji w sprzęt, oprogramowanie i szkolenia.
- Zbyt mały poziom szczegółowości oferowanej przez system zarządzania bazą danych w zakresie definiowania i przetwarzania danych.
- Negatywne skutki dla wydajności, będące efektem stosowania mechanizmów bezpieczeństwa, sterowania współbieżnego, odzyskiwania i zapewniania spójności danych.

Zatem zastosowanie tradycyjnych rozwiązań opartych na przetwarzaniu zwykłych plików z danymi może być usprawiedliwione w następujących okolicznościach:

- Aplikacje są proste, dobrze zdefiniowane i nie będą w przyszłości zmieniane.
- Zostały zdefiniowane ściśle wymagania dotyczące projektowanego pakietu oprogramowania, których nie można spełnić przy użyciu systemu zarządzania bazą danych (np. z powodu mniejszej wydajności).
- Używane są systemy wbudowane z ograniczoną pamięcią masową, w której nie zmieści się uniwersalny system zarządzania bazą danych.
- Reprezentowane dane nie muszą być dostępne dla wielu użytkowników.

W niektórych branżach i zastosowaniach zdecydowano się nie korzystać z uniwersalnych systemów zarządzania bazą danych. Przykładowo, w wielu narzędziach CAD (ang. *computer-aided design*) używanych przez inżynierów mechaników i inżynierów budownictwa stosuje się niestandardowe oprogramowanie do zarządzania plikami i danymi, rozwijane pod kątem wewnętrznego operowania na rysunkach i obiektach trójwymiarowych. Podobnie systemy komunikacji i przełączników projektowane przez firmy takie jak AT&T były wczesnym oprogramowaniem bazodanowym, które miało działać bardzo szybko dla hierarchicznie uporządkowanych danych, aby umożliwić szybki dostęp do nich i przełączanie rozmów. W systemach informacji geograficznej często implementowane są niestandardowe schematy porządkowania danych, aby umożliwić rozwijanie wydajnych funkcji przetwarzania map, konturów, linii, wielokątów itd.

## 1.9. Podsumowanie

W tym rozdziale zdefiniowaliśmy bazę danych jako zbiór wzajemnie powiązanych danych, gdzie przez *dane* rozumiemy zarejestrowane fakty. Typowa baza reprezentuje jakiś aspekt tego świata i jest wykorzystywana do określonych celów przez jedną lub więcej grup użytkowników. System zarządzania bazą danych jest uniwersalnym pakietem oprogramowania, który umożliwia implementowanie i utrzymywanie skomputeryzowanej bazy danych. Baza danych w połączeniu z tym oprogramowaniem tworzy system bazy danych. Wymieniliśmy w tym rozdziale szereg właściwości, które odróżniają rozwiązania oparte na bazach danych od tradycyjnych aplikacji przetwarzających pliki z danymi. W dalszej kolejności omówiliśmy najważniejsze kategorie użytkowników baz danych — nazwaliśmy ich *aktorami na scenie*. Stwierdziliśmy także, że poza wymienionymi rodzajami użytkowników baz danych istnieje także kilka kategorii personelu wspierającego tych użytkowników podczas ich pracy w środowisku bazy danych — nazwaliśmy ich *pracownikami poza sceną*.

Przedstawiliśmy także listę możliwości i funkcji, które powinny być dostępne na poziomie systemów zarządzania bazami danych dla administratorów, projektantów oraz użytkowników baz danych, i które powinny im ułatwiać projektowanie, administrowanie i wykorzystywanie tychże baz. Dalej krótko omówiliśmy historię i ewolucję rozwiązań opartych na bazach danych. Zwróciliśmy uwagę na niedawny szybki wzrost ilości i rodzajów danych, które muszą być przechowywane w bazach. Opisaliśmy też pojawienie się nowych systemów do obsługi zastosowań z obszaru big data. Wreszcie skupiliśmy się na kosztach wynikających ze stosowania systemów zarządzania bazami danych oraz przedstawiliśmy sytuacje, w których wykorzystywanie tego typu rozwiązań może powodować więcej strat niż korzyści.

## Pytania powtórkowe

- 1.1. Zdefiniuj następujące pojęcia: *dane, baza danych, system zarządzania bazą danych, system bazy danych, katalog bazy danych, niezależność programu od danych, perspektywa użytkownika, administrator bazy danych, użytkownik końcowy, transakcja zapuszkowana, dedukcyjny system bazy danych, trwały obiekt, metadane oraz aplikacja przetwarzająca transakcje.*
- 1.2. Jakie cztery główne typy działań wiążą się ze stosowaniem baz danych? Krótko omów każdy z wymienionych typów.
- 1.3. Omów główne właściwości rozwiązań opartych na bazach danych i przedstaw krótko różnice dzielące te rozwiązania od ich odpowiedników mających postać tradycyjnych systemów plików.
- 1.4. Jakie są zadania administratora bazy danych i projektanta bazy danych?
- 1.5. Wymień różne typy użytkowników końcowych bazy danych. Omów główne czynności podejmowane przez użytkowników należących do poszczególnych typów.
- 1.6. Przedstaw możliwości, jakie powinny być oferowane przez systemy zarządzania bazami danych.
- 1.7. Wyjaśnij różnice między systemami baz danych a systemami wyszukiwania informacji.

## Ćwiczenia

- 1.8. Przedstaw kilka nieformalnych zapytań i operacji aktualizujących, które chciałbyś zastosować dla bazy danych przedstawionej na rysunku 1.2.
- 1.9. Jaka jest różnica pomiędzy kontrolowaną a niekontrolowaną nadmiarowością? Odpowiedź zilustruj odpowiednimi przykładami.
- 1.10. Wymień wszystkie związki łączące rekordy bazy danych przedstawionej na rysunku 1.2.
- 1.11. Podaj kilka dodatkowych perspektyw, które mogą być przydatne dla innych grup użytkowników bazy danych zademonstrowanej na rysunku 1.2.
- 1.12. Przedstaw kilka przykładów więzów integralności, które Twoim zdaniem powinny zostać zdefiniowane dla bazy danych zaprezentowanej na rysunku 1.2.
- 1.13. Przedstaw przykładowe systemy, w których sensowne może być zastosowanie tradycyjnego przetwarzania plików zamiast baz danych.
- 1.14. Przyjrzyj się rysunkowi 1.2.
  - a) Załóżmy, że nazwa wydziału INF (Informatyki) zmienia się na wydział INFIIO (Informatyki i Inżynierii Oprogramowania). Modyfikowany jest też powiązany przedrostek numeru kursu. Wymień kolumny bazy danych, które trzeba będzie zaktualizować.

- b) Czy potrafisz tak zmienić strukturę kolumn PRZEDMIOT, DZIAŁ i WYMAGANIA\_WSTĘPNE, aby konieczne było zaktualizowanie tylko jednej kolumny?

## Wybrane publikacje

Wydanie miesięcznika *Communications of the ACM* z października roku 1991 oraz książka Kima (1995) zawierają wiele artykułów i materiałów opisujących systemy zarządzania bazami danych następnej generacji; wiele spośród rozwiązań omówionych w tym pierwszym czasopiśmie jest już dostępnych w oferowanych obecnie komercyjnych pakietach oprogramowania. Wydanie miesięcznika *ACM Computing Surveys* z marca roku 1976 zawiera wczesne wprowadzenie do systemów baz danych, które dla zainteresowanych czytelników może stanowić ciekawy materiał historyczny na ten temat. Referencje do innych zagadnień, systemów i zastosowań przedstawionych w tym rozdziale umieszczamy dalej, w rozdziałach, gdzie poszczególne tematy są opisane szczegółowo.





## Architektura systemów baz danych i związane z nimi pojęcia

Architektura pakietów systemów zarządzania bazami danych ewoluowała od wczesnych systemów monolitycznych, w których cały pakiet oprogramowania SZBD był ściśle zintegrowanym systemem, do współczesnych pakietów tego typu, które składają się z modułów i są konstruowane w oparciu o model klient-serwer. Widoczny ostatnio wzrost ilości danych wymagających składowania doprowadził do powstania systemów baz danych o architekturze rozproszonej, składających się z tysięcy komputerów zarządzających magazynami danych. Ewolucja systemów zarządzania bazami danych odzwierciedla pojawiające się w technice komputerowej trendy, które sprawiły, że wielkie scentralizowane komputery zostały zastąpione setkami rozproszonych stacji roboczych i komputerów osobistych połączonych za pośrednictwem sieci komunikacyjnej z serwerami rozmaitych typów — serwerami WWW, serwerami baz danych, serwerami plików, serwerami aplikacji itp. Nowe środowiska **chmur obliczeniowych** obejmują tysiące dużych serwerów zarządzających tzw. **big data** na potrzeby użytkowników internetu.

W podstawowej architekturze systemu zarządzania bazą danych typu klient-serwer funkcjonalność tego systemu jest dzielona pomiędzy dwa typy modułów<sup>1</sup>. **Moduł klienta** jest zwykle projektowany w taki sposób, aby możliwe było jego uruchamianie w urządzeniach mobilnych, na stacjach roboczych lub komputerach osobistych. Moduły klienta są zwykle używane do obsługi aplikacji wykorzystujących bazę danych oraz do udostępniania odpowiednich interfejsów. Oznacza to, że tak naprawdę to moduł klienta obsługuje działania użytkownika na bazie danych i udostępnia mu przyjazny interfejs — np. aplikację na urządzenia mobilne albo oparty na formularzach lub menu graficzny interfejs użytkownika (ang. *Graphical User Interface* — *GUI*) w komputerach osobistych. Drugi rodzaj modułów — tzw. **moduły serwera** — obsługuje zazwyczaj przechowywanie danych, operacje dostępu do danych, operacje przeszukiwania danych i inne, podobne funkcje. Architekturę typu klient-serwer omówimy bardziej szczegółowo w podrozdziale 2.5. Wcześniej musimy poświęcić chwilę na przestudiowanie podstawowych zagadnień, które umożliwią nam lepsze zrozumienie architektur współczesnych baz danych.

W tym rozdziale zaprezentujemy terminologię i podstawowe pojęcia, które będą wykorzystywane w książce. Rozpocznemy (w podrozdziale 2.1) od omówienia modeli danych oraz zdefiniowania pojęć schematów i egzemplarzy, które mają zasadnicze znaczenie podczas rozważań na temat systemów baz danych. Następnie, w podrozdziale 2.2 omówimy

---

<sup>1</sup> W podrozdziale 2.5 przekonamy się, że istnieją różne odmiany tej prostej, *dwuwarstwowej* architektury klient-serwer.

trójwarstwową architekturę systemów zarządzania bazami danych (ang. *three-schema DBMS*) oraz niezależność danych — w ten sposób dojdziemy do problemu oczekiwanych działań systemu zarządzania bazą danych z perspektywy jej użytkownika. W podrozdziale 2.3 opisujemy typy interfejsów i języków oferowanych przez większość współczesnych systemów zarządzania bazami danych. Podrozdział 2.4 poświęcono omówieniu środowiska oprogramowania systemu bazy danych. W podrozdziale 2.5 zawarto przegląd rozmaitych typów architektur typu klient-serwer. I wreszcie w podrozdziale 2.6 zaprezentowaliśmy klasyfikację typów pakietów systemów zarządzania bazami danych. Podrozdział 2.7 zawiera podsumowanie tego rozdziału.

Materiał zawarty w podrozdziałach od 2.4 do 2.6 dotyczy bardziej szczegółowych zagadnień, które nie muszą być omawiane w trakcie podstawowego kursu wprowadzającego w świat baz danych.

## 2.1. Modele danych, schematy i egzemplarze

Jedną z podstawowych właściwości rozwiązań opartych na bazach danych jest to, że ich użytkownicy korzystają z abstrakcji danych. **Abstrakcja danych** oznacza ukrywanie szczegółów struktury i przechowywania danych oraz podkreślanie najważniejszych ich aspektów, co pozwala lepiej zrozumieć informacje. Jedną z głównych cech podejścia wykorzystującego bazy jest zapewnianie abstrakcji danych, dzięki czemu różni użytkownicy mogą korzystać z nich na preferowanym poziomie szczegółowości. **Model danych** — czyli zbiór pojęć, które można wykorzystywać do opisywania struktury bazy danych — zapewnia środki niezbędne do osiągnięcia oczekiwanego poziomu abstrakcji<sup>2</sup>. Przez *strukturę bazy danych* rozumiemy typy danych, związki i ograniczenia dotyczące danych. Większość modeli danych zawiera także zbiór **podstawowych operacji**, które umożliwiają otrzymywanie i aktualizowanie informacji zawartych w bazie danych.

Poza podstawowymi operacjami zdefiniowanymi w modelu danych, coraz bardziej popularne staje się dołączanie do tego modelu także elementów określających **aspekty dynamiczne i zachowania** aplikacji bazodanowych. Dzięki temu projektanci baz danych mogą stosunkowo łatwo określać zbiory poprawnych **operacji definiowanych przez użytkowników**, które będą bez problemów wykonywane na obiektach bazy danych<sup>3</sup>. Przykładem takich operacji definiowanych przez użytkowników może być procedura *OBLICZ\_ŚROC*, którą można stosować dla obiektów przechowywanych w pliku *STUDENT*. Z drugiej strony, uniwersalne operacje wstawiania, usuwania, modyfikowania i odczytywania dowolnych rodzajów obiektów są zwykle dołączane do *podstawowych operacji modelu danych*. Elementy definiujące zachowania są nie tylko zasadniczym elementem obiektowych

---

<sup>2</sup> Słowo *model* jest niekiedy wykorzystywane w odniesieniu do konkretnego opisu bazy danych lub do schematu — przykładowo, można się spotkać z wyrażeniem *model danych marketingowych*. W tej książce nie będziemy używać słowa *model* w tym znaczeniu.

<sup>3</sup> Dołączanie elementów opisujących zachowanie baz danych odzwierciedla kierunek rozwoju systemów tego typu, który przewiduje, że działania związane z projektowaniem baz danych i projektowaniem oprogramowania z czasem zostaną połączone w zbiór tych samych czynności. Określanie zachowania oprogramowania tradycyjnie było elementem projektowania oprogramowania.

modeli danych (patrz rozdział 12.), ale także zostały z powodzeniem wprowadzone do bardziej tradycyjnych modeli danych. Przykładowo, modele obiektowo-relacyjne (patrz rozdział 12.) rozszerzają tradycyjny model relacyjny między innymi o elementy tego typu. W podstawowym modelu relacyjnym możliwe jest dodawanie zachowań do relacji za pomocą trwałych modułów składowanych, nazywanych zwykle procedurami składowanymi (patrz rozdział 10.).

### 2.1.1. Kategorie modeli danych

Istnieje wiele różnych modeli danych, które możemy skategoryzować np. według elementów wykorzystywanych do opisywania struktury bazy danych. Przykładowo, **wysokopoziomowe** i **konceptyjne modele danych** opierają się na rozwiązaniach, które są zbliżone do abstrakcyjnego sposobu postrzegania danych przez wielu użytkowników, natomiast **niskopoziomowe** i **fizyczne modele danych** opierają się na szczegółowych opisach metod przechowywania danych na dysku komputera (zwykle na dyskach magnetycznych). Niskopoziomowe modele danych są przeznaczone przede wszystkim dla specjalistów w zakresie sprzętu komputerowego, nie dla typowych użytkowników końcowych systemów baz danych. Pomiędzy wspomnianą parą skrajnych rozwiązań istnieje klasa **reprezentacyjnych** (lub **implementacyjnych**<sup>4</sup>) **modeli danych**, które z jednej strony opierają się na pojęciach zrozumiałych dla przeciętnego użytkownika końcowego, ale jednocześnie nie odbiegają znacząco od sposobu organizowania danych na fizycznym nośniku w komputerze. Reprezentacyjne modele danych ukrywają co prawda niektóre szczegóły związane z przechowywaniem danych, ale mogą być implementowane w systemach komputerowych w sposób bezpośredni.

Konceptyjne modele danych opierają się na elementach takich jak encje, atrybuty czy związki. Pojedyncza **encja** reprezentuje obiekt lub pojęcie z mini-świata (np. pracownika lub projekt), które zostało opisane w bazie danych. **Atrybut** reprezentuje pewną interesującą własność, która dodatkowo opisuje daną encję — może to być np. nazwisko lub pensja pracownika. **Związek** pomiędzy dwoma (lub więcej) encjami reprezentuje łączące je powiązanie — przykładowo, może to być związek *pracuje nad*, który łączy encje reprezentujące pracownika i projekt. W rozdziale 3. zaprezentujemy model związków encji (ang. *Entity-Relationship*, w skrócie *ER*), który jest popularnym wysokopoziomym konceptyjnym modelem danych. W rozdziale 4. omówimy dodatkowe elementy charakterystyczne dla konceptyjnego modelowania danych, w tym generalizację, specjalizację oraz kategorie (typy unii).

W tradycyjnych, komercyjnych systemach zarządzania bazami danych wykorzystuje się najczęściej reprezentacyjne lub implementacyjne modele danych. Takie modele obejmują zarówno popularny obecnie **relacyjny model danych**, jak i bardziej tradycyjne, **sieciowe** i **hierarchiczne modele danych**, które były powszechnie stosowane w przeszłości. Część 3. tej książki poświęcono relacyjnemu modelowi danych, charakterystycznym dla niego operacjom i językom, a także niektórym technikom programowania aplikacji operujących na relacyjnych bazach danych. Standard języka SQL dla relacyjnych baz danych opisano w rozdziałach 6. i 7. Reprezentacyjne modele danych reprezentują dane za

---

<sup>4</sup> Określenie *implementacyjny model danych* nie jest standardowym pojęciem. Wprowadziliśmy je tutaj na potrzeby opisu modeli danych dostępnych w komercyjnych systemach bazodanowych.

pomocą struktur rekordów, stąd czasami są nazywane **modelami danych opartymi na rekordach**.

**Obiektowe modele danych** należy traktować jak zupełnie nową rodzinę wysokopoziomowych implementacyjnych modeli danych, które są nieco bliższe koncepcyjnym modelom danych. Ogólne właściwości obiektowych baz danych i standardu *ODMG* (ang. *Object Data Management Group*) zaproponowanego przez grupę o tej samej nazwie omówimy w rozdziale 12. Obiektowe modele danych są często wykorzystywane także w roli wysokopoziomowych modeli koncepcyjnych (w szczególności dotyczy to dziedziny inżynierii oprogramowania).

Fizyczne modele danych opisują sposób przechowywania danych na komputerze w postaci plików, zatem reprezentują takie informacje jak formaty rekordów, uporządkowanie rekordów czy ścieżki dostępu. **Ścieżka dostępu** jest strukturą, która zwiększa efektywność operacji przeglądania bazy danych w poszukiwaniu konkretnych rekordów (np. dzięki indeksowaniu lub mieszaniu). Techniki fizycznego przechowywania danych i struktury dostępowe omówimy w rozdziałach 16. i 17. **Indeks** to przykładowa ścieżka umożliwiająca bezpośredni dostęp do danych za pomocą pojęcia z indeksu lub słowa kluczowego. Taka struktura przypomina indeks z końcowej części tej książki, przy czym może być uporządkowana liniowo, hierarchicznie (za pomocą struktury drzewiastej) lub w jeszcze inny sposób.

Inną kategorią są **samoopisujące się modele danych**. W systemach opartych na takich modelach opis danych jest połączony z ich wartościami. W tradycyjnych systemach SZBD opis (schemat) jest oddzielony od danych. Modele z tej kategorii to **XML** (rozdział 12.), a także liczne **magazyny danych typu klucz-wartość** i **systemy NOSQL** (rozdział 24.), opracowane ostatnio na potrzeby zarządzania big data.

## 2.1.2. Schematy, egzemplarze i stany baz danych

W każdym modelu danych bardzo ważne jest rozróżnienie *opisu* bazy danych od *samej bazy danych*. Opis bazy danych jest najczęściej nazywany **schematem bazy danych** — w założeniu taki schemat powstaje podczas projektowania bazy danych i nie powinien ulegać częstym zmianom<sup>5</sup>. W większości modeli danych istnieją pewne konwencje prezentowania schematów w postaci diagramów. Graficzna prezentacja schematu jest nazywana **diagramem schematu**. Na rysunku 2.1 zademonstrowano diagram schematu dla bazy danych zaprezentowanej w poprzednim rozdziale na rysunku 1.2 — diagram przedstawia jedynie strukturę poszczególnych typów rekordów, zatem nie obejmuje rzeczywistych egzemplarzy tych rekordów. Każdy obiekt w tego typu schemacie (np. *STUDENT* lub *PRZEDMIOT*) nazywamy **konstrukcją schematu**.

Diagram schematu przedstawia tylko *niektóre aspekty* reprezentowanego schematu bazy danych, w tym nazwy typów rekordów i elementów danych oraz niektóre rodzaje ograniczeń. Pozostałe aspekty schematu bazy danych w ogóle nie są wyrażane na tego

---

<sup>5</sup> Zmiany schematów baz danych są często nieuniknione w przypadku zmian wymagań leżących u podstaw danego rozwiązania. Nowsze systemy baz danych oferują nawet specjalne operacje stworzone z myślą o modyfikowaniu schematów.

typu diagramach; przykładowo, na rysunku 2.1 nie są widoczne ani typy danych poszczególnych elementów danych, ani związki łączące poszczególne pliki. Na diagramach schematów nie jest reprezentowanych wiele typów ograniczeń. Warto pamiętać, że ograniczenia w postaci „studenci informatyki muszą zdać egzamin z przedmiotu *INF1310* zanim zostanie im zaliczony drugi rok” są dosyć trudne do reprezentowania w postaci graficznej.



RYСУNEK 2.1. Diagram schematu dla bazy danych przedstawionej na rysunku 1.2

Rzeczywiste informacje przechowywane w bazie danych mogą się dosyć często zmieniać. Przykładowo, zawartość bazy danych zaprezentowanej na rysunku 1.2 zmienia się za każdym razem, gdy dodajemy nowego studenta lub wpisujemy nową ocenę dla istniejącego studenta. Informacje przechowywane w bazie danych w konkretnym momencie czasu są nazywane **stanem bazy danych** lub **migawką bazy danych (snapshotem)**. Stan bazy danych jest także nazywany *bieżącym* zbiorem zawartych w niej **wystąpień** lub **egzemplarzy**. W danym stanie bazy danych każda konstrukcja schematu musi zawierać własny bieżący *zbiór wystąpień*; przykładowo, konstrukcja *STUDENT* będzie zawierała zbior encji (rekordów) reprezentujących pojedynczych studentów, który będzie stanowił jej egzemplarze. Wiele stanów baz danych można konstruować w taki sposób, aby odpowiadały konkretnym schematom baz danych. Za każdym razem, gdy dodajemy lub usuwamy rekord albo przynajmniej zmieniamy wartość elementu danych w istniejącym rekordzie, zmieniamy stan bazy danych — a więc wprowadzamy bazę danych w nowy stan.

Rozróżnienie pomiędzy schematem bazy danych a stanem bazy danych jest bardzo ważne. Kiedy **definiujemy** nową bazę danych, określamy w systemie zarządzania bazą danych jedynie jej schemat. Na tym etapie baza danych znajduje się w tzw. *pustym stanie* (nie zawiera żadnych informacji). Bazę danych wprowadzamy w *stan początkowy* w momencie, w którym po raz pierwszy **wypełniamy** ją początkowymi danymi. Od tej pory każda zastosowana dla naszej bazy danych operacja aktualizacji powoduje, że otrzymujemy inny stan bazy danych. Warto pamiętać, że w każdym momencie można wyznaczyć dla bazy danych jej *stan bieżący*<sup>6</sup>. Za zapewnienie, aby każdy stan bazy danych był **stanem pra-**

<sup>6</sup> Stan bieżący bazy danych jest także nazywany *bieżącą migawką bazy danych*. Nazywano go też egzemplarzem bazy danych, jednak określenie *egzemplarz* wolimy stosować do poszczególnych rekordów.

**widłowym** (zgodnym z ograniczeniami i strukturą ze schematu), odpowiada po części system zarządzania bazą danych. Oznacza to, że szczególnie ważne jest zdefiniowanie na potrzeby systemu zarządzania bazą danych jej prawidłowego schematu — schemat bazy danych musi więc być projektowany z wyjątkową ostrożnością. System zarządzania bazą danych przechowuje konstrukcje zdefiniowanego schematu wraz z ograniczeniami (nazywanymi łącznie **metadany**mi) w swoim katalogu, dzięki czemu oprogramowanie systemu zarządzania bazą danych może się do takiego schematu odwoływać za każdym razem, gdy jest to konieczne. Schemat bazy danych jest niekiedy nazywany **intensją**, natomiast stan bazy danych jest nazywany ekstensją (**rozwinieciem**) określonego schematu.

Chociaż (jak już wspominaliśmy) schemat bazy danych nie powinien ulegać częstym zmianom, nierzadko okazuje się, że modyfikacje wprowadzone w wymaganiach wymuszają zastosowanie zmian także w schemacie. Przykładowo, możemy zdecydować, że każdy z rekordów danego pliku musi dodatkowo zawierać jeszcze jeden element danych — taka zmiana mogłaby polegać np. na dodaniu do schematu pliku *STUDENT* (patrz rysunek 2.1) elementu danych *DataUrodzenia*. Takie działania są niekiedy nazywane **ewolucją schematu**. Większość współczesnych systemów zarządzania bazami danych oferuje pewne operacje ułatwiające przeprowadzanie ewolucji danych, gdy baza już działa.

## 2.2. Trójwarstwowa architektura i niezależność danych

Oto trzy z czterech wymienionych w podrozdziale 1.3 ważnych cech rozwiązań opartych na bazach danych: (1) wykorzystywanie katalogu do przechowywania opisu (schematu) bazy danych, (2) oddzielenie programów od danych (tzw. niezależność programu od danych i niezależność programu od operacji) oraz (3) obsługa wielu perspektyw użytkowników. W tym podrozdziale omówimy jedną z popularnych architektur systemów baz danych, powszechnie znaną jako **architektura trójwarstwowa**<sup>7</sup>, która została zaproponowana z myślą o ułatwieniu uzyskiwania (i wizualnego przedstawiania) tych własności w praktyce. W dalszej części tego podrozdziału omówimy pojęcia związane z niezależnością danych.

### 2.2.1. Architektura trójwarstwowa

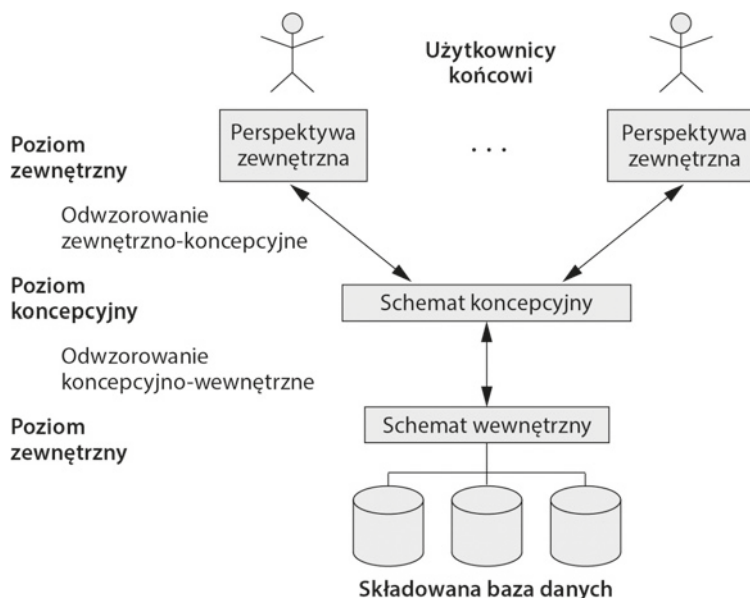
Celem opracowania architektury trójwarstwowej (patrz rysunek 2.2) było oddzielenie aplikacji użytkownika od fizycznej bazy danych. Architektura trójwarstwowa przewiduje możliwość definiowania schematów na następujących trzech poziomach:

- (1) **Poziom wewnętrzny** zawiera **wewnętrzny schemat**, który opisuje fizyczną strukturę przechowywania bazy danych. Wewnętrzny schemat wykorzystuje fizyczny model danych i opisuje wszystkie szczegóły związane z przechowywaniem informacji zawartej w bazie danych oraz z jej ścieżkami dostępu.

---

<sup>7</sup> Architektura trójwarstwowa jest także nazywana *architekturą ANSI/SPARC* od skrótów organizacji, która to rozwiązanie zaproponowała (Tsichritzis i Klug, 1978).





RYSUNEK 2.2. Architektura trójwarstwowa

- (2) **Poziom koncepcyjny** zawiera **schemat koncepcyjny**, który opisuje strukturę całej bazy danych na potrzeby społeczności końcowych użytkowników bazy danych. Schemat koncepcyjny ukrywa szczegóły fizycznych struktur przechowywania danych i koncentruje się na opisywaniu encji, typów danych, związków, operacji użytkownika oraz ograniczeń. Do opisywania schematów koncepcyjnych podczas implementowania systemu bazy danych wykorzystuje się zwykle reprezentacyjny model danych. *Implementacyjny schemat koncepcyjny* opiera się często na *koncepcyjnym projekcie schematu* w wysokopoziomowym modelu danych.
- (3) **Poziom zewnętrzny** (nazywany także **poziomem perspektyw**) zawiera wiele **schematów zewnętrznych** lub **schematów użytkownika**. Każdy taki schemat zewnętrzny opisuje tylko tę część bazy danych, która znajduje się w obszarze zainteresowań konkretnej grupy użytkowników, i ukrywa przed tą grupą użytkowników pozostałe części bazy danych. Podobnie jak w poprzednim przypadku, każdy schemat zewnętrzny jest zazwyczaj implementowany w oparciu o reprezentacyjny model danych, często bazujący na zewnętrznym projekcie schematu w wysokopoziomowym modelu danych.

Architektura trójwarstwowa jest wygodnym narzędziem, za pomocą którego użytkownik może łatwo wizualizować poziomy schematów wykorzystywanego systemu bazy danych. Większość systemów zarządzania bazami danych nie oddziela tych trzech poziomów od siebie całkowicie — obsługuje architekturę tego typu tylko do pewnego stopnia. Niektóre starsze systemy zarządzania bazami danych mogą dołączać do schematów koncepcyjnych także szczegóły warstwy fizycznej. Trójwarstwowa architektura ANSI zajmuje ważne miejsce w rozwoju technologii bazodanowych, ponieważ wyraźnie odgranicza w kontekście projektowania baz poziomy zewnętrzny (użytkowników), koncepcyjny (bazy danych) i wewnętrzny (składowania danych). Nawet dziś ta architektura



znajduje zastosowania w projektowaniu SZBD. W większości systemów SZBD, które obsługują perspektywy użytkownika, schematy zewnętrzne są definiowane w oparciu o ten sam model danych, na bazie którego opisano schemat poziomu koncepcyjnego (w relacyjnych SZBD, takich jak Oracle lub SQL Server, używany jest do tego SQL).

Warto pamiętać, że wymienione trzy poziomy schematów stanowią wyłącznie *opisy* właściwych danych; informacje są zapisywane jedynie na poziomie fizycznym. W architekturze trójwarstwowej każda grupa użytkowników odwołuje się wyłącznie do własnego schematu zewnętrznego. Oznacza to, że system zarządzania bazą danych musi przekształcić żądanie zdefiniowane na schemacie zewnętrznym w odpowiednie żądanie na schemacie koncepcyjnym, po czym przekształcić otrzymany wynik w prawidłowe żądanie na schemacie wewnętrznym, które umożliwi przetwarzanie rzeczywistych informacji przechowywanych w bazie danych. Jeśli żądanie dotyczy odczytania wybranych informacji zawartych w bazie danych, dane wyciągnięte z bazy muszą zostać ponownie sformatowane do postaci odpowiadającej zewnętrznej perspektywie użytkownika. Zachodzące pomiędzy wspomnianymi trzema poziomami procesy przekształcania żądań i uzyskanych wyników są nazywane **odwzorowaniami**. Tego typu operacje mogą być kosztowne czasowo, co powoduje, że niektóre SZBD (w szczególności te, które w założeniu mają obsługiwać niewielkie bazy danych) w ogóle nie obsługują zewnętrznych perspektyw. Jednak nawet w takich systemach wykonywanie pewnych operacji odwzorowywania okazuje się niezbędne podczas przekształcania żądań pomiędzy poziomem koncepcyjnym a poziomem wewnętrznym.

## 2.2.2. Niezależność danych

Omówioną przed chwilą architekturę trójwarstwową możemy wykorzystać podczas omawiania pojęcia **niezależności danych**, które można zdefiniować jako możliwość modyfikowania schematu na jednym poziomie systemu bazy danych bez konieczności wprowadzania zmian w schemacie znajdującym się na kolejnym (wyższym) poziomie. Możemy zdefiniować dwa typy niezależności danych:

- (1) **Logiczna niezależność danych** oznacza możliwość zmiany schematu koncepcyjnego bez konieczności modyfikowania schematów zewnętrznych ani aplikacji. Zmiana schematu koncepcyjnego może mieć na celu rozbudowę bazy danych (przez dodanie nowego typu rekordu lub nowego elementu danych), modyfikację ograniczeń lub odchudzenie bazy danych (przez usunięcie typu rekordu lub elementu danych). W tym ostatnim przypadku schematy zewnętrzne odwołujące się wyłącznie do danych, które nie są usuwane i pozostają w bazie danych, w ogóle nie powinny podlegać zmianom. Przykładowo, schemat zewnętrzny z rysunku 1.5(a) nie powinien ulec zmianie na skutek zaprezentowanych na rysunku 1.6(a) modyfikacji w pliku RAPORT\_OCEN (względem stanu przedstawionego na rysunku 1.2). System zarządzania bazą danych, który zapewnia logiczną niezależność danych, musi jedynie odpowiednio zmodyfikować definicję perspektywy i właściwe reguły odwzorowywania. Po logicznej reorganizacji schematu koncepcyjnego programy aplikacji, które odwołują się do konstrukcji schematu zewnętrznego, mogą pracować tak samo jak wcześniej. Ewentualne zmiany ograniczeń mogą być stosowane dla schematu koncepcyjnego bez wpływu ani na schematy zewnętrzne, ani na aplikacje.

- (2) **Fizyczna niezależność danych** oznacza możliwość zmiany schematu wewnętrznego bez konieczności modyfikowania schematu koncepcyjnego. Oznacza to, że konieczność zmian nie dotyczy także schematów zewnętrznych. Zmiany w schematach wewnętrznych mogą się natomiast okazać niezbędne z uwagi na potrzebę reorganizacji niektórych plików fizycznych (np. przez stworzenie dodatkowych struktur dostępu) w celu poprawienia wydajności operacji odczytywania lub aktualizacji danych. Jeśli po wprowadzeniu zmian baza danych zawiera dokładnie takie same dane jak przedtem, modyfikowanie schematu koncepcyjnego nie powinno być konieczne. Przykładowo, dodanie ścieżki dostępu, która ma poprawić szybkość wyszukiwania żądanych rekordów z pliku KURS (patrz rysunek 1.2) według wartości elementów danych *Semestr* i *Rok*, nie powinno wymagać modyfikowania takich zapytań jak „wymień wszystkie kursy omawiane w semestrze jesiennym roku 1998”, chociaż dzięki nowej ścieżce dostępu wspomniane zapytanie będzie szybciej wykonywane przez system zarządzania bazą danych.

Fizyczna niezależność danych występuje w większości środowisk bazodanowych i plikowych, w których fizyczne szczegóły (takie jak dokładna lokalizacja danych na dysku oraz sprzętowe aspekty kodowania, rozmieszczania, kompresji, podziału i scalania rekordów) są ukryte przed użytkownikami. Aplikacje nie znają tych szczegółów. Trudniej jest osiągnąć logiczną niezależność danych, ponieważ ma ona umożliwiać wprowadzanie zmian struktury i ograniczeń bez wpływu na aplikacje, co jest dużo bardziej restrykcyjnym wymogiem.

Katalog każdego wielowarstwowego SZBD musi być stale rozszerzany w taki sposób, aby zawierał informacje niezbędne do odwzorowywania żądań i danych pomiędzy różnymi poziomami. Systemy zarządzania bazami danych wykorzystują do takiego odwzorowywania dodatkowe oprogramowanie, które odwołuje się do odpowiednich reguł w katalogach. Mamy więc do czynienia z niezależnością danych, ponieważ zmiana schematu na jednym poziomie nie wiąże się z koniecznością zmiany schematu na kolejnym, wyższym poziomie — niezbędne modyfikacje należy jedynie wprowadzić w regułach *odwzorowywania* na obu poziomach. Oznacza to, że aplikacje odwołujące się do schematu na wyższym poziomie w ogóle nie muszą być zmieniane.

## 2.3. Języki i interfejsy baz danych

W podrozdziale 1.4 omówiliśmy różne kategorie użytkowników końcowych wykorzystujących systemy zarządzania bazami danych. Takie systemy muszą oczywiście udostępniać odpowiednie języki i interfejsy dla każdej z tych kategorii. W tym podrozdziale omówimy rozmaite typy języków i interfejsów oferowanych poszczególnym kategoriom użytkowników przez systemy zarządzania bazami danych.

### 2.3.1. Języki systemów zarządzania bazami danych

Kiedy już zakończy się faza projektowania bazy danych i zostanie wybrany system zarządzania bazą danych, w którym nowa baza danych będzie implementowana, najważniejszym zadaniem jest opracowanie koncepcyjnego i wewnętrznego schematu tej bazy danych

oraz reguł odwzorowywania pomiędzy tymi poziomami. W wielu systemach zarządzania bazami danych, w których nie ma ścisłego rozgraniczenia pomiędzy utrzymywanymi poziomami, do definiowania obu schematów przez administratorów i projektantów baz danych wykorzystuje się jeden język, nazywany **językiem definicji danych** (ang. *Data Definition Language* — DDL). Takie systemy zarządzania bazami danych zawierają specjalne kompilatory języka DDL, które odpowiadają za przetwarzanie instrukcji opisujących konstrukcje schematu oraz za umieszczanie takiego przetworzonego opisu w katalogu SZBD.

W tych systemach zarządzania bazami danych, w których jest utrzymywany ścisły podział pomiędzy poziomem koncepcyjnym a poziomem wewnętrznym, język DDL wykorzystuje się wyłącznie do określania schematu koncepcyjnego. Do określania schematu wewnętrznego wykorzystywany jest wówczas inny język, nazywany **językiem definicji składowania** (ang. *Storage Definition Language* — SDL). Reguły odwzorowywania pomiędzy tymi dwoma schematami mogą być definiowane za pomocą odpowiednich instrukcji jednego z tych języków. Obecnie w większości relacyjnych SZBD *nie występuje konkretny język* używany jako SDL. Schemat wewnętrzny jest definiowany za pomocą funkcji, parametrów i specyfikacji dotyczących składowania plików. Administratorzy baz danych mogą dzięki temu wybierać sposoby indeksowania i odwzorowywania danych na mechanizmy ich przechowywania. Prawdziwa architektura trójwarstwowa będzie oczywiście wymagała jeszcze jednego języka — **języka definicji perspektyw** (ang. *View Definition Language* — VDL), za pomocą którego projektant lub administrator bazy danych będzie mógł określać perspektywy użytkownika wraz z ich odwzorowaniami do schematu koncepcyjnego — jednak w większości systemów zarządzania bazami danych *do definiowania schematu koncepcyjnego i zewnętrznego wykorzystuje się język DDL*. W relacyjnych SZBD jako VDL stosowany jest SQL, używany do definiowania **perspektyw** użytkowników lub aplikacji za pomocą wcześniej przygotowanych zapytań (patrz rozdziały 6. i 7.).

Kiedy już schematy bazy danych zostaną prawidłowo skompilowane, a sama baza danych zostanie wypełniona właściwymi danymi, użytkownicy muszą jeszcze otrzymać jakieś mechanizmy, które umożliwią im operowanie na gotowej bazie danych. Typowe działania w tym zakresie obejmują odczytywanie, wstawianie, usuwanie i modyfikowanie danych. W tym celu systemy zarządzania bazami danych udostępniają odpowiedni zbiór operacji lub tzw. **język manipulowania danymi** (ang. *Data Manipulation Language* — DML).

We współczesnych systemach zarządzania bazami danych wymienione przed chwilą rodzaje języków zwykle *nie są od siebie ściśle oddzielane* — zamiast tego, stosuje się jeden uniwersalny, zintegrowany język, który zawiera konstrukcje niezbędne do definiowania schematu koncepcyjnego, do definiowania perspektyw oraz do operowania na danych. Jedynie język definicji składowania jest zwykle wyodrębniany, ponieważ wykorzystuje się go do definiowania fizycznych struktur przechowywania danych głównie po to, aby poprawić wydajność systemu bazy danych, za co zwykle odpowiada administrator bazy danych. Typowym przykładem wspomnianego wszechstronnego języka bazy danych jest stosowany w relacyjnych bazach danych język SQL (patrz rozdziały 6. i 7.), który nie tylko stanowi połączenie języków DDL, VDL i DML, ale także obejmuje instrukcje dla specyfikacji ograniczeń, modyfikowania schematu i wielu innych zadań. Wczesne wersje SQL-a zawierały język SDL, jednak w pewnym momencie zarzucono pomysł umieszczania tego języka w tej formie w SQL-u i pozostawiono wyłącznie obsługę poziomu koncepcyjnego i zewnętrznego.

Istnieją dwa główne typy języków manipulowania danymi (DML). **Wysokopoziomowy (nieproceduralny)** język DML może być wykorzystywany do stosunkowo zwięzłego definiowania skomplikowanych operacji na bazie danych. Wiele systemów zarządzania bazami danych obsługuje zarówno te konstrukcje wysokopoziomowego języka DML, które zostaną wpisane przez użytkownika w odpowiednim terminalu, jak i te, które programista osadzi w kodzie uniwersalnego języka programowania. W tym drugim przypadku konstrukcje języka DML muszą być tak oznaczone w kodzie programu, aby prekompilator mógł je wyodrębnić i przekazać do przetworzenia przez system zarządzania bazą danych. **Niskopoziomowy (proceduralny)** język DML *musi* być osadzany w kodzie napisanym w uniwersalnych językach programowania. Język DML tego typu jest zwykle wykorzystywany do odczytywania z bazy danych pojedynczych rekordów lub obiektów i ich odrębnego przetwarzania. Oznacza to, że odczytywanie i przetwarzanie poszczególnych rekordów należących do całego zbioru wymaga użycia konstrukcji (np. pętli) wykorzystywanego języka programowania. Z tego powodu niskopoziomowe języki DML są często nazywane językami typu **record-at-a-time**. Wysokopoziomowe języki DML, w tym SQL, mogą określać i odczytywać wiele rekordów za pomocą pojedynczej konstrukcji i w związku z tym są nazywane językami typu **set-at-a-time** lub **set-oriented**. Zapytanie zdefiniowane w wysokopoziomowym języku DML zwykle określa tylko to, *które* dane mają zostać odczytane, nie określa natomiast *sposobu* ich odczytania — takie języki są w związku z tym często nazywane językami **deklaratywnymi**.

W sytuacji, gdy polecenia języka DML (niezależnie od tego, czy jest to język wysokopoziomowy, czy niskopoziomowy) są osadzane w uniwersalnym języku programowania, język ten jest nazywany **językiem nadrzędnym**, natomiast język DML jest nazywany **podjęzykiem danych**<sup>8</sup>. Z drugiej strony, wysokopoziomowy język DML wykorzystywany przez użytkownika w sposób odrębny jest nazywany **językiem zapytań**. Generalnie, zarówno polecenia odczytywania, jak i polecenia aktualizacji wysokopoziomowego języka DML mogą być wykorzystywane bezpośrednio przez użytkownika, zatem są traktowane jak części języka zapytań<sup>9</sup>.

Dorywczy użytkownicy końcowi wykorzystują zwykle wysokopoziomowe języki zapytań do określania swoich żądań, natomiast programiści korzystają najczęściej z języków DML w formie instrukcji osadzonych. Niedoświadczeni użytkownicy parametryczni wykorzystują zwykle specjalnie przygotowane **przyjazne interfejsy użytkownika** bazy danych — takie interfejsy mogą również być stosowane przez użytkowników doraźnych i inne osoby, które nie chcą się uczyć szczegółów związanych z używaniem wysokopoziomowych języków zapytań. Omówimy te typy interfejsów w kolejnym punkcie.

---

<sup>8</sup> W obiektowych bazach danych język nadrzędny i podjęzyk danych tworzą zwykle jeden zintegrowany język — przykładowo, język programowania C++ z pewnymi rozszerzeniami obsługuje funkcje bazy danych. Niektóre relacyjne systemy zarządzania bazami danych dodatkowo udostępniają zintegrowane języki — przykładowo, system Oracle oferuje język PL/SQL.

<sup>9</sup> Zgodnie z ogólnie przyjętym znaczeniem słowa *zapytanie* należałoby go używać wyłącznie w odniesieniu do operacji odczytywania danych, a nie do aktualizacji.

## 2.3.2. Interfejsy systemów zarządzania bazami danych

Poniżej wymieniono przyjazne interfejsy użytkownika oferowane przez systemy zarządzania bazami danych.

**Interfejsy oparte na menu dla klientów WWW lub przeglądarek.** Interfejsy tego typu prezentują użytkownikom listy dostępnych opcji, nazywane **menu**, które przeprowadzają użytkowników przez proces formułowania żądań. Wykorzystywane w ten sposób menu eliminują konieczność zapamiętywania konkretnych poleceń i składni języka zapytań; zamiast tego zapytanie jest tworzone krok po kroku przez wybieranie kolejnych opcji z menu wyświetlanych przez oprogramowanie systemu bazy danych. W przypadku **interfejsów internetowych** (opartych na stronach WWW) bardzo popularne są tzw. menu rozwijane. Takie menu są także popularne w **interfejsach przeglądania**, które umożliwiają użytkownikom przeszukiwanie zawartości bazy danych (choćby w celach orientacyjnych).

**Aplikacje dla urządzeń mobilnych.** Takie interfejsy umożliwiają użytkownikom urządzeń mobilnych dostęp do danych. Przykładowo, banki, systemy rezerwacji i firmy ubezpieczeniowe udostępniają aplikacje umożliwiające użytkownikom dostęp do danych za pomocą telefonów komórkowych i innych urządzeń mobilnych. Takie aplikacje mają wbudowane zaprogramowane interfejsy, które zwykle pozwalają użytkownikom zalogować się za pomocą nazwy i hasła, a następnie udostępniają ograniczone menu z opcjami mobilnego dostępu do danych, a także z operacjami takimi jak opłacanie rachunków (w bankach) lub dokonywanie rezerwacji (w serwisach internetowych obsługujących rezerwacje).

**Interfejsy oparte na formularzach.** Interfejsy oparte na formularzach wyświetlają każdemu użytkownikowi formularz. Użytkownicy mogą wypełniać wszystkie pola takiego **formularza**, aby wstawić nowe dane, lub tylko niektóre z nich, aby system zarządzania bazą danych zwrócił właściwe dane wypełniające pozostałe pola. Formularze są zwykle projektowane i programowane z myślą o niedoświadczonych użytkownikach i pełnią rolę interfejsu dla wykonywanych przez nich podstawowych transakcji. Wiele systemów zarządzania bazami danych oferuje specjalne **języki definiowania formularzy**, ułatwiające programistom tworzenie tego typu struktur. SQL\*Forms to oparty na formularzach język, w którym zapytania są tworzone za pomocą formularza projektowanego na podstawie schematu relacyjnej bazy danych. Oracle Forms to komponent pakietu produktów firmy Oracle udostępniający bogaty zestaw funkcji do projektowania i budowania aplikacji z użyciem formularzy. Niektóre systemy udostępniają narzędzia, za pomocą których użytkownicy końcowi mogą interaktywnie budować proste formularze w środowisku graficznym.

**Graficzne interfejsy użytkownika.** Graficzny interfejs użytkownika (ang. *Graphical User Interface* — *GUI*) przeważnie wyświetla użytkownikowi schemat bazy danych w formie graficznej. Użytkownik może wówczas tworzyć zapytania przez operowanie diagramem. W wielu przypadkach graficzne interfejsy użytkownika opierają się zarówno na menu, jak i na formularzach.

**Interfejsy języka naturalnego.** Interfejsy tego typu akceptują żądania pisane w języku naturalnym (najczęściej angielskim) i próbują je *zrozumieć* i odpowiednio przetworzyć. Interfejsy języka naturalnego zawierają zwykle nie tylko własne *schematy*, które są zbliżone do koncepcyjnego schematu bazy danych, ale także słowniki ważnych słów. Podczas interpretowania otrzymanego żądania interfejs języka naturalnego odwołuje się zarówno do słów zdefiniowanych w tym schemacie, jak i do zbioru standardowych słów zawartych w jego słowniku. Jeśli proces interpretacji żądania zakończy się powodzeniem, interfejs języka naturalnego generuje odpowiednie zapytanie wysokopoziomowe i przekazuje je do przetworzenia do systemu zarządzania bazą danych; w przeciwnym przypadku nawiązywany jest dialog z użytkownikiem, mający na celu uszczegółowienie żądania.

**Przeszukiwanie baz na podstawie słów kluczowych.** Te interfejsy przypominają nieco wyszukiwarki internetowe, które przyjmują słowa w języku naturalnym (np. angielskim lub polskim) i dopasowują je do dokumentów z konkretnej witryny (w wyszukiwarkach lokalnych) lub stron WWW z całego internetu (w wyszukiwarkach takich jak Google lub Ask). Stosuje się tu predefiniowane indeksy słów i funkcje rankingowe do pobierania i wyświetlania dokumentów w kolejności wyznaczonej przez poziom dopasowania. W ustrukturyzowanych bazach relacyjnych tego rodzaju dające pełną swobodę interfejsy tekstowe do tworzenia zapytań nie są na razie częste, choć niedawno powstała dziedzina badań dotycząca **zapytań opartych na słowach kluczowych** w kontekście baz relacyjnych.

**Głosowe przekazywanie danych wejściowych i wyjściowych.** Coraz popularniejsze staje się ograniczone wykorzystanie mowy do podawania zapytań oraz zwracania odpowiedzi lub wyników. Aplikacje z ograniczonym słownikiem (np.: informacji z książki telefonicznej, o odlotach i przylotach czy o kontaktach do kart kredytowych) pozwalają na wejściu i na wyjściu stosować mowę, aby umożliwić klientom dostęp do danych. Wejściowe dane głosowe są wykrywane na podstawie biblioteki predefiniowanych słów i używane do określenia parametrów przekazywanych do zapytania. Na wyjściu wykonywana jest podobna konwersja tekstu lub liczb na mowę.

**Interfejsy dla użytkowników parametrycznych.** Użytkownicy parametryczni, tacy jak kasjerzy bankowi, często wielokrotnie wykonują stosunkowo niewielki i stały zbiór operacji. Przykładowo, kasjer może za pomocą jednego przycisku funkcyjnego uruchamiać rutynowe i powtarzalne transakcje, takie jak wpłaty lub wypłaty, a także odpowiadać na zapytania o stan konta. Analitycy systemowi i programiści odpowiednio projektują i implementują specjalny interfejs dla każdej znanej klasy takich niedoświadczonych użytkowników. Tworzone w ten sposób interfejsy obsługują zwykle niewielki zbiór skróconych poleceń, ponieważ celem jest zminimalizowanie liczby naciskanych klawiszy niezbędnej do wywołania poszczególnych żądań.

**Interfejsy dla administratorów baz danych.** Większość systemów baz danych zawiera uprzywilejowane polecenia, które mogą być wykorzystywane wyłącznie przez administratorów. Należą do nich polecenia tworzenia kont użytkowników, ustawiania parametrów systemowych, przypisywania uprawnień do kont użytkowników, modyfikowania schematu oraz reorganizowania struktur przechowywania bazy danych.

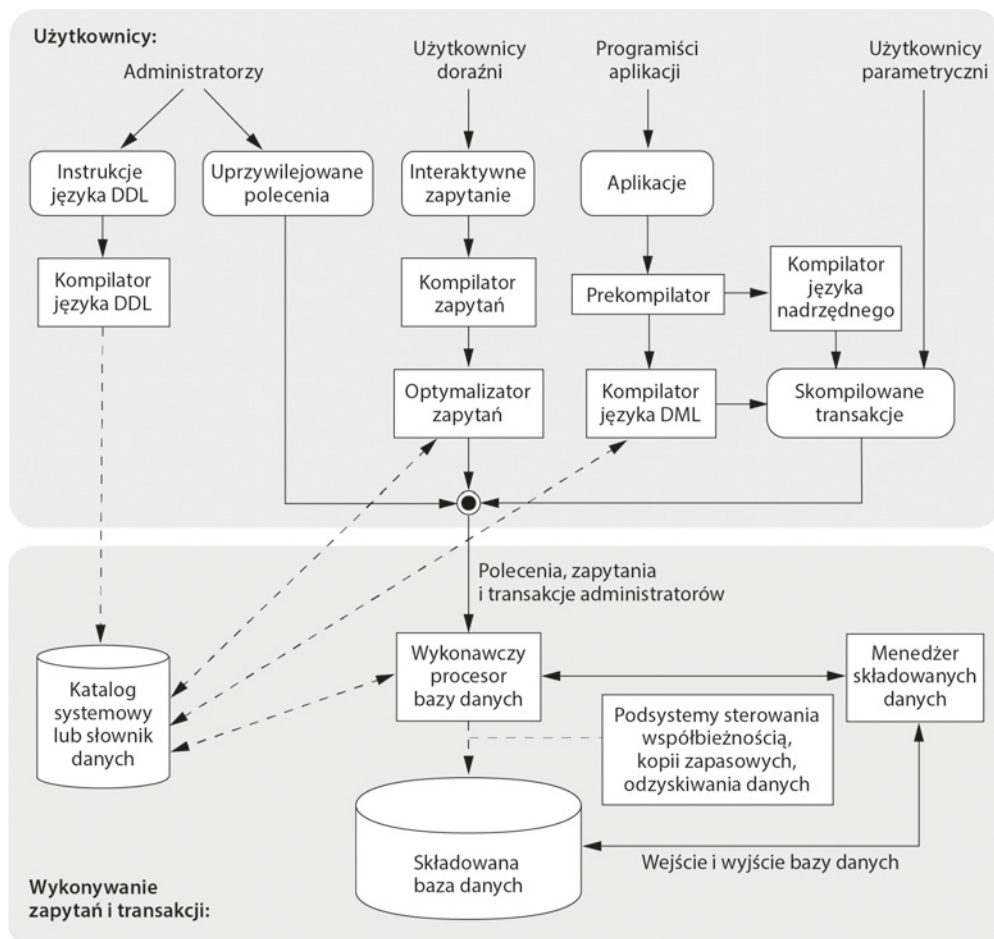


## 2.4. Środowisko systemu bazy danych

System zarządzania bazą danych jest skomplikowanym systemem oprogramowania. W tym podrozdziale omówimy typy składników programowych, które połączone tworzą SZBD, wraz z rodzajami oprogramowania systemu komputerowego, z którymi SZBD współpracuje.

### 2.4.1. Moduły składające się na system zarządzania bazą danych

Na rysunku 2.3 przedstawiono w uproszczonej formie typowe składniki systemu zarządzania bazą danych. Rysunek jest podzielony na dwie części. Górna przedstawia różnych użytkowników środowiska bazy danych i ich interfejsy. W dolnej widoczne są wewnętrzne moduły SZBD odpowiedzialne za składowanie danych i przetwarzanie transakcji.



RYSUNEK 2.3. Moduły składające się na system zarządzania bazą danych i związane z nimi interakcje



Sama baza danych i katalog systemu zarządzania bazą danych są zwykle przechowywane na dysku. Dostęp do dysku jest kontrolowany głównie przez **system operacyjny** (ang. *Operating System* — *OS*), który ustala kolejność wykonywania operacji odczytu-zapisu. Liczne systemy zarządzania bazami danych zawierają własne **moduły zarządzania buforami**, służące do szeregowania operacji odczytu i zapisu na dysku. Te moduły są używane, ponieważ zarządzanie zawartością bufora ma istotny wpływ na wydajność. Przyspieszenie odczytów i zapisów znacząco zwiększa wydajność. Znajdujący się na wyższym poziomie i należący do systemu zarządzania bazą danych moduł **menadżera składowanych danych** kontroluje dostęp do informacji umieszczonych na dysku niezależnie od tego, czy są to informacje znajdujące się w bazie danych, czy w katalogu.

Przyjrzyj się najpierw górnej części rysunku 2.3. Widoczne są tam interfejsy dla administratorów bazy danych, użytkowników doraźnych (którzy formułują zapytania za pomocą interaktywnych interfejsów), programistów aplikacji (tworzących programy za pomocą nadrzędnych języków programowania) i użytkowników parametrycznych (wprowadzających dane przez podawanie parametrów predefiniowanych transakcji). Administratorzy bazy danych definiują i dostrajają bazę, wprowadzając zmiany w jej definicji za pomocą języka DDL i poleceń wymagających wysokich uprawnień.

Kompilator języka DDL przetwarza zdefiniowane w tym języku specyfikacje schematów i zapisuje opisy przetworzonych schematów (metadane) w katalogu systemu zarządzania bazą danych. Katalog zawiera takie informacje jak nazwy i rozmiary poszczególnych plików, nazwy i typy danych poszczególnych elementów danych, szczegóły przechowywania każdego z plików, informacje o regułach odwzorowywania pomiędzy schematami, specyfikacje ograniczeń oraz wiele dodatkowych rodzajów informacji, które są niezbędne do pracy rozmaitych modułów systemu zarządzania bazą danych.

Użytkownicy doraźni i osoby, które sporadycznie potrzebują danych z bazy, komunikują się z nią za pomocą interfejsu do zgłaszania **interaktywnych zapytań** (patrz rysunek 2.3). *Nie przedstawiliśmy tu bezpośrednio* żadnych opartych na menu, formularzu lub rozwiązaniach mobilnych interakcji, które zwykle są używane do automatycznego generowania interaktywnych zapytań lub dostępu do transakcji zapuszkowanych. Takie zapytania są parsowane i sprawdzane pod kątem poprawności składni oraz nazw plików i elementów danych przez **kompilator zapytań**, który kompiluje je do wewnętrznej postaci. Zapytania w wewnętrznej postaci są optymalizowane (patrz rozdziały 18. i 19.). **Optymalizator zapytań** odpowiada m.in. za zmianę układu operacji, eliminowanie nadmiarowości i wykorzystanie wydajnych algorytmów wyszukiwania w trakcie przetwarzania zapytania. Optymalizator sprawdza w katalogu systemu statystyczne i fizyczne informacje o składowanych danych oraz generuje kod wykonywalny, który przeprowadza operacje potrzebne do obsługi zapytania i kieruje wywołania do procesora wykonawczego.

Programiści aplikacji piszą programy w językach nadrzędnych takich jak Java, C i C++. Te programy są przekazywane do prekompilatora. **Prekompilator** wyodrębnia polecenia języka DML z kodu aplikacji napisanego w języku nadrzędnym. Takie polecenia są następnie przekazywane do kompilatora języka DML, który kompiluje je do postaci kodu obiektów wykorzystywanych w operacjach dostępu do bazy danych. Reszta programu jest przesyłana do kompilatora języka nadrzędnego. Kody obiektów dla poleceń języka DML pozostają jednak połączone ze skompilowanym programem języka nadrzędnego i tworzą tzw. zapuszkowane transakcje, których kod zawiera wywołania operacji wy-

konawczego procesora bazy danych. Do pisania programów bazodanowych coraz częściej stosuje się języki skryptowe takie jak PHP i Python. Zapuszkowane transakcje są wykonywane wielokrotnie przez użytkowników parametrycznych za pomocą komputerów osobistych lub aplikacji mobilnych. Tacy użytkownicy określają tylko parametry transakcji. Każde wywołanie jest wtedy uznawane za odrębną transakcję. Przykładem jest tu transakcja płatności w banku, gdzie jako parametry można podać numer konta, odbiorcę i kwotę.

W dolnej części rysunku 2.3 **wykonawczy procesor bazy danych** uruchamia (1) uprzywilejowane polecenia, (2) wykonywalne plany zapytań i (3) zapuszkowane transakcje z parametrami wykonawczymi. Ten procesor korzysta z **katalogu systemu** i może aktualizować zapisane w nim statystyki. Używa też **menedżera danych składowanych**, który z kolei korzysta z podstawowych usług systemu operacyjnego do wykonywania niskopoziomowych operacji wejścia-wyjścia (odczytu i zapisu) z wykorzystaniem dysku i pamięci głównej. Wykonawczy procesor bazy obsługuje też inne aspekty przenoszenia danych, takie jak zarządzanie buforami w pamięci głównej. Niektóre SZBD mają własny moduł zarządzania buforami, natomiast inne korzystają w tym obszarze z systemu operacyjnego. Na rysunku systemy **sterowania współbieżnego** oraz **tworzenia kopii bezpieczeństwa i odtwarzania bazy** są przedstawione jako odrębny moduł. Na potrzeby transakcji te komponenty są integrowane z działaniem wykonawczego procesora bazy danych.

Popularnym rozwiązaniem jest wykorzystywanie **programu klienta**, który uzyskuje dostęp do systemu zarządzania bazą danych, działając na innym komputerze niż ten, na którym znajduje się baza danych. Pierwszy komputer jest wówczas nazywanym **komputerem klienta**, natomiast drugi komputer nosi nazwę **serwera bazy danych**. W niektórych przypadkach klient wykorzystuje dodatkowo komputer pośredniczący, nazywany **serwerem aplikacji**, który w imieniu klienta uzyskuje bezpośredni dostęp do serwera bazy danych. Omówimy to zagadnienie bardziej szczegółowo w podrozdziale 2.5.

Rysunek 2.3 nie ma ilustrować żadnego konkretnego systemu zarządzania bazą danych, a jedynie przedstawiać typowe związki pomiędzy modułami SZBD. System zarządzania bazą danych wykorzystuje usługi systemu operacyjnego w momencie uzyskiwania dostępu do informacji zapisanych na dysku — w bazie danych lub w katalogu. Jeśli dany system komputerowy jest współużytkowany przez wielu użytkowników, system operacyjny odpowiada za właściwe zaplanowanie kolejności obsługi otrzymywanych żądań dostępu oraz prawidłowe zarządzanie czasem przyznawanym procesowi systemu zarządzania bazą danych i innym działającym procesom. Z drugiej strony, jeśli dany system komputerowy jest przeznaczony przede wszystkim do zapewnienia odpowiedniego środowiska działania dla serwera bazy danych, system zarządzania bazą danych będzie kontrolował buforowanie stron dyskowych w głównej pamięci. System zarządzania bazą danych odpowiada także za dostarczanie niezbędnych interfejsów nie tylko dla kompilatorów uniwersalnych, nadrzędnych języków programowania, ale także dla serwerów aplikacji i programów klienckich działających na innych komputerach i komunikujących się za pośrednictwem interfejsu sieciowego.

## 2.4.2. Narzędzia systemu bazy danych

Poza opisanymi przed chwilą modułami programowymi większość systemów zarządzania bazami danych dodatkowo udostępnia **narzędzia baz danych**, które ułatwiają ich administratorom zarządzanie systemami baz danych. Do najczęściej stosowanych typów i funkcji oferowanych przez narzędzia tego typu należą:

- **Wczytywanie:** Narzędzia wczytujące są wykorzystywane do kopiowania zawartości zewnętrznych plików z danymi (np. plików tekstowych lub plików sekwencyjnych) do bazy danych. Narzędzia tego typu otrzymują na wejściu bieżący (źródłowy) format pliku z danymi oraz oczekiwaną (docelową) strukturę pliku bazy danych, i na tej podstawie automatycznie formatują i zapisują odpowiednie informacje w bazie danych. Wraz z rosnącą popularnością systemów zarządzania bazami danych, konieczność przenoszenia danych pomiędzy różnymi pakietami tego typu staje się w wielu organizacjach codziennością. Niektórzy producenci oferują nawet **narzędziami konwersji** generujące — w oparciu o opisy (schematy wewnętrzne) istniejącego formatu składowania informacji i docelowego formatu przechowywania informacji w bazie danych — odpowiednie programy wczytujące.
- **Tworzenie kopii zapasowej:** Narzędzia tego typu tworzą kopię zapasową bazy danych (takie działania polegają zwykle na wykonaniu rzutu całej bazy danych i zapisaniu go na taśmie lub innym nośniku). Kopia zapasowa może być następnie wykorzystana do przywrócenia bazy danych w przypadku katastrofalnej awarii dysku. Często stosuje się technikę tworzenia przyrostowych kopii zapasowych, w której rejestrowane są wyłącznie zmiany, jakie miały miejsce od ostatniej operacji utworzenia takiej kopii. Wykonywanie przyrostowej kopii zapasowej jest co prawda bardziej skomplikowane, ale pozwala oszczędzić przestrzeń na wykorzystywanym nośniku.
- **Reorganizacja plików:** Narzędzia tego typu mogą być wykorzystywane do reorganizowania plików baz danych z wykorzystaniem innej struktury i nowych ścieżek dostępu, najczęściej z myślą o poprawie wydajności.
- **Monitorowanie wydajności:** Takie narzędzia monitorują obciążenie bazy danych i na tej podstawie generują odpowiednie dane statystyczne dla administratora tej bazy. Administrator bazy danych może wykorzystać te statystyki podczas podejmowania decyzji np. w kwestii ewentualnej reorganizacji plików albo dodanie lub usunięcia indeksów celem poprawy wydajności.

Systemy zarządzania bazami danych mogą także oferować funkcje sortowania plików, obsługi kompresji danych, monitorowania operacji dostępu wykonywanych przez użytkowników, obsługi interfejsów sieciowych oraz wykonywania innych przydatnych zadań.

### 2.4.3. Narzędzia, środowiska aplikacji oraz mechanizmy komunikacji

Projektanci, użytkownicy i SZBD mogą korzystać także z innych przydatnych narzędzi. W fazie projektowania systemów baz danych można wykorzystywać narzędzia typu CASE<sup>10</sup>. Do narzędzi, które mogą być przydatne w wielkich organizacjach, należy także rozszerzony **system słownika danych**, nazywany także **systemem repozytorium danych**. Poza danymi katalogowymi na temat schematów i ograniczeń słownik danych może zawierać także inne informacje, takie jak decyzje projektowe, standardy wykorzystywania bazy danych, opisy aplikacji i dane o użytkownikach. Systemy tego typu są także nazywane **repozytoriami danych**. W razie potrzeby użytkownicy lub administrator bazy danych mogą mieć *bezpośredni* dostęp do informacji zawartych w takim repozytorium. Rozwiązania obsługujące słowniki danych przypominają co prawda katalogi systemów zarządzania bazami danych, jednak zawierają więcej różnych informacji i są udostępniane przede wszystkim użytkownikom, nie oprogramowaniu SZBD (jak w przypadku katalogów).

**Środowiska wytwarzania aplikacji**, takie jak systemy PowerBuilder (firmy Sybase) czy JBuilder (Borland), są dość popularne. Takie systemy oferują środowisko ułatwiające szybkie wytwarzanie aplikacji baz danych, a także upraszczają wiele aspektów opracowywania systemów baz danych, w tym projektowania baz danych, tworzenia graficznych interfejsów użytkownika, przygotowywania zapytań i operacji aktualizujących, oraz budowania aplikacji.

Systemy zarządzania bazami danych potrzebują także interfejsów do **oprogramowania komunikacyjnego**, które ma umożliwiać zdalnym użytkownikom korzystanie z systemu bazy danych i dostęp do zawartych tam danych za pośrednictwem komputerowych terminali, stacji roboczych lub ich lokalnych komputerów osobistych. Komputery takich użytkowników są połączone z serwerem bazy danych za pomocą odpowiedniego sprzętu komunikacyjnego — routerów internetowych, linii telefonicznych, sieci rozległych, sieci lokalnych lub urządzeń komunikacji satelitarnej. Wiele komercyjnych systemów baz danych oferuje specjalne pakiety komunikacyjne, które współpracują z systemami zarządzania bazami danych. Systemy tego typu zintegrowane z systemami komunikacyjnymi są nazywane systemami **DB/DC**. Niektóre rozproszone systemy zarządzania bazami danych są dodatkowo fizycznie instalowane na wielu komputerach. W takim przypadku do łączenia komputerów wykorzystuje się sieci komunikacyjne. Najczęściej są to **sieci lokalne** (ang. *Local Area Network* — LAN), jednak mogą to być także inne typy sieci komputerowych.

---

<sup>10</sup> Chociaż CASE oznacza wspomaganą komputerowo inżynierię oprogramowania (ang. *Computer-Aided Software Engineer*), wiele narzędzi typu CASE jest wykorzystywanych przede wszystkim do projektowania baz danych.

## 2.5. Architektury systemów zarządzania bazami danych — scentralizowane i typu klient-serwer

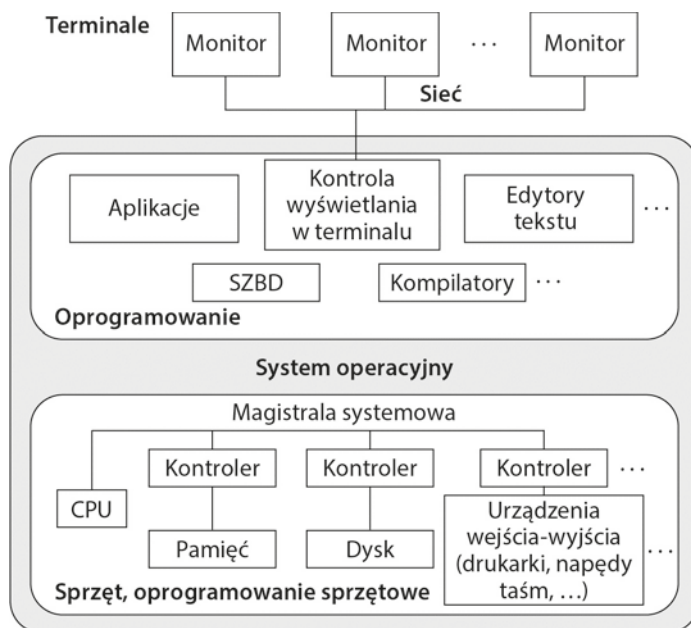
### 2.5.1. Scentralizowane architektury systemów zarządzania bazami danych

Architektury systemów zarządzania bazami danych ewoluowały zgodnie z ogólnymi trendami panującymi w świecie systemów komputerowych. Wcześniejsze architektury opierały się na centralnych komputerach, które obsługiwały przetwarzanie dla wszystkich funkcji systemu, włącznie z udostępnianiem aplikacji użytkowników i programów pełniących rolę interfejsów użytkowników, a także całą funkcjonalność systemu zarządzania bazą danych. Stosowanie takich rozwiązań wynikało z faktu, że większość użytkowników uzyskiwała dostęp do tych systemów za pośrednictwem komputerowych terminali, które nie oferowały wystarczającej mocy obliczeniowej, i które w związku z tym pełniły wyłącznie rolę sprzętu wyświetlającego otrzymywane dane. Wszystkie operacje związane z przetwarzaniem danych były wówczas zdalnie wykonywane w centralnym systemie komputerowym, a terminale odpowiadały jedynie za wyświetlanie otrzymanych z tego komputera informacji i instrukcji sterujących — terminale były połączone z centralnym komputerem za pośrednictwem rozmaitych typów sieci komunikacyjnych.

Wraz z postępującym spadkiem cen sprzętu komputerowego większość użytkowników zastąpiła swoje terminale komputerami osobistymi (ang. *Personal Computer* — *PC*) oraz stacjami roboczymi, a później także urządzeniami mobilnymi. Systemy baz danych początkowo wykorzystywały te komputery w taki sam sposób, w jaki wcześniej używały terminali wyświetlających, zatem nadal mieliśmy do czynienia ze **scentralizowanymi** systemami zarządzania bazami danych, w których obsługa wszystkich funkcji, wykonywanie programów aplikacji oraz przetwarzanie interfejsu użytkownika odbywało się na jednym komputerze. Na rysunku 2.4 przedstawiono fizyczne składniki architektury scentralizowanej. Systemy zarządzania bazami danych z czasem zaczęły wykorzystywać moce przetwarzania dostępne po stronie użytkownika, co doprowadziło ostatecznie do opracowania architektury systemów zarządzania bazami danych typu klient-serwer.

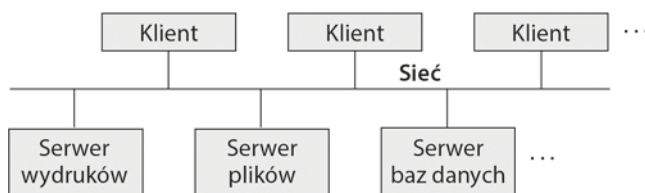
### 2.5.2. Podstawowe architektury typu klient-serwer

W pierwszej kolejności omówimy ogólną architekturę typu klient-serwer, dopiero potem przystąpimy do analizy jej zastosowań w systemach zarządzania bazami danych. **Architektura typu klient-serwer** została opracowana z myślą o środowiskach komputerowych z dużą liczbą komputerów osobistych, stacji roboczych, serwerów plików, drukarek, serwerów baz danych, serwerów WWW, serwerów poczty elektronicznej i innego sprzętu połączonego ze sobą za pomocą sieci komputerowej. Zasadniczym elementem tej koncepcji jest definiowanie **wyspecjalizowanych serwerów** realizujących tylko określone funkcje. Przykładowo, istnieje możliwość połączenia wielu komputerów osobistych lub niewielkich stacji roboczych w roli klientów **serwera plików**, który utrzymuje pliki wykorzystywane na komputerach klienckich. W tym samym środowisku możemy przydzielić innemu komputerowi rolę **serwera wydruków** i połączyć go z różnymi drukarkami — wszystkie generowane przez klientów żądania drukowania będą wówczas przekazywane właśnie do



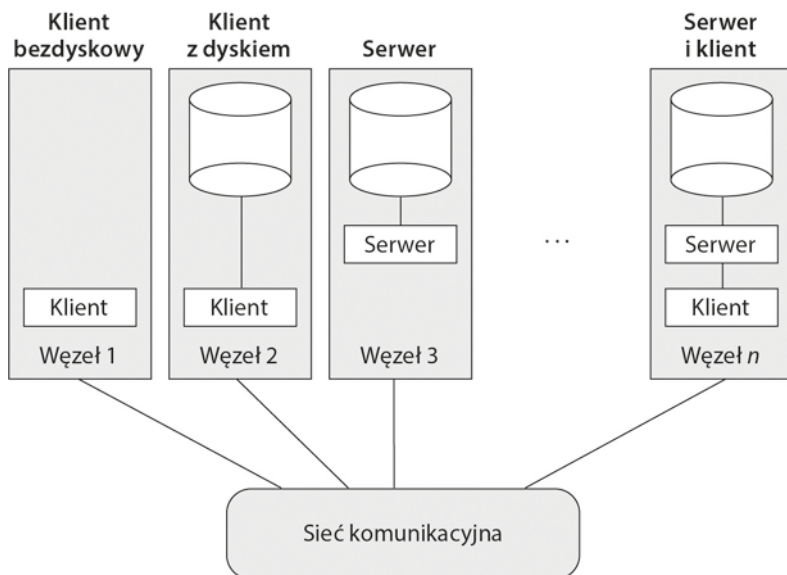
RYSUNEK 2.4. Fizycznie scentralizowana architektura

tego komputera. Także **serwery WWW** lub **serwery poczty elektronicznej** należą do kategorii serwerów wyspecjalizowanych. W ten sposób zasoby oferowane przez wyspecjalizowane serwery mogą być jednocześnie udostępniane wielu **komputerom klienckim**. Takie komputery oferują swoim użytkownikom nie tylko odpowiednie interfejsy umożliwiające wygodne i efektywne wykorzystywanie funkcjonalności wyspecjalizowanych serwerów, ale także lokalną moc przetwarzania umożliwiającą pracę z lokalnymi aplikacjami. Koncepcję stosowania wyspecjalizowanych serwerów można przenieść także na grunt samego oprogramowania, gdzie wyspecjalizowane pakiety — takie jak oprogramowanie typu *CAD* (ang. *Computer-Aided Design*) — może się znajdować na określonych serwerach i być udostępniane jednocześnie wielu klientom. Na rysunku 2.5 przedstawiono architekturę typu klient-serwer na poziomie logicznym, natomiast na rysunku 2.6 zademonstrowano uproszczony diagram pokazujący, jak wyglądałaby fizyczna architektura takiego środowiska. Niektóre komputery pełniłyby w takim systemie jedynie rolę urządzeń klienckich (przykładowo, może to dotyczyć urządzeń mobilnych albo też komputerów osobistych lub stacji roboczych z zainstalowanym wyłącznie oprogramowaniem klienckim). Inne komputery w tak skonstruowanym środowisku byłyby dedykowanymi serwerami, a jeszcze inne oferowałyby zarówno funkcjonalność klienta, jak i serwera.



RYSUNEK 2.5. Logiczna dwuwarstwowa architektura typu klient-serwer





RYSUNEK 2.6. Fizyczna dwuwarstwowa architektura typu klient-serwer

Koncepcja architektury typu klient-serwer przewiduje istnienie całej infrastruktury sieciowej obejmującej wiele komputerów osobistych, stacji roboczych i urządzeń przenośnych oraz znacznie mniej liczną grupę serwerów połączonych ze sobą za pomocą sieci bezprzewodowej, lokalnej lub innego typu. **Klient** w takiej infrastrukturze ma zwykle postać takiego komputera użytkownika, który oferuje pewne możliwości zarówno w zakresie obsługi interfejsu użytkownika, jak i w kwestii lokalnego przetwarzania. Kiedy klient zażąda dostępu do dodatkowych funkcji lub zasobów (np. do bazy), które nie są utrzymywane na jego lokalnym komputerze, komputer kliencki nawiąże połączenie z odpowiednim serwerem oferującym środki niezbędne do zrealizowania tego żądania. **Serwer** jest systemem obejmującym sprzęt i oprogramowanie, które pozwalają udostępniać komputerom klienckim takie usługi jak dostęp do plików, drukowanie, archiwizowanie czy dostęp do bazy danych. Niektóre komputery mogą mieć zainstalowane wyłącznie oprogramowanie klienckie, inne wyłącznie oprogramowanie serwera, jeszcze inne mogą natomiast zawierać zarówno oprogramowanie klienta, jak i oprogramowanie serwera (patrz rysunek 2.6). Znacznie bardziej popularnym rozwiązaniem jest jednak wykorzystywanie oprogramowania klienta i serwera na różnych komputerach. Na podstawie architektury klient-serwer stworzono dwa główne typy architektury systemów zarządzania bazami danych: **dwuwarstwową** i **trójwarstwową**<sup>11</sup>. Omówimy je w kolejnych punktach.

<sup>11</sup> Istnieje jeszcze wiele innych odmian architektury klient-serwer. W tym miejscu omówimy tylko wspomnianą parę podstawowych architektury tego typu.



### 2.5.3. Dwuwarstwowe architektury typu klient-serwer dla systemów zarządzania bazami danych

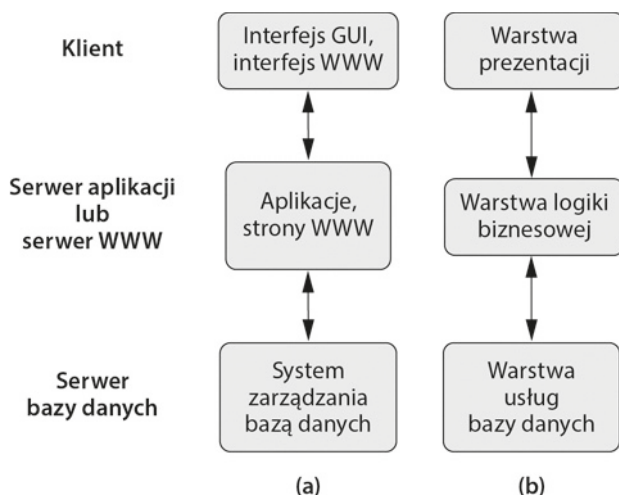
W relacyjnych systemach zarządzania bazami danych, których większość była początkowo systemami scentralizowanymi, do warstwy klienta w pierwszej kolejności przeniesiono takie elementy jak interfejs użytkownika oraz aplikacje. Ponieważ SQL (patrz rozdziały 6. i 7.) z czasem zyskał pozycję standardowego języka relacyjnych systemów zarządzania bazami danych, takie posunięcie stworzyło logiczny punkt podziału pomiędzy klientem a serwerem. Oznacza to, że obsługa zapytań i transakcji pozostały po stronie serwera. W związku z tym w rozwiązaniach zbudowanych zgodnie z tą architekturą serwer jest często nazywany **serwerem zapytań** lub **serwerem transakcji**. W relacyjnych systemach zarządzania bazami danych taki serwer jest równie często nazywany **serwerem SQL**.

W takich architekturach programy interfejsu użytkownika i aplikacje mogą działać po stronie klienta. Kiedy okaże się, że niezbędny jest dostęp do systemu zarządzania bazą danych, odpowiedni program nawiąże połączenie z takim systemem (działającym po stronie serwera); kiedy odpowiednie połączenie zostanie utworzone, program klienta będzie mógł się komunikować z systemem zarządzania bazą danych. Standard nazywany **otwartym łączem baz danych** (ang. *Open Database Connectivity* — *ODBC*) oferuje **interfejs programowy aplikacji** (ang. *Application Programming Interface* — *API*), który umożliwia programom działającym po stronie klienta wywoływanie systemu zarządzania bazą danych — jedynym wymaganiem jest zainstalowanie na komputerach klienta i serwera odpowiedniego oprogramowania. Większość producentów systemów zarządzania bazami danych oferuje w swoich pakietach specjalne sterowniki ODBC. Oznacza to, że pojedynczy program klienta może się łączyć z wieloma relacyjnymi systemami zarządzania bazami danych i wysyłać — za pośrednictwem interfejsu API standardu ODBC — zapytania i żądania transakcji, które będą następnie przetwarzane po stronie serwerów baz danych. Uzyskane w ten sposób wyniki zapytań zostaną odesłane do programu klienta, który może je dodatkowo przetworzyć lub od razu zaprezentować użytkownikowi. Istnieje także powiązany standard (nazwany **JDBC**) dla języka programowania Java. Zastosowanie takiego rozwiązania umożliwia dostęp do systemów zarządzania bazami danych z poziomu programu klienta napisanego w języku Java za pośrednictwem standardowego interfejsu.

Opisane w tym punkcie architektury są nazywane **architekturami dwuwarstwowymi**, ponieważ składniki oprogramowania są dzielone pomiędzy dwa systemy: klienta i serwera. Zaletą tego typu architektur jest ich prostota i bezproblemowa zgodność z istniejącymi systemami informatycznymi. Rozwój internetu (w szczególności popularność stron WWW) odmienił nieco role klientów i serwera, doprowadzając ostatecznie do powstania architektury trójwarstwowej.

### 2.5.4. Trójwarstwowe i n-warstwowe architektury typu klient-serwer dla aplikacji internetowych

Wiele aplikacji internetowych wykorzystuje model nazywany **architekturą trójwarstwową**, która dodaje pomiędzy klientem a serwerem bazy danych jeszcze jedną warstwę pośredniczącą (patrz rysunek 2.7(a)).



RYSUNEK 2.7. Logiczna trójwarstwowa architektura typu klient-serwer i wybrane standardowe pojęcia

Taka **środkowa** warstwa pośrednicząca jest niekiedy nazywana **warstwą aplikacji** lub **serwerem WWW**, w zależności od jej rzeczywistego zastosowania. Taki serwer pełni rolę pośrednika przechowującego reguły biznesowe (procedury lub ograniczenia), które są wykorzystywane do udostępniania danych z serwera bazy danych. Dodatkowa warstwa może także zwiększyć bezpieczeństwo bazy danych przez sprawdzanie danych uwierzytelniających klienta jeszcze przed przekazaniem otrzymanych żądań do serwera bazy danych. Komputery warstwy klienta zawierają interfejsy użytkownika i przeglądarki internetowe. Serwer pośredniczący przyjmuje i przetwarza żądania nadesłane przez klienta, po czym przesyła je dalej do serwera bazy danych (w postaci zapytań i poleceń). Następnie przekazuje (częściowo) przetworzone dane z serwera bazy danych do klientów, gdzie dane te mogą być dalej przetwarzane i filtrowane w celu ich zaprezentowania użytkownikom. Zatem rolę trzech warstw pełnią: *interfejs użytkownika*, *reguły aplikacji* oraz *dostęp do danych*. Na rysunku 2.7(b) pokazane jest inne ujęcie architektury trójwarstwowej używanej przez producentów baz danych i innych pakietów aplikacji. Warstwa prezentacji wyświetla tu informacje użytkownikowi i umożliwia wprowadzanie danych. Warstwa logiki biznesowej obsługuje reguły i ograniczenia pośrednie, stosowane przed przekazaniem danych w górę (do użytkownika) lub w dół (do SZBD). Dolna warstwa obejmuje wszystkie usługi zarządzania danymi. Warstwa pośrednia może też pełnić funkcję serwera WWW, który pobiera wyniki zapytań z serwera bazy danych i formatuje je do postaci dynamicznych stron WWW wyświetlanych za pomocą przeglądarki po stronie klienta. Maszyną klienta jest zwykle komputer osobisty lub urządzenie mobilne podłączone do internetu.

Zaproponowano także inne architektury. Warstwy pomiędzy użytkownikiem a składowanymi danymi można podzielić na bardziej szczegółowe komponenty, co skutkuje powstaniem architektur *n*-warstwowych (gdzie *n* to cztery lub pięć). Warstwa logiki biznesowej jest zwykle podzielona na wiele warstw. Oprócz możliwości udostępniania kodu i danych w sieci aplikacje *n*-warstwowe mają tę zaletę, że każdą warstwę można obsługiwać niezależnie oraz zastosować dla niej odpowiedni procesor lub system opera-

cyjny. Producenci pakietów ERP (ang. *enterprise resource planning*) i CRM (ang. *customer relationship management*) często stosują *warstwę pośrednią*, aby uwzględnić moduły frontonu (klientów) komunikujące się z licznymi bazami zaplecza (serwerami).

Postęp w obszarze technologii szyfrowania i deszyfrowania sprawił, że przesyłanie w postaci zaszyfrowanej poufnych danych z serwera do klienta (gdzie zostaną one odszyfrowane) jest teraz znacznie bezpieczniejsze. Operacja deszyfrowania może się opierać na odpowiednim sprzęcie lub na zaawansowanym oprogramowaniu. Wspomniana technologia zapewnia wyższy poziom bezpieczeństwa danych, problemem pozostaje jednak bezpieczeństwo komunikacji w sieci. Warto też pamiętać, że podczas przesyłania ogromnych ilości danych z serwerów do klientów za pośrednictwem sieci przewodowych lub bezprzewodowych pomocne są rozmaite technologie kompresji danych.

## 2.6. Klasyfikacja systemów zarządzania bazami danych

Podczas klasyfikowania systemów zarządzania bazami danych bierze się zwykle pod uwagę wiele kryteriów. Pierwszym jest **model danych**, na którym opiera się dany SZBD. Najczęściej spotykanym modelem tego typu wykorzystywanym w wielu współczesnych komercyjnych systemach zarządzania bazami danych jest **relacyjny model danych**. Oparte na nim systemy są nazywane **systemami SQL-owymi**. **Obiektowy model danych** co prawda został już zaimplementowany w kilku komercyjnych systemach, jednak nie jest na razie popularny. W nowszych **systemach typu big data** (inne nazwy to **systemy składowania danych typu klucz-wartość** lub **systemy NOSQL**) używane są różne modele danych: **oparte na dokumentach**, **oparte na grafach**, **oparte na kolumnach** i **typu klucz-wartość**. Wiele starszych rozwiązań nadal opiera się na **hierarchicznych** lub **sieciowych modelach danych**.

Relacyjne systemy zarządzania bazami danych stale ewoluują i są wzbogacane o nowe rozwiązania — dotyczy to zwłaszcza wielu cech charakterystycznych dla obiektowych baz danych. Taki kierunek rozwoju relacyjnych baz danych zaowocował powstaniem nowej klasy systemów zarządzania bazami danych, nazwanych **obiektowo-relacyjnymi** SZBD. Systemy zarządzania bazami danych możemy więc podzielić na kategorie według zastosowanego modelu danych: relacyjnego, obiektowego, obiektowo-relacyjnego, NOSQL, klucz-wartość, hierarchicznego, sieciowego itp.

Niektóre eksperymentalne SZBD są oparte na modelu XML. Jest to **drzewiasty model danych**. Są to tzw. **natywne XML-owe SZBD**. W kilku komercyjnych relacyjnych SZBD dodano XML-owe interfejsy i magazyny danych.

Drugim istotnym kryterium podziału systemów zarządzania bazami danych jest obsługiwana przez nie **liczba użytkowników**. Systemy **jednodostępne** obsługują w tym samym czasie tylko jednego użytkownika i są wykorzystywane przede wszystkim na komputerach osobistych. **Systemy wielodostępne**, do których należy większość współczesnych systemów zarządzania bazami danych, umożliwiają jednoczesną pracę wielu użytkownikom.

Trzecim kryterium jest **liczba węzłów**, pomiędzy które baza danych jest rozpraszana. System zarządzania bazą danych jest **scentralizowany**, jeśli dane są przechowywane na jednym komputerze. Scentralizowany system zarządzania bazą danych może co prawda

obsługiwać wielu użytkowników pracujących przy wielu komputerach, jednak sam system tego typu wraz z udostępnianą bazą danych musi się znajdować na jednym komputerze. **Rozproszony** system zarządzania bazą danych (ang. *Distributed Database Management System* — *DDBMS*) może wykorzystywać bazę danych i moduły oprogramowania SZBD działające na wielu różnych węzłach połączonych za pomocą sieci. Systemy big data są często wysoce rozproszone i działają w setkach węzłów. Dane są nieraz replikowane w wielu węzłach, dlatego awaria jednego z nich nie powoduje niedostępności danych.

**Homogeniczne** rozproszone systemy zarządzania bazami danych wykorzystują to samo oprogramowanie na wielu komputerach. W systemach **heterogenicznych** w każdym węźle używane może być inne oprogramowanie. Można też opracować **oprogramowanie warstwy pośredniej**, aby uzyskać dostęp do wielu autonomicznych, już istniejących baz danych zarządzanych przez heterogeniczne systemy SZBD. Kolejnym krokiem w tym kierunku są **federacyjne** systemy zarządzania bazami danych (tzw. **systemy wielo-bazodanowe**), w których wykorzystuje się wiele luźno powiązanych ze sobą systemów zachowujących w całej tej strukturze część lokalnej autonomii. W wielu rozproszonych systemach zarządzania bazami danych stosuje się architekturę typu klient-serwer, co opisano w podrozdziale 2.5.

Czwartym kryterium jest **koszt** systemu zarządzania bazą danych. Trudno jest zaproponować klasyfikację SZBD opartą na kosztach. Obecnie dostępne są otwarte (bezpłatne) produkty tego typu, np. MySQL i PostgreSQL, wzbogacane przez niezależnych producentów o dodatkowe usługi. Popularne relacyjne SZBD są dostępne w postaci bezpłatnych 30-dniowych wersji próbnych i wersji osobistych, które mogą kosztować tylko kilkaset złotych i udostępniać wiele funkcji. Bardzo rozbudowane systemy są sprzedawane w postaci modułowej z komponentami do obsługi dystrybucji danych, replikacji, przetwarzania równoległego, mechanizmów mobilnych itd. Konfiguracja takich produktów wymaga zdefiniowania licznych parametrów. Ponadto omawiane systemy są sprzedawane w formie licencji. Licencje na lokalizacje umożliwiają nieograniczone użytkowanie systemu bazy danych i uruchamianie go w dowolnej liczbie egzemplarzy w siedzibie klienta. Inny rodzaj licencji ogranicza liczbę jednocześnie użytkowników lub stanowisk w danej lokalizacji. Niezależne wersje dla pojedynczych użytkowników (np. systemu Microsoft Access) są sprzedawane na sztuki albo dodawane do konfiguracji komputera stacjonarnego lub laptopa. Za dodatkową opłatą udostępniane są funkcje hurtowni i drażenia danych, a także obsługa innych typów danych. Instalacja i utrzymanie dużego systemu bazy danych może kosztować miliony złotych rocznie.

Systemy zarządzania bazami danych możemy sklasyfikować także według dostępnych opcji w zakresie **typów ścieżek dostępu** do przechowywanych plików. Przykładowo, jedna z dobrze znanych rodzin systemów tego typu opiera się na odwróconych strukturach plików. Wreszcie, system zarządzania bazą danych może być rozwiązaniem **uniwersalnym** lub **wyspecjalizowanym**. W sytuacjach, gdy kluczowe wymaganie dotyczy wydajności systemu, warto rozważyć zaprojektowanie i budowę wyspecjalizowanego SZBD dla konkretnego zastosowania, który bez wprowadzenia istotnych zmian nie będzie mógł być wykorzystywany do innych zastosowań. Wiele opracowanych w przeszłości systemów zarządzania bazami danych wykorzystywanych w biurach rezerwacji biletów lotniczych lub przez operatorów telefonicznych było właśnie rozwiązaniami wyspecjalizowanymi. Tak powstały systemy **przetwarzania transakcji na bieżąco** (ang. *Online Transaction Processing* — *OLTP*), które muszą obsługiwać ogromną liczbę współbieżnych transakcji, nie powodując przy tym zbyt dużych opóźnień.

Spróbujmy teraz krótko omówić model danych — główne kryterium klasyfikacji systemów zarządzania bazami danych. Podstawowy **relacyjny model danych** reprezentuje bazę danych w postaci zbioru tabel, z których każda może być składowana w osobnym pliku. Baza danych zademonstrowana na rysunku 1.2 jest bardzo zbliżona do reprezentacji relacyjnej. Większość relacyjnych baz danych wykorzystuje wysokopoziomowy język zapytań SQL i obsługuje (choć w ograniczonym wymiarze) perspektywy użytkownika. Model relacyjny wraz z jego językami i operacjami techniki programowania relacyjnych aplikacji omówimy w rozdziałach od 5. do 8.; techniki programowania relacyjnych aplikacji omówimy w rozdziałach 10. i 11.

**Obiektowy model danych** definiuje bazę danych w oparciu o obiekty, ich własności oraz ich operacje. Obiekty z taką samą strukturą i zachowaniem należą do jednej **klasy**, natomiast same klasy są organizowane w **hierarchie** (nazywane także **grafami acyklicznymi**). Operacje dla poszczególnych klas są konstruowane na bazie predefiniowanych procedur nazywanych **metodami**. Modele wykorzystywane w wielu relacyjnych systemach zarządzania bazami danych zostały przystosowane do obsługi pojęć typowych dla obiektowych baz danych oraz funkcji innych systemów i są w związku z tym nazywane systemami **obiektowo-relacyjnymi** lub **rozszerzonymi systemami relacyjnymi**. Obiektowe bazy danych oraz systemy obiektowo-relacyjne omówimy w rozdziale 12.

Systemy big data są oparte na różnych modelach danych. Najczęściej stosowane są cztery z nich. **Model danych typu klucz-wartość** łączy z każdą wartością (może nią być rekord lub obiekt) unikatowy klucz i zapewnia bardzo szybki dostęp do wartości na podstawie klucza. **Model danych oparty na dokumentach** wykorzystuje format JSON (ang. *Java Script Object Notation*) i przechowuje dane jako dokumenty przypominające nieco złożone obiekty. **Model danych oparty na grafie** przechowuje obiekty jako węzły grafu, a relacje między obiektami jako krawędzie grafu skierowanego. **Kolumnowe modele danych** przechowują kolumny wierszy pogrupowane na stronach dysku, aby umożliwić szybki dostęp do danych i przechowywanie ich w wielu wersjach. Niektóre z tych modeli są opisane szczegółowo w rozdziale 24.

**Model XML-owy** powstał jako standard wymiany danych w internecie i był używany do zaimplementowania kilku prototypowych natywnych systemów XML-owych. W XML-u używane są hierarchiczne struktury drzewiaste. Ten model łączy mechanizmy baz danych z rozwiązaniami z modeli reprezentacji dokumentów. Dane są tu reprezentowane jako elementy. Za pomocą znaczników dane można zagnieżdżać i tworzyć w ten sposób złożone struktury drzewiaste. Ten model koncepcyjnie przypomina model obiektowy, choć stosowana jest tu inna terminologia. Mechanizmy XML-owe zostały dodane do wielu komercyjnych SZBD. Omówienie XML-a prezentujemy w rozdziale 13.

Dwoma znacznie starszymi, historycznie ważnymi modelami danych, uważanymi obecnie za **przestarzałe**, są modele sieciowy i hierarchiczny. **Model sieciowy** reprezentuje dane w postaci typów rekordów i umożliwia (choć w ograniczonym zakresie) reprezentowanie relacji typu 1:N, tzw. **typu zbiorowego**. Relacja 1:N (jeden do wielu) łączy jeden egzemplarz rekordu z wieloma innymi za pomocą wskaźników. Model sieciowy, znany także jako model CODASYL DBTG<sup>12</sup>, jest ściśle związany z językiem typu *record-at-a-time*, któ-

---

<sup>12</sup> Skrót CODASYL DBTG pochodzi od nazwy komitetu Conference of Data Systems Languages Data Base Task Group, który zdefiniował model sieciowy i jego język.

rego instrukcje muszą być osadzone w kodzie nadrzędnego języka programowania. Sieciowy język DML został zaproponowany w pracy 1971 Database Task Group (DBTG) Report jako rozszerzenie języka COBOL.

**Model hierarchiczny** reprezentuje dane w postaci hierarchicznej struktury drzewiastej. Każda taka hierarchia reprezentuje wiele powiązanych ze sobą rekordów. Dla modelu hierarchicznego nie istnieje standardowy język. Popularnym hierarchicznym językiem DML jest DL/1 z systemu IMS. Był to najpopularniejszy SZBD w latach 1965 – 1985. Używany w nim język DML (DL/1) był przez długi czas w zasadzie standardem w branży<sup>13</sup>.

## 2.7. Podsumowanie

W tym rozdziale wprowadziliśmy podstawowe terminy wykorzystywane w systemach baz danych. Zdefiniowaliśmy pojęcie modelu danych i wyróżniliśmy trzy główne kategorie modeli tego typu:

- wysokopoziomowe modele danych (nazywane też koncepcyjnymi), które opierają się na encjach i związkach;
- niskopoziomowe modele danych (nazywane też fizycznymi);
- reprezentacyjne modele danych (nazywane też implementacyjnymi), które opierają się na rekordach lub obiektach.

W rozdziale zdefiniowaliśmy pojęcie schematu (opisu) bazy danych, które oddzieliliśmy od samej bazy. Schemat bazy danych nie jest zbyt często modyfikowany, natomiast stan bazy danych zmienia się za każdym razem, gdy wstawimy, usuniemy lub zmodyfikujemy dane. Następnie opisaliśmy architekturę trójwarstwową systemów zarządzania bazami danych, która umożliwia stosowanie aż trzech poziomów schematów bazy danych:

- schemat wewnętrzny, który opisuje fizyczną strukturę przechowywania bazy danych;
- schemat koncepcyjny, który jest wysokopoziomowym opisem całej bazy danych;
- schemat zewnętrzny, który opisuje perspektywy różnych grup użytkowników.

System zarządzania bazą danych, który ściśle oddziela te trzy poziomy, musi zawierać reguły odwzorowywania pomiędzy poszczególnymi schematami, aby właściwie przekształcać zapytania i wyniki przekazywane z jednego poziomu do drugiego. Większość systemów zarządzania bazami danych nie oddziela tych trzech poziomów całkowicie. Wykorzystaliśmy w tym rozdziale architekturę trójwarstwową głównie po to, aby zdefiniować pojęcia logicznej i fizycznej niezależności danych.

W dalszej części tego rozdziału omówiliśmy najważniejsze typy języków i interfejsów obsługiwanych przez systemy zarządzania bazami danych. Język definicji danych (DDL) jest wykorzystywany do definiowania schematu koncepcyjnego bazy danych. W większości systemów tego typu język DDL jest ponadto wykorzystywany do definiowania per-

---

<sup>13</sup> Kompletne rozdziały na temat modeli sieciowych i hierarchicznych z drugiego wydania tej książki są dostępne w witrynie tej pozycji: <http://www.aw.com/elmasri>.



spektyw użytkownika, a niekiedy także struktur składowania; w innych systemach do określania struktur składowania danych mogą być używane odrębne języki lub funkcje. We współczesnych implementacjach systemów relacyjnych różnice między opisanymi funkcjami zanikają, ponieważ SQL działa jak uniwersalny język do wykonywania wielu zadań, w tym do definiowania perspektyw. We wczesnych wersjach SQL-a znajdował się język definiowania struktur składowania (język SDL), jednak obecnie w relacyjnych SZBD zwykle implementuje się go w formie specjalnych poleceń dla administratorów baz danych. System SZBD kompiluje wszystkie definicje schematów i umieszcza ich opisy w swoim katalogu.

Do określania operacji odczytywania i aktualizowania danych wykorzystuje się język manipulowania danymi (DML). Języki tego typu mogą być wysokopoziomowe (*set-oriented*, lub nieproceduralne) lub niskopoziomowe (*record-oriented*, lub proceduralne). Wysokopoziomowe języki DML można albo osadzać w kodzie nadrzędnego języka programowania, albo wykorzystywać w roli języków autonomicznych; w tym drugim przypadku języki tego typu są często nazywane językami zapytań.

Omówiliśmy także różne typy interfejsów udostępnianych przez systemy zarządzania bazami danych oraz rozmaite kategorie użytkowników systemów tego typu, dla których tworzone są poszczególne interfejsy. Dalej opisaliśmy środowisko systemu zarządzania bazą danych, typowe moduły programowe takiego systemu oraz narzędzia ułatwiające użytkownikom i administratorom wykonywanie ich zadań. W dalszej części tego rozdziału zaprezentowano przegląd architektur dwu- i trójwarstwowych używanych w aplikacjach bazodanowych.

W ostatnich fragmentach sklasyfikowaliśmy systemy zarządzania bazami danych według kilku kryteriów: modelu danych, liczby użytkowników, liczby węzłów, typów ścieżek dostępu oraz kosztów. Omówiliśmy dostępność SZBD i dodatkowych modułów — od rozwiązań darmowych w postaci oprogramowania otwartego po konfigurację, których utrzymanie kosztuje miliony złotych rocznie. Wspomnieliśmy też o różnorodnych licencjach na SZBD i powiązane produkty. Najważniejszym kryterium podziału systemów zarządzania bazami danych jest oczywiście stosowany model danych. W tym rozdziale omówiliśmy krótko główne modele danych wykorzystywane we współczesnych komercyjnych systemach tego typu.

## Pytania powtórkowe

- 2.1. Zdefiniuj następujące pojęcia: *model danych*, *schemat bazy danych*, *stan bazy danych*, *schemat wewnętrzny*, *schemat koncepcyjny*, *schemat zewnętrzny*, *niezależność danych*, *język DDL*, *język DML*, *język SDL*, *język VDL*, *język zapytań*, *język nadrzędny*, *podjęzyk danych*, *narzędzie bazy danych*, *katalog*, *architektura typu klient-serwer*, *architektura trójwarstwowa* i *architektura n-warstwowa*.
- 2.2. Przedstaw główne kategorie modeli danych. Jakie są podstawowe różnice między modelami relacyjnym, obiektywowym a XML-owym?
- 2.3. Jaka jest różnica pomiędzy schematem bazy danych a stanem bazy danych?
- 2.4. Opisz architekturę trójwarstwową. Wyjaśnij, dlaczego niezbędne jest stosowanie reguł odwzorowywania pomiędzy poziomami schematu. W jaki sposób różne języki definiowania schematu wspomagają stosowanie tej architektury?



- 2.5. Jaka jest różnica pomiędzy logiczną niezależnością danych a fizyczną niezależnością danych? Którą z nich trudniej jest osiągnąć? Dlaczego?
- 2.6. Jaka jest różnica pomiędzy proceduralnymi a nieproceduralnymi językami DML?
- 2.7. Przedstaw różne typy przyjaznych dla użytkownika interfejsów oraz kategorie użytkowników, które zazwyczaj korzystają z poszczególnych typów.
- 2.8. Z jakimi innymi pakietami oprogramowania komputerowego współpracują systemy zarządzania bazami danych?
- 2.9. Jaka jest różnica pomiędzy dwu- i trójwarstwowymi architekturami typu klient-serwer?
- 2.10. Przedstaw niektóre typy przydatnych programów i narzędzi baz danych wraz z oferowanymi przez nie funkcjami.
- 2.11. Jakie dodatkowe funkcje obejmuje architektura  $n$ -warstwowa ( $n > 3$ )?

## Ćwiczenia

- 2.12. Omów rodzaje potencjalnych użytkowników bazy danych z rysunku 1.2. Jakich typów aplikacji będą potrzebowali poszczególni użytkownicy? Do której z kategorii użytkowników należałoby ich przypisać i jakich typów interfejsów będą ci użytkownicy potrzebowali?
- 2.13. Wybierz znane Ci zastosowanie bazy danych. Zaprojektuj odpowiedni schemat i przedstaw przykładową bazę danych dla tego zastosowania (zgodnie z notacją użytą na rysunkach 2.1 i 1.2). Jakiego rodzaju dodatkowych informacji i ograniczeń chciałbyś użyć do reprezentowania tego schematu? Przemyśl rolę wielu różnych użytkowników swojej bazy danych i dla każdego z nich zaprojektuj osobną perspektywę.
- 2.14. Gdybyś miał zaprojektować internetowy system rezerwacji biletów lotniczych, którą architekturę SZBD z podrozdziału 2.5 byś wybrał? Dlaczego? Z jakiego powodu inne architektury nie będą dobrym wyborem?
- 2.15. Przyjrzyj się rysunkowi 2.1. Obok ograniczeń wiążących wartości kolumn z jednej tabeli z kolumnami innej tabeli występują też ograniczenia dotyczące wartości kolumn (pojedynczych lub ich kombinacji) tabeli. Jedno z takich ograniczeń powoduje, że kolumna lub grupa kolumn musi mieć unikatowe wartości w skali wszystkich wierszy tabeli. Na przykład w tabeli STUDENT unikatowa musi być wartość kolumny NumerIndeksu (aby zapobiec posiadaniu identycznego numeru przez dwóch studentów). Wskaż w innych tabelach kolumny lub grupy kolumn, które muszą być unikatowe w skali wszystkich wierszy danej tabeli.

## Wybrane publikacje

Wiele książek poświęconych bazom danych, włącznie z publikacjami Date'a (2004), Ramakrishnana i Gehrke'a (2003), Garcia-Moliny i in. (2002, 2009) oraz Abiteboula i in. (1995), zawiera szczegółowe omówienie rozmaitych zaprezentowanych w tym rozdziale zagadnień związanych z bazami danych. Książka autorstwa Tsichritzisa i Lochovsky'ego (1982)

była jedną z pierwszych publikacji na temat modeli danych. Zarówno książka napisana przez Tsichritzisa i Kluga (1978), jak i książka autorstwa Jardine'a (1977) prezentowały architekturę trójwarstwową, której koncepcję zaproponowano po raz pierwszy w raporcie komisji DBTG CODASYL (1971) i później w raporcie Amerykańskiego Instytutu Normalizacyjnego — ANSI (1975). Dogłębną analizę relacyjnego modelu danych i niektórych z jego potencjalnych rozszerzeń można znaleźć w książce Codd'a (1990). Cattell i in. (2000) zawarli w swojej książce propozycję standardu dla obiektowych baz danych. Wiele materiałów opisujących język XML jest dostępnych w internecie, np. XML (2005).

Przykładami programów narzędziowych wykorzystywanych z bazami danych są: narzędzia Connect, Analyze i Transform firmy ETI (<http://www.eti.com>) oraz przeznaczone dla administratorów baz danych narzędzie DB Artisan firmy Embarcadero Technologies (<http://www.embarcadero.com>).



---

# Koncepcyjne modelowanie danych i projektowanie baz danych



## Modelowanie danych zgodnie z modelem związków encji

Modelowanie koncepcyjne jest bardzo ważną fazą skutecznego projektowania aplikacji bazy danych. Ogólnie rzecz biorąc, termin **aplikacja bazy danych** odwołuje się do konkretnej bazy danych i powiązanych z nią programów, które implementują zapytania i operacje aktualizacji informacji zawartych w tej bazie danych. Przykładowo, aplikacja bazy danych BANK, która zarządza kontami klientów, powinna zawierać programy implementujące operacje aktualizacji tej bazy danych właściwe dla wykonywanych przez klientów wpłat i wypłat. Takie programy powinny z kolei oferować przyjazne dla użytkownika interfejsy graficzne (interfejsy GUI), których formularze i menu będą stanowiły ułatwienie dla końcowych użytkowników aplikacji — w tym przypadku klientów lub kasjerów. Obecnie klientom aplikacji BANK często udostępniane są też interfejsy w postaci **aplikacji mobilnych**, przeznaczonych na urządzenia mobilne. Duża część aplikacji bazy danych wymaga opracowania projektu, wykonania implementacji i przeprowadzenia testów tych aplikacji. Projektowanie i testowanie **aplikacji** jest tradycyjnie uważane raczej za domenę *inżynierii oprogramowania* niż za element *projektowania bazy danych*. Jednak w wielu narzędziach do projektowania oprogramowania metody projektowania baz danych i inżynierii oprogramowania są połączone, ponieważ zadania te są ze sobą ściśle powiązane.

W tym rozdziale będziemy postępować zgodnie z tradycyjnym modelem, w którym przedmiotem naszej szczególnej uwagi w fazie projektowania będą struktury oraz ograniczenia baz danych. Projektowanie aplikacji jest zwykle omawiane na kursach z zakresu inżynierii oprogramowania. Zaprezentujemy zagadnienia związane z **modelem związków encji** (ang. *Entity-Relationship* — *ER*), który jest popularnym, wysokopoziomowym, koncepcyjnym modelem danych. Model ER i jego odmiany są często wykorzystywane podczas koncepcyjnego projektowania aplikacji baz danych i wiele wykorzystywanych do tego celu narzędzi opiera się na tej metodzie. W tym rozdziale omówimy podstawowe zagadnienia związane z tworzeniem struktur danych oraz ograniczeniami w modelu ER, po czym skupimy się na zastosowaniu tych technik podczas projektowania koncepcyjnych schematów aplikacji baz danych. Zaprezentujemy także notację diagramów powiązanych z modelem ER, znanych jako **diagramy ER**.

Takie metody modelowania obiektowego jak **zunifikowany język modelowania** (ang. *Universal Modeling Language* — *UML*) stają się coraz bardziej popularne w obszarze projektowania aplikacji i inżynierii oprogramowania. Metody tego typu są wykorzystywane do szczegółowego projektowania modułów programowych i interakcji w oparciu

o rozmaite rodzaje diagramów. Istotną częścią tych metod są *diagramy klas*<sup>1</sup>, które pod wieloma względami przypominają diagramy związków encji (ER). W diagramie klas, poza strukturą schematu bazy danych, definiuje się *operacje* na obiektach. Operacje mogą być dalej wykorzystywane do określania *wymagań funkcjonalnych* w czasie projektowania bazy danych (patrz podrozdział 3.1). W podrozdziale 3.8 zaprezentujemy niektóre elementy notacji UML i te zagadnienia związane z diagramami klas, które są szczególnie ważne w odniesieniu do projektowania baz danych, porównamy także demonstrowaną metodę do notacji ER i związanych z nią pojęć. Notację i koncepcję modelowania UML przedstawimy ponadto w podrozdziale 4.6.

Ten rozdział został zorganizowany w następujący sposób. W podrozdziale 3.1 omówiono rolę wysokopoziomowych, koncepcyjnych modeli danych w projektowaniu baz danych. W podrozdziale 3.2 wprowadziliśmy wymagania dla przykładowej aplikacji bazy danych, aby zilustrować praktyczne zastosowanie prezentowanych pojęć z modelu związków encji (ER). Tę samą przykładową bazę danych będziemy wykorzystywali także w kolejnych rozdziałach tej książki. W podrozdziale 3.3 zaprezentujemy pojęcia encji i atrybutów oraz zaczniemy stopniowo wprowadzać techniki prezentowania schematów ER za pomocą odpowiednich diagramów. W podrozdziale 3.4 wprowadzimy pojęcia związków binarnych (wraz z ich praktycznymi znaczeniami) oraz ograniczeń strukturalnych. Podrozdział 3.5 zawiera wprowadzenie tzw. słabych typów encji. W podrozdziale 3.6 zaprezentujemy wzbogacenie projektu schematu o informacje o związkach. W podrozdziale 3.7 raz jeszcze przeanalizujemy notację diagramów ER, podsumujemy problemy pojawiające się w czasie projektowania schematów oraz omówimy sposób, w jaki należy dobierać nazwy dla poszczególnych konstrukcji schematu bazy danych. Podrozdział 3.8 zawiera wprowadzenie niektórych zagadnień związanych z diagramami klas UML, porównanie tych aspektów z odpowiednimi elementami modelu ER oraz zastosowania UML-a w przykładowej bazie FIRMA. Podrozdział 3.9 obejmuje opis złożonych rodzajów relacji. Podrozdział 3.10 jest podsumowaniem całego rozdziału 3.

Omawianie materiału zawartego w podrozdziałach 3.8 i 3.9 można pominąć w trakcie zajęć wprowadzających. Z kolei jeśli prowadzący takie zajęcia chce się w większym stopniu skupić na pojęciach związanych z modelowaniem danych i na koncepcyjnym projektowaniu baz danych, należałoby przejść do omawiania materiału z rozdziału 4. Rozdział 4. opisuje rozszerzenia modelu ER, które tworzą tzw. rozszerzony model związków encji (ang. *Enhanced-ER* — *EER*), obejmujący takie pojęcia jak specjalizacja, generalizacja, dziedziczenie czy typy unii (kategorie).

---

<sup>1</sup> Pojęcie **klasy** w wielu aspektach przypomina pojęcie *typu encji*.

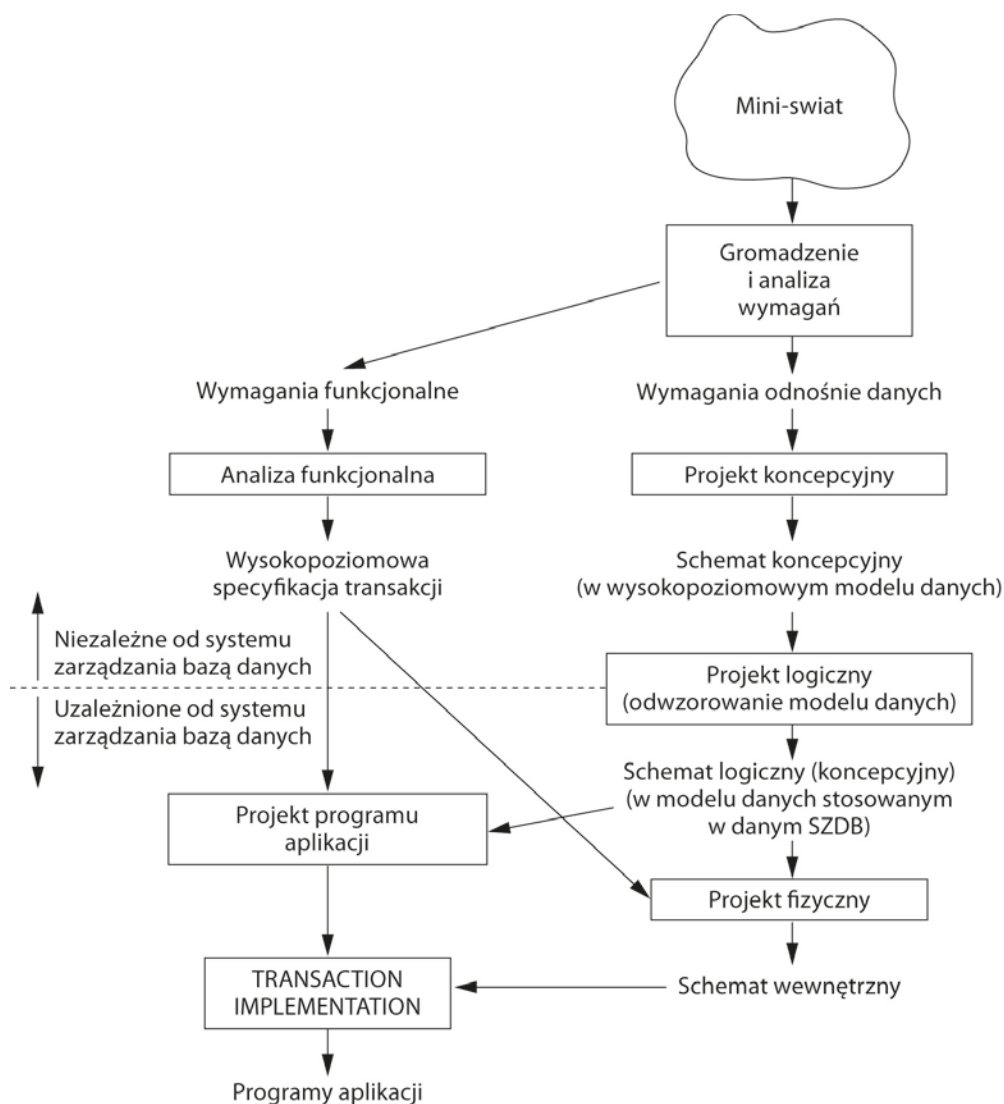
## 3.1. Stosowanie wysokopoziomowych, koncepcyjnych modeli danych podczas projektowania bazy danych

Na rysunku 3.1 przedstawiono uproszczony opis procesu projektowania bazy danych. Pierwszym zaprezentowanym krokiem jest **zgromadzenie i analiza wymagań**. Ten etap wymaga od projektantów bazy danych przeprowadzenia wywiadów z przewidywanymi użytkownikami końcowymi, aby dokładnie zrozumieć i udokumentować ich **wymagania danych**. Wynikiem tego kroku jest zwięźle zapisany zbiór wymagań użytkowników. Wymagania te powinny być tak szczegółowe i kompletne jak to tylko możliwe. Równoległe z określaniem wymagań danych warto zdefiniować znane na tym etapie **wymagania funkcjonalne** dla projektowanej aplikacji. Takie wymagania powinny się składać z określonych przez przyszłych użytkowników **operacji** (lub **transakcji**), które będą wykonywane na bazie (dotyczy to zarówno operacji odczytywania danych, jak i operacji ich aktualizowania). Podczas projektowania oprogramowania często wykorzystuje się *diagramy przepływu danych*, *diagramy sekwencji*, *scenariusze* oraz inne techniki definiowania wymagań funkcjonalnych. W tej książce nie będziemy szczegółowo omawiać wspomnianych technik, ponieważ ich dogłębną analizę można znaleźć w materiałach dotyczących inżynierii oprogramowania.

Kiedy już wszystkie wymagania zostaną zgromadzone i przeanalizowane, kolejnym krokiem jest stworzenie dla nowej bazy danych **schematu koncepcyjnego** w oparciu o wysokopoziomowy, koncepcyjny model danych. Ten krok nosi nazwę **projektowania koncepcyjnego**. Schemat koncepcyjny jest zwięzłym opisem zgłoszonych przez docelowych użytkowników wymagań danych i zawiera szczegółowe opisy typów encji, związków oraz ograniczeń; wszystkie te składniki są wyrażane za pomocą elementów wykorzystywanego w tej fazie wysokopoziomowego modelu danych. Ponieważ wymienione składniki nie obejmują opisu szczegółów implementacji, ich zrozumienie jest przeważnie łatwiejsze, zatem mogą być wykorzystywane do komunikowania się z użytkownikami nieposiadającymi przygotowania technicznego. Wysokopoziomowy schemat koncepcyjny może także być wykorzystywany w roli punktu odniesienia podczas oceny stopnia realizacji wszystkich wymagań danych zdefiniowanych przez użytkowników i w czasie sprawdzania ewentualnych konfliktów pomiędzy tymi wymaganiami. Takie rozwiązanie umożliwia projektantom baz danych koncentrowanie się na określaniu cech danych bez konieczności rozważania szczegółów związanych z ich fizycznym składowaniem. W efekcie, przygotowywanie dobrego projektu koncepcyjnego bazy danych jest dla osób zaangażowanych w to przedsięwzięcie znacznie łatwiejsze.

W trakcie lub bezpośrednio po projektowaniu schematu koncepcyjnego, do określania wysokopoziomowych operacji użytkownika (identyfikowanych podczas analizy funkcjonalnej) można wykorzystać podstawowe operacje modelu danych. W ten sposób projektant bazy danych może dodatkowo potwierdzić zgodność schematu koncepcyjnego ze wszystkimi zidentyfikowanymi wymaganiami funkcjonalnymi. Jeśli okaże się, że niektóre z wymagań funkcjonalnych nie mogą zostać określone w początkowym schemacie, można wprowadzić niezbędne modyfikacje do schematu koncepcyjnego.





RYSUNEK 3.1. Uproszczony diagram ilustrujący główne etapy projektowania bazy danych

Kolejnym krokiem w procesie projektowania bazy danych jest jej właściwa implementacja w oparciu o jeden z komercyjnych systemów zarządzania bazami danych. Większość współczesnych systemów tego typu wykorzystuje odpowiedni implementacyjny model danych (np. model relacyjnej — SQL-owy), co oznacza, że schemat koncepcyjny jest przekształcany z wysokopoziomowego modelu danych w implementacyjny model danych. Taki krok jest często nazywany **projektowaniem logicznym** lub **odzworowaniem modelu danych**, a jego wynikiem jest schemat bazy danych zgodny z modelem implementacyjnym wykorzystywanym w danym systemie zarządzania bazą danych.

Ostatnim krokiem jest etap **projektowania fizycznego**, w czasie którego definiuje się wewnętrzne struktury składowania, organizację plików, indeksy, ścieżki dostępu oraz fizyczne aspekty plików bazy danych. Równolegle z tymi działaniami można projektować i implementować programy aplikacji bazy danych, które będą odpowiadały za realizację odpowiednich transakcji opisanych w wysokopoziomowych specyfikacjach.

W tym rozdziale prezentujemy tylko najbardziej podstawowe zagadnienia związane z projektowaniem schematów koncepcyjnych zgodnie z modelem ER. Więcej informacji na ten temat można znaleźć w rozdziale 4., gdzie wprowadziliśmy rozszerzony model ER (EER).

## 3.2. Przykładowa aplikacja bazy danych

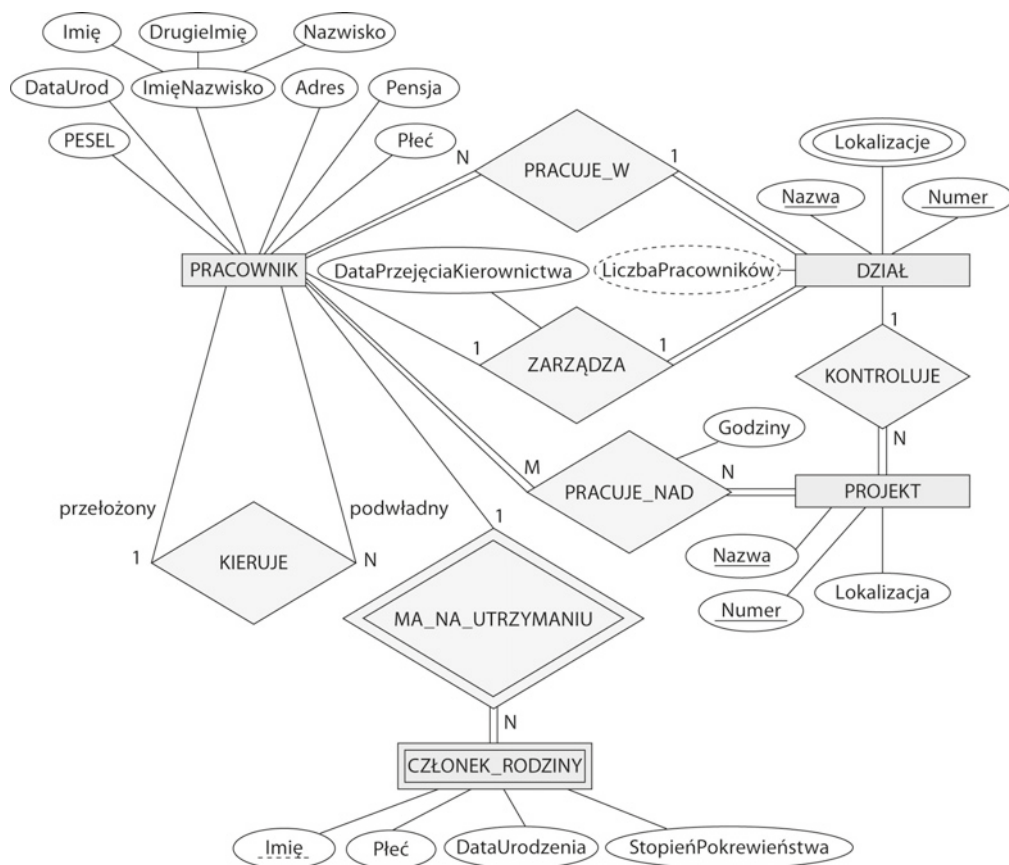
W tym podrozdziale opiszemy przykładową aplikację bazy danych (nazwaną FIRMA), która ma ilustrować podstawowe zagadnienia związane z modelem związków encji (ER) wraz z ich zastosowaniem podczas projektowania schematu. Poniżej przedstawiono listę wymagań danych dla projektowanej bazy; w dalszej kolejności będziemy krok po kroku tworzyć schemat koncepcyjny, wprowadzając jednocześnie kolejne elementy modelu ER. Baza FIRMA ma umożliwiać zarządzanie danymi pracowników, działów oraz realizowanych projektów. Przypuśćmy, że po fazie gromadzenia i analizy wymagań projektanci bazy danych opracowali następujący opis „mini-świata” — czyli tej części firmy, która ma być reprezentowana w bazie danych:

- Firma jest podzielona na działy. Każdy z tych działów ma unikatową nazwę, unikatowy numer oraz przydzielonego konkretnego pracownika, który tym działem kieruje. W bazie danych należy utrzymywać datę początkową, od której dany pracownik kieruje wskazanym działem. Każdy dział może być rozproszony i znajdować się w wielu miejscach.
- Dział kontroluje wiele projektów, z których każdy ma unikatową nazwę, unikatowy numer oraz jedno miejsce realizacji.
- W bazie danych musi być przechowywane nazwisko i numer PESEL<sup>2</sup>, adres, wysokość pensji, płeć i data urodzenia każdego z pracowników firmy. Pracownik firmy musi być przypisany do jednego działu, ale może pracować nad wieloma projektami, które niekoniecznie muszą być kontrolowane przez ten sam dział. W bazie danych będziemy śledzić liczbę godzin, które pracownicy poświęcają poszczególnym projektom w ciągu tygodnia. Dla każdego pracownika będziemy dodatkowo przechowywali informację o bezpośrednim zwierzchniku.
- Chcemy przechowywać (np. w celach ubezpieczeniowych) informacje o rodzinach poszczególnych pracowników. Dla każdego członka rodziny pracownika firmy w bazie danych będą utrzymywane następujące elementy danych: imię, płeć, data urodzenia oraz stopień pokrewieństwa z pracownikiem.

---

<sup>2</sup> Numer PESEL jest unikatowym, 11-cyfrowym identyfikatorem przydzielanym każdemu obywatelowi Polski, wykorzystywanym w bardzo różnych celach — przez pracodawców, firmy ubezpieczeniowe czy urzędy skarbowe. Inne kraje mogą wykorzystywać podobne metody identyfikacji obywateli, jak choćby numer ubezpieczenia społecznego SSN (od ang. *Social Security Number*) w Stanach Zjednoczonych czy identyfikatory dowodów osobistych — *przyp. tłum.*

Na rysunku 3.2 przedstawiono sposób, w jaki schemat dla tej aplikacji bazy danych można reprezentować za pomocą notacji graficznej znanej jako **diagramy ER**. Rysunek ten będzie objaśniany stopniowo, wraz z omawianiem zagadnień związanych z modelem ER. Wyjaśnimy notację diagramów ER, wprowadzając najważniejsze zagadnienia związane z modelem ER i opisując krok po kroku proces tworzenia tego schematu na podstawie uzgodnionych wymagań.



RYСУNEK 3.2. Diagram schematu związków encji (ER) dla bazy danych FIRMA. Notacja używana na diagramie będzie omawiana stopniowo w tym rozdziale, a jej podsumowanie przedstawia rysunek 3.14

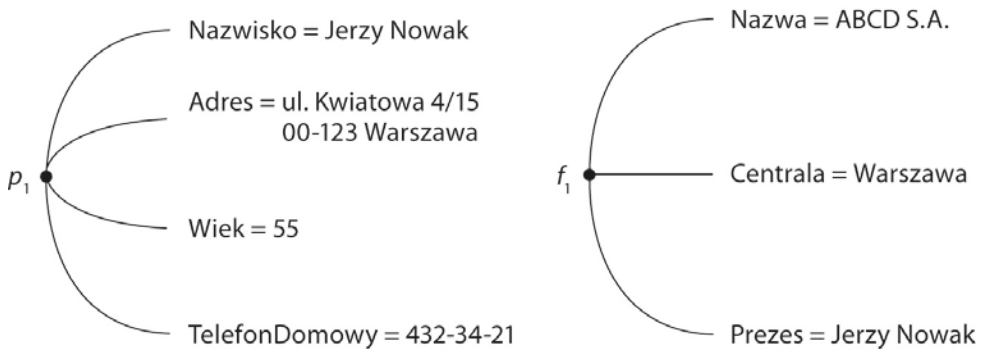
### 3.3. Typy encji, zbiory encji, atrybuty i klucze

Model ER opisuje takie dane jak *encje*, *związki* i *atrybuty*. W punkcie 3.3.1 wprowadzimy pojęcia encji i ich atrybutów. Typy encji oraz atrybuty klucza omówimy w punkcie 3.3.2. Następnie (w punkcie 3.3.3) zdefiniujemy początkowy projekt koncepcyjny dla typów encji bazy danych FIRMA. Opisem związków zajmiemy się w punkcie 3.3.4.

### 3.3.1. Encje i atrybuty

**Encje i ich atrybuty.** Podstawowym obiektem reprezentowanym w modelu ER jest **encja**, czyli *byt* ze świata rzeczywistego, którego istnienie nie zależy od innych elementów. Encja może być fizycznie istniejącym obiektem (przykładowo, może to być konkretna osoba, samochód, dom lub pracownik) lub obiektem, którego istnienie ma charakter koncepcyjny (czyli np. firmą, pracą lub przedmiotem wykładanym na uniwersytecie). Każda encja ma **atomy**, czyli opisujące ją szczegółowe właściwości. Przykładowo, encja pracownika może być opisywana za pomocą jego nazwiska, wieku, adresu zamieszkania, wysokości pensji oraz realizowanych zadań. Konkretna encja będzie zawierała po jednej wartości dla każdego takiego atrybutu. Wartości atrybutów opisujące poszczególne encje stają się główną częścią danych składowanych w bazie danych.

Na rysunku 3.3 przedstawiono dwie encje wraz z wartościami ich atrybutów. Encja pracownika  $p_1$  ma cztery atrybuty: Nazwisko, Adres, Wiek i TelefonDomowy; wartości tych atrybutów to odpowiednio Jan Nowak, ul. Kwiatowa 4/15, 00-123 Warszawa, 55 oraz 432-34-21. Encja firmy  $f_1$  ma trzy atrybuty: Nazwa, Centrala i Prezes; wartości tych atrybutów to odpowiednio: ABCD S.A., Warszawa i Jan Nowak.

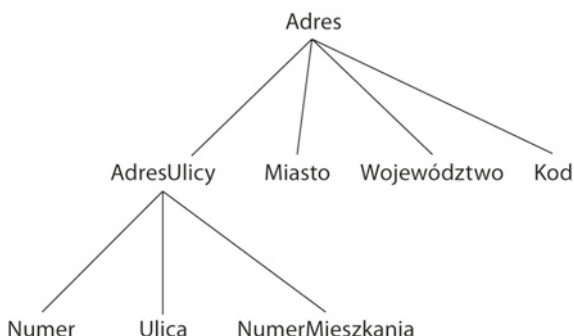


RYSunEK 3.3. Dwie encje (pracownik  $p_1$  i firma  $f_1$ ) wraz z atrybutami

W modelu ER wykorzystuje się wiele typów atrybutów: *proste* i *złożone*, *jednowartościowe* i *wielowartościowe* oraz *stałe* i *po pochodne*. W pierwszej kolejności zdefiniujemy te atrybuty i zilustrujemy ich praktyczne zastosowania za pomocą odpowiednich przykładów. Następnie wprowadzimy pojęcie *wartości pustej* dla atrybutów.

**Atrybuty złożone kontra atrybuty proste (atomowe).** Atrybuty **złożone** można podzielić na mniejsze podgrupy reprezentujące prostsze atrybuty, których znaczenia są niezależne. Przykładowo, zaprezentowany na rysunku 3.3 atrybut Adres encji reprezentującej pracownika można podzielić na Ulicę, Numer, Kod i Miejscowość z wartościami Kwiatowa, 4/15, 00-123 i Warszawa. Atrybuty niepodzielne są nazywane atrybutami **prostymi** lub **atomowymi**. Atrybuty złożone mogą tworzyć hierarchie; przykładowo atrybut Numer może być dalej podzielony na dwa atrybuty proste: NumerDomu, Ulicę oraz NumerMieszkania (patrz rysunek 3.4). Wartość atrybutu złożonego jest w takim przypadku konkatencją wartości składowych atrybutów prostych.

Atrybuty złożone są przydatne podczas modelowania sytuacji, w których użytkownik od czasu do czasu odwołuje się do danych złożonych tak jak do pojedynczego elementu danych, jednak w pozostałych przypadkach operuje na jego wyodrębnionych składnikach. Jeśli do atrybutu złożonego użytkownik odwołuje się wyłącznie jak do jednej całości, oczywiście nie ma potrzeby jego rozbijania na atrybuty składowe. Przykładowo, jeśli nie ma konieczności odwoływania się do poszczególnych składników adresu (kodu, ulicy itp.), podczas projektowania schematu bazy danych można przydzielić do tego elementu pojedynczy, niepodzielny atrybut obejmujący wszystkie potrzebne dane.



RYSUNEK 3.4. Hierarchia atrybutów złożonych

**Atrybuty jednowartościowe kontra atrybuty wielowartościowe.** Większość atrybutów reprezentuje pojedynczą wartość dla konkretnej encji — takie atrybuty nazywamy **jednowartościowymi**. Przykładowo, *Wiek* jest jednowartościowym atrybutem osoby reprezentowanej za pomocą encji. W niektórych przypadkach atrybut może mieć cały zbiór wartości dla tej samej encji — przykładowo, może to dotyczyć atrybutu *Kolory* dla encji reprezentującej samochód lub atrybutu *StopnieNaukowe* encji reprezentującej osobę. Jednokolorowe samochody mają pojedynczą wartość, natomiast samochody dwukolorowe mają dwie wartości jednego atrybutu *Kolory*. Podobnie, jedna osoba może nie posiadać żadnych tytułów naukowych, inna osoba może mieć jeden taki tytuł, a jeszcze inna osoba może mieć dwa lub trzy tytuły; zatem różne osoby mogą mieć różne *liczby wartości* atrybutu *StopnieNaukowe*. Takie atrybuty są nazywane **wielowartościowymi**. Atrybuty wielowartościowe mogą mieć przypisane dolne i górne ograniczenia definiujące akceptowane liczby wartości dla pojedynczych encji. Przykładowo, atrybut *Kolory* encji reprezentującej samochód może mieć od jednej do dwóch wartości, jeśli przyjmiemy, że jeden samochód może mieć najwyżej dwie kolory.

**Atrybuty stałe kontra atrybuty pochodne.** W niektórych przypadkach dwie (lub więcej) wartości atrybutu są ze sobą powiązane — przykładowo, może to dotyczyć atrybutu *Wiek* i *DataUrodzenia* w encji reprezentującej osobę. W konkretnej encji wartość atrybutu *Wiek* może być wyznaczana na podstawie bieżącej daty i wartości atrybutu *DataUrodzenia* w tej samej encji. Atrybut *Wiek* jest w takim przypadku nazywany **atrybutem pochodnym** i mówi się, że jego wartość jest **wywodzona z** wartości atrybutu *DataUrodzenia*, który z kolei jest nazywany **atrybutem stałym**. Niektóre wartości atrybutów mogą być wywodzone z *powiązanych encji*; przykładowo, wartość atrybutu *LiczbaPracowników* w encji reprezentującej dział firmy może być wyznaczana w oparciu o liczbę pracowników związanych z tym działem (pracujących w nim).

**Wartości puste.** W niektórych przypadkach konkretna encja może nie mieć żadnej właściwej wartości dla jakiegoś atrybutu. Przykładowo, atrybut `NumerMieszkania` (będący składową adresu) ma zastosowanie tylko w przypadku pracowników mieszkających w blokach, a nie np. w domach wolnostojących (np. domach jednorodzinnych). Podobnie, atrybut `StopnieNaukowe` ma zastosowanie tylko w przypadku osób, które ukończyły wyższe studia. W tego typu sytuacjach tworzona jest specjalna **wartość pusta** (ang. *null value*). Adres domu jednorodzinnego będzie zawierał taką wartość w swoim atrybucie `NumerMieszkania`, natomiast osoba, która nie ukończyła wyższych studiów, będzie miała wartość zerową w atrybucie `StopnieNaukowe`. Wartość pusta może także być stosowana dla atrybutów, których wartości w przypadku pewnych encji po prostu nie są znane — przykładowo, jeśli nie znamy numeru telefonu domowego Jana Nowaka z rysunku 3.3, możemy przypisać atrybutowi `TelefonDomowy` reprezentującej go encji wartość pustą. Znaczenie tej wartości we wcześniejszych przykładach to *nie dotyczy*, natomiast w ostatnim przykładzie jej znaczenie to *brak danych*. Kategorię *brak danych* dla wartości pustych można jeszcze rozbić na dwa przypadki. Pierwszy z nich dotyczy sytuacji, w której wiadomo, że wartość danego atrybutu istnieje, ale została *pominięta* — przykładowo, może to dotyczyć pustej wartości atrybutu `Wzrost` w encji reprezentującej osobę. Z drugim przypadkiem mamy do czynienia wtedy, gdy *nie wiadomo*, czy wartość danego atrybutu w ogóle istnieje — może to mieć miejsce np. w odniesieniu do pustej wartości wspomnianego atrybutu `TelefonDomowy`.

**Skomplikowane atrybuty.** Warto pamiętać, że atrybuty złożone i atrybuty wielowartościowe mogą być w dowolny sposób zagnieżdżane. Możemy reprezentować dowolne zagnieżdżenia np. przez grupowanie składników atrybutu złożonego w nawiasach okrągłych, przez oddzielanie tych składników za pomocą przecinków i przez umieszczanie atrybutów wielowartościowych w nawiasach klamrowych. Takie atrybuty są nazywane **atrybutami skomplikowanymi**. Przykładowo, jeśli jakaś osoba może mieć więcej niż jedno miejsce zamieszkania i każde z tych mieszkań bądź domów ma podłączonych wiele linii telefonicznych, atrybut `TelefonDomowy` w encji reprezentującej tę osobę może mieć postać podobną do tej z rysunku 3.5<sup>3</sup>. `Telefon` i `Adres` są tu atrybutami złożonymi.

```
{NumerTelefonu({Telefon(KodObszaru,Numer)},Adres(AdresUlicy
(Numer,Ulica,NumerMieszkania),Miasto,Województwo,Kod))}
```

Rysunek 3.5. Skomplikowany atrybut `TelefonDomowy`

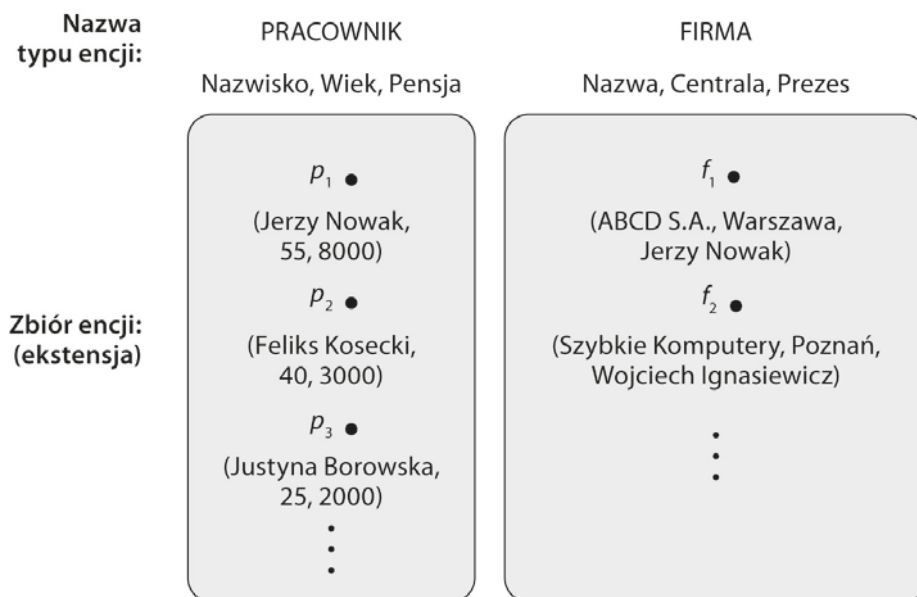
### 3.3.2. Typy encji, zbiory encji, klucze i zbiory wartości

**Typy encji i zbiory encji.** Bazy danych częstokroć zawierają grupy podobnych encji. Przykładowo, firma zatrudniająca setki pracowników może przechowywać bardzo podobne informacje dla każdego z tych pracowników. Wiele encji reprezentujących tych pracowników ma takie same wartości w niektórych swoich atrybutach, jednak każda z nich utrzymuje *własną kopię tej wartości* dla każdego atrybutu. **Typ encji** definiuje *zbiór (kolekcję)* encji, które mają takie same atrybuty. Każdy typ encji w bazie danych jest opisywany za pomocą jego nazwy i atrybutów. Na rysunku 3.6 przedstawiono dwa typy encji (nazwane `PRACOWNIK`

<sup>3</sup> Osoby dobrze znające język XML z pewnością zauważyły, że skomplikowane atrybuty są podobne do występujących w tym języku elementów złożonych (patrz rozdział 13.).



i FIRMA) oraz listę wartości przypisanych do zawartych w nich atrybutach. Na tym samym rysunku zaprezentowano także kilka pojedynczych encji (wraz z wartościami ich atrybutów) dla obu zademonstrowanych typów. Wyznaczony w dowolnym punkcie czasu zbiór wszystkich encji konkretnego typu w bazie danych jest nazywany **zbiorem encji** — zbiór encji ma zwykle taką samą nazwę jak odpowiedni typ encji. Przykładowo, nazwa PRACOWNIK odwołuje się zarówno do *typu encji*, jak i do bieżącego *zbioru wszystkich encji reprezentujących pracowników* w bazie danych. Obecnie częściej typom encji i zbiorom encji przypisywane są różne nazwy. Dzieje się tak np. w obiektowych i obiektowo-relacyjnych modelach danych (patrz rozdział 12.).



RYСУNEK 3.6. Dwa typy encji (PRACOWNIK i FIRMA) wraz z kilkoma encjami składowymi

Typy encji są reprezentowane na diagramach ER<sup>4</sup> (patrz rysunek 3.2) w formie prostokątów z nazwami typów w środku. Nazwy atrybutów są umieszczane w elipsach dołączonych do typów encji za pomocą prostych linii. Także za pomocą prostych linii łączymy na diagramach atrybuty złożone z ich atrybutami składowymi. Atrybuty wielowartościowe są prezentowane w postaci podwójnych elips z nazwami w środku. Na rysunku 3.7(a) przedstawiony jest zapisany w tej notacji typ encji SAMOCHÓD.

Typ encji opisuje **schemat** i **intensję kolekcji encji**, które mają taką samą strukturę. Kolekcje encji konkretnego typu są grupowane w zbiory encji, które są czasami nazywane **ekstensjami** typu encji.

<sup>4</sup> Zastosowano tu notację diagramów ER, która jest zbliżona do proponowanej początkowo notacji oryginalnej (tzw. notacji Chena, 1976). Obecnie wykorzystuje się także wiele innych notacji. Niektóre z tych pozostałych notacji zilustrowaliśmy w dodatku A i w dalszej części tego rozdziału (podczas prezentowania diagramów klas języka UML).



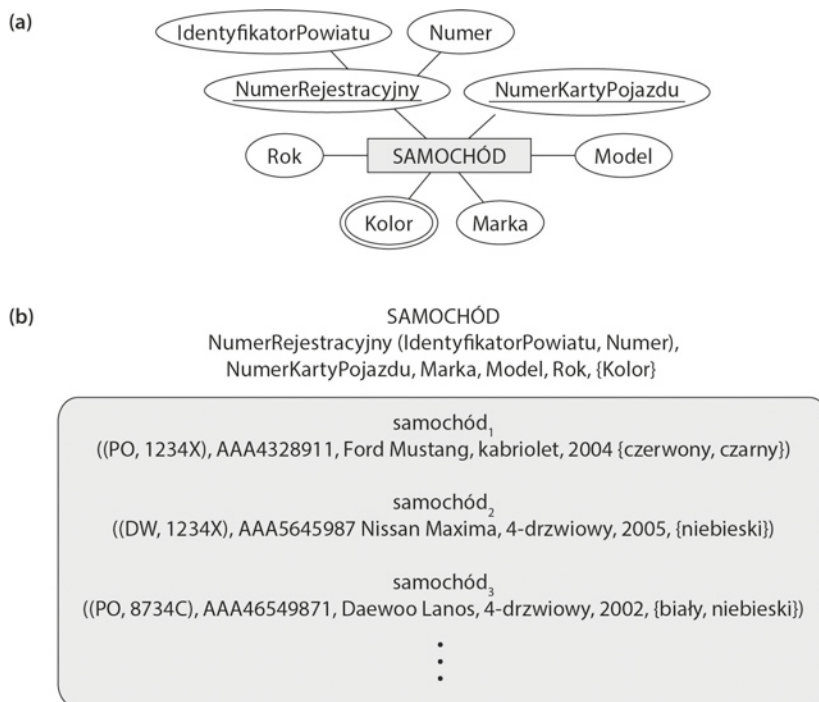
**Atrybuty klucza typu encji.** Ważnym ograniczeniem nakładanym na encje jednego typu jest **ograniczenie unikatowości** dla wskazanych atrybutów (tzw. **kluczy**). Większość typów encji zawiera atrybut, którego wartości są różne w każdej konkretnej encji danego zbioru. Taki atrybut jest nazywany **atrybutem klucza**, a jego wartości można wykorzystywać do unikatowego identyfikowania poszczególnych encji. Przykładowo, atrybut *Nazwa* jest kluczem typu encji *FIRMA* (patrz rysunek 3.6), ponieważ nie mogą istnieć dwie firmy o takiej samej nazwie. W przypadku typu encji *OSOBA* (lub *PRACOWNIK*) naturalnym atrybutem klucza jest *PESEL*. Niekiedy klucz tworzy kilka połączonych atrybutów — wówczas *kombinacja* wartości tych atrybutów musi być inna w każdej z encji. Jeśli dany zbiór atrybutów ma taką właściwość, prawidłowym sposobem jego reprezentowania w opisywanym w tym rozdziale modelu ER jest budowa *atrybutu złożonego* i wyznaczenie mu roli klucza w danym typie encji. Warto pamiętać, że taki klucz złożony musi być *minimalny*, co oznacza, że mogą należeć do niego tylko i wyłącznie te atrybuty składowe, które zapewniają własność unikatowości. W notacji diagramów ER każdy atrybut klucza jest oznaczany owalem z **podkreśloną** nazwą w środku (patrz rysunek 3.7(a)).

Wyznaczenie danego atrybutu do roli klucza w typie encji oznacza, że wspomniana już własność unikatowości musi być spełniona dla *każdego zbioru encji* danego typu. Mamy więc do czynienia z ograniczeniem, które zakazuje dwóm dowolnym encjom jednoczesnego reprezentowania tej samej wartości atrybutu klucza. Nie należy traktować tej własności jak części konkretnej ekstensji, ale raczej jak ograniczenie nałożone na *wszystkie ekstensje* danego typu encji. Ograniczenie unikatowości klucza (i wszystkie inne ograniczenia, które omówimy w dalszej części tego rozdziału) powinny wynikać z rzeczywistych reguł występujących w mini-świecie, który ma być reprezentowany przez projektowaną bazę danych.

Niektóre typy encji zawierają *więcej niż jeden* atrybut klucza. Przykładowo, każdy z pary atrybutów *NumerRejestracyjny* i *NumerKartyPojazdu* typu encji *SAMOCHÓD* (patrz rysunek 3.7) jest na swój sposób kluczem tego typu. Atrybut *NumerRejestracyjny* jest przykładem klucza złożonego, który składa się z dwóch atrybutów prostych (*IdentyfikatorPowiatu* oraz *Numer*), z których żaden w pojedynkę nie może tworzyć klucza (nie spełnia warunku unikatowości). Typ encji może również nie mieć *żadnego klucza* — wówczas jest nazywany *słabym typem encji* (patrz podrozdział 3.5).

Jeśli w stosowanej tu notacji diagramów dwa atrybuty są podkreślone osobno, to *każdy z nich stanowi samodzielny klucz*. W modelu ER (inaczej niż w modelu relacyjnym; patrz punkt 5.2.2) nie występuje klucz główny. Taki klucz jest wybierany na etapie odwzorowywania tego modelu na schemat relacyjny (patrz rozdział 9.).

**Zbiory wartości (dziedziny) atrybutów.** Każdy prosty atrybut typu encji jest powiązany z odpowiednim **zbiorem wartości** (lub **dziedziną** wartości), który — jak sama nazwa wskazuje — określa dopuszczalny zbiór wartości przypisywanych do tego atrybutu w poszczególnych encjach. Jeśli przedział wieku pracowników (patrz rysunek 3.6) musi się mieścić w przedziale od 16 do 70, powinniśmy zdefiniować zbiór wartości dla atrybutu *Wiek* typu encji *PRACOWNIK* jako zbiór liczb całkowitych większych lub równych 16 i mniejszych lub równych 70. Podobnie możemy określić zbiór wartości dla atrybutu *Nazwisko* jako ciągi znaków alfabetu oddzielonych spacjami. Zbiory wartości nie są prezentowane na diagramach związków encji (ER) — zwykle są definiowane w oparciu o podstawowe **typy danych** dostępne w większości języków programowania, takie jak liczby całkowite, ciągi



RYSUNEK 3.7. Typ encji SAMOCHÓD z dwoma atrybutami klucza — NumerRejestracyjny oraz NumerKartyPojazdu. (a) Notacja diagramów ER, (b) zbiór encji z trzema encjami

znaków, wartości logiczne, liczby zmiennoprzecinkowe, typy wyliczeniowe, podzakresy itp. Typy danych atrybutów można jednak przedstawić na UML-owych diagramach klas (patrz podrozdział 3.8) i w innych notacjach diagramów stosowanych w narzędziach do projektowania baz. Często wykorzystuje się także dodatkowe typy danych reprezentujące datę, czas i inne przydatne informacje.

Z matematycznego punktu widzenia, atrybut  $A$  typu encji  $E$ , którego zbiorem wartości jest  $V$ , można zdefiniować za pomocą **funkcji** z  $E$  do zbioru potęgowego<sup>5</sup>  $P(V)$  zbioru  $V$ :

$$A : E \rightarrow P(V)$$

Do wartości atrybutu  $A$  należącego do encji  $e$  odwołujemy się za pomocą  $A(e)$ . Powyższa definicja dotyczy zarówno atrybutów jednowartościowych i wielowartościowych, jak i atrybutów z wartością pustą. Wartość pusta jest reprezentowana przez *zbiór pusty*. W przypadku atrybutów jednowartościowych  $A(e)$  musi być *zbiorem jednoelementowym* dla każdej encji  $e$  typu  $E$ ; takiego ograniczenia oczywiście nie ma w przypadku atrybutów wielowartościowych. Zbiór wartości  $V$  dla złożonego atrybutu  $A$  jest iloczynem kartezjańskim zbiorów potęgowych  $P(V_1), P(V_2), \dots, P(V_n)$ , gdzie  $V_1, V_2, \dots, V_n$  są zbiorami wartości poszczególnych atrybutów składowych, które tworzą atrybut złożony  $A$ :

$$V = P(V_1) \times P(V_2) \times \dots \times P(V_n)$$

<sup>5</sup> **Zbiór potęgowy**  $P(V)$  zbioru  $V$  jest zbiorem wszystkich podzbiorów zbioru  $V$ .

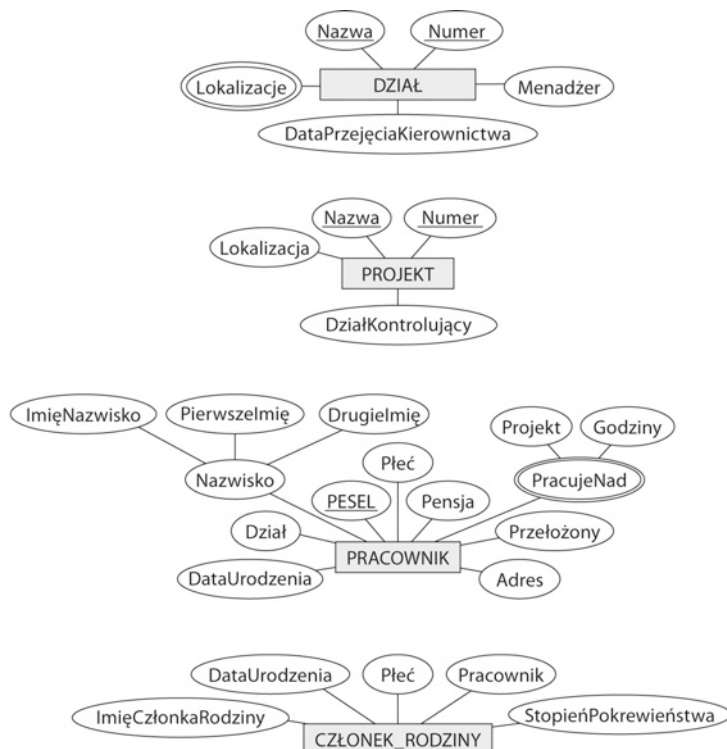
Zbiór wartości obejmuje wszystkie możliwe wartości. W danym momencie w bazie używana jest zwykle tylko niewielka ich część. Te wartości reprezentują aktualny stan mini-świata i odpowiadają danym z określonego momentu.

### 3.3.3. Początkowy projekt koncepcyjny bazy danych FIRMA

Możemy teraz zdefiniować typy encji dla bazy danych FIRMA w oparciu o wymagania opisane w podrozdziale 3.2. Po określeniu kilku niezbędnych typów encji wraz z ich atrybutami i wprowadzeniu pojęcia związku, spróbujemy (w podrozdziale 3.4) udoskonalić nasz projekt. Na bazie wymagań wymienionych w podrozdziale 3.2 możemy zidentyfikować cztery typy encji — po jednym dla każdego z czterech elementów tamtej specyfikacji (patrz rysunek 3.8):

- (1) Typ encji DZIAŁ z atrybutami Nazwa, Numer, Lokalizacja, Kierownik oraz DataPrzejęciaKierownictwa. Lokalizacja jest jedynym atrybutem wielowartościowym tego typu encji. Możemy wyznaczyć do roli klucza typu DZIAŁ albo atrybut Nazwa, albo atrybut Numer (ale nie oba jednocześnie), ponieważ każdy z nich musi być unikatowy.
- (2) Typ encji PROJEKT z atrybutami Nazwa, Numer, Lokalizacja oraz DziałNadzorujący. Podobnie jak w przypadku typu encji DZIAŁ możemy wyznaczyć do roli klucza albo atrybut Nazwa, albo atrybut Numer (ale nie oba jednocześnie).
- (3) Typ danych PRACOWNIK z atrybutami Nazwisko, PESEL, Płeć, Adres, Pensja, DataUrodzenia, Dział oraz Przełożony. Zarówno Nazwisko, jak i Adres może być atrybutem złożonym, jednak tego typu informacji nie określono w wymaganiach. Musimy wobec tego raz jeszcze spotkać się z przyszłymi użytkownikami projektowanej aplikacji bazy danych i zapytać ich, czy będą się odwoływali do atrybutu Nazwisko w oparciu o jego poszczególne składniki (Imię, InicjałDrImienia, Nazwisko), oraz czy będą traktowali adres pracownika jak jedną całość.
- (4) Typ encji CZŁONEK\_RODZINY z atrybutami Pracownik, ImięCzłonkaRodziny, Płeć, DataUrodzenia oraz StopieńPokrewieństwa (względem pracownika).

Inny wymóg dotyczy tego, że pracownik może realizować kilka projektów jednocześnie, a baza ma przechowywać liczbę godzin poświęcanych tygodniowo przez pracownika na realizację poszczególnych projektów. Te właściwości są częścią wymagań zdefiniowanych w punkcie 3. w podrozdziale 3.2 i mogą być reprezentowane w typie encji PRACOWNIK za pomocą wielowartościowego, złożonego atrybutu PracujeNad ze składnikami (Projekt, Godziny). Alternatywnym rozwiązaniem jest zastosowanie w typie encji PROJEKT wielowartościowego, złożonego atrybutu nazwanego Pracownicy, który będzie się składał z atrybutów prostych (Pracownik, Godziny). W projekcie przedstawionym na rysunku 3.8 wybraliśmy to pierwsze rozwiązanie. W następnym podrozdziale, po wprowadzeniu związków, atrybut ten zostanie przekształcony w związek wiele do wielu.



Rysunek 3.8. Wstępny projekt typów encji dla bazy danych FIRMA. Niektóre z pokazanych atrybutów zostaną przekształcone w związki

### 3.4. Typy związków, zbiory związków, role i ograniczenia strukturalne

Na rysunku 3.8 istnieje kilka *niejawnych związków* pomiędzy różnymi typami encji. W praktyce okazuje się, że za każdym razem, gdy atrybut jednego typu encji odwołuje się do innego typu encji, między taką parą typów musi istnieć jakiś związek. Przykładowo, atrybut *Kierownik* typu *DZIAŁ* odwołuje się do pracownika, który kieruje danym działem; atrybut *DziałNadzorujący* typu *PROJEKT* odwołuje się do działu, który odpowiada za realizację danego projektu; atrybut *Przełożony* typu *PRACOWNIK* odwołuje się do innego pracownika (który jest jego bezpośrednim przełożonym); atrybut *Dział* typu *PRACOWNIK* odwołuje się do działu, w którym dana osoba pracuje; itp. W modelu związków encji takie odwołania nie powinny być reprezentowane w formie atrybutów, tylko za pomocą odpowiednich **związków**. Schemat bazy danych *FIRMA* z rysunku 3.8 zostanie w podrozdziale 3.6 poprawiony w taki sposób, aby jawnie reprezentował związki występujące pomiędzy poszczególnymi typami encji. W początkowych projektach typów encji związki mają zazwyczaj postać atrybutów. Kiedy taki projekt jest później poprawiany, atrybuty tego typu przekształca się w związki pomiędzy typami encji.

Ten podrozdział został podzielony na następujące punkty. W punkcie 3.4.1 wprowadzono pojęcia typów związków, zbiorów związków oraz egzemplarzy związków. W dalszej części tego podrozdziału zdefiniowaliśmy pojęcia stopnia związku, nazw ról oraz związków rekurencyjnych (patrz punkt 3.4.2), a także omówiliśmy zagadnienia związane z nakładanymi na związki ograniczeniami strukturalnymi, jak choćby współczynniki licznosci czy warunki istnienia (patrz punkt 3.4.3). W punkcie 3.4.4 zaprezentowano sposób, w jaki można nadawać atrybuty poszczególnym typom związków.

### 3.4.1. Typy, zbiory i egzemplarze związków

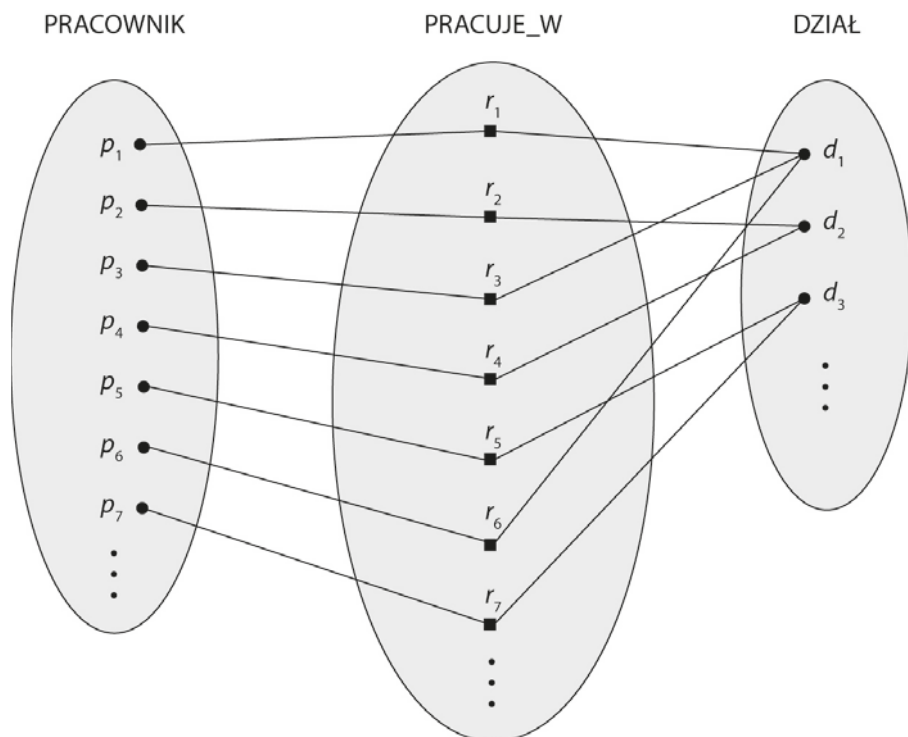
**Typ związku**  $R$  łączącej  $n$  typów encji ( $E_1, E_2, \dots, E_n$ ) definiuje zbiór powiązań (nazywany także **zbiorem związków**) pomiędzy encjami zgodnymi z tymi typami encji. Podobnie jak w przypadku typów i zbiorów encji, typy związków i odpowiadające im zbiory związków są zwyczajowo oznaczane tą samą nazwą,  $R$ . Z matematycznego punktu widzenia  $R$  jest zbiorem **egzemplarzy związku**  $r_i$ , gdzie każdy związek  $r_i$  łączy  $n$  pojedynczych encji ( $e_1, e_2, \dots, e_n$ ), a każda encja  $e_j$  w związku  $r_i$  jest typu  $E_j$ , gdzie  $1 \leq j \leq n$ . Oznacza to, że typ związku jest w istocie relacją matematyczną na typach  $E_1, E_2, \dots, E_n$  (alternatywnym rozwiązaniem jest zdefiniowanie typu związku w postaci podzbioru iloczynu kartezjańskiego  $E_1 \times E_2 \times \dots \times E_n$ ). O każdym z typów encji  $E_1, E_2, \dots, E_n$  możemy powiedzieć, że jest **elementem** związku typu  $R$ ; podobnie, o każdym ze związków  $e_1, e_2, \dots, e_n$  możemy powiedzieć, że jest **elementem** egzemplarza związku  $r_i = (e_1, e_2, \dots, e_n)$ .

Mówiąc nieformalnie, każdy egzemplarz  $r_i$  typu związku  $R$  jest takim powiązaniem encji, w którym występuje dokładnie jedna encja z każdego typu encji będącego elementem danego związku. Każdy taki egzemplarz związku  $r_i$  w rzeczywistości reprezentuje fakt jakiegoś powiązania (występującego w odpowiednim mini-świecie) między encjami będącymi elementami  $r_i$ . Przykładowo, przeanalizujmy typ związku PRACUJE\_W pomiędzy dwoma typami encji: PRACOWNIK i DZIAŁ. Typ związku PRACUJE\_W wiąże każdego pracownika z działem, w którym pracuje. Każdy egzemplarz związku w zbiorze PRACUJE\_W wiąże jedną encję reprezentującą pracownika z jedną encją reprezentującą dział firmy. Wspomniany przykład przedstawiono na rysunku 3.9, gdzie każdy egzemplarz związku  $r_i$  jest połączony z tymi encjami reprezentującymi pracownika i dział, które są elementami związku  $r_i$ . W reprezentowanym na rysunku 3.9 mini-świecie pracownicy  $p_1, p_3$  i  $p_6$  pracują w dziale  $d_1$ ; pracownicy  $p_2$  i  $p_4$  pracują w dziale  $d_2$ ; natomiast pracownicy  $p_5$  i  $p_7$  w dziale  $d_3$ .

Na diagramach związków encji (ER) typy związków są przedstawiane w postaci rombów połączonych prostymi liniami z prostokątami reprezentującymi poszczególne typy encji. Nazwa związku zawsze znajduje się wewnątrz reprezentującego ją rombu (patrz rysunek 3.2).

### 3.4.2. Stopień związku, nazwy ról oraz związki rekurencyjne

**Stopień typu związku.** Stopień typu związku jest liczbą typów encji składających się na dany związek. Oznacza to, że PRACUJE\_W jest związkiem drugiego stopnia. Typ związku drugiego stopnia jest nazywany typem **binarnym**, natomiast typ związku trzeciego stopnia (a więc obejmujący trzy typy encji) jest nazywany typem **trójskładnikowym** (ang. *ternary*).

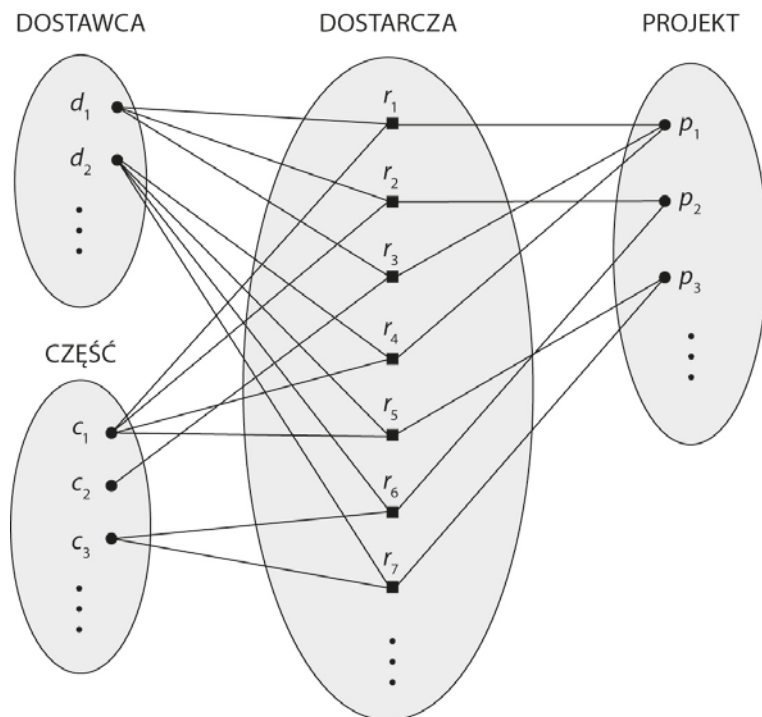


RYSUNEK 3.9. Wybrane egzemplarze zbioru związku PRACUJE\_W, który reprezentuje typ związku PRACUJE\_W pomiędzy typami encji PRACOWNIK i DZIAŁ

Przykładem związku trójskładnikowego jest przedstawiony na rysunku 3.10 typ DOSTARCZA, którego każdy egzemplarz  $r_i$  wiąże trzy encje — dostawcę  $d$ , część  $c$  oraz projekt  $p$  (taki związek występuje w sytuacji, gdy dostawca  $d$  dostarcza część  $c$  dla projektu  $p$ ). Związki mogą w zasadzie mieć dowolne stopnie, jednak najbardziej popularne są związki binarne. Należy pamiętać, że związki wyższych stopni są bardziej skomplikowane od związków binarnych (więcej szczegółów na ten temat można znaleźć w podrozdziale 3.9).

**Związki w postaci atrybutów.** W niektórych sytuacjach wygodnym rozwiązaniem jest traktowanie typów związków tak, jakby były atrybutami (a więc zgodnie z modelem, który omówiliśmy w podrozdziale 3.3.3). Przykładowo, przeanalizujmy raz jeszcze typ związku PRACUJE\_W z rysunku 3.9. Moglibyśmy w typie encji PRACOWNIK użyć atrybutu nazwanego Dział, którego wartość w każdej encji reprezentującej pracownika wskazywałaby na encję działu, w którym dana osoba pracuje. Oznaczałoby to, że zbiorem wartości dla atrybutu Dział byłby zbiór *wszystkich* encji typu DZIAŁ, czyli zbiór encji typu DZIAŁ. Dokładnie w ten sposób postąpiliśmy konstruując schemat przedstawiony na rysunku 3.8, który stanowił reprezentację naszego wstępnego projektu typu encji PRACOWNIK dla bazy danych FIRMA. Warto jednak pamiętać, że kiedy chcemy reprezentować związek binarny w postaci atrybutu, zawsze mamy do wyboru jedną z dwóch opcji. W tym przypadku rozwiązaniem alternatywnym jest stworzenie w typie encji DZIAŁ wielowartościowego atrybutu Pracownicy, którego wartości dla każdej encji reprezentującej dział stanowiłyby zbiór encji pracowników, którzy w danym dziale pracują. Zbiór wartości takiego atrybutu





RYSUNEK 3.10. Niektóre egzemplarze związków w trójskładnikowym zbiorze związku DOSTARCZA

Pracownicy jest więc zbiorem potęgowym zbioru encji PRACOWNIK. Związek PRACUJE\_W może być reprezentowana albo przez atrybut Dział typu encji PRACOWNIK, albo przez atrybut Pracownicy typu encji DZIAŁ. Gdybyśmy użyli obu reprezentacji, oznaczałoby to, że każda z nich jest odwrotnością drugiej<sup>6</sup>.

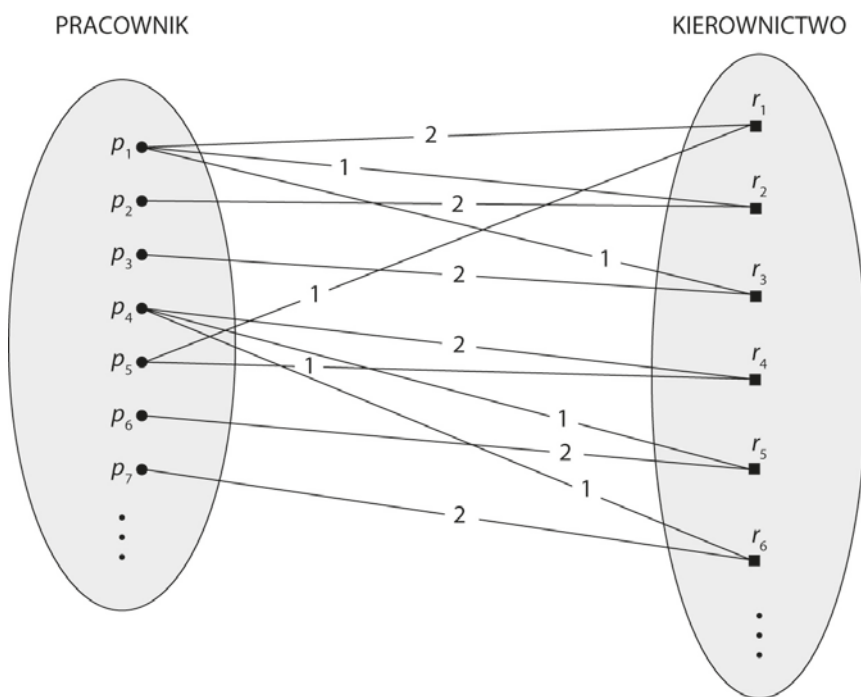
**Nazwy ról i związki rekurencyjne.** Każdy typ encji będący składnikiem jakiegoś typu związku pełni w tym związku konkretną rolę. **Nazwy ról** określają znaczenie poszczególnych encji (typów będących elementami składowymi związków) w każdym z egzemplarzy danego związku i znacznie ułatwiają interpretowanie rzeczywistego celu definiowania takich powiązań. Przykładowo, w typie związku PRACUJE\_W encje typu PRACOWNIK odgrywają rolę *pracowników* lub *pracobiorców*, natomiast encje typu DZIAŁ odgrywają rolę *działów* lub *pracodawców*.

Z technicznego punktu widzenia, nazwy ról nie są niezbędne (przynajmniej w sytuacji, gdy wszystkie składowe typy encji są różne), ponieważ można je zastąpić nazwami typów encji składających się na poszczególne związki. Warto jednak pamiętać, że w niektórych

<sup>6</sup> Pojęcie odwrotności reprezentacji typów związków opartej na atrybutach jest wykorzystywane w jednej z klas modeli danych (nazywanej **funkcjonalnymi modelami danych**). W obiektowych bazach danych (patrz rozdział 12.) związki mogą być reprezentowane za pomocą atrybutów referencyjnych albo w jednym kierunku, albo (na zasadzie odwrotności) w obu kierunkach. W relacyjnych bazach danych (patrz rozdział 5.) rolę atrybutów referencyjnych reprezentujących związki pełnią tzw. klucze obce.



przypadkach *ten sam* typ encji występuje w jednym typie związku więcej niż raz, za każdym razem w *innej roli*. Nazwy ról stają się wówczas podstawowym elementem odróżniającym poszczególne znaczenia tych udziałów. Takie typy związków są nazywane **związkami rekurencyjnymi**. Na rysunku 3.11 przedstawiono przykład związku rekurencyjnego. Typ związku KIEROWNICTWO wiąże pracownika z jego przełożonym, gdzie zarówno encje reprezentujące pracowników, jak i encje reprezentujące przełożonych są elementami tego samego typu encji — PRACOWNIK. Oznacza to, że typ encji PRACOWNIK występuje w związku KIEROWNICTWO dwukrotnie: raz w roli *przełożonego* (szefa) i raz w roli *podwładnego*. Każdy egzemplarz związku  $r_i$  typu KIEROWNICTWO łączy dwie encje reprezentujące pracowników  $p_j$  i  $p_k$ , z których jedna pełni rolę przełożonego, a druga — podwładnego. Na rysunku 3.11 linie oznaczone cyfrą 1 reprezentują rolę przełożonego, natomiast linie oznaczone cyfrą 2 — rolę podwładnego; zatem  $p_1$  kieruje  $p_2$  i  $p_3$ ,  $p_4$  kieruje  $p_6$  i  $p_7$ , a  $p_5$  kieruje  $p_1$  i  $p_4$ . W tym przykładzie każdemu egzemplarzowi związku muszą odpowiadać dwie linie — jedna oznaczona cyfrą 1 (przełożony) i jedna oznaczona cyfrą 2 (podwładny).



RYSUNEK 3.11. Związek rekurencyjny KIEROWNICTWO pomiędzy typem encji PRACOWNIK w roli przełożonego (1) oraz typem encji PRACOWNIK w roli podwładnego (2)

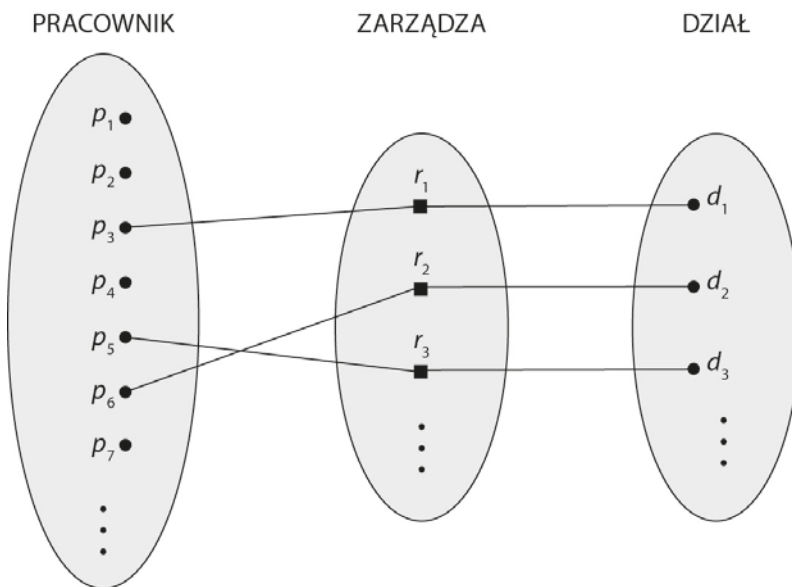
### 3.4.3. Ograniczenia dla typów związków

Typy związków mają zwykle definiowane pewne obostrzenia, które ograniczają możliwe kombinacje encji występujących w odpowiednich zbiorach związków. Takie ograniczenia wynikają z analizy sytuacji obserwowanych w reprezentowanym przez te związki miniświecie. Przykładowo, gdyby w modelowanej firmie istniała reguła określająca, że każdy

pracownik musi pracować w dokładnie jednym dziale, chcielibyśmy oczywiście takie wymaganie uwzględnić w schemacie zaprezentowanym na rysunku 3.9. Możemy wyróżnić dwa główne typy ograniczeń nakładanych na związki: *ograniczenia współczynnika licznosci* oraz *ograniczenia udziału*.

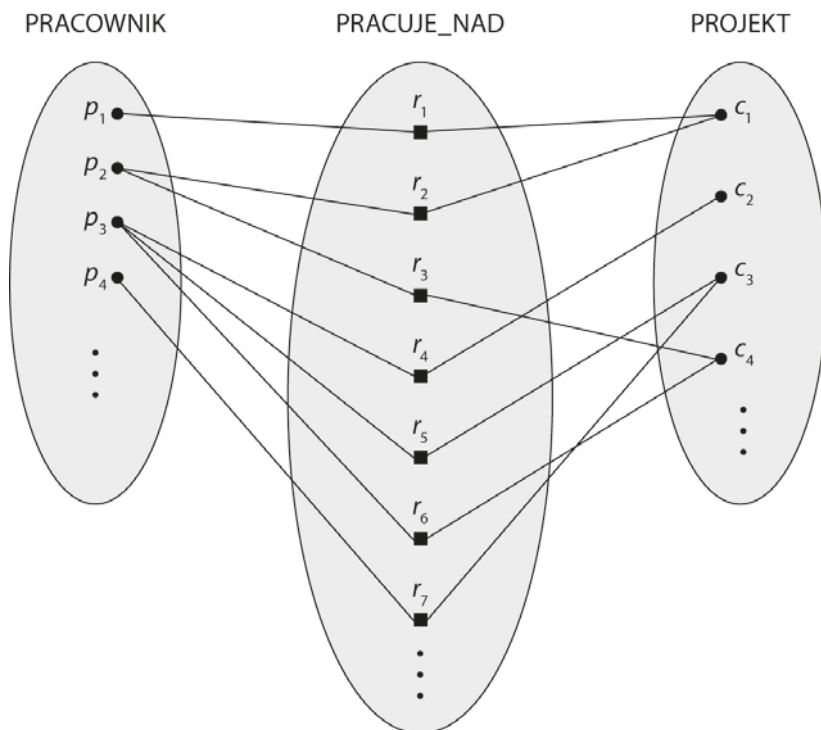
**Współczynniki licznosci dla związków binarnych.** Współczynnik licznosci dla związku binarnego określa *maksymalną* liczbę egzemplarzy tego związku, w których może występować pojedyncza encja. Przykładowo, w typie związku binarnego PRACUJE\_W współczynnik licznosci DZIAŁ:PRACOWNIK wynosi  $1:N$ , co oznacza, że każdy dział może być związany (może zatrudniać) dowolną liczbę pracowników ( $N$ )<sup>7</sup>, ale pojedynczy pracownik może być związany tylko z jednym działem (1). To oznacza, że w związku typu PRACUJE\_W konkretna encja działu może być powiązana z dowolną liczbą pracowników ( $N$  wskazuje na brak ograniczenia liczby maksymalnej). Istnieją następujące możliwe współczynniki licznosci dla typów związków binarnych:  $1:1$ ,  $1:N$ ,  $N:1$  oraz  $M:N$ .

Przykładem związku  $1:1$  jest KIERUJE (patrz rysunek 3.12), który wiąże encję reprezentującą dział z pracownikiem, który tym działem kieruje. W ten sposób możemy reprezentować istniejące w mini-świecie ograniczenie, które przewiduje, że w dowolnym punkcie czasu jeden pracownik może kierować tylko jednym działem i jeden dział może mieć tylko jednego kierownika. Współczynnik licznosci typu związku PRACUJE\_NAD (patrz rysunek 3.13) wynosi  $M:N$ , ponieważ odpowiednia reguła w mini-świecie mówi, że jeden pracownik może realizować wiele projektów, oraz że projekt może być realizowany przez wielu pracowników.



RYСУNEK 3.12. Związek KIERUJE typu  $1:1$

<sup>7</sup>  $N$  oznacza dowolną liczbę (zero lub więcej) powiązanych encji. W niektórych notacjach zamiast  $N$  używany jest symbol gwiazdki (\*).



RYСУNEK 3.13. Związek PRACUJE\_NAD typu M:N

Współczynniki liczności dla związków binarnych są reprezentowane na diagramach związków encji (ER) za pomocą oznaczeń 1,  $M$  i  $N$  przy rombikach reprezentujących typy związków (patrz rysunek 3.2). Zauważ, że w tej notacji można zapisać brak liczności maksymalnej ( $N$ ) lub licznosc maksymalną równą jeden (1). W innej notacji (patrz punkt 3.7.4) projektant może określić konkretną *licznosc maksymalną*, np. 4 lub 5.

**Ograniczenia udziału oraz zależność istnienia.** **Ograniczenia udziału** określają, czy istnienie danej encji jest uzależnione od tego, czy jest ona związana z inną encją za pośrednictwem związku danego typu. Takie ograniczenia wyznaczają *minimalną* liczbę egzemplarzy związku, w których każda z encji musi występować, zatem niekiedy są nazywane **ograniczeniami minimalnej liczności**. Istnieją dwa typy ograniczeń udziału — całkowite i częściowe — oba rodzaje zilustrujemy na przykładzie. Jeśli reguły przyjęte w danym przedsiębiorstwie określają, że *każdy* pracownik musi pracować w jakimś dziale, wówczas encja reprezentująca pracownika może istnieć tylko wtedy, gdy występuje przynajmniej w jednym egzemplarzu związku typu PRACUJE\_W (patrz rysunek 3.9). Występowanie encji typu PRACOWNIK w egzemplarzu związku typu PRACUJE\_W jest wówczas nazywane **pełnym udziałem** (lub **udziałem całkowitym**), ponieważ każda encja należaca do *kompletnego zbioru* encji reprezentujących pracowników musi być powiązana z encją reprezentującą dział za pośrednictwem PRACUJE\_W. Ograniczenie pełnego udziału jest także nazywane **zależnością istnienia**. Ze związku przedstawionego na rysunku 3.12 nie wynika, że każdy pracownik musi kierować działem, zatem udział encji typu PRACOWNIK w związku typu KIERUJE jest **częściowy**, ponieważ tylko niektóre encje reprezentujące pracowników

(wybrana część zbioru tych encji, a więc niekoniecznie wszystkie) są powiązane z encjami reprezentującymi działy za pośrednictwem związku KIERUJE. Kombinacje współczynników liczności i ograniczeń udziału są łącznie nazywane **ograniczeniami strukturalnymi** typów związków.

Na diagramach związków encji ograniczenia pełnego udziału (zależności istnienia) są prezentowane w postaci *podwójnych linii* łączących występujący w związku typ encji z samym związkiem, natomiast udział częściowy jest reprezentowany za pomocą *linii pojedynczej* (patrz rysunek 3.2). Zauważ, że w tej notacji można zapisać brak liczności minimalnej (udział częściowy) lub licznosc minimalną równą 1 (udział pełny). W innej notacji (patrz punkt 3.7.4) projektant może podać konkretną *licznosc minimalną* (np. 4 lub 5) dotyczącą udziału w związku.

Ograniczenia dotyczące związków wyższych stopni są opisane w podrozdziale 3.9.

### 3.4.4. Atrybuty typów związków

Typy związków mogą także mieć własne atrybuty, podobne do tych, które znamy z typów encji. Przykładowo, aby zarejestrować w bazie danych liczbę godzin poświęconych w ciągu tygodnia przez konkretnego pracownika na realizację konkretnego projektu, musimy dołączyć atrybut *Godziny* do typu związku PRACUJE\_NAD (patrz rysunek 3.13). Innym, podobnym przykładem takiego posunięcia jest dołączanie daty, od której dany kierownik przejął kierownictwo nad danym działem w oparciu o atrybut *DataPoczątkowa* dla typu związku KIERUJE (patrz rysunek 3.12).

Warto pamiętać, że atrybuty typów związków o współczynnikach licznosci równych 1:1 i 1:N mogą być przenoszone do jednego z występujących w tych związkach typów encji. Przykładowo, atrybut *DataPoczątkowa* typu związku KIERUJE może być równie dobrze atrybutem typu encji PRACOWNIK lub DZIAŁ, chociaż koncepcyjnie powinien należeć właśnie do typu związku KIERUJE. Wynika to z faktu, że KIERUJE jest związkiem 1:1, zatem każda encja reprezentująca dział lub pracownika występuje w pojedynczym egzemplarzu tego związku *najwyżej raz*. Oznacza to, że wartość atrybutu *DataPoczątkowa* może być określana albo przez występującą w związku encję działu, albo przez występującą w związku encję pracownika (kierownika).

W przypadku typów związków o współczynniku licznosci równym 1:N, atrybuty związku mogą być przenoszone *wyłącznie* do typów encji występujących po stronie licznosci *N*. Przykładowo, gdyby związek PRACUJE\_W z rysunku 3.9 zawierał dodatkowo atrybut *DataPoczątkowa* określający dzień, w którym pracownik rozpoczął pracę w danym dziale, moglibyśmy przenieść ten atrybut jedynie do typu encji PRACOWNIK. Wynika to z faktu, że każdy pracownik może pracować tylko w jednym dziale, zatem nie może występować w więcej niż jednym egzemplarzu związku PRACUJE\_W. Jednak dział może zatrudniać wielu pracowników, z których każdy rozpoczął pracę innego dnia. Zarówno w typach związków 1:1, jak i w typach związków 1:N, decyzja dotycząca miejsca przechowywania atrybutów opisujących związku (albo w postaci atrybutów typów związków, albo w postaci atrybutów występujących w tym związku typów encji) zależy od subiektywnej oceny projektanta schematu bazy danych.

W przypadku typów związków o współczynniku liczności równym  $M:N$ , niektóre atrybuty mogą być wyznaczane w oparciu o *kombinację encji* występujących w egzemplarzu związku, nie mogą natomiast być określone na bazie pojedynczej encji. Takie atrybuty *muszą być określone w postaci atrybutów związku*. Przykładem takiego rozwiązania jest atrybut Godziny związku PRACUJE\_NAD, którego liczność wynosi  $M:N$  (patrz rysunek 3.13) — liczba godzin poświęconych przez pracownika na realizację danego projektu jest określana w oparciu o kombinację pracownik-projekt i nie może być wyznaczana na bazie którejkolwiek z tych encji występującej osobno.

## 3.5. Słabe typy encji

Typy encji, które nie zawierają atrybutów klucza, są nazywane **słabymi typami encji**. I odwrotnie, **prawidłowe typy encji** ze zdefiniowanymi atrybutami klucza (ten warunek spełniają wszystkie prezentowane do tej pory przykłady) są określane mianem **silnych typów encji**. Encje słabych typów są identyfikowane w oparciu o kombinacje powiązań z encjami innych typów oraz wartości jednego z ich atrybutów. Te inne typy związane ze słabymi typami encji nazywamy **identyfikującymi** lub **właścicielskimi typami encji**<sup>8</sup>, natomiast typy związków wiążących słabe typy danych z ich właścicielami są nazywane **związkami identyfikującymi** te typy<sup>9</sup>. Słabe typy danych zawsze mają nałożone *ograniczenie pełnego udziału* (zależności istnienia) względem identyfikujących je związków, ponieważ słabe encje nie mogłyby być identyfikowane bez udziału encji właścicielskich. Nie zawsze jednak zastosowanie warunku zależności istnienia jest równoważne z definiowaniem słabego typu encji. Przykładowo, encja PRAWO\_JAZDY nie może istnieć bez powiązanej z nią encji OSOBA, chociaż ma swój własny, unikatowy klucz (NumerPrawaJazdy), a więc nie jest słabą encją.

Przeanalizujmy teraz powiązany z typem PRACOWNIK typ encji CZŁONEK\_RODZINY, który jest wykorzystywany do utrzymywania informacji o rodzinach poszczególnych pracowników w oparciu o związek  $1:N$  (patrz rysunek 3.2). Typ CZŁONEK\_RODZINY ma zdefiniowane następujące atrybuty: Imię (tylko pierwsze imię członka rodziny), DataUrodzenia, Płeć oraz Stopień ↗ Pokrewieństwa (względem pracownika). Nie można wykluczyć sytuacji, w której dwóch członków rodzin *dwóch różnych pracowników* będzie miało takie same wartości atrybutów Imię, DataUrodzenia, Płeć oraz Stopień ↗ Pokrewieństwa, nadal jednak obie osoby powinny być reprezentowane za pomocą dwóch różnych encji. Obie encje można od siebie odróżnić dopiero po określeniu *encji konkretnych pracowników*, które występują w związkach z encjami reprezentującymi członków rodzin pracowników. O każdej z encji pracowników możemy powiedzieć, że *jest właścicielem* powiązanych ze sobą encji członków rodzin.

Słabe typy encji zawierają zazwyczaj **klucze częściowe**, które mają postać zbioru atrybutów unikatowo identyfikujących słabe encje związane z *tą samą encją właścicielską*<sup>10</sup>. Jeśli w naszym przykładzie przyjmujemy, że wśród członków rodziny tego samego pracownika nie może istnieć para krewnych o takim samym imieniu, atrybut Imię typu encji

<sup>8</sup> Identyfikujący typ encji jest niekiedy nazywany także **macierzystym** lub **dominującym typem encji**.

<sup>9</sup> Słaby typ danych jest niekiedy nazywany **potomnym** lub **podległym typem danych**.

<sup>10</sup> Klucz częściowy jest niekiedy nazywany **dyskryminatorem**.

CZŁONEK\_RODZINY będzie kluczem częściowym tego typu. W najgorszym przypadku klucz częściowy będzie atrybutem złożonym ze *wszystkich atrybutów słabego typu encji*.

Na diagramach związków encji (ER) zarówno słabe typy encji, jak i ich związki identyfikujące są odróżniane od pozostałych elementów za pomocą podwójnych linii otaczających reprezentujące je odpowiednio prostokąty i romby (patrz rysunek 3.2). Klucz częściowy jest podkreślany za pomocą przerywanej lub kropkowanej linii.

Słabe typy encji mogą być niekiedy reprezentowane w postaci skomplikowanych (złożonych i wielowartościowych) atrybutów silnych typów encji. W jednym z przykładów mogliśmy zastosować wielowartościowy atrybut CzłonkowieRodziny typu encji PRACOWNIK, który obejmowałby atrybuty składowe Imię, DataUrodzenia, Płeć oraz StopieńPokrewieństwa. Wybór właściwej metody reprezentacji należy do projektanta bazy danych. Jednym z kryteriów stosowania słabych typów encji jest sytuacja, w której słaba encja występuje niezależnie w typach związków innych niż typ związku identyfikującego.

Generalnie możemy definiować dowolną liczbę poziomów słabych typów encji; także właścicielski typ encji może sam w sobie być słabym typem encji. Co więcej, słaby typ encji może mieć więcej niż jeden identyfikujący typ encji oraz typ związku identyfikującego stopnia wyższego niż dwa (patrz rysunek w podrozdziale 3.9).

## 3.6. Udoskonalanie projektu ER dla bazy danych FIRMA

Możemy teraz przystąpić do udoskonalania naszego projektu bazy danych z rysunku 3.8, które będzie polegało na zamianie atrybutów reprezentujących związki w typy związków. Współczynnik liczności i ograniczenia udziału każdego z nowych typów związków będą określone na podstawie listy wymagań przedstawionej w podrozdziale 3.2. Jeśli któregoś z tych współczynników lub ograniczeń nie będziemy mogli określić w oparciu o te wymagania, musimy zwrócić się do przyszłych użytkowników z prośbą o doprecyzowanie odpowiednich ograniczeń strukturalnych.

W naszym przykładzie zdefiniowaliśmy następujące typy związków:

- KIERUJE, typ związku 1:1 łączący typy encji PRACOWNIK i DZIAŁ. Udział typu encji PRACOWNIK w tym typie związku jest częściowy. Zadaliśmy przyszłym użytkownikom końcowym dodatkowe pytanie; z uzyskanej odpowiedzi wynika, że każdy dział przez cały czas musi mieć przydzielonego kierownika, co oznacza, że mamy do czynienia z pełnym udziałem typu związku<sup>11</sup>. Do tego typu związku dodatkowo przypisano atrybut DataPoczątkowa.
- PRACUJE\_W, typ związku 1:N łączący typy encji DZIAŁ i PRACOWNIK. W obu przypadkach są to udziały pełne.
- NADZORUJE, typ związku 1:N łączący typy encji DZIAŁ i PROJEKT. Udział typu encji PROJEKT jest pełny, natomiast udział typu encji DZIAŁ został określony przez użytkowników końcowych jako częściowy (w trakcie konsultacji przyszli użytkownicy stwierdzili, że niektóre działy nie muszą realizować żadnych projektów).

---

<sup>11</sup> Istniejące w mini-świecie reguły określające tego typu ograniczenia są niekiedy nazywane *regułami biznesowymi*, ponieważ są definiowane przez jednostki „biznesowe” lub organizacje, które będą wykorzystywały projektowaną bazę danych.



- KIERUJE, typ związku 1:N łączący typu encji PRACOWNIK (w roli przełożonego) z tym samym typem encji (w roli podwładnego). Udział obu występujących w tym związku encji jest częściowy, ponieważ — jak powiedzieli nam przyszli użytkownicy końcowi tworzonej aplikacji bazy danych — nie każdy pracownik jest przełożonym i nie każdy pracownik ma przełożonego.
- PRACUJE\_NAD — po tym, jak przyszli użytkownicy stwierdzili, że nad pojedynczym projektem może pracować wielu użytkowników, uznaliśmy, że ten typ związku powinien zawierać atrybut Godziny, a jego licznosc powinna wynosić  $M:N$ . Udziały obu typów encji (PRACOWNIK i PROJEKT) w tym typie związku są pełne.
- JEST\_NA\_UTRZYMANIU, typ związku 1:N łączący typy encji PRACOWNIK i CZŁONEK\_RODZINY, który dodatkowo identyfikuje słaby typ encji CZŁONEK\_RODZINY. Udział typu encji PRACOWNIK jest częściowy, natomiast udział typu encji CZŁONEK\_RODZINY jest pełny.

Po określeniu wymienionych powyżej sześciu typów związków, powinniśmy usunąć z typów encji przedstawionych na rysunku 3.8 wszystkie atrybuty, które zostały przekształcone w odpowiednie związki. Chodzi o takie atrybuty jak Kierownik i DataPrzejęcia ↗ Kierownictwa z typu encji DZIAŁ; DziałNadzorujący z typu encji PROJEKT; Dział, Przełożony i PracujeNad z typu encji PRACOWNIK oraz Pracownik z typu encji CZŁONEK\_RODZINY. Bardzo ważne jest, aby podczas projektowania koncepcyjnego schematu bazy danych maksymalnie ograniczać ewentualną nadmiarowość. Jeśli z jakiegoś powodu nadmiarowość jest pożądana na poziomie składowania danych lub na poziomie perspektyw użytkownika, można ją wprowadzić później (patrz punkt 1.6.1).

## 3.7. Diagramy ER, konwencje nazewnictwa oraz zagadnienia związane z projektowaniem

### 3.7.1. Podsumowanie notacji diagramów związków encji (ER)

Na rysunkach od 3.9 do 3.13 zilustrowano przykłady występowania typów encji w różnych typach związków w oparciu o prezentacje pojedynczych egzemplarzy encji i związków należących odpowiednio do właściwych zbiorów encji i związków. Na diagramach związków encji główny nacisk kładzie się raczej na reprezentowanie ogólnych schematów, nie konkretnych egzemplarzy encji czy związków. Takie rozwiązanie jest bardziej praktyczne podczas projektowania baz danych, ponieważ schematy stosunkowo rzadko ulegają zmianom, natomiast zawartość zbiorów encji jest poddawana częstym modyfikacjom. Schemat jest ponadto łatwiejszy (w porównaniu z danymi) do przedstawiania w postaci diagramów, ponieważ jest po prostu znacznie mniejszy.

Na rysunku 3.2 przedstawiono **schemat związków encji w bazie danych FIRMA** w postaci **diagramu ER**. Dalej dokonamy pełnego przeglądu notacji tego typu diagramów. Zwykle (silne) typy encji jak PRACOWNIK, DZIAŁ czy PROJEKT są reprezentowane w postaci prostokątów. Typy związków (w tym PRACUJE\_W, KIERUJE, NADZORUJE oraz PRACUJE\_NAD) mają postać rombów połączonych za pomocą prostych linii z odpowiednimi typami encji. Atrybuty zostały przedstawione w formie elips — każdy tak reprezentowany atrybut jest połączony



z odpowiednim typem encji lub związku za pomocą linii prostej. Atrybuty składające się na atrybuty złożone są połączone z elipsami reprezentującymi właściwe atrybuty złożone (tak jest np. w przypadku atrybutu *Nazwisko* typu encji *PRACOWNIK*). Atrybuty wielowartościowe są reprezentowane za pomocą podwójnych elips (patrz atrybut *Lokalizacja* typu danych *DZIAŁ*). Nazwy atrybutów klucza są na diagramach ER podkreślane. Atrybuty pochodne są przedstawiane w postaci kropkowanych elips (przykładem takiej reprezentacji jest atrybut *LiczbaPracowników* typu danych *DZIAŁ*).

Słabe typy encji są odróżniane od silnych typów przez umieszczanie ich nazw w podwójnych prostokątach oraz stosowanie podwójnych rombów do reprezentowania właściwych związków identyfikujących (przykładem takiej reprezentacji jest typ encji *CZŁONEK* → *RODZINY* oraz identyfikujący typ związku *JEST\_NA\_UTRZYMANIU*). Klucze częściowe słabych typów encji są podkreślane za pomocą kropkowanych linii.

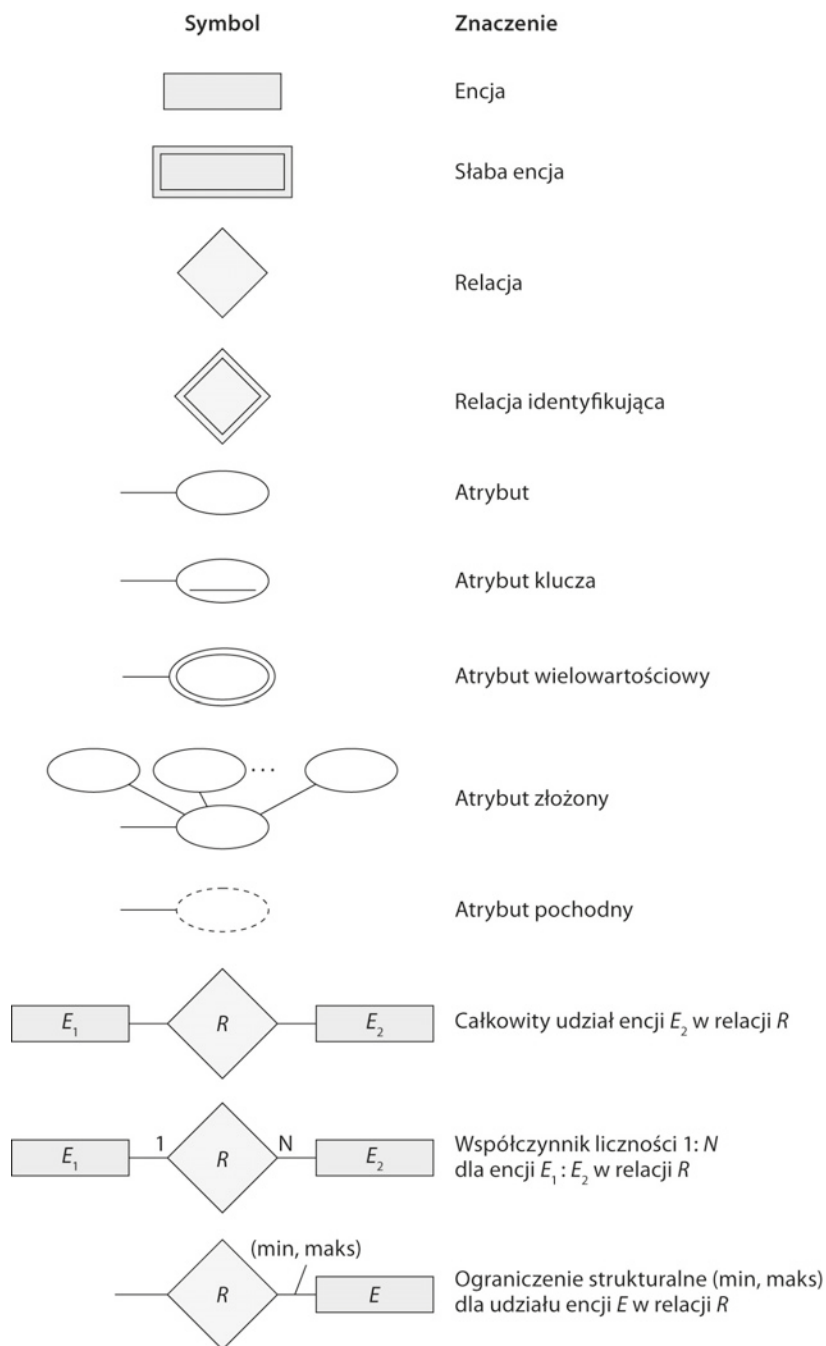
Na rysunku 3.2 współczynniki licznosci poszczególnych *binarnych* typów związków są reprezentowane za pomocą oznaczeń 1, *M* lub *N* przy każdej krawędzi odpowiadającej udziałowi w poszczególnych związkach. Współczynnik licznosci typu związku *KIERUJE* pomiędzy typami encji *DZIAŁ:PRACOWNIK* wynosi 1:1, 1:*N* w przypadku typu związku *PRACUJE\_W* pomiędzy typami encji *DZIAŁ:PRACOWNIK* oraz *M:N* w przypadku typu związku *PRACUJE\_NAD*. Ograniczenia udziału są reprezentowane przez pojedyncze linie (w przypadku udziału częściowego) oraz przez podwójne linie (w przypadku udziału pełnego, nazywanego także zależnością istnienia).

Na rysunku 3.2 zaprezentowano nazwy ról dla typu związku *KIERUJE*, ponieważ występujący w nim typ encji *PRACOWNIK* odgrywa obie role w tak zdefiniowanym związku binarnym. Warto pamiętać, że współczynnik licznosci tego typu związku wynosi 1:*N* (odpowiednio dla przełożonego i podwładnych), ponieważ każdy pracownik w roli podwładnego może mieć najwyżej jednego bezpośredniego przełożonego, natomiast pracownik w roli przełożonego może kierować dowolną liczbą pracowników.

Na rysunku 3.14 przedstawiono podsumowanie konwencji stosowanych podczas opracowywania diagramów związków encji. Warto zauważyć, że istnieje też wiele innych notacji do tworzenia diagramów (patrz punkt 3.7.4 i dodatek A).

### 3.7.2. Prawidłowe nazewnictwo konstrukcji schematu

Okazuje się, że dobór nazw typów encji, atrybutów, typów związków i (szczególnie) ról podczas projektowania schematu bazy danych nie zawsze jest oczywisty. Zasadniczo powinniśmy używać takich nazw, których znaczenie w jak największym stopniu będzie odpowiadało przeznaczeniu poszczególnych konstrukcji w tworzonym schemacie. W przypadku typów encji zdecydowaliśmy się stosować *nazwy w liczbie pojedynczej* (zamiast nazw w liczbie mnogiej), ponieważ każda taka nazwa jest wykorzystywana do określania poszczególnych egzemplarzy encji danego typu. W naszych diagramach związków encji będziemy postępowali zgodnie z konwencją, która mówi, że nazwy typów encji i związków powinny być pisane wielkimi literami, nazwy atrybutów powinny się rozpoczynać od wielkich liter, natomiast nazwy ról powinny być pisane małymi literami. Powyższą konwencję stosowaliśmy już na rysunku 3.2.



Rysunek 3.14. Podsumowanie notacji diagramów związków encji (ER)

Generalnie, z praktycznego punktu widzenia dobrym rozwiązaniem jest przekształcanie opisu wymagań zdefiniowanych dla bazy danych w następujący sposób: występujące tam *rzeczowniki* powinny tworzyć nazwy typów encji, natomiast występujące w specy-

fikacji wymagań *czasowniki* powinny w projektowanym schemacie pełnić role nazw typów związków. Nazwy atrybutów zwykle tworzy się na podstawie dodatkowych rzeczowników opisujących rzeczowniki określające typy encji.

Kolejnym elementem nazewnictwa wymagającym rozważenia jest takie oznaczanie związków binarnych, które zapewni możliwość łatwego odczytywania diagramów związków encji od lewej strony do prawej i z góry do dołu. Zgodnie z tą konwencją nadawaliśmy nazwy typom związków prezentowanym na rysunku 3.2. Aby lepiej wyjaśnić znaczenie takiego nazywania związków, na rysunku 3.2 wprowadziliśmy jeden wyjątek od tej reguły — chodzi o typ związku `JEST_NA_UTRZYMANIU`, który należy odczytywać od dołu do góry. Kiedy opisujemy ten związek, możemy powiedzieć, że `CZŁONEK_RODZINY` (ten typ encji znajduje się na dole) `JEST_NA_UTRZYMANIU` (to nazwa związku) encji `PRACOWNIK` (ten typ encji znajduje się na górze). Aby zmienić sposób odczytywania tego związku na zgodny ze wspomnianą konwencją (z góry na dół), moglibyśmy zmodyfikować jego nazwę na `MA_NA_UTRZYMANIU` — wówczas moglibyśmy odczytywać ten fragment diagramu ER w następujący sposób: encja `PRACOWNIK` (typu przedstawionego na górze) `MA_NA_UTRZYMANIU` (nazwa związku) encję `CZŁONEK_RODZINY` (typu przedstawionego na dole). Warto pamiętać, że takie rozwiązanie nie zawsze jest możliwe — problem dotyczy sytuacji, w których każdy ze związków binarnych może być opisywany począwszy od dowolnego z pary występujących w niej typów encji (patrz początek podrozdziału 3.4).

### 3.7.3. Decyzje projektowe związane z tworzeniem schematu koncepcyjnego ER

W niektórych przypadkach podjęcie właściwej decyzji dotyczącej modelowania konkretnego elementu mini-świata w formie właściwego typu encji, atrybutu lub typu związku może być trudne. W tym podrozdziale spróbujemy przedstawić kilka krótkich wskazówek związanych z doбором odpowiednich konstrukcji w rozmaitych sytuacjach.

Zasadniczo projektowanie schematu powinno być traktowane jako iteracyjny proces udoskonalania, gdzie najpierw tworzony jest projekt początkowy, który następnie podlega iteracyjnym udoskonaleniom aż do otrzymania najlepszego z możliwych rozwiązań. Do najczęściej stosowanych działań udoskonalających należą poniższe:

- Pojęcie może najpierw zostać zamodelowane w postaci atrybutu i dopiero potem zostać przekształcone w związek, gdy okaże się, że dany atrybut jest w istocie odwołaniem do innego typu encji. Często zdarza się, że para takich atrybutów (stanowiących wzajemne odwrotności) jest przekształcana w związek binarny. Takie udoskonalenia schematów szczegółowo omówiliśmy w podrozdziale 3.6. Warto zauważyć, że w stosowanej tu notacji po zastąpieniu atrybutu przez związek należy usunąć ten atrybut z typu encji, aby uniknąć duplikatów i nadmiarowości.
- Podobnie atrybut występujący w wielu typach encji może zostać przeniesiony do niezależnego typu encji. Przykładowo, przypuśćmy, że w naszym początkowym projekcie schematu wiele typów encji w bazie danych `UNIwersytet` (takich jak `STUDENT`, `WYKŁADOWCA` czy `PRZEDMIOT`) zawiera atrybut `Wydział`; projektant może wówczas podjąć decyzję o utworzeniu typu encji `WYDZIAŁ` z pojedynczym atrybutem `NazwaWydziału` i powiązać go za pomocą odpowiednich związków z wymienionymi

trzema typami encji (STUDENT, WYKŁADOWCA oraz PRZEDMIOT). Pozostałe atrybuty tego typu encji i (lub) związki z jego udziałem mogą być odkrywane w kolejnych iteracjach projektu budowania schematu.

- Można stosować także udoskonalenia odwrotne względem opisanego powyżej przypadku — przykładowo, jeśli w projekcie początkowym istnieje typ encji WYDZIAŁ z pojedynczym atrybutem NazwaWydziału, który występuje w związku tylko z jednym z pozostałych typów encji (np. z typem STUDENT), możemy zdegradować typ encji WYDZIAŁ do roli atrybutu typu encji STUDENT.
- W podrozdziale 3.9 opiszemy wybory związane ze stopniem związków. W rozdziale 4. omówimy udoskonalenia dotyczące specjalizacji i generalizacji.

### 3.7.4. Notacje alternatywne względem tradycyjnych diagramów związków encji (ER)

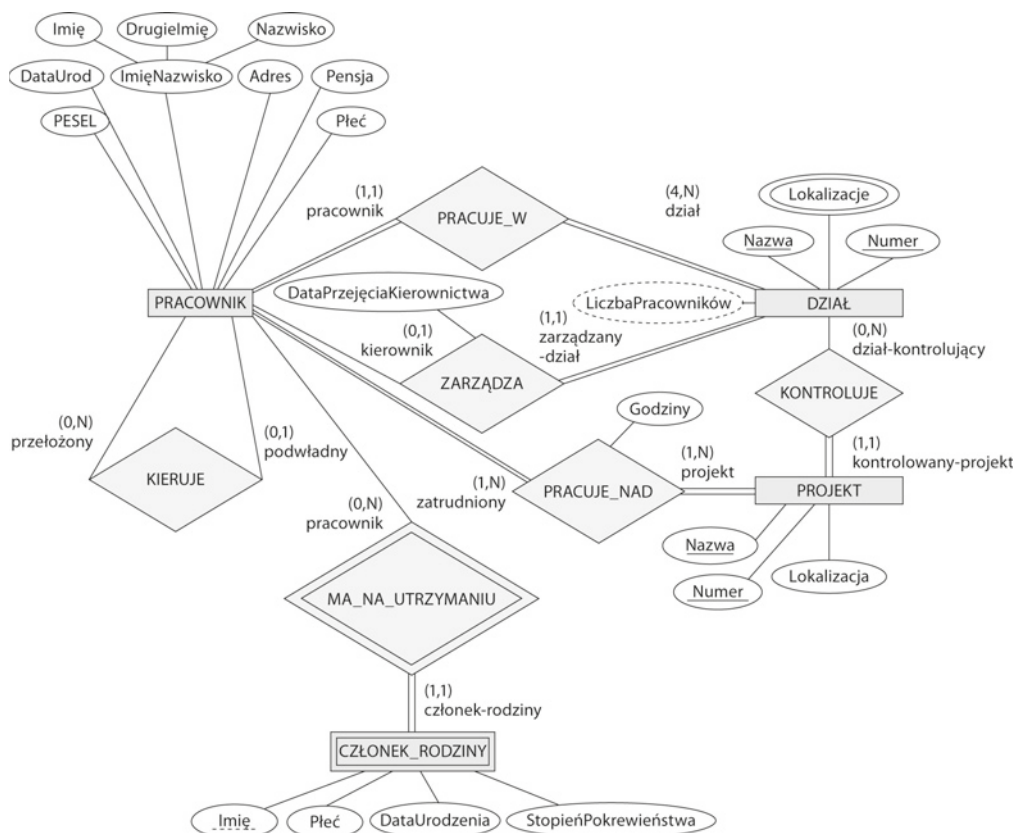
Istnieje wiele alternatywnych notacji stosowanych do przedstawiania diagramów związków encji. W dodatku A przedstawiono najbardziej popularne notacje diagramów tego typu. W podrozdziale 3.8 wprowadzimy notację języka *UML* (ang. *Universal Modeling Language*) dla diagramów klas, która została zaproponowana do roli standardu koncepcyjnego modelowania obiektów.

W tym podrozdziale opiszemy jedną alternatywę dla tradycyjnej notacji diagramów ER, która umożliwia określanie ograniczeń strukturalnych dla projektowanych związków. Nie obejmuje ona współczynników licznosci (1:1, 1:N, M:N) ani pojedynczych i podwójnych linii związanych z ograniczeniami udziału. Prezentowana notacja wymaga od projektanta przypisania do każdego wystąpienia typu encji *E* w typie związku *R* pary liczb całkowitych (*min*, *maks*), gdzie  $0 \leq \text{min} \leq \text{maks}$  oraz  $\text{maks} \geq 1$ . Te liczby określają, że w dowolnym momencie każda encja *e* typu *E* musi występować przynajmniej *min* i najwyżej *maks* razy w egzemplarzach typu związku *R*. W tej metodzie dopuszczalna równość *min* = 0 oznacza udział częściowy, natomiast warunek *min* > 0 oznacza udział pełny.

Na rysunku 3.15 przedstawiono schemat bazy danych FIRMA opracowany zgodnie z notacją (*min*, *maks*)<sup>12</sup>. Zwykle stosuje się albo notację ze współczynnikami licznosci oraz pojedynczymi i podwójnymi liniami, albo notację (*min*, *maks*). Notacja (*min*, *maks*) jest bardziej precyzyjna i umożliwia łatwe określanie ograniczeń strukturalnych dla typów związków *dowolnego stopnia*. Reguły tej notacji nie stanowią jednak wystarczającego narzędzia do definiowania niektórych z kluczowych ograniczeń związków wyższego stopnia (patrz podrozdział 3.9).

Na rysunku 3.15 przedstawiono także wszystkie nazwy ról dla schematu bazy danych FIRMA.

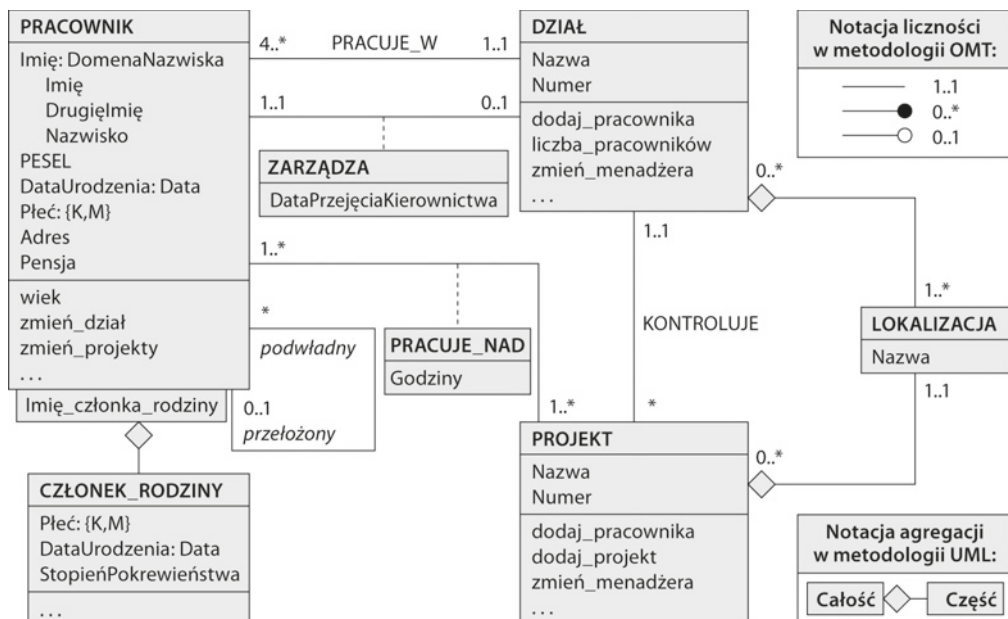
<sup>12</sup> W niektórych notacjach (szczególnie tych, które wykorzystuje się w metodach modelowania obiektowego, takich jak np. UML) wartości (*min*, *maks*) są umieszczane po przeciwnej stronie względem układu zastosowanego na rysunku 3.15. Przykładowo, w przypadku relacji PRACUJE\_W z rysunku 3.15 oznaczenie (1, 1) znajdowałoby się po stronie typu encji DZIAŁ, natomiast oznaczenie (4, N) zostałoby umieszczone przy typie encji PRACOWNIK. W tej książce stosujemy oryginalną notację opisaną przez Abrial'a (1974).



RYСУNEK 3.15. Diagramy ER dla schematu bazy danych FIRMA z ograniczeniami strukturalnymi zdefiniowanymi zgodnie z notacją (min, maks)

## 3.8. Przykładowa inna notacja: diagramy klas UML

Metoda oparta na UML-u jest coraz częściej wykorzystywana podczas projektowania oprogramowania i oferuje wiele typów diagramów stosowanych w różnych fazach procesów projektowych. W tym podrozdziale omówimy tylko ogólne cechy **diagramów klas UML** i porównamy to podejście z diagramami związków encji (ER). Diagramy klas można w niektórych przypadkach traktować jak alternatywę dla notacji diagramów ER. Dodatkowe notacje i zagadnienia związane z językiem UML zostaną zaprezentowane w podrozdziale 8.6. Na rysunku 3.16 zademonstrowano sposób, w jaki schemat bazy danych FIRMA z rysunku 3.15 (stworzony zgodnie z notacją ER) można przedstawić zgodnie z notacją diagramu klas języka UML. Typy encji z rysunku 3.15 zostały zamodelowane w postaci klas na rysunku 3.16. Encja w modelu ER odpowiada obiektowi w języku UML.



RYSUNEK 3.16. Schemat koncepcyjny bazy danych FIRMA w notacji diagramu klas języka UML

Na diagramach klas UML **klasa** (przypomina ona typ encji z diagramów ER) jest reprezentowana za pomocą prostokąta (patrz rysunek 3.16) podzielonego na trzy części. Górna część zawiera **nazwę klasy** (przypomina nazwę typu encji), część środkowa zawiera **atrybuty** pojedynczych obiektów danej klasy, natomiast dolna część zawiera **operacje**, które mogą być stosowane dla tych obiektów (zbliżonych do poszczególnych encji z ich zbioru). Łatwo zauważyć, że operacje *nie* są definiowane na diagramach ER. Przyjrzyjmy się klasie PRACOWNIK z rysunku 3.16. Klasa zawiera następujące atrybuty: Nazwisko, Pesel, DataUr, Płeć, Adres oraz Pensja. Projektant klasy może — w razie konieczności — określić opcjonalną **domenę** (typ danych) atrybutu przez umieszczenie po nazwie atrybutu znaku dwukropka i nazwy lub opisu domeny (jak w przypadku widocznych na rysunku 3.16 atrybutów Nazwisko, Płeć i DataUr klasy PRACOWNIK). Atrybuty złożone są modelowane w postaci **domen strukturalnych** (przykładem jest atrybut Nazwisko klasy PRACOWNIK). Atrybuty wielowartościowe są modelowane w postaci osobnych klas (przykładem takiego rozwiązania jest widoczna na rysunku 3.16 klasa LOKALIZACJA).

Typy związków są w terminologii UML nazywane **powiązaniem**, natomiast egzemplarze związków noszą nazwę **łączy**. **Powiązanie binarne** (binarny typ związku) jest reprezentowane za pomocą linii łączącej powiązane klasy (występujące w związku typy encji) i opcjonalnie może mieć nazwę. Atrybut związku, nazywany **atrybutem powiązania**, jest umieszczany w prostokącie połączonym z linią reprezentującą powiązanie za pomocą dodatkowej linii kreskowanej. Opisana w punkcie 3.7.4 notacja (min, maks) jest wykorzystywana do określania ograniczeń związku nazywanych w terminologii UML **licznością**. Na diagramach klas języka UML licznosc definiuje się w postaci oznaczeń *min..maks*, natomiast znak gwiazdki (\*) reprezentuje brak ograniczenia maksimum. Należy jednak pamiętać, że w porównaniu z notacją omawianą w punkcie 3.7.4 oznaczenia licznosci na diagramach klas UML umieszcza się *na przeciwnym końcu związku* (warto teraz porównać



odpowiednie reprezentacje na rysunkach 3.15 i 3.16). W języku UML pojedyncza gwiazdka oznacza licznosc 0..\*, natomiast pojedyncza wartosc 1 oznacza licznosc 1..1. Związki rekurencyjne (patrz punkt 3.4.2) są w terminologii UML nazywane **powiązaniem zwrotnym**, natomiast nazwy ról — podobnie jak w przypadku licznosci — są umieszczane na przeciwnych końcach powiązań (przynajmniej w porównaniu z rozmieszczeniem nazw odpowiednich ról na rysunku 3.15).

W języku UML istnieją dwa typy związków: powiązania i agregacje. **Agregacja** ma w założeniu reprezentować związek pomiędzy całym obiektem a jego częściami składowymi — do reprezentowania tego typu związków wykorzystuje się odmienną notację. Na rysunku 3.16 zamodelowano wiele lokalizacji jednego działu i pojedyncze lokalizacje poszczególnych projektów właśnie w postaci agregacji. Agregacje i powiązania nie różnią się jednak pod względem własności strukturalnych, a wybór typu związku zależy od subiektywnej oceny projektanta. W modelu związków encji (ER) oba rodzaje powiązań są reprezentowane za pomocą związków.

Język UML dodatkowo odróżnia powiązania (lub agregacje) **jednokierunkowe** i **dwukierunkowe**. W powiązaniach jednokierunkowych linia łącząca klasy jest oznaczana strzałką pozwalającą określić jedyny kierunek powiązania (co ma zasadnicze znaczenie podczas uzyskiwania dostępu do powiązanych obiektów). Jeśli linia reprezentująca powiązanie nie jest zakończona grotem strzałki, przyjmuje się, że mamy do czynienia z (domyślnym) powiązaniem dwukierunkowym. Przykładowo, jeśli uznamy, że w każdej sytuacji będziemy się odwoływali do kierownika działu za pośrednictwem obiektu reprezentującego ten dział (obiektu klasy DZIAŁ), powinniśmy narysować linię reprezentującą powiązanie KIERUJE z postaci strzałki skierowanej od klasy DZIAŁ do klasy PRACOWNIK. Egzemplarze związku mogą dodatkowo być oznaczane jako **uporządkowane**. Przykładowo, możemy określić, że obiekty reprezentujące pracowników, które występują w związku z odpowiednimi działami za pośrednictwem powiązania PRACUJE\_W, powinny być uporządkowane według wartości ich atrybutu DataPoczątkowa. Nazwy powiązań (związków) w notacji UML są *opcjonalne*, natomiast atrybuty związków powinny być umieszczane wewnątrz prostokątów dołączanych do linii reprezentujących powiązania lub agregacje za pomocą linii kreskowanych (patrz atrybuty DataPoczątkowa i Godziny na rysunku 3.16).

Definiowane w poszczególnych klasach listy operacji wynikają wprost z wymagań funkcjonalnych określonych dla projektowanej aplikacji (patrz podrozdział 3.1). W zasadzie początkowo wystarczy samo określenie nazw operacji dla logicznych działań, które mają być stosowane dla poszczególnych obiektów projektowanej klasy (patrz rysunek 3.16). W kolejnych iteracjach procesu udoskonalania schematu można dodawać coraz więcej szczegółów, takich jak precyzyjne typy argumentów (parametrów) oraz opisy funkcjonalne dla poszczególnych operacji. Język UML definiuje specjalne *opisy funkcji* i *diagramy sekwencji*, za pomocą których można wygodnie reprezentować niektóre szczegóły związane z operacjami, które jednak wykraczają poza zakres tej książki.

Słabe encje mogą być modelowane w oparciu o konstrukcje nazywane w języku UML **powiązaniem kwalifikowanym** (lub **agregacją kwalifikowaną**) — można za ich pomocą reprezentować zarówno związki identyfikujące, jak i klucze częściowe (umieszczane w prostokątach dołączanych do klas właścicielskich). Przykładem takiej reprezentacji na rysunku 3.16 jest klasa CZŁONEK\_RODZINY wraz z jej kwalifikowaną agregacją względem klasy PRACOWNIK. Klucz częściowy ImięCzłonkaRodziny jest tu nazywany **dys-**



**kryminatorem**, ponieważ jego wartość odróżnia obiekty klasy CZŁONEK\_RODZINY powiązane z tym samym obiektem klasy PRACOWNIK. Powiązania kwalifikowane nie muszą być stosowane wyłącznie do modelowania słabych encji — język UML przewiduje możliwość ich wykorzystywania także do modelowania innych sytuacji.

Ten podrozdział nie ma stanowić kompletnego omówienia diagramów klas UML, a tylko zilustrować popularną alternatywną notację do tworzenia diagramów, którą można zastosować do reprezentowania zagadnień z obszaru modelowania ER.

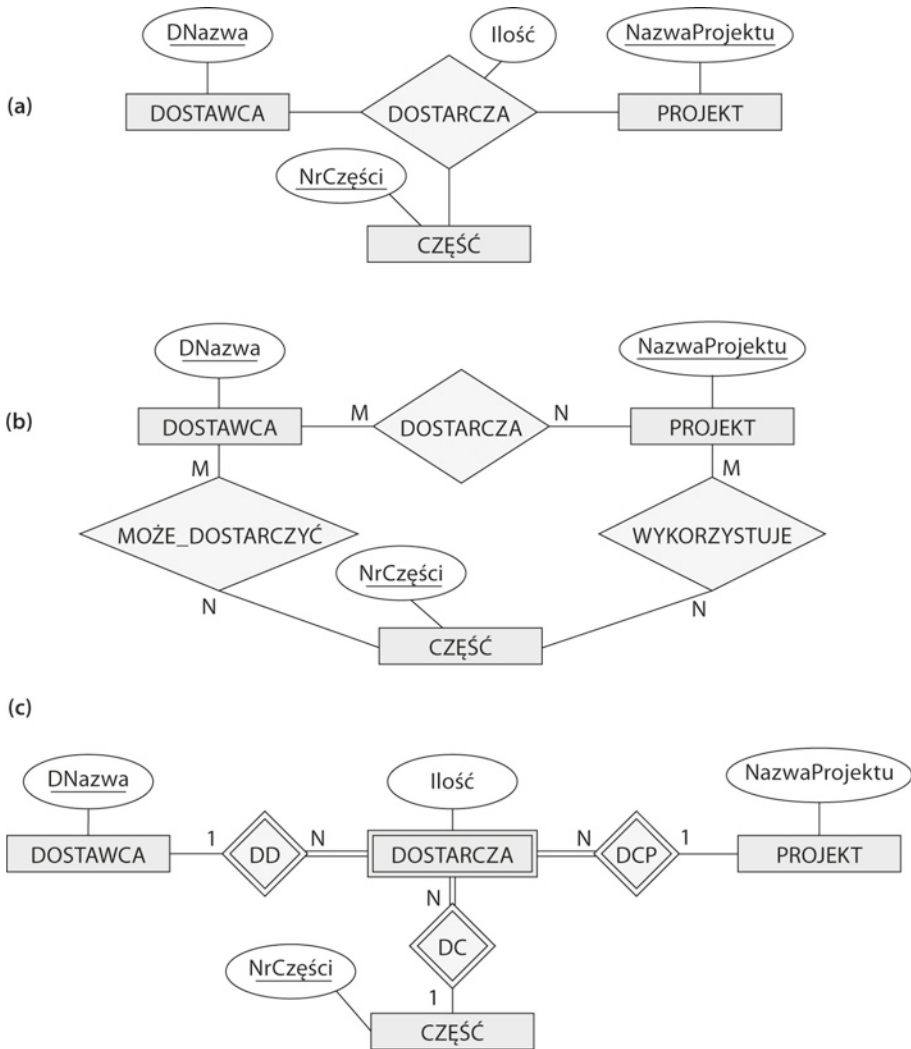
## 3.9. Typy związków stopnia wyższego niż drugi

W punkcie 3.4.2 zdefiniowaliśmy **stopień** typu związku jako liczbę typów encji występujących w tym typie; nazwaliśmy także typ związku drugiego stopnia związkiem *binarnym* oraz typ związku trzeciego stopnia związkiem *trójskładnikowym*. W tym podrozdziale przeanalizujemy różnice pomiędzy związkami binarnymi a związkami wyższych stopni, przedstawimy metody oceny właściwego rozwiązania w poszczególnych sytuacjach oraz omówimy ograniczenia dla związków wyższych stopni.

### 3.9.1. Wybór pomiędzy związkami binarnymi a trójskładnikowymi (lub wyższych stopni)

Na rysunku 3.17(a) przedstawiono notację diagramów ER dla typów związków trójskładnikowych — rysunek ilustruje schemat dla typu związku DOSTARCZA, który na poziomie poszczególnych egzemplarzy przedstawiono na rysunku 3.10. Warto pamiętać, że zbiór związku typu DOSTARCZA jest w istocie zbiorem egzemplarzy związku  $(d, c, p)$ , gdzie  $s$  jest encją typu DOSTAWCA, który aktualnie dostarcza CZĘŚĆ  $c$  dla encji typu PROJEKT  $p$ . Ogólnie rzecz biorąc, typ związku  $R$  stopnia  $n$ -tego będzie na diagramie ER połączony z typami encji za pomocą  $n$  krawędzi — po jednej dla każdego typu encji występującego w danym typie związku  $R$ .

Na rysunku 3.17(b) przedstawiono diagram ER dla trzech typów związków binarnych: MOŻE\_DOSTARCZYĆ, WYKORZYSTUJE oraz DOSTARCZA. Zasadniczo, typ związku trójskładnikowego reprezentuje nieco inne informacje niż te reprezentowane przez trzy typy związków binarnych. Przeanalizujemy na przykład wspomniane trzy typy związków binarnych (MOŻE\_DOSTARCZYĆ, WYKORZYSTUJE i DOSTARCZA). Przypuśćmy, że typ związku MOŻE\_DOSTARCZYĆ pomiędzy typami encji DOSTAWCA i CZĘŚĆ zawiera egzemplarz  $(d, c)$  za każdym razem, gdy dostawca  $d$  może dostarczyć część  $c$  (dla dowolnego projektu); przypuśćmy także, że typ związku WYKORZYSTUJE pomiędzy typami encji PROJEKT i CZĘŚĆ zawiera egzemplarz  $(p, c)$  za każdym razem, gdy w projekcie  $p$  jest wykorzystywana część  $c$ ; i wreszcie przyjmijmy, że typ związku DOSTARCZA pomiędzy typami encji DOSTAWCA i PROJEKT zawiera egzemplarz  $(d, p)$  za każdym razem, gdy dostawca  $d$  dostarcza jakąś część dla projektu  $p$ . Istnienie trzech egzemplarzy związku  $(d, c)$ ,  $(p, c)$  oraz  $(d, p)$  odpowiednio dla typów encji MOŻE\_DOSTARCZYĆ, WYKORZYSTUJE i DOSTARCZA wcale nie musi oznaczać, że istnieje egzemplarz  $(d, c, p)$  typu związku trójskładnikowego DOSTARCZA, ponieważ *znaczenie obu wystąpień jest nieco inne*. Decydowanie o tym, czy dany związek powinien być reprezentowany w postaci typu związku stopnia  $n$ -tego, czy może powinien zostać rozbity na wiele typów związków



RYSUNEK 3.17. Typy związków trójskładnikowych. (a) Związek DOSTARCZA. (b) Trzy związki binarne, które nie są równoważne związkowi DOSTARCZA. (c) Związek DOSTARCZA reprezentowana w postaci słabego typu encji

niższych stopni, jest w wielu przypadkach procesem dosyć skomplikowanym. Projektant schematu bazy danych musi podejmować taką decyzję w oparciu o semantykę lub znaczenie konkretnych sytuacji, które mają być reprezentowane w określony sposób. Jeśli okaże się, że oba wspomniane rozwiązania reprezentują inne znaczenia i oba te znaczenia muszą być uwzględniane w projektowanej aplikacji, typowym rozwiązaniem jest stworzenie *zarówno* związku trójskładnikowego, *jak i* jednego lub więcej związków binarnych.

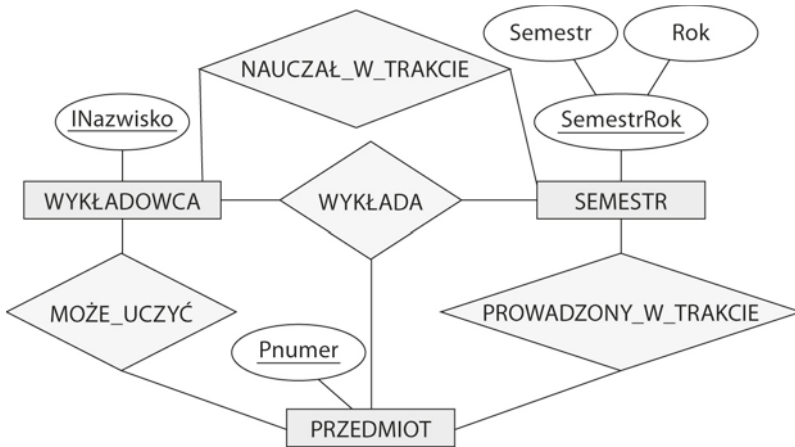
Niektóre z narzędzi wspomagających projektowanie baz danych bazują na różnych odmianach modelu ER, które zezwalają wyłącznie na stosowanie związków binarnych. W takim przypadku związek trójskładnikowy (jak choćby DOSTARCZA) musi być reprezentowany za pomocą słabego typu encji (bez klucza częściowego) oraz trzech związków identyfi-

kujących. Trójka typów encji występujących w tych związkach identyfikujących (DOSTAWCA, CZĘŚĆ i PROJEKT) stanowi łącznie właścicielskie typy encji (patrz rysunek 3.17(c)). Oznacza to, że każda encja widocznego na rysunku 3.17(c) słabego typu encji DOSTARCZA jest identyfikowana przez kombinację trzech encji właścicielskich typów DOSTAWCA, CZĘŚĆ oraz PROJEKT.

Związek trójskładnikowy można też reprezentować jako zwykły typ encji, stosując klucz sztuczny. W omawianym przykładzie dla typu encji dostawcy można zastosować atrybut klucza IdentyfikatorDostawcy i przekształcić ten typ w zwykły typ encji. Trzy binarne związki N:1 będą wtedy łączyć typ DOSTAWCA z trzema powiązanymi typami encji.

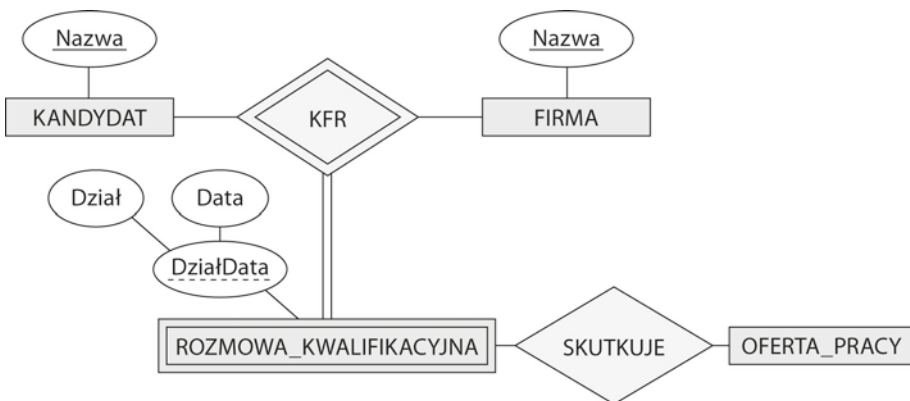
Na rysunku 3.18 przedstawiono jeszcze jeden przykład omawianych zagadnień. Tu typ związku WYKŁADA reprezentuje informacje na temat wykładowców prowadzących zajęcia z rozmaitych przedmiotów w poszczególnych semestrach roku akademickiego — każdy egzemplarz tego typu związku ma więc postać trójki  $(w, p, s)$  reprezentującej sytuację, w której WYKŁADOWCA  $w$  prowadzi zajęcia z przedmiotu  $p$  (egzemplarz typu encji PRZEDMIOT) w semestrze  $s$  (egzemplarz typu encji SEMESTR). Przedstawiona na rysunku 3.18 trójka typów związków binarnych ma następujące znaczenia: MOŻE\_UCZYĆ wiąże przedmiot z wykładowcami, którzy *mogą prowadzić zajęcia* z danego przedmiotu; NAUCZANY\_W\_TRAKCIE wiąże semestr z wykładowcami, którzy w trakcie danego semestru prowadzą zajęcia z *jakiegoś przedmiotu*; i wreszcie typ związku PROWADZONY\_W\_TRAKCIE wiąże semestr z przedmiotami, które są nauczane w danym semestrze *przez kogoś z wykładowców*. Także w tym przypadku przedstawiona trójka związków binarnych i związek trójskładnikowy reprezentują różne informacje, co nie zmienia faktu, że pomiędzy tymi związkami istnieją pewne ścisłe powiązania. Przykładowo, egzemplarz  $(w, p, s)$  typu związku WYKŁADA w ogóle nie powinien istnieć, *jeśli nie istnieje* egzemplarz  $(w, p)$  typu związku NAUCZANY\_W\_TRAKCIE, egzemplarz  $(p, s)$  typu związku PROWADZONY\_W\_TRAKCIE oraz egzemplarz  $(w, s)$  typu związku MOŻE\_UCZYĆ. Odwrotność tej implikacji nie zawsze jednak jest prawdziwa — mogą przecież istnieć egzemplarze  $(w, p)$ ,  $(p, s)$  i  $(w, s)$  trzech typów związków binarnych przy jednoczesnym braku egzemplarza  $(w, p, s)$  typu związku trójskładnikowego WYKŁADA. Warto pamiętać, że w prezentowanym przykładzie możemy (w oparciu o same znaczenia przedstawionych związków) wnioskować o istnieniu egzemplarzy związków NAUCZANY\_W\_TRAKCIE oraz PROWADZONY\_W\_TRAKCIE na bazie egzemplarzy typu związku WYKŁADA, ale nie możemy zastosować tej samej metody dla typu związku MOŻE\_UCZYĆ; oznacza to, że typy związków NAUCZANY\_W\_TRAKCIE oraz PROWADZONY\_W\_TRAKCIE są nadmiarowe i mogą zostać usunięte ze schematu i z diagramu.

Chociaż generalnie trzy związki binarne nie mogą zastępować pojedynczego związku trójskładnikowego, można uzyskać pożądany efekt przy pewnych *dodatkowych ograniczeniach*. W naszym przykładzie moglibyśmy na przykład przyjąć, że liczność typu związku MOŻE\_UCZYĆ wynosi 1:1 (jeden wykładowca może prowadzić zajęcia z jednego przedmiotu, jeden przedmiot może być nauczany tylko przez jednego wykładowcę) — wówczas możliwa jest rezygnacja z typu związku trójskładnikowego WYKŁADA, ponieważ istnienie jego encji można bez trudu wnioskować z trzech pozostałych typów związków binarnych (MOŻE\_UCZYĆ, NAUCZANY\_W\_TRAKCIE oraz PROWADZONY\_W\_TRAKCIE). Projektant schematu musi poddać dokładnej analizie znaczenia poszczególnych sytuacji, aby podjąć właściwe decyzje dotyczące potrzebnych typów związków binarnych i trójskładnikowych.



RYSUNEK 3.18. Jeszcze jeden przykład typu związku trójskładnikowego w zestawieniu z typami związków binarnych

Warto pamiętać, że istnieje także możliwość stworzenia słabego typu encji z identyfikującym typem związku trójskładnikowego (lub  $n$ -składnikowego). W takim przypadku stworzony w ten sposób słaby typ encji może mieć *wiele* właścicielskich typów encji. Przykład takiego rozwiązania przedstawiono na rysunku 3.19. W tym przykładzie pokazany jest fragment bazy, która rejestruje kandydatów biorących udział w rozmowach kwalifikacyjnych w różnych firmach. Ta baza może być częścią bazy agencji pracy. Zgodnie z wymaganiami kandydat może uczestniczyć w kilku rozmowach w tej samej firmie (np. w różnych działach lub dniach), ale oferta jest składana tylko w powiązaniu z jedną z rozmów. Tu ROZMOWA\_KWALIFIKACYJNA to słaby typ encji o dwóch typach właścicielskich (KANDYDAT i FIRMA) i z kluczem częściowym DziałData. Encja ROZMOWA\_KWALIFIKACYJNA jest jednoznacznie identyfikowana na podstawie kandydata, firmy i połączenia daty oraz działu.



RYSUNEK 3.19. Słaby typ encji ROZMOWA\_KWALIFIKACYJNA z identyfikującym typem związku trójskładnikowego

### 3.9.2. Ograniczenia związków trójskładnikowych (i wyższych stopni)

Istnieją dwie notacje określające sposób przedstawiania ograniczeń strukturalnych nakładanych na związki  $n$ -składnikowe — każda z tych notacji dotyczy innych ograniczeń. Jeśli więc istotne znaczenie ma określenie ograniczeń strukturalnych dla związków trójskładnikowych lub wyższego stopnia, powinniśmy stosować *obie notacje jednocześnie*. Pierwsza notacja opiera się na zaprezentowanym na rysunku 3.2 sposobie graficznego prezentowania współczynników liczności związków binarnych. W związkach wyższych stopni oznaczenia 1,  $M$  i  $N$  są umieszczane przy każdym łuku reprezentującym występowanie encji w związku (zarówno oznaczenie  $M$ , jak i oznaczenie  $N$  symbolizuje *wiele* lub *dowolną liczbę*)<sup>13</sup>. Spróbujmy teraz zilustrować to ograniczenie na przykładzie związku DOSTARCZA z rysunku 3.17.

Zbiór związków typu DOSTARCZA jest w istocie zbiorem egzemplarzy związku  $(d, c, p)$ , gdzie  $s$  jest encją typu DOSTAWCA, który aktualnie dostarcza CZĘŚĆ  $c$  dla encji typu PROJEKT  $p$ . Przypuśćmy, że istnieje ograniczenie dla konkretnej kombinacji projektu i części, które mówi, że można korzystać z usług tylko jednego dostawcy (tylko jeden dostawca może dostarczać określoną część dla określonego projektu). W takim przypadku powinniśmy na rysunku 3.17 umieścić symbol 1 przy linii oznaczającej udział typu encji DOSTAWCA oraz symbole  $M$  i  $N$  przy liniach reprezentujących występowanie w tym związku typów encji PROJEKT i CZĘŚĆ. W ten sposób określamy, że konkretna kombinacja egzemplarzy  $(p, c)$  może występować w danym zbiorze związków najwyżej raz, ponieważ każda taka para (projekt, część) unikatowo wskazuje na pojedynczego dostawcę. Oznacza to, że każdy egzemplarz związku  $(d, c, p)$  jest unikatowo identyfikowany w zbiorze związków przez połączenie  $(p, c)$ , co czyni z pary  $(p, c)$  naturalny klucz dla tego zbioru związków. W tej notacji udziały typów encji oznaczone symbolem 1 nie muszą być uwzględniane w zdefiniowanym dla danego zbioru związków kluczu identyfikującym<sup>14</sup>. Jeśli liczność każdego z trzech elementów wynosi  $M$  lub  $N$ , klucz jest połączeniem wszystkich trzech encji.

Druga notacja opiera się na oznaczeniach (*min*, *maks*) — przykład zastosowania tych oznaczeń dla związków binarnych przedstawiono na rysunku 3.15. Para wartości (*min*, *maks*) umieszczona przy linii reprezentującej udział typu encji w typie związku oznacza, że każda encja jest powiązana z co najmniej *min* i najwyżej *maks egzemplarzami związku* w danym zbiorze związków. Ograniczenia tego typu nie mają co prawda wpływu na sposób dobierania kluczy związków  $n$ -składnikowych (dla  $n > 2$ )<sup>15</sup>, ale określają wartości graniczne dla liczb egzemplarzy związków, w których mogą występować poszczególne encje.

<sup>13</sup> Ta notacja umożliwia nam także określanie klucza *powiązań związków* (patrz omówienie odpowiednich zagadnień w rozdziale 9.).

<sup>14</sup> To samo dotyczy współczynników liczności związków binarnych.

<sup>15</sup> Ograniczenia w postaci par wartości (*min*, *maks*) mogą natomiast wyznaczać klucze dla związków binarnych.

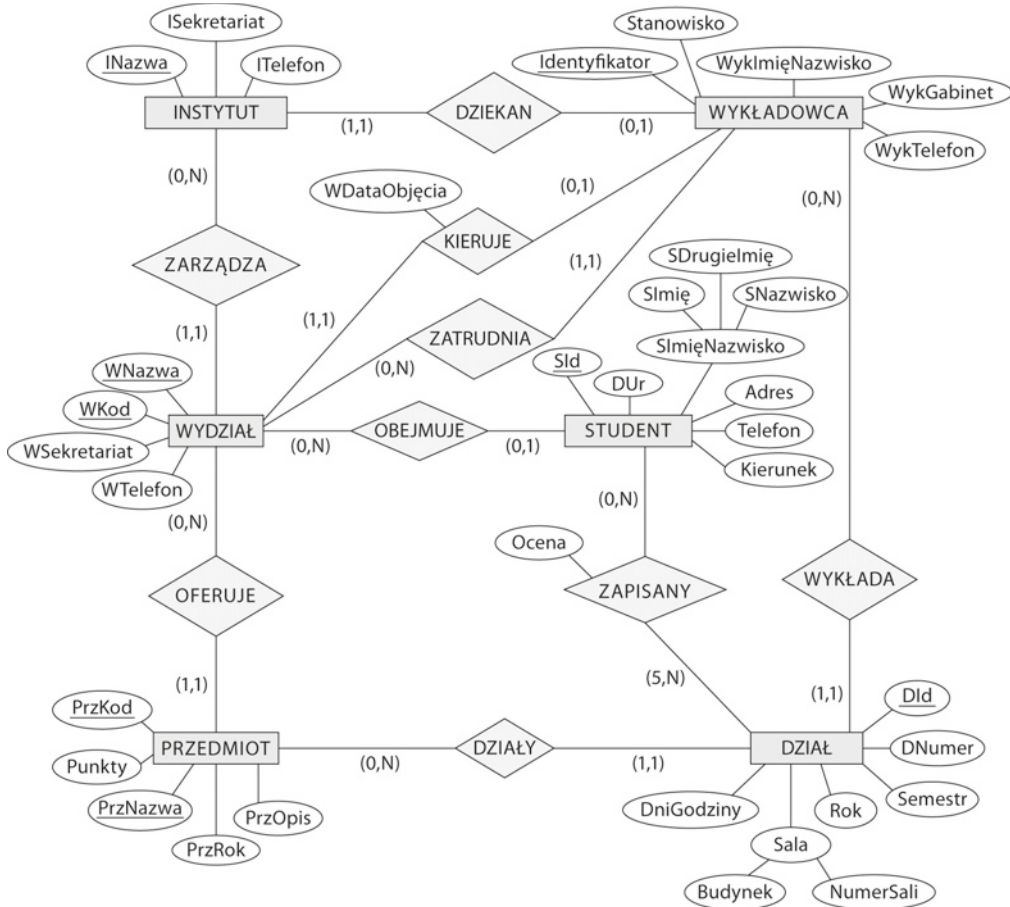
## 3.10. Inny przykład — baza danych UNIWERSYTET

Teraz przedstawimy następny przykład, bazę UNIWERSYTET, aby zilustrować zagadnienia z obszaru modelowania ER. Założmy, że ta baza służy do przechowywania list studentów zapisanych na kursy i ocen końcowych. Po przeanalizowaniu reguł mini-świata i potrzeb użytkowników ustalono następujące wymogi (aby zachować zwieżłość, w opisie wymogów w nawiasach podano nazwy typów encji i atrybutów ze schematu koncepcyjnego; nazwy typów związków znajdują się tylko na diagramie schematu ER):

- Uniwersytet jest podzielony na instytuty (INSTYTUT), a każdy instytut ma unikatową nazwę (INazwa), sekretariat (ISekretariat), numer telefonu (ITelefon) i osobę z kadry naukowej przypisaną do roli dziekana. Każdy instytut zarządza zbiorem wydziałów (WYDZIAŁ). Wydział ma unikatową nazwę (WNazwa), unikatowy numer (WNumer), sekretariat (WSekretariat) i telefon (WTelefon), a także dyrektora. Zapisywana jest też data początkowa (WDataObjęcia), od kiedy dana osoba pełni funkcję dyrektora instytutu.
- Wydział oferuje zestaw przedmiotów (PRZEDMIOT), z których każdy ma unikatową nazwę (PrzNazwa), unikatowy numer (PrzNumer), rok (PrzRok; 1 to przedmiot z roku pierwszego, 2 z drugiego itd.) i opis (PrzOpis). W bazie zapisani są też wykładowcy (WYKŁADOWCA), z których każdy ma unikatowy identyfikator (Identyfikator), imię i nazwisko (WykImięNazwisko), gabinet (WykGabinet), telefon (WykTelefon) i stanowisko (Stanowisko). Ponadto każdy wykładowca pracuje dla jednego z instytutów uczelni.
- Baza przechowuje dane studentów (STUDENT), w tym nazwisko każdego z nich (SImięNazwisko; składa się ono z pierwszego imienia — SImię, drugiego imienia — SDrugieImię, nazwiska — SNazwisko), identyfikator (SID; jest on unikatowy dla każdego studenta), adres (Adres), telefon (Telefon), kod kierunku (Kierunek) i datę urodzenia (DUR). Student jest przypisany do jednego instytutu. Trzeba też przechowywać oceny studenta z każdego działu, jaki student zakończył.
- Przedmioty są oferowane w formie działów (DZIAŁ). Każdy dział jest powiązany z jednym kursem i jednym wykładowcą oraz ma unikatowy identyfikator (DID). Dział ma też numer (DNumber; 1, 2, 3 itd., jeśli w tym samym semestrze i roku prowadzonych jest wiele działów), semestr (Semestr), rok (Rok), salę (Sala; jest ona podawana jako połączenie kodu budynku — Budynek i numeru sali — NumerSali) oraz dni i godziny (DniGodziny; np. „PnŚrPt 9.00 – 9.50” lub „Czw 15.30 – 17.20” z ograniczeniem do dozwolonych dni i godzin). *Uwaga:* ta baza przechowuje wszystkie działy oferowane w kilku ostatnich latach, a także obecną ofertę. Identyfikator DID jest unikatowy na poziomie wszystkich działów, a nie tylko w ramach aktualnego semestru. Baza przechowuje studentów każdego działu i ich oceny (jeśli są dostępne). Oznacza to relację wiele do wielu między studentami a działami. Do działu przypisanych musi być przynajmniej pięciu studentów.

Diagram ER oparty na tych wymaganiach jest przedstawiony na rysunku 3.20. Zastosowano tam notację (min, maks). Zauważ, że dla typu encji DZIAŁ jako podkreślony klucz przedstawiono tylko DID, jednak z powodu ograniczeń występujących w mini-świecie każda encja działu musi mieć unikatową kombinację kilku innych wartości. Na przykład przy typowych ograniczeniach mini-świata unikatowe muszą być następujące kombinacje:





RYSUNEK 3.20. Diagram ER schematu bazy danych UNIWERSYTET

- (1) (*DNumer*, *Semestr*, *Rok*, *PrzKod* — dla przedmiotu powiązanego z działem). To oznacza, że numery działów konkretnego przedmiotu muszą być różne w poszczególnych semestrach i latach.
- (2) (*Semestr*, *Rok*, *Sala*, *DniGodziny*). To oznacza, że w konkretnym semestrze danego roku sala nie może być używana dla dwóch różnych działów w tych samych dniach i godzinach.
- (3) (*Semestr*, *Rok*, *DniGodziny*, *Identyfikator* — dla pracownika uczącego danego działu). To oznacza, że w konkretnym semestrze danego roku wykładowca nie może w tych samych dniach i godzinach uczyć dwóch działów. Zauważ, że ta reguła nie obowiązuje, jeśli na danej uczelni wykładowca może jednocześnie uczyć dwa powiązane działy.

Czy potrafisz wymyślić inne kombinacje atrybutów, które muszą być unikatowe?



## 3.11. Podsumowanie

W tym rozdziale zaprezentowaliśmy zagadnienia związane z wysokopoziomowym, koncepcyjnym modelowaniem danych w oparciu o model związków encji (ER). Rozpoczęliśmy od omówienia roli tego modelu danych w procesie projektowania baz danych, po czym przedstawiliśmy zbiór wymagań dotyczących prostej bazy danych FIRMA (która jest jednym z dwóch przykładów przytaczanych w różnych rozdziałach tej książki). Następnie zdefiniowaliśmy podstawowe pojęcia stosowane w modelu ER, takie jak encje oraz ich atrybuty. W dalszej części tego rozdziału omówiliśmy wartości puste i zaprezentowaliśmy różne typy atrybutów, które można dowolnie zagnieżdżać, tworząc tym samym skomplikowane atrybuty. Przedstawiliśmy następujące rodzaje atrybutów:

- proste lub niepodzielne,
- złożone,
- wielowartościowe.

W tym rozdziale zaprezentowaliśmy także krótkie zestawienie atrybutów stałych (składowanych) i pochodnych. Następnie omówiliśmy wymienione poniżej pojęcia związane z poziomem schematu (nazywanym także intensją) modelu ER:

- typy encji wraz z odpowiadającymi im zbiorami encji,
- atrybuty klucza typów encji,
- zbiory wartości (dziedziny) atrybutów,
- typy związków wraz z odpowiadającymi im zbiorami związków,
- role udziału typów encji w typach związków.

W dalszej części tego rozdziału przedstawiliśmy dwie metody określania ograniczeń strukturalnych dla typów związków. W pierwszej z tych metod wyróżnia się dwa typy ograniczeń strukturalnych:

- współczynniki liczebności (1:1, 1:N, M:N dla związków binarnych),
- ograniczenie udziału (pełne i częściowe).

Wspomnieliśmy o istnieniu alternatywnej metody definiowania ograniczeń strukturalnych przez określanie par liczb minimalnych i maksymalnych (*min*, *maks*) reprezentujących udział poszczególnych typów encji w typach związków. Omówiliśmy także słabe typy encji i związane z nimi pojęcia właścicielskich typów encji, identyfikujących typów związków oraz atrybutów klucza częściowego.

Schematy związków encji mogą być reprezentowane graficznie w postaci diagramów ER. Zaprezentowaliśmy w tym rozdziale sposób projektowania schematów związków encji na bazie przykładu bazy danych FIRMA — w pierwszej kolejności zdefiniowaliśmy typy encji wraz z odpowiednimi atrybutami, następnie dołączyliśmy do schematu typy związków. Przedstawiliśmy także gotowy diagram ER dla schematu wspomnianej bazy danych FIRMA. Dalej omówiliśmy kilka podstawowych pojęć związanych z diagramami klas UML wraz z ich zestawieniem z odpowiednimi zagadnieniami stosowanymi w modelu związków encji. Ponadto opisaliśmy szczegółowo typy związków trójskładnikowych i wyższych stopni, a także omówiliśmy warunki, w jakich różnią się one od związków binarnych.

Na koniec podaliśmy wymogi dotyczące schematu bazy UNIWERSYTET i przedstawiliśmy jego projekt na diagramie ER.

Zaprezentowane do tej pory elementy modelowania związków encji (a więc typy encji, typy związków, atrybuty, klucze oraz ograniczenia strukturalne) umożliwiają modelowanie wielu aplikacji baz danych. Jednak wiele nowszych, bardziej skomplikowanych aplikacji (jak choćby projektowanie na poziomie inżynierskim, systemy informacji medycznej czy rozwiązania telekomunikacyjne) wymaga dodatkowych elementów do precyzyjnego modelowania. Niektóre zaawansowane aspekty modelowania omówimy w rozdziale 8., a do technik z tego obszaru wrócimy w rozdziale 26.

## Pytania powtórkowe

- 3.1. Omów rolę wysokopoziomowego modelu danych w procesie projektowania bazy danych.
- 3.2. Wymień różne przypadki, w których właściwe jest stosowanie pustej wartości.
- 3.3. Zdefiniuj następujące pojęcia: *encja*, *atrybut*, *wartość atrybutu*, *egzemplarz związku*, *atrybut złożony*, *atrybut wielowartościowy*, *atrybut pochodny*, *atrybut skomplikowany*, *atrybut klucza* oraz *zbiór wartości (dziedzina)*.
- 3.4. Czym jest typ, a czym jest zbiór encji? Wyjaśnij różnicę pomiędzy encją, typem encji a zbiorem encji.
- 3.5. Wyjaśnij różnicę pomiędzy atrybutem a zbiorem wartości.
- 3.6. Czym jest typ związku? Wyjaśnij różnice pomiędzy takimi pojęciami jak egzemplarz związku, typ związku oraz zbiór związku.
- 3.7. Czym jest rola wystąpienia w związku? Kiedy stosowanie nazw ról jest konieczne w opisie typów związków?
- 3.8. Opisz dwie alternatywne metody określania ograniczeń strukturalnych dla typów związków. Jakie są zalety i wady każdego z tych rozwiązań?
- 3.9. Pod jakimi warunkami atrybut binarnego typu związku można przekształcić w atrybut jednego z występujących w danym związku typów encji?
- 3.10. Na jakim etapie traktujemy związki jak atrybuty, czym są zbiory wartości tych atrybutów? Która klasa modeli danych bazuje na takich rozwiązaniach?
- 3.11. Co oznacza rekurencyjny typ związku? Podaj kilka przykładów takich typów.
- 3.12. Kiedy w modelowaniu danych mamy do czynienia z pojęciem słabej encji? Zdefiniuj następujące terminy: *właścicielski typ encji*, *słaby typ encji*, *identyfikujący typ związku* oraz *klucz częściowy*.
- 3.13. Czy związek identyfikujący słabego typu encji może mieć stopień większy niż dwa? Podaj przykłady ilustrujące Twoją odpowiedź.
- 3.14. Omów konwencje stosowane do graficznego reprezentowania schematów związków encji w postaci diagramów ER.
- 3.15. Omów konwencje nazewnictwa wykorzystywane w diagramach schematów ER.

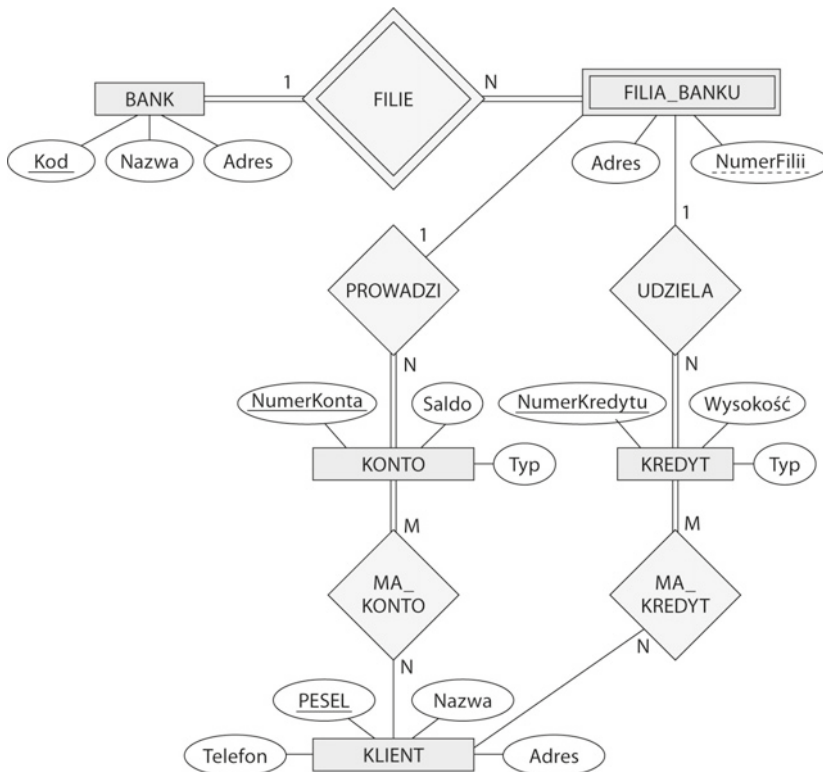
## Ćwiczenia

- 3.16. Które kombinacje atrybutów muszą być unikatowe dla poszczególnych encji DZIAŁ z bazy UNIWERSYTET z rysunku 3.20, aby spełnić następujące wymagania dotyczące mini-świata:
- a) W określonym semestrze i roku oraz dla ustalonej wartości DniGodziny konkretna sala może być używana tylko na potrzeby jednego działu.
  - b) W określonym semestrze i roku wykładowca może uczyć tylko jednego działu dla ustalonej wartości DniGodziny.
  - c) W określonym semestrze i roku numery wszystkich działów oferowanych w ramach jednego przedmiotu muszą być różne.
- Czy potrafisz wymyślić inne podobne ograniczenia?
- 3.17. Złożone i wielowartościowe atrybuty mogą być zagnieżdżane na dowolnej liczbie poziomów. Przypuśćmy, że chcemy zaprojektować atrybut dla typu encji STUDENT, który będzie umożliwiał przechowywanie informacji o wcześniejszym przebiegu nauki. Taki atrybut będzie przechowywał jeden wpis dla każdego wydziału, na jaki student uczęszczał — każdy z takich wpisów będzie się składał z nazwy wydziału, daty rozpoczęcia i zakończenia nauki, uzyskanych stopni naukowych (jeśli dany wydział je przyznaje) oraz z ewentualnej listy zaliczonych przedmiotów dodatkowych (opcjonalne). Każdy wpis dla uzyskanego stopnia składa się z nazwy stopnia oraz miesiąca i roku przyznania go; każdy wpis z listą przedmiotów dodatkowych zawiera nazwę przedmiotu, semestr, rok i ocenę. Zaprojektuj atrybut reprezentujący te informacje. Zastosuj konwencje, których użyliśmy na rysunku 3.5.
- 3.18. Przedstaw alternatywny projekt dla atrybutu opisanego w ćwiczeniu 3.17, który będzie wykorzystywał wyłącznie typy encji (włącznie ze słabymi typami encji, w razie potrzeby) i typy związków.
- 3.19. Przeanalizuj diagram ER przedstawiony na rysunku 3.21, który reprezentuje uproszczony schemat dla systemu rezerwacji biletów lotniczych. Spróbuj określić na podstawie tego diagramu wymagania i ograniczenia, na bazie których opracowano przedstawiony schemat. W swojej specyfikacji wymagań i ograniczeń staraj się zachować jak największą precyzję.
- 3.20. W rozdziałach 1. i 2. omówiliśmy środowisko i użytkowników baz danych. Do opisywania takich środowisk (w tym systemów zarządzania bazami danych, składowanych baz danych, administratorów baz danych czy katalogów i słowników danych) możemy wykorzystywać wiele typów encji. Spróbuj zdefiniować wszystkie typy encji, za pomocą których można stworzyć kompletną specyfikację systemu i środowiska bazy danych; następnie wskaż typy związków pomiędzy tymi typami encji i narysuj diagram związków encji opisujący takie ogólne środowisko bazy danych.
- 3.21. Zaprojektuj schemat związków encji dla bazy danych reprezentującej informacje na temat głosowań (ze wszystkimi oddanymi głosami) w Izbie Reprezentantów Stanów Zjednoczonych z ostatnich dwóch lat. Baza danych musi zawierać nie tylko nazwy wszystkich stanów (Teksas, Nowy Jork, Kalifornia itp.), ale także nazwy regionów (atrybut Region typu encji STAN) w poszczególnych



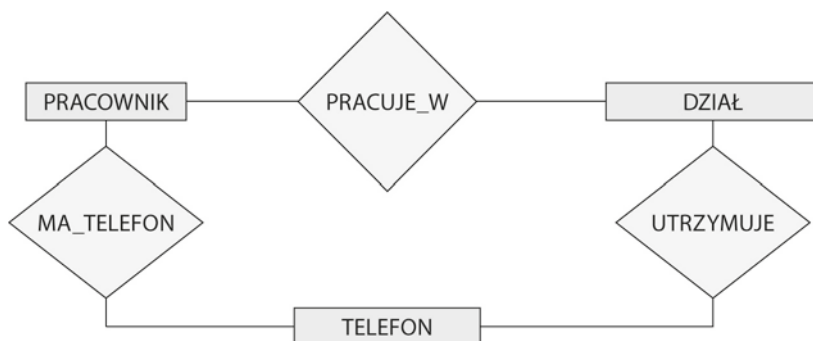
Nieobecny}}). Narysuj diagram schematu związków encji dla opisanej przed chwilą aplikacji bazy danych. Dokładnie opisz wszystkie przyjęte przez siebie założenia.

- 3.22. Przypuśćmy, że konstruujemy bazę danych, która ma umożliwiać śledzenie zespołów i zawodów przeprowadzanych w ramach jakiejś ligi sportowej. Każdy zespół składa się z pewnej liczby zawodników, którzy nie muszą koniecznie występować w tym samym składzie we wszystkich kolejkach spotkań. Baza danych powinna umożliwiać rejestrowanie zawodników występujących w poszczególnych zawodach w barwach swoich zespołów wraz z zajmowaną przez nich pozycją na boisku oraz uzyskanymi wynikami. Zaprojektuj diagram schematu ER dla wspomnianej aplikacji bazy danych, precyzyjnie opisując wszystkie przyjmowane założenia. Wybierz swój ulubiony sport (np. piłkę nożną, koszykówkę lub siatkówkę).
- 3.23. Przeanalizuj diagram przedstawiony na rysunku 3.22, który reprezentuje fragment bazy danych BANK. Każdy bank może utrzymywać wiele filii, a każda filia może obsługiwać wiele kont i kredytów lub pożyczek.
- Wymień wszystkie silne typy encji widoczne na przedstawionym diagramie ER.
  - Czy diagram zawiera słabe typy encji? Jeśli tak, podaj ich nazwy, częściowe klucze oraz związki identyfikujące.



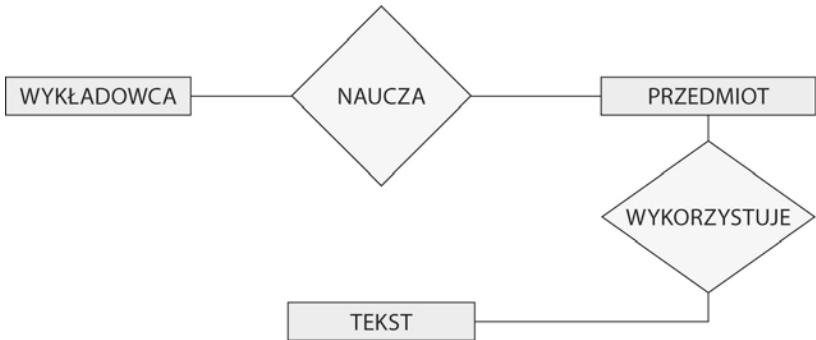
RYСУNEK 3.22. Diagram związków encji dla schematu bazy danych BANK

- c) Jakie ograniczenia wynikają z ewentualnego występowania na diagramie klucza częściowego i związków identyfikujących dla słabych typów encji?
- d) Wymień nazwy wszystkich typów związków i określ dla każdego wystąpienia typu encji w typie związku ograniczenie (*min*, *maks*). Postaraj się uzasadnić dokonane wybory.
- e) Wymień i krótko omów wymagania użytkownika, które mogły doprowadzić do opracowania przedstawionego projektu schematu związków encji.
- f) Przypuśćmy, że każdy klient banku musi mieć co najmniej jedno konto, ale nie może mieć więcej niż dwa zaciągnięte kredyty. Przypuśćmy także, że pojedyncza filia banku nie może udzielić więcej niż 1000 kredytów. Jak opisana sytuacja mogłaby zostać wyrażona w postaci ograniczeń (*min*, *maks*)?
- 3.24. Przeanalizuj diagram związków encji z rysunku 3.23. Przyjmijmy, że jeden pracownik może pracować w maksymalnie dwóch działach firmy, ale może też nie być przypisany do żadnego z działów. Załóżmy także, że każdy dział musi mieć przypisany przynajmniej jeden numer telefonu, ale nie może mieć tych numerów więcej niż trzy. Nanieś na przedstawiony diagram ER odpowiednie ograniczenia (*min*, *maks*). *Dokładnie opisz wszystkie dodatkowo przyjęte założenia.* Kiedy związek MA\_TELEFON byłby w zaprezentowanym przykładzie nadmiarowy?



Rysunek 3.23. DIAGRAM ZWIĄZKÓW ENCJI DLA SCHEMATU BAZY DANYCH FIRMA

- 3.25. Przeanalizuj diagram związków encji z rysunku 3.24. Przyjmijmy, że jeden przedmiot może, ale nie musi się opierać na podręczniku, ale definicja typu encji TEKST mówi, że jest to książka wykorzystywana w trakcie zajęć z jednego z przedmiotów. Jeden przedmiot nie może mieć przypisanych więcej niż pięć podręczników. Jeden wykładowca może prowadzić zajęcia z dwóch do czterech przedmiotów. Nanieś na przedstawiony diagram ER ograniczenia (*min*, *maks*). *Dokładnie opisz wszystkie dodatkowo przyjęte założenia.* Gdybyśmy dodali typ związku WYKORZYSTUJE (określający podręczniki używane przez wykładowcę na kursie), to czy powinien to być związek binarny pomiędzy typami encji WYKŁADOWCA i TEKST, czy związek trójskładnikowy obejmujący wszystkie trzy wymienione typy encji? Jakich wartości *min* i *maks* musielibyśmy użyć? Odpowiedź uzasadnij.



RYSUNEK 3.24. Diagram związków encji dla schematu bazy danych PRZEDMIOTY

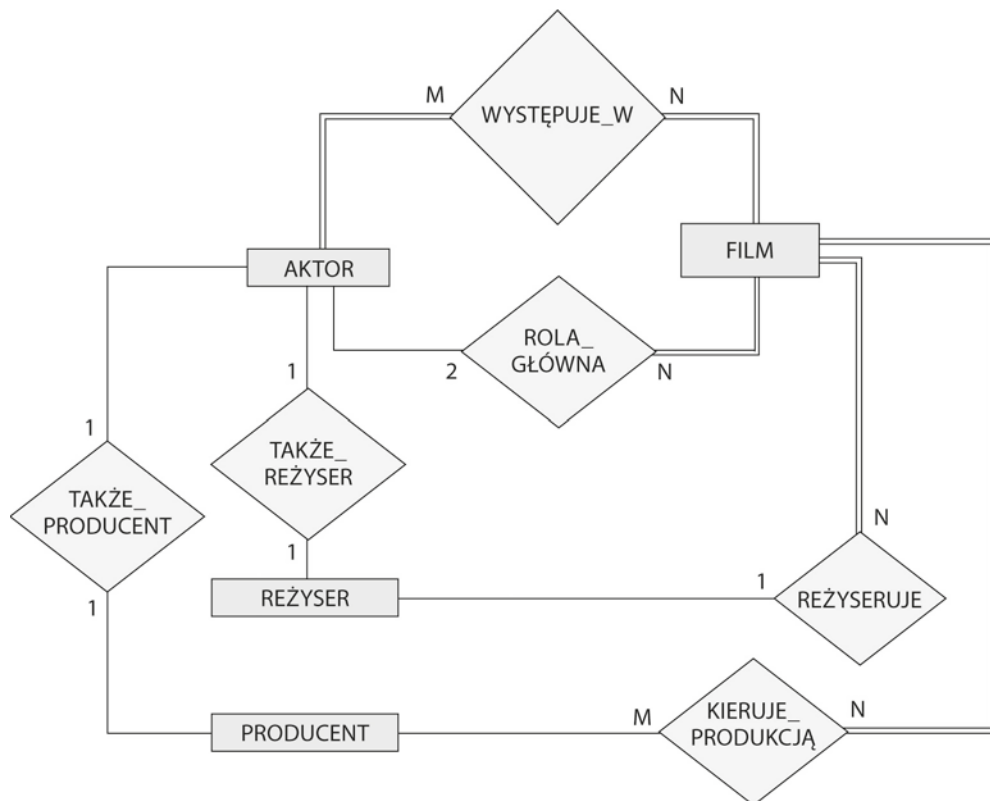
- 3.26. Przeanalizuj należący do bazy danych UNIWERSYTET typ encji DZIAŁ, który opisuje poszczególne działy z wykładanych przedmiotów. Typ encji DZIAŁ obejmuje następujące argumenty: NumerDziału, Semestr, Rok, NumerPrzedmiotu, Wykładowca, NumerSali (gdzie dział jest wykładany), Budynek, DniTygodnia (dziedziną tego argumentu jest zbiór możliwych kombinacji dni tygodnia, w których dany kurs jest nauczany: {PnŚrPt, PnŚr, Wt, itp.}) oraz Godziny (w tym przypadku dziedziną są wszystkie możliwe przedziały czasowe, w których mogą być prowadzone zajęcia z danego działu {9 – 9:45, 10 – 10:45, ..., 15:30 – 16:45, 17:30 – 18:15, itp.}). Przyjmij, że atrybut NumerDziału jest unikatowy dla każdego przedmiotu należącego do określonej kombinacji semestru i roku (co oznacza, że jeśli zajęcia z jakiegoś przedmiotu są prowadzone wiele razy w ciągu konkretnego semestru, działy składające się na ten przedmiot są numerowane zgodnie ze schematem: 1, 2, 3, itd.). Istnieje kilka złożonych kluczy dla typu encji DZIAŁ, a niektóre z jego atrybutów składają się na więcej niż jeden klucz. Wymień przynajmniej trzy klucze złożone i przedstaw właściwy sposób ich reprezentowania na diagramie schematu związków encji.
- 3.27. Współczynniki licznosci często wpływają na szczegółowy projekt bazy danych. Współczynnik licznosci zależy od znaczenia typów encji w świecie rzeczywistym i od konkretnej aplikacji. Zaproponuj współczynniki licznosci dla podanych poniżej relacji binarnych, posługując się typowym rozumieniem wymienionych typów encji. Jednoznacznie określ poczynione założenia.

Encja 1.	Współczynnik licznosci	Encja 2.
1. STUDENT		KARTA_UBEZPIECZENIA_SPOLECZNEGO
2. STUDENT		NAUCZYCIEL
3. SALA		ŚCIANA
4. PAŃSTWO		OBECNY_PREZYDENT
5. PRZEDMIOT		PODRĘCZNIK
6. POZYCJA (z zamówienia)		ZAMÓWIENIE
7. STUDENT		ZAJĘCIA
8. ZAJĘCIA		WYKŁADOWCA
9. WYKŁADOWCA		GABINET
10. LICYT_TOWAR_EBAY		OFERTA_EBAY



3.28. Przyjrzyj się schematowi ER bazy danych FILMY z rysunku 3.25.

Założmy, że baza FILMY zawiera dane. AKTOR to ogólne określenie obejmujące też aktorki. Na podstawie ograniczeń z pokazanego schematu ER oceń podane zdania za pomocą określeń *Prawda*, *Fałsz* lub *Może*. Określenie *Może* zastosuj do zdań, których nie da się zgodnie ze schematem ocenić jednoznacznie jako prawdziwych lub fałszywych. Każdą odpowiedź uzasadnij.



RYSUNEK 3.25. Diagram ER schematu bazy danych FILMY

- W bazie nie ma aktorów, którzy nie występują w żadnym filmie.
- Niektórzy aktorzy wystąpili w więcej niż dziesięciu filmach.
- Niektórzy aktorzy grali główną rolę w wielu filmach.
- Film może mieć maksymalnie dwie główne role.
- Każdy reżyser był aktorem w jakimś filmie.
- Producenci nigdy nie są aktorami.
- Producent nie może być aktorem w innym filmie.
- W niektórych filmach występuje więcej niż dwunastu aktorów.
- Niektórzy producenci są też reżyserami.
- Większość filmów ma jednego reżysera i jednego producenta.

- k) Niektóre filmy mają jednego reżysera i kilku producentów.
  - l) Niektórzy aktorzy grali główną rolę w filmie, reżyserowali go i kierowali produkcją.
  - m) Nie ma filmów, w których reżyser także występuje.
- 3.29. Na podstawie schematu ER bazy FILMY z rysunku 3.25 narysuj diagram egzemplarzy z trzema filmami, które ostatnio pojawiły się w kinach. Narysuj egzemplarze wszystkich typów encji: FILM, AKTOR, PRODUCENT, REŻYSER. Dodaj egzemplarze związków w rzeczywistości występujących w tych filmach.
- 3.30. Narysuj diagram UML do ćwiczenia 3.16. Uwzględnij następujące wymogi:
- a) Student powinien mieć możliwość obliczenia średniej ocen i dodawania lub usuwania kierunków oraz specjalności.
  - b) Każdy instytut powinien mieć możliwość dodawania i usuwania przedmiotów oraz zatrudniania i zwalniania kadry.
  - c) Każdy wykładowca powinien mieć możliwość dodawania i modyfikowania ocen studentów z danych przedmiotów.
- Uwaga:* niektóre z tych funkcji mogą być obsługiwane za pomocą kilku klas.

## Ćwiczenia laboratoryjne

- 3.31. Wróć do bazy UNIWERSYTET z ćwiczenia 3.16. Zbuduj schemat ER tej bazy, używając narzędzia do modelowania danych (np. ERwin lub Rational Rose).
- 3.32. Wyobraź sobie bazę ZAMÓWIENIA\_POCZTOWE, używaną przez pracowników do obsługi nadsyłanych przez klientów zamówień na części. Oto wymogi dotyczące danych z tej bazy:
- Firma realizująca zamówienia zatrudnia pracowników, z których każdy ma unikatowy numer, imię i nazwisko oraz kod pocztowy.
  - Każdy klient firmy jest identyfikowany za pomocą unikatowego numeru klienta, imienia i nazwiska oraz kodu pocztowego.
  - Każda część sprzedawana przez firmę jest identyfikowana za pomocą unikatowego numeru, a ponadto ma określoną nazwę, cenę i liczbę sztuk w magazynie.
  - Każde zamówienie składane przez klienta jest przyjmowane przez pracownika i otrzymuje unikatowy numer. Każde zamówienie obejmuje określoną liczbę sztuk części przynajmniej jednego rodzaju. Każde zamówienie ma też datę przyjęcia, a także oczekiwaną datę wysyłki. Rejestrowana jest też rzeczywista data wysyłki.
- Zaprojektuj diagram ER dla bazy z zamówieniami. Utwórz projekt za pomocą narzędzia do modelowania danych (np. ERwin lub Rational Rose).
- 3.33. Wyobraź sobie bazę danych FILM, w której zapisywane są dane dotyczące branży filmowej. Oto wymogi związane z danymi:
- Każdy film jest identyfikowany za pomocą tytułu i roku produkcji. Wszystkie filmy mają długość (w minutach), studio odpowiedzialne

za produkcję i gatunek (jeden lub kilka, np.: horror, film akcji, dramat). Każdy film ma przynajmniej jednego reżysera i aktora, a także zarys fabuły. Ponadto z każdym filmem powiązana jest dowolna liczba kwestii (może ona wynosić zero) wypowiedzianych przez określonego aktora z danej produkcji.

- Aktorzy są identyfikowani na podstawie imienia i nazwiska oraz daty urodzenia, oraz występują w przynajmniej jednym filmie. Każdy aktor ma przypisaną rolę w filmie, w którym bierze udział.
- Reżyserzy także są identyfikowani na podstawie imienia i nazwiska oraz daty urodzenia. Reżyserują przynajmniej jeden film i mogą grać w filmach (także w tych, które reżyserują).
- Studia filmowe są identyfikowane na podstawie nazwy i mają adresy. Studio wyprodukowało przynajmniej jeden film.

Zaprojektuj diagram ER reprezentujący tę bazę danych z filmami. Zapisz ten projekt za pomocą narzędzia do modelowania danych (np. ERwin lub Rational Rose).

- 3.34. Wyobraź sobie bazę danych RECENZJE\_PRAC\_NA\_KONFERENCJĘ z pracami naukowymi nadesłanymi przez badaczy do recenzji. W bazie zapisywane są oceny recenzentów na potrzeby wyboru prac do konferencji. Ten system bazy danych jest przeznaczony przede wszystkim dla recenzentów, którzy zapisują odpowiedzi na pytania ewaluacyjne dotyczące każdej recenzowanej pracy i rekomendują przyjęcie lub odrzucenie tekstu. Oto wymogi związane z danymi:

- Autorzy prac są w unikatowy sposób identyfikowani na podstawie adresu e-mail. Zapisywane są też imiona i nazwiska autorów.
- Każdej pracy system przypisuje unikatowy identyfikator. Z pracami powiązane są też: tytuł, streszczenie i nazwa pliku z tekstem.
- Praca może mieć wielu autorów, jednak jeden z nich jest wskazany jako osoba kontaktowa.
- Recenzenci prac są w unikatowy sposób identyfikowani na podstawie adresu e-mail. Zapisywane są też: imię, nazwisko, numer telefonu, organizacja i tematy zainteresowań każdego recenzenta.
- Każda praca jest przydzielana kilku recenzentom (od dwóch do czterech). Recenzent każdą pracę ocenia na skali od 1 do 10 w czterech kategoriach: wartość merytoryczna, czytelność, oryginalność i związek z konferencją. Każdy recenzent przedstawia też ogólną rekomendację dotyczącą każdej pracy.
- Każda recenzja obejmuje dwa rodzaje uwag: jeden tylko dla komitetu akceptacyjnego i drugi z informacjami zwrotnymi dla autorów.

Zaprojektuj diagram ER bazy RECENZJE\_PRAC\_NA\_KONFERENCJĘ i utwórz projekt za pomocą narzędzia do modelowania danych (np. ERwin lub Rational Rose).

- 3.35. Przyjrzyj się diagramowi ER bazy danych LINIE\_LOTNICZE z rysunku 3.21. Utwórz ten projekt za pomocą narzędzia do modelowania danych (np. ERwin lub Rational Rose).

## Wybrane publikacje

Model związków encji został wprowadzony przez Chena (1976), a omówienie zagadnień związanych z tym modelem można znaleźć w książkach Schmidta i Swensona (1975), Wiederholda i Elmasriego (1979) oraz Senki (1975). Od tamtego czasu w rozmaitych publikacjach zaproponowano mnóstwo modyfikacji oryginalnego modelu ER. Niektóre z tych sugestii zastosowaliśmy w powyższej prezentacji. Nakładane na związki ograniczenia strukturalne zostały omówione w książkach Abriala (1974), Elmasriego i Wiederholda (1980) oraz Lenzeriniego i Santucciiego (1983). Dołączenie do modelu związków encji atrybutów wielowartościowych i atrybutów złożonych po raz pierwszy zaproponowano w książce Elmasri'ego i in. (1985). W tym rozdziale nie omawialiśmy co prawda żadnych języków wykorzystywanych w modelu związków encji ani rozszerzeń tego modelu, warto jednak pamiętać o istnieniu wielu ciekawych propozycji w tym zakresie. Elmasri i Wiederhold (1981) zaproponowali język zapytań GORDAS dla modelu ER. Jeszcze inny język zapytań dla modelu związków encji zaproponowali w swojej książce Markowitz i Raz (1983). Senko (1980) zaproponował język zapytań dla własnego modelu DIAM. Parent i Spaccapietra (1985) przedstawili zbiór operacji nazwany algebrą związków encji. Nieco inny język formalny dla modelu ER zaproponowali w swojej książce Gogolla i Hohenstein (1991). Campbell i in. (1985) przedstawili zbiór operacji ER i udowodnili, że są relacyjnie kompletne. Od roku 1979 regularnie są organizowane konferencje naukowe poświęcone analizie dokonań badawczych w obszarze modelu związków encji. Konferencje z tego cyklu (obecnie noszące nazwę International Conference on Conceptual Modeling) odbywały się w Los Angeles (w latach 1979, 1983 oraz 1997); w Waszyngtonie (w roku 1981); w Chicago (w roku 1985), w Dijon we Francji (w roku 1986); w Nowym Jorku (w roku 1987); w Rzymie (w roku 1988); w Toronto (w roku 1989); w Lozannie w Szwajcarii (w roku 1990); w San Mateo w Kalifornii (w roku 1991); w Karlsruhe w Niemczech (w roku 1992); w Arlington w Teksasie (w roku 1993); w Manchesterze (w roku 1994); w Brisbane w Australii (w roku 1995); w Cottbus w Niemczech (w roku 1996); w Singapurze (w roku 1998); w Paryżu we Francji (w roku 1999); w Salt Lake City w stanie Utah (w roku 2000); w Jokohamie w Japonii (w roku 2001); w Tampere w Finlandii (w roku 2002); w Chicago w stanie Illinois (w roku 2003); w Szanghaju w Chinach (w roku 2004); w Klagenfurcie w Austrii (w roku 2005); w Tucson w stanie Arizona (w roku 2006); w Auckland w Nowej Zelandii (w roku 2007); w Barcelonie w Katalonii w Hiszpanii (w roku 2008); w Gramado w Brazylii (w roku 2009); w Vancouver w Kolumbii Brytyjskiej w Kanadzie (w roku 2010); w Brukseli w Belgii (w roku 2011); we Florencji we Włoszech (w roku 2012); w Hongkongu w Chinach (w roku 2013); w Atlancie w stanie Georgia (w roku 2014) i w Sztokholmie w Szwecji (w roku 2015).



## Rozszerzony model związków encji

Omówione w rozdziale 3. pojęcia związane z modelowaniem związków encji w zupełności wystarczają do reprezentowania wielu schematów baz danych dla *tradycyjnych* aplikacji tego typu, do których należą głównie aplikacje przetwarzające dane wykorzystywane w biznesie i przemyśle. Od późnych lat 70-tych projektanci aplikacji baz danych podejmują jednak próby tworzenia bardziej dokładnych schematów baz danych, które zapewnią większą precyzję reprezentowania właściwości danych i ograniczeń modelowanego mini-świata. Dostępność dokładniejszych metod okazała się szczególnie ważna w przypadku nowszych zastosowań technologii baz danych, takich jak bazy danych wykorzystywane w projektowaniu inżynierskim i produkcji przemysłowej (CAD/CAM<sup>1</sup>), telekomunikacji, skomplikowanych systemach oprogramowania, systemach informacji geograficznej (GIS) oraz wielu innych zastosowaniach. Bazy danych należące do wymienionych kategorii mają bardziej skomplikowane wymagania niż tradycyjne aplikacje. W odpowiedzi na to zapotrzebowanie zdefiniowano dodatkowe pojęcia *semantycznego modelowania danych*, które z powodzeniem można stosować w takich koncepcyjnych modelach danych jak model związków encji (ER). W rozmaitych publikacjach zaproponowano wiele różnych semantycznych modeli danych. Wiele z nowych pojęć zostało zdefiniowanych zupełnie niezależnie w spokrewnionych obszarach informatyki, takich jak **reprezentacja wiedzy** w sztucznej inteligencji czy **modelowanie obiektowe** w inżynierii oprogramowania.

W tym rozdziale nie tylko opiszemy elementy zaproponowane dla semantycznych modeli danych, ale także zaprezentujemy sposób, w jaki można rozszerzyć o te pojęcia tradycyjny model związków encji (ER), otrzymując tym samym **rozszerzony model ER**, w skrócie **EER**<sup>2</sup>. Na początku, w podrozdziale 4.1 spróbujemy przenieść takie pojęcia jak *związki klasy-podklasy* i *dziedziczenie typów* na grunt modelu związków encji. Następnie, w podrozdziale 4.2 wprowadzimy pojęcia *specjalizacji* i *generalizacji*. W podrozdziale 4.3 omówimy różne typy *ograniczeń* nakładanych na związki specjalizacji-generalizacji, natomiast w podrozdziale 4.4 przedstawimy sposób, w jaki można modelować konstrukcję UNII — rozszerzymy tradycyjny model związków encji (ER) o dodatkowe pojęcie *kategorii*. W podrozdziale 4.5 zaprezentujemy przykład schematu bazy danych UNIWERSYTET w modelu EER i podsumujemy zagadnienia związane z tym modelem w oparciu o formalne definicje poszczególnych elementów. W tym rozdziale wymiennie stosujemy pojęcia *obiekt* i *encja*, ponieważ wiele omawianych tu zagadnień jest powszechnie stosowanych w modelach obiektowych.

---

<sup>1</sup> Skrót CAD/CAM pochodzi od ang. *Computer-Aided Design/Computer-Aided Manufacturing*.

<sup>2</sup> Skrót EER pochodzi od ang. *Extended Entity-Relationship*, czyli *Extended ER*.

W kolejnym podrozdziale (4.6) zaprezentujemy notację diagramu klas UML umożliwiającą reprezentowanie związków specjalizacji i generalizacji (wraz z ich krótkim zestawieniem z odpowiednimi elementami i pojęciami z notacji EER). Jest to przykładowa notacja alternatywna, a wspomniany podrozdział stanowi kontynuację podrozdziału 3.8, gdzie zaprezentowano podstawowe składniki notacji diagramu klas UML odpowiadające podstawowemu modelowi ER. W podrozdziale 4.7 przedstawiamy podstawowe abstrakcje, które są wykorzystywane w roli formalnej podstawy dla wielu semantycznych modeli danych. Podrozdział 4.8 zawiera podsumowanie rozdziału.

Materiał zawarty w tym rozdziale powinien być traktowany jako logiczne przedłużenie treści rozdziału 3. — oba rozdziały stanowią łącznie szczegółowe wprowadzenie do modelowania koncepcyjnego. Jeśli jednak celem zajęć jest jedynie wprowadzenie podstaw modelowania związków encji, ten rozdział można w całości pominąć. Alternatywnym rozwiązaniem jest jeszcze pominięcie niektórych lub wszystkich podrozdziałów z końca tego rozdziału (od 4.4 do 4.8).

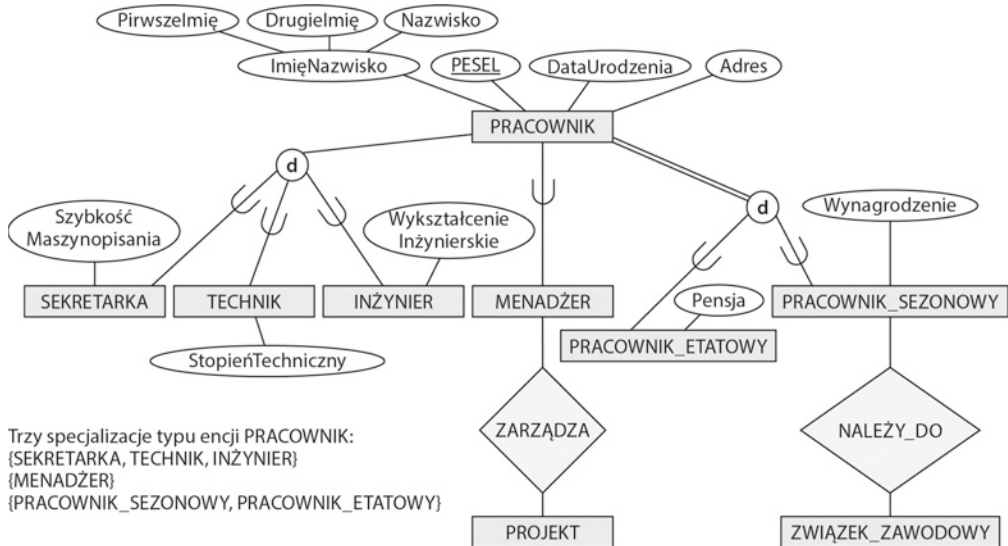
## 4.1. Podklasy, nadklasy i dziedziczenie

W rozszerzonym modelu związków encji (rozszerzonym modelu ER, w skrócie EER) mają zastosowanie *wszystkie pojęcia związane z modelowaniem danych z modelu ER* zaprezentowane w rozdziale 3. W modelu EER wykorzystuje się dodatkowo takie pojęcia jak **podklasa** i **nadklasa** oraz związane z nimi terminy **specjalizacji** i **generalizacji** (patrz podrozdziały 4.2 i 4.3). Kolejnym elementem wykorzystywanym w modelu EER jest **kategoria** lub **typ unii** (patrz podrozdział 4.4), który jest wykorzystywany do reprezentowania zbioru obiektów, czyli *unii* obiektów różnych typów encji. Z wymienionymi pojęciami wiąże się istotny w modelu EER mechanizm **dziedziczenia atrybutów i związków**. Niestety, dla wymienionych składników rozszerzonego modelu ER nie ma jeszcze przyjętej standardowej terminologii, zatem w tym rozdziale będziemy się posługiwali najbardziej popularnymi nazwami poszczególnych struktur. W razie konieczności, informacje o spotykanej w literaturze alternatywnej terminologii będą umieszczane w przypisach. W tym podrozdziale opiszemy dodatkowo technikę prezentowania elementów schematu EER w postaci diagramów. Takie diagramy nazywamy **rozszerzonymi diagramami związków encji (ER)** lub **diagramami EER**.

Pierwszym pojęciem typowym dla rozszerzonego modelu ER, którym się zajmujemy, jest **podklasa** typu encji. Zgodnie z tym, co napisano w rozdziale 3., typ encji jest wykorzystywany do reprezentowania zarówno samego *typu encji*, jak i *zbioru encji określonego typu*, który istnieje w bazie danych. Przykładowo, typ encji PRACOWNIK nie tylko opisuje atrybuty i związki każdej encji reprezentującej pojedynczego pracownika, ale także definiuje bieżący zbiór encji typu PRACOWNIK w bazie danych FIRMA. W wielu przypadkach pojedynczy typ encji zawiera wiele mniejszych podgrup swoich encji, które mają określony sens i (z uwagi na swoje znaczenie dla aplikacji bazy danych) muszą być reprezentowane jawnie. Przykładowo, encje składające się na typ encji PRACOWNIK mogą być podzielone na następujące podgrupy: SEKRETARKA, INŻYNIER, KIEROWNIK, TECHNIK, PRACOWNIK\_ETATOWY, PRACOWNIK\_↪SEZONOWY itp. Zbiór encji w każdej z wymienionych podgrup jest podzbiorem wszystkich encji należących do zbioru encji PRACOWNIK, co oznacza, że każda encja należąca do jednej



z tych podgrup jednocześnie reprezentuje pracownika. Każdą z takich podgrup nazywamy **podklasą** typu encji PRACOWNIK, a sam typ encji PRACOWNIK jest nazywany **nadklasą** dla każdej z tych podklas. Na rysunku 4.1 przedstawiono sposób graficznego reprezentowania tych pojęć na diagramie EER. Notacja z okręgiem z rysunku 4.1 jest opisana w podrozdziale 4.2.



RYСУNEK 4.1. Notacja diagramu EER dla reprezentacji podklas i specjalizacji

Związek pomiędzy nadklasą a każdą z jej podklas nazywamy **związkiem nadklasy-podklasy** lub po prostu **związkiem klasy-podklasy**<sup>3</sup>. W naszym wcześniejszym przykładzie, związki PRACOWNIK-SEKRETARKA oraz PRACOWNIK-TECHNIK stanowiły dwa osobne związki klasy-podklasy. Warto pamiętać, że encja należąca do podklasy reprezentuje *ten sam byt świata rzeczywistego*, co pewna encja należąca do nadklasy; przykładowo, encja typu SEKRETARKA Anna Kowalska jest także encją Anna Kowalska typu PRACOWNIK. Oznacza to, że element składowy podklasy jest w istocie tym samym, co odpowiednia encja nadklasy, jednak występuje w innej, *dokładniej sprecyzowanej roli*. Okazuje się jednak, że kiedy implementujemy związek nadklasy-podklasy w systemie bazy danych, możemy reprezentować encje podklasy w postaci osobnych obiektów bazy danych — powiedzmy, że w postaci osobnych rekordów powiązanych z encjami nadklasy za pomocą atrybutu klucza. W podrozdziale 9.2 omówimy rozmaite możliwości w zakresie reprezentowania związków nadklasy-podklasy w relacyjnych bazach danych.

Encja nie może istnieć w bazie danych wyłącznie w postaci elementu podklasy — każda encja będąca egzemplarzem podklasy musi także występować w bazie danych w postaci odpowiedniego egzemplarza nadklasy. Taka encja może być opcjonalnie dołączana jako element składowy dowolnej liczby podklas. Przykładowo, pracownik zatrudniony na

<sup>3</sup> Związek klasy-podklasy jest często nazywany **związkiem JEST** (lub **związkiem IS-A, IS-AN**), co wynika ze sposobu odwoływania się do tego typu związków. Mówimy „SEKRETARKA jest PRACOWNIKIEM”, „TECHNIK jest PRACOWNIKIEM” itp.

podstawie umowy o pracę, który dodatkowo jest inżynierem, należy do dwóch podklas (INŻYNIER i PRACOWNIK\_ETATOWY) typu encji PRACOWNIK. Warto jednak pamiętać, że nie dla każdej encji nadklasy musi istnieć odpowiednia encja w którejś z podklas.

Ważnym składnikiem rozszerzonego modelu ER, który jest ściśle związany z podklasami, jest **dziedziczenie typu**. Przypomnijmy sobie, że *typ* encji jest definiowany zarówno za pomocą posiadanych przez siebie atrybutów, jak i typów związków, w których występuje. Ponieważ każda encja podklasy reprezentuje ten sam byt ze świata rzeczywistego co odpowiednia encja nadklasy, encja podklasy powinna przechowywać nie tylko wartości własnych atrybutów, *ale także* wartości atrybutów pochodzących z nadklasy. Mówimy, że encja należąca do podklasy **dziedziczy** wszystkie atrybuty (wraz z wartościami) odpowiedniej encji należącej do nadklasy. Encja podklasy dziedziczy także wszystkie związki, w których występuje jej nadklasa. Należy pamiętać, że podklasa z jej charakterystycznymi (lokalnymi) atrybutami i związkami dopiero w połączeniu ze wszystkimi atrybutami i związkami odziedziczonymi z nadklasy może być traktowana jak prawidłowy *typ encji*<sup>4</sup>.

## 4.2. Specjalizacja i generalizacja

### 4.2.1. Specjalizacja

**Specjalizacja** jest procesem definiowania *zbioru podklas* jednego typu encji, który jest w takim przypadku nazywany **nadklasą** specjalizacji. Zbiór podklas, który tworzy taką specjalizację, jest definiowany w oparciu o pewne wyróżniające charakterystyki encji wyodrębnione w nadklasie. Przykładowo, zbiór podklas {SEKRETARKA, INŻYNIER, TECHNIK} jest taką specjalizacją nadklasy PRACOWNIK, którą można wyróżnić spośród kompletnego zbioru encji reprezentujących pracowników w oparciu o *rodzaj pracy* poszczególnych encji. W oparciu o rozmaite charakterystyki wyróżniające możemy wyznaczać wiele specjalizacji dla tego samego typu encji. Przykładowo, po zastosowaniu innej specjalizacji dla typu encji PRACOWNIK możemy otrzymać zbiór podklas {PRACOWNIK\_ETATOWY, PRACOWNIK\_SEZONOWY} — taka specjalizacja różnicuje pracowników według *rodzaju umowy*.

Na rysunku 4.1 przedstawiono, w jaki można graficznie reprezentować specjalizację na diagramie **EER**. Podklasy, które definiują specjalizację, są połączone za pomocą linii bezpośrednio z okręgiem reprezentującym samą specjalizację, który z kolei jest połączony za pomocą linii z nadklasą. *Symbol podzbioru* umieszczany na każdej linii łączącej podklasę z tym okręgiem oznacza kierunek związku nadklasy-podklasy<sup>5</sup>. Atrybuty, które mają zastosowanie wyłącznie w przypadku encji należących do konkretnej podklasy (np. SzybkośćMaszynopisania w przypadku podklasy SEKRETARKA) są dołączane na diagramie

<sup>4</sup> W niektórych obiektowych językach programowania występuje popularne ograniczenie, które mówi, że pojedyncza encja (lub obiekt) może należeć *tylko do jednego typu*. Takie ograniczenie jest zasadniczo zbyt restrykcyjne w przypadku koncepcyjnego modelowania baz danych.

<sup>5</sup> Istnieje wiele alternatywnych notacji dla związku specjalizacji; w podrozdziale 4.6 zaprezentujemy odpowiednią notację języka UML, pozostałe zaproponowane w literaturze notacje przedstawimy w dodatku A.

mie wyłącznie do prostokątów reprezentujących te podklasy. Takie atrybuty są zazwyczaj nazywane **atributami specyficznymi** lub **atributami lokalnymi** podklasy. Podobnie, podklasa może występować w **specyficznych (lokalnych) typach encji**, przykładem może być podklasa `PRACOWNIK_SEZONOWY` występująca w związku `NALEŻY_DO` (patrz rysunek 4.1). Znaczenie symbolu **d** (umieszczanego w okręgach reprezentujących na diagramie specyfikacje — patrz rysunek 4.1) oraz innych elementów notacji diagramów EER omówimy za chwilę.

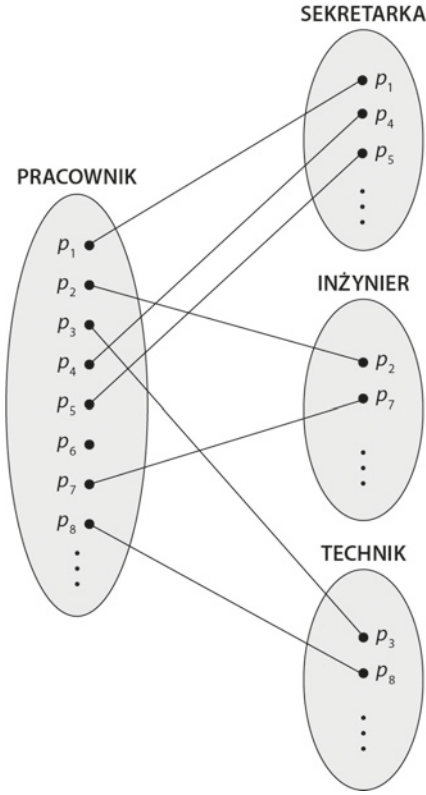
Na rysunku 4.2 przedstawiono kilka egzemplarzy encji należących do podklas specjalizacji {`SEKRETARKA`, `INŻYNIER`, `TECHNIK`}. Warto w tym miejscu jeszcze raz przypomnieć, że encja należąca do podklasy reprezentuje *ten sam byt ze świata rzeczywistego* co połączona z nią (za pomocą związku specyfikacji) encja w nadklasie `PRACOWNIK`, zatem ten sam byt jest reprezentowany na diagramie dwukrotnie; przykładowo, na rysunku 4.2 encja  $p_1$  występuje zarówno w nadklasie `PRACOWNIK`, jak i podklasie `SEKRETARKA`. Zgodnie z sugestią zawartą na rysunku takie związki nadklasy-podklasy jak `PRACOWNIK-SEKRETARKA` przypominają *na poziomie egzemplarzy* związki 1:1 (patrz rysunek 3.12). Główna różnica polega na tym, że związek 1:1 wiąże *dwie różne encje*, natomiast w przypadku związku nadklasy-podklasy encja w podklasie reprezentuje *ten sam byt ze świata rzeczywistego* co encja w nadklasie, tyle że w innej, *wyspecjalizowanej roli* — przykładowo, typ encji `PRACOWNIK` występuje w wyspecjalizowanej roli sekretarki (podklasa `SEKRETARKA`) oraz w wyspecjalizowanej roli technika (podklasa `TECHNIK`).

Istnieją dwa główne powody, dla których do tradycyjnego modelu danych dołączono związki klasy-podklasy i związki specjalizacji. Pierwszym z nich jest fakt, że pewne atrybuty mogą mieć zastosowane tylko dla niektórych encji danej nadklasy. Podklasa jest wówczas definiowana w celu właściwego pogrupowania właśnie tych encji, w przypadku których stosowanie określonego atrybutu lub podzbioru atrybutów jest uzasadnione. Przykładowo, na rysunku 4.1 podklasa `SEKRETARKA` zawiera lokalny atrybut `SzybkośćMaszynopisania`, natomiast podklasa `INŻYNIER` zawiera lokalny atrybut `WyszkolenieInżynierskie`, ale obie te podklasy dzielą ten sam zbiór atrybutów odziedziczonych z typu encji `PRACOWNIK`.

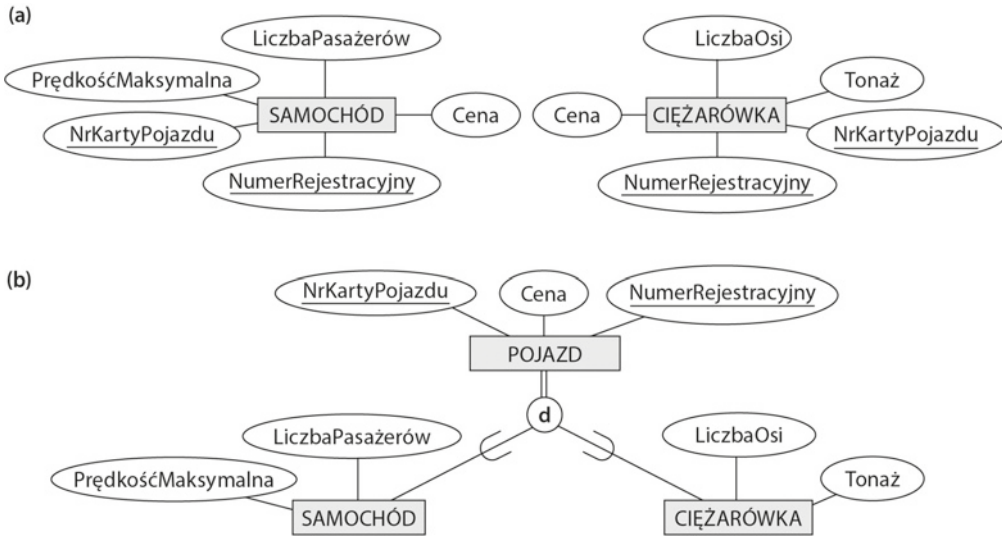
Drugim powodem stosowania podklas jest to, że w niektórych typach związków mogą występować wyłącznie encje należące do określonej podklasy. Przykładowo, jeśli do związku zawodowego pracowników sezonowych mogą należeć wyłącznie osoby reprezentowane przez encje podklasy `PRACOWNIK_SEZONOWY`, możemy ten fakt reprezentować właśnie przez stworzenie podklasy `PRACOWNIK_SEZONOWY` klasy `PRACOWNIK` i związać tę podklasę z typem encji `ZWIĄZEK_ZAWODOWY` za pomocą typu związku `NALEŻY_DO` (patrz rysunek 4.1).

### 4.2.2. Generalizacja

Możemy zdefiniować *odwrotny proces* abstrahowania, w którym likwidujemy różnice pomiędzy rozmaitymi typami encji, identyfikujemy ich wspólne właściwości i **uogólniamy (generalizujemy)** je do postaci pojedynczej **nadklasy**, dla której oryginalne typy encji stanowią wyspecjalizowane **podklasy**. Przykładowo, przeanalizujemy zaprezentowane na rysunku 4.3(a) typy encji `SAMOCHÓD` i `CIĘŻARÓWKA`. Ponieważ oba typy zawierają kilka wspólnych atrybutów, można je zgeneralizować do postaci pojedynczego typu encji `POJAZD` (patrz rysunek 4.3(b)). Zarówno typ encji `SAMOCHÓD`, jak i typ encji `CIĘŻARÓWKA` stanowi teraz podklasę **zgeneralizowanej nadklasy** `POJAZD`. Pojęcia **generalizacja** używamy w odniesieniu do procesu definiowania uogólnionego typu encji na bazie danych wyspecjalizowanych typów encji.



RYSUNEK 4.2. Egzemplarze specjalizacji



RYSUNEK 4.3. Generalizacja. (a) Dwa typy encji: SAMOCHÓD i CIĘŻARÓWKA. (b) Generalizacja typów SAMOCHÓD i CIĘŻARÓWKA w oparciu o nadklasę POJAZD

Warto zwrócić uwagę na to, że proces generalizacji można traktować jak funkcjonalną odwrotność procesu specjalizacji. Oznacza to, że parę typów encji {SAMOCHÓD, CIĘŻARÓWKA} można postrzegać jak specjalizację typu encji POJAZD, zamiast traktować typ POJAZD jak generalizację typów SAMOCHÓD i CIĘŻARÓWKA. Warto pamiętać, że w niektórych metodach projektowych stosuje się specjalne elementy notacji do odróżniania generalizacji i specjalizacji. Strzałka wskazująca na uogólnioną nadklasę reprezentuje generalizację, natomiast strzałki wskazujące na doprecyzowane podklasy reprezentują specjalizację. W tej książce *nie* będziemy stosowali tej notacji, ponieważ decyzja dotycząca wykorzystywania w poszczególnych sytuacjach właściwych procesów (specjalizacji lub generalizacji) często jest subiektywna.

Do tej pory wprowadziliśmy nie tylko pojęcie podklasy i związku nadklasy-podklasy, ale także procesy specjalizacji i generalizacji. Generalnie, nadklasa lub podklasa reprezentuje zbiór encji pewnego typu, a więc w pewnym sensie opisuje *typ encji* — dlatego też nadklasy i podklasy są przedstawiane na diagramach EER w postaci prostokątów (czyli tak samo jak typy encji).

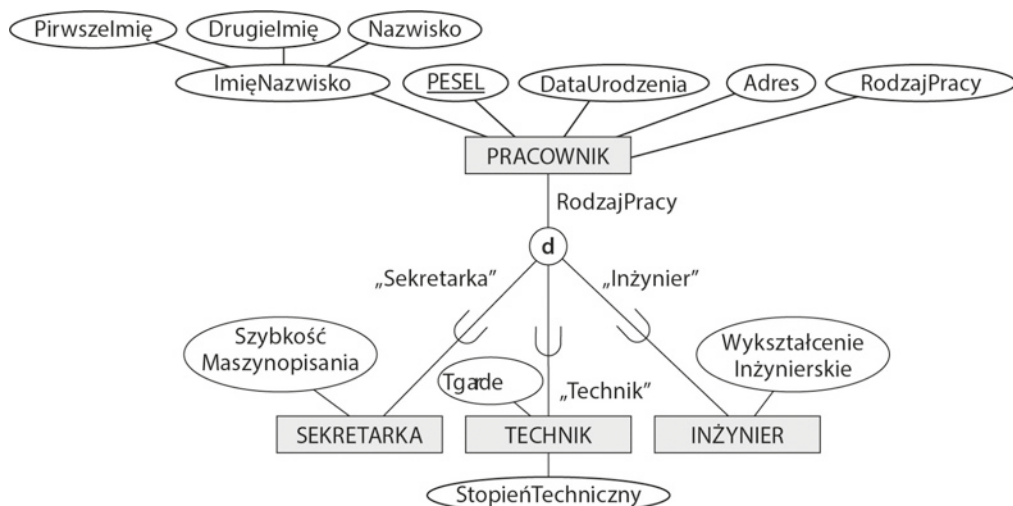
## 4.3. Ograniczenia i właściwości hierarchii specjalizacji i generalizacji

W pierwszej kolejności zajmiemy się ograniczeniami, które można stosować dla pojedynczych procesów specjalizacji lub generalizacji. Dla skrócenia naszych rozważań skupimy się wyłącznie na *specjalizacji*, chociaż prezentowane mechanizmy mają zastosowanie *zarówno* dla specjalizacji, jak i dla generalizacji. Następnie omówimy różnice pomiędzy *kramami* specjalizacji-generalizacji (występującymi w przypadku *dziedziczenia wielokrotnego*) a *hierarchiami* specjalizacji-generalizacji (występującymi w przypadku *dziedziczenia jednokrotnego*) i przejdziemy do szczegółowej analizy różnic pomiędzy procesami specjalizacji i generalizacji podczas projektowania koncepcyjnego schematu bazy danych.

### 4.3.1. Ograniczenia dotyczące specjalizacji i generalizacji

Możemy mieć do czynienia z wieloma specjalizacjami zdefiniowanymi dla tego samego typu encji (nadklasy) — widać to np. na diagramie przedstawionym na rysunku 4.1. W takich przypadkach encje mogą należeć do podklas wyznaczonych w poszczególnych procesach specjalizacji. Nie można jednak wykluczyć sytuacji, w której specjalizacja obejmuje tylko *jedną* podklasę (jak choćby widoczna na rysunku 4.1 specjalizacja {KIEROWNIK}) — wówczas nie stosuje się notacji z okręgiem.

W przypadku niektórych specjalizacji możemy dokładnie określić encje, które będą należały do poszczególnych podklas, przez wykorzystanie odpowiedniego warunku dla wybranego atrybutu lub grupy atrybutów nadklasy. Tak wyznaczone podklasy są wówczas nazywane **podklasami zdefiniowanymi przez predykat** lub **zdefiniowanymi przez warunek**. Przykładowo, gdyby typ encji PRACOWNIK zawierał atrybut RodzajPracy (rysunek 4.4), moglibyśmy zdefiniować warunek przynależności do podklasy SEKRETARKA przez zastosowanie równości (RodzajPracy = "Sekretarka"), którą moglibyśmy nazywać



Rysunek 4.4. Notacja diagramu EER dla specjalizacji zdefiniowanej przez atrybut RodzajPracy

**predykatem definiującym** tę podklasę. Taki warunek jest *ograniczeniem* określającym, że tylko i wyłącznie te encje typu encji PRACOWNIK, w których atrybut RodzajPracy ma wartość "Sekretarka", należą do podklasy SEKRETARKA. Na diagramach EER takie podklasy zdefiniowane przez predykat możemy reprezentować zapisując warunek predykatu bezpośrednio obok linii łączącej podklasę z okręgiem symbolizującym specjalizację.

Gdyby *wszystkie* podklasy w danej specjalizacji miały zdefiniowany warunek członkostwa w oparciu o *ten sam* atrybut nadklasy, sam proces specjalizacji moglibyśmy nazywać **specjalizacją zdefiniowaną przez atrybut**, a atrybut, którego wartość ma wpływ na przydział encji do poszczególnych podklas, moglibyśmy nazywać **atrybutem definiującym** dany proces specjalizacji<sup>6</sup>. W tym scenariuszu wszystkie encje o tej samej wartości atrybutu należą do tej samej podklasy. Specjalizację zdefiniowaną przez atrybut będziemy oznaczali na diagramie przez umieszczenie nazwy atrybutu definiującego bezpośrednio obok łuku łączącego okrąg reprezentujący samą specjalizację z prostokątem reprezentującym nadklasę (patrz rysunek 4.4).

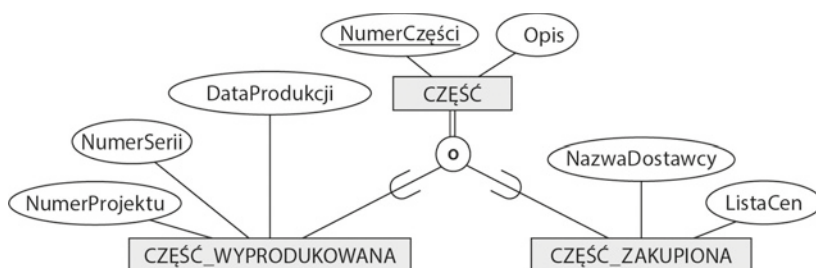
W sytuacji, gdy nie ma zdefiniowanego warunku umożliwiającego proste przypisywanie encji do poszczególnych podklas, mówi się, że podklasy są **definiowane przez użytkownika**. Przynależność encji do podklasy jest bowiem określana przez użytkowników bazy danych w momencie wykonywania operacji dodawania encji do tej podklasy; oznacza to, że przynależność jest *określana przez użytkownika niezależnie dla każdej encji*, nie przez jakiegokolwiek warunki, których spełnienie mogłoby być określane w sposób automatyczny.

W przypadku specjalizacji można stosować jeszcze dwa inne ograniczenia. Pierwszym z nich jest **ograniczenie (wymagania) rozłączności**, które określa, że wszystkie podklasy danej specjalizacji muszą być rozłączne. Oznacza to, że poszczególne encje mogą należeć *najwyżej* do jednej podklasy danej specjalizacji. Specjalizacja zdefiniowana przez argument jest równoznaczna z ograniczeniem rozłączności w sytuacji, gdy atrybut wykorzystywany

<sup>6</sup> W terminologii UML taki atrybut jest nazywany *dyskryminatorem*.



do definiowania predykatu przynależności jest jednowartościowy. Na rysunku 4.4 przedstawiono przypadek, w którym litera **d** w okręgu reprezentującym specjalizację oznacza właśnie *rozłączność* (od ang. *disjoint*). Notację z literą **d** wykorzystujemy także do określania ograniczenia rozłączności dla specjalizacji z podklasami zdefiniowanymi przez użytkownika (patrz specjalizacja z podklasami {PRACOWNIK\_ETATOWY, PRACOWNIK\_SEZONOWY} na rysunku 4.1). Gdybyśmy na wspomniane podklasy nie nałożyli ograniczenia rozłączności, zbiory ich encji mogłyby się częściowo lub w całości **pokrywać**; oznacza to, że ta sama encja ze świata rzeczywistego mogłaby należeć do więcej niż jednej podklasy tej specyfikacji. W takim przypadku (który jest rozwiązaniem domyślnym) w okręgu reprezentującym specjalizację umieszcza się literę **o** (patrz rysunek 4.5).



RYСУNEK 4.5. Notacja diagramu EER dla częściowo pokrywających się (nierozłącznych) specjalizacji

Drugie ograniczenie nakładane na specjalizacje nosi nazwę **ograniczenia (wymagania) kompletności**, które może być pełne lub częściowe. Ograniczenie **kompletnej specjalizacji** oznacza, że *każda* encja w nadklasie musi należeć przynajmniej do jednej podklasy tej specyfikacji. Przykładowo, jeśli każdy PRACOWNIK musi być albo pracować na pełnym etacie, albo być zatrudnionym na czas określony (wspomniane rodzaje zatrudnienia są reprezentowane przez podklasy PRACOWNIK\_ETATOWY i PRACOWNIK\_SEZONOWY), wówczas specjalizacja {PRACOWNIK\_ETATOWY, PRACOWNIK\_SEZONOWY} z rysunku 4.1 musi być kompletna względem nadklasy PRACOWNIK. Ograniczenie kompletności jest reprezentowane na diagramach EER za pomocą podwójnych linii łączących nadklasę z okręgiem reprezentującym samą specjalizację. Linia pojedyncza jest wykorzystywana do przedstawiania **częściowej specjalizacji** dopuszczającej możliwość występowania encji, która nie należy do żadnej z podklas. Przykładowo, jeśli jakieś encje typu PRACOWNIK nie należą do żadnej z podklas {SEKRETARKA, INŻYNIER, TECHNIK} z rysunków 4.1 i 4.4, wówczas mamy do czynienia z częściową specjalizacją<sup>7</sup>.

Warto pamiętać, że ograniczenia rozłączności i kompletności są od siebie *niezależne*. Oznacza to, że istnieją cztery możliwe kombinacje ograniczeń nakładanych na specjalizacje:

- rozłączne, kompletne;
- rozłączne, częściowe;
- pokrywające się, kompletne;
- pokrywające się, częściowe.

<sup>7</sup> Wykorzystywana na diagramach EER notacja z pojedynczymi i podwójnymi liniami przypomina notację stosowaną do reprezentowania częściowego lub całkowitego udziału typu encji w typie związku (patrz rozdział 3.).



Prawidłowe ograniczenie jest oczywiście określane w oparciu o znaczenie poszczególnych specjalizacji w rzeczywistym świecie. Generalnie nadklasa, która została zdefiniowana w procesie *generalizacji*, najczęściej jest **kompletna**, ponieważ *wywodzi się* z podklas, a więc może zawierać tylko encje zawarte w tych podklasach.

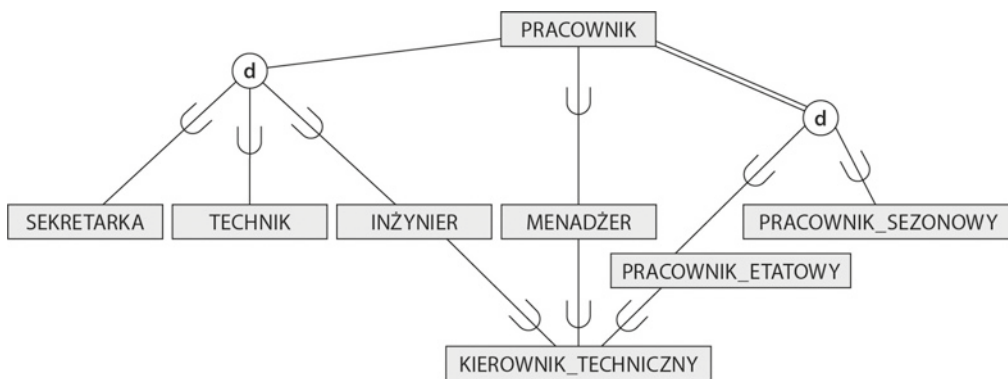
Konsekwencją wymienionych przed chwilą ograniczeń są pewne reguły wstawiania i usuwania encji występujących w procesach specjalizacji i generalizacji. Niektóre z tych reguł zawarto poniżej:

- Usuwanie encji z nadklasy wiąże się z koniecznością jej automatycznego usunięcia ze wszystkich podklas, w których występuje.
- Wstawienie encji do nadklasy wiąże się z obowiązkiem jej wstawienia we wszystkich tych podklasach *zdefiniowanych przez predykat (przez atrybut)*, w przypadku których dana encja spełnia predykat definiujący.
- Wstawienie encji do nadklasy *kompletnej specjalizacji* wiąże się z koniecznością wstawienia tej encji w przynajmniej jednej z podklas tej specjalizacji.

Zachęcamy Czytelnika do opracowania pełnej listy reguł wstawiania i usuwania encji w poszczególnych typach specjalizacji.

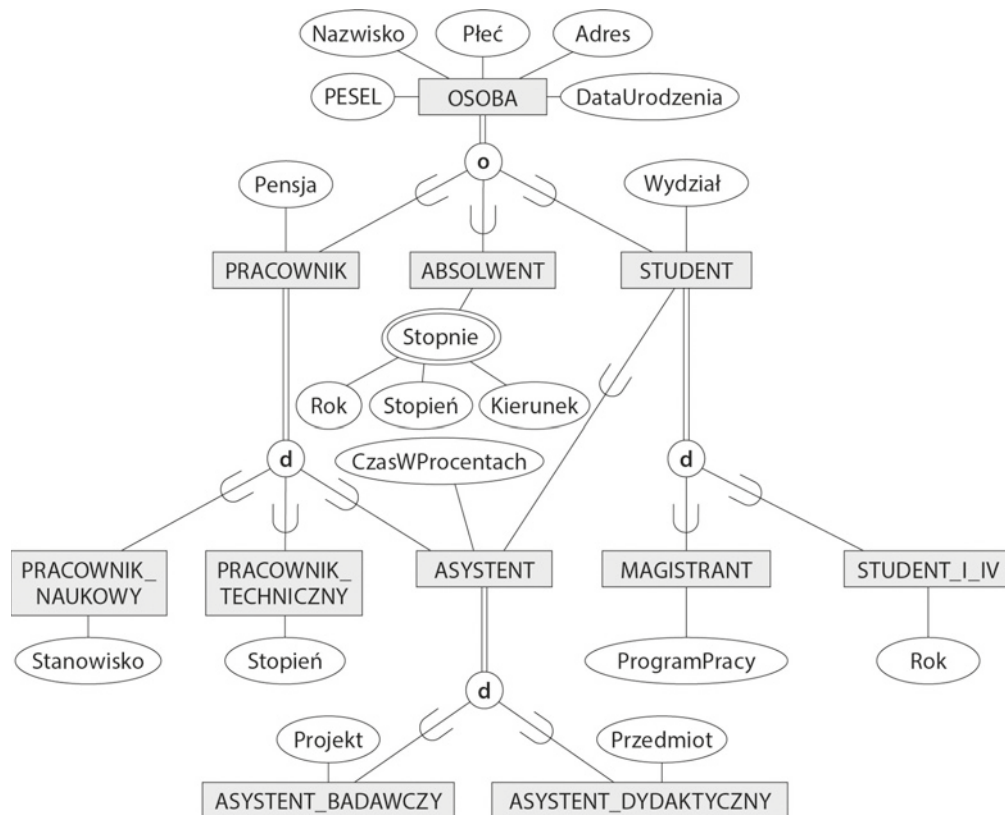
### 4.3.2. Hierarchie i kraty specjalizacji i generalizacji

Sama podklasa może mieć zdefiniowane dalsze podklasy — w ten sposób możemy stworzyć całą hierarchię lub kratę specjalizacji. Przykładowo, na rysunku 4.6 przedstawiono typ encji INŻYNIER będący jednocześnie podklasą typu PRACOWNIK i nadklasą dla podklasy KIEROWNIK\_TECHNICZNY; w ten sposób reprezentujemy istniejące w świecie rzeczywistym ograniczenie, które wymaga od kierownika technicznego stopnia inżyniera. Z **hierarchią specjalizacji** wiąże się istotne ograniczenie, które mówi, że każda podklasa może występować *w tej roli tylko w jednym* związku klasy-podklasy; oznacza to, że żadna z podklas nie może mieć więcej niż jedną klasę macierzystą, co prowadzi do otrzymania **struktury drzewiastej**. Inaczej jest w przypadku **krat specjalizacji**, w przypadku których każda podklasa może występować w roli podklasy *w więcej niż jednym* związku klasy-podklasy. Oznacza to, że na rysunku 4.6 przedstawiono właśnie kratę specjalizacji.



RYСУNEK 4.6. Krata specjalizacji z dzieloną podklasą KIEROWNIK\_TECHNICZNY

Na rysunku 4.7 przedstawiono jeszcze inną kratę specjalizacji, w której występuje więcej niż jeden poziom. Zaprezentowany diagram może być fragmentem schematu koncepcyjnego bazy danych UNIWERSYTET. Łatwo zauważyć, że gdyby zaprezentowany schemat nie zawierał podklasy ASYSTENT (występującej w roli podklasy w dwóch różnych związkach klasy-nadklasy), mógłby tworzyć hierarchię specjalizacji.



RYСУNEK 4.7. Krata specjalizacji z wielokrotnym dziedziczeniem w bazie danych UNIWERSYTET

Oto wymagania dotyczące fragmentu bazy UNIWERSYTET przedstawionego na rysunku 4.7:

- (1) Baza przechowuje dane trzech typów osób: pracowników, absolwentów i studentów. Każda osoba może należeć do jednej, dwóch lub wszystkich trzech tych kategorii. Dla każdej osoby określone są: imię i nazwisko, PESEL, płeć, adres i data urodzenia.
- (2) Każdy pracownik ma wynagrodzenie. Występują trzy typy pracowników: pracownicy naukowcy, pracownicy techniczni i studenci-asystenci. Dla każdego absolwenta przechowywany jest zapis stopni uzyskanych na uniwersytecie (w tym nazwa stopnia, rok przyznania i kierunek studiów). Każdy student ma określony kierunek.
- (3) Każdy pracownik naukowy ma określony stopień, a każdy pracownik techniczny ma stanowisko. Asystenci są podzieleni na badawczych i dydaktycznych, a w bazie zapisany jest (w procentach) czas, jaki poświęcają na pracę. Dla asystentów

badawczych przechowywane są projekty, jakimi się zajmują, a dla asystentów naukowych — kursy, przy których pracują.

- (4) Studenci są dodatkowo podzieleni na magistrantów i studentów lat I – IV. Magistranci mają określony stopień, na jaki pracują (magister, doktor, MBA itd.), a dla studentów młodszych lat określony jest rok (pierwszy, drugi itd.).

Na rysunku 4.7 wszystkie encje reprezentujące w bazie danych osoby należą do typu encji OSOBA, który jest specjalizowany do podklas {PRACOWNIK, ABSOLWENT, STUDENT}. Wspomniana specjalizacja częściowo się pokrywa — przykładowo, absolwent może być nie tylko pracownikiem, ale także studentem na studiach dających wyższy tytuł naukowy (magisterskich lub doktoranckich). Podklasa STUDENT jest jednocześnie nadklasą dla specjalizacji {MAGISTRANT, STUDENT\_I\_IV}, natomiast typ encji PRACOWNIK jest nadklasą dla specjalizacji {ASYSTENT, PRACOWNIK\_NAUKOWY, PRACOWNIK\_TECHNICZNY}. Łatwo zauważyć, że typ encji ASYSTENT jest jednocześnie podklasą typu STUDENT. I wreszcie typ encji ASYSTENT jest nadklasą dla specjalizacji {ASYSTENT\_BADAWCZY, ASYSTENT\_DYDAKTYCZNY}.

W tego typu kracie lub hierarchii specjalizacji każda podklasa dziedziczy atrybuty nie tylko ze swojej bezpośredniej nadklasy, ale także pośrednio ze wszystkich jej nadklas *aż do samego korzenia* hierarchii lub kraty. Przykładowo, każda encja klasy MAGISTRANT dziedziczy nie tylko wszystkie atrybuty odpowiedniej encji nadklasy STUDENT, *ale także wszystkie atrybuty nadklasy OSOBA*. Warto pamiętać, że pojedyncza encja może istnieć w wielu węzłach liści danej hierarchii, gdzie **węzeł liścia** jest taką klasą, która *nie ma własnych podklas*. Przykładowo, egzemplarz klasy MAGISTRANT może być jednocześnie egzemplarzem klasy ASYSTENT\_BADAWCZY.

Podklasa z więcej niż jedną nadklasą jest nazywana **podklasą dzieloną** (przykładem takiej podklasy jest KIEROWNIK\_TECHNICZNY z rysunku 4.6). W ten sposób doszliśmy do pojęcia nazywanego **wielokrotnym dziedziczeniem**, czyli takim, w którym dzielona klasa KIEROWNIK\_TECHNICZNY dziedziczy atrybuty i związki bezpośrednio z wielu klas. Warto zauważyć, że istnienie choćby jednej dzielonej podklasy powoduje, że mamy do czynienia z kratą specjalizacji (a więc także z *wielokrotnym dziedziczeniem*); jeśli natomiast schemat nie zawiera żadnych dzielonych podklas, możemy mówić wyłącznie o istnieniu hierarchii (nie kraty) dziedziczenia i **jednokrotnym dziedziczeniu**. Widoczna na rysunku 4.7 dzielona podklasa ASYSTENT ilustruje ważną regułę związaną z wielokrotnym dziedziczeniem atrybutów (w tym przypadku z nadklas PRACOWNIK i STUDENT). Tu *zarówno* klasa PRACOWNIK, *jak i* klasa STUDENT dziedziczy *te same atrybuty* z nadklasy OSOBA. Wspomniana reguła mówi, że jeśli jakiś atrybut (lub związek) pochodzi z *tej samej nadklasy* (w tym przypadku z nadklasy OSOBA) jest dziedziczony na różnych ścieżkach dziedziczenia więcej niż raz (w tym przypadku przez podklasy PRACOWNIK i STUDENT) w kracie specjalizacji, atrybut ten (lub związek) powinien występować w dzielonej podklasie (w tym przypadku podklasie ASYSTENT) *tylko raz*. Oznacza to, że atrybuty nadklasy OSOBA powinny być dziedziczone *tylko raz* w podklasie ASYSTENT (patrz rysunek 4.7).

Warto w tym miejscu nadmienić, że niektóre modele i języki obsługują tylko **jednokrotne dziedziczenie** w ogóle *nie dają możliwości* stosowania wielokrotnego dziedziczenia (i tym samym dzielonych podklas). Należy także pamiętać, że niektóre mechanizmy dziedziczenia, które umożliwiają dziedziczenie wielokrotne, nie zezwalają na stosowanie wielu typów dla pojedynczych encji, co oznacza, że każda z encji musi należeć *tylko do jednej klasy*<sup>8</sup>.

<sup>8</sup> W niektórych modelach pojęcie klasy jest dodatkowo ograniczane do *węzła liścia* w hierarchii lub kracie specjalizacji (lub generalizacji).

W takim modelu niezbędne jest tworzenie dodatkowych podklas obejmujących wszystkie możliwe kombinacje klas, które będą mogły reprezentować encje należące do wszystkich tych klas jednocześnie. Przykładowo, nierozłączna specjalizacja klasy OSOBA z podklasy {PRACOWNIK, ABSOLWENT, STUDENT} (lub w skrócie {P, A, S}) będzie wymagała stworzenia siedmiu kolejnych podklas klasy OSOBA, które pokryją przestrzeń wszystkich możliwych typów encji: P, A, S, P\_A, P\_S, A\_S oraz P\_A\_S. Takie rozwiązanie prowadzi oczywiście do znacznego zwiększenia złożoności schematu.

Chociaż w tym punkcie prezentowaliśmy pewne zagadnienia wyłącznie w oparciu o przykładowe procesy specjalizacji, bardzo podobne mechanizmy *mają takie samo zastosowanie* w przypadku procesów generalizacji (zgodnie z tym, o czym wspominaliśmy na początku tego podrozdziału). Oznacza to, że równie dobrze możemy mówić o **hierarchiach** i **kra- tach generalizacji**.

### 4.3.3. Stosowanie procesów specjalizacji i generalizacji podczas udoskonalania schematów koncepcyjnych

W tym punkcie nie tylko przedstawimy różnice pomiędzy procesami specjalizacji i generalizacji, ale także omówimy sposoby ich wykorzystywania do udoskonalania schematów koncepcyjnych w trakcie koncepcyjnego projektowania baz danych. Proces specjalizacji polega zwykle na analizie istniejącego typu encji i definiowaniu jego coraz bardziej szczegółowych podklas; oznacza to, że możemy wielokrotnie grupować reprezentowane encje w postaci coraz bardziej precyzyjnych typów encji. Przykładowo, podczas projektowania widocznej na rysunku 4.7 kraty specjalizacji mogliśmy w pierwszej kolejności określić ogólny typ encji OSOBA dla właśnie tworzonej uniwersyteckiej bazy danych. Następnie mogliśmy stwierdzić, że projektowana baza danych powinna reprezentować trzy typy osób: pracowników uniwersytetu, absolwentów i studentów. Stworzyliśmy więc specjalizację {PRACOWNIK, ABSOLWENT, STUDENT} i wprowadziliśmy ograniczenie nakładania się (nierozłączności), ponieważ encje reprezentujące jedną osobę mogą należeć do więcej niż jednej z wymienionych podklas. Następnie przeprowadziliśmy specjalizację typów PRACOWNIK: {ASYSTENT, PRACOWNIK\_NAUKOWY, PRACOWNIK\_TECHNICZNY} i STUDENT: {MAGISTRANT, STUDENT\_I\_IV}. Wreszcie wdrożyliśmy proces specjalizacji dla typu encji ASYSTENT, który podzieliłiśmy na dwie podklasy: {ASYSTENT\_BADAWCZY, ASYSTENT\_DYDAKTYCZNY}. Jest to proces **koncepcyjnego udoskonalania z góry na dół**. Do tej pory projekt zawierał hierarchię, jednak typ encji ASYSTENT jest w istocie dzieloną podklasą, ponieważ dziedziczy także z nadklasy STUDENT — ostatecznie otrzymaliśmy więc kratę specjalizacji.

Istnieje możliwość dojścia do takiej samej hierarchii lub kraty specjalizacji w zupełnie inny sposób (z innego kierunku). Cały proces opiera się wówczas na stosowaniu generalizacji zamiast specjalizacji, a więc wykonywane działania odpowiadają procesowi **koncepcyjnego udoskonalania z dołu do góry**. W naszym przypadku projektanci mogliby najpierw zdefiniować takie typy encji jak PRACOWNIK\_NAUKOWY, PRACOWNIK\_TECHNICZNY, ABSOLWENT, MAGISTRANT, STUDENT\_I\_IV, ASYSTENT\_BADAWCZY, ASYSTENT\_DYDAKTYCZNY itp.; dopiero potem projektanci mogą przystąpić do generalizowania pary {MAGISTRANT, STUDENT\_I\_IV} do postaci nadklasy STUDENT, pary {ASYSTENT\_BADAWCZY, ASYSTENT\_DYDAKTYCZNY} do postaci nadklasy ASYSTENT, trójki {ASYSTENT, PRACOWNIK\_NAUKOWY, PRACOWNIK\_TECHNICZNY} do postaci nadklasy PRACOWNIK i wreszcie trójki {PRACOWNIK, ABSOLWENT, STUDENT} do postaci najbardziej ogólnej nadklasy OSOBA.

Ostateczne hierarchie lub kraty uzyskane w oparciu o oba procesy mogą być identyczne; jedyna różnica może wynikać z innego sposobu lub innej kolejności definiowania nadklas lub podklas. W praktyce przeważnie wykorzystuje się kombinację obu procesów. Warto zauważyć, że metoda reprezentowania danych i wiedzy w oparciu o hierarchie i kraty nadklas-podklas jest powszechnie stosowana w systemach operujących na bazach wiedzy oraz systemach eksperckich, które łączą w sobie technologie baz danych z technikami sztucznej inteligencji. Przykładowo, schematy reprezentacji wiedzy oparte na ramach bardzo przypominają prezentowane w tym podrozdziale hierarchie klas. Stosowanie specjalizacji jest popularne także w obiektowych metodach projektowania wykorzystywanych w inżynierii oprogramowania.

## 4.4. Modelowanie typów UNII w oparciu o kategorie

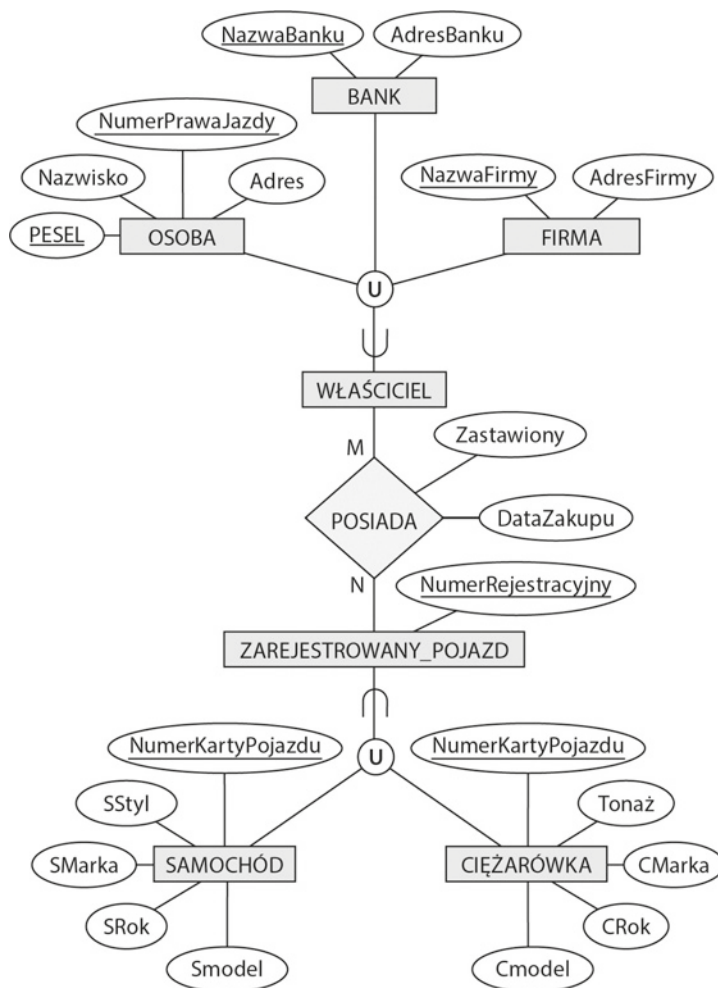
Czasem konieczne jest utworzenie modelu zbioru encji różnych typów. W takich przypadkach podklasa musi reprezentować zbiór obiektów stanowiący podzbiór UNII różnych typów encji — taką *podklasę* nazywamy **typem unii** lub **kategorią**<sup>9</sup>.

Przykładowo, przypuśćmy, że mamy w schemacie następujące trzy typy encji: OSOBA, BANK i FIRMA. W przypadku bazy danych obsługującej rejestrację pojazdów właścicielem samochodu może być osoba prywatna, bank (w przypadku majątku stanowiącego zastaw pożyczki lub kredytu) lub firma. Musimy stworzyć klasę (zbiór encji), która będzie zawierała encje wszystkich trzech typów i reprezentowała rolę *właściciela pojazdu*. W tym celu powinniśmy stworzyć kategorię WŁAŚCICIEL, która będzie *podklasą* UNII wspomnianych trzech zbiorów encji (FIRMA, BANK i OSOBA). Kategorie przedstawia się także na diagramach EER (patrz rysunek 4.8). Nadklasy FIRMA, BANK i OSOBA są połączone z okręgiem zawierającym symbol U, który reprezentuje *operację na zbiorze unii*. Łuk oznaczony tym symbolem łączy ten okrąg z kategorią (podklasą) WŁAŚCICIEL. Na rysunku 4.8 przedstawiono dwie kategorie: kategorię WŁAŚCICIEL, która jest podklasą unii typów encji OSOBA, BANK i FIRMA, oraz kategorię ZAREJESTROWANY\_POJAZD, która jest podklasą unii typów encji SAMOCHÓD i CIĘŻARÓWKA.

Kategoria zawiera dwie lub więcej nadklas, które mogą reprezentować *różne typy encji* — wszystkie pozostałe związki nadklas-podklas zawsze zawierają po jednej nadklasie. Aby lepiej zrozumieć tę różnicę, możemy porównać kategorię (choćby przedstawioną na rysunku 4.8 kategorię WŁAŚCICIEL) np. z dzieloną podklasą KIEROWNIK\_TECHNICZNY z rysunku 4.6. KIEROWNIK\_TECHNICZNY jest osobną podklasą dla *każdej* z trzech swoich nadklas (INŻYNIER, KIEROWNIK i PRACOWNIK\_ETATOWY), zatem encja należąca do typu KIEROWNIK\_TECHNICZNY musi występować we *wszystkich trzech zbiorach*. W ten sposób reprezentujemy wymaganie (ograniczenie), które mówi, że każda osoba na stanowisku kierownika technicznego (a więc reprezentowana przez encję typu KIEROWNIK\_TECHNICZNY) musi być jednocześnie inżynierem, kierownikiem i pracownikiem etatowym (odpowiednio INŻYNIER, KIEROWNIK i PRACOWNIK\_ETATOWY); oznacza to, że KIEROWNIK\_TECHNICZNY jest podzbiorem *części wspólnej* tych trzech podklas (zbiorów encji). Z drugiej strony, kategoria jest podzbiorem *unii*, do

<sup>9</sup> Stosowane w tym kontekście pojęcie *kategorii* zostało zaproponowane przez Elmasri'ego i in. (1985) wraz z modelem związków kategorii i encji (ang. *Entity-Category-Relationship* — ECR).

której należą jej nadklasy. Tak więc encja typu WŁAŚCICIEL musi występować w *tylko jednej* z jej nadklas. W taki właśnie sposób reprezentujemy ograniczenie, które mówi, że właściciel może być firmą, bankiem *lub* osobą prywatną (patrz rysunek 4.8).



RYСУNEK 4.8. Dwie kategorie (typy unii): WŁAŚCICIEL oraz ZAREJESTROWANY\_POJAZD

Dziedziczenie atrybutów jest w przypadku kategorii znacznie bardziej selektywne. Przykładowo, na rysunku 4.8 każda encja typu WŁAŚCICIEL dziedziczy atrybuty typu encji FIRMA, OSOBA albo BANK w zależności od tego, do której nadklasy ta encja należy. Z drugiej strony taka dzielona podklasa jak KIEROWNIK\_TECHNICZNY (patrz rysunek 4.6) dziedziczy *wszystkie* atrybuty nadklas PRACOWNIK\_ETATOWY, INŻYNIER i KIEROWNIK.

Warto zwrócić uwagę na różnicę pomiędzy kategorią ZAREJESTROWANY\_POJAZD (patrz rysunek 4.8) a uogólnioną (zgeneralizowaną) nadklasą POJAZD (rysunek 4.3(b)). Na rysunku 4.3(b) każdy samochód i każda ciężarówka jest encją klasy POJAZD, natomiast na rysunku



4.8 kategoria ZAREJESTROWANY\_POJAZD obejmuje tylko część samochodów i ciężarówek, ale niekoniecznie wszystkie (niektóre samochody i ciężarówki mogą nie być zarejestrowane). Gdyby takie specjalizacje i generalizacje jak te przedstawione na rysunku 4.3(b) były *częściowe*, nie moglibyśmy wykluczyć możliwości obejmowania przez klasę POJAZD innych typów encji, np. reprezentujących motocykle. Kategorie takie jak ZAREJESTROWANY\_POJAZD (patrz rysunek 4.8) reprezentują jednak wyłącznie samochody i ciężarówki — do zdefiniowanej w ten sposób kategorii nie mogą należeć żadne inne typy encji.

Kategoria może być **kompletna** lub **częściowa**. Kategoria kompletna przechowuje *unię* wszystkich encji należących do jej nadklas, natomiast kategoria częściowa przechowuje jedynie *podzbiór tej unii*. Kategoria kompletna jest przedstawiana na diagramach za pomocą podwójnych linii łączących samą kategorię z okręgiem reprezentującym unię, natomiast kategorie częściowe są oznaczane umieszczanymi w tych samych miejscach liniami pojedynczymi.

Nadklasy kategorii mogą mieć albo różne atrybuty klucza (patrz kategoria WŁAŚCICIEL na rysunku 4.8), albo takie same atrybuty klucza (patrz kategoria ZAREJESTROWANY\_POJAZD na tym samym rysunku). Należy pamiętać, że jeśli dana kategoria jest kompletna (nieczęściowa), możliwe jest zastosowanie alternatywnej reprezentacji w postaci kompletnej specjalizacji (lub kompletnej generalizacji). W takim przypadku wybór stosowanej reprezentacji zależy od subiektywnej oceny projektanta. Jeśli dwie klasy reprezentują ten sam typ encji i dzielą wiele atrybutów (włącznie z tymi samymi atrybutami klucza), zaleca się użycie procesu specjalizacji-generalizacji; w przeciwnym przypadku lepszym rozwiązaniem jest zastosowanie kategoryzacji (typu unii).

Warto zauważyć, że w niektórych metodach modelowania nie występują typy unii. W takich modelach typ unii trzeba przedstawiać w niestandardowy sposób (patrz podrozdział 9.2).

## 4.5. Przykład schematu EER dla bazy danych UNIwersytet oraz formalne definicje dla modelu EER

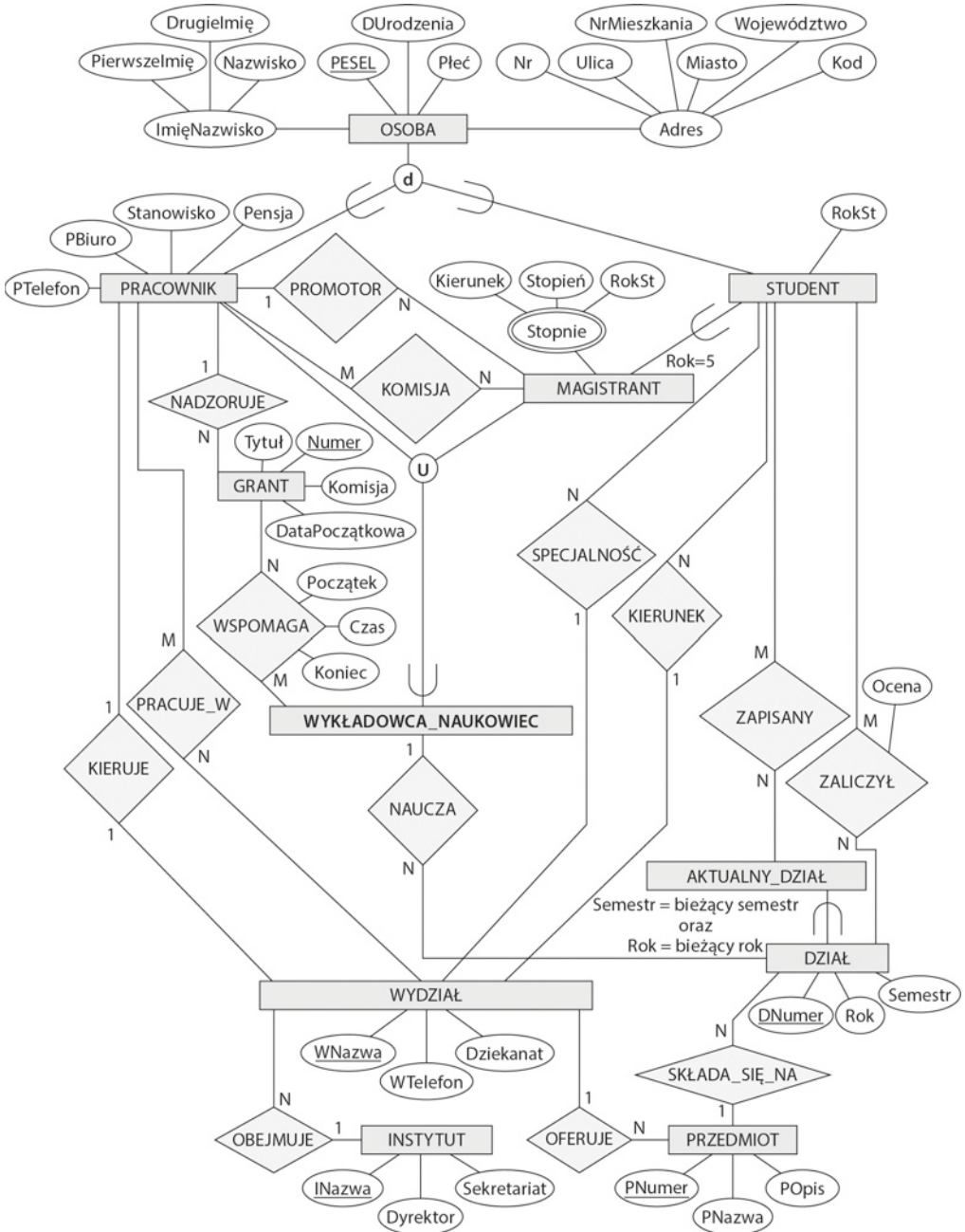
W tym podrozdziale w pierwszej kolejności zaprezentujemy zbudowany w oparciu o model danych EER przykład schematu bazy danych, który dobrze ilustruje możliwości stosowania rozmaitych mechanizmów omówionych w rozdziale 3. Dalej omówimy wybory projektowe związane ze schematami koncepcyjnymi, a następnie podsumujemy pojęcia wykorzystywane w modelu EER i zdefiniujemy je w sposób formalny — w taki sam sposób, jak w rozdziale 3. formalnie zdefiniowaliśmy elementy podstawowego modelu ER.

### 4.5.1. Inny przykład bazy danych UNIwersytet

Wyobraź sobie bazę UNIwersytet z *innymi wymaganiami* niż te stawiane bazie z podrozdziału 3.10. Nowa baza zawiera informacje o studentach, wybranych przez nich kierunkach studiów, rejestrze ocen i zapisach, a także dane na temat przedmiotów oferowanych na danym uniwersytecie. Dodatkowo nasza baza danych przechowuje informacje na temat projektów badawczych prowadzonych przez kadrę naukową i studentów piątego roku.



Cały schemat bazy danych przedstawiono na rysunku 4.9. Poniżej znajduje się omówienie wymagań, których analiza doprowadziła do opracowania tego schematu.



RYСУNEK 4.9. Schemat koncepcyjny EER dla innej bazy danych UNIWERSYTET

Dla każdej osoby baza danych utrzymuje informacje reprezentujące nazwisko (atrybut *ImięNazwisko*), numer PESEL (atrybut *Pesel*), adres (atrybut *Adres*), płeć (atrybut *Płeć*) oraz datę urodzenia (atrybut *DataUr*). Podczas analizy wymagań projektant bazy danych zidentyfikował dwie podklasy dla typu encji OSOBA: PRACOWNIK\_NAUKOWY i STUDENT. Podklasa PRACOWNIK\_NAUKOWY zawiera następujące atrybuty: Stanowisko (*asystent, adiunkt, pracownik dydaktyczny, pracownik naukowy itp.*), BiuroPracN, TelefonPracN oraz Pensja. Każdy pracownik uniwersytetu jest przypisany (związek PRACUJE\_W) do wydziału lub wydziałów akademickich (ponieważ pojedynczy pracownik może współpracować z wieloma wydziałami, liczność związku PRACUJE\_W wynosi  $M:N$ ). Atrybutem charakterystycznym dla podklasy STUDENT jest RokSt (*pierwszy = 1, drugi = 2, ..., ostatni = 5*). Każdy student jest dodatkowo powiązany ze swoim głównym wydziałem oraz (jeśli istnieje) wybranym wydziałem specjalności (odpowiednio związki KIERUNEK i SPECJALNOŚĆ), z aktualnie omawianymi kursami z przedmiotów (typ związku ZAPISANY), a także z zaliczonymi kursami z poszczególnych przedmiotów (typ związku REJESTR\_OCEN). Każdy egzemplarz związku REJESTR\_OCEN zawiera ocenę uzyskaną przez studenta z danego kursu (atrybut *Ocena*).

MAGISTRANT jest podklasą klasy STUDENT z następującym predykatem definiującym: *RokSt = 5*. Dla każdego takiego studenta utrzymujemy listę uzyskanych ocen w postaci złożonego, wielowartościowego atrybutu *Oceny*. Wszystkich studentów piątego roku dodatkowo wiążemy z promotorami przydzielonymi do prowadzenia ich prac magisterskich (typ związku PROMOTOR) oraz z wyznaczonymi (opcjonalnie) komisjami egzaminacyjnymi (typ związku KOMISJA).

Typ encji dla wydziałów akademickich zawiera atrybuty reprezentujące nazwę (*WNazwa*), numer telefonu (*WTelefon*) oraz numer dziekanatu (*Dziekanat*). Dodatkowo, typ encji WYDZIAŁ jest powiązany odpowiednim związkiem z jednym z pracowników naukowych, który pełni funkcję dziekana tego wydziału (typ związku KIERUJE). Wydział występuje także w związku 1:N (typ związku OBEJMUJE) z instytutami (typ encji INSTYTUT), które z kolei są opisywane przez nazwę (*INazwa*), numer sekretariatu (*Sekretariat*) oraz nazwisko dyrektora (*Dyrektor*).

Każdy przedmiot (egzemplarz typu encji PRZEDMIOT) jest reprezentowany za pomocą unikatowego numeru (*PNumer*), nazwy (*PNazwa*) oraz opisu (*POpis*). Poszczególne przedmioty dzielą się na wiele działów (egzemplarze typu encji DZIAŁ), z których każdy jest opisywany przez numer (*DNumer*), rok i semestr prowadzenia działu (odpowiednio atrybuty *Rok* i *Semestr*)<sup>10</sup>. Numery działów unikatowo identyfikują poszczególne encje typu DZIAŁ. Działy oferowane w bieżącym semestrze znajdują się w podklasie BIEŻĄCY\_DZIAŁ klasy DZIAŁ — odpowiednia specjalizacja jest definiowana przez predykat *Semestr = BieżącySemestr* oraz *Rok = BieżącyRok*. Każdy kurs jest powiązany z pracownikiem dydaktycznym, który prowadził lub prowadzi odpowiednie wykłady (związek NAUCZA), jeśli tylko dane wykładowcy znajdują się w bazie danych.

Kategoria WYKŁADOWCA\_NAUKOWIEC jest podzbiorem unii klas PRACOWNIK i MAGISTRANT, zatem zawiera wszystkie encje reprezentujące tych pracowników i studentów piątego roku, którzy prowadzą zajęcia lub biorą udział w badaniach naukowych. I wreszcie typ encji GRANT reprezentuje informacje opisujące granty przyznawane uniwersytetowi na badania i dydaktykę. Każda encja typu GRANT zawiera takie atrybuty jak tytuł grantu (*Tytuł*), numer

<sup>10</sup> Zakładamy, że na danym uniwersytecie obowiązuje system *semestralny*, a nie np. system *kwartalny*.

grantu (Numer), instytucja zarządzająca (Komisja) oraz data początkowa (DataPoczątkowa). Grant jest powiązany z pracownikiem nadzorującym (typ związku NADZORUJE) oraz wszystkimi pracownikami naukowymi, którzy korzystają z tego grantu (typ związku WSPOMAGA). Każdy egzemplarz związku WSPOMAGA zawiera atrybuty reprezentujące datę początkową wykorzystywania grantu (Początek), datę końcową (jeśli jest już znana — atrybut Koniec) oraz wyrażony w procentach czas poświęcony na projekt przez pracowników i studentów, którzy korzystają z danego grantu (atrybut Czas).

## 4.5.2. Wybory projektowe związane ze specjalizacją i generalizacją

Nie zawsze łatwo jest wybrać najodpowiedniejszy projekt koncepcyjny dla aplikacji bazy danych. W punkcie 3.7.3 przedstawiliśmy niektóre typowe problemy, z jakimi styka się projektant bazy danych, gdy wybiera typy encji, typy związków i atrybuty reprezentujące określony mini-świat na schemacie ER. W tym punkcie omawiamy wytyczne i wybory projektowe dotyczące specjalizacji, generalizacji i kategorii (typów unii) w modelu EER.

Jak wspomniano w punkcie 3.7.3, tworzenie koncepcyjnego projektu bazy należy traktować jak iteracyjny proces doskonalenia go do momentu uzyskania najodpowiedniejszego rozwiązania. Przedstawione tu wytyczne mogą pomóc w procesie projektowania komponentów modelu EER.

- Zwykle można zdefiniować wiele specjalizacji i podklas, aby zapewnić precyzję modelu koncepcyjnego. Jednak wadą tego podejścia jest to, że projekt może stać się mało przejrzysty. Ważne jest, aby uwzględniać tylko te podklasy, które są niezbędne. Pozwala to uniknąć nadmiernego przeładowania schematu koncepcyjnego.
- Jeśli podklasa ma niewiele specyficznych (lokalnych) atrybutów i nie ma specyficznych związków, można ją połączyć z nadklasą. W encjach, które nie należą do danej podklasy, jej specyficzne atrybuty będą miały wartość pustą. Można też zastosować atrybut *typu*, określający, czy encja należy do określonej podklasy.
- Podobnie jeśli wszystkie podklasy w hierarchii specjalizacji i generalizacji mają niewiele specyficznych atrybutów i nie mają specyficznych związków, można je scalić w nadklasę i zastosować jeden lub kilka atrybutów *typu*, określających, do jakich podklas należy każda encja (w podrozdziale 9.2 opisano, jak to kryterium jest wykorzystywane w bazach relacyjnych).
- Typów unii i kategorii zwykle należy unikać, chyba że sytuacja jednoznacznie uzasadnia ich stosowanie (w praktyce zdarza się to w pewnych scenariuszach). W miarę możliwości staramy się tworzyć modele z użyciem specjalizacji i generalizacji, co opisano pod koniec podrozdziału 4.4.
- Stosowanie rozłącznych lub pokrywających się i pełnych lub częściowych ograniczeń specjalizacji i generalizacji wynika z reguł obowiązujących w modelowanym mini-świecie. Jeśli wymagania nie wskazują na konkretne rozwiązanie, domyślnie stosuje się ograniczenia pokrywające się i częściowe (ponieważ nie nakładają żadnych ograniczeń na przynależność do podklas).

W ramach przykładu zastosowania tych wytycznych przyjrzyj się rysunkowi 4.6, gdzie nie pokazano atrybutów specyficznych (lokalnych). Wszystkie podklasy można tu scalić w typ encji PRACOWNIK i dodać do niego następujące atrybuty:

- Atrybut `Typ_pracy` o zbiorze wartości {'Sekretarka', 'Inżynier', 'Technik'} określa, do której podklasy z pierwszej specjalizacji należy każdy pracownik.
- Atrybut `Typ_wynagrodzenia` o zbiorze wartości {'Etat', 'Godziny'} informuje, do której klasy z drugiej specjalizacji należy każdy pracownik.
- Atrybut `Jest_menedżerem` o zbiorze wartości {'Tak', 'Nie'} określa, czy pracownik reprezentowany przez encję jest menedżerem.

### 4.5.3. Formalne definicje pojęć stosowanych w modelu EER

Spróbujemy teraz podsumować pojęcia typowe dla rozszerzonego modelu związków encji (EER) i przedstawić ich formalne definicje. **Klasa**<sup>11</sup> definiuje typ encji i reprezentuje zbiór encji tego typu; pojęcie klasy obejmuje więc dowolne konstrukcje schematu EER, które grupują encje — w tym typy encji, podklasy, nadklasy i kategorie. **Podklasa**  $S$  jest klasą, której encje zawsze muszą stanowić podzbiór encji innej klasy, nazywanej nadklasą  $C$  **związku nadklasy-podklasy** (nazywanego także **związkiem JEST**). Zapisujemy taki związek w postaci  $C/S$ . Aby związek pomiędzy dwiema klasami  $C$  i  $S$  faktycznie był związkiem nadklasy-podklasy, musi być spełniony następujący warunek:

$$S \subseteq C.$$

**Specjalizacją**  $Z = \{S_1, S_2, \dots, S_n\}$  nazywamy zbiór podklas mających wspólną nadklasę  $G$ ; oznacza to, że  $G/S_i$  jest związkiem nadklasy-podklasy dla  $i = 1, 2, \dots, n$ .  $G$  jest nazywane **uogólnionym (zgeneralizowanym) typem encji (nadklasą specyfikacji lub generalizacją podklas  $\{S_1, S_2, \dots, S_n\}$ )**. Mówimy, że  $Z$  jest **kompletne**, jeśli zawsze (w dowolnym punkcie w czasie) spełniona jest równość:

$$\bigcup_{i=1}^n S_i = G$$

W przeciwnym wypadku  $Z$  jest nazywane specjalizacją **częściową**. Mówimy, że  $Z$  jest specjalizacją **rozłączną**, jeśli zawsze spełniony jest warunek:

$$S_i \cap S_j = \emptyset \text{ (zbiór pusty) dla } i \neq j.$$

W przeciwnym wypadku mówimy, że  $Z$  jest specjalizacją, która **częściowo się pokrywa**.

Mówimy, że podklasa  $S$  nadklasy  $C$  jest **zdefiniowana przez predykat**, jeśli do określania, które encje nadklasy  $C$  są składowymi podklasy  $S$ , wykorzystuje się predykat  $p$  dla atrybutów klasy  $C$ ; oznacza to, że  $S = C[p]$ , gdzie  $C[p]$  jest zbiorem tych encji w klasie  $C$ , które spełniają predykat  $p$ . Podklasa, która nie jest definiowana przez predykat, jest nazywana podklasą **zdefiniowaną przez użytkownika**.

<sup>11</sup> Znaczenie wykorzystywanego w tym kontekście słowa *klasa* różni się od jego bardziej rozpowszechnionej interpretacji w obiektowych językach programowania takich jak np. C++. W języku C++ klasa jest definicją typu strukturalnego wraz z jego funkcjami (operacjami).

Mówimy, że specjalizacja  $Z$  (lub generalizacja  $G$ ) jest **zdefiniowana przez atrybut**, jeśli istnieje predykat  $(A = c_i)$ , gdzie  $A$  jest atrybutem  $G$ , a  $c_i$  jest stałą wartością z dziedziny tego atrybutu, który jest wykorzystywany do określania przynależności do poszczególnych podklas  $S_i$  specjalizacji  $Z$ . Łatwo zauważyć, że jeśli  $c_i \neq c_j$  (dla  $i \neq j$ ) oraz  $A$  jest atrybutem jednowartościowym, wówczas specjalizacja  $Z$  będzie rozłączna.

**Kategoria**  $T$  to klasa, która jest podklasą unii  $n$  definiujących nadklas  $D_1, D_2, \dots, D_n$  (gdzie  $n > 1$ ), i którą można formalnie zdefiniować w sposób następujący:

$$T \subseteq (D_1 \cup D_2 \cup \dots \cup D_n).$$

Predykat  $p_i$  dla atrybutów nadklasy  $D_i$  może być wykorzystywany do określania przynależności do każdej nadklasy  $D_i$ , która należy do kategorii  $T$ . Jeśli taki predykat zostanie zdefiniowany dla każdej nadklasy  $D_i$ , otrzymamy:

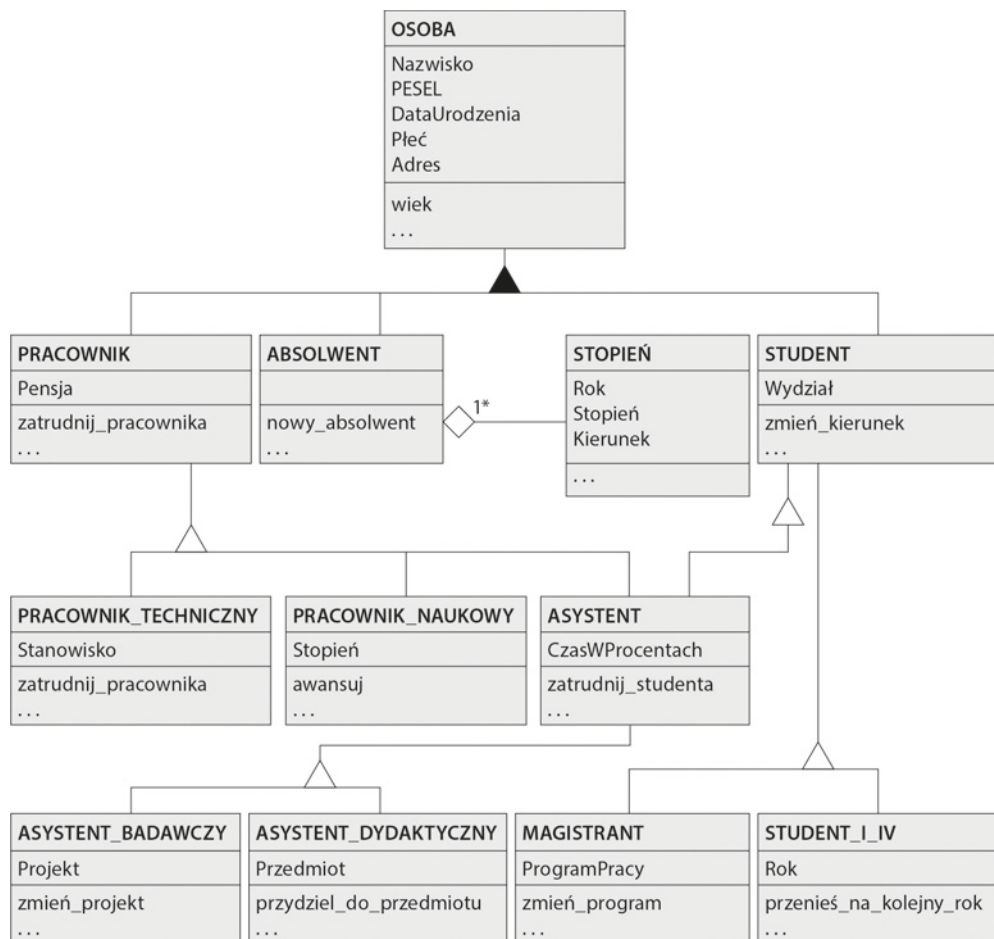
$$T = (D_1[p_1] \cup D_2[p_2] \cup \dots \cup D_n[p_n]).$$

Powinniśmy teraz rozszerzyć przedstawioną w rozdziale 3. definicję **typu związku** przez dopuszczenie możliwości udziału w związku nie tylko dowolnych typów encji, ale także dowolnych klas. Oznacza to, że powinniśmy zastąpić w tej definicji określenie *typ encji* bardziej ogólnym słowem *klasa*. Graficzna notacja dla modelu EER (diagramy EER) jest zgodna z modelem ER, ponieważ wszystkie klasy są reprezentowane za pomocą prostokątów (a więc tak samo jak typy encji).

## 4.6. Przykładowa inna notacja: reprezentowanie specjalizacji-generalizacji na diagramach klas języka UML

Omówimy teraz graficzną notację języka UML dla takich elementów jak generalizacja i specjalizacja oraz dziedziczenie. W podrozdziale 3.8 zaprezentowaliśmy już podstawową notację diagramów klas UML i terminologię stosowaną w tym języku. Na rysunku 4.10 przedstawiono jedną z możliwości w zakresie reprezentowania diagramu EER z rysunku 4.7 właśnie w postaci diagramu klas UML. Podstawowa notacja reprezentowania procesu generalizacji wymaga stosowania pionowych linii łączących podklasy z jedną linią poziomą, która jest połączona za pomocą trójkąta z inną pionową linią prowadzącą do nadklasy (patrz rysunek 4.10). Pusty trójkąt oznacza proces specjalizacji-generalizacji z ograniczeniem *rozłączności*, natomiast wypełniony trójkąt reprezentuje ten sam proces z ograniczeniem *pokrywania* (*nierozłączności*). Najwyższa nadklasa jest nazywana **klasą bazową**, natomiast węzły liści są nazywane **klasami liści**.

Powyższa analiza i przykład przedstawiony na rysunku 4.10 (a także treść podrozdziału 3.8) stanowią krótki przegląd terminologii i notacji diagramów klas języka UML. Koncentrujemy się tu na zagadnieniach powiązanych z modelami ER i EER, a nie na kwestiach dotyczących w większym stopniu inżynierii oprogramowania. W języku UML istnieje



RYSUNEK 4.10. Diagram klas UML (który ilustruje notację UML dla procesów specjalizacji-generalizacji) odpowiadający diagramowi EER z rysunku 4.7

oczywiście wiele dodatkowych szczegółów, o których nie wspominaliśmy ze względu na ograniczony zakres tematyczny tej książki, i które w przeważającej części są związane z inżynierią oprogramowania. Przykładowo, klasy mogą należeć do różnych typów:

- Klasy abstrakcyjne definiują co prawda atrybuty i operacje, jednak nie mogą zawierać własnych obiektów. Klasy abstrakcyjne są w związku z tym wykorzystywane przede wszystkim do określania zbiorów atrybutów i operacji przeznaczonych do dziedziczenia.
- Klasy konkretne mogą zawierać własne obiekty (encje), stworzone jako egzemplarze tych klas.
- Klasy szablonowe umożliwiają tworzenie szablonów, które mogą być następnie wykorzystywane do definiowania innych klas.



Podczas projektowania baz danych projektanci powinni skupiać się przede wszystkim na klasach konkretnych, w przypadku których zbiory obiektów są trwale składowane w bazach danych. W notce bibliograficznej na końcu tego rozdziału wymieniliśmy kilka ważnych publikacji, w których można znaleźć szczegółowy opis wszystkich elementów języka UML.

## 4.7. Abstrakcja danych, reprezentacja wiedzy oraz zagadnienia związane z ontologią

W tym podrozdziale omówimy na ogólnym poziomie niektóre spośród zagadnień związanych z modelowaniem, które dosyć szczegółowo opisaliśmy w naszej prezentacji modeli ER i EER w rozdziale 3. i we wcześniejszych podrozdziałach tego rozdziału. Prezentowana tutaj terminologia jest stosowana zarówno w literaturze poświęconej koncepcyjnemu modelowaniu danych, jak i w materiałach na temat sztucznej inteligencji, w których pojawia się problematyka **reprezentacji wiedzy** (ang. *knowledge representation*, w skrócie *KR*). Omówimy tu podobieństwa i różnice między modelowaniem koncepcyjnym a reprezentacją wiedzy oraz przedstawimy nową terminologię i kilka dodatkowych zagadnień.

Celem technik reprezentowania wiedzy jest opracowywanie struktur umożliwiających precyzyjne modelowanie wybranej **dziedziny wiedzy** przez tworzenie **ontologii**<sup>12</sup>, która opisuje elementy tej dziedziny. Takie struktury są następnie wykorzystywane do składowania i przetwarzania wiedzy w celu wnioskowania, podejmowania decyzji lub po prostu znajdowania odpowiedzi na postawione pytania. Reprezentacja wiedzy jest stosowana w podobny sposób jak semantyczne modele danych, jednak pomiędzy wspomnianymi podejściami istnieje zarówno wiele istotnych podobieństw, jak i różnic:

- W obu podejściach wykorzystuje się proces abstrakcji do identyfikowania wspólnych właściwości i ważnych aspektów obiektów występujących w mini-świecie (nazywanym w reprezentacji wiedzy *dziedziną dyskursu*) przy jednoczesnym ignorowaniu nieistotnych różnic i niezwiązanych z przedmiotem modelowania szczegółów.
- W obu podejściach podczas definiowania danych i tworzenia reprezentacji wiedzy stosuje się takie elementy jak związki, ograniczenia, operacje i języki.
- Reprezentacja wiedzy (KR) jest zasadniczo szerszym pojęciem niż semantyczne modele danych. W schematach KR mogą być reprezentowane różne formy wiedzy, w tym reguły (wykorzystywane w procesach wnioskowania, dedukcji i przeszukiwania), wiedza niekompletna i wiedza domyślna, a także wiedza czasowa i przestrzenna. Niektóre z tych rozwiązań zostały wprowadzone w rozszerzonych modelach baz danych (patrz rozdział 26.).
- Schematy reprezentacji wiedzy obejmują **mechanizmy wnioskowania**, które odpowiadają za dedukowanie dodatkowych faktów na podstawie faktów składowanych w bazie danych. Oznacza to, że o ile większość współczesnych systemów baz danych ogranicza się do generowania odpowiedzi na bezpośrednie

<sup>12</sup> *Ontologia* pod wieloma względami przypomina schemat koncepcyjny, jednak zawiera więcej wiedzy, reguł i wyjątków.



pytania, systemy oparte na wiedzy (systemy wykorzystujące schematy reprezentacji wiedzy) mogą odpowiadać na pytania wymagające **dedukowania** na bazie składowanych danych. Mechanizmy wnioskowania są wprowadzane do najnowszych technologii baz danych (patrz podrozdział 26.5).

- ile większość modeli danych koncentruje się na reprezentowaniu schematów baz danych (lub tzw. meta-wiedzy), schematy reprezentacji danych często łączą te schematy z samymi ich egzemplarzami i w ten sposób zapewniają niezbędną elastyczność w zakresie reprezentowania wyjątków. Takie rozwiązania mogą prowadzić do obniżenia efektywności w czasie implementowania skonstruowanych w ten sposób schematów KR, szczególnie w sytuacji, gdy porównamy je z bazami danych i gdy mamy do czynienia z ogromną ilością składowanych danych (faktów).

W tym podrozdziale omówimy cztery **abstrakcyjne zagadnienia**, które są wykorzystywane w obu prezentowanych tutaj semantycznych modelach danych (w modelu EER i schematach KR): (1) klasyfikacja i tworzenie egzemplarzy, (2) identyfikacja, (3) specjalizacja i generalizacja oraz (4) agregacja i asocjacja. Połączone w pary pojęcia klasyfikacji i tworzenia egzemplarzy oraz generalizacji i specyfikacji stanowią wzajemne odwrotności. Powiązane są też agregacja i asocjacja. Wszystkie wymienione przed chwilą zagadnienia zostaną omówione w odniesieniu do konkretnych reprezentacji stosowanych w modelu EER, aby jak najdokładniej wyjaśnić proces tworzenia abstrakcji danych i lepiej zrozumieć związany z nim proces projektowania schematu koncepcyjnego. Ten podrozdział zakończymy krótkim omówieniem pojęcia *ontologii*, które jest szeroko stosowane w najnowszych badaniach nad technikami reprezentowania wiedzy.

### 4.7.1. Klasyfikacja i tworzenie egzemplarzy

Proces **klasyfikacji** wiąże się z systematycznym przypisywaniem podobnych obiektów (encji) do odpowiednich klas obiektów (lub typów encji). Możemy obecnie opisywać (w bazach danych) lub wnioskować (w schematach reprezentacji wiedzy) klasy, zamiast poszczególnych obiektów. Zbiory obiektów dzielą te same typy atrybutów, związków i ograniczeń, a dzięki klasyfikowaniu obiektów możemy znacznie uprościć proces odkrywania ich właściwości. **Tworzenie egzemplarzy (konkretyzowanie)** jest procesem odwrotnym względem klasyfikacji i wiąże się z generowaniem i dokładnym analizowaniem poszczególnych obiektów jednej klasy. Oznacza to, że egzemplarz obiektu jest powiązany z odpowiednią klasą za pomocą związku **JEST-EGZEMPLARZEM** lub **JEST-SKŁADOWĄ**. Co prawda diagramy EER nie zezwalają na prezentowanie egzemplarzy wprost, diagramy UML można jednak wykorzystywać do reprezentowania pewnej formy tworzenia egzemplarzy przez nanoszenie na nie pojedynczych obiektów. W materiale wprowadzającym język UML do tej pory *nie wspominaliśmy* o takiej możliwości.

Obiekty pojedynczej klasy powinny się zasadniczo charakteryzować podobną strukturą. Niektóre obiekty mogą jednak zawierać właściwości, które w jakimś stopniu różnią się od pozostałych obiektów należących do tej samej klasy — takie **obiekty wyjątków** także muszą być uwzględniane w tworzonym projekcie i okazuje się, że schematy reprezentacji wiedzy nadają się do tego celu lepiej niż modele baz danych. Warto pamiętać, że istnieją także właściwości, które mają zastosowanie dla klasy jako całości, nie zaś dla jej poszczególnych obiektów — reprezentowanie takich **właściwości klas** jest możliwe zarówno na schematach KR, jak i na diagramach UML.

W rozszerzonym modelu ER (EER) encje są klasyfikowane na podstawie typów zgodnie z ich podstawowymi atrybutami i związkami, w których występują. Tak pogrupowane encje mogą być dalej klasyfikowane w podklasy i kategorie na podstawie łączących je podobieństw i dzielących różnic (wyjątków). Podobnie, egzemplarze związków są klasyfikowane na podstawie typów związków. Oznacza to, że typy encji, podklasy, kategorie i typy związków służą w modelu EER do klasyfikowania. Model EER nie daje możliwości jawnego reprezentowania właściwości klas, ale może zostać łatwo rozszerzony w tym kierunku. W języku UML obiekty są klasyfikowane na podstawie klas i istnieje możliwość prezentowania na diagramach zarówno właściwości klas, jak i pojedynczych obiektów.

Modele reprezentacji wiedzy umożliwiają wielokrotne stosowanie schematów klasyfikacji, w których jedna klasa jest *egzemplarzem* innej klasy (nazywanej **meta-klasą**). Warto pamiętać, że takich związków *nie można* bezpośrednio reprezentować w modelu EER, ponieważ mamy tam do czynienia tylko z dwoma poziomami — klas i egzemplarzy. Jedynym dopuszczalnym związkiem pomiędzy klasami w modelu EER jest związek nadklasy-podklasy, natomiast w niektórych schematach reprezentacji wiedzy istnieje możliwość umieszczania dodatkowych związków klasy-egzemplarzy w hierarchii klas. Egzemplarz sam może stanowić inną klasę, umożliwiając tym samym zastosowanie wielopoziomowego schematu klasyfikacji.

## 4.7.2. Identyfikacja

**Identyfikacja** jest procesem abstrakcji, w którym klasy i obiekty są unikatowo identyfikowane w oparciu o jakiś **identyfikator**. Przykładowo, nazwa klasy unikatowo identyfikuje całą klasę. Do unikatowego odróżniania poszczególnych egzemplarzy obiektów niezbędne jest zastosowanie jakiegoś mechanizmu identyfikowania obiektów. Co więcej, konieczne jest unikatowe identyfikowanie wielu składowanych w bazie danych wystąpień tego samego obiektu ze świata rzeczywistego. Przykładowo, nasza baza danych może zawierać następującą trójkę <Jan Kowalski, 75062156124, 865-23-14> w relacji OSOBA i inną trójkę <45875, INF, 3,8> w relacji STUDENT — oba wystąpienia mogą reprezentować ten sam byt (encję) ze świata rzeczywistego. Nie mamy możliwości identyfikowania faktu reprezentowania przez oba wymienione obiekty (trójki) w bazie danych tej samej encji ze świata rzeczywistego, chyba że już *na etapie projektowania* uwzględnimy konieczność zastosowania odpowiedniego mechanizmu identyfikującego. Oznacza to, że proces identyfikacji jest niezbędny na dwóch poziomach:

- rozróżniania obiektów i klas z bazy danych,
- identyfikowania obiektów baz danych oraz ich wiązania z odpowiednimi encjami świata rzeczywistego.

W modelu EER identyfikacja konstrukcji schematu opiera się na systemie unikatowych nazw przypisywanych do tych konstrukcji. Przykładowo, każda klasa w rozszerzonym schemacie związków encji (niezależnie od tego, czy ma postać typu encji, podklasy, kategorii czy typu związku) musi mieć unikatową nazwę. Także nazwy atrybutów na poziomie poszczególnych klas muszą być unikatowe. W praktyce niezbędne są także reguły jednoznacznego identyfikowania odwołań do nazw atrybutów w kratkach i hierarchiach specjalizacji lub generalizacji.

Na poziomie obiektów do odróżniania poszczególnych encji jednego typu wykorzystuje się wartości atrybutów kluczy. W przypadku słabych typów, encje są identyfikowane za pomocą kombinacji wartości ich własnych kluczy częściowych i powiązanych z nimi encji typu lub typów właścicielskich. Egzemplarze związków są identyfikowane w oparciu o rozmaite kombinacje wiązanych za ich pomocą encji (w zależności od określonych współczynników liczności).

### 4.7.3. Specjalizacja i generalizacja

**Specjalizacja** jest procesem klasyfikowania klasy obiektów do postaci większej liczby wyspecjalizowanych podklas. **Generalizacja** jest procesem odwrotnym, w którym wiele klas jest uogólnianych i przenoszonych na wyższy poziom abstrakcji — w ten sposób powstaje klasa abstrakcyjna z wyższego poziomu, obejmująca obiekty należące do wszystkich uogólnionych klas. Specjalizacja jest jednym z działań udoskonalających schemat koncepcyjny, natomiast generalizacja jest koncepcyjną syntezą konstrukcji zastosowanych na schemacie. W modelu EER do reprezentowania specjalizacji i generalizacji wykorzystuje się podklasy. Związek łączący podklasę z jej nadklasą nazywamy związkiem **JEST-PODKLASĄ** lub po prostu związkiem **JEST**. To ten sam związek **JEST**, który opisano wcześniej w punkcie 4.5.3.

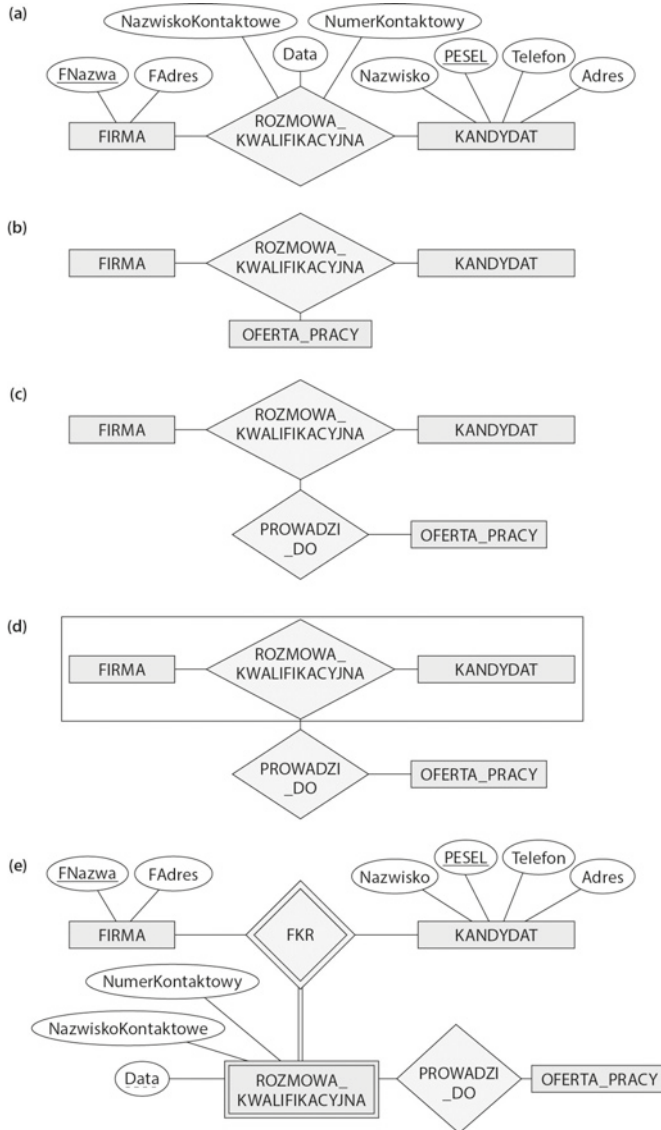
### 4.7.4. Agregacja i asocjacja

**Agregacja** jest abstrakcyjnym pojęciem opisującym budowę obiektów złożonych z ich obiektów składowych. Istnieją trzy przypadki, w których można stosować agregację w rozszerzonym modelu związków encji. Pierwszym z nich jest sytuacja, w której agregujemy wartości atrybutów celem stworzenia kompletnego obiektu. Drugim — sytuacja, kiedy reprezentujemy związek agregacji w postaci zwykłego związku. Trzeci przypadek ma miejsce wtedy, gdy model EER nie zapewnia możliwości jawnego łączenia obiektów powiązanych za pomocą konkretnego egzemplarza związku i ich przekształcania w *zagregowany obiekt na wyższym poziomie abstrakcji*. Takie rozwiązanie jest przydatne w sytuacji, gdy zagregowany obiekt wyższego poziomu sam ma występować w związku z innym obiektem. Związek pomiędzy obiektami prymitywnymi a ich obiektami zagregowanymi nazywamy związkiem **JEST-CZĘŚCIĄ**; odwrotnością tego związku jest związek **SKŁADA-SIĘ-Z**. Język UML przewiduje możliwość stosowania wszystkich trzech typów agregacji.

Abstrakcyjne pojęcie **asocjacji** jest wykorzystywane do łączenia obiektów z wielu *niezależnych* klas. Oznacza to, że proces ten pod wieloma względami przypomina drugie zastosowanie techniki agregacji. Asocjacje są reprezentowane za pomocą typów związków (w modelu EER) lub właściwych agregacji (w języku UML). Te abstrakcyjne powiązania są nazywane związkami **JEST-POWIĄZANY**.

Aby lepiej zrozumieć zastosowania techniki agregacji, przeanalizujmy schemat związków encji przedstawiony na rysunku 4.11(a), który reprezentuje informacje o prowadzonych w różnych firmach rozmowach kwalifikacyjnych podczas rekrutacji pracowników. Klasa `FIRMA` jest agregacją następujących atrybutów (lub obiektów składowych): `NazwaF` (nazwa firmy) oraz `AdresF` (adres firmy). Klasa `KANDYDAT` jest agregacją atrybutów `Pesel`, `Nazwisko`, `Adres` i `Telefon`. Atrybuty `NazwiskoKontaktowe` i `NumerKontaktowy` związku `ROZMOWA_KWALIFIKACYJNA` reprezentują nazwisko i numer telefonu osoby pracującej w danej firmie, która odpowiada za rekrutację. Przypuśćmy, że efektem niektórych rozmów są oferty pracy, rezultatem pozostałych jest brak takich ofert. Chcielibyśmy traktować typ związku

ROZMOWA\_KWALIFIKACYJNA jako klasę, aby połączyć ją z klasą OFERTA\_PRACY. Schemat przedstawiony na rysunku 4.11(b) jest *nieprawidłowy*, ponieważ wymaga od każdego egzemplarza związku reprezentującego rozmowę kwalifikacyjną zawierania oferty pracy. Rozwiązaniem nie może być także schemat przedstawiony na rysunku 4.11(c), ponieważ model ER nie zezwala na stosowanie związków łączących inne związki.



RYSUNEK 4.11. Agregacja. (a) Typ związku ROZMOWA\_KWALIFIKACYJNA. (b) Dołączenie typu encji OFERTA\_PRACY do typu związku trójskładnikowego (niepoprawne). (c) Wymuszenie na związku PROWADZI\_DO udziału w innym związku (zabronione w tradycyjnym modelu ER). (d) Stosowanie agregacji i złożonego (molekularnego) obiektu (zwykle niemożliwe w modelu ER, ale dopuszczalne w niektórych narzędziach do modelowania). (e) Prawidłowa reprezentacja w modelu ER

Jednym ze sposobów reprezentowania tej sytuacji jest stworzenie na wyższym poziomie abstrakcji klasy zagregowanej, która będzie się składała z klas FIRMA, KANDYDAT i ROZMOWA\_KWALIFIKACYJNA i występowała w związku z typem encji OFERTA\_PRACY (patrz rysunek 4.11(d)). Model EER w wersji opisywanej w tej książce co prawda nie udostępnia tego typu możliwości, jednak niektóre semantyczne modele danych zezwalają na stosowanie takich rozwiązań i nazywają obiekty wynikowe **obiektami złożonymi** lub **obiektami molekularnymi**. Jeszcze inne modele jednakowo traktują typy encji i typy związków, i tym samym nie ograniczają możliwości definiowania związków pomiędzy związkami (w takim przypadku prawidłowy jest diagram przedstawiony na rysunku 4.11(c)).

Aby prawidłowo reprezentować opisaną sytuację w modelu ER, musimy stworzyć nowy, słaby typ encji ROZMOWA\_KWALIFIKACYJNA (patrz rysunek 4.11(e)) i powiązać go odpowiednim związkiem z typem encji OFERTA\_PRACY. Oznacza to, że zawsze możemy reprezentować podobne przypadki w tradycyjnym modelu związków encji przez proste tworzenie dodatkowych typów encji, choć z punktu widzenia spójności projektu koncepcyjnego w wielu przypadkach lepszym rozwiązaniem byłoby zastosowanie bezpośredniej reprezentacji w postaci agregacji (jak na rysunku 4.11(d)) lub dopuszczenie związków łączących związki (jak na rysunku 4.11(c)).

Główna różnica strukturalna pomiędzy agregacją a asocjacją polega na tym, że w momencie usuwania egzemplarza asocjacji wszystkie występujące w niej obiekty mogą dalej istnieć. Jeśli jednak dopuścimy w naszym schemacie możliwość stosowania obiektów zagregowanych (przykładowo, obiektu SAMOCHÓD składającego się obiektów SILNIK, PODWOZIE i OPONY), usunięcie każdego takiego obiektu będzie równoznaczne z usunięciem wszystkich jego obiektów składowych (w przypadku obiektu SAMOCHÓD będą to obiekty SILNIK, PODWOZIE i OPONY).

### 4.7.5. Ontologia i sieć semantyczna

W ostatnich latach ogromne ilości skomputeryzowanych danych i informacji dostępnych w internecie pozostają poza jakąkolwiek kontrolą. Obecnie stosuje się wiele różnych modeli i formatów przechowywania danych. Poza wspomnianymi w tej książce modelami baz danych, wiele informacji jest składowanych w formie **dokumentów**, których struktura jest znacznie prostsza niż informacji przechowywanych w bazach danych. Jeden z projektów badawczych będących próbą stworzenia mechanizmu wymiany informacji pomiędzy komputerami połączonymi za pośrednictwem internetu nosi nazwę **sieci semantycznej** (ang. *Semantic Web*). Celem wspomnianego projektu jest stworzenie modeli reprezentacji wiedzy, które będą na tyle ogólne, by umożliwiać łatwą wymianę informacji i wyszukiwanie danych pomiędzy rozproszonymi komputerami. Pojęcie *ontologii* (ang. *ontology*) — które jest ściśle związane z obszarem reprezentacji wiedzy — jest uznawane za najbardziej obiecujący kierunek rozwoju sieci semantycznych na drodze do osiągnięcia tego celu. W tym punkcie przedstawimy krótkie wprowadzenie do tego, czym jest ontologia i jak może być wykorzystywana w roli podstawowego elementu umożliwiającego automatyzację rozumienia, przeszukiwania i wymiany informacji.

Badania nad ontologiami polegają na próbach opisanie struktur i związków, które mogą występować w rzeczywistym świecie w ramach popularnych słowników, zatem mogą być traktowane jako pewien sposób opisywania wiedzy o rzeczywistym świecie z perspektywy pewnego środowiska. Pojęcie ontologii pochodzi z obszaru filozofii i metafizyki.

zyki. W obszarze sieci semantycznych często stosuje się definicję, która mówi, że ontologia jest *specyfikacją konceptualizacji*<sup>13</sup>.

W przedstawionej definicji **konceptualizacja** jest zbiorem pojęć wykorzystywanych do reprezentowania wybranego wycinka rzeczywistości lub wiedzy, który jest przedmiotem zainteresowania środowiska użytkowników. **Specyfikacja** odwołuje się do języka i elementów słownika (słownictwa) wykorzystywanych do określania konceptualizacji. Ontologia obejmuje zarówno *specyfikację*, jak i *konceptualizację*. Przykładowo, ta sama konceptualizacja może być zdefiniowana w dwóch różnych językach, co prowadzi do otrzymania dwóch różnych ontologii. W oparciu o tę dosyć ogólną definicję nie można jednoznacznie stwierdzić, czym dokładnie jest ontologia. Poniżej umieściliśmy listę czterech możliwych technik opisywania ontologii:

- **Tezaurus** (lub nawet **słownik** czy **glosariusz** terminów) opisujący związki pomiędzy słowami, które reprezentują różne pojęcia ze świata rzeczywistego.
- **Systematyka** opisuje sposób, w jaki pojęcia z konkretnego obszaru wiedzy są wzajemnie powiązane — wykorzystuje struktury podobne do tych wykorzystywanych w specjalizacji i generalizacji.
- Szczegółowy **schemat bazy danych** jest traktowany przez wielu specjalistów jak ontologia opisująca pojęcia (encje i atrybuty) oraz związki mini-świata będącego wybranym fragmentem rzeczywistości.
- **Logiczna teoria** wykorzystująca pojęcia z logiki matematycznej do definiowania pojęć i występujących pomiędzy nimi wzajemnych związków.

Środki wykorzystywane do opisywania ontologii są zwykle dosyć podobne do pojęć, które omówiliśmy podczas analizy technik modelowania koncepcyjnego, a więc encji, atrybutów, związków, specjalizacji itp. Główną różnicą pomiędzy ontologią a np. schematem bazy danych jest to, że schematy są najczęściej ograniczone przez możliwość opisywania tylko niewielkiego podzbioru aspektów rzeczywistego mini-świata, który będzie składowany i zarządzany w formie danych. Ontologia jest zazwyczaj uważana za rozwiązanie bardziej ogólne, które może opisywać wybrany fragment rzeczywistości lub dziedzinę zainteresowań (np. pojęcia medyczne, aplikacje z obszaru handlu elektronicznego, sport itd.) tak dokładnie, jak to tylko możliwe.

## 4.8. Podsumowanie

W tym rozdziale w pierwszej kolejności omówiliśmy popularne rozszerzenia modelu ER, które zwiększają możliwości reprezentacyjne tego modelu. Rozbudowany w ten sposób model danych nazwaliśmy rozszerzonym modelem ER lub modelem EER. W rozdziale zaprezentowano pojęcia podklasy i jej nadklasy oraz związany z nimi mechanizm dziedziczenia atrybutów i związków. Przekonaliśmy się także o konieczności tworzenia w niektórych sytuacjach dodatkowych klas encji — albo z powodu pojawienia się dodatkowych, specyficznych atrybutów, albo z uwagi na specyficzne typy związków. W rozdziale omówiliśmy dwa główne procesy definiowania hierarchii i krat nadklas-podklas: proces specjalizacji i odwrotny względem niego proces generalizacji.

---

<sup>13</sup> Przedstawiona definicja pochodzi z książki Grubera (1995).



Następnie przedstawiliśmy sposób, w jaki nowe konstrukcje można prezentować w postaci graficznej na diagramach EER. Omówiliśmy także różne typy ograniczeń, które można stosować dla procesów specjalizacji i generalizacji. Dwa główne rodzaje tych ograniczeń (wymagań) to specjalizacja-generalizacja kompletna lub częściowa oraz specjalizacja-generalizacja rozłączna lub nierozłączna (pokrywająca się). Przeanalizowaliśmy pojęcia kategorii i typu unii (który reprezentuje podzbiór unii dwóch lub więcej klas) oraz przedstawiliśmy formalne definicje wszystkich prezentowanych pojęć.

W kolejnych podrozdziałach wprowadziliśmy niektóre elementy notacji i terminologii stosowanej w języku UML do reprezentowania specjalizacji i generalizacji. W podrozdziale 4.7 krótko omówiliśmy obszar badań nazywany reprezentacją wiedzy oraz jego związki z semantycznym modelowaniem danych. Przeanalizowaliśmy i podsumowaliśmy takie typy abstrakcyjnych reprezentacji danych jak klasyfikacja i tworzenie egzemplarzy, identyfikacja, specjalizacja i generalizacja oraz agregacja i asocjacja. Pokazaliśmy związki tych mechanizmów z odpowiednimi elementami modelu EER i języka UML.

## Pytania powtórkowe

- 4.1. Co to jest podklasa? Kiedy zastosowanie podklasy jest niezbędne na etapie modelowania danych?
- 4.2. Zdefiniuj następujące pojęcia: *nadklasa podklasy*, *związek nadklasy-podklasy*, *związek "jest"*, *specjalizacja*, *generalizacja*, *kategoria*, *specyficzne (lokalne) atrybuty* oraz *specyficzne związki*.
- 4.3. Omów mechanizm dziedziczenia atrybutów i związków. Dlaczego ten mechanizm jest przydatny?
- 4.4. Omów podklasy definiowane przez użytkownika i podklasy definiowane przez predykaty; wymień różnice pomiędzy tymi typami podklas.
- 4.5. Omów specjalizacje definiowane przez użytkownika i specjalizacje definiowane przez predykaty; wymień różnice pomiędzy tymi rodzajami specjalizacji.
- 4.6. Przedstaw dwa główne typy ograniczeń nakładanych na specjalizacje i generalizacje.
- 4.7. Jaka jest różnica pomiędzy hierarchią specjalizacji a kratą specjalizacji?
- 4.8. Jaka jest różnica pomiędzy specjalizacją a generalizacją? Dlaczego oba procesy są reprezentowane na diagramach schematu w identyczny sposób?
- 4.9. Czym kategoria różni się od zwykłej podklasy dzielonej przez wiele nadklas? Do czego wykorzystuje się kategorie? Swoją odpowiedź spróbuj zilustrować odpowiednimi przykładami.
- 4.10. Dla każdego z wymienionych poniżej terminów stosowanych w języku UML (patrz podrozdziały 3.8 i 4.6) przedstaw i krótko omów odpowiedni element w modelu EER (jeśli taki element w ogóle istnieje): *obiekt*, *klasa*, *asocjacja*, *agregacja*, *generalizacja*, *liczność*, *atrybuty*, *dyskryminator*, *powiązanie*, *atrybut powiązania*, *asocjacja zwrotna*, *asocjacja kwalifikowana*.
- 4.11. Omów główne różnice pomiędzy notacją diagramów schematu EER a notacją diagramów klas UML, porównując metody reprezentowania poszczególnych elementów w obu notacjach.



- 4.12. Wymień różne elementy abstrakcji danych wraz z odpowiednimi konstrukcjami wykorzystywanymi do ich modelowania w modelu EER.
- 4.13. Której z cech agregacji brakuje w modelu EER? Jak można ten model dalej rozbudować, aby także ta cecha była w nim obsługiwana?
- 4.14. Wymień najważniejsze podobieństwa i różnice pomiędzy technikami koncepcyjnego modelowania baz danych a technikami opartymi na reprezentacji wiedzy.
- 4.15. Omów podobieństwa i różnice pomiędzy ontologią a schematem bazy danych.

## Ćwiczenia

- 4.16. Zaprojektuj schemat EER dla aplikacji bazy danych, którą z jakichś powodów jesteś zainteresowany. Określ wszystkie ograniczenia i wymagania, które taka baza danych powinna spełniać. Upewnij się, że zaprojektowany przez Ciebie schemat zawiera co najmniej pięć typów encji, cztery typy związków, słaby typ encji, związek nadklasy-podklasy, kategorię oraz typ związku  $n$ -składnikowego (dla  $n > 2$ ).
- 4.17. Przeanalizuj przedstawiony na rysunku 3.21 schemat ER bazy danych BANK i przyjmij, że niezbędne jest utrzymywanie informacji o różnych typach kont (KONTO\_OSZCZĘDNOŚCIOWE, KONTO\_CZEKOWE, ...) oraz kredytów (KREDYT\_SAMOCHODOWY, KREDYT\_MIESZKANIOWY, ...). Przyjmij także, że dla każdego konta w bazie danych powinny być rejestrowane transakcje (wpłaty, wypłaty, realizacje czeków, ...) oraz operacje spłaty kredytów — oba typy encji powinny zawierać informacje o kwocie, dacie i dokładnym czasie reprezentowanego zdarzenia. Zmodyfikuj schemat bazy danych BANK wykorzystując dostępne w modelach ER i EER mechanizmy specjalizacji i generalizacji. Opisz wszystkie dodatkowe założenia dotyczące nowych wymagań.
- 4.18. Poniższy scenariusz opisuje uproszczoną wersję organizacji infrastruktury sportowej, która ma być wykorzystana podczas letniej olimpiady. Narysuj diagram EER, który będzie zawierał wszystkie niezbędne typy encji, atrybuty, związki i specjalizacje dla odpowiedniej aplikacji bazy danych. Wymień wszystkie założenia, które przyjąłeś. Infrastruktura olimpijska jest podzielona na tzw. kompleksy sportowe. Kompleksy sportowe dzielą się na typy *jedno-sportowe* i *wielo-sportowe*. Kompleksy wielo-sportowe obejmują obszary przydzielone do poszczególnych dyscyplin wraz z wyszczególnionym położeniem (środek, północno-wschodni narożnik itp.). Każdy kompleks ma swoje położenie, osobę odpowiedzialną za organizowanie zawodów, łączny wykaz zajmowanego obszaru itp. Dla każdego kompleksu jest także przechowywana lista kolejnych wydarzeń sportowych (dla biegni mogą to być kolejne wyścigi). Dla każdego takiego zdarzenia istnieje planowana data, czas trwania, liczba uczestników, liczba oficjalnych gości itp. Lista wszystkich oficjalnych gości będzie przechowywana łącznie z listą zdarzeń, w których ci goście będą uczestniczyli. Do przygotowania i obsługi poszczególnych zdarzeń wymagane jest oczywiście rozmaite wyposażenie (bramki, tyczki, poprzeczki). Wspomniane dwa typy kompleksów (jedno-sportowe i wielo-sportowe) będą reprezentowane za pomocą różnych rodzajów informacji. Przykładowo, dla każdego z tych typów musi być przechowywana niezbędna liczba obiektów sportowych i obiektów zaplecza wraz z odpowiednim budżetem.

- 4.19. Wymień wszystkie ważne byty reprezentowane w bibliotecznej bazie danych opisanej w poniższym scenariuszu. Przedstaw zwłaszcza niezbędne abstrakcje klasyfikacji (typy encji i związków), agregacji, identyfikacji oraz specjalizacji-generalizacji. Wszędzie tam, gdzie jest to możliwe, zdefiniuj ograniczenia liczności w postaci par wartości (*min*, *maks*). Wymień wszystkie zauważone szczegóły, które Twoim zdaniem wpłyną na ostateczny kształt projektu, ale które nie zostały uwzględnione w projekcie koncepcyjnym. Stwórz osobną listę ograniczeń semantycznych. Narysuj diagram EER dla opisanej bazy danych biblioteki.

**Scenariusz:** Biblioteka Techniczna Miasta Gliwice (w skrócie BTMG) ma około 16 tysięcy zarejestrowanych czytelników, 100 tysięcy tytułów oraz 250 tysięcy tomów (a więc średnio 2,5 kopii każdego z tytułów). W dowolnym momencie około 10 procent książek jest wypożyczonych. Pracownicy biblioteki za każdym razem sprawdzają, czy książki, które dany członek biblioteki chce pożyczyć, są w danej chwili dostępne. Bibliotekarze muszą także wiedzieć, ile kopii każdej z książek znajduje się w bibliotece, a ile jest w danym momencie wypożyczonych. Katalog książek jest dostępny w internecie — można go przeszukiwać i sortować według autorów, tytułów oraz tematyki. Dla każdego tytułu w bibliotece istnieje w katalogu opis, który może się składać z jednego zdania, ale może także zajmować kilka stron. Bibliotekarza muszą mieć stały dostęp do tych opisów, aby odpowiadać na ewentualne pytania zainteresowanych czytelników. Pracownicy biblioteki są podzieleni na głównego bibliotekarza, adiunktów bibliotecznych, pracowników informacji naukowej, pracowników wydających i przyjmujących książki oraz asystentów.

Książki mogą być wypożyczane na 21 dni. Każdy z członków biblioteki może mieć jednocześnie wypożyczonych maksymalnie pięć książek. Członkowie biblioteki zwykle zwracają książki w ciągu trzech do czterech tygodni. Większość z nich wie, że zanim otrzymają stosowny monit (wzywający do zwrócenia zaległych książek) pracownicy biblioteki odczekają tydzień po terminie. Takie upomnienia są wysyłane do około 5 procent członków biblioteki. Większość zaległych książek jest zwracana w ciągu miesiąca od normalnego terminu. Około 5 procent z tych książek jest zwracana po upływie miesiąca lub nie jest zwracana wcale. Za najbardziej aktywnych czytelników uważa się tych członków biblioteki, którzy wypożyczają co najmniej dziesięć książek w ciągu roku. 1 procent najbardziej aktywnych członków biblioteki generuje około 15 procent wszystkich wypożyczeń, natomiast 10 procent najbardziej aktywnych członków biblioteki generuje około 40 procent wypożyczeń. Około 20 procent członków biblioteki jest zupełnie nieaktywnych w tym sensie, że od momentu zapisania się do biblioteki nie wypożyczyli żadnej książki.

Aby zostać użytkownikiem biblioteki, należy wypełnić odpowiedni formularz, w którym znajdują się takie dane jak PESEL, adres domowy i adres zamieszkania oraz numer telefonu. Bibliotekarze przygotowują następnie dla kandydata magnetyczną kartę członkowską ze zdjęciem. Każda taka karta jest ważna przez cztery lata. Miesiąc przed wygaśnięciem ważności karty wysyłana jest odpowiednia informacja z propozycją odnowienia. Członkostwo pracowników naukowych instytutu jest automatyczne. Kiedy instytut zatrudnia nowego pracownika, jego informacje są automatycznie przesyłane z odpowiedniego rekordu w pracowniczey bazie danych — na tej podstawie bibliotekarze przygotowują i wysyłają pocztą kartę członkowską. Pracownicy naukowci mają prawo wypożyczania książek na trzy miesiące i korzystają z dwutygodniowego okresu ochronnego (w przypadku przekroczenia tego terminu). Propozycje odnowienia członkostwa są wysyłane do pracowników naukowych na adres odpowiedniej jednostki organizacyjnej wewnątrz instytutu.

Niektóre materiały (encyklopedie, rzadkie tytuły oraz mapy) nie są wypożyczane przez bibliotekę. Bibliotekarze muszą odróżniać książki przeznaczone do wypożyczania od tych, których wynoszenie poza teren biblioteki jest zabronione. Dodatkowo, pracownicy biblioteki dysponują listą książek, które znajdują się w polu ich zainteresowań, ale z jakichś przyczyn w chwili obecnej są niemożliwe do zdobycia — mogą to być rzadkie tytuły lub materiały, których wydania nie były wznawiane, a także książki, które zostały zagubione lub zniszczone i których nie udało się zastąpić nowymi egzemplarzami. Bibliotekarze muszą dysponować systemem, który będzie rejestrował zarówno te książki, które nie mogą być wypożyczane, jak i te, których pozyskanie w danym momencie nie jest możliwe. Niektóre książki mogą mieć ten sam tytuł — oznacza to, że sam tytuł nie może być wykorzystywany w roli unikatowego identyfikatora. Każda książka jest identyfikowana w oparciu o Międzynarodowy System Numeracji Książek (ang. *International Standard Book Number* — *ISBN*), który gwarantuje przypisywanie unikatowych kodów książkom wydawanym na całym świecie. Dwie książki opatrzone takim samym tytułem mogą mieć różne numery ISBN, jeśli zostały wydane w różnych językach lub w innej oprawie (w twardej lub miękkiej okładce). Różne numery ISBN mają także kolejne wydania tej samej książki.

Zaproponowany system bazy danych musi przechowywać informacje o użytkownikach biblioteki, rejestr książek, katalog oraz dane o aktywności czytelników.

4.20. Zaprojektuj bazę danych, która będzie zawierała informacje wykorzystywane w muzeum sztuki. Przyjmij, że zebrano następujące wymagania:

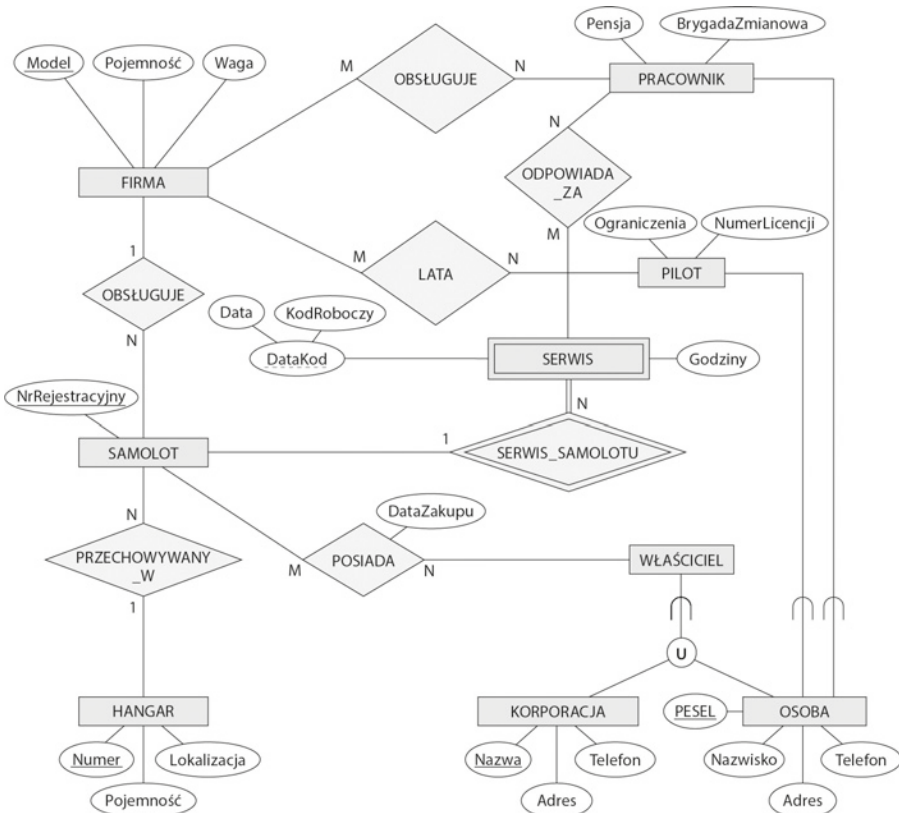
- Muzeum oferuje kolekcję dzieł sztuki (encji typu `DZIEŁO_SZTUKI`). Każde dzieło sztuki (encja typu `DZIEŁO_SZTUKI`) ma przypisany unikatowy numer (atribut `NumerIdentyfikacyjny`), nazwisko artysty (atribut `Artysta`, jeśli jest znany), rok powstania (atribut `Rok`, jeśli jest znany), tytuł (atribut `Tytuł`) oraz opis (atribut `Opis`). Dzieła sztuki są podzielone na różne kategorie według różnych kryteriów (patrz poniższe podpunkty).
- Egzemplarze klasy `DZIEŁO_SZTUKI` są podzielone na kategorie według ich typu. Istnieją trzy główne typy: `OBRAZ`, `RZEŹBA` i `POSĄG`, a także pozostałe typy należące do kategorii `INNE`, która obejmuje dzieła nie mieszczące się w żadnym z trzech głównych typów.
- Typ encji `OBRAZ` obejmuje następujące atrybuty: `TypObrazu` (olejny, akwarela itp.), `NamalowanyNa` (papier, płótno, drewno itp.) oraz `Styl` (modernistyczny, abstrakcyjny itp.).
- Typy encji `RZEŹBA` i `POSĄG` zawierają następujące atrybuty: `Materiał` (drewno, kamień itp.), `Wysokość`, `Waga` i `Styl`.
- Każde dzieło sztuki należące do kategorii `INNE` jest opisywane przez atrybuty `Typ` (rycina, fotografia itp.) oraz `Styl`.
- Encje typu `DZIEŁO_SZTUKI` są podzielone na należące do muzeum (typ `STAŁA_EKSPOZYCJA`) i wypożyczone (typ `WYPOŻYCZONE`). Informacje o obiektach będących własnością muzeum obejmują atrybuty `DataPozyskania`, `Status` (na wystawie, udostępnione innej jednostce lub w magazynie) i `Wartość`. Dane na temat wypożyczonych obiektów to atrybuty `Kolekcja` (z której dzieło pochodzi), `DataWypożyczenia` i `DataZwrotu`.

- Dla każdej encji typu `DZIEŁO_SZTUKI` dodatkowo utrzymywane są informacje opisujące kraj (kulturę) i epokę pochodzenia — odpowiednio atrybuty `Pochodzenie` (Włochy, Egipt, Ameryka, Indie itp.) oraz `Epoka` (Renesans, Czasy współczesne, Starożytność itp.).
- Muzeum utrzymuje także następujące informacje o znanych autorach dzieł sztuki (typ encji `ARTYSTA`): `Nazwisko`, `DataUrodzenia` (jeśli znana), `DataŚmierci` (jeśli autor nie żyje), `KrajPochodzenia`, `Epoka`, `GłównyStyl` oraz `Opis`. Przyjmuje się, że `Nazwisko` jest atrybutem unikatowym.
- Muzeum organizuje różne wystawy (reprezentowane przez typ encji `WYSTAWA`), które są opisywane za pomocą takich atrybutów jak `Nazwa`, `DataOtwarcia` i `DataZamknięcia`. Egzemplarze typu encji `WYSTAWA` są powiązane ze wszystkimi dziełami sztuki, które były prezentowane podczas danej wystawy.
- W bazie danych powinny się znajdować informacje o zewnętrznych zbiorach (typ encji `KOLEKCJA`) wykorzystywanych podczas wystaw organizowanych w muzeum — każda taka kolekcja jest opisywana przez takie atrybuty jak unikatowa `Nazwa`, `Typ` (muzealna, prywatna itp.), `Opis`, `Adres`, `Telefon` oraz aktualna `OsobaKontaktowa`.

Narysuj diagram schematu EER dla tej aplikacji. Omów wszystkie założenia przyjęte w trakcie opracowywania tego diagramu i wyjaśnij decyzje, które podjąłeś.

- 4.21. Na rysunku 4.12 przedstawiono przykład diagramu EER dla bazy danych niewielkiego, prywatnego lotniska, która jest wykorzystywana do przechowywania informacji o samolotach, ich właścicielach, pracownikach lotniska oraz pilotach. Na podstawie wymagań zdefiniowanych dla tej bazy danych zgromadzono następujące informacje: Każdy `SAMOLOT` ma przypisany unikatowy numer rejestracyjny (atrybut `NrRejestracyjny`), konkretny typ samolotu (związek `JEST_TYPU`) oraz znajduje się w określony hangarze (związek `PRZECHOWYWANY_W`). Każdy `TYP_SAMOLOTU` ma unikatowy numer modelu (atrybut `Model`), ładowność (atrybut `Ładowność`) oraz wagę (atrybut `Waga`). Każdy `HANGAR` ma unikatowy numer (atrybut `Numer`), pojemność (atrybut `Pojemność`) oraz lokalizację (atrybut `Lokalizacja`). Baza danych zawiera także informacje o właścicielach poszczególnych samolotów (typ encji `WŁAŚCICIEL` występujący w związku `POSIADA`) oraz pracownikach (typ encji `PRACOWNICY`), którzy te samoloty obsługują (związek `OBSŁUGUJE`). Każdy egzemplarz typu związku `POSIADA` wiąże samolot z jego właścicielem i zawiera atrybut reprezentujący datę zakupu (`DataZakupu`). Każdy egzemplarz typu związku `ODPOWIADA_ZA` wiąże pracownika lotniska z rekordem typu encji `SERWIS`. Każdy samolot jest wielokrotnie serwisowany, co oznacza, że jest związany za pomocą typu związku `SERWIS_SAMOLOTU` z wieloma rekordami reprezentującymi przeglądy i naprawy. Każdy taki rekord zawiera atrybuty odpowiadające dacie wykonania (`Data`), liczbie poświęconych godzin (`Godziny`) oraz typowi wykonanej pracy (`KodRoboczy`). Do reprezentowania przeglądów i napraw serwisowych używamy typu encji `SERWIS`, ponieważ do identyfikowania rekordów tego typu jest wykorzystywany numer rejestracyjny samolotu. Właścicielem samolotu może być albo osoba prywatna, albo korporacja. W związku z tym stosujemy typ unii (kategorię) `WŁAŚCICIEL`, który jest podzbiorem unii typów encji reprezentujących korporacje (`KORPORACJA`) i osoby (`OSOBA`).

Zarówno piloci (egzemplarze klasy PILOT), jak i pracownicy (egzemplarze klasy PRACOWNIK) są podklasami klasy OSOBA. Każdy pilot ma przypisane specyficzne (lokalne) atrybuty: NumerLicencji i Ograniczenia; natomiast każdy pracownik ma przypisane specyficzne atrybuty Pensja i BrygadaZmianowa. Wszystkie przechowywane w bazie danych encje typu OSOBA mają przypisane takie atrybuty jak numer PESEL (atrybut Pesel), nazwisko (Nazwisko), adres (Adres) oraz numer telefonu (Telefon). W przypadku encji typu KORPORACJA przechowywane dane obejmują nazwę (atrybut Nazwa), adres (Adres) i numer telefonu (Telefon). Baza danych zawiera także informacje o typach samolotów, które mogą być pilotowane przez poszczególnych pilotów (typ związku LATA), oraz typach samolotów, które mogą być serwisowane przez poszczególnych pracowników (typ encji OBSŁUGUJE). Pokaż, jak przedstawiony na rysunku 4.12 schemat EER bazy danych NIEWIELKIE\_LOTNISKO może być reprezentowany w notacji UML (*Uwaga: nie omawialiśmy w tym rozdziale technik reprezentowania kategorii (typów unii) w notacji UML, zatem w tym i następnym ćwiczeniu nie musisz odwzorowywać kategorii*).



RYСУNEK 4.12. Schemat EER dla bazy danych NIEWIELKIE\_LOTNISKO

- 4.22. Pokaż, jak przedstawiony na rysunku 4.9 schemat EER bazy danych UNIwersytet może być reprezentowany w notacji UML.

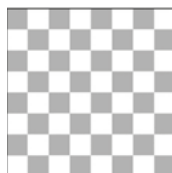
4.23. Przyjrzyj się zbiorom encji i atrybutom z pokazanej poniżej tabeli. W każdym wierszu umieść w jednej kolumnie haczyk, aby określić związek między elementami z lewej i prawej kolumny.

- Występuje związek między elementem po lewej stronie a elementem przedstawionym po prawej.
- Element po prawej stronie jest atrybutem elementu przedstawionego po lewej.
- Element po lewej stronie jest specjalizacją elementu przedstawionego po prawej.
- Element po lewej stronie jest generalizacją elementu przedstawionego po prawej.

Zbiór encji	(a) Jest w związku z	(b) Ma atrybut będący	(c) Jest specjalizacją	(d) Jest generalizacją	Zbiór encji lub atrybut
1. MATKA					OSOBA
2. CÓRKA					MATKA
3. STUDENT					OSOBA
4. STUDENT					IdStudenta
5. SZKOŁA					STUDENT
6. SZKOŁA					SALA
7. ZWIERZĘ					KOŃ
8. KOŃ					Rasa
9. KOŃ					Wiek
10. PRACOWNIK					PESEL
11. MEBEL					KRZESŁO
12. KRZESŁO					Waga
13. CZŁOWIEK					KOBIETA
14. ŻOŁNIERZ					OSOBA
15. WROGI_					OSOBA
↪ ŻOŁNIERZ					

4.24. Narysuj diagram UML na potrzeby przechowywania w bazie stanu gry w szachy. Aplikację podobną do projektowanej znajdziesz na stronie <http://www.chessgames.com>. Opisz wszystkie założenia poczynione w trakcie tworzenia diagramu UML. Oto przykładowe założenia:

- Gra w szachy odbywa się między dwoma graczami.
- Gra toczy się na planszy o wymiarach 8×8 podobnej do tej poniżej:

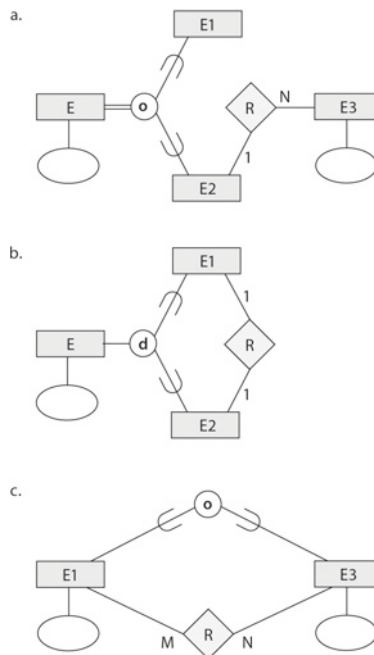


- Graczom na początku gry przypisywany jest kolor biały lub czarny.

4. Każdy gracz początkowo dysponuje następującymi bierkami (nazywanymi tradycyjnie szachami lub figurami):
  - a) królem,
  - b) hetmanem,
  - c) dwiema wieżami,
  - d) dwoma gońcami,
  - e) dwoma skoczkami,
  - f) ośmioma pionami.
5. Każda bierka ma początkową pozycję.
6. Każda bierka ma zestaw dozwolonych ruchów, zależny od stanu gry. Nie musisz przejmować się tym, które ruchy są dozwolone. Wyjątkiem są następujące kwestie:
  - a) Bierka może zająć puste pole lub zbić bierkę przeciwnika.
  - b) Jeśli bierka zostaje zbita, należy ją usunąć z planszy.
  - c) Gdy pion dojdzie do ostatniego rzędu, zostaje „promowany”. Polega to na zastąpieniu go inną bierką (hetmanem, wieżą, gońcem lub skoczkiem).

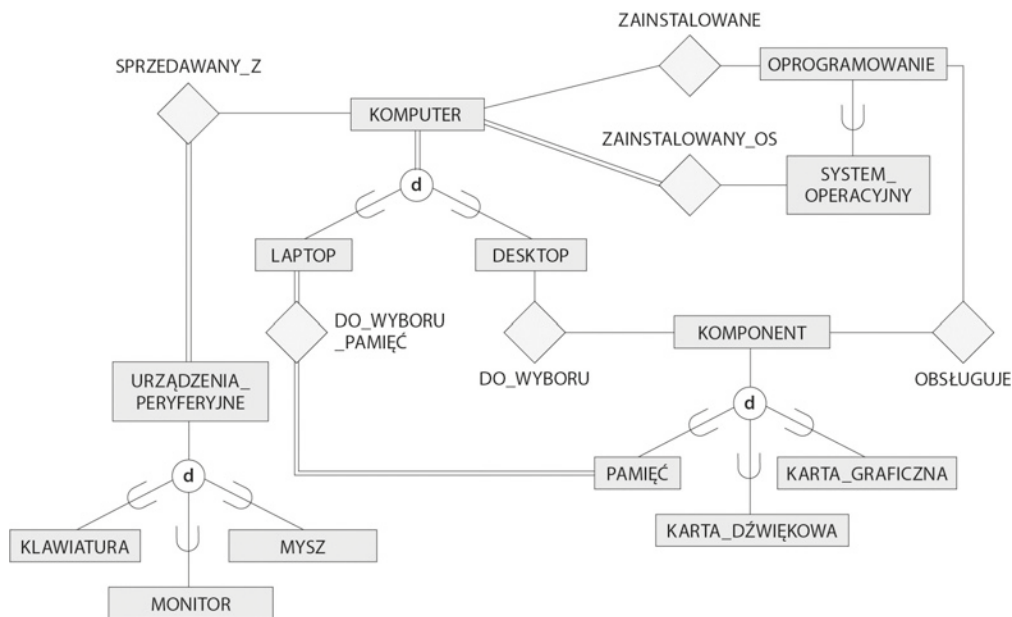
*Uwaga:* niektóre funkcje mogą być realizowane za pomocą kilku klas.

- 4.25. Narysuj diagram EER na potrzeby gry w szachy zgodnie z opisem z ćwiczenia 4.24. Skoncentruj się na aspektach trwałego składowania danych. Przykładowo, system powinien umożliwiać pobranie po kolei wszystkich ruchów z danej gry.
- 4.26. Które z poniższych diagramów EER są nieprawidłowe i dlaczego? Przedstaw wszelkie poczynione założenia.





- 4.27. Przyjrzyj się poniższemu diagramowi EER opisującemu system informatyczny firmy. Podaj atrybuty i klucze dla wszystkich typów encji. Określ ograniczenia liczności maksymalnej i uzasadnij je. Przedstaw kompletny opis tego, co reprezentuje ten diagram EER.



## Ćwiczenia laboratoryjne

- 4.28. Wyobraź sobie bazę **DZIENNIK\_OCEN**, w której wykładowca z instytutu uniwersytetu zapisuje punkty zdobyte przez poszczególnych studentów na zajęciach. Oto wymogi dotyczące danych:

- Każdy student jest identyfikowany za pomocą unikatowego identyfikatora, imienia i nazwiska oraz adresu e-mail.
- Każdy wykładowca w poszczególnych semestrach prowadzi określone przedmioty. Każdy przedmiot jest identyfikowany za pomocą numeru przedmiotu, numeru działu i semestru. Dla każdego przedmiotu wykładowca określa minimalną liczbę punktów potrzebnych do uzyskania ocen 5, 4, 3 i 2. Przykładowo, 90 punktów daje ocenę 5, 80 punktów to 4, 70 punktów to 3.
- Studenci są zapisywani na przedmioty prowadzone przez wykładowcę.
- Każdy przedmiot ma zestaw zadań częściowych (egzamin semestralny, egzamin końcowy, projekt itd.). Dla każdego zadania częściowego określone są maksymalna liczba punktów (np. 100 lub 50) i waga (np. 20% lub 100%). Wagi wszystkich zadań z przedmiotu zwykle sumują się do 100%.

- Wykładowca zapisuje punkty otrzymane przez studenta za każde zadanie cząstkowe z każdego przedmiotu. Na przykład student 1234 otrzymał 84 punkty za egzamin semestralny z drugiego działu kursu INF2310 w semestrze jesiennym 2009 r. To zadanie pozwalało uzyskać 100 punktów i miało wagę 20% (określa ona wpływ zadania na ocenę z kursu).

Zaprojektuj diagram EER bazy na potrzeby dziennika ocen. Utwórz projekt tej bazy za pomocą narzędzia do modelowania danych (np. ERwin lub Rational Rose).

4.29. Wyobraź sobie system bazy danych AUKCJE\_INTERNETOWE, dotyczące użytkowników serwisu (kupujących i sprzedających) uczestniczących w transakcjach z udziałem produktów. Oto wymagania związane z danymi z tego systemu:

- Witryna internetowa ma użytkowników, z których każdy jest identyfikowany za pomocą unikatowego numeru użytkownika i opisywany przy użyciu adresu e-mail, nazwiska, hasła, adresu zamieszkania i numeru telefonu.
- Użytkownik może być kupującym lub sprzedającym. Dla kupujących w bazie zapisany jest adres dostawy. Dla sprzedającego zapisywane są numer konta bankowego i numer transakcji.
- Produkty są wystawiane przez sprzedającego na sprzedaż i identyfikowane za pomocą przypisywanego przez system unikatowego numeru produktu. Produkty są też opisywane przy użyciu tytułu, opisu, ceny wyjściowej, kroku cenowego, daty rozpoczęcia aukcji i daty zakończenia aukcji.
- Produkty są ponadto dzielone na kategorie na podstawie stałej hierarchii (np. modelem może należeć do kategorii KOMPUTERY → SPRZĘT → MODEMY).
- Kupujący oferują cenę za interesujące ich produkty. Rejestrowane są proponowana cena i czas zgłoszenia oferty. Osoba, która na koniec aukcji zaproponowała najwyższą cenę, jest uznawana za zwycięzcę, po czym można zrealizować transakcję między kupującym a sprzedającym.
- Kupujący i sprzedający mogą publikować opinie dotyczące zakończonych transakcji. Opinie obejmują ocenę drugiej strony uczestniczącej w transakcji (od 1 do 10) i komentarz.

Zaprojektuj diagram EER bazy AUKCJE\_INTERNETOWE i utwórz jej projekt za pomocą narzędzia do modelowania danych (np. ERwin lub Rational Rose).

4.30. Wyobraź sobie system bazy danych używany na potrzeby rozgrywek baseballa (np. pierwszej ligi). Oto wymagania dotyczące danych:

- Pracownicy związani z rozgrywkami to gracze, trenerzy, menedżerowie i sędziowie. Każda z tych osób jest identyfikowana za pomocą unikatowego identyfikatora. Do opisu pracowników służą imię i nazwisko, a także data i miejsce urodzenia.
- Do opisu graczy służą też inne atrybuty, takie jak kierunek wybijania (prawa, lewa, na zmianę) i średnia uderzeń z całej kariery.
- Wśród graczy znajduje się podzbiór miotaczy. Z miotaczami powiązany jest wskaźnik ERA (ang. *earned run average*) z całej kariery.

- Zespoły są w unikatowy sposób identyfikowane za pomocą nazw. Do opisu zespołów używana jest też miejscowość, w jakiej drużyna jest zlokalizowana, i liga, w której dany zespół gra (np. dywizja centralna ligi amerykańskiej).
- Drużyny mają jednego menedżera, kilku trenerów oraz grupę zawodników.
- Gry toczą się między dwoma zespołami, z których jeden jest gospodarzem, a drugi gościem danego dnia. Dla każdego zespołu zapisywane są wyniki (punkty, wybiecia i błędy). Zespół z największą liczbą punktów wygrywa grę.
- Po każdej zakończonej grze rejestrowani są miotacze, którym zapisano wygraną lub przegraną. Jeśli przyznano save, rejestrowany jest też miotacz, któremu zapisano tę statystykę.
- Po zakończonej grze dla każdego gracza rejestrowana jest też liczba wybić (typu single, double, triple i home run).

Zaprojektuj diagram EER bazy danych BASEBALL i utwórz jej projekt za pomocą narzędzia do modelowania danych (np. ERwin lub Rational Rose).

- 4.31. Przyjrzyj się diagramowi EER bazy UNIWERSYTET z rysunku 4.9. Utwórz jej projekt za pomocą narzędzia do modelowania danych (np. ERwin lub Rational Rose). Utwórz listę różnic między notacją z diagramu z tekstu a analogiczną notacją z używanego narzędzia.
- 4.32. Przyjrzyj się diagramowi bazy NIEWIELKIE\_LOTNISKO z rysunku 4.12. Utwórz jej projekt za pomocą narzędzia do modelowania danych (np. ERwin lub Rational Rose). Zwróć uwagę na model kategorii WŁAŚCICIEL na tworzonym diagramie.  
*Wskazówka:* rozważ zastosowanie dwóch różnych typów związków: WŁAŚCICIEL\_TO\_KORPORACJA i WŁAŚCICIEL\_TO\_OSoba.
- 4.33. Przyjrzyj się bazie UNIWERSYTET opisanej w ćwiczeniu 3.16. W ćwiczeniu laboratoryjnym 3.31 zbudowałeś już schemat ER tej bazy, używając narzędzia do modelowania danych (np. ERwin lub Rational Rose). Zmodyfikuj utworzony diagram, dzieląc typ encji PRZEDMIOT na kategorie PRZEDMIOTY\_I\_IV i PRZEDMIOTY\_MAGISTRANCI oraz typ encji WYKŁADOWCA na kategorie MŁODSZY\_WYKŁADOWCA i STARSZY\_WYKŁADOWCA. Dodaj odpowiednie atrybuty dla nowych typów encji. Następnie określ związek, zgodnie z którym młodsi wykładowcy mogą prowadzić kursy dla studentów młodszych lat, a starsi wykładowcy uczyć na kursach dla magistrantów.

## Wybrane publikacje

W rozmaitych publikacjach zaproponowano wiele koncepcyjnych i semantycznych modeli danych. W tym miejscu przedstawiamy możliwie reprezentatywną listę takich tekstów. Można je podzielić na dwie grupy. Pierwsza z nich, która obejmuje książkę Abrial'a (1974), model DIAM Senki (1975), metodę NIAM Vergeijena i VanBekkuma (1982) oraz publikację Bracchiego i in. (1976), prezentuje modele semantyczne bazujące na koncepcji związków binarnych. Drugą grupę publikacji stanowią wczesne książki omawiające metody rozszerzania modelu relacyjnego mające na celu rozbudowę jego możliwości w obszarze modelowania danych — do tej grupy można zaliczyć książkę Schmida i Swensona (1975), Navathe'a i Schkolnicka (1978), zaproponowany przez Codda model RM/T (1979), książkę Furtado (1978) oraz zaproponowany przez Wiederholda i Elmasriego model strukturalny (1979).

Tradycyjny model ER został zaproponowany przez Chena (1976) i sformalizowany w Ng (1981). Od tamtego czasu przedstawiono mnóstwo propozycji rozszerzeń możliwości modelowania względem oryginalnej koncepcji: Scheuermann i in. (1979), Dos Santos i in. (1979), Teorey i in. (1986), Gogolla i Hohenstein (1991) oraz opracowany przez Elmasri'ego i in. model związków, kategorii i encji ECR (1985). Smith i Smith (1977) zaprezentowali mechanizmy generalizacji i agregacji. Zaproponowany przez Hammera i McLeoda (1981) semantyczny model danych wprowadzał nie tylko pojęcie klasy-podklasy, ale także inne zaawansowane elementy modelowania danych.

Elementy analizy zagadnień związanych z semantycznym modelowaniem danych można znaleźć w pracy Hulla i Kinga (1987). Eick (1991) omówił techniki projektowania i transformowania schematów koncepcyjnych. Soutou (1998) przedstawił analizę ograniczeń nakładanych na związki  $n$ -składnikowe. Język UML został szczegółowo opisany w książce Boocha, Rumbaugh'a i Jacobsona (1999). Krótkie wprowadzenie zagadnień związanych z tym językiem można znaleźć także w książkach Fowlera i Scotta (2000) oraz Stevensa i Pooley'a (2000).

Prace Fensela (2000, 2003) dotyczą sieci semantycznych i zastosowań ontologii. Uschold i Gruninger (1996) oraz Gruber (1995) omówili ontologie. Wydanie miesięcznika *Communications of the ACM* z czerwca 2002 roku zostało poświęcone praktycznym zastosowaniom ontologii. Fensel (2003) opisał ontologie i handel elektroniczny.





---

# Relacyjny model danych i SQL





## Relacyjny model danych i ograniczenia relacyjnych baz danych

Ten rozdział otwiera trzecią część książki, poświęconą relacyjnym bazom danych. Relacyjny model danych został po raz pierwszy wprowadzony w roku 1970 w klasycznym tekście pracującego wówczas w dziale badań firmy IBM Teda Codd (Codd, 1970). Zaprezentowany model natychmiast przykuł uwagę ze względu na swą prostotę i solidne podstawy matematyczne. Model ten opiera się na pojęciu *matematycznej relacji*, która przypomina tabelę wartości i pełni rolę podstawowego elementu w tym podejściu. Model relacyjny bazuje także na teorii zbiorów, logice pierwszego rzędu i rachunku predykatów. W tym rozdziale omówimy podstawowe własności tego modelu i związane z nim ograniczenia.

Pierwsze komercyjne implementacje modelu relacyjnego pojawiły się na rynku we wczesnych latach 80-tych. Były to np. system zarządzania bazą danych firmy Oracle oraz systemu SQL/DS dla systemu operacyjnego MVS firmy IBM. Od tamtego czasu model relacyjny został zaimplementowany w ogromnej liczbie różnych systemów komercyjnych, a także w wielu systemach o otwartym dostępie do kodu źródłowego. Do najbardziej popularnych relacyjnych systemów zarządzania bazami danych należą obecnie DB2 (firmy IBM), Oracle (firmy Oracle), Sybase (obecnie firmy SAP) oraz SQL Server i Access (firmy Microsoft). Ponadto dostępnych jest kilka systemów o otwartym dostępie do kodu źródłowego, takich jak MySQL i PostgreSQL.

Z uwagi na znaczenie modelu relacyjnego, poświęciliśmy mu i związanym z nim językom całą trzecią część tej książki. W rozdziałach 6. i 7. omówimy wybrane aspekty języka SQL, który jest kompletnym modelem i językiem będącym *standardem* w komercyjnych relacyjnych SZBD. Pozostałe elementy tego języka opiszemy w innych rozdziałach. W rozdziale 8. przedstawimy operacje algebry relacyjnej i wprowadzimy notację rachunku relacyjnego. Są to dwa formalne języki powiązane z modelem relacyjnym. Rachunek relacyjny jest uznawany za podstawę języka SQL, a algebra relacyjna jest używana w wielu implementacjach baz na potrzeby przetwarzania i optymalizowania zapytań (patrz część 8. tej książki).

Inne mechanizmy z modelu relacyjnego są zaprezentowane w dalszych częściach książki. W rozdziale 9. zestawimy struktury modelu relacyjnego z odpowiadającymi im konstrukcjami modeli ER i EER, a także zaprezentujemy algorytmy projektowania schematów relacyjnych baz danych przez odwzorowywanie schematu koncepcyjnego w modelu ER lub EER (patrz rozdziały 3. i 4.) do postaci odpowiedniej reprezentacji relacyjnej. Algorytmy tego typu są obecnie implementowane w wielu systemach wspomagających projek-

townie baz danych i narzędziach typu CASE<sup>1</sup>. Należące do czwartej części tej książki rozdziały 10. i 11. zawierają omówienie technik programistycznych używanych do uzyskania dostępu do baz relacyjnych za pomocą standardowych protokołów ODBC i JDBC. W rozdziale 11. wprowadzamy programowanie internetowych baz danych. W rozdziałach 14. i 15. z części 6. prezentujemy jeszcze inne aspekty relacyjnego modelu danych, w tym ograniczenia formalne dla zależności funkcyjnych i wielowartościowych (takie zależności są wykorzystywane w badaniach nad teorią projektowania relacyjnych baz danych, które bazują na pojęciu nazywanym *normalizacją*).

W tym rozdziale skoncentrujemy się na opisywaniu podstawowych reguł relacyjnego modelu danych. Rozpocznemy od zdefiniowania najważniejszych pojęć i elementów notacji tego modelu (podrozdział 5.1). W podrozdziale 5.2 skupimy się na omówieniu ograniczeń relacyjnych, które są obecnie uważane za istotną część relacyjnego modelu danych i które w większości relacyjnych systemów zarządzania bazami są automatycznie wymuszane. W podrozdziale 5.3 zdefiniujemy operacje aktualizacji z modelu relacyjnego i omówimy mechanizmy obsługi ewentualnych naruszeń więzów integralności. Wprowadzamy tu też pojęcie transakcji. Podrozdział 5.4 stanowi podsumowanie tego rozdziału.

Niniejszy rozdział wraz z rozdziałem 8. dotyczą przede wszystkim formalnych podstaw modelu relacyjnego. Rozdziały 6. i 7. koncentrują się na praktycznym modelu relacyjnym z języka SQL, który stanowi podstawę większości komercyjnych i otwartych relacyjnych SZBD. Modele formalne i praktyczne w dużym stopniu pokrywają się ze sobą, występują jednak między nimi pewne różnice, na które należy zwrócić uwagę.

## 5.1. Pojęcia z modelu relacyjnego

Model relacyjny reprezentuje bazę danych w postaci zbioru *relacji*. Mówiąc nieformalnie, każda relacja przypomina nie tylko tabelę wartości, a także (w pewnym sensie) *plaski* plik rekordów. Nazwa **plaski plik** pochodzi od tego, że każdy rekord ma prostą strukturę liniową (*plaską*). Przykładowo, przedstawiona na rysunku 1.2 baza danych składająca się z kilku plików jest bardzo podobna do relacyjnej reprezentacji danych. Istnieją jednak istotne różnice pomiędzy relacjami a plikami (omówimy je poniżej).

Jeśli przyjmiemy, że relacja ma postać **tabeli** wartości, każdy wiersz w tej tabeli będzie reprezentował zbiór powiązanych ze sobą wartości danych. W modelu relacyjnym każdy wiersz tabeli reprezentuje fakt, który odpowiada zwykle bytowi (encji) lub związkowi występującemu w świecie rzeczywistym. Nazwa tabeli i nazwy kolumn mają na celu ułatwienie interpretacji znaczeń wartości zawartych w poszczególnych kolumnach. Przykładowo, pierwsza tabela na rysunku 1.2 została nazwana STUDENT, ponieważ każdy wiersz reprezentuje fakty dotyczące encji konkretnego studenta. Nazwy kolumn (Nazwisko, NumerIndeksu, Rok i Kierunek) określają sposób, w jaki należy interpretować wartości danych w poszczególnych wierszach — zasadnicze znaczenie ma oczywiście to, w której kolumnie znajdują się interesujące nas wartości. Wszystkie wartości w jednej kolumnie muszą należeć do tego samego typu danych.

---

<sup>1</sup> Skrót CASE pochodzi od ang. *Computer-Aided Software Engineering*, czyli wspomaganej komputerowo inżynierii oprogramowania.

W formalnej terminologii modelu relacyjnego wiersz jest nazywany *krotką*, nagłówek (nazwa) kolumny jest nazywana *atrybutem*, natomiast tabela nosi nazwę *relacji*. Typ danych opisujący rodzaje wartości, które mogą być przechowywane w każdej z kolumn, jest reprezentowany przez *dziedzinę* możliwych wartości. W poniższym punkcie dokładnie zdefiniujemy wspomniane pojęcia: *dziedziny*, *krotki*, *atrybutu* oraz *relacji*.

### 5.1.1. Dziedziny, atrybuty, krotki i relacje

**Dziedzina**  $D$  jest zbiorem wartości atomowych. Przez określenie wartości **atomowe** rozumiemy, że każda wartość w danej dziedzinie jest niepodzielna (przynajmniej w danym modelu relacyjnym). Powszechną metodą określania dziedziny jest wyznaczanie typu danych, z którego muszą pochodzić wszystkie wartości tworzące daną dziedzinę. Warto pamiętać o nadawaniu nazw definiowanym dziedzinom, ponieważ można w ten sposób bardzo ułatwić interpretowanie ich wartości. Poniżej przedstawiono kilka przykładowych dziedzin wartości:

- **Numery\_telefonów\_w\_Polsce**: Zbiór dziewięciocyfrowych (dwie cyfry dla numeru kierunkowego i siedem cyfr numeru lokalnego) prawidłowych numerów telefonów na terenie Polski.
- **Numery\_PESEL**: Zbiór poprawnych, jedenastocyfrowych numerów PESEL.
- **Nazwiska**: Zbiór ciągów znakowych reprezentujących nazwiska.
- **Średnie\_ocen**: Możliwe wartości reprezentujące obliczone średnie ocen; każda z takich wartości musi być liczbą rzeczywistą (zmiennoprzecinkową) z przedziału od 2 do 5.
- **Wiek\_pracowników**: Możliwe wartości reprezentujące wiek poszczególnych pracowników firmy; każda z takich wartości musi być liczbą całkowitą z przedziału od 15 do 80.
- **Nazwy\_wydziałów\_akademickich**: Zbiór nazw wydziałów akademickich danego uniwersytetu — np. Wydział Informatyki, Wydział Ekonomii czy Wydział Fizyki.
- **Kody\_wydziałów\_akademickich**: Zbiór kodów wydziałów akademickich — np. INF, EKON lub FIZ.

Wymienione powyżej przykłady są w takiej postaci nazywane *logicznymi* definicjami dziedzin. Dla każdej dziedziny należy dodatkowo określić **typ** lub **format danych**. Przykładowo, typem danych dla dziedziny **Numery\_telefonów\_w\_Polsce** może być ciąg znaków w postaci  $(ddd)ddd-dd-dd$ , gdzie każda litera  $d$  reprezentuje cyfrę dziesiętną, a pierwsze dwie cyfry tworzą numer kierunkowy. Typem danych dla dziedziny **Wiek\_pracowników** powinien być zbiór liczb całkowitych z przedziału od 15 do 80. W przypadku dziedziny **Nazwy\_wydziałów\_akademickich** typem danych jest zbiór wszystkich ciągów znaków reprezentujących prawidłowe nazwy wydziałów. Jak widać, dziedzina jest opisywana za pomocą nazwy, typu danych i formatu. Można stosować także dodatkowe informacje ułatwiające interpretowanie wartości z danej dziedziny; przykładowo, dziedzina liczbowa (np. **Waga\_osoby**) powinna być dodatkowo opisana stosowaną jednostką (mogą to być np. *kilogramy* lub *funt*y).

**Schemat relacji**  $R$ , zapisywany w postaci  $R(A_1, A_2, \dots, A_n)$ , składa się z nazwy relacji  $R$  oraz listy jej atrybutów  $A_1, A_2, \dots, A_n$ . Każdy **atrybut**  $A_i$  jest nazwą roli odgrywanej przez określoną dziedzinę  $D$  w schemacie relacji  $R$ .  $D$  jest nazywane **dziedziną** atrybutu  $A_i$  i zapisywane w postaci **dom( $A_i$ )** (od ang. *domain*). Schemat relacji jest wykorzystywany do opisywania relacji;  $R$  jest określane mianem **nazwy** takiej relacji. **Stopień** relacji (ang. *degree* lub *arity*) jest liczbą atrybutów jej schematu, czyli  $n$ .

Poniżej przedstawiono przykład schematu relacji siódmego stopnia, która opisuje studentów uniwersytetu i obejmuje siedem atrybutów:

STUDENT(Nazwisko, PESEL, TelefonDomowy, Adres, TelefonDoBiura, Wiek, ŚrOcen)

Gdybyśmy chcieli określić typ danych dla każdego z siedmiu atrybutów, zapisalibyśmy schemat relacji STUDENT w następujący sposób:

STUDENT(Nazwisko: string, PESEL: string, TelefonDomowy: string, Adres: string,  
→ TelefonDoBiura: string, Wiek: integer, ŚrOcen: real)

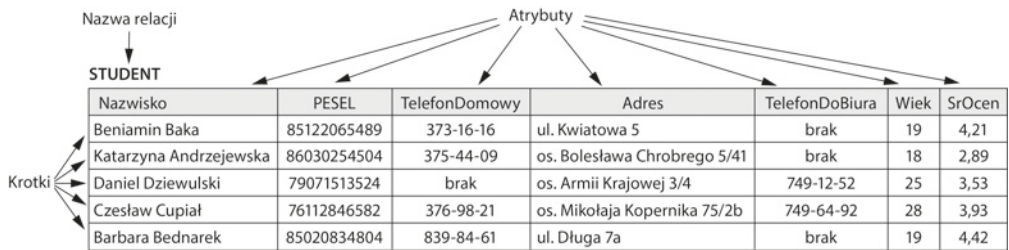
STUDENT jest nazwą relacji w zaprezentowanym powyżej schemacie relacji — relacja STUDENT zawiera siedem atrybutów. W powyższej definicji przedstawiliśmy jeden ze sposobów przypisywania atrybutom relacji uniwersalnych typów (takich jak integer dla liczb całkowitych czy string dla ciągów znaków). Mówiąc dokładniej, dla niektórych atrybutów relacji STUDENT możemy określić następujące (wcześniej zdefiniowane) dziedziny:  $\text{dom}(\text{Nazwisko}) = \text{Nazwiska}$ ;  $\text{dom}(\text{PESEL}) = \text{Numery\_PESEL}$ ;  $\text{dom}(\text{TelefonDomowy}) = \text{Numery\_telefonów\_lokalnych}^2$ ;  $\text{dom}(\text{TelefonDoBiura}) = \text{Numery\_telefonów\_lokalnych}$  oraz  $\text{dom}(\text{ŚrOcen}) = \text{Średnie\_ocen}$ . Do atrybutów w schemacie relacji możemy się odwoływać także w oparciu o ich pozycje wewnątrz relacji (a więc niekoniecznie w oparciu o ich nazwy); oznacza to, że drugim atrybutem relacji STUDENT jest PESEL, natomiast czwartym atrybutem tej samej relacji jest Adres.

**Relacja** (lub **stan relacji**)<sup>3</sup>  $r$  (zapisywane także w postaci  $r(R)$ ) schematu relacji  $R(A_1, A_2, \dots, A_n)$  jest zbiorem  $n$ -krotek, zatem  $r = \{t_1, t_2, \dots, t_m\}$ . Każda  **$n$ -krotka**  $t$  jest uporządkowaną listą  $n$  wartości:  $t = \langle v_1, v_2, \dots, v_n \rangle$ , gdzie każda wartość  $v_i$  ( $1 \leq i \leq n$ ) jest elementem dziedziny  $\text{dom}(A_i)$  lub jest specjalną wartością pustą (takie wartości są opisane poniżej oraz w punkcie 5.1.2). Do  $i$ -tej wartości w krotce  $t$  (która odpowiada atrybutowi  $A_i$ ) odwołujemy się za pomocą wyrażenia  $t[A_i]$  albo  $t.A_i$  (lub  $t[i]$ , jeśli używamy notacji pozycyjnej). W literaturze można niekiedy spotkać określenia **intensji relacji** dla schematu  $R$  oraz **ekstensji relacji** dla stanu  $r(R)$ .

Na rysunku 5.1 przedstawiono przykład relacji STUDENT, który odpowiada zaprezentowanemu przed chwilą schematowi relacji STUDENT. Każda krotka w tej relacji reprezentuje encję konkretnego studenta. Przedstawiliśmy tę relację w postaci tabeli, w której każda krotka ma postać *wiersza*, a każdemu atrybutowi odpowiada *nagłówek kolumny* reprezentujący rolę i zalecaną interpretację zawartych w danej kolumnie wartości. *Wartości puste*

<sup>2</sup> Przy stale zwiększającej się liczbie numerów telefonów wynikającej choćby z rosnącej popularności telefonii komórkowej, niektórzy operatorzy mogą z czasem rozszerzać istniejące systemy numeracji. W takim przypadku zamiast dziedziny *Numery\_telefonów\_lokalnych* powinniśmy użyć dziedziny *Numery\_telefonów\_w\_Polsce*.

<sup>3</sup> Stan relacji często jest także nazywany **egzemplarzem relacji**. W tej książce nie będziemy się posługiwali tym określeniem, ponieważ słowo *egzemplarz* jest wykorzystywane także w odniesieniu do pojedynczej krotki lub wiersza.



RYSUNEK 5.1. Atrybuty i krotki relacji STUDENT

(na rysunku oznaczone słowem *brak*) reprezentują atrybuty, których wartości są nieznane lub nie istnieją dla konkretnej krotki relacji STUDENT.

Możemy teraz przedstawić *zmodyfikowaną*, bardziej formalną wersję wcześniejszej definicji relacji, wykorzystując teorię zbiorów. Relacją (lub stanem relacji)  $r(R)$  nazywamy taką **relację matematyczną** stopnia  $n$  zdefiniowaną na dziedzinach  $\text{dom}(A_1)$ ,  $\text{dom}(A_2)$ , ...,  $\text{dom}(A_n)$ , która jest **podzbiorem iloczynu kartezjańskiego** dziedzin definiujących  $R$ :

$$r(R) \subseteq (\text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n))$$

Iloczyn kartezjański wyznacza wszystkie możliwe kombinacje wartości z wymienionych dziedzin. Oznacza to, że jeśli przez  $|D|$  oznaczymy łączną liczbę wartości, inaczej **liczność** dziedziny  $D$  (przy założeniu, że wszystkie dziedziny są skończone), łączna liczba krotek w tym iloczynie kartezjańskim wyniesie:

$$|\text{dom}(A_1)| \times |\text{dom}(A_2)| \times \dots \times |\text{dom}(A_n)|$$

Ten iloczyn licznosci wszystkich dziedzin określa łączną liczbę możliwych egzemplarzy lub krotek, które mogą istnieć w stanie relacji  $r(R)$ . Stan relacji w określonym momencie (tzw. **bieżący stan relacji**) odzwierciedla tylko jeden poprawny zbiór krotek (ze wszystkich możliwych kombinacji), który reprezentuje konkretny stan świata rzeczywistego. Zasadniczo wraz ze zmianami występującymi w świecie rzeczywistym zmianom powinna ulegać także odpowiednia relacja — powinna przechodzić z jednego stanu w inny stan relacji. Schemat  $R$  jest jednak względnie statyczny i nie ulega zmianom z wyjątkiem bardzo rzadkich zdarzeń — przykładowo, w efekcie dodania atrybutu reprezentującego nowe informacje, które początkowo nie były przechowywane w tej relacji.

Istnieje możliwość zdefiniowania wielu atrybutów, których *wartości należą do tej samej dziedziny*. Takie atrybuty reprezentują jednak inne **role** (interpretacje) tej samej dziedziny. Przykładowo, w relacji STUDENT ta sama dziedzina Numery\_telefonów\_lokalnych występuje zarówno w roli *telefonu domowego studenta* (atrybut TelefonDomowy), jak i w roli *telefonu do biura studenta* (atrybut TelefonDoBiura). Trzecim możliwym atrybutem z tej samej dziedziny może być Telefon\_komórkowy.

## 5.1.2. Właściwości relacji

Z przedstawioną w poprzednim punkcie definicją relacji wiąże się kilka właściwości, które odróżniają relacje od plików i tabel. W tym punkcie omówimy niektóre z tych własności.

**Porządkowanie krotek w relacji.** Relacja została zdefiniowana jako *zbiór* krotek. Z matematycznego punktu widzenia, elementy zbioru *nie* są uporządkowane, co oznacza, że także krotki w relacji nie powinny mieć określonej kolejności. W relacjach kolejność krotek nie ma więc znaczenia. Należy jednak pamiętać, że rekordy w pliku muszą być fizycznie przechowywane na dysku (lub w innej pamięci), zatem musi istnieć między nimi jakiś porządek. Taki porządek określa pierwszy, drugi, *i*-ty i ostatni rekord w pliku. Podobnie, podczas prezentowania relacji w postaci tabeli musimy uporządkować jej wiersze według określonej kolejności.

Kolejność (porządek) krotek nie jest częścią definicji relacji, ponieważ relacja ma reprezentować fakty na poziomie logicznym lub abstrakcyjnym. Dla każdej relacji można określić wiele różnych porządków logicznych. Przykładowo, krotki w przedstawionej na rysunku 5.1 relacji STUDENT można logicznie uporządkować według wartości atrybutu Nazwisko, PESEL, Wiek lub dowolnego innego atrybutu. Ponieważ definicja relacji nie określa kolejności krotek, żaden z logicznych porządków *nie jest uprzywilejowany* względem pozostałych. Oznacza to, że relacja przedstawiona na rysunku 5.2 jest uważana za *identyczną* z relacją przedstawioną na rysunku 5.1. Kiedy dana relacja jest implementowana w postaci pliku lub prezentowana w formie tabeli, można oczywiście tymczasowo wyznaczyć porządek rekordów pliku lub wierszy tabeli.

Nazwisko	PESEL	TelefonDomowy	Adres	TelefonDoBiura	Wiek	SrOcen
Daniel Dziwulski	79071513524	brak	os. Armii Krajowej 3/4	749-12-52	25	3,53
Barbara Bednarek	85020834804	839-84-61	ul. Długa 7a	brak	19	4,42
Czesław Cupiał	76112846582	376-98-21	os. Mikołaja Kopernika 75/2b	749-64-92	28	3,93
Katarzyna Andrzejewska	86030254504	375-44-09	os. Bolesława Chrobrego 5/41	brak	18	2,89
Beniamin Baka	85122065489	373-16-16	ul. Kwiatowa 5	brak	19	4,21

RYСУNEK 5.2. Relacja STUDENT z rysunku 5.1 z inną kolejnością krotek

**Porządkowanie wartości wewnątrz krotek i alternatywna definicja relacji.** Zgodnie z przedstawioną wcześniej definicją relacji, *n*-krotka jest *uporządkowaną* listą *n* wartości, zatem porządek wartości w krotce (a więc kolejność atrybutów w schemacie relacji) jest istotna. Na poziomie logicznym kolejność atrybutów i ich wartości *nie* jest jednak ważna, przynajmniej w sytuacji, gdy utrzymywana jest zgodność pomiędzy atrybutami i ich wartościami.

Możemy teraz podać **alternatywną definicję** relacji, która *wyeliminuje* konieczność określania porządku wartości w poszczególnych krotkach. W nowej definicji schemat relacji  $R = \{A_1, A_2, \dots, A_n\}$  jest *zbiorem* atrybutów, a stan relacji  $r(R)$  jest skończonym zbiorem odwzorowań  $r = \{t_1, t_2, \dots, t_m\}$ , gdzie każda krotka  $t_i$  jest **odwzorowaniem**  $R$  w  $D$ , a  $D$  jest **sumą** dziedzin atrybutów (oznaczoną przez  $\cup$ ); oznacza to, że  $D = \text{dom}(A_1) \cup \text{dom}(A_2) \cup \dots \cup \text{dom}(A_n)$ . W naszej nowej definicji  $t[A_i]$  musi należeć do  $\text{dom}(A_i)$  dla  $1 \leq i \leq n$  oraz dla każdego odwzorowania  $t$  w zbiorze  $r$ . Każde odwzorowanie  $t_i$  jest nazywane krotką.

Zgodnie z powyższą definicją krotki (będące odwzorowaniem), każdą **krotkę** możemy traktować jak **zbiór** par  $\langle \text{< atrybut >, < wartość >} \rangle$ , z których każda daje wartość odwzorowania z atrybutu  $A_i$  w wartość  $v_i$  z dziedziny  $\text{dom}(A_i)$ . Porządek atrybutów *nie jest* w takim przypadku istotny, ponieważ wraz z *wartościami* każdego atrybutu występuje jego *nazwa*. Zgodnie z tą definicją, dwie krotki przedstawione na rysunku 5.3 są identyczne. Takie rozwiązanie jest uzasadnione na poziomie abstrakcyjnym, ponieważ w rzeczywistości nie



ma żadnego powodu, aby umieszczać wartość jednego atrybutu przed wartością innego atrybutu. Gdy w krotce przechowywane są zarówno nazwa, jak i wartość atrybutu, mamy do czynienia z **danymi samoopisowymi**, ponieważ krotka obejmuje opis każdej wartości (nazwę atrybutu).

```
t = < (Name, Dick Davidson),(SSN, 422-11-2320),(Home_phone, NULL),(Address, 3452 Elgin Road),
      (Office_phone, (817)749-1253),(Age, 25),(GPA, 3.53)>
```

```
t = < (Address, 3452 Elgin Road),(Name, Dick Davidson),(SSN, 422-11-2320),(Age, 25),
      (Office_phone, (817)749-1253),(GPA, 3.53),(Home_phone, NULL)>
```

RYSUNEK 5.3. Dwie identyczne krotki w sytuacji, gdy porządek atrybutów i wartości nie jest częścią definicji relacji

Generalnie będziemy stosowali **pierwszą definicję** relacji, w której atrybuty ze schematu relacji i wartości wewnątrz krotek są *uporządkowane*, ponieważ takie podejście zasadniczo upraszcza notację. Warto jednak pamiętać, że przedstawiona przed chwilą definicja alternatywna jest bardziej ogólna<sup>4</sup>.

**Wartości i wartości puste w krotkach.** Każda wartość w krotce jest **atomowa**, czyli niepodzielna na składniki w podstawowej strukturze modelu relacyjnego. Oznacza to, że w modelu relacyjnym nie mogą być stosowane atrybuty złożone i atrybuty wielowartościowe (patrz rozdział 3.). Model ten jest niekiedy nazywany **płaskim modelem relacyjnym**. Duża część teorii, na której opiera się model relacyjny, została stworzona właśnie przy założeniu atomowości wartości atrybutów, które jest nazywane założeniem **pierwszej postaci normalnej**<sup>5</sup>. Atrybuty wielowartościowe muszą więc być reprezentowane przez osobne relacje, natomiast atrybuty złożone muszą być reprezentowane za pomocą prostych (atomowych) atrybutów składowych<sup>6</sup>.

Ważnym elementem są specjalne **wartości puste** (ang. *null values*), które wykorzystuje się do reprezentowania nieznanego wartości atrybutów lub wartości, które dla danej krotki nie mają zastosowania. Przykładowo, na rysunku 5.1 niektóre krotki reprezentujące studentów zawierają wartości puste w atrybutach przechowujących numery telefonów do biura, ponieważ niektórzy studenci nie posiadają biur (oznacza to, że telefon do biura *nie ma zastosowania* w przypadku tych studentów). Inni studenci mają wartości puste w atrybutach reprezentujących numer telefonu domowego, prawdopodobnie dlatego, że albo nie mają takich telefonów, albo mają, ale ich numer nie jest znany (wartość jest *nieznana*). Generalnie, wartości puste mogą mieć wiele znaczeń: „wartość nieznana”, „wartość istnieje, ale nie jest dostępna” lub „atrybut nie ma zastosowania dla tej krotki” („wartość niezdefiniowana”). Przykładem ostatniego z wymienionych znaczeń wartości pustej mógłby być atrybut StatusWizowy w relacji STUDENT, który miałby zastosowanie wyłącznie w przypadku krotek reprezentujących studiujących obcokrajowców.

<sup>4</sup> Jak się przekonamy w rozdziałach 15. i 16., alternatywna definicja relacji jest przydatna podczas omawiania zagadnień związanych z przetwarzaniem zapytań.

<sup>5</sup> Omówimy to założenie bardziej szczegółowo w rozdziale 10.

<sup>6</sup> Niektóre rozszerzenia relacyjnego modelu danych eliminują te ograniczenia. Przykładowo, systemy obiektowo-relacyjne zezwalają na stosowanie atrybutów o złożonej strukturze; to samo dotyczy modeli relacyjnych **baz pierwszej postaci normalnej** i **zagnieżdżonych** modeli relacyjnych (patrz rozdział 22.).



Istnieje możliwość opracowania różnych kodów dla różnych znaczeń wartości pustych. Wykorzystywanie różnych typów wartości pustych w operacjach modelu relacyjnego jest trudne i wykracza poza zakres tematyczny tej książki.

Precyzyjne znaczenie wartości pustych wpływa na to, jak należy je traktować w agregacjach arytmetycznych lub w porównaniach z innymi wartościami. Przykładowo, porównanie dwóch wartości pustych jest niejednoznaczne. Jeśli klienci A i B mają puste adresy, *nie oznacza to*, że mają ten sam adres. W trakcie projektowania baz danych najlepiej jest w miarę możliwości unikać wartości pustych. Zagadnienie to omawiamy w rozdziałach 7. i 8. (w kontekście operacji i zapytań) oraz w rozdziale 14. (w kontekście projektowania i normalizacji bazy danych).

**Interpretacja (znaczenie) relacji.** Schemat relacji może być interpretowany jak deklaracja lub typ **twierdzenia**. Przykładowo, schemat relacji STUDENT z rysunku 5.1 zapewnia, że zasadniczo encja reprezentująca studenta ma Nazwisko, PESEL, TelefonDomowy, Adres, Telefon ↗ DoBiura, Wiek i ŚrOcen. Każda krotka tej relacji może być interpretowana jak **fakt** lub konkretny egzemplarz tego twierdzenia. Przykładowo, pierwsza krotka z rysunku 5.1 potwierdza fakt istnienia studenta nazywającego się Benjamin Baka, z numerem PESEL 85122065489, w wieku 19 lat itp.

Warto pamiętać, że niektóre relacje mogą reprezentować fakty o istniejących *bytach* (*encjach*), inne zaś — fakty o istniejących *związkach* między bytami. Przykładowo, schemat relacji STUDIUJE (StudentPESEL, KodWydziału) reprezentuje fakt studiowania danego studenta na określonym wydziale akademickim. Każda krotka w tej relacji wiąże studenta z jego wydziałem. Oznacza to, że model relacyjny umożliwia *jednakowe* (w postaci relacji) reprezentowanie zarówno faktów istnienia bytów, jak i faktów występowania związków. Takie rozwiązanie w niektórych sytuacjach może prowadzić do nieporozumień, ponieważ osoby analizujące schemat relacji muszą się domyślać, czy dana relacja reprezentuje typ encji, czy typ relacji. W rozdziale 3. szczegółowo przedstawiliśmy model ER, w którym dokładnie opisane są encje i związki. W rozdziale 9. omówimy procedury odwzorowywania, za pomocą których można przekształcać różne konstrukcje koncepcyjnych modeli ER i EER (patrz część druga) w odpowiednie relacje.

Alternatywnym sposobem interpretacji schematu relacji jest traktowanie go jako **predykatu**; w takim przypadku wartości poszczególnych krotek są uznawane za elementy *spełniające* dany predykat. Przykładowo, predykat STUDENT (Nazwisko, PESEL, ...) jest prawdziwy dla pięciu krotek z relacji STUDENT z rysunku 5.1. Te krotki reprezentują pięć różnych faktów ze świata rzeczywistego. Taka interpretacja okazuje się przydatna w kontekście logicznych języków programowania (takich jak Prolog), ponieważ umożliwia wykorzystywanie modelu relacyjnego w tych językach (patrz podrozdział 26.5). Zgodnie z **założeniem o zamkniętym świecie** jedynymi faktami na temat świata są te przedstawione za pomocą ekstensji (stanu) relacji. Wszystkie inne kombinacje wartości sprawiają, że predykat jest fałszywy. Ta interpretacja będzie przydatna w trakcie omawiania zapytań dotyczących relacji z wykorzystaniem rachunku relacji (patrz podrozdział 8.6).

### 5.1.3. Notacja modelu relacyjnego

W naszej prezentacji będziemy się posługiwać następującymi elementami notacji:

- Schemat relacji  $R$  stopnia  $n$  zapisujemy jako  $R(A_1, A_2, \dots, A_n)$ .
- Litery  $Q, R$  i  $S$  są nazwami relacji.
- Litery  $q, r$  i  $s$  reprezentują stany relacji.
- Litery  $t, u, v$  reprezentują pojedyncze krotki.
- Generalnie nazwa schematu relacji (np. STUDENT) *jednocześnie wskazuje* na bieżący zbiór krotek w tej relacji, a więc *aktualny stan relacji*; natomiast zapis w postaci STUDENT(Nazwisko, PESEL, ...) reprezentuje *wyłącznie* schemat relacji.
- Atrybut  $A$  może być kwalifikowany za pomocą nazwy relacji  $R$ , do której należy, przez zastosowanie popularnej notacji kropki:  $R.A$  (np. STUDENT.Nazwisko lub STUDENT.Wiek). Konieczność stosowania tej notacji może wynikać z istnienia w dwóch różnych relacjach dwóch atrybutów z taką samą nazwą. Należy jednak pamiętać, że wszystkie nazwy atrybutów *wewnątrz jednej relacji* muszą być unikatowe.
- $n$ -krotkę  $t$  w relacji  $r(R)$  zapisujemy jako  $t = \langle v_1, v_2, \dots, v_n \rangle$ , gdzie  $v_i$  jest wartością odpowiadającą atrybutowi  $A_i$ . Poniższa notacja dotyczy **wartości składowych** krotek:
  - Zarówno  $t[A_i]$ , jak i  $t.A_i$  (czasami także  $t[i]$ ) są odwołaniami do wartości  $v_i$  atrybutu  $A_i$  krotki  $t$ .
  - Zarówno  $t[A_u, A_w, \dots, A_z]$ , jak i  $t.(A_u, A_w, \dots, A_z)$ , gdzie  $A_u, A_w, \dots, A_z$  jest listą atrybutów relacji  $R$ , są odwołaniami do podkrotki z wartościami  $\langle v_u, v_w, \dots, v_z \rangle$  z krotki  $t$ , które odpowiadają atrybutom określonym na liście  $A_u, A_w, \dots, A_z$ .

Przykładowo, przeanalizujemy krotkę  $t = \langle \text{'Barbara Bednarek'}, '85020834804', '839-84-61', \text{'ul. Długa 7a'}, \text{brak}, 19, 4.42 \rangle$  z relacji STUDENT (patrz rysunek 5.1), dla której możemy użyć następującej notacji:  $t[\text{Nazwisko}] = \langle \text{'Barbara Bednarek'} \rangle$  oraz  $t[\text{PESEL}, \text{ŚrOcen}, \text{Wiek}] = \langle '839-84-61', 4.42, 19 \rangle$ .

## 5.2. Ograniczenia modelu relacyjnego i schematy relacyjnych baz danych

Do tej pory omówiliśmy właściwości pojedynczych relacji. W relacyjnych bazach danych istnieje zwykle wiele relacji, a należące do nich krotki przeważnie są ze sobą na różne sposoby powiązane. Stan całej bazy danych w określonym punkcie czasu będzie odpowiadał stanom wszystkich jego relacji. Zasadniczo, może istnieć wiele **ograniczeń** nakładanych na rzeczywiste wartości w poszczególnych stanach bazy danych. Takie ograniczenia są definiowane na podstawie reguł istniejących w reprezentowanym przez bazę danych mini-świecie (patrz punkt 1.6.8).

W tym podrozdziale omówimy różne ograniczenia nakładane na dane, które można definiować w relacyjnych bazach danych. Ograniczenia określane w bazie danych można podzielić na trzy główne kategorie:

- (1) Ograniczenia wynikające ze stosowanego modelu danych — nazywamy je **charakterystycznymi ograniczeniami opartymi na modelu danych** lub ograniczeniami pośrednimi.
- (2) Ograniczenia, które mogą być bezpośrednio wyrażane w schematach danego modelu danych, zwykle przez tworzenie odpowiednich definicji w języku DDL (języku definiowania danych, patrz punkt 2.3.1). Nazywamy je **ograniczeniami opartymi na schemacie** lub **ograniczeniami bezpośrednimi**.
- (3) Ograniczenia, które *nie mogą* być wyrażane bezpośrednio w schematach stosowanego modelu danych, zatem muszą być wyrażane i wymuszane na poziomie programów aplikacji. Nazywamy je **ograniczeniami opartymi na aplikacjach**, **ograniczeniami semantycznymi** lub **regułami biznesowymi**.

Omówione w punkcie 5.1.2 właściwości relacji stanowią zestaw ograniczeń należących do pierwszej kategorii (a więc wynikających ze stosowanego modelu danych); przykładowo, ograniczenie, które mówi, że żadna relacja nie może zawierać dwóch takich samych krotek, jest charakterystyczne dla relacyjnego modelu danych. Wszystkie ograniczenia, które będziemy omawiali w kolejnych punktach tego podrozdziału, należą do drugiej kategorii, a więc mogą być wyrażane bezpośrednio w schemacie modelu relacyjnego za pomocą języka DDL. Ograniczenia należące do ostatniej kategorii są bardziej ogólne, dotyczą znaczenia, a także działania atrybutów, są trudne do wyrażania i wymuszania na poziomie modelu danych, zatem zgodność danych z tymi ograniczeniami jest często sprawdzana na poziomie aplikacji aktualizujących bazę. W niektórych sytuacjach takie ograniczenia można zapisać jako asercje w języku SQL (patrz rozdział 7.).

Inną ważną kategorią ograniczeń są *zależności danych*, które obejmują *zależności funkcyjne* i *zależności wielowartościowe*. Ograniczenia tego typu są wykorzystywane przede wszystkim do testowania jakości projektu relacyjnej bazy danych i w trakcie procesu nazywanego *normalizacją* (patrz rozdział 14. i 15.).

Do ograniczeń opartych na schemacie należą ograniczenia dziedziny, ograniczenia klucza, ograniczenia wartości pustych, więzy integralności encji oraz więzy integralności odwołań (powiązań).

### 5.2.1. Ograniczenia dziedziny

Ograniczenia dziedziny określają, że wewnątrz każdej z krotek zawartość każdego atrybutu *A* musi być atomową wartością należącą do dziedziny *dom(A)*. Sposoby, w jakie można określać dziedziny wartości atrybutów, omówiliśmy już w punkcie 5.1.1. Do powiązanych z dziedzinami typów danych należą najczęściej liczby całkowite (krótkie liczby całkowite, liczby całkowite i długie liczby całkowite) oraz liczby rzeczywiste (zmiennoprzecinkowe i zmiennoprzecinkowe podwójnej precyzji). Zwykle mamy do dyspozycji także znaki, wartości logiczne, ciągi znaków stałej długości, ciągi znaków zmiennej długości, datę, czas, znacznik czasu oraz specjalne typy danych. Inne możliwe dziedziny mogą być opisywane przez zakresy wartości wymienionych typów danych lub wyliczeniowe typy danych, w których wszystkie dopuszczalne wartości są jawnie wymienione. Zamiast omawiać w tym miejscu szczegóły związane z dziedzinami wartości atrybutów, w podrozdziale 6.1 omówimy typy danych oferowane w standardzie relacyjnym SQL.

## 5.2.2. Ograniczenia klucza i ograniczenia wartości pustych

Zgodnie z definicją *relacja* jest *zbiorem krotek*. Skoro relacja jest zbiorem, wszystkie jej elementy muszą być różne, co oznacza, że każda z krotek musi być unikatowa na poziomie danej relacji. Nie mogą więc istnieć dwie krotki będące taką samą kombinacją wartości we *wszystkich* atrybutach. Przeważnie istnieją także inne **podzbiory atrybutów** schematu relacji  $R$  z dodatkową własnością, która określa, że w żadnym stanie  $r$  relacji  $R$  nie mogą istnieć dwie krotki z taką samą kombinacją wartości tych wybranych atrybutów. Przypuśćmy, że jeden z takich podzbiorów oznaczamy przez  $SK$ ; wówczas dla każdej pary *różnych* krotek  $t_1$  i  $t_2$  w stanie  $r$  relacji  $R$  możemy zapisać następujące ograniczenie:

$$t_1[SK] \neq t_2[SK]$$

Każdy taki zbiór atrybutów  $SK$  jest nazywany **nadkluczem** (ang. *superkey*) schematu relacji  $R$ . Nadklucz  $SK$  określa *ograniczenie unikatowości*, które mówi, że w żadnym stanie  $r$  relacji  $R$  nie mogą istnieć dwie różne krotki mające taką samą wartość nadklucza  $SK$ . Każda relacja ma co najmniej jeden domyślny nadklucz — zbiór wszystkich swoich atrybutów. Nadklucz może zawierać atrybuty nadmiarowe, zatem bardziej przydatne są *klucze*, które nie mogą zawierać nadmiarowości. **Klucz**  $K$  schematu relacji  $R$  jest takim nadkluczem tego schematu, który spełnia dodatkowy warunek: usunięcie dowolnego atrybutu  $A$  z klucza  $K$  powoduje, że otrzymany zbiór atrybutów  $K'$  nie jest już nadkluczem schematu relacji  $R$ . Oznacza to, że klucz podlega dwóm ograniczeniom:

- (1) Dwie różne krotki w żadnym stanie relacji nie mogą zawierać identycznych wartości we (wszystkich) atrybutach należących do klucza. Cecha *unikatowości* dotyczy także nadklucza.
- (2) Klucz jest *minimalnym nadkluczem*, czyli takim, z którego nie możemy usunąć żadnych atrybutów, jeśli chcemy zachować ograniczenie unikatowości. Cecha *minimalności* jest wymagana dla klucza, ale opcjonalna dla nadklucza.

Tak więc klucz zawsze jest nadkluczem, ale już nie na odwrót. Nadklucz może być kluczem (jeśli jest minimalny), ale nie musi nim być (jeżeli nie jest minimalny). Przeanalizujmy raz jeszcze relację STUDENT z rysunku 5.1. Zbiór atrybutów {PESEL} jest kluczem tej relacji, ponieważ nie mogą istnieć dwie krotki reprezentujące studentów z takimi samymi wartościami atrybutu PESEL<sup>7</sup>. Oznacza to, że każdy zbiór atrybutów, do którego należy atrybut PESEL — np. {PESEL, Nazwisko, Wiek} — jest nadkluczem. Warto jednak pamiętać, że nadklucz {PESEL, Nazwisko, Wiek} nie jest kluczem relacji STUDENT, ponieważ usunięcie z tego zbioru atrybutu Nazwisko, atrybutu Wiek lub obu tych atrybutów jednocześnie nie spowoduje naruszenia ograniczenia nadklucza. Każdy nadklucz będący zbiorem jednoelementowym (obejmujący pojedynczy atrybut) jest także kluczem. Klucz składający się z wielu atrybutów musi wymagać od *wszystkich* swoich atrybutów spełnienia właściwości unikatowości.

Wartość atrybutu klucza może być wykorzystywana do unikatowego identyfikowania poszczególnych krotek w relacji. Przykładowo, wartość 85122065489 atrybutu PESEL unikatowo identyfikuje krotkę reprezentującą w relacji STUDENT osobę Beniamina Baki. Warto pamiętać, że zbiór atrybutów tworzących klucz jest własnością całego schematu

---

<sup>7</sup> Łatwo zauważyć, że PESEL jest także nadkluczem.

relacji — należy ten zbiór traktować jak ograniczenie, które powinno być spełnione w *każdym* poprawnym stanie relacji tego schematu. Klucz jest wyznaczany w oparciu o znaczenia poszczególnych atrybutów. Ta własność schematu relacji jest *niezmienna w czasie* — musi być zachowana także wtedy, gdy będziemy wstawiali nowe krotki do relacji. Przykładowo, nigdy nie powinniśmy do roli klucza wyznaczać atrybutu *Nazwisko* relacji STUDENT (patrz rysunek 5.1), ponieważ możliwe jest występowanie dwóch (lub więcej) studentów z takim samym nazwiskiem w prawidłowym stanie relacji<sup>8</sup>.

Zasadniczo, schemat relacji może mieć więcej niż jeden klucz. W takim przypadku każdy z kluczy jest nazywany **kluczem kandydującym**. Przykładowo, relacja SAMOCHÓD (patrz rysunek 5.4) zawiera dwa klucze kandydujące: NumerRejestracyjny i NumerSilnika. Zwykle do roli **klucza głównego** relacji wyznacza się właśnie jeden z jej kluczy kandydujących. Taki klucz kandydujący zawiera wówczas wartości wykorzystywane do *identyfikowania* krotek w relacji. W tej książce stosujemy konwencję, zgodnie z którą atrybuty składające się na klucz główny schematu relacji są podkreślane (patrz rysunek 5.4). Należy pamiętać, że w schematach relacji zawierających wiele kluczy kandydujących wybór jednego z nich do roli klucza głównego jest dowolny — w wielu przypadkach lepszym rozwiązaniem jest jednak wyznaczenie takiego klucza głównego, który zawiera jak najmniejszą liczbę atrybutów (najlepiej jeden). Pozostałe klucze kandydujące są **kluczami unikatowymi** i nie są podkreślane.

SAMOCHÓD

<u>NumerRejestracyjny</u>	NumerNadwozia	Marka	Model	Rok
DW 5345E	A69352	Ford	Mustang	02
FZ 3242B	B43696	Oldsmobile	Cutlass	05
PO 2943H	X83554	Oldsmobile	Delta	01
GDA C545	C43742	Mercedes	190-D	99
GDA E629	Y82935	Toyota	Camry	04
DW 6521N	U028365	Jaguar	XJS	04

RYСУNEK 5.4. Relacja SAMOCHÓD z dwoma kluczami kandydującymi: NumerRejestracyjny oraz NumerSilnika

Jeszcze inne ograniczenie nakładane na atrybuty określa, czy stosowanie wartości pustych jest dopuszczalne czy zabronione. Przykładowo, jeśli każda krotka relacji STUDENT musi zawierać poprawną, niepustą wartość atrybutu *Nazwisko*, wówczas na atrybut *Nazwisko* relacji STUDENT nakładane jest ograniczenie niedopuszczalności wartości pustej.

### 5.2.3. Relacyjne bazy danych i schematy relacyjnych baz danych

Omówione do tej pory definicje i ograniczenia dotyczą pojedynczych relacji oraz ich atrybutów. Relacyjna baza danych składa się zwykle z wielu relacji, których krotki są ze sobą powiązane na wiele różnych sposobów. W tym podrozdziale zdefiniujemy relacyjną bazę danych oraz schemat relacyjnej bazy danych.

<sup>8</sup> Nazwiska są w niektórych sytuacjach wykorzystywane w roli unikatowych kluczy, jednak należy wówczas koniecznie stosować jakieś dodatkowe oznaczenia, które odróżnią krotki z identycznymi nazwiskami — np. przez dołączenie liczb porządkowych.

**Schemat relacyjnej bazy danych**  $S$  jest zbiorem schematów relacji, zatem  $S = \{R_1, R_2, \dots, R_m\}$ , oraz zbiorem **więzów integralności**  $WI$ . **Stan relacyjnej bazy danych**<sup>9</sup>  $BD$  schematu  $S$  jest zbiorem stanów relacji  $BD = \{r_1, r_2, \dots, r_m\}$ , gdzie każde  $r_i$  jest takim stanem relacji  $R_i$ , że spełnia więzy integralności zdefiniowane w zbiorze  $WI$ . Na rysunku 5.5 przedstawiono schemat relacyjnej bazy danych, którą będziemy nazywali FIRMA = {PRACOWNIK, DZIAŁ, LOKALIZACJE\_DZIAŁÓW, PROJEKT, PRACUJE\_NAD, CZŁONEK\_RODZINY}. Podkreślone atrybuty reprezentują klucze główne poszczególnych relacji. Na rysunku 5.6 przedstawiono stan relacyjnej bazy danych odpowiadający schematowi FIRMA. Będziemy wykorzystywali zaprezentowany schemat i stan bazy danych podczas tworzenia i analizowania przykładowych zapytań w różnych językach relacyjnych w rozdziałach od 4. do 6.

**PRACOWNIK**

IMIĘ	INICJAŁDRIMENIA	NAZWISKO	<u>PESEL</u>	DATAUR	ADRES	PŁEĆ	PENSJA	PESELPRZEŁOŻ	NRDZ
------	-----------------	----------	--------------	--------	-------	------	--------	--------------	------

**DZIAŁ**

NAZWADZ	<u>NUMERDZ</u>	PESELKIEROWNIKA	DATAPRZEJKIEROWNICTWA
---------	----------------	-----------------	-----------------------

**LOKALIZACJE\_DZIAŁÓW**

<u>NUMERDZ</u>	<u>LOKALIZACJADZ</u>
----------------	----------------------

**PROJEKT**

NAZWAPROJ	<u>NUMERPROJ</u>	LOKALIZACJAPROJ	NR_DZ
-----------	------------------	-----------------	-------

**PRACUJE\_NAD**

<u>PESELPRAC</u>	<u>NR_PROJ</u>	GODZINY
------------------	----------------	---------

**CZŁONEK\_RODZINY**

<u>PESELPRAC</u>	<u>IMIĘ_CZŁONKA_RODZINY</u>	PŁEĆ	DATAUR	STOPIEŃ_POKREWIEŃSTWA
------------------	-----------------------------	------	--------	-----------------------

RYSUNEK 5.5. Diagram schematu dla schematu relacyjnej bazy danych FIRMA

Kiedy będziemy w tej książce wspominali o relacyjnej bazie danych, tak naprawdę będą to odwołania zarówno do jej schematu, jak i jej bieżącego stanu. Stan bazy danych, który nie spełnia więzów integralności, jest nazywany **stanem nieprawidłowym**; natomiast stan, który spełnia wszystkie więzy integralności zdefiniowane w zbiorze  $WI$ , jest nazywany **stanem prawidłowym**.

Na rysunku 5.5 atrybut NUMERDZ, występujący zarówno w relacji DZIAŁ, jak i w relacji LOKALIZACJE\_DZIAŁÓW, reprezentuje ten sam element świata rzeczywistego — numer przypisany do działu firmy. Ten sam element jest reprezentowany przez atrybut NRDZ w relacji PRACOWNIK i NR\_DZ w relacji PROJEKT. Atrybuty reprezentujące w różnych relacjach to samo pojęcie ze świata rzeczywistego mogą, ale nie muszą mieć takich samych nazw. Co więcej, atrybuty, które reprezentują różne elementy występujące w świecie rzeczywistym, mogą

<sup>9</sup> Stan relacyjnej bazy danych jest w niektórych sytuacjach nazywany *egzemplarzem* relacyjnej bazy danych. Wspominaliśmy już jednak, że nie będziemy się posługiwali pojęciem *egzemplarza* w tym kontekście, ponieważ w literaturze można je spotkać w odniesieniu do pojedynczych krotek.



PRACOWNIK									
IMIĘ	INICJAŁDRIMIENIA	NAZWISKO	PESEL	DATAUR	ADRES	PLEĆ	PENSJA	PESELPRZEŁOŻ	NRDZ
Jan	B	Szewczyk	65010912345	1965-01-09	ul. Kręta 4/3, Szczecin	M	3000	55120834598	5
Franciszek	T	Wieszczycki	55120834598	1955-12-08	os. T. Kościuszki 4a/11, Szczecin	M	4000	37111045873	5
Alicja	J	Zalewska	68011932514	1968-01-19	os. Piastowskie 2/23, Swarzędz	K	2500	41062013258	4
Janina	S	Wojtczak	41062013258	1941-06-20	ul. Kwiatowa 78, Mosina	K	4300	37111045873	4
Robert	K	Napierski	62091502054	1962-09-15	ul. Podgórna 7, Police	M	3800	55120834598	5
Joanna	A	Englert	72073110039	1972-07-31	ul. Wielka 40, Szczecin	K	2500	55120834598	5
Albert	W	Janiszewski	69032923149	1969-03-29	os. Jana Pawła II 13/4, Szczecin	M	2500	41062013258	4
Józef	E	Bąk	37111045873	1937-11-10	os. Centrum 45, Szczecin	M	5500	brak	1

DZIAŁ			
NAZWADZ	NUMERDZ	PESELKIEROWNIKA	DATAPRZEJKIEROWNICTWA
Badania	5	55120834598	1988-05-22
Administracja	4	41062013258	1995-01-01
Centrala	1	37111045873	1981-06-19

LOKALIZACJE_DZIAŁÓW	
NUMERDZ	LOKALIZACJADZ
1	Szczecin
4	Poznań
5	Bydgoszcz
5	Świebodzin
5	Szczecin

PRACUJE_NAD		
PESELPRAC	NR_PROJ	GODZINY
65010912345	1	32.5
65010912345	2	7.5
62091502054	3	40.0
72073110039	1	20.0
72073110039	2	20.0
55120834598	2	10.0
55120834598	3	10.0
55120834598	10	10.0
55120834598	20	10.0
68011932514	30	30.0
68011932514	10	10.0
69032923149	10	35.0
69032923149	30	5.0
41062013258	30	20.0
41062013258	20	15.0
37111045873	20	null

PROJEKT			
NAZWAPROJ	NUMERPROJ	LOKALIZACJAPROJ	NR_DZ
ProduktX	1	Bydgoszcz	5
ProduktY	2	Świebodzin	5
ProduktZ	3	Szczecin	5
Komputeryzacja	10	Poznań	4
Reorganizacja	20	Szczecin	1
ZwiększenieZysków	30	Poznań	4

CZŁONEK_RODZINY				
PESELPRAC	IMIĘ_CZŁONKA_RODZINY	PLEĆ	DATAUR	STOPIEŃ_POKREWIEŃSTWA
55120834598	Alicja	K	1986-04-05	CÓRKA
55120834598	Teodor	M	1983-10-25	SYN
55120834598	Janina	K	1958-05-03	ZONA
41062013258	Alojzy	M	1942-02-28	MAŻ
65010912345	Michał	M	1988-01-04	SYN
65010912345	Alicja	K	1988-12-30	CÓRKA
65010912345	Elżbieta	K	1967-05-05	ZONA

RYSunek 5.6. Jeden z możliwych stanów bazy danych dla schematu relacyjnej bazy danych FIRMA

mieć takie same nazwy w różnych relacjach. Przykładowo, moglibyśmy użyć po prostu atrybutu NAZWA zamiast atrybutów NAZWAPROJ i NAZWADZ odpowiednio w relacjach PROJEKT i DZIAŁ — w takim przypadku mielibyśmy do czynienia z dwoma atrybutami oznaczonymi tą samą nazwą, ale reprezentującymi dwa różne pojęcie ze świata rzeczywistego (nazwy projektów i nazwy działów).

Niektóre z wczesnych wersji modelu relacyjnego opierały się na założeniu, które przewidywało stosowanie *identycznych* nazw dla atrybutów reprezentujących we wszystkich relacjach ten sam element świata rzeczywistego. Takie podejście rodziło problemy w sytuacjach, gdy ten sam element świata rzeczywistego był wykorzystywany w różnych rolach (znaczeniach) w jednej relacji. Przykładowo, w relacji PRACOWNIK numer PESEL występuje dwa razy (patrz rysunek 5.5): raz w roli numeru PESEL pracownika, drugi raz w roli numeru PESEL bezpośredniego przełożonego tego pracownika (a więc innego pracownika). Nadaliśmy tym elementom dwie różne nazwy atrybutów (odpowiednio PESEL i PESELPRZEŁOŻ), aby umożliwić odróżnianie ich znaczeń.



Każdy relacyjny system zarządzania bazą danych musi oferować język definiowania danych (ang. *Data Definition Language* — *DDL*), za pomocą którego projektant będzie mógł definiować schematy relacyjnych baz danych. Współczesne relacyjne systemy zarządzania bazami danych w zdecydowanej większości wykorzystują do tego celu język SQL. Używany w nim język DDL zaprezentujemy w podrozdziałach 6.1 i 6.2.

Dla schematu bazy danych określa się wspomniane już więzy integralności, które muszą być spełnione w *każdym poprawnym stanie bazy danych* zgodnej z tym schematem. Poza dziedziną, kluczem i ograniczeniami stosowania wartości pustych za nieodłączną część modelu relacyjnego uważa się także dwa inne rodzaje ograniczeń: integralność encji i integralność odwołań.

### 5.2.4. Integralność encji, integralność odwołań i klucze obce

**Więzy integralności encji** określają, że żadna wartość atrybutu pełniącego rolę klucza głównego nie może być pusta. Wynika to z faktu, że wartości klucza głównego są wykorzystywane do unikatowego identyfikowania poszczególnych krotek w relacji. Umieszczenie wartości pustej w kluczu głównym oznaczałoby brak możliwości identyfikacji niektórych krotek. Przykładowo, gdyby dwie lub więcej krotek miało przypisaną wartość pustą do atrybutu pełniącego rolę klucza głównego, nie mielibyśmy możliwości ich odróżnienia przy próbie odwołania się do nich z poziomu innych relacji.

Ograniczenia klucza oraz więzy integralności encji są określane osobno dla poszczególnych relacji. **Więzy integralności odwołań** są definiowane pomiędzy parami relacji — wykorzystuje się je do utrzymywania spójności powiązanych ze sobą krotek, które należą do tych dwóch relacji. Mówiąc nieformalnie, więzy integralności odwołań oznaczają, że krotka w jednej relacji, która odwołuje się do innej relacji, zawsze (w każdym stanie bazy danych) musi wskazywać na *istniejącą krotkę* tamtej relacji. Przykładowo, atrybut NRDZ relacji PRACOWNIK zawiera numer działu, w którym pracuje dany pracownik (patrz rysunek 5.6); oznacza to, że każda krotka relacji PRACOWNIK musi zawierać w atrybucie NRDZ wartość równą wartości atrybutu NUMERDZ którejś z krotek relacji DZIAŁ.

Aby zdefiniować *więzy integralności odwołań* w sposób bardziej formalny, musimy najpierw wprowadzić pojęcie *klucza obcego*. Podane poniżej warunki dla prawidłowego klucza obcego pośrednio określają więzy integralności odwołań pomiędzy dwoma schematami relacji:  $R_1$  i  $R_2$ . Zbiór atrybutów *FK* w schemacie relacji  $R_1$  jest **kluczem obcym** (ang. *foreign key*) tego schematu, który **odwołuje się** do relacji  $R_2$  tylko wtedy, gdy spełnione są następujące warunki:

- (1) Wartości atrybutów w zbiorze *FK* należą do tej samej dziedziny (dziedzin) co atrybuty pełniące rolę klucza głównego (należące do zbioru *PK*) schematu relacji  $R_2$ ; mówimy, że atrybuty w zbiorze *FK* **odwołują się** do relacji  $R_2$  lub **wskazują** na tę relację.
- (2) Wartość klucza obcego (*FK*) w krotce  $t_1$  bieżącego stanu  $r_1(R_1)$  albo musi być równa wartości klucza głównego (*PK*) pewnej krotki  $t_2$  w bieżącym stanie  $r_2(R_2)$ , albo musi być *wartością pustą*. W pierwszym przypadku mamy równość  $t_1[FK] = t_2[PK]$  i mówimy, że krotka  $t_1$  **odwołuje się** do krotki  $t_2$  lub **wskazuje** na tę krotkę.

W powyższej definicji relacja  $R_1$  jest nazywana **relacją odwołującą**, natomiast relacja  $R_2$  jest nazywana **relacją wskazowaną**. Jeśli oba wymienione powyżej warunki są spełnione,

uważa się, że **więzy integralności odwołań** relacji  $R_1$  do relacji  $R_2$  są spełnione. W bazach danych składających się z wielu relacji istnieje zwykle wiele takich więzów integralności odwołań.

Zanim przystąpimy do definiowania tego typu więzów, musimy najpierw dokładnie zrozumieć znaczenie i rolę odgrywaną przez każdy zbiór atrybutów w poszczególnych schematach relacji interesującej nas bazy danych. Więzy integralności odwołań są zwykle określane w oparciu o *powiązania istniejące pomiędzy encjami* reprezentowanymi przez różne schematy relacji. Przykładowo, przeanalizujmy raz jeszcze bazę danych przedstawioną na rysunku 5.6. Atrybut NRDZ relacji PRACOWNIK odwołuje się do działu, w którym dany pracownik jest zatrudniony; oznacza to, że wyznaczamy atrybut NRDZ do roli klucza obcego relacji PRACOWNIK, który odwołuje się do relacji DZIAŁ. Wartość atrybutu NRDZ w dowolnej krotce  $t_1$  relacji PRACOWNIK musi więc albo odpowiadać wartości klucza głównego relacji DZIAŁ (atrybutu NUMERDZ) w pewnej krotce  $t_2$  tej relacji, albo być *wartością pustą* (jeśli reprezentowany przez krotkę  $t_1$  pracownik nie jest zatrudniony w żadnym dziale). Na rysunku 5.6 krotka reprezentująca pracownika Jana Nowaka odwołuje się do krotki reprezentującej dział badawczy, co oznacza, że Jan Nowak pracuje w dziale badawczym.

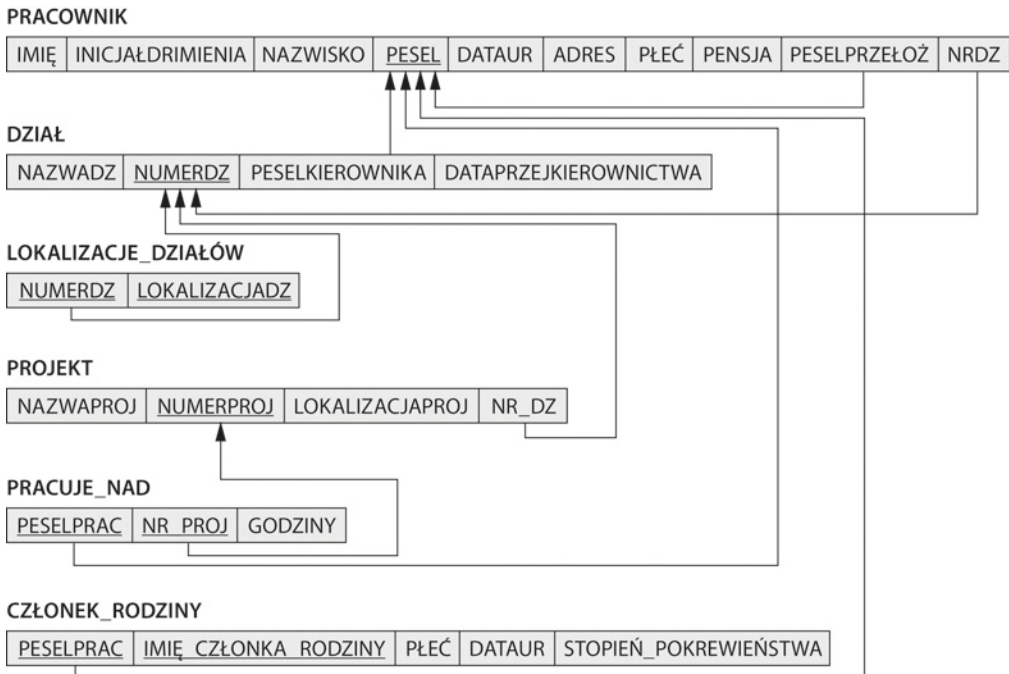
Warto pamiętać, że klucz obcy może się *odwoływać do własnej relacji*. Przykładowo, atrybut PESELPRZEŁOŻ relacji PRACOWNIK wskazuje na bezpośredniego przełożonego danego pracownika, który jest innym pracownikiem reprezentowanym przez inną krotkę w tej samej relacji PRACOWNIK. Oznacza to, że atrybut PESELPRZEŁOŻ jest kluczem obcym wskazującym na tę samą relację PRACOWNIK. W bazie danych przedstawionej na rysunku 5.6 krotka reprezentująca pracownika Jana Nowaka odwołuje się do krotki reprezentującej pracownika Fryderyka Kowalskiego, co oznacza, że Fryderyk Kowalski jest zwierzchnikiem Jana Nowaka.

*Więzy integralności odwołań możemy przedstawiać na diagramach* za pomocą skierowanych łuków łączących każdy klucz obcy ze wskazywaną przez niego relacją. Dla jasności grot takich strzałek może wskazywać na klucz główny relacji, do której odwołuje się tak reprezentowany klucz obcy. Na rysunku 5.7 przedstawiono schemat z rysunku 5.5 z oznaczonymi w ten sposób więzami integralności odwołań.

Wszystkie więzy integralności powinny być definiowane na poziomie schematu relacyjnej bazy danych (czyli w jej definicji), jeśli zgodność z nimi ma być wymuszana we wszystkich stanach tej bazy danych. Oznacza to, że język definiowania danych (DDL) powinien oferować środki niezbędne do określania różnego rodzaju więzów, które będą następnie automatycznie wymuszane przez system zarządzania bazą danych. W języku SQL instrukcja CREATE TABLE z języka DDL umożliwi m.in. zdefiniowanie ograniczeń klucza głównego, klucza unikatowego i występowania wartości pustych oraz więzów integralności encji i integralności odwołań (patrz podrozdziały 6.1 i 6.2).

### 5.2.5. Pozostałe typy ograniczeń

Przedstawione więzy integralności są w języku DDL, ponieważ występują w większości aplikacji baz danych. Język DDL nie obejmuje innej klasy ogólnych więzów (nazywanych niekiedy *ograniczeniami integralności semantycznej*), które często musimy definiować i wymuszać w inny sposób. Przykłady takich ograniczeń (wyzów) mogą mieć postać: „pensja pracownika nie powinna przekraczać pensji jego bezpośredniego przełożonego” lub „maksymalna liczba



RYSUNEK 5.7. Więzy integralności odwołań naniesione na schemat relacyjnej bazy danych FIRMA

godzin poświęcanych w ciągu tygodnia przez pracownika na realizację wszystkich projektów nie może przekraczać 56<sup>10</sup>. Takie ograniczenia mogą być określone i wymuszane już na poziomie programów aplikacji, które aktualizują bazę danych, lub w systemie zarządzania bazą danych, gdzie można wykorzystać uniwersalny **język definiowania więzów**. Można do tego celu stosować mechanizmy nazywane **wyzwalaczami** oraz **asercjami**. W standardzie SQL do tworzenia tego typu ograniczeń służą polecenia CREATE ASSERTION i CREATE TRIGGER (patrz rozdział 7.). Bardziej popularnym rozwiązaniem jest sprawdzanie tych ograniczeń wewnątrz programów aplikacji zamiast ich definiowania w językach definiowania więzów, ponieważ w tym drugim przypadku prawidłowa obsługa więzów jest znacznie trudniejsza i bardziej skomplikowana (patrz podrozdział 26.1).

Omówione do tej pory odmiany więzów i ograniczeń można nazywać **ograniczeniami stanów**, ponieważ definiują reguły, które muszą być spełnione przez każdy *poprawny stan* bazy danych. Zupełnie innym typem ograniczeń są tzw. **ograniczenia przejść**, które można definiować dla przejść pomiędzy stanami bazy danych<sup>10</sup>. Przykład ograniczenia przejścia może mieć postać: „pensja pracownika może tylko rosnąć (nie może maleć)”. Tego typu ograniczenia są zwykle wymuszane przez programy aplikacji lub też definiowane w oparciu o aktywne reguły i wyzwalacze (patrz podrozdział 26.1).

<sup>10</sup> Ograniczenia stanów są czasami nazywane *ograniczeniami statycznymi*, natomiast ograniczenia przejść nazywa się czasami *ograniczeniami dynamicznymi*.

## 5.3. Operacje aktualizacji, transakcje i obsługa naruszeń więzów integralności

Operacje obsługiwane w modelu relacyjnym można podzielić na dwie główne kategorie: *wyszukiwania* i *aktualizacji* danych. Operacje algebry relacyjnej, które mogą być wykorzystywane do definiowania operacji **pobierania** (lub wyszukiwania), szczegółowo omówimy w rozdziale 8. Wyrażenie algebry relacyjnej tworzy (po zastosowaniu kilku operatorów algebraicznych dla istniejącego zbioru relacji) nową relację — głównym zastosowaniem takich wyrażeń jest wykonywanie zapytań na bazie danych. Użytkownik formułuje zapytanie, które opisuje interesujące go dane; natomiast do wyszukania tych danych niezbędne jest stworzenie nowej relacji przez zastosowanie operacji relacyjnych na istniejącym zbiorze relacji. Tak utworzona **relacja wynikowa** staje się odpowiedzią na zapytanie użytkownika. W rozdziale 8. wprowadzimy dodatkowo język nazywany rachunkiem relacji, który jest wykorzystywany do definiowania nowych relacji bez konieczności precyzowania porządku wykonywanych operacji.

W tym podrozdziale skupimy się na operacjach **modyfikowania** i **aktualizacji** bazy danych. Istnieją trzy podstawowe operacje aktualizujące wykonywane na relacjach: wstawianie, usuwanie i modyfikowanie. **Wstawianie** jest wykorzystywane do dodawania do istniejącej relacji nowej krotki lub krotek. **Usuwanie** służy do usuwania krotek, natomiast **aktualizacja (modyfikacja)** ma na celu zmianę wartości niektórych atrybutów w istniejących krotkach. Za każdym razem, gdy wymienione operacje są stosowane na relacjach relacyjnej bazy danych, należy się upewnić, że zdefiniowane w schemacie tej bazy danych więzy integralności nie zostaną naruszone w wyniku tych operacji. W tym podrozdziale omówimy rodzaje ograniczeń, które mogą być naruszane przez poszczególne typy operacji aktualizacji, oraz działania, które można podjąć w przypadku wystąpienia tego typu naruszeń. W naszych przykładach będziemy się posługiwać bazą danych przedstawioną na rysunku 5.6; dodatkowo, nasza analiza będzie dotyczyła wyłącznie ograniczeń dziedziny, ograniczeń klucza, więzów integralności encji oraz więzów integralności odwołań, które są widoczne na rysunku 5.7. Dla każdego typu operacji aktualizacji podajemy odpowiedni przykład i omawiamy wszystkie ograniczenia, które mogą zostać naruszone w czasie wykonywania prezentowanych działań.

### 5.3.1. Operacja wstawiania

Operacja **wstawiania** wykorzystuje listę wartości atrybutów dla nowej krotki  $t$ , która ma zostać wstawiona w relacji  $R$ . Operacja wstawiania może naruszyć każdy z czterech omówionych w poprzednim podrozdziale rodzajów ograniczeń i więzów. Ograniczenia dziedziny mogą być naruszone w sytuacji, gdy wartość atrybutu nie należy do odpowiedniego zbioru, który został wyznaczony do roli dziedziny wartości tego atrybutu, lub jest niewłaściwego typu. Ograniczenia klucza mogą być naruszone wtedy, gdy wartość klucza nowej krotki  $t$  już istnieje w innej krotce relacji  $r(R)$ . Naruszenie więzów integralności encji może polegać na przekazaniu wartości pustej dla atrybutu klucza nowej krotki  $t$ . Więzy integralności odwołań można naruszyć, wstawiając taką wartość któregośkolwiek z kluczy obcych nowej krotki  $t$ , która nie istnieje we wskazywanej relacji. Poniżej przedstawiono kilka przykładów ilustrujących omówione przed chwilą sytuacje:

- *Operacja:*

Wstaw <'Cecylia', 'Iłona', 'Kowalska', brak, '1960-04-05', 'ul. Wiatraczna 3. Leśmierz', K, 2800, null, 4> do PRACOWNIK.

*Wynik:* To polecenie ilustruje naruszenie więzów integralności encji (wartość pusta dla klucza głównego PESEL), zatem zostanie odrzucone.

- *Operacja:*

Wstaw <'Alicja', 'Joanna', 'Żak', '68011932514', '1968-01-19', 'ul. Wiatraczna 3. Leśmierz', K, 2800, '410620132586', 4> do PRACOWNIK.

*Wynik:* To polecenie narusza ograniczenie klucza, ponieważ w relacji PRACOWNIK istnieje inna krotka z taką samą wartością atrybutu PESEL — operacja zostanie więc odrzucona.

- *Operacja:*

Wstaw <'Cecylia', 'Iłona', 'Kowalska', '60040512839', '1960-04-05', 'os. Bajkowe 34/4h', K, 2800, '410620132586', 7> do PRACOWNIK.

*Wynik:* To polecenie narusza zdefiniowane dla atrybutu NRDZ więzy integralności odwołań, ponieważ w relacji DZIAŁ nie istnieje krotka z wartością atrybutu NUMERDZ równą 7.

- *Operacja:*

Wstaw <'Cecylia', 'Iłona', 'Kowalska', '60040512839', '1960-04-05', 'ul. Wiatraczna 3. Leśmierz', K, 2800, null, 4> do PRACOWNIK.

*Wynik:* To polecenia wstawienia krotki spełnia wszystkie wymogi, zatem zostanie prawidłowo wykonane.

Jeśli dana operacja wstawiania narusza jedno lub więcej ograniczeń, domyślnym rozwiązaniem jest jej *odrzućenie*. W takim przypadku dobrym zwyczajem jest taka obsługa operacji przez system zarządzania bazą danych, która pozwoli wyjaśnić użytkownikowi powody odrzucenia próby wstawienia krotki. Zupełnie innym rozwiązaniem jest podjęcie próby *poprawienia elementu, który spowodował odrzucenie operacji*, jednak tego typu działania *rzadko spotyka się w przypadku naruszeń wynikających z nieprawidłowych operacji wstawiania* — znacznie częściej dotyczą one takich operacji jak usuwanie czy aktualizacja. W przypadku pierwszej z przedstawionych powyżej operacji system zarządzania bazą danych mógłby wezwać użytkownika do podania wartości atrybutu PESEL i zaakceptować nową krotkę po odpowiednim uzupełnieniu danych wejściowych. W przypadku trzeciej operacji system zarządzania bazą danych mógłby albo zażądać od użytkownika zmiany wartości atrybutu NRDZ na prawidłową (czyli odpowiadającą istniejącemu działowi lub pustą), albo wskazać użytkownikowi możliwość wstawienia nowej krotki w relacji DZIAŁ z wartością NUMERDZ = 7 i zaakceptować oryginalną operację wstawienia dopiero po wykonaniu tej dodatkowej operacji. Warto pamiętać, że w tym drugim przypadku naruszenie wynikające z operacji wstawiania może zostać z powrotem **przeniesione** do relacji PRACOWNIK w sytuacji, gdy użytkownik spróbuje wstawić krotkę reprezentującą 7. dział firmy z taką wartością atrybutu PESELKIEROWNIKA, która nie istnieje w relacji PRACOWNIK.

### 5.3.2. Operacja usuwania

Operacja **usuwania** może naruszać wyłącznie więzy integralności odwołań w sytuacji, gdy usuwana krotka jest wskazywana przez klucze obce występujące w pozostałych krotkach bazy danych. Zdefiniowanie operacji usuwania wymaga określenia takich warunków dla atrybutów relacji, które pozwolą pobrać krotkę (lub krotki) przeznaczone do usunięcia. Poniżej przedstawiono kilka przykładów:

- *Operacja:*

Usuń z relacji PRACUJE\_NAD krotkę z wartością PESELPRAC = '68011932514' oraz NR\_PROJ = 10.

*Wynik:* To polecenie usunięcia jest dopuszczalne i usunie jedną krotkę.

- *Operacja:*

Usuń z relacji PRACOWNIK krotkę z wartością PESEL = '68011932514'.

*Wynik:* To polecenie nie zostanie wykonane, ponieważ do określonej w ten sposób krotki reprezentującej pracownika odwołują się krotki w relacji PRACUJE\_NAD. Oznacza to, że usunięcie tej krotki spowodowałoby naruszenie więzi integralności odwołań.

- *Operacja:*

Usuń z relacji PRACOWNIK krotkę z wartością PESEL = '55120834598'.

*Wynik:* To polecenie usunięcia spowodowałoby jeszcze bardziej niebezpieczne naruszenia więzów integralności odwołań, ponieważ przeznaczona do usunięcia krotka relacji PRACOWNIK jest wskazywana przez krotki relacji PRACOWNIK, DZIAŁ, PRACUJE\_NAD oraz CZŁONEK\_RODZINY.

W przypadku naruszenia więzów integralności odwołań przy próbie wykonania operacji usunięcia krotki lub krotek istnieje wiele możliwych rozwiązań. Pierwszym z nich jest prosta *odmowa wykonania operacji usunięcia (ograniczenie)*. Drugim jest tzw. *próba przeniesienia (propagowania) operacji usunięcia* przez usunięcie wszystkich krotek, które odwołują się do krotki lub krotek przeznaczonych do usunięcia w oryginalnej operacji (**kaskadowe przeniesienie**). Przykładowo, w przypadku drugiej z opisanych powyżej sytuacji system zarządzania bazą danych mógłby automatycznie usunąć przeszkadzające w przeprowadzeniu żadanego działania krotki z atrybutem PESELPRAC = '68011932514' z relacji PRACUJE\_NAD. Trzecim rozwiązaniem jest odpowiednie *zmodyfikowanie wartości atrybutu odwołującego*, który jest źródłem naruszenia więzów integralności (**ustawienie wartości pustej lub domyślnej**) — każda z nich może albo mieć przypisaną wartość pustą, albo zostać zmieniona na odwołanie do innej, domyślnej prawidłowej krotki. Warto pamiętać, że jeśli atrybut odwołujący, który powoduje naruszenie więzów integralności, jest *częścią klucza głównego* relacji, w żadnym przypadku *nie można* mu przypisać wartości pustej, ponieważ takie działanie naruszyłoby więzy integralności encji.

Istnieje także możliwość łączenia trzech wymienionych rozwiązań. Przykładowo, aby uniknąć naruszenia więzów integralności w przypadku trzeciej z opisanych operacji, system zarządzania bazą danych może automatycznie usunąć wszystkie te krotki z relacji PRACUJE\_NAD i CZŁONEK\_RODZINY, w których PESELPRAC = '55120834598'. Krotki w relacji PRACOWNIK z wartością atrybutu PESELPRZEŁOŻ równą '55120834598' oraz krotki w relacji DZIAŁ z iden-



tyczną wartością atrybutu PESELKIEROWNIKA mogą mieć zmienione wartości atrybutów PESELPRZEŁOŻ i PESELKIEROWNIKA na inne prawidłowe odwołania lub na wartości puste. O ile automatyczne usuwanie tych krotek z relacji PRACUJE\_NAD i CZŁONEK\_ RODZINY, które odwołują się do danej krotki relacji PRACOWNIK, jest w tym przypadku uzasadnione, o tyle usuwanie odpowiednich krotek z relacji PRACOWNIK i DZIAŁ może być ryzykowne.

Zasadniczo, kiedy więzy integralności odwołań są określone w języku definiowania danych (DDL), system zarządzania bazą danych może zezwalać użytkownikom na *określanie, które z wymienionych rozwiązań* powinno być stosowane w przypadku naruszania tych więzów. Sposób określania dostępnych opcji w języku DDL standardu SQL omówimy w rozdziale 6.

### 5.3.3. Operacja aktualizacji

Operacja **aktualizacji** (lub **modyfikacji**) jest wykorzystywana do zmiany wartości jednego lub więcej atrybutów w krotce (lub krotkach) pewnej relacji *R*. Wykonywanie tego typu operacji wymaga oczywiście określenia warunku dla atrybutów, na podstawie którego zostanie wyselekcjonowana krotka (lub krotki) przeznaczone do zmodyfikowania. Poniżej przedstawiono kilka przykładów:

- *Operacja:*

Aktualizuj atrybut PENSJA krotki relacji PRACOWNIK z wartością PESEL = '68011932514' do wartości 2800.

*Wynik:* Dopuszczalne.

- *Operacja:*

Aktualizuj atrybut NRDZ krotki relacji PRACOWNIK z wartością PESEL = '68011932514' do wartości 1.

*Wynik:* Dopuszczalne.

- *Operacja:*

Aktualizuj atrybut NRDZ krotki relacji PRACOWNIK z wartością PESEL = '68011932514' do wartości 7.

*Wynik:* Polecenie niepoprawne, ponieważ narusza więzy integralności odwołań.

- *Operacja:*

Aktualizuj atrybut PESEL krotki relacji PRACOWNIK z wartością PESEL = '68011932514' do wartości '410620132586'.

*Wynik:* Polecenie niepoprawne, ponieważ narusza własności klucza głównego (z powodu powtórzenia wartości, która już istnieje jako klucz główny w innej krotce) i więzy integralności odwołań (ponieważ występują inne relacje odwołujące się do istniejącej wartości PESEL).



Aktualizacja wartości atrybutu, który nie jest *ani częścią klucza głównego, ani częścią klucza obcego*, zwykle nie powoduje problemów; system zarządzania bazą danych musi jedynie sprawdzić, czy nowa wartość ma właściwy typ danych i należy do określonej wcześniej dziedziny wartości. Modyfikowanie wartości klucza głównego przypomina usunięcie krotki i wstawienie nowej w jej miejsce, ponieważ klucz główny jest wykorzystywany do unikatowego identyfikowania krotek. Oznacza to, że możemy w takim przypadku mieć do czynienia ze wszystkimi utrudnieniami omówionymi w punktach 5.3.1 (wstawianie) i 5.3.2 (usuwanie). Jeśli modyfikacja dotyczy klucza obcego, system zarządzania bazą danych musi się upewnić, że nowa wartość odwołuje się do istniejącej krotki we wskazywanej relacji lub że zmiana wprowadza wartość pustą. Możliwości postępowania w przypadku naruszenia więzów integralności odwołań przez operację aktualizacji są podobne do rozwiązań, które możemy stosować do obsługi naruszeń wynikających z nieprawidłowych operacji usuwania. Kiedy więzy integralności odwołań są definiowane w języku DDL, system zarządzania bazą danych umożliwia użytkownikom wybór osobnych opcji postępowania w przypadku naruszeń generowanych przez operacje usuwania i osobnych rozwiązań dla podobnych sytuacji wynikających z niewłaściwych operacji aktualizacji (patrz podrozdział 6.2).

### 5.3.4. Transakcje

Aplikacja bazy danych używająca bazy relacyjnej zwykle wykonuje *transakcje*. **Transakcja** to wykonywany program, który obejmuje operacje na bazie danych — np.: wczytuje z niej dane, wstawia je, usuwa lub aktualizuje. Transakcja po zakończeniu musi pozostawić bazę w prawidłowym lub spójnym stanie zgodnym ze wszystkimi ograniczeniami określonymi w schemacie bazy. Jedna transakcja może obejmować dowolną liczbę operacji pobierania (omawianych w ramach algebry relacyjnej i rachunku relacyjnego w rozdziale 8. oraz w kontekście języka SQL w rozdziałach 6. i 7.) i aktualizowania. Te operacje wspólnie tworzą atomową jednostkę pracy wykonywanej na bazie. Przykładowo, transakcja wypłaty środków z banku zwykle wczytuje stan konta klienta, sprawdza, czy jest on wystarczający, a następnie aktualizuje rekord o wypłaconą kwotę.

Wiele komercyjnych aplikacji operujących na bazach relacyjnych **w systemach OLTP** wykonuje transakcje z szybkością sięgającą kilkuset na sekundę. Zagadnienia związane z przetwarzaniem transakcji, współbieżnym ich wykonywaniem i przywracaniem stanu po awariach omówimy w rozdziałach od 20. do 22.

## 5.4. Podsumowanie

W tym rozdziale zaprezentowaliśmy zagadnienia związane z modelowaniem, strukturami danych oraz ograniczeniami (wzami) występującymi w relacyjnym modelu danych. Rozpoczęliśmy od wprowadzenia takich pojęć jak dziedziny wartości, atrybuty i krotki. Następnie zdefiniowaliśmy schemat relacji jako listę atrybutów opisujących strukturę relacji. Relacja (nazywana także stanem relacji) jest zbiorem krotek, które składają się na taki schemat.

Istnieje wiele właściwości, które odróżniają relacje od zwykłych tabel czy plików. Pierwszą z nich jest brak uporządkowania krotek w relacji. Druga wiąże się z uporządkowaniem atrybutów w schemacie relacji oraz odpowiednim uporządkowaniem wartości na po-

ziomie poszczególnych krotek. Przedstawiliśmy także alternatywną definicję relacji, nie wymagającą stosowania żadnego z wymienionych dwóch uporządkowań, jednak — dla wygody — w dalszej analizie stosowaliśmy pierwszą definicję, która wymaga określenia kolejności atrybutów i wartości krotek. Zaraz potem omówiliśmy rolę wartości w krotkach i wprowadziliśmy pojęcie wartości pustych, które reprezentują brakujące lub nieznane informacje. Podkreśliliśmy, że w miarę możliwości należy unikać wartości pustych.

W dalszej części tego rozdziału podzieliliśmy ograniczenia (więzy) na oparte na modelu danych, oparte na schemacie (inaczej bezpośrednie) oraz oparte na aplikacji (inaczej semantyczne lub reguły biznesowe). Następnie omówiliśmy ograniczenia schematów relacji ściśle związane z modelem relacyjnym, poczynwszy od ograniczeń dziedziny, przez ograniczenia klucza (włącznie z pojęciami nadklucza, klucza kandydującego oraz klucza głównego), do ograniczeń wartości pustych dla atrybutów. Potem przedstawiliśmy definicje relacyjnych baz danych oraz schematów relacyjnych baz danych. Do dodatkowych ograniczeń modelu relacyjnego należą też więzy integralności encji, które zakazują stosowania wartości pustych dla atrybutów tworzących klucz główny. Następnie opisaliśmy występujące między relacjami więzy integralności odwołań, które są wykorzystywane do zapewniania spójności odwołań pomiędzy krotkami należącymi do różnych relacji.

Operacje modyfikowania danych przechowywanych w modelu relacyjnym to takie działania jak wstawianie, usuwanie i aktualizacja. Każda z takich operacji może powodować naruszenie określonych typów ograniczeń (patrz podrozdział 5.3). Za każdym razem, gdy któraś z tych operacji jest wykonywana, zmieniony w ten sposób stan bazy danych musi prawidłowy. Na koniec przedstawiliśmy transakcje, które są istotne w relacyjnych SZBD, ponieważ umożliwiają połączenie zbioru operacji bazodanowych w jedno atomowe działanie na bazie.

## Pytania powtórkowe

- 5.1. Zdefiniuj następujące pojęcia dotyczące relacyjnego modelu danych: *dziedzina*, *atrybut*, *n-krotka*, *schemat relacji*, *stan relacji*, *stopień relacji*, *schemat relacyjnej bazy danych* oraz *stan relacyjnej bazy danych*.
- 5.2. Dlaczego krotki w relacji nie są uporządkowane?
- 5.3. Dlaczego w jednej relacji nie mogą się znajdować dwie takie same krotki?
- 5.4. Jaka jest różnica pomiędzy kluczem a nadkluczem?
- 5.5. Dlaczego musimy wyznaczać jeden z kluczy kandydujących relacji do roli klucza głównego?
- 5.6. Omów te właściwości relacji, które odróżniają je od zwykłych tabel i plików.
- 5.7. Omów różne uzasadnienia wprowadzania do krotek relacji wartości pustych.
- 5.8. Omów więzy integralności encji i więzy integralności odwołań. Dlaczego oba rodzaje ograniczeń są uważane za ważne?
- 5.9. Zdefiniuj pojęcie *klucza obcego*. Do czego wykorzystuje się klucze obce?
- 5.10. Czym jest transakcja? Czym różni się od operacji aktualizacji?

## Ćwiczenia

- 5.11. Przypuśćmy, że każda z poniższych operacji aktualizacji jest bezpośrednio stosowana dla stanu bazy danych, który został przedstawiony na rysunku 5.6. Omów *wszystkie* więzy integralności naruszone przez poszczególne operacje (jeśli takie naruszenia występują) wraz z różnymi sposobami wymuszania zgodności z tymi więzami.
- a) Wstaw <'Robert', 'Franciszek', 'Słowiński', '52062165483', '1952-06-21', 'ul. Mogileńska 17/32. Poznań', M, 5800, '37111045873', 1> do relacji PRACOWNIK.
  - b) Wstaw <'ProduktA', 4, 'Bydgoszcz', 2> do relacji PROJEKT.
  - c) Wstaw <'Produkcja', 4, '79072106544', '1998-10-01'> do relacji DZIAŁ.
  - d) Wstaw <'72073110039', null, '40.0'> do relacji PRACUJE\_NAD.
  - e) Wstaw <'72073110039', 'Jan', M, '1970-12-12', 'MAŻ'> do relacji CZŁONEK\_RODZINY.
  - f) Usuń z relacji PRACUJE\_NAD krotki z wartością PESELPRAC = '55120834598'.
  - g) Usuń z relacji PRACOWNIK krotki z wartością PESEL = '410620132586'.
  - h) Usuń z relacji PROJEKT krotki z wartością NAZWAPROJ = 'ProduktX'.
  - i) Zmień wartości atrybutów PESELKIEROWNIKA i DATAPRZEJKIEROWNICTWA w relacji DZIAŁ w krotkach z wartością NUMERDZ = 5 odpowiednio na wartości '65010912345' i '2007-10-01'.
  - j) Zmień wartość atrybutu PESELPRZEŁOŻ w relacji PRACOWNIK w krotce z wartością PESEL = '68011932514' na wartość '52062165483'.
  - k) Zmień wartość atrybutu GODZINY w relacji PRACUJE\_NAD w krotce z wartościami PPESEL = '68011932514' i NR\_PROJEKTU = 10 na wartość '5.0'.
- 5.12. Przeanalizuj przedstawiony na rysunku 5.8 schemat relacyjnej bazy danych, który opisuje bazę danych wykorzystywaną przez biuro informacji o lotach w liniach lotniczych. Każdy lot (relacja LOT) jest identyfikowany przez numer lotu (atrybut NUMER) i składa się z jednego lub więcej przelotów (relacja PRZELOT) oznaczonych numerami 1, 2, 3, ... (atrybut NUMER). Każdy przelot ma zaplanowany czas i lotnisko wylotu, czas i lotnisko przylotu oraz wiele egzemplarzy przelotu (relacja EGZEMPLARZE\_PRZELOTÓW) — po jednym dla każdej daty podróży. Dla każdego lotu przechowywane są dodatkowo informacje o opłatach (relacja OPŁATY). Dla każdego egzemplarza przelotu baza danych zawiera informacje o rezerwacjach biletów (relacja REZERWACJA\_MIEJSC). Każda krotka reprezentująca przelot wskazuje na wykorzystany samolot (relacja SAMOLOT) oraz przechowuje informacje o rzeczywistych czasach i lotniskach odlotu i przylotu. Samolot jest identyfikowany przez IDENTYFIKATOR\_SAMOLOTU i należy do konkretnego typu (relacja TYP\_SAMOLOTU). Relacja MOŻE\_LĄDOWAĆ wiąże typy samolotów i lotniska (odpowiednio relacje TYP\_SAMOLOTU i LOTNISKO), na których poszczególne typy samolotów mogą lądować. Każde LOTNISKO jest identyfikowane przez KOD\_LOTNISKA. Rozważ możliwość aktualizacji bazy danych LINIE\_LOTNICZE, która będzie polegała na wpisaniu rezerwacji na konkretny lot lub przelot w dniu wskazanym przez użytkownika.
- a) Przedstaw operacje niezbędne do przeprowadzenia takiej aktualizacji.
  - b) Jakie typy ograniczeń należy sprawdzać podczas wykonywania tych operacji?

- c) Które z tych ograniczeń dotyczą kluczy, które wiążą się z integralnością encji, a które wynikają z więzów integralności odwołań?
- d) Zdefiniuj wszystkie więzy integralności odwołań, które występują w schemacie bazy danych przedstawionym na rysunku 5.8.

**LOTNISKO**

KOD_LOTNISKA	Nazwa	Miasto	Województwo
--------------	-------	--------	-------------

**LOT**

NUMER	LINIE_LOTNICZE	DNI_TYGODNIA
-------	----------------	--------------

**PRZELOT**

NUMER_LOTU	NUMER_PRZELOTU	KOD_LOTNISKA_WYLOTU	PLANOWANY_CZAS_WYLOTU
		KOD_LOTNISKA_PRZYLOTU	PLANOWANY_CZAS_PRZYLOTU

**EGZEMPLARZ\_PRZELOTU**

NUMER_LOTU	NUMER_PRZELOTU	DATA	LICZBA_WOLNYCH_MIEJSC	IDENTYFIKATOR_SAMOLOTU
		KOD_LOTNISKA_WYLOTU	CZAS_WYLOTU	KOD_LOTNISKA_PRZYLOTU
				CZAS_PRZYLOTU

**OPŁATY**

NUMER_LOTU	KOD_OPŁATY	WYSOKOŚĆ	OGRANICZENIA
------------	------------	----------	--------------

**TYP\_SAMOLOTU**

NAZWA_TYPU	MAKSYMALNA_LICZBA_MIEJSC	PRODUCENT
------------	--------------------------	-----------

**MOŻE\_LĄDOWAĆ**

NAZWA_TYPU	KOD_LOTNISKA
------------	--------------

**SAMOLOT**

IDENTYFIKATOR_SAMOLOTU	ŁĄCZNA_LICZBA_MIEJSC	TYP_SAMOLOTU
------------------------	----------------------	--------------

**REZERWACJA\_MIEJSC**

NUMER_LOTU	NUMER_PRZELOTU	DATA	NUMER_MIEJSCA	NAZWISKO_KLIENTA	TELEFON_KLIENTA
------------	----------------	------	---------------	------------------	-----------------

RYSUNEK 5.8. Schemat relacyjnej bazy danych LINIE\_LOTNICZE

- 5.13. Przeanalizuj relację PRZEDMIOT(NrPrzedmiotu, NrWydziału, NazwiskoWykładowcy, Semestr, KodBudynku, NrSalii, CzasTrwania, DniTygodnia, Godziny). Relacja PRZEDMIOT reprezentuje przedmioty nauczane na wyższej uczelni, a dokładnie na określonym wydziale (unikatowo identyfikowanym przez atrybut NrWydziału). Zidentyfikuj atrybuty, które Twoim zdaniem powinny być kluczami kandydującymi, i opisz własnymi słowami ograniczenia, przy których poszczególne klucze kandydujące będą prawidłowe.

- 5.14. Przeanalizuj sześć wymienionych poniżej relacji dla firmowej bazy danych odpowiedzialnej za przetwarzanie zamówień:

KLIENT(NrKlienta, NazwiskoK, Miasto)  
 ZAMÓWIENIE(NrZamówienia, DataZ, NrKlienta, Wartość)  
 ELEMENT\_ZAMÓWIENIA(NrZamówienia, NrElementu, LiczbaJednostek)  
 ELEMENT(NrElementu, CenaJednostki)  
 DOSTAWA(NrZamówienia, NrMagazynu, DataDostawy)  
 MAGAZYN(NrMagazynu, Miasto)

Atrybut *Wartość* relacji *ZAMÓWIENIE* odwołuje się do łącznej (wyrażonej w złotych) wartości zamówienia; atrybut *ZData* reprezentuje datę złożenia zamówienia; atrybut *DataDostawy* reprezentuje dzień, w którym dane zamówienie (lub jego część) zostało wysłane z magazynu. Przyjmijmy, że jedno zamówienie może dotyczyć towarów znajdujących się w wielu magazynach. Zdefiniuj dla tego schematu klucze obce i opisz wszystkie przyjęte przez siebie założenia. Jakie inne ograniczenia potrafisz wymyślić dla tej bazy danych?

- 5.15. Przeanalizuj trzy wymienione poniżej relacje bazy danych (wykorzystywanej przez firmę handlową) zawierającej informacje o podróżach odbywanych przez sprzedawców:

SPRZEDAWCA(PESEL, Nazwisko, RokRozpoczęciaWspółpracy, NrDziału)  
 WYJAZD(PESEL, ZMiasta, DoMiasta, DataPrzybycia, DataPowrotu, NrWyjazdu)  
 WYDATKI(NrWyjazdu, NrKonta, Wartość)

Wydatki mogą obciążać jedno lub więcej kont. Zdefiniuj dla tego schematu klucze obce i opisz wszystkie przyjęte przez siebie założenia.

- 5.16. Przeanalizuj wymienione poniżej relacje należące do bazy danych zawierającej informacje o udziale studentów w zajęciach z poszczególnych przedmiotów oraz o książkach wykorzystywanych w trakcie tych zajęć:

STUDENT(PESEL, Nazwisko, Kierunek, DataUrodzenia)  
 PRZEDMIOT(NrPrzedmiotu, NazwaPrzedmiotu, Wydział)  
 ZAPIS(PESEL, NrPrzedmiotu, Semestr, Ocena)  
 STOSOWANE\_KSIĄŻKI(NrKursu, Semestr, ISBNKsiążki)  
 KSIĄŻKA(ISBNKsiążki, TytułKsiążki, Wydawca, Autor)

Zdefiniuj dla tego schematu klucze obce i opisz wszystkie przyjęte przez siebie założenia.

- 5.17. Przeanalizuj poniższe relacje składające się na bazę danych, która przechowuje informacje na temat sprzedaży samochodów w sieci dilerów (relacja *OPCJE* reprezentuje dodatkowe wyposażenie instalowane i sprzedawane w nowych samochodach):

SAMOCHÓD(NrSeryjny, Model, Producent, Cena)  
 OPCJE(NrSeryjny, NazwaOpcji, Cena)  
 SPRZEDAJE(IdentyfikatorSprzedawcy, NrSeryjny, Data, WartośćTransakcji)  
 SPRZEDAWCA(IdentyfikatorSprzedawcy, Nazwisko, Telefon)

W pierwszej kolejności zdefiniuj dla tego schematu klucze obce i opisz wszystkie przyjęte przez siebie założenia. Następnie wypełnij opisane powyżej relacje kilkoma przykładowymi krotkami i podaj taki przykład operacji wstawiania nowych

krotek do relacji SPRZEDAJE i SPRZEDAWCA, który *naruszy* więzy integralności odwołań, i przykład, który tych więzów *nie* naruszy.

- 5.18. W projekcie bazy danych często trzeba określić, jak przechowywać atrybuty. Przykładowo, numer PESEL można przechowywać jako jeden atrybut lub rozbić go na trzy części XXXXXX-XXXX-X. Zwykle numer PESEL jest przedstawiany jako pojedynczy atrybut. Decyzja o podziale zależy od tego, jak baza będzie używana. Pomyśl o sytuacjach, w jakich podział numeru PESEL może być przydatny.
- 5.19. Zastanów się nad relacją STUDENT w bazie UNIWERSYTET. Relacja ta ma następujące atrybuty: (ImięNazwisko, Pesel, Telefon\_stacjonarny, Adres, Telefon\_komórkowy, Wiek, Średnia\_ocen). Oto możliwa krotka reprezentująca tę relację:

ImięNazwisko	Pesel	Telefon_ stacjonarny	Adres	Telefon_ komórkowy	Wiek	Średnia_ocen
Grzegorz Szymański		12 611 22 33	ul. Długa 123, 31-475 Kraków	696 111 222	19	3,75

- Wskaż istotne brakujące informacje z atrybutów Telefon\_stacjonarny i Telefon\_ ↪ komórkowy. *Wskazówka*: czy możesz zadzwonić do kogoś, kto mieszka w innym państwie?
  - Czy te brakujące informacje umieściłbyś w atrybutach Telefon\_stacjonarny i Telefon\_ ↪ komórkowy, czy w nowych atrybutach schematu STUDENT?
  - Przyjrzyj się atrybutowi Nazwisko. Jakie są wady i zalety podziału tego pola z jednego atrybutu na trzy (pierwsze imię, drugie imię i nazwisko)?
  - Zaproponuj ogólne wskazówki dotyczące decydowania o tym, kiedy zapisywać informacje w jednym atrybucie, a kiedy dzielić dane.
  - Założmy, że student może mieć od 0 do 5 numerów telefonów. Zaproponuj dwa różne projekty umożliwiające zapis tego rodzaju informacji.
- 5.20. Wyobraź sobie, że zmiany w prawie z zakresu prywatności uniemożliwiają firmom stosowanie numerów PESEL do identyfikowania osób (chyba że spełnione są pewne warunki). Wskutek tego większość uniwersytetów nie może korzystać z takich numerów jako kluczy głównych (wyjątkiem są dane finansowe). W praktyce jako klucz główny prawdopodobnie używany będzie unikatowy i przypisywany każdemu studentowi identyfikator Id\_studenta, a nie numer PESEL, ponieważ ten identyfikator można stosować w całym systemie.
- Niektórzy projektanci niechętnie posługują się generowanymi kluczami (nazywanymi też *kluczami sztucznymi*), np. atrybutem Id\_studenta, jako kluczami głównymi, ponieważ są to sztuczne dane. Czy potrafisz zaproponować klucze naturalne, które można wykorzystać do identyfikowania studentów w bazie UNIWERSYTET?
  - Założmy, że możesz zagwarantować unikatowość klucza naturalnego obejmującego nazwisko. Czy masz pewność, że nazwisko nie zmieni się w trakcie użytkowania bazy? Jeśli zmiana nazwiska jest możliwa, jaki sposób tworzenia klucza głównego proponujesz, aby nadal obejmował on nazwisko, a przy tym był unikatowy?
  - Jakie są wady i zalety stosowania generowanych (sztucznych) kluczy?

## Wybrane publikacje

Model relacyjny został wprowadzony przez Codd (1970) w jego klasycznej pracy. W kolejnych pozycjach Codd wprowadził także algebrę relacyjną i przedstawił teoretyczne podstawy dla relacyjnego modelu danych (1971, 1972, 1972a i 1974); niedługo potem Codd otrzymał za swoją pracę nad modelem relacyjnym Nagrodę Turinga, najwyższe odznaczenie przyznawane przez Stowarzyszenie Producentów Maszyn Cyfrowych (ang. *Association for Computing Machinery* — *ACM*). W swojej późniejszej pracy Codd (1979) omówił możliwe rozszerzenia modelu relacyjnego, które w większym stopniu uwzględniały możliwość reprezentowania metadanych i semantyki relacji; zaproponował także logikę trójwartościową wykorzystywaną do obsługi niepewności w relacjach oraz wprowadził do swojej algebry relacyjnej wartości puste. Efektem jego prac był model RM/T. Jeszcze przed publikacjami Codd Childs (1968) zaproponował metody wykorzystywania teorii zbiorów do modelowania baz danych. Później Codd (1990) opublikował książkę poświęconą ponad 300 najważniejszym właściwościom relacyjnego modelu danych i systemów baz danych. Date (2001) zaprezentował retrospektywny przegląd i analizę relacyjnego modelu danych.

Od czasu pierwszych publikacji Codd przeprowadzono wiele badań poświęconych różnym aspektom relacyjnego modelu danych. Todd (1976) opisał eksperymentalny system zarządzania bazą danych nazwany PRTV, który był bezpośrednią implementacją operacji algebry relacyjnej. Schmidt i Swenson (1975) wprowadzili do modelu relacyjnego dodatkową semantykę przez sklasyfikowanie różnych typów relacji. Opracowany przez Chena (1976) model związków encji (patrz rozdział 3.) jest jednym ze środków wzajemnego łączenia (na poziomie koncepcyjnym) semantyk świata rzeczywistego reprezentowanych w relacyjnych bazach danych. Wiederhold i Elmasri (1979) wprowadzili różne typy związków łączących relacje, które pozwoliły rozszerzyć ich ograniczenia. Najważniejsze rozszerzenia relacyjnego modelu danych omówimy w rozdziałach 11. i 26. Informacje o innych przydatnych materiałach poświęconych modelowi relacyjnemu i jego językom, systemom, rozszerzeniom i teorii można znaleźć w punktach „Wybrane publikacje” kończących rozdziały 6. – 9., 14., 15., 23. oraz 30. Maier (1983) oraz Atzeni i De Antonellis (1993) przedstawili rozbudowane teoretyczne omówienie relacyjnego modelu danych.



## Podstawy języka SQL

Język SQL można traktować jako jedno z głównych źródeł popularności relacyjnych baz danych w świecie rozwiązań komercyjnych. Ponieważ język ten stał się standardem w relacyjnych systemach zarządzania bazami danych, z punktu widzenia ich użytkowników pokusa przenoszenia aplikacji baz danych z innych systemów baz danych (przykładowo, z systemów sieciowych lub hierarchicznych) do systemów relacyjnych jest znacznie silniejsza. Wynika to z faktu, że nawet jeśli użytkownicy nie będą usatysfakcjonowani z wykorzystywanego do tej pory relacyjnego systemu zarządzania bazą danych, wdrożenie innego relacyjnego systemu zarządzania bazą danych nie powinno się wiązać ani z nadmiernymi kosztami finansowymi, ani z dużymi opóźnieniami, ponieważ oba systemy będą wykorzystywały te same standardy językowe. W praktyce istnieje oczywiście wiele różnic pomiędzy poszczególnymi komercyjnymi pakietami relacyjnych systemów zarządzania bazami danych. Jeśli jednak użytkownik wykorzystuje w swojej pracy tylko te elementy, które należą do standardu, konwersja pomiędzy dwoma systemami powinna być znacznie prostsza. Inną zaletą stosowania tego standardu jest możliwość pisania takich poleceń w aplikacji bazy danych, które będą mogły uzyskiwać dostęp do informacji zawartych w dwóch lub więcej relacyjnych systemach zarządzania bazami danych bez konieczności zmiany podjęzyka bazy danych (SQL), przynajmniej jeśli oba (lub wszystkie) relacyjne systemy zarządzania bazami danych obsługują standard SQL.

W tym rozdziale prezentujemy główne elementy standardu SQL stosowanego w *komercyjnych* relacyjnych systemach zarządzania bazami danych, natomiast w rozdziale 5. omówiliśmy najważniejsze pojęcia związane z *formalnym* relacyjnym modelem danych, który leży u podstaw tego standardu. W rozdziale 8. (konkretnie w podrozdziałach od 8.1 do 8.5) przedstawiliśmy operacje *algebry relacyjnej*, których znajomość jest bardzo przydatna podczas analizy rozmaitych typów żądań dotyczących informacji zawartych w relacyjnej bazie danych. Rozumienie operacji algebry relacyjnej ma zasadnicze znaczenie także podczas studiowania takich zagadnień jak przetwarzanie czy optymalizacja zapytań w relacyjnych systemach zarządzania bazami danych (przekonamy się o tym w rozdziałach 18. i 19.). Operacje algebry relacyjnej są jednak często uważane za zbyt niskopoziomowe i — tym samym — niełatwe do zrozumienia dla większości użytkowników komercyjnych systemów zarządzania bazami danych, ponieważ zapytanie w algebrze relacyjnej ma postać sekwencji operacji, których wykonanie pozwala wygenerować oczekiwany wynik. Oznacza to, że użytkownik musi określać *sposób (kolejność)* wykonywania operacji składających się na zapytanie. Z drugiej strony, język SQL zapewnia wysokopoziomowy interfejs języka *deklaratywnego*, dzięki czemu użytkownik może określać tylko to, *co* ma być wynikiem danego zapytania — odpowiedzialność za faktyczną optymalizację zapytania oraz za wszelkie decyzje odnośnie do sposobu jego wykonania jest więc przeniesiona na system zarządzania bazą danych. Standard SQL co prawda ma

pewne cechy algebry relacyjnej, ale w znacznie większym stopniu opiera się na *relacyjnym rachunku krotek*, który omawiamy w podrozdziale 8.6. Składnia języka SQL jest jednak bardziej przyjazna dla użytkownika niż reguły składniowe obowiązujące w obu wymienionych językach formalnych.

Nazwa **SQL** pochodzi od ang. *Structured Query Language*, czyli strukturalnego język zapytań. Język SQL był początkowo nazywany językiem *SEQUEL* (ang. *Structured English QUery Language*) i został zaprojektowany i zaimplementowany w IBM Research jako część interfejsu dla eksperymentalnego systemu relacyjnej bazy danych o nazwie SYSTEM R. SQL jest obecnie standardowym językiem wykorzystywanym w komercyjnych relacyjnych systemach zarządzania bazami danych. Dzięki połączonym wysiłkom instytutu ANSI (ang. *American National Standards Institute*) oraz organizacji ISO (ang. *International Standards Organization*) stworzono standardową wersję języka SQL (ANSI 1986) nazywaną SQL-86 lub SQL1. Następnie opracowano poprawioną i znacznie rozszerzoną wersję tego standardu nazywaną SQL-92 (często nazywaną także SQL2). Kolejna wersja standardu była początkowo nazywana standardem SQL3, jednak obecnie częściej spotykaną nazwą jest SQL:1999. Kolejnymi aktualizacjami standardu były wersje SQL:2003 i SQL:2006, w których m.in. dodano mechanizmy języka XML (patrz rozdział 13.). W innej aktualizacji, z 2008 r., do języka SQL dodano więcej mechanizmów obiektowych (patrz rozdział 12.). Następną aktualizacją była wersja SQL:2011. Wszędzie tam, gdzie będzie to możliwe, będziemy się starali skupiać właśnie na najnowszej wersji standardu SQL. Jednak niektóre z nowszych funkcji są opisane w dalszych rozdziałach. Ponadto w tej książce nie jest możliwe omówienie całego języka. Należy zauważyć, że gdy do języka SQL dodawane są nowe funkcje, zwykle wprowadzenie ich w komercyjnych SZBD używających tego języka zajmuje kilka lat.

SQL jest wszechstronnym językiem baz danych — obejmuje polecenia związane z definiowaniem danych, tworzeniem zapytań oraz aktualizacją danych. Oznacza to, że SQL *jednocześnie* pełni rolę języka DDL (definiowania danych) i języka DML (modelowania danych). Standard SQL oferuje dodatkowo funkcje umożliwiające definiowanie perspektyw dla bazy danych, określanie zabezpieczeń i metod uwierzytelniania, definiowanie więzów integralności oraz sterowanie wykonywaniem transakcji. Standard SQL definiuje także reguły osadzania wyrażeń języka SQL w takich uniwersalnych językach programowania jak Java czy C/C++<sup>1</sup>.

Nowsze standardy (począwszy od **SQL:1999**) są podzielone na **rdzenną** specyfikację i osobno zdefiniowane opcjonalne **pakiety** (rozszerzenia). Elementy zdefiniowane w rdzennej specyfikacji w założeniu mają być implementowane przez wszystkich producentów relacyjnych systemów zarządzania bazami danych, którzy chcą zapewnić zgodność swoich produktów ze standardem SQL. Pakiety mogą być implementowane w postaci opcjonalnych modułów sprzedawanych niezależnie z myślą o konkretnych zastosowaniach baz danych, takich jak drążenie danych, dane przestrzenne, dane zależne od czasu, hurtownie danych, przetwarzanie OLAP, dane multimedialne itp.

---

<sup>1</sup> Początkowo język SQL zawierał także polecenia umożliwiające tworzenie i usuwanie indeksów dla plików reprezentujących relacje, jednak możliwość operowania na takich indeksach od jakiegoś czasu nie jest uwzględniana w kolejnych wersjach tego standardu.

Ponieważ standard SQL jest bardzo ważny (i dość obszerny), poświęciliśmy omawianiu jego podstawowych elementów dwa rozdziały tej książki. W pierwszym podrozdziale (6.1) tego rozdziału opiszemy polecenia języka SQL DDL wykorzystywane do tworzenia schematów i tabel oraz zaprezentujemy krótki przegląd podstawowych typów danych obsługiwanych w standardzie SQL. W podrozdziale 6.2 przedstawimy metodę definiowania podstawowych ograniczeń (wiązków), w tym ograniczeń klucza oraz wiązków integralności odwołań. Podrozdział 6.3 zawiera opis podstawowych konstrukcji języka SQL, w oparciu o które definiuje się zapytania pobierające dane. W podrozdziale 6.4 opiszemy polecenia języka SQL wykorzystywane do wstawiania, usuwania i aktualizowania danych.

W rozdziale 7. opiszemy bardziej złożone zapytania języka SQL pobierające dane, a także polecenia `ALTER` służące do modyfikowania schematu. Omówimy też instrukcję `CREATE ASSERTION`, która umożliwia specyfikowanie ogólniejszych ograniczeń bazy danych, i wyzwalacze, szczegółowo przedstawione w rozdziale 26. W rozdziale 7. opiszemy też mechanizm języka SQL przeznaczony do definiowania perspektyw. Perspektywy są nazywane *tabelami wirtualnymi* lub *pochodnymi*, ponieważ prezentują użytkownikom zawartość tabel, przy czym informacje te są określane na podstawie wcześniej zdefiniowanych tabel.

Podrozdział 6.5 zawiera listę niektórych spośród konstrukcji języka SQL zaprezentowanych w pozostałych rozdziałach tej książki — chodzi o takie elementy jak omówione w rozdziale 12. mechanizmy obiektowe, przedstawiony w rozdziale 13. język XML, opisane w rozdziale 20. sterowanie transakcjami, opisane w rozdziale 26. aktywne bazy danych (tzw. wyzwalacze), zawarte w rozdziale 29. techniki obiektowe oraz zaprezentowane w rozdziale 28. metody przetwarzania analitycznego dostępnego bezpośrednio (OLAP) i przedstawione w rozdziale 30. problemy bezpieczeństwa i uwierzytelniania. Podrozdział 6.6 zawiera podsumowanie treści tego rozdziału. W rozdziałach 10. i 11. opiszemy różne techniki programowania baz danych z użyciem języka SQL.

## 6.1. Definicje danych i typy danych języka SQL

Język SQL wykorzystuje terminy **tabeli**, **wiersza** i **kolumny**, zamiast tradycyjnych (stosowanych w formalnym modelu relacyjnym) pojęć — odpowiednio — *relacji*, *krotki* i *atrybutu*. W tym rozdziale będziemy stosowali odpowiadające sobie określenia wymiennie. Podstawowym poleceniem języka SQL wykorzystywanym do definiowania danych jest słowo `CREATE`, które służy do tworzenia schematów, tabel (relacji), typów oraz dziedzin (a także pozostałych konstrukcji standardu SQL takich jak perspektywy, asercje czy wyzwalacze). Zanim przystąpimy do analizy poszczególnych wyrażeń opartych na poleceniu `CREATE`, w punkcie 6.1.1 omówimy niezbędne pojęcia schematu i katalogu, co zapewni niezbędny kontekst. W punkcie 6.1.2 opiszemy sposób tworzenia tabel, natomiast w punkcie 6.1.3 przedstawimy najważniejsze typy danych, które można wykorzystywać podczas definiowania atrybutów. Ponieważ specyfikacja standardu SQL jest bardzo obszerna, zajmiemy się jedynie jej najważniejszymi elementami. Więcej szczegółów można znaleźć w rozmaitych publikacjach i dokumentach standardu SQL (patrz bibliografia na końcu tego rozdziału).

### 6.1.1. Stosowane w języku SQL pojęcia schematu i katalogu

We wczesnych wersjach standardu SQL nie stosowano pojęcia schematu relacyjnej bazy danych; wszystkie tabele (relacje) były traktowane jako część tego samego schematu. Pojęcie schematu SQL wprowadzono dopiero w standardzie SQL2 — taki zabieg był niezbędny do prawidłowego reprezentowania grupowanych tabel i innych konstrukcji należących do tej samej aplikacji bazy danych. **Schemat SQL** jest identyfikowany za pomocą **nazwy schematu** i obejmuje **identyfikator uwierzytelniania**, który wskazuje na użytkownika lub konto występujące w roli właściciela danego schematu, oraz **deskryptory poszczególnych elementów** tego schematu. Do **elementów** schematu należą tabele, typy, ograniczenia (więzy), perspektywy, dziedziny i inne konstrukcje (w tym ewentualne przyznania dostępu), które opisują dany schemat. Schemat jest tworzony za pomocą polecenia `CREATE SCHEMA`, które może zawierać wszystkie niezbędne definicje elementów nowego schematu. Alternatywnym rozwiązaniem jest nadanie nowemu schematowi tylko nazwy oraz identyfikatora uwierzytelniania — elementy schematu można zdefiniować później. Przykładowo, poniższe polecenie tworzy schemat nazwany `FIRMA` i posiadany przez użytkownika z identyfikatorem uwierzytelniania `JNOWAK`. Zauważ, że każda instrukcja języka SQL kończy się średnikiem.

```
CREATE SCHEMA FIRMA AUTHORIZATION 'JNOWAK';
```

Zasadniczo nie wszyscy użytkownicy dysponują uprawnieniami niezbędnymi do tworzenia schematów i należących do nich elementów. Prawo tworzenia schematów, tabel i innych konstrukcji musi być jawnie przypisane do odpowiednich kont użytkowników przez administratora systemowego lub administratora bazy danych.

Poza pojęciem schematu, w standardzie SQL stosuje się także pojęcie **katalogu** — nazwanego zbioru schematów<sup>2</sup>. Instalacje baz danych obejmują zwykle domyślne środowisko i schemat, dlatego gdy użytkownik połączy się z określoną instalacją i zaloguje do niej, będzie mógł bezpośrednio wskazywać tabele i inne konstrukty schematu bez konieczności podawania jego nazwy. Katalog zawsze posiada specjalny schemat nazwany `INFORMATION_SCHEMA`, który zawiera informacje na temat wszystkich pozostałych schematów w katalogu oraz wszystkich deskryptorów elementów w tych schematach. Więzy integralności (np. więzy integralności odwołań) mogą być definiowane pomiędzy relacjami tylko w sytuacji, gdy dotyczą relacji należących do schematów, które znajdują się w jednym katalogu. Schematy wewnątrz tego samego katalogu mogą także współdzielić pewne elementy, np. definicje typów i dziedziny.

### 6.1.2. Polecenie `CREATE TABLE` języka SQL

Polecenie `CREATE TABLE` jest wykorzystywane do definiowania nowej relacji przez podanie jego nazwy oraz określenie jego atrybutów i początkowych ograniczeń (więzów). W pierwszej kolejności definiuje się atrybuty — każdy z nich musi mieć przypisaną nazwę, typ danych określający dziedzinę wartości oraz wszelkie ograniczenia, np. `NOT NULL` (zabronione wartości puste). Ograniczenia klucza, więzy integralności encji oraz więzy integralności odwołań mogą być albo określane wewnątrz polecenia `CREATE TABLE` bezpośrednio za deklara-

---

<sup>2</sup> W języku SQL występuje też pojęcie *klastra* katalogów.

cjami atrybutów, albo dodane później za pomocą polecenia ALTER TABLE (patrz rozdział 7.). Na rysunku 6.1 przedstawiono przykładowe polecenia definiowania danych w języku SQL — w tym przypadku operacja dotyczy schematu relacyjnej bazy danych FIRMA z rysunku 3.7.

```
CREATE TABLE PRACOWNIK (
    IMIĘ                VARCHAR(15)    NOT NULL,
    INICJAŁDRIMIENIA    CHAR,
    NAZWISKO            VARCHAR(15)    NOT NULL,
    PESEL               CHAR(11)       NOT NULL,
    DATAUR             DATE,
    ADRES               VARCHAR(30),
    PŁEĆ                CHAR,
    PENSJA              DECIMAL(10, 2),
    PESELPRZEŁOŻ        CHAR(11),
    NRDZ                INT            NOT NULL,
    PRIMARY KEY (PESEL),
    FOREIGN KEY (PESELPRZEŁOŻ) REFERENCES PRACOWNIK(PESEL),
    FOREIGN KEY (NRDZ) REFERENCES DZIAŁ(NUMERDZ)
);

CREATE TABLE DZIAŁ (
    NAZWADZ             VARCHAR(15)    NOT NULL,
    NUMERDZ             INT            NOT NULL,
    PESELKIEROWNIKA     CHAR(11)       NOT NULL,
    DATAPRZEJKIEROWNICTWA DATE,
    PRIMARY KEY (NUMERDZ),
    UNIQUE (NAZWADZ),
    FOREIGN KEY (PESELKIEROWNIKA) REFERENCES PRACOWNIK(PESEL)
);

CREATE TABLE LOKALIZACJE_DZIAŁÓW (
    NUMERDZ             INT            NOT NULL,
    LOKALIZACJADZ       VARCHAR(15)    NOT NULL,
    PRIMARY KEY (NUMERDZ, LOKALIZACJADZ),
    FOREIGN KEY (NUMERDZ) REFERENCES DZIAŁ(NUMERDZ)
);

CREATE TABLE PROJEKT (
    NAZWAPROJ           VARCHAR(15)    NOT NULL,
    NUMERPROJ           INT            NOT NULL,
    LOKALIZACJAPROJ     VARCHAR(15),
    NR_DZ               INT            NOT NULL,
    PRIMARY KEY (NUMERPROJ),
    UNIQUE (NAZWAPROJ),
    FOREIGN KEY (NR_DZ) REFERENCES DZIAŁ(NUMERDZ)
);

CREATE TABLE PRACUJE_NAD (
    PESELPRAC           CHAR(11)       NOT NULL,
    NR_PROJ             INT            NOT NULL,
```

```

GODZINY                DECIMAL(3, 1) NOT NULL,
PRIMARY KEY (PESELPRAC, NR_PROJ),
FOREIGN KEY (PESELPRAC) REFERENCES PRACOWNIK(PESEL),
FOREIGN KEY (NR_PROJ) REFERENCES PROJEKT(NUMERPROJ)
);

CREATE TABLE CZŁONEK_RODZINY (
    PESELPRAC            CHAR(11)        NOT NULL,
    IMIĘ_CZŁONKA_RODZINY VARCHAR(15)    NOT NULL,
    PŁEĆ                 CHAR,
    DATAUR              DATE,
    STOPIEŃ_POKREWIEŃSTWA VARCHAR(8),
    PRIMARY KEY (PESELPRAC, IMIĘ_CZŁONKA_RODZINY),
    FOREIGN KEY (PESELPRAC) REFERENCES PRACOWNIK(PESEL)
);

```

RYСУNEK 6.1. Polecenia CREATE TABLE wykorzystywane do definiowania danych w języku SQL — definicja schematu bazy danych FIRMA z rysunku 5.7

W środowisku wykonywania poleceń CREATE TABLE zazwyczaj niejawnie określa się schemat SQL, w którym nowe relacje są deklarowane. Alternatywnym rozwiązaniem jest jawne dołączanie nazw schematów do nazw nowotworzonych relacji i oddzielenie obu elementów za pomocą kropki. Przykładowo, wpisując polecenie:

```
CREATE TABLE FIRMA.PRACOWNIK ...
```

zamiast (jak na rysunku 6.1) polecenia w postaci:

```
CREATE TABLE PRACOWNIK ...
```

możemy jawnie określić (a więc niezależnie od domyślnych, niejawnych ustawień), że nowa tabela PRACOWNIK ma być częścią schematu FIRMA.

Relacje zadeklarowane za pomocą poleceń CREATE TABLE są nazywane **tabelami bazowymi** (lub relacjami bazowymi); oznacza to, że tak utworzona relacja wraz ze swoimi krotkami rzeczywiście zostaje utworzona i zapisana w odpowiednim pliku systemu zarządzania bazą danych. Przeciwnieństwem relacji bazowych są **relacje wirtualne**, które można tworzyć za pomocą poleceń CREATE VIEW (patrz rozdział 7.), i które mogą, ale nie muszą odpowiadać fizycznie istniejącym plikom. W standardzie SQL uważa się, że atrybuty tabeli bazowej są *uporządkowane w takiej kolejności, w jakiej zostały zdefiniowane* w poleceniu CREATE TABLE. Okazuje się jednak, że należące do tej samej struktury wiersze (krotki) nie są uporządkowane wewnątrz relacji.

Należy zauważyć, że na rysunku 6.1 występują *klucze obce, które mogą spowodować błędy*, ponieważ zostały podane z użyciem odwołań cyklicznych lub dotyczą nieutworzonej jeszcze tabeli. Na przykład klucz obcy PESELPRZEŁOŻ w tabeli PRACOWNIK tworzy odwołanie cykliczne, ponieważ dotyczy samej tabeli PRACOWNIK. Klucz obcy NRDZ z tabeli PRACOWNIK dotyczy tabeli DZIAŁ, która nie została jeszcze utworzona. Aby poradzić sobie z problemami tego rodzaju, takie ograniczenia można pominąć w pierwotnej instrukcji CREATE TABLE, a następnie dodać za pomocą instrukcji ALTER TABLE (patrz rozdział 7.). Na rysunku 6.1 umieściliśmy wszystkie klucze obce, aby przedstawić kompletny schemat FIRMA w jednym miejscu.

### 6.1.3. Typy danych atrybutów oraz dziedziny wartości w standardzie SQL

Do podstawowych **typów danych** dostępnych dla atrybutów należą typy numeryczne, ciągi znakowe, ciągi bitowe, wartości logiczne, data oraz czas.

- **Numeryczne** typy danych obejmują liczby całkowite o zróżnicowanych rozmiarach (INTEGER lub INT oraz SMALLINT) oraz liczby zmiennoprzecinkowe (rzeczywiste) o zróżnicowanej precyzji (FLOAT lub REAL oraz DOUBLE PRECISION). Liczby formatowane można deklarować za pomocą słów kluczowych DECIMAL(*i*, *j*), DEC(*i*, *j*) lub NUMERIC(*i*, *j*), gdzie *i* oznacza *precyzję*, a więc reprezentuje łączną liczbę cyfr dziesiętnych, natomiast *j* oznacza *skale*, a więc reprezentuje liczbę cyfr po przecinku. Domyślną wartością dla skali jest zero, natomiast domyślna wartość dla precyzji zależy od konkretnej implementacji.
- **Ciągi znaków** mogą być albo stałej długości — CHAR(*n*) lub CHARACTER(*n*), gdzie *n* jest liczbą znaków — albo zmiennej długości — VARCHAR(*n*), CHAR VARYING(*n*) lub CHARACTER VARYING(*n*), gdzie *n* jest maksymalną liczbą znaków. Stałe wartości ciągów znaków są umieszczane w pojedynczych apostrofach i *uwzględniają wielkość liter* (podczas przetwarzania takich wartości wielkie litery są odróżniane od małych)<sup>3</sup>. W przypadku typów znakowych stałej długości ewentualne krótsze wartości są dopełniane do prawego końca znakami pustymi. Przykładowo, jeśli atrybutowi typu CHAR(10) przypiszemy wartość 'Nowak', zawartość tego atrybutu zostanie dopełniona pięcioma znakami pustymi i będzie miała postać 'Nowak '. Znaki puste stanowiące dopełnienie wartości takich atrybutów są zazwyczaj ignorowane podczas porównywania ciągów znaków. W operacjach tego typu zakłada się, że ciągi znaków są uporządkowane zgodnie z kolejnością alfabetyczną (leksykograficzną) — jeśli przyjmiemy, że zgodnie z porządkiem alfabetycznym ciąg znajdów *str1* znajduje się przed ciągiem *str2*, ciąg *str1* będzie uważany za mniejszy od ciągu *str2*<sup>4</sup>. W języku SQL istnieje także binarny operator konkatenacji (zapisywany w postaci pary znaków ||, czyli ||), za pomocą którego można łączyć dwa ciągi znaków. Przykładowo, wynikiem wyrażenia 'abc' || 'XYZ' będzie pojedynczy ciąg w postaci 'abcXYZ'. Inny tekstowy typ danych o zmiennej długości to CHARACTER LARGE OBJECT (CLOB). Umożliwia on tworzenie kolumn o długich wartościach tekstowych, np. w postaci dokumentów. Maksymalną długość wartości typu CLOB można podać w kilobajtach (K), megabajtach (M) lub gigabajtach (G). Na przykład wyrażenie CLOB(20M) oznacza długość maksymalną równą 20 megabajtów.
- **Ciągi bitowe** mogą być albo stałej długości *n* — wówczas są deklarowane ze słowem kluczowym BIT(*n*) — albo zmiennej długości — w takim przypadku do ich deklaracji wykorzystuje się słowo kluczowe BIT VARYING(*n*), gdzie *n* jest maksymalną liczbą bitów. Domyślną wartością *n*, czyli długością ciągów znakowych i ciągów bitowych, jest 1. Stałe ciągi bitowe zapisujemy pomiędzy znakami apostrofu i dodatkowo poprzedzamy literą B, która odróżnia je od ciągów znakowych;

<sup>3</sup> Ta reguła nie dotyczy słów kluczowych samego języka SQL (np. CREATE lub CHAR). W ich przypadku użyta *wielkość liter nie ma znaczenia*, co oznacza, że standard SQL traktuje wielkie i małe litery w słowach kluczowych dokładnie tak samo.

<sup>4</sup> W przypadku znaków niealfabetycznych istnieje sztucznie zdefiniowany porządek.



przykładowo, ciąg bitowy może mieć postać: `B'10101'`<sup>5</sup>. Innym typem ciągów bitowych o zmiennej długości jest `BINARY LARGE OBJECT (BLOB)`. Umożliwia on tworzenie kolumn o długich wartościach binarnych, np. w postaci zdjęć. Maksymalną długość wartości typu `BLOB` (podobnie jak typu `CLOB`) można podać w kilobitach (K), megabitach (M) lub gigabitach (G). Na przykład wyrażenie `BLOB(20G)` oznacza długość maksymalną równą 20 gigabitów.

- **Logiczny** typ danych reprezentuje tradycyjne wartości `TRUE` i `FALSE` (odpowiednio: prawda i fałsz). Z uwagi na istnienie wartości `NULL` (pustej) w języku SQL faktycznie stosowana jest logika trójwartościowa, zatem trzecią wartością logicznego typu danych jest `UNKNOWN` (wartość nieznana). Konieczność obsługi wartości `UNKNOWN` oraz logikę trójwartościową omówimy w rozdziale 7.
- Typ danych `DATE` składa się z dziesięciu pozycji i obejmuje takie elementy jak `YEAR`, `MONTH` i `DAY` (rok, miesiąc i dzień) w formacie `YYYY-MM-DD`. Typ danych `TIME` składa się przynajmniej z ośmiu pozycji i obejmuje takie elementy jak `hour`, `minute` i `second` (reprezentujące odpowiednio godzinę, minuty i sekundy) w formacie `HH:MM:SS`. Implementacje standardu SQL powinny dopuszczać do dalszego przetwarzania wyłącznie te wartości typów danych `DATE` i `TIME`, które mają prawidłowy format. To oznacza, że miesiące powinny mieć numer od 1 do 12, a dni — od 01 do 31; ponadto liczba dni musi być odpowiednia dla danego miesiąca. Dla dat i czasów można stosować operator porównania (`<`) — *wcześniejsza* data jest oznaczana jako mniejsza od późniejszej daty, podobnie traktowane są dane reprezentujące czas. Wartości stałe (literały) zapisuje się w apostrofach poprzedzonych słowem kluczowym `DATE` lub `TIME`; przykładowo, prawidłowe zapisy tego typu stałych mogą mieć postać: `DATE '2014-09-27'` lub `TIME '09:12:47'`. Istnieje dodatkowy typ danych deklarowany ze słowem kluczowym `TIME(i)`, gdzie *i* jest nazywane *precyzją ułamka sekund*, które określa *i*+1 dodatkowych pozycji dla wartości typu `TIME` — jedna pozycja dla dodatkowego separatora wartości dziesiętnych (`.`) oraz *i* pozycji reprezentujących ułamkową część sekundy. Typ danych `TIME WITH TIME ZONE` obejmuje dodatkowe sześć pozycji reprezentujących *przesunięcie* względem standardowej, uniwersalnej strefy czasowej — takie przesunięcie musi należeć do przedziału od +13:00 do -12:59 i jest wyrażane w formacie `HOURS:MINUTES` (godziny:minuty). Jeśli do deklaracji atrybutu nie dołączymy kwalifikatora `WITH TIME ZONE`, domyślną wartością w sesji języka SQL będzie lokalna strefa czasowa.

Poniżej opisujemy kilka dodatkowych typów danych. Lista omówionych tu typów nie jest kompletna. W różnych implementacjach język SQL może obejmować więcej dodatkowych typów danych.

- Typ danych **znacznika czasowego** (słowo kluczowe `TIMESTAMP`) obejmuje zarówno pole `DATE`, jak i pole `TIME`, co najmniej sześć dodatkowych pozycji dla części ułamkowej sekundy (wyrażonej w postaci wartości dziesiętnej) oraz opcjonalny kwalifikator `WITH TIME ZONE`. Wartości stałe są zapisywane w apostrofach i poprzedzane słowem kluczowym `TIMESTAMP`, a data i czas są od siebie oddzielone znakiem pustym; przykładowo, prawidłowy zapis tego typu stałej może mieć postać: `TIMESTAMP '2014-09-27 09:12:47.648302'`.

<sup>5</sup> Ciągi bitowe, których długość jest wielokrotnością liczby 4, mogą być zapisywane także w notacji szesnastkowej — stała ciągu bitowego jest wówczas poprzedzana literą `X`, a każdy znak szesnastkowy reprezentuje 4 bity.

- Kolejnym typem danych powiązany z typami danych DATE, TIME i TIMESTAMP jest INTERVAL. Za pomocą tego typu możemy definiować **przedział czasowy**, czyli *wartość względną*, która może być wykorzystywana do zwiększania lub zmniejszania wartości bezwzględnej reprezentującej czas, datę lub znacznik czasowy. Przedziały czasowe dzieli się na dwie grupy: YEAR/MONTH (rok-miesiąc) oraz DAY/TIME (dzień-czas).

Format typów danych DATE, TIME i TIMESTAMP można traktować jako specjalny rodzaj ciągu znaków. Oznacza to, że generalnie można stałe wartości tych typów wykorzystywać w operacjach porównywania ciągów znaków — wymagane jest jedynie odpowiednie **rzutowanie** (konwersja) do odpowiednich typów znakowych.

Projektant bazy danych ma możliwość bezpośredniego określania typów danych dla poszczególnych atrybutów (jak na rysunku 6.1); alternatywnym rozwiązaniem jest deklarowanie dziedzin wartości i wykorzystywanie ich nazw w kodzie deklarującym atrybuty. W ten sposób można bardzo ułatwić ewentualne zmiany typu danych dla dziedziny wartości wykorzystywanej przez wiele atrybutów w schemacie, a to z kolei upraszcza cały schemat bazy danych. Przykładowo, za pomocą poniższego polecenia możemy stworzyć dziedzinę wartości PESEL\_TYP:

```
CREATE DOMAIN PESEL_TYP AS CHAR(11);
```

Od tej pory możemy wykorzystywać nazwę PESEL\_TYP zamiast typu CHAR(11) z rysunku 6.1 — zmiana może dotyczyć atrybutów PESEL i PESELPRZEŁOŻ tabeli PRACOWNIK, atrybutu PESELKIEROWNIKA tabeli DZIAŁ, atrybutu PESELPRAC tabeli PRACUJE\_NAD oraz atrybutu PESELPRAC tabeli CZŁONEK\_RODZINY. Dziedzina może także mieć przypisaną opcjonalną specyfikację domyślną; służy do tego klauzula DEFAULT, którą omówimy w dalszej części rozdziału (podczas analizy atrybutów). Zauważ, że w niektórych implementacjach języka SQL dziedziny nie są obsługiwane.

W języku SQL dostępne jest też polecenie CREATE TYPE, które służy do tworzenia typów UDT (ang. *user defined type*, czyli typy definiowane przez użytkownika). Takie typy można następnie stosować jako typy danych atrybutów lub do tworzenia tabel. Szczegółowe omówienie instrukcji CREATE TYPE przedstawimy w rozdziale 12., ponieważ często jest ona używana razem z obiektowymi mechanizmami baz danych, wprowadzonymi w nowszych wersjach języka SQL.

## 6.2. Określanie ograniczeń w języku SQL

Opiszemy teraz podstawowe ograniczenia (więzy), które można definiować w języku SQL podczas tworzenia tabel. Możemy w ten sposób określać nie tylko ograniczenia klucza i więzy integralności odwołań, ale także reguły dla dziedzin atrybutów, dotyczące stosowania wartości pustych oraz ograniczenia nakładane na poszczególne krotki wewnątrz tworzonej relacji. Proces definiowania bardziej ogólnych ograniczeń (nazywanych asercjami) omówimy w rozdziale 7.

## 6.2.1. Definiowanie ograniczeń i wartości domyślnych dla atrybutów

Ponieważ standard SQL przewiduje możliwość przypisywania wartości pustych (NULL) do atrybutów, możemy stosować *ograniczenie* NOT NULL, aby wykluczyć taką ewentualność w przypadku konkretnego atrybutu. Takie ograniczenie jest zawsze niejawnie nakładane na atrybuty będące częścią *klucza głównego* danej relacji, ale równie dobrze możemy je definiować dla dowolnych pozostałych atrybutów, które z jakichś powodów nie powinny zawierać wartości pustych (patrz rysunek 6.1).

Projektant bazy danych ma także możliwość definiowania *wartości domyślnych* atrybutów przez dołączanie do ich deklaracji klauzuli DEFAULT <wartość>. Wartość domyślna jest umieszczana w danym atrybucie każdej nowej krotki, dla której nie określono wprost żadnej innej wartości w czasie tworzenia. Rysunek 6.2 ilustruje przykład określania domyślnej wartości reprezentującej kierownika nowego działu oraz domyślnego identyfikatora działu, w którym jest zatrudniany nowy pracownik. Gdybyśmy nie użyli w tych przypadkach klauzul wartości domyślnych, *wartością domyślną* dla wszystkich atrybutów, które nie są objęte ograniczeniem NOT NULL, byłaby wartość pusta (NULL).

Jeszcze innym rodzajem ograniczeń, które można nakładać na atrybuty lub dziedziny wartości, jest ograniczenie wynikające z klauzuli CHECK, którą umieszcza się bezpośrednio za definicją atrybutu lub dziedziny<sup>6</sup>. Przykładowo, przypuśćmy, że numery działów mogą być liczbami całkowitymi z przedziału od 1 do 20 — w takim przypadku powinniśmy zmodyfikować naszą deklarację atrybutu NUMERDZ tabeli DZIAŁ (patrz rysunek 6.1) w następujący sposób:

```
NUMERDZ INT NOT NULL CHECK (NUMERDZ > 0 AND NUMERDZ < 21);
```

Klauzula CHECK może być stosowana także w połączeniu ze wspomnianym już poleceniem CREATE DOMAIN. Przykładowo, do zadeklarowania potrzebnej dziedziny wartości możemy użyć następującego wyrażenia:

```
CREATE DOMAIN D_NUM AS INTEGER  
CHECK (NUM_DZ > 0 AND NUM_DZ < 21);
```

Tak utworzoną dziedzinę wartości NUM\_DZ możemy wykorzystywać w roli typu wszystkich atrybutów odwołujących się do numerów działów (patrz rysunek 6.1), w tym atrybutu NUMERDZ tabeli DZIAŁ, atrybutu NR\_DZ tabeli PROJEKT, atrybutu NRDZ tabeli PRACOWNIK itp.

---

<sup>6</sup> Klauzula CHECK może być wykorzystywana także do innych celów (przekonamy się o tym w dalszej części tego rozdziału).

```

CREATE TABLE PRACOWNIK (
    ...
    NRDZ                INT                NOT NULL    DEFAULT 1,
    CONSTRAINT PRACKG
        PRIMARY KEY (PESEL),
    CONSTRAINT PRACPRZEŁOŻKO
        FOREIGN KEY (PESELPRZEŁOŻ) REFERENCES PRACOWNIK(PESEL)
            ON DELETE SET NULL ON UPDATE CASCADE,
    CONSTRAINT PRACDZIAŁKO
        FOREIGN KEY (NRDZ) REFERENCES DZIAŁ(NUMERDZ)
            ON DELETE SET DEFAULT ON UPDATE CASCADE
);

CREATE TABLE DZIAŁ (
    ...
    PESELKIEROWNIKA     CHAR(11)          NOT NULL    DEFAULT '37111045873',
    ...
    CONSTRAINT DZIAŁKG
        PRIMARY KEY (NUMERDZ),
    CONSTRAINT DZIAŁKW
        UNIQUE (NAZWADZ),
    CONSTRAINT DZIAŁKIERKO
        FOREIGN KEY (PESELKIEROWNIKA) REFERENCES PRACOWNIK(PESEL)
            ON DELETE SET DEFAULT ON UPDATE CASCADE
);

CREATE TABLE LOKALIZACJE_DZIAŁÓW (
    ...
    PRIMARY KEY (NUMERDZ, LOKALIZACJADZ),
    FOREIGN KEY (NUMERDZ) REFERENCES DZIAŁ(NUMERDZ)
        ON DELETE CASCADE ON UPDATE CASCADE
);

```

RYSUNEK 6.2. Przykład ilustrujący sposób określania w języku SQL domyślnych wartości atrybutów i akcji wywoływanych za pośrednictwem odwołań

## 6.2.2. Definiowanie ograniczeń klucza i więzów integralności odwołań

Ponieważ ograniczenia klucza i więzy integralności odwołań są bardzo ważne, przewidziano możliwość stosowania wewnątrz polecenia CREATE TABLE specjalnych klauzul do ich tworzenia. Kilka przykładów ilustrujących specyfikację ograniczeń klucza i więzów integralności

odwołań przedstawiono już na rysunku 6.1<sup>7</sup>. Klauzula `PRIMARY KEY` określa jeden lub więcej atrybutów, które składają się na klucz główny tworzonej relacji (tabeli). Jeśli klucz główny ma postać *pojedynczego* atrybutu, klauzulę `PRIMARY KEY` można umieścić bezpośrednio za deklaracją tego atrybutu. Przykładowo, klucz główny relacji `DZIAŁ` można zadeklarować w sposób następujący (inaczej niż w przykładowej deklaracji z rysunku 6.1):

```
NUMERDZ INT PRIMARY KEY;
```

Klauzula `UNIQUE` określa klucze alternatywne (unikatowe), nazywane też kluczami kandydującymi — odpowiednie przykłady dotyczące tabel `DZIAŁ` i `PROJEKT` zawarto na rysunku 6.1. Klauzulę `UNIQUE` można też zastosować bezpośrednio dla klucza unikatowego, jeśli jest on *pojedynczym* atrybutem:

```
NAZWADZ VARCHAR(15) UNIQUE,
```

Do definiowania więzów integralności odwołań służy klauzula `FOREIGN KEY` (patrz rysunek 6.1). Zgodnie z wnioskami, do jakich doszliśmy w punkcie 5.2.4, więzy integralności odwołań mogą być naruszane nie tylko podczas wstawiania i usuwania krotek, ale także w trakcie modyfikowania wartości atrybutu klucza obcego lub klucza głównego. Domyślnym działaniem podejmowanym przez język SQL w przypadku naruszenia więzów integralności jest **odmowa** wykonania operacji aktualizacji, która takie naruszenie powoduje (opcja `RESTRICT`). Projektant schematu bazy danych może jednak określić alternatywne kroki realizowane w przypadku ewentualnych naruszeń więzów integralności odwołań przez dołączenie klauzuli **akcji wywoływanych za pośrednictwem odwołań** do dowolnych wybranych przez niego ograniczeń klucza obcego. Służą do tego klauzule `SET NULL`, `CASCADE` oraz `SET DEFAULT`. Każda z tych opcji musi być opatrzona kwalifikatorem `ON DELETE` lub `ON UPDATE`. Przykłady zastosowania tych klauzul wraz z kwalifikatorami zawarto na rysunku 6.2. Warto zauważyć, że w przedstawionym przypadku projektant bazy danych wybrał klauzule `ON DELETE SET NULL` oraz `ON UPDATE CASCADE` dla klucza obcego `PESELPRZEŁOŻ` tabeli (relacji) `PRACOWNIK`. Oznacza to, że jeśli krotka reprezentująca *pracownika pełniącego rolę przełożonego* zostanie *usunięta*, do atrybutu `PESELPRZEŁOŻ` automatycznie zostanie przypisana wartość pusta (`NULL`) w tych krotkach reprezentujących pracowników, które odwołują się do usuniętej krotki pracownika. Z drugiej strony, jeśli zostanie *zaktualizowana* wartość `PESEL` pracownika występującego w roli przełożonego (np. z powodu wykrycia błędnego wpisu w bazie danych), nowa wartość zostanie przekazana *kaskadowo* do atrybutu `PESELPRZEŁOŻ` tych wszystkich krotek reprezentujących pracowników, które odwołują się do zaktualizowanej krotki pracownika<sup>8</sup>.

Działania podejmowane przez system zarządzania bazą danych podczas realizacji klauzul `SET NULL` i `SET DEFAULT` są w zasadzie takie same dla obu kwalifikatorów (`ON DELETE` i `ON UPDATE`) — wartość atrybutów odwołujących się do usuniętych lub zmienionych atrybutów jest zmieniana na wartość pustą `NULL` w przypadku użycia klauzuli `SET NULL` lub na wartość domyślną w przypadku użycia klauzuli `SET DEFAULT`. Działanie wynikające z za-

<sup>7</sup> Definiowanie ograniczeń klucza i więzów integralności odwołań nie było możliwe we wczesnych wersjach standardu SQL. W niektórych implementacjach klucze relacji (tabel) były wskazywane niejawnie na wewnętrznym poziomie, za pomocą polecenia `CREATE INDEX`.

<sup>8</sup> Zauważ, że klucz obcy `PESELPRZEŁOŻ` w tabeli `PRACOWNIK` to odwołanie cykliczne, dlatego konieczne może być dodanie go później jako ograniczenia nazwanego za pomocą instrukcji `ALTER TABLE` (patrz końcówka punktu 6.1.2).

stosowania klauzuli `CASCADE ON DELETE` polega na usunięciu wszystkich krotek zawierających odwołania do usuniętej krotki, natomiast w przypadku użycia klauzuli `CASCADE ON UPDATE` odpowiedzią na zmianę wartości klucza głównego jest odpowiednia aktualizacja wskazujących na niego kluczy obcych (przekazanie nowej wartości zmienionego klucza głównego do tych kluczy obcych). Za wybór właściwych działań i ich poprawne zdefiniowanie w schemacie bazy danych zawsze odpowiada jej projektant. Można sformułować ogólną zasadę, zgodnie z którą opcję `CASCADE` stosuje się do obsługi relacji reprezentujących związkę (tzw. relacje związków, patrz podrozdział 9.1), np. `PRACUJE_NAD`; w przypadku relacji reprezentujących atrybuty wielowartościowe (np. `LOKALIZACJE_DZIAŁÓW`) oraz do modelowania relacji, które reprezentują słabe typy encji (np. `CZŁONEK_RODZINY`).

### 6.2.3. Nadawanie nazw definiowanym ograniczeniom

Przykładowy kod zawarty na rysunku 6.2 ilustruje także sposób, w jaki — bezpośrednio za słowem kluczowym `CONSTRAINT` — można definiować **nazwy ograniczeń**. Nazwy wszystkich ograniczeń (więzów) wewnątrz pojedynczego schematu muszą być unikatowe. Tak zdefiniowane nazwy mogą być wykorzystywane do identyfikowania poszczególnych ograniczeń w sytuacjach konieczności usuwania ograniczeń lub ich zastępowania innymi ograniczeniami (patrz rozdział 7.). Nadawanie ograniczeniom unikatowych nazw jest działaniem opcjonalnym. Można też tymczasowo *odroczyć* sprawdzanie ograniczeń do czasu zakończenia transakcji. Wyjaśnimy to w rozdziale 20. w ramach omawiania transakcji.

### 6.2.4. Stosowanie klauzuli CHECK do określania ograniczeń dla krotek

Poza ograniczeniami klucza i więzami integralności odwołań, które definiuje się przez umieszczanie specjalnych słów kluczowych, projektant bazy danych może wykorzystywać dodatkowe klauzule `CHECK` (dopisywane na końcu polecenia `CREATE TABLE`) do określania pozostałych *ograniczeń tabeli*. Tego typu konstrukcje można nazywać **ograniczeniami krotek**, ponieważ są one stosowane *osobno* dla pojedynczych krotek i podlegają weryfikacji przy każdej próbie wstawienia lub zmodyfikowania krotki. Przykładowo, przypuśćmy, że zdefiniowana za pomocą kodu z rysunku 6.1 tabela `DZIAŁ` zawiera dodatkowy atrybut `DATA_UTWORZENIA_DZIAŁU`, który reprezentuje datę utworzenia danego działu. Moglibyśmy wówczas dodać następującą klauzulę `CHECK` na koniec tak zmodyfikowanego polecenia `CREATE TABLE` (tworzącego tabelę `DZIAŁ`), aby upewnić się, że wprowadzona data rozpoczęcia kierowania działem przez aktualnego kierownika jest późniejsza w porównaniu z datą utworzenia tego działu:

```
CHECK (DATA_UTWORZENIA_DZIAŁU < DATAPRZEJKIEROWNICTWA);
```

Klauzula `CHECK` może być stosowana także do definiowania (w oparciu o polecenie `CREATE ASSERTION`) bardziej ogólnych ograniczeń języka SQL. Tego typu ograniczenia omówimy w rozdziale 7., ponieważ ich zrozumienie wymaga pełnej wiedzy na temat zapytań, którym poświęcimy podrozdziały 6.3 i 7.1.

## 6.3. Podstawowe zapytania języka SQL

Język SQL oferuje jedno podstawowe polecenie przeznaczone do wyszukiwania informacji w bazie danych — popularne polecenie SELECT. Polecenie to *nie jest związane* z należącą do algebry relacyjnej operacją selekcji, którą omawiamy w rozdziale 8. W języku SQL istnieje bardzo wiele zróżnicowanych opcji i wersji polecenia SELECT, zatem będziemy wprowadzali jego możliwości stopniowo. W tym rozdziale nie tylko będziemy wykorzystywali przykładowe zapytania zdefiniowane dla schematu przedstawionego na rysunku 5.5, ale także będziemy się odwoływali do przykładowego stanu bazy danych z rysunku 5.6, co umożliwi nam analizę otrzymywanych wyników niektórych spośród omawianych zapytań. W tym podrozdziale prezentujemy funkcje języka SQL używane w *prostych zapytaniach pobierających dane*. Mechanizmy języka SQL służące do tworzenia bardziej złożonych zapytań tego rodzaju zaprezentujemy w podrozdziale 7.1.

Zanim przystąpimy do omawiania konkretnych zapytań, musimy podkreślić *ważną różnicę* pomiędzy językiem SQL a przedstawionym w rozdziale 5. formalnym modelem relacyjnym. W języku SQL tabela (relacja) może zawierać dwie lub więcej krotek mających identyczne wartości we wszystkich swoich atrybutach. Oznacza to, że tabela języka **SQL** nie jest *zbiorem krotek*, ponieważ żaden zbiór nie może zawierać dwóch identycznych elementów; tabela języka SQL jest więc **wielozbiorem** (ang. *multiset*, *bag*) krotek. Niektóre relacje języka SQL podlegają *ograniczeniu wymuszającemu bycie zbiorem*, albo z uwagi na zadeklarowane ograniczenie klucza, albo z powodu zastosowania w poleceniu SELECT opcji DISTINCT (omówionej w dalszej części tego podrozdziału). Podczas analizy prezentowanych w tym podrozdziale przykładów powinniśmy mieć na uwadze to istotne rozróżnienie.

### 6.3.1. Struktura podstawowych zapytań języka SQL: SELECT-FROM-WHERE

Zapytania definiowane w języku SQL mogą być bardzo skomplikowane. Nasze rozważania rozpoczniemy oczywiście od najprostszych zapytań, następnie będziemy stopniowo (krok po kroku) analizowali coraz bardziej skomplikowane przykłady. Podstawowa postać polecenia SELECT (nazywana czasami **odwzorowaniem** lub **blokiem** select-from-where) składa się z trójki klauzul SELECT, FROM oraz WHERE i ma następujący format<sup>9</sup>:

```
SELECT <lista atrybutów>  
FROM   <lista tabel>  
WHERE  <warunek>;
```

gdzie:

- *<lista atrybutów>* jest listą nazw atrybutów, których wartości mają być wyszukane podczas wykonywania danego zapytania;
- *<lista tabel>* jest listą nazw relacji niezbędnych do przetworzenia danego zapytania;
- *<warunek>* jest (logicznym) wyrażeniem warunkowym identyfikującym krotki, które mają być wyszukane podczas wykonywania danego zapytania.

---

<sup>9</sup> Klauzule SELECT i FROM są niezbędne we wszystkich zapytaniach w języku SQL. Klauzula WHERE jest opcjonalna (patrz punkt 6.3.3).



Do podstawowych logicznych operatorów porównania wykorzystywanych w języku SQL do porównywania ze sobą wartości tych samych lub różnych atrybutów, a także wartości stałych, należą: =, <, <=, >, >= oraz <>. Wymienione operatory odpowiadają następującym operatorom algebry relacyjnej (odpowiednio): =, <, ≤, >, ≥ oraz ≠ i następującym operatorom stosowanym w językach programowania C/C++: =, <, <=, >, >= oraz !=. Jak nietrudno zauważyć, główna różnica dotyczy operatora *różne*. Język SQL oferuje także wiele dodatkowych operatorów porównania, które będziemy wprowadzać stopniowo.

Przeanalizujemy teraz kilka prostych, zdefiniowanych w języku SQL poleceń SELECT z odpowiednimi przykładami zapytań. Dla ułatwienia opatrujemy kolejne zapytania taką samą numeracją, jaką stosujemy w rozdziale 8.

**Zapytanie 0.** Pobierz datę urodzenia i adres pracownika (lub pracowników), który nazywa się 'Jan B. Nowak'.

```
Z0:  SELECT DATAUR, ADRES
      FROM  PRACOWNIK
      WHERE IMIĘ = 'Jan' AND INICJAŁDRIMIONA = 'B' AND NAZWISKO = 'Nowak';
```

Wykonanie powyższego zapytania wymaga dostępu tylko do relacji PRACOWNIK (wymienionej wewnątrz klauzuli FROM). Zapytanie to *wybiera (selekcja)* te krotki relacji PRACOWNIK, które spełniają warunek zdefiniowany w klauzuli WHERE, po czym *rzutuje (projekcja)* uzyskany wynik do wymienionych w klauzuli SELECT atrybutów DATAUR i ADRES.

Stosowana w języku SQL klauzula SELECT określa atrybuty, których wartości mają zostać pobrane. W algebrze relacyjnej są to *atrybuty projekcji* (patrz rozdział 8.). Klauzula WHERE określa warunek logiczny, który musi być spełniony dla każdej pobieranej krotki. W algebrze relacyjnej jest to **warunek selekcji**. Wynik wykonania zapytania Z0 dla stanu bazy danych z rysunku 5.6 przedstawiono na rysunku 6.3(a).

Możemy wobec tego przyjąć, że niejawna **zmienna krotki (iterator)** w zapytaniu języka SQL obejmuje każdą pojedynczą krotkę przechowywaną w tabeli PRACOWNIK i podlega w *pętli* weryfikacji pod kątem zgodności z warunkiem zdefiniowanym w klauzuli WHERE. Do relacji wynikowej (rezultatu zapytania) kwalifikowane są tylko krotki spełniające ten warunek — a więc krotki, dla których warunek ten jest prawdziwy (po zastąpieniu nazw atrybutów odpowiednimi wartościami).

**Zapytanie 1.** Pobierz imiona i nazwiska oraz adresy wszystkich pracowników zatrudnionych w dziale badawczym.

```
Z1:  SELECT IMIĘ, NAZWISKO, ADRES
      FROM  PRACOWNIK, DZIAŁ
      WHERE NAZWADZ = 'Badania' AND NUMERDZ = NRDZ;
```

W klauzuli WHERE zapytania Z1 zdefiniowano warunek NAZWADZ = 'Badania', który jest w tym przypadku **warunkiem selekcji**, powodującym pobranie konkretnej krotki z tabeli DZIAŁ, ponieważ NAZWADZ jest atrybutem tej tabeli. Użyty w tej samej klauzuli warunek NUMERDZ = NRDZ pełni rolę **warunku złączenia**, ponieważ łączy dwie krotki: jedną z tabeli DZIAŁ i jedną z tabeli PRACOWNIK, dla których wartość atrybutu NUMERDZ z tabeli DZIAŁ jest równa wartości atrybutu NRDZ z tabeli PRACOWNIK. Wynik zastosowania zapytania Z1 przedstawiono na rysunku 6.3(b). Pojedyncze zapytanie języka SQL może w zasadzie składać się z dowolnej liczby warunków selekcji i złączenia.

(a)

DATAUR	ADRES
1965-01-09	ul. Kręta 4/3, Szczecin

(b)

IMIĘ	NAZWISKO	ADRES
Jan	Szewczyk	ul. Kręta 4/3, Szczecin
Franciszek	Wieszczycki	os. T. Kościuszki 4a/11, Szczecin
Robert	Napierski	ul. Podgórna 7, Police
Joanna	Englert	ul. Wielka 40, Szczecin

(c)

Pnumber	NRDZ	NAZWISKO	ADRES	DATAUR
10	4	Wojtczak	ul. Kwiatowa 78, Mosina	1941-06-20
30	4	Wojtczak	ul. Kwiatowa 78, Mosina	1941-06-20

(d)

E.IMIĘ	E.NAZWISKO	P.IMIĘ	NAZWISKO
Jan	Szewczyk	Franciszek	Wieszczycki
Franciszek	Wieszczycki	Józef	Bąk
Alicja	Zalewska	Janina	Wojtczak
Janina	Wojtczak	Józef	Bąk
Robert	Napierski	Franciszek	Wieszczycki
Joanna	Englert	Franciszek	Wieszczycki
Albert	Janiszewski	Janina	Wojtczak

(e)

PESEL
65010912345
55120834598
68011932514
41062013258
62091502054
72073110039
69032923149
37111045873

(f)

PESEL	NAZWADZ
65010912345	Badania
55120834598	Badania
68011932514	Badania
41062013258	Badania
62091502054	Badania
72073110039	Badania
69032923149	Badania
37111045873	Badania
65010912345	Administracja
55120834598	Administracja
68011932514	Administracja
41062013258	Administracja
62091502054	Administracja
72073110039	Administracja
69032923149	Administracja
37111045873	Administracja
65010912345	Centrala
55120834598	Centrala
68011932514	Centrala
41062013258	Centrala
62091502054	Centrala
72073110039	Centrala
69032923149	Centrala
37111045873	Centrala

(g)

IMIĘ	INICJAŁDRIMIENIA	NAZWISKO	PESEL	DATAUR	ADRES	PŁEĆ	PENSJA	PESELPRZEŁOŻ	NRDZ
Jan	B	Szewczyk	65010912345	1965-09-01	ul. Kręta 4/3, Szczecin	M	3000	55120834598	5
Franciszek	T	Wieszczycki	55120834598	1955-12-08	os. T. Kościuszki 4a/11, Szczecin	M	4000	37111045873	5
Robert	K	Napierski	62091502054	1962-09-15	ul. Podgórna 7, Police	M	3800	55120834598	5
Joanna	A	Englert	72073110039	1972-07-31	ul. Wielka 40, Szczecin	K	2500	55120834598	5

RYSUNEK 6.3. Wyniki wykonania zapytań języka SQL na bazie danych FIRMA znajdującej się w stanie przedstawionym na rysunku 5.6: (a) Z0, (b) Z1, (c) Z2, (d) Z8, (e) Z9, (f) Z10, (g) Z1C

Zapytanie, które obejmuje tylko selekcję, warunki złączenia i atrybuty projekcji, to zapytanie **selekcji-projekcji-łączenia**. Naszym kolejnym przykładem będzie zapytanie selekcji-projekcji-łączenia z *dwoma* warunkami złączenia.

**Zapytanie 2.** Dla każdego projektu realizowanego w Gliwicach znajdź numer projektu, numer działu nadzorującego oraz nazwisko, adres i datę urodzenia kierownika zarządzającego danym działem.

```
Z2:  SELECT NUMERPROJ, NR_DZ, NAZWISKO, ADRES, DATAUR
      FROM PROJEKT, DZIAŁ, PRACOWNIK
      WHERE NR_DZ = NUMERDZ AND PESELKIEROWNIKA = PESEL AND
            LOKALIZACJAPROJ = 'Gliwice';
```

Warunek złączenia `NR_DZ = NUMERDZ` wiąże projekt z jego działem nadzorującym, natomiast warunek złączenia `PESELKIEROWNIKA = PESEL` wiąże dział nadzorujący z pracownikiem, który tym działem kieruje. Każda krotka w wyniku to *połączenie* jednego projektu, jednego działu (nadzorującego projekt) i jednego pracownika (kierującego tym działem). Atrybuty projekcji są używane do wyboru wyświetlanych atrybutów każdej połączonej krotki. Wynik wykonania zapytania Z2 przedstawiono na rysunku 6.3(c).

## 6.3.2. Niejednoznaczne nazwy atrybutów, mechanizm nazw zastępczych (aliasów) oraz zmienne krotek

W języku SQL ta sama nazwa może być stosowana dla dwóch (lub więcej) atrybutów, dopóki atrybuty te należą do *różnych tabel*. Jeśli mamy do czynienia z takim przypadkiem i zapytanie odwołuje się do dwóch lub większej liczby atrybutów z taką samą nazwą, *musimy kwalifikować* (oznaczyć **kwalifikatorem**) nazwę atrybutu przez dodanie nazwy odpowiedniej relacji, aby w ten sposób uniknąć wieloznaczności odwołania. W tym celu powinniśmy *poprzedzić* nazwę atrybutu nazwą interesującej nas relacji i oddzielić oba elementy kropką. Aby zilustrować taki przypadek, przypuśćmy, że schemat i stan bazy danych z rysunków 5.5 i 5.6 zostały zmodyfikowane w następujący sposób: atrybuty `NRDZ` i `NAZWISKO` relacji `PRACOWNIK` zostały nazwane `NUMERDZ` i `NAZW`, a atrybut `NAZWADZ` relacji `DZIAŁ` też został nazwany `NAZW`. W takim przypadku zachowanie jednoznaczności wymagałoby zmiany przedstawionego w poprzednim punkcie zapytania Z1 (do postaci zapytania Z1A). W zmodyfikowanym zapytaniu musimy poprzedzić nazwy atrybutów `NAZW` i `NUMERDZ` odpowiednimi prefiksami, aby jednoznacznie określić, do których atrybutów chcemy się odwoływać (ponieważ takie same nazwy występują w więcej niż jednej relacji):

```
Z1A: SELECT IMIĘ, PRACOWNIK.NAZW, ADRES
      FROM PRACOWNIK, DZIAŁ
      WHERE DZIAŁ.NAZW = 'Badania' AND
            DZIAŁ.NUMERDZ = PRACOWNIK.NUMERDZ;
```

Kwalifikowane nazwy atrybutów można stosować dla przejrzystości nawet wtedy, gdy nazwy atrybutów są jednoznaczne. Zapytanie Z1 można zmodyfikować do postaci Z1' z użyciem kwalifikowanych nazw atrybutów. Można też zmienić nazwy tabel na krótsze, tworząc dla każdej z nich *alias*. Pozwala to uniknąć wielokrotnego wprowadzania długich nazw tabel (patrz zapytanie Z8).

```
Z1b: SELECT PRACOWNIK.IMIĘ, PRACOWNIK.NAZWISKO,
           PRACOWNIK.ADRES
      FROM PRACOWNIK, DZIAŁ
      WHERE DZIAŁ.NAZWADZ = 'Badania' AND
            DZIAŁ.NUMERDZ = PRACOWNIK.NRDZ;
```

Niejednoznaczność może się pojawiać także w przypadku zapytań, które dwukrotnie odwołują się do tej samej relacji (jak w poniższym przykładzie).

**Zapytanie 8.** Dla każdego pracownika znajdź jego imię i nazwisko oraz imię i nazwisko jego bezpośredniego przełożonego.

```
Z8: SELECT E.IMIĘ, E.NAZWISKO, P.IMIĘ, P.NAZWISKO
      FROM PRACOWNIK AS E, PRACOWNIK AS P
      WHERE E.PESELPRZEŁOŻ = P.PESEL;
```

W tym i podobnych przypadkach konieczne jest zadeklarowanie alternatywnych nazw relacji E i P (nazywanych **aliasami** lub **zmiennymi krotek**) dla tej samej relacji (tabeli) PRACOWNIK. Alias może być albo poprzedzony słowem kluczowym AS (jak w powyższym przykładowym zapytaniu Z8), albo występować bezpośrednio za nazwą relacji — przykładowo, równie dobrze możemy w klauzuli FROM zapytania Z8 zadeklarować niezbędne aliasy w postaci: PRACOWNIK E, PRACOWNIK P. Język SQL przewiduje także możliwość zmiany nazw atrybutów relacji (nadania tym atrybutom aliasów) w ramach samego zapytania. Przykładowo, gdybyśmy w klauzuli FROM zapisali odwołanie do relacji PRACOWNIK w następującej postaci:

```
PRACOWNIK AS P(I, IDI, N, PESEL, DU, AD, PŁ, PEN, PESELP, NRDZ)
```

P stałoby się aliasem atrybutu IMIĘ, IDI stałoby się aliasem atrybutu INICJAŁDRIMIENTA, N byłoby aliasem atrybutu NAZWISKO, itd.

Zadeklarowane w zapytaniu Z8 aliasy E i P możemy traktować jak *dwie różne kopie* relacji PRACOWNIK; pierwsza z nich (E) reprezentuje pracowników występujących w roli podwładnych, natomiast druga (P) — pracowników występujących w roli przełożonych. Możemy teraz połączyć obie kopie. W rzeczywistości istnieje oczywiście *tylko jedna* relacja PRACOWNIK, a warunek złączenia ma na celu złączenie relacji z nią samą przez dopasowanie krotek spełniających warunek złączenia w postaci E.PESELPRZEŁOŻ = P.PESEL. Łatwo zauważyć, że przedstawiony przykład jest jednopoziomowym zapytaniem rekursywnym (patrz omówienie tego zagadnienia w punkcie 8.4.2). We wcześniejszych wersjach standardu SQL definiowanie w postaci pojedynczych wyrażeń ogólnych zapytań rekursywnych z nieznaną liczbą poziomów nie było możliwe. Odpowiednią konstrukcję dla zapytań rekursywnych wprowadzono dopiero w standardzie SQL:1999 (patrz rozdział 7.).

Wynik wykonania zapytania Z8 przedstawiono na rysunku 6.3(d). Za każdym razem, gdy pojedynczej relacji przypisujemy jeden lub więcej aliasów, możemy wykorzystywać te nazwy zastępcze do reprezentowania różnych odwołań do tej samej relacji. Oznacza to, że wewnątrz pojedynczego zapytania mamy możliwość stosowania wielu odwołań do jednej relacji.

Nietrudno zauważyć, że jeśli uznamy takie działania za uzasadnione, będziemy mogli wykorzystywać mechanizm **przypisywania nazw zastępczych** (aliasów) we wszystkich tworzonych zapytaniach języka SQL do definiowania zmiennych krotek dla każdej z tabel wymienionych w ramach klauzuli WHERE (nawet niezależnie od tego, czy do tej samej relacji będziemy się odwoływali więcej niż raz). W praktyce takie postępowanie jest zalecane, ponieważ w efekcie nadawania aliasów definicje zapytań często są bardziej zrozumiałe dla osób, które muszą je analizować. Przykładowo, moglibyśmy przekształcić zapytanie Z1A w następujące zapytanie Z1B:

```
Z1B: SELECT E.IMIĘ, E.NAZWISKO, E.ADRES
FROM PRACOWNIK AS E, DZIAŁ AS D
WHERE D.NAZWADZ = 'Badania' AND D.NUMERDZ = E.NRDZ;
```

### 6.3.3. Nieokreślona klauzula WHERE i zastosowania symbolu gwiazdki

W tym punkcie omówimy dwie kolejne interesujące własności języka SQL. *Brak* klauzuli WHERE oznacza, że twórca zapytania nie określił żadnego warunku dla procesu doboru (selekcji) krotek; oznacza to, że do wyniku zapytania zostaną zakwalifikowane *wszystkie* krotki z relacji wskazanej w klauzuli FROM. Jeśli w klauzuli FROM wymieniono więcej niż jedną klauzulę i zapytanie nie zostało opatrzone klauzulą WHERE, wówczas wynikiem zapytania jest ILOCZYN KARTEZJAŃSKI (a więc *wszystkie możliwe kombinacje krotek*) tych relacji. Przykładowo, zapytanie 9. selekcjonuje wszystkie numery PESEL (wartości atrybutu PESEL) przechowywane w relacji PRACOWNIK (patrz rysunek 6.3(e)), natomiast zapytanie 10. znajduje wszystkie kombinacje wartości atrybutu PESEL z relacji PRACOWNIK i atrybutu NAZWADZ z relacji DZIAŁ niezależnie od tego, czy dany pracownik jest zatrudniony w określonym dziale (patrz rysunek 6.3(f)).

**Zapytania 9. i 10.** Z9: Znajdź w bazie danych wszystkie numery PESEL pracowników (wartości atrybutu PESEL relacji PRACOWNIK). Z10: Znajdź w bazie danych wszystkie kombinacje numerów PESEL i nazw działów firmy (wartości atrybutów PESEL i NAZWADZ odpowiednio relacji PRACOWNIK i DZIAŁ).

```
Z9:  SELECT PESEL
      FROM  PRACOWNIK;
```

```
Z10: SELECT PESEL, NAZWADZ
      FROM  PRACOWNIK, DZIAŁ;
```

Określenie w klauzuli WHERE wszystkich niezbędnych warunków selekcji i złączenia jest niezwykle ważne; jeśli którekolwiek z takich ograniczeń zostanie omyłkowo pominięte w procesie definiowania zapytania, wynikiem jego wykonania będą niepoprawne i bardzo duże relacje. Łatwo zauważyć, że zapytanie Z10 jest podobne do operacji projekcji poprzedzonej operacją iloczynu kartezjańskiego (obie operacje należą do algebry relacyjnej; patrz rozdział 8.). Gdybyśmy w zapytaniu Z10 wymienili wszystkie atrybuty relacji PRACOWNIK i DZIAŁ, w rzeczywistości otrzymalibyśmy iloczyn kartezjański (oczywiście z wyjątkiem eliminacji ewentualnych powtórzeń).

Aby po wykonaniu zapytania języka SQL uzyskać wartości wszystkich atrybutów wybranych krotek, nie musimy jawnie wymieniać ich nazw — wystarczy, że użyjemy symbolu *gwiazdki* (\*), która oznacza w tym języku *wszystkie atrybuty*. Gwiazdkę (\*) można też poprzedzić nazwą lub aliasem relacji. Przykładowo, wyrażenie PRACOWNIK.\* oznacza wszystkie atrybuty z tabeli PRACOWNIK.

Przykładowo, zapytanie Z1C wyszukuje wartości wszystkich atrybutów pracowników (krotek relacji PRACOWNIK), którzy pracują w dziale (reprezentowanym przez krotki relacji DZIAŁ) oznaczonym numerem piątym (patrz rysunek 6.3(g)); natomiast zapytanie Z1D wyszukuje wartości wszystkich atrybutów krotek relacji PRACOWNIK i wartości wszystkich atrybutów tych krotek relacji DZIAŁ, w których dany pracownik jest zatrudniony — generowany wynik obejmuje wyłącznie pracowników działu 'Badania'. Zapytanie Z10A określa w istocie iloczyn kartezjański relacji PRACOWNIK i DZIAŁ.

```
Z1C:  SELECT *  
      FROM  PRACOWNIK  
      WHERE NRDZ = 5;
```

```
Z1C:  SELECT *  
      FROM  PRACOWNIK, DZIAŁ  
      WHERE NAZWADZ = 'Badania' AND NUMERDZ = NRDZ;
```

```
Z10A: SELECT *  
      FROM  PRACOWNIK, DZIAŁ;
```

### 6.3.4. Tabele i zbiory w języku SQL

Jak już wspominaliśmy, język SQL często traktuje tabelę nie jako zbiór, ale raczej jako **wielozbiór**; oznacza to, że pojedyncza tabela *może zawierać więcej niż jedną taką samą krotkę*. Efektem przyjęcia takiego rozwiązania jest brak automatycznego eliminowania powtarzających się krotek z wyników zapytań języka SQL; powodów jest kilka:

- Eliminacja powtórzeń jest operacją kosztowną. Jednym ze sposobów jej implementowania jest posortowanie wszystkich krotek i przeszukanie posortowanej tabeli pod kątem zawierania pary sąsiadujących ze sobą identycznych krotek.
- Użytkownik może oczekiwać, że rezultat wykonanego zapytania będzie zawierał powtarzające się krotki.
- Kiedy dla krotek jest stosowana funkcja agregująca (patrz punkt 7.1.7), w większości przypadków eliminacja powtórzeń jest nieuzasadniona.

Tabela języka SQL z zadeklarowanym kluczem musi być zbiorem, ponieważ wartość klucza musi być inna w każdej z krotek<sup>10</sup>. Jeśli uznamy, że wyeliminowanie ewentualnych powtórzeń z wyniku zapytania języka SQL *jest uzasadnione*, możemy użyć słowa kluczowego DISTINCT w klauzuli SELECT, które oznacza, że w wygenerowanym wyniku powinny się znaleźć wyłącznie unikatowe krotki. Ogólnie rzecz biorąc, zapytanie rozpoczynające się od słów SELECT DISTINCT eliminuje znalezione powtórzenia, natomiast zapytanie rozpoczynające się od słów SELECT ALL kwalifikuje te powtórzenia do relacji wynikowej. Użycie klauzuli SELECT bez słowa kluczowego ALL i bez słowa kluczowego DISTINCT (a więc tak, jak we wszystkich prezentowanych do tej pory przykładach) jest równoważne zastosowaniu słowa kluczowego ALL. Przykładowo, zapytanie 11. pobiera pensje wszystkich pracowników; jeśli wielu pracowników otrzymuje taką samą pensję, jej wartość będzie występowała w wygenerowanym wyniku zapytania wielokrotnie (patrz rysunek 6.4(a)). Jeśli interesują nas tylko unikatowe wartości wynagrodzeń i chcemy, aby każda z tych wartości występowała w wygenerowanym wyniku tylko raz (niezależnie od liczby pracowników, których dotyczy), powinniśmy użyć słowa kluczowego DISTINCT. Takie rozwiązanie zastosowaliśmy w zapytaniu Z11A, którego wynik przedstawiono na rysunku 6.4(b).

---

<sup>10</sup> W zasadzie tabela języka SQL nie musi mieć zadeklarowanego klucza, jednak w zdecydowanej większości przypadków projektanci baz danych wskazują unikatowe atrybuty pełniące tę rolę.

(a)

Pensja
3000
4000
2500
4300
3800
2500
2500
5500

(b)

Pensja
3000
4000
2500
4300
3800
5500

(c)

Imię	Nazwisko

(d)

Imię	Nazwisko
Józef	Bąk

RYSunek 6.4. Wyniki dodatkowych zapytań języka SQL wykonanych na bazie danych FIRMA znajdującej się w stanie przedstawionym na rysunku 5.6: (a) Z11, (b) Z11A, (c) Z16, (d) Z18

**Zapytanie 11.** Znajdź pensje pobierane przez wszystkich pracowników (zapytanie Z11) oraz znajdź tylko różne wysokości wynagrodzeń (zapytanie Z11A).

```
Z11: SELECT ALL PENSJA
      FROM PRACOWNIK;
```

```
Z11A: SELECT DISTINCT PENSJA
       FROM PRACOWNIK;
```

Autorzy standardu SQL bezpośrednio przenieśli do tego języka niektóre spośród operacji na zbiorach z matematycznej *teorii zbiorów* składającej się na algebrę relacyjną (patrz rozdział 8.). Należy do nich operacja sumy zbiorów (UNION), różnicy zbiorów (EXCEPT)<sup>11</sup> oraz iloczynu zbiorów (INTERSECT). Relacje wygenerowane za pomocą wymienionych operacji na zbiorach są zbiorami krotek, co oznacza, że *powtórzenia krotek są eliminowane z wyników*. Ponieważ wspomniane operacje mają zastosowanie tylko dla *relacji ze zgodnymi typami*, za każdym razem, gdy próbujemy zastosować którąś z tych operacji dla pary relacji, musimy się upewnić, że obie relacje zawierają takie same atrybuty występujące w takiej samej kolejności. Przykładowe zastosowanie operacji UNION ilustruje poniższe zapytanie.

**Zapytanie 4.** Wygeneruj listę numerów dla wszystkich projektów, w których realizację jest zaangażowany pracownik nazwiskiem 'Nowak' (niezależnie od tego, czy jego udział w projekcie polega na bezpośredniej realizacji, czy jest kierownikiem działu nadzorującego dany projekt).

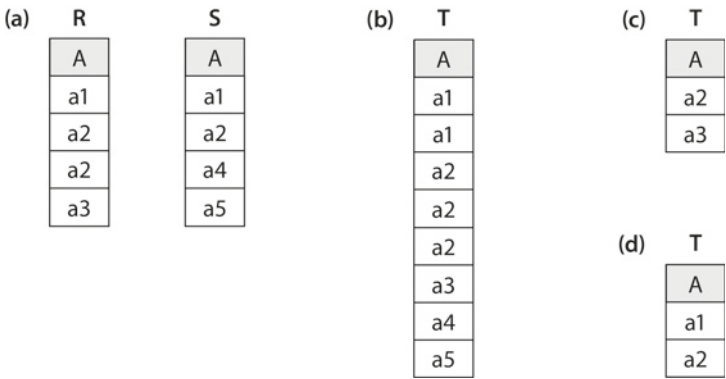
```
Z4: (SELECT DISTINCT NUMERPROJ
     FROM PROJEKT, DZIAŁ, PRACOWNIK
     WHERE NR_DZ = NUMERDZ AND PESELKIEROWNIKA = PESEL
      AND NAZWISKO = 'Nowak')
UNION
(SELECT DISTINCT NUMERPROJ
 FROM PROJEKT, PRACUJE_NAD, PRACOWNIK
 WHERE NUMERPROJ = NR_PROJ AND PESELPRAC = PESEL
      AND NAZWISKO = 'Nowak');
```

<sup>11</sup> W niektórych systemach operacja różnicy zbiorów jest reprezentowana za pomocą słowa kluczowego MINUS, a nie przy użyciu słowa kluczowego EXCEPT.



Pierwsze zapytanie SELECT pobiera projekty, w których uczestniczy pracownik nazwiskiem 'Nowak' w roli kierownika działu nadzorującego ten projekt; drugie zapytanie SELECT pobiera wszystkie te projekty, które są realizowane przez pracownika nazwiskiem 'Nowak'. Łatwo zauważyć, że jeśli wielu pracowników nazywa się 'Nowak', drugie zapytanie zwróci nazwy projektów, w których realizację zaangażowany jest którykolwiek z nich. Dopiero zastosowanie operacji sumy zbiorów (UNION) dla obu zapytań SELECT prowadzi do uzyskania oczekiwanego rezultatu.

Język SQL zawiera także odpowiednie operacje na wielozbiorach, których stosowanie wymaga jedynie dołączania słowa kluczowego ALL (UNION ALL, EXCEPT ALL oraz INTERSECT ALL). Wynikami tych operacji są także wielozbiory (ewentualne powtórzenia krotek nie są więc eliminowane). Działanie wszystkich trzech operacji zilustrowano za pomocą odpowiednich przykładów na rysunku 6.5. Mówiąc w skrócie, podczas wykonywania tych operacji każda krotka (niezależnie od tego, czy jest powtórzeniem) jest traktowana tak, jakby była unikatowa.



RYSunEK 6.5. Wyniki stosowania dostępnych w języku SQL operacji na wielozbiorach: (a) para tabel R(A) i S(A), (b) R(A) UNION ALL S(A), (c) R(A) EXCEPT ALL S(A), (d) R(A) INTERSECT ALL S(A)

### 6.3.5. Dopasowywanie podciągów znaków do wzorca oraz operacje arytmetyczne

W tym punkcie omówimy kilka istotnych mechanizmów dostępnych w języku SQL. Pierwszy z nich umożliwia stosowanie operacji porównywania tylko dla fragmentów ciągów znakowych — służy do tego operator porównania LIKE. Operator ten doskonale nadaje się do **dopasowywania ciągów do wzorców**. Ciągi częściowe można definiować za pomocą dwóch specjalnych znaków zastępczych: symbol procenta (%) zastępuje dowolną liczbę (a więc zero lub więcej) znaków, natomiast symbol podkreślenia ( ) zastępuje pojedynczy znak. Jako przykład, przeanalizujemy teraz poniższe zapytanie:

**Zapytanie 12.** Znajdź wszystkich pracowników, którzy mieszkają w Opolu w województwie Opolskim.

```
Z12: SELECT IMIĘ, NAZWISKO
      FROM PRACOWNIK
      WHERE ADRES LIKE '%Opole, Opolskie%';
```

Aby wyszukać wszystkich pracowników, którzy urodzili się w latach 70., możemy użyć poniższego zapytania 12A. W tym przypadku cyfra '7' musi być trzecim znakiem ciągu reprezentującego datę urodzenia (zgodnie z zastosowanym formatem daty), zatem wykorzystujemy wzorzec w postaci: ' \_\_7\_\_\_\_\_', gdzie każdy znak podkreślenia pełni rolę symbolu zastępczego dla dowolnego innego znaku.

**Zapytanie 12A.** Znajdź wszystkich pracowników, którzy urodzili się w latach 70.

```
Z12A: SELECT IMIĘ, NAZWISKO
      FROM PRACOWNIK
      WHERE DATAUR LIKE ' __7_____' ;
```

Jeśli w definiowanym wzorcu niezbędne jest użycie symbolu procenta lub podkreślenia w postaci stałego znaku należącego do ciągu, powinniśmy ten symbol poprzedzić tzw. *znakiem ucieczki*, który definiujemy bezpośrednio za danym ciągiem używając słowa kluczowego ESCAPE. Przykładowo, wzorzec 'AB\_CD\%EF' ESCAPE '\' reprezentuje ciąg stały 'AB\_CD%EF', ponieważ symbol lewego ukośnika (\) został określony jako znak ucieczki. Do roli znaku ucieczki można wybierać dowolne symbole, które nie znajdują się w oryginalnej stałej znakowej. Brakuje nam już tylko reguły umożliwiającej umieszczanie w stałych znakowych symbolu apostrofu ('), co w niektórych sytuacjach jest nie do uniknięcia — specjalne interpretowanie tego znaku wynika z faktu, że w języku SQL wykorzystuje się go do rozpoczynania i kończenia ciągów znaków. Jeśli użycie w ciągu znaku apostrofu (') jest konieczne, należy go reprezentować za pomocą pary apostrofów znajdujących się bezpośrednio obok siebie (' '), dzięki czemu nie zostanie on zinterpretowany jak zakończenie ciągu. Warto zauważyć, że porównywanie podciągów oznacza, iż wartości atrybutów są podzielne (nie są atomowe); jest to niezgodne z tym, co przyjęto w formalnym modelu relacyjnym (patrz podrozdział 5.1).

Inną cechą jest możliwość stosowania w zapytaniach operatorów arytmetycznych. Dla wartości numerycznych oraz atrybutów z dziedzinami numerycznymi można stosować takie standardowe operatory arytmetyczne jak dodawanie (+), odejmowanie (-), mnożenie (\*) i dzielenie (/). Przykładowo, przypuśćmy, że chcemy przeanalizować skutki przyznania dziesięcioprocentowej podwyżki wszystkim pracownikom zaangażowanym w realizację projektu 'ProjektX'; możemy do tego celu wykorzystać przedstawione poniżej zapytanie 13. Zaprezentowany przykład pokazuje także, jak można zmieniać nazwy atrybutów relacji wynikowej (za pomocą słowa kluczowego AS w klauzuli SELECT).

**Zapytanie 13.** Przedstaw poziom wynagrodzeń przy założeniu, że każdy pracownik realizujący projekt 'ProjektX' otrzyma dziesięcioprocentową podwyżkę.

```
Z13: SELECT E.IMIĘ, E.NAZWISKO, 1.1*E.PENSJA AS POWIĘKSZONA_PENSJA
      FROM PRACOWNIK AS E, PRACUJE_NAD AS PN, PROJEKT AS P
      WHERE E.PESEL = PN.PESELPRAC AND PN.NR_PROJ = P.NUMERPROJ
      AND P.NAZWAPROJ = 'ProjektX' ;
```

W przypadku znakowych typów danych do łączenia dwóch wartości znakowych należy oczywiście wykorzystywać wspomniany już operator konkatencji (||). W przypadku typów danych daty, czasu, znacznika czasowego oraz przedziału czasowego mamy do dyspozycji operatory zwiększania (+) i zmniejszania (-) daty, czasu lub znacznika czasowego o dany przedział czasowy. Wynikiem operacji odejmowania dat, czasu lub znacznika

czasowego jest przedział czasowy. Kolejnym wygodnym operatorem porównania, który może być wykorzystywany w zapytaniach języka SQL, jest BETWEEN (przykładem zastosowania tego operatora jest poniższe zapytanie 14.).

**Zapytanie 14.** Wyszukaj wszystkich pracowników działu 5., których pensja mieści się w przedziale od 3000 do 4000 złotych.

```
Z14: SELECT *
      FROM PRACOWNIK
      WHERE (PENSJA BETWEEN 3000 AND 4000) AND NRDZ = 5;
```

Użyty w zapytaniu Z14 warunek (PENSJA BETWEEN 3000 AND 4000) jest równoważny warunkowi ((PENSJA >= 3000) AND (PENSJA <= 4000)).

### 6.3.6. Sortowanie wyników zapytań

Język SQL oferuje możliwość sortowania krotek w relacji będącej wynikiem zapytania według wartości jednego lub więcej atrybutów — do tego celu służy klauzula ORDER BY. Przykładem zastosowania tej klauzuli jest zaprezentowane poniżej zapytanie 15.

**Zapytanie 15.** Pobierz listę pracowników i projektów, nad którymi pracują. Lista ma być posortowana według działów, a w ramach każdego działu — alfabetycznie według nazwiska, a następnie według imienia.

```
Z15: SELECT D.NAZWADZ, E.NAZWISKO, E.IMIĘ, P.NAZWAPROJ
      FROM DZIAŁ AS D, PRACOWNIK AS E, PRACUJE_NAD AS PN, PROJEKT AS P
      WHERE D.NUMERDZ = E.NRDZ AND E.PESEL = PN.PESELPRAC
            AND PN.NR_PROJ = P.NUMERPROJ
      ORDER BY D.NAZWADZ, E.NAZWISKO, E.IMIĘ;
```

Domyślnym uporządkowaniem sortowanych krotek jest kolejność rosnąca. Jeśli jednak chcemy, aby wynik zapytania zawierał krotki uporządkowane malejąco, możemy użyć słowa kluczowego DESC. Do jawnego wymuszania sortowania rosnącego służy słowo kluczowe ASC. Przykładowo, jeśli chcemy otrzymać w wyniku zapytania nazwy działów (wartości atrybutu NAZWADZ) posortowane malejąco oraz nazwiska i imiona pracowników (odpowiednio wartości atrybutów NAZWISKO i IMIĘ) posortowane rosnąco, nasza klauzula z zapytania Z15 powinna mieć postać:

```
ORDER BY D.NAZWADZ DESC, E.NAZWISKO ASC, E.IMIĘ ASC
```

### 6.3.7. Omówienie i podsumowanie prostych zapytań języka SQL

Proste zapytanie zdefiniowane w języku SQL może się składać z czterech klauzul, ale tylko dwie pierwsze (SELECT i FROM) są wymagane. Klauzule są deklarowane w następującej kolejności (nawiasami kwadratowymi oznaczono klauzule opcjonalne):

```
SELECT <LISTA ATRYBUTÓW>
FROM <LISTA TABEL>
[WHERE <WARUNEK>]
[ORDER BY <LISTA ATRYBUTÓW>]
```

Klauzula **SELECT** zawiera listę atrybutów, które mają się znaleźć w wyniku zapytania. Klauzula **FROM** określa wszystkie relacje (tabele) niezbędne do wykonania danego zapytania. Klauzula **WHERE** określa warunki selekcji krotek z tych relacji, włącznie z ewentualnymi warunkami złączenia (jeśli ich zastosowanie jest niezbędne). Klauzula **ORDER BY** określa kolejność wyświetlania krotek w relacji wynikowej zapytania. W punkcie 7.1.8 opiszemy dwie dodatkowe klauzule: **GROUP BY** i **HAVING**.

W rozdziale 7. przedstawimy bardziej skomplikowane mechanizmy zapytań pobierania danych w języku SQL. Te mechanizmy to: zagnieżdżone zapytania umożliwiające zastosowanie jednego zapytania jako części innego, funkcje agregujące używane do tworzenia podsumowań informacji z tabel, dwie dodatkowe klauzule: **GROUP BY** i **HAVING**, zwiększające możliwości funkcji agregujących, a także różnego rodzaju złączenia pozwalające łączyć na różne sposoby rekordy z rozmaitych tabel.

## 6.4. Dostępne w języku SQL polecenia INSERT, DELETE i UPDATE

Język SQL oferuje trzy polecenia, które mogą być wykorzystywane do modyfikowania baz danych: **INSERT**, **DELETE** oraz **UPDATE**. Kolejno omówimy każde z tych poleceń.

### 6.4.1. Polecenie INSERT

W najprostszej formie polecenie **INSERT** jest wykorzystywane do dodawania pojedynczych krotek (wierszy) do relacji (tabel). Musimy określić nazwę relacji i listę wartości dla nowej krotki. Wartości powinniśmy zapisywać w *takiej samej kolejności*, w jakiej zdefiniowano odpowiadające im atrybuty jeszcze w poleceniu **CREATE TABLE**. Przykładowo, aby dodać nową krotkę do tabeli przedstawionej na rysunku 5.5 i zdefiniowanej za pomocą polecenia **CREATE TABLE PRACOWNIK** ... z rysunku 6.1 relacji **PRACOWNIK**, powinniśmy użyć polecenia A1:

```
A1: INSERT INTO PRACOWNIK
VALUES ('Ryszard', 'Krzysztof', 'Maciejewski', '62123087022', '1962-12-30',
       'ul. Szpitalna 4/8, Kłodawa', 'M', 3700, '410620132586', 4);
```

Druga postać polecenia **INSERT** umożliwia użytkownikowi dodatkowe określenie nazw atrybutów odpowiadających kolejnym wartościom wstawianym za pomocą tego polecenia. Takie rozwiązanie jest szczególnie przydatne w sytuacji, gdy modyfikowana relacja zawiera wiele atrybutów, ale tylko niektóre z nich mają przypisywane wartości należące do nowej krotki. Wstawiana krotka musi jednak zawierać wartości zarówno dla wszystkich atrybutów zabezpieczonych ograniczeniem **NOT NULL** oraz bez określonej wartości domyślnej. W takiej postaci polecenia **INSERT** można *pominąć* atrybuty, w przypadku których dozwolone jest stosowanie wartości pustych lub dla których określono wartości domyślne. Przykładowo, aby dodać do relacji **PRACOWNIK** nową krotkę reprezentującą pracownika, dla którego znamy tylko wartości atrybutów **IMIE**, **NAZWISKO**, **NRDZ** i **PESEL**, możemy użyć przedstawionego poniżej polecenia A1A:

```
A1A: INSERT INTO PRACOWNIK(IMIE, NAZWISKO, NRDZ, PESEL)
VALUES ('Ryszard', 'Krzysztof', 4, '62123087022');
```

Atrybuty, których nie wymieniliśmy w poleceniu A1A, będą miały przypisane wartości domyślne (z klauzuli DEFAULT) lub wartości puste (NULL), natomiast wartości dla nowej krotki zostały wymienione w takiej samej kolejności jak *atrybuty wskazane w samym poleceniu INSERT*. Istnieje także możliwość wykorzystania pojedynczej operacji INSERT do wstawiania do relacji *wielu krotek* oddzielonych przecinkami. Wartości atrybutów dla *każdej z tych krotek* należy wówczas umieścić w nawiasach.

System zarządzania bazą danych, który w pełni implementuje standard SQL, powinien prawidłowo obsługiwać i wymuszać wszystkie więzy integralności, które mogą być definiowane w języku DDL. Przykładowo, jeśli spróbujemy wykonać polecenie A2 na bazie danych przedstawionej na rysunku 5.6, system zarządzania bazą danych powinien *odrzuścić* operację, ponieważ w bazie nie istnieje krotka DZIAŁ z atrybutem NUMERDZ = 2. Podobnie polecenie A2A powinno zostać *odrzucone*, ponieważ nie definiuje wartości dla atrybutu PESEL nowej krotki, a jest on kluczem głównym i nie może mieć wartości pustej.

```
A2: INSERT INTO PRACOWNIK(IMIĘ, NAZWISKO, PESEL, NRDZ)
VALUES ('Ryszard', 'Krzysztof', '62123087022', 4);
(* Polecenie A2 zostanie odrzucone, jeśli SZBD weryfikuje zgodność z więzami integralności odwołań. *)
A2A: INSERT INTO PRACOWNIK(IMIĘ, NAZWISKO, NRDZ)
VALUES ('Robert', 'Haremski', 5);
(* Polecenie A2A zostanie odrzucone, jeśli SZBD weryfikuje zgodność z ograniczeniem NOT NULL. *)
```

Istnieje jeszcze jedna odmiana polecenia INSERT, która wykonuje operację wstawiania do relacji wielu krotek w połączeniu z operacją utworzenia tej relacji i wypełnienia jej danymi otrzymanymi w wyniku zapytania. Przykładowo, aby stworzyć tymczasową tabelę, która będzie zawierała nazwisko pracownika, nazwę projektu i liczbę godzin poświęconych przez dane osoby tygodniowo na określony projekt, możemy użyć przedstawionych poniżej poleceń A3A i A3B:

```
A3A: CREATE TABLE INFO_O_PROJEKTACH
(NAZWISKO_PRAC VARCHAR(15),
NAZWA_PROJEKTU VARCHAR(15),
GODZINY_TYGODNIOWO DECIMAL(3,1) );
A3B: INSERT INTO INFO_O_PROJEKTACH(NAZWISKO_PRAC, NAZWA_PROJEKTU,
GODZINY_TYGODNIOWO)
SELECT E.NAZWISKO, P.NAZWAPROJ, PN.GODZINY
FROM PROJEKT P, PRACUJE_NAD PN, PRACOWNIK E
WHERE P.NUMERPROJ = PN.NR_PROJ AND PN.PESELPRAC = E.PESEL;
```

Za utworzenie tabeli INFO\_O\_PROJEKTACH odpowiada polecenie A3A, natomiast polecenie A3B jest wykorzystywane do wypełnienia nowej tabeli danymi z bazy. Możemy teraz wykonać na tabeli INFO\_O\_PROJEKTACH dowolne interesujące nas zapytanie; kiedy stwierdzimy, że tabela ta nie jest nam już potrzebna, możemy ją po prostu usunąć za pomocą polecenia DROP TABLE (patrz rozdział 7.). Warto zauważyć, że tak utworzona tabela INFO\_O\_PROJEKTACH może zawierać nieaktualne dane — w przypadku zaktualizowania danych zawartych w relacji PROJEKT, PRACUJE\_NAD lub PRACOWNIK już po wykonaniu polecenia A3B, informacje zawarte w relacji INFO\_O\_PROJEKTACH staną się *nieaktualne*. Stałe utrzymywanie aktualnych danych w tego typu strukturach wymaga zastosowania perspektywy (patrz rozdział 7.).

Większość SZBD obejmuje narzędzia do *wczytywania masowego*, które umożliwiają użytkownikom wczytywanie sformatowanych danych z pliku bez konieczności pisania dużej liczby poleceń INSERT. Użytkownik może też napisać program, który wczytuje każdy rekord z pliku, formatuje dane do postaci wiersza tabeli i wstawia dane za pomocą pętli z języka programowania (patrz omówienie technik programowania baz danych w rozdziałach 10. i 11.).

Innym sposobem na wczytanie danych jest utworzenie nowej tabeli TNEW o tych samych atrybutach co istniejąca tabela T i przeniesienie części danych z T do TNEW. Służy do tego klauzula LIKE. Jeśli np. chcesz utworzyć tabelę D5PRAC o strukturze analogicznej do tabeli PRACOWNIK i zapisać nową tabelę wierszami pracowników zatrudnionych w dziale piątym, możesz napisać następującą instrukcję w języku SQL:

```
CREATE TABLE D5PRAC LIKE PRACOWNIK
(SELECT E.*
FROM PRACOWNIK AS E
WHERE E.NRDZ = 5) WITH DATA;
```

Klauzula WITH DATA oznacza, że tabela zostanie utworzona i wypełniona danymi określonymi w zapytaniu (choć w niektórych implementacjach może ona zostać zignorowana).

## 6.4.2. Polecenie DELETE

Polecenie DELETE służy w języku SQL do usuwania krotek z relacji. Polecenie to obejmuje klauzulę WHERE (podobną do odpowiedniej klauzuli stosowanej w zapytaniach języka SQL), za pomocą której użytkownik może określać warunki selekcji krotek przeznaczonych do usunięcia. Jednocześnie można usuwać krotki bezpośrednio tylko z jednej tabeli. Operacja usuwania może jednak być propagowana na krotki przechowywane w innych relacjach, jeśli w więzach integralności odwołań zdefiniowano (w języku DDL) odpowiednie *akcje wywoływane za pośrednictwem odwołań* (patrz punkt 6.2.2)<sup>12</sup>. W zależności od liczby krotek wyselekcjonowanych w oparciu o warunek zadeklarowany w klauzuli WHERE, pojedyncze polecenie DELETE może usunąć zero, jedną lub wiele krotek. W przypadku braku klauzuli WHERE polecenie DELETE usuwa ze wskazanej relacji wszystkie zawarte w niej krotki; sama tabela jednak pozostaje w bazie danych (w postaci tabeli puste). Operacje DELETE użyte w poleceniach od A4A do A4D (jeśli zostaną zastosowane niezależnie od siebie dla stanu bazy danych z rysunku 5.6) usuną odpowiednio zero, jedną, cztery oraz wszystkie krotki z relacji PRACOWNIK:

```
A4A: DELETE FROM PRACOWNIK
      WHERE NAZWISKO = 'Boliłowa';
A4B: DELETE FROM PRACOWNIK
      WHERE PESEL = '65010912345';
A4C: DELETE FROM PRACOWNIK
      WHERE NRDZ = 5;
A4D: DELETE FROM PRACOWNIK;
```

<sup>12</sup> Pozostałe działania mogą być stosowane automatycznie za pośrednictwem tzw. wyzwalaczy (patrz podrozdział 26.1) i innych mechanizmów.

### 6.4.3. Polecenie UPDATE

Polecenie UPDATE jest wykorzystywane do modyfikowania wartości atrybutów w jednej lub w większej liczbie wyselekcjonowanych krotek. Podobnie jak w przypadku omówionego przed chwilą polecenia DELETE, stosowana w poleceniach UPDATE klauzula WHERE zawiera warunki selekcji krotek pochodzących z pojedynczej relacji i przeznaczonych do zmodyfikowania. Warto jednak pamiętać, że zmiana wartości klucza głównego może być propagowana na wartości kluczy obcych w krotkach należących do innych relacji, jeśli w więzach integralności odwołań zdefiniowano (w języku DDL) odpowiednie *akcje wywoływane za pośrednictwem odwołań* (patrz punkt 6.2.2). Stosowana w poleceniu UPDATE dodatkowa klauzula SET określa atrybuty, które mają zostać zmodyfikowane, wraz z ich docelowymi wartościami. Przykładowo, aby zmienić lokalizację i numer działu nadzorującego dla projektu oznaczonego numerem 10 odpowiednio na 'Kielce' i 5, należy użyć następującego polecenia A5:

```
A5: UPDATE PROJEKT
     SET   LOKALIZACJAPROJ = 'Kielce', NR_DZ = 5
     WHERE NUMERPROJ = 10;
```

Za pomocą pojedynczego polecenia UPDATE można zmodyfikować wiele krotek. Przykładem takiego zastosowania tego polecenia jest przyznanie wszystkim pracownikom działu 'Badania' dziesięcioprocentowej podwyżki (patrz poniższe polecenie A6). W przypadku tego żądania zmodyfikowana wartość atrybutu PENSJA jest oczywiście uzależniona od oryginalnej wartości tego atrybutu w danej krotce, zatem konieczne jest umieszczenie w odpowiednim poleceniu dwóch odwołań do atrybutu PENSJA. Oba odwołania muszą się znaleźć w klauzuli SET — odwołanie do atrybutu PENSJA po prawej stronie wskazuje na jego starą wartość, a więc istniejącą *przed modyfikacją*, natomiast odwołanie do tego samego atrybutu umieszczone po lewej stronie wskazuje na jego nową wartość, a więc liczbę docelową *po wprowadzeniu modyfikacji*.

```
A6: UPDATE PRACOWNIK
     SET   PENSJA = PENSJA * 1.1
     WHERE NRDZ = 5;
```

Istnieje także możliwość określenia wartości pustej lub wartości domyślnej (słowa kluczowe NULL oraz DEFAULT) jako docelowej wartości modyfikowanego atrybutu. Warto pamiętać, że każde polecenie UPDATE jawnie może się odwoływać tylko do jednej relacji. Aby zmodyfikować większą ich liczbę, musimy użyć wielu poleceń tego typu.

## 6.5. Dodatkowe własności języka SQL

Język SQL oferuje wiele dodatkowych mechanizmów, których nie omówiliśmy w tym rozdziale, ale które zaprezentujemy w dalszej części tej książki. Należą do nich następujące elementy:

- W rozdziale 7., stanowiącym ciąg dalszy tego rozdziału, prezentujemy następujące mechanizmy języka SQL: różne techniki tworzenia złożonych zapytań pobierających dane (te techniki to m.in.: zapytania zagnieżdżone, funkcje agregujące, grupowanie, tabele połączone, złączenia zewnętrzne, instrukcje CASE i zapytania rekurencyjne), perspektywy, wyzwalacze, asercje i polecenia modyfikowania schematów.



- Język SQL oferuje możliwość stosowania wielu różnych technik w zakresie tworzenia w rozmaitych językach programowania programów, które mogą zawierać polecenia języka SQL uzyskujące dostęp do jednej lub wielu baz danych. Jest to między innymi osadzony (i dynamiczny) język SQL, interfejs SQL/*CLI* (ang. *Call Language Interface*) i jego poprzednik *ODBC* (ang. *Open DataBase Connectivity*) oraz moduły SQL/*PSM* (ang. *Program Stored Modules*). Te techniki są opisane w rozdziale 10. Przedstawimy także metody uzyskiwania dostępu do baz danych SQL z poziomu języka programowania Java (za pośrednictwem interfejsów *JDBC* i *SQLJ*).
- Każdy komercyjny system zarządzania relacyjną bazą danych udostępnia (poza poleceniami języka SQL) zbiór poleceń umożliwiających określanie fizycznych parametrów projektu bazy danych, plikowej struktury relacji oraz ścieżek dostępu dla tego typu indeksów. W rozdziale 2. nazwaliśmy te polecenia językiem definicji składowania (ang. *Storage Definition Language — SDL*). Wcześniejsze wersje języka SQL oferowały co prawda polecenia zaprojektowane specjalnie z myślą o **tworzeniu indeksów**, jednak z czasem zrezygnowano z implementowania podobnych mechanizmów na poziomie tego języka, ponieważ nie należą one do poziomu schematu koncepcyjnego. W wielu systemach nadal dostępne są polecenia *CREATE INDEX*, jednak wymagają one specjalnych uprawnień. Opiszemy je w rozdziale 17.
- Język SQL oferuje polecenia umożliwiające sterowanie transakcjami. Za ich pomocą można określać jednostki przetwarzania informacji zawartych w bazie danych odpowiedzialne za sterowanie współbieżne i odzyskiwanie danych. Polecenia z tej grupy omówimy w rozdziale 20. bezpośrednio po tym, jak szczegółowo przedstawimy pojęcie transakcji.
- SQL zawiera pewne konstrukcje językowe pozwalające na *przydzielanie i odbieranie użytkownikom uprawnień*. Uprawnienia (nazywane także przywilejami) odpowiadają zwykle możliwości wykorzystywania przez poszczególnych użytkowników określonych poleceń języka SQL lub uzyskiwania dostępu do określonych relacji. Każda relacja ma przypisanego właściciela — zazwyczaj zarówno właściciel, jak i użytkownicy z uprawnieniami administratorów bazy danych mogą przyznawać wybranym przez siebie użytkownikom prawa do stosowania dla danej relacji poleceń języka SQL (takich jak *SELECT*, *INSERT*, *DELETE* lub *UPDATE*). Dodatkowo, administratorzy baz danych mogą nadawać niektórym użytkownikom prawo tworzenia schematów, tabel lub perspektyw. Odpowiednie polecenia języka SQL (*GRANT* i *REVOKE*) omówimy w rozdziale 20., który w całości jest poświęcony bezpieczeństwu baz danych i mechanizmom autoryzacji.
- Język SQL zawiera konstrukcje niezbędne do tworzenia tzw. wyzwalaczy (ang. *triggers*). Odpowiednie mechanizmy są zasadniczo związane z technikami **aktywnych baz danych**, ponieważ określają działania podejmowane (wyzwalane) automatycznie w odpowiedzi na występowanie takich zdarzeń jak aktualizacje informacji zawartych w bazie danych. Omówimy te mechanizmy w podrozdziale 26.1, w którym będziemy analizowali pojęcia związane z aktywnymi bazami danych.
- Współczesne implementacje języka SQL oferują wiele elementów pochodzących z obiektowego modelu danych — rozszerzenie modelu relacyjnego o elementy obiektowe doprowadziło do stworzenia nowego modelu, nazywanego modelem **obiekto-relacyjnym**. Mechanizmy tworzenia atrybutów o skomplikowanej strukturze, określania dla atrybutów i tabel abstrakcyjnych typów danych

(nazywanych typami **UDT** lub typami użytkownika), tworzenia **obiektowych identyfikatorów** dla krotek oraz definiowania **operacji** dla typów danych omówimy w rozdziale 12.

- Język SQL i relacyjne bazy danych mogą współpracować z nowymi technologiami, w tym technologią *XML* (ang. *eXtended Markup Language*), patrz rozdział 13., oraz systemami *OLAP* (ang. *OnLine Analytical Processing for Data Warehouses*) i hurtowniami danych, patrz rozdział 29.

## 6.6. Podsumowanie

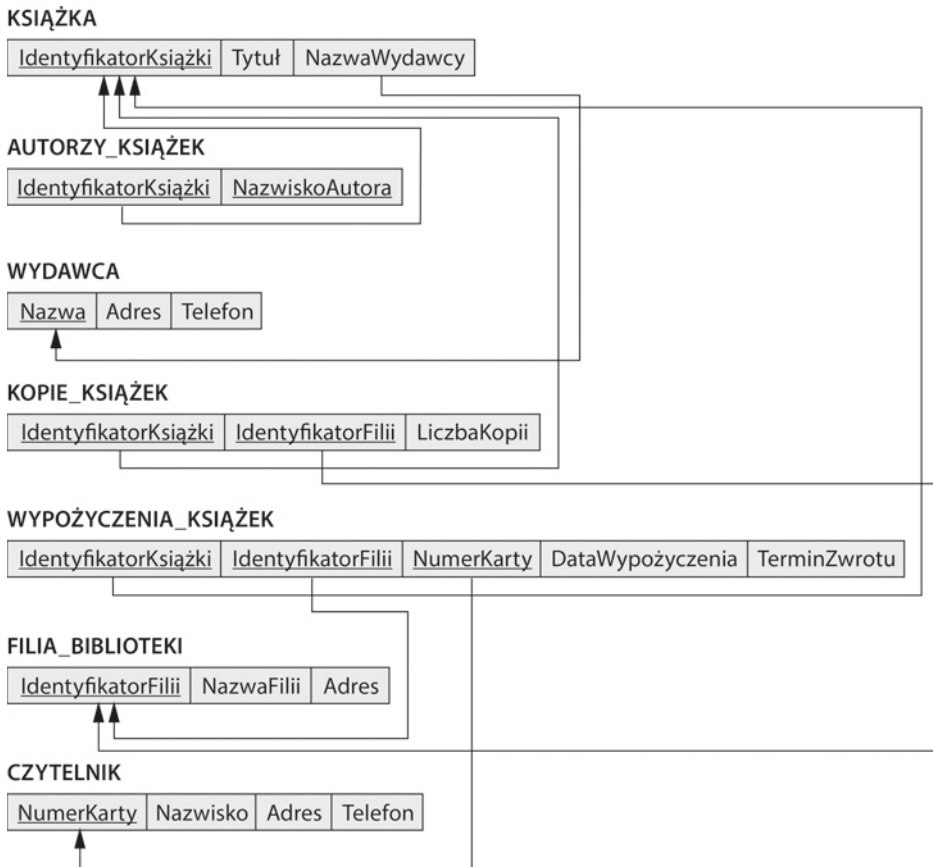
W tym rozdziale zaprezentowaliśmy popularny język baz danych — SQL. Różne odmiany tego języka zostały zaimplementowane w roli interfejsów do wielu komercyjnych systemów zarządzania relacyjnymi bazami danych, w tym systemów firmy Oracle, produktów DB2 firmy IBM, systemu SQL Server firmy Microsoft, a wielu innych systemów takich jak Sybase i INGRES. Także niektóre systemy o otwartym dostępie do kodu źródłowego (np. MySQL i PostgreSQL) obsługują język SQL. Oryginalna wersja standardu SQL została zaimplementowana jako część opracowanego w ośrodku IBM Research eksperymentalnego systemu zarządzania relacyjną bazą danych, który nazwano SYSTEM R. Standard SQL zaprojektowano z myślą o zapewnieniu uniwersalnego języka obejmującego polecenia definiowania danych, konstruowania zapytań i operacji aktualizujących, definiowania perspektyw oraz określania ograniczeń. W tym rozdziale omówiliśmy następujące mechanizmy języka SQL: polecenia definiowania danych służące do tworzenia tabel, podstawowe typy danych, polecenia do określania ograniczeń, proste zapytania pobierające dane i polecenia aktualizujące bazę. W następnym rozdziale przedstawimy następujące funkcje języka SQL: złożone zapytania pobierające dane, perspektywy, wyzwalacze, asercje i polecenia do modyfikowania schematów.

## Pytania powtórkowe

- 6.1. Co odróżnia relacje (tabele) przetwarzane za pomocą poleceń języka SQL od relacji zdefiniowanych formalnie w rozdziale 3.? Omów także pozostałe różnice w terminologii. Dlaczego język SQL dopuszcza możliwość występowania powtarzających się krotek w tabelach lub w wynikach zapytań?
- 6.2. Wymień typy danych, które można stosować dla atrybutów języka SQL.
- 6.3. W jaki sposób można w języku SQL deklarować więzy integralności encji i więzy integralności odwołań (oba typy ograniczeń opisano w rozdziale 3.)? Jaką rolę odgrywają akcje wywoływane za pośrednictwem odwołań?
- 6.4. Opisz cztery klauzule stosowane w składni prostych zapytań języka SQL i przedstaw dostępne rodzaje konstrukcji, które można deklarować w każdej z nich. Które z tych klauzul są wymagane, a które mają charakter opcjonalny?

## Ćwiczenia

- 6.5. Przeanalizuj bazę danych zaprezentowaną na rysunku 1.2, której schemat przedstawiono na rysunku 2.1. Jakie więzy integralności odwołań powinny być spełnione w przypadku tego schematu? Zapisz polecenia języka SQL DDL, które stworzą odpowiednią definicję tej bazy danych.
- 6.6. Powtórz ćwiczenie 6.5 dla schematu bazy danych LINIE LOTNICZE z rysunku 5.8.
- 6.7. Przeanalizuj schemat relacyjnej bazy danych BIBLIOTEKA z rysunku 6.6. Wybierz właściwą odpowiedź (odrzuć żądanie, kaskadowe propagowanie żądania, ustawienie wartości pustej lub ustawienie wartości domyślnej) dla każdego z więzów integralności odwołań, zarówno tych związanych z *usuwaniami* wskazywanych krotek, jak i związanych z *aktualizacjami* wartości atrybutów kluczy głównych wskazywanych krotek. Odpowiedź uzasadnij.
- 6.8. Zapisz odpowiednie polecenia języka SQL DDL deklarujące schemat relacyjnej bazy danych BIBLIOTEKA (patrz rysunek 6.6). Wskaż odpowiednie klucze i ewentualne akcje wywoływane za pośrednictwem odwołań.



RYСУNEK 6.6. Schemat relacyjnej bazy danych BIBLIOTEKA

- 6.9. Jak na poziomie systemu zarządzania bazą danych można wymuszać przestrzeganie ograniczeń klucza i ograniczeń klucza obcego? Czy zaproponowana przez Ciebie technika wymuszania tych ograniczeń jest trudna do zaimplementowania? Czy mechanizm weryfikacji ewentualnych naruszeń tych ograniczeń może być efektywnie stosowany w czasie aktualizacji bazy danych?
- 6.10. Zdefiniuj w języku SQL poniższe zapytania dodatkowe dla bazy danych z rysunku 5.5. Przedstaw wyniki tych zapytań przy założeniu, że baza danych FIRMA znajduje się w stanie zaprezentowanym na rysunku 5.6.
- a) Pobierz nazwiska wszystkich pracowników działu piątego, którzy pracują więcej niż 10 godzin tygodniowo nad projektem ProduktX.
  - b) Wyświetl nazwiska wszystkich pracowników, którzy mają członka rodziny o takim samym imieniu jak oni sami.
  - c) Znajdź nazwiska wszystkich pracowników będących bezpośrednimi podwładnymi 'Franciszek Wieszczycki'.
- 6.11. Zapisz operacje aktualizacji z ćwiczenia 3.11 za pomocą odpowiednich poleceń modyfikujących, które są dostępne w języku SQL.
- 6.12. Zdefiniuj w języku SQL następujące zapytania dla schematu bazy danych przedstawionego na rysunku 1.2.
- a) Pobierz nazwiska wszystkich studentów piątego roku kierunku 'INF' (studentów informatyki).
  - b) Pobierz nazwy wszystkich przedmiotów nauczanych przez profesora Kaczmarka w latach 2007 i 2008.
  - c) Dla każdego działu nauczanego przez profesora Kaczmarka znajdź numer przedmiotu, semestr, rok i liczbę studentów uczestniczących w zajęciach z danego działu.
  - d) Pobierz nazwisko i listę ocen każdego ze studentów piątego roku ( $RokSt = 5$ ) kierunku informatyka ( $Kierunek = 'INF'$ ). Lista ocen powinna obejmować nazwy przedmiotów, numery przedmiotów, godziny, semestr, rok i ocenę z każdego przedmiotu zaliczonego przez studenta.
- 6.13. Zapisz polecenia aktualizacji z języka SQL, za pomocą których można zrealizować wymienione poniżej operacje na schemacie bazy danych przedstawionym na rysunku 1.2.
- a) Wstaw do bazy danych informacje na temat nowego studenta: `<'Janiszewski', 25, 1, 'MAT'>`.
  - b) Zmień rok studenta nazwiskiem 'Nowak' na drugi.
  - c) Wstaw do bazy danych informacje na temat nowego przedmiotu: `<'Inżynieria wiedzy', 'INF4390', 3, 'INF'>`.
  - d) Usuń z bazy danych rekord reprezentujący studenta z nazwiskiem 'Nowak' i numerem indeksu równym 17.
- 6.14. Zaprojektuj schemat relacyjnej bazy danych dla swojej aplikacji bazy danych.
- a) Zadeklaruj relacje za pomocą odpowiednich poleceń języka SQL DDL.
  - b) Spróbuj określić liczbę zapytań języka SQL, których zdefiniowanie jest niezbędne w tej aplikacji.

- c) Bazując na swojej wiedzy odnośnie do spodziewanych zastosowań nowej bazy danych, wybierz kilka atrybutów, dla których należałoby stworzyć indeksy.
- d) Jeśli dysponujesz systemem zarządzania bazą danych, który obsługuje język SQL, zaimplementuj swoją bazę danych.

6.15. Załóżmy, że ograniczenie tabeli PRACOWNIK z rysunku 6.2 zostało zmienione w następujący sposób:

```
CONSTRAINT PRACPRZEŁOŻKO  
FOREIGN KEY (PESELPRZEŁOŻ) REFERENCES PRACOWNIK(PESEL)  
ON DELETE CASCADE ON UPDATE CASCADE.
```

Odpowiedz na następujące pytania:

- a) Co się stanie, jeśli poniższe polecenie zostanie uruchomione w bazie o stanie z rysunku 5.6?

```
DELETE EMPLOYEE WHERE NAZWISKO = 'Bąk'
```

- b) Czy w klauzuli ON DELETE ograniczenia PRACPRZEŁOŻKO lepiej jest zastosować ustawienie CASCADE czy SET NULL?

6.16. Zapisz instrukcje w języku SQL, aby utworzyć tabelę PRACOWNIK\_KOPIA\_ZAPASOWA w celu zarchiwizowania tabeli PRACOWNIK z rysunku 5.6.

## Wybrane publikacje

Język SQL (nazywany początkowo językiem SEQUEL) opierał się na opisanym przez Boyce'a i in. (1975) języku *SQUARE* (ang. *Specifying Queries as Relational Expressions*, czyli język definiowania zapytań w postaci wyrażeń relacyjnych). Składnia języka *SQUARE* została nieco zmodyfikowana w języku SEQUEL (Chamberlin i Boyce, 1974) i w później opracowanym języku SEQUEL 2 (Chamberlin i in., 1976), na którym bazował język SQL. Oryginalna implementacja języka SEQUEL jest dziełem ośrodka IBM Research w San Jose, w Kalifornii. Dodatkowe źródła dotyczące różnych aspektów języka SQL przedstawiamy w końcowej części rozdziału 7.



# 7

---

## Jeszcze o języku SQL — złożone zapytania, wyzwalacze, perspektywy i modyfikowanie schematów

W tym rozdziale przedstawimy zaawansowane mechanizmy języka SQL związane z bazami relacyjnymi. Zaczniemy od zaprezentowania w podrozdziale 7.1 złożonych aspektów zapytań pobierających dane — zapytań zagnieżdżonych, tabel połączonych, złączeń zewnętrznych, funkcji agregujących, grupowania i instrukcji CASE. W podrozdziale 7.2 opiszemy polecenie `CREATE ASSERTION`, które umożliwia definiowanie bardziej ogólnych ograniczeń dla bazy danych. Przedstawimy też wyzwalacze i instrukcję `CREATE TRIGGER`, której szczegółowy opis zawiera podrozdział 26.1, gdzie opisane są zasady działania aktywnych baz danych. Następnie (w podrozdziale 7.3) omówimy dostępne w języku SQL konstrukcje definiowania perspektyw dla bazy danych. Perspektywy są często nazywane *tabelami wirtualnymi* lub *tabelami pochodnymi*, ponieważ prezentują użytkownikowi dane w postaci tabel, ale w rzeczywistości generują zawarte w nich informacje na podstawie danych pochodzących z wcześniej zdefiniowanych relacji. W podrozdziale 7.4 przedstawiona jest instrukcja `ALTER TABLE`, używana do modyfikowania tabel i ograniczeń baz danych. Podrozdział 7.5 zawiera podsumowanie rozdziału.

Ten rozdział stanowi kontynuację rozdziału 6. Wykładowcy mogą pominąć fragmenty tego rozdziału, jeśli zamierzają przedstawić tylko ogólne wprowadzenie do języka SQL.

### 7.1. Bardziej skomplikowane zapytania języka SQL pobierające dane

W podrozdziale 6.3 opisaliśmy kilka podstawowych typów zapytań, jakie możemy definiować w języku SQL. Język ten charakteryzuje się jednak wyjątkową uniwersalnością i mocą wyrażania, istnieje więc wiele innych mechanizmów, które umożliwiają użytkownikom definiowanie znacznie bardziej skomplikowanych zapytań. Kilka takich mechanizmów omówimy w niniejszym podrozdziale.



### 7.1.1. Operacje porównania z wartością pustą (NULL) oraz logika trójwartościowa

Język SQL zawiera wiele różnych reguł dotyczących obsługi wartości pustych (NULL). W punkcie 5.1.2 wspominaliśmy, że wartość pustą wykorzystuje się generalnie do reprezentowania brakującej wartości, jednak zazwyczaj można ją interpretować na trzy różne sposoby — jako wartość *nieznana* (istniejąca, ale niepodana; możliwe też, że nie wiadomo, czy ta wartość istnieje), wartość *niedostępna* (istniejąca, ale celowo niepodana) lub wartość atrybutu *niemającego zastosowania* w danym przypadku (niedotyczącego lub niezdefiniowanego dla tej krotki). Przeanalizujemy teraz kilka wymienionych poniżej przykładów, które dobrze ilustrują każde z trzech znaczeń wartości pustej.

- (1) **Wartość nieznana:** Data urodzenia danej osoby jest nieznana, zatem jest reprezentowana w bazie danych przez wartość NULL. Innym przykładem tej kategorii jest wartość NULL atrybutu reprezentującego numer stacjonarny telefonu, ponieważ nie wiadomo, czy dana osoba posiada taki telefon.
- (2) **Wartość niedostępna lub zatajona:** Osoba reprezentowana w bazie danych ma telefon domowy, ale odmówiła podania jego numeru, a zatajona informacja jest reprezentowana w bazie danych w postaci wartości pustej.
- (3) **Brak zastosowania danego atrybutu:** Atrybut *Wykształcenie* wyższe będzie zawierał wartość NULL w przypadku osób, które nie ukończyły wyższych studiów, ponieważ atrybut ten nie ma zastosowania dla tych osób.

W wielu przypadkach określenie, z którym z wymienionych trzech znaczeń mamy do czynienia, nie jest możliwe; przykładowo, wartość NULL reprezentująca numer telefonu danej osoby może mieć każde z tych znaczeń. Możemy więc przyjąć, że język SQL w ogóle nie oferuje możliwości odróżniania trzech różnych znaczeń wartości pustej.

Zasadniczo, każda wartość pusta (NULL) jest traktowana jako wartość różna od wszystkich pozostałych wartości NULL, które są przechowywane w bazie danych. Kiedy wartość NULL występuje w operacji porównania, zakłada się, że wynik tej operacji jest NIEZNANY (może być PRAWDA lub FAŁSZEM). Oznacza to, że język SQL w praktyce stosuje logikę trójwartościową z wartościami PRAWDA, FAŁSZ i NIEZNANA (odpowiednio TRUE, FALSE i UNKNOWN), zamiast standardowej logiki dwuwartościowej z wartościami PRAWDA i FAŁSZ. W tej sytuacji niezbędne jest zdefiniowanie wyników wyrażeń logiki trójwartościowej dla dostępnych w języku SQL spójników (operatorów) AND, OR oraz NOT. Wartości wynikowe tych wyrażeń przedstawiono w tabeli 7.1.

W tabelach 7.1(a) i 7.1(b) wiersze i kolumny reprezentują wartości wyników porównań, które zwykle znajdują się w klauzuli WHERE zapytań języka SQL. Wynik każdego wyrażenia ma wartość TRUE, FALSE lub UNKNOWN. Wyniki połączenia dwóch wartości za pomocą wyrażenia logicznego AND są pokazane w tabeli 7.1(a). Tabela 7.1(b) obejmuje wyniki zastosowania wyrażenia logicznego OR. Na przykład wynik wyrażenia FALSE AND UNKNOWN to FALSE, a wynik wyrażenia FALSE OR UNKNOWN to UNKNOWN. W tabeli 7.1(c) znajdują się wyniki operacji logicznej NOT. Zauważ, że w standardowej logice boolowskiej dozwolone są tylko wartości TRUE i FALSE. Wartość UNKNOWN w niej nie występuje.

TABELA 7.1. Wyniki wyrażeń w logice trójwartościowej

(a)	AND	TRUE	FALSE	UNKNOWN
	TRUE	TRUE	FALSE	UNKNOWN
	FALSE	FALSE	FALSE	FALSE
	UNKNOWN	UNKNOWN	FALSE	UNKNOWN

(b)	OR	TRUE	FALSE	UNKNOWN
	TRUE	TRUE	TRUE	TRUE
	FALSE	TRUE	FALSE	UNKNOWN
	UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(c)	NOT	
	TRUE	FALSE
	FALSE	TRUE
	UNKNOWN	UNKNOWN

Istnieje ogólna zasada mówiąca, że w zapytaniach selekcji-projekcji-łączenia wybierane są tylko te krotki, dla których wyrażenie logiczne z klauzuli `WHERE` jest prawdziwe (ma wartość `TRUE`). Kombinacje krotek, dla których to samo wyrażenie jest fałszywe lub dla których wynik tego wyrażenia nie jest znany (odpowiednio wartości `FALSE` i `UNKNOWN`), nie są kwalifikowane do wyniku relacji. W przypadku niektórych operacji języka SQL (np. złączenia zewnętrznego) istnieją jednak wyjątki od tej reguły (patrz punkt 7.1.6).

Język SQL oferuje możliwość sprawdzania, czy wartość danego atrybutu jest równa `NULL`. Zamiast stosować operatory `=` lub `<>` do porównywania wartości atrybutu z wartością `NULL`, w języku SQL należy wykorzystywać operatory `IS` lub `IS NOT` (odpowiednio `JEST` lub `NIE JEST`). Wynika to z faktu, że język SQL traktuje każdą wartość pustą `NULL` tak, jakby była różna od wszystkich pozostałych wartości `NULL`, zatem stosowanie w wyrażeniach logicznych operatora równości jest błędne. Efektem takiej interpretacji jest odrzucanie krotek z wartością `NULL` reprezentowaną przez atrybuty uwzględniane w warunku złączenia (chyba że użyjemy operacji złączenia zewnętrznego; patrz punkt 7.1.6). W zapytaniu 18. przedstawiono porównanie obejmujące wartość `NULL` w instrukcji pobierającej dane pracowników, którzy nie mają przełożonego.

**Zapytanie 18.** Pobierz imiona i nazwiska wszystkich pracowników, którzy nie mają przełożonych.

```

Z18: SELECT IMIĘ, NAZWISKO
      FROM PRACOWNIK
      WHERE PESELPRZEŁOŻ IS NULL;
```

```
SELECT DISTINCT PESELPRAC
FROM PRACUJE_NAD
WHERE (NR_PROJ, GODZINY) IN (SELECT NR_PROJ, GODZINY
FROM PRACUJE_NAD
WHERE PESELPRAC = '65010912345');
```

Powyższe zapytanie wybiera numery PESEL wszystkich pracowników, którzy poświęcają pracy nad tym samym projektem taką samą liczbę godzin (a więc związanych z identyczną kombinacją atrybutów (NR\_PROJ, GODZINY)) jak pracownik 'Jan Nowak' (którego numer PESEL jest równy '65010912345'). W przedstawionym przykładzie operator IN porównuje umieszczoną w nawiasie podkrotkę wartości (NR\_PROJ, GODZINY) dla każdej krotki w relacji (tabeli) PRACUJE\_NAD ze zbiorem krotek zwróconym przez zapytanie zagnieżdżone (który musi mieć typ zgodny ze wspomnianą podkrotką wartości).

Poza operatorem IN język SQL oferuje możliwość wykorzystywania w zapytaniach wielu innych operatorów porównania do zestawiania pojedynczej wartości *v* (zwykle reprezentowanej przez nazwę atrybutu) ze zbiorem lub wielozbiorem *V* (mającym zwykle postać zagnieżdżonego zapytania). Operator = ANY (lub = SOME) zwraca wartość TRUE, jeśli wartość *v* jest równa  *pewnej wartości* w zbiorze *V*; operator ten jest więc równoważny operatorowi IN. Słowa kluczowe ANY i SOME mają w języku SQL identyczne znaczenie. Do pozostałych operatorów porównania, które mogą być łączone ze słowem kluczowym ANY (lub SOME), należą: >, >=, <, <= oraz <>. Każdy z wymienionych operatorów może być stosowany także ze słowem kluczowym ALL. Przykładowo, wyrażenie warunkowe (*v* > ALL *V*) ma wartość TRUE, jeśli wartość *v* jest większa od *wszystkich* wartości w zbiorze (lub wielozbiorze) *V*. Przykładem zastosowania tego operatora porównania jest poniższe zapytanie, które zwraca nazwiska i imiona pracowników, których pensja jest większa od pensji wszystkich pracowników działu 5.

```
SELECT NAZWISKO, IMIE
FROM   PRACOWNIK
WHERE  PENSJA > ALL (SELECT PENSJA
                     FROM   PRACUJE_NAD
                     WHERE  NRDZ = 5);
```

Zauważ, że to zapytanie można zapisać także za pomocą funkcji agregującej MAX (patrz punkt 7.1.7).

Zasadniczo możemy stosować w zapytaniach języka SQL wiele poziomów zagnieżdżania zapytań. Jeśli jednak w przeszukiwanych relacjach (tabelach) występują atrybuty oznaczone takimi samymi nazwami — jeden w relacji wymienionej w klauzuli FROM *zapytania zewnętrznego* i jeden w relacji wymienionej w klauzuli FROM *zapytania zagnieżdżonego* — możemy ponownie mieć do czynienia z niejednoznacznością odwołań do nazw atrybutów. W języku SQL stosowana jest reguła, która mówi, że odwołanie do *atrybutu nieoznaczonego kwalifikatorem* wskazuje na relację zadeklarowaną w **skrajnie wewnętrznym zapytaniu zagnieżdżonym**. Przykładowo, w klauzuli SELECT i klauzuli WHERE pierwszego zagnieżdżonego zapytania zdefiniowanego w zapytaniu Z4A odwołanie do dowolnego atrybutu relacji PROJEKT, którego nie oznaczono odpowiednim kwalifikatorem, wskazuje na relację PROJEKT wymienioną w klauzuli FROM zapytania zagnieżdżonego. Aby odwołać się do atrybutu relacji PROJEKT wymienionej w zewnętrznym zapytaniu, możemy dla tej relacji zdefiniować *alias* (zmienną krotki) i wprowadzić odwołania do tego aliasu. Wspomniane reguły są podobne do zasad określających zasięg zmiennych, które obowiązują w większości języków programowania umożliwiających stosowanie zagnieżdżonych procedur i funkcji. Aby lepiej zrozumieć ewentualność wystąpienia niejednoznaczności nazw atrybutów w zagnieżdżonych zapytaniach, przeanalizuj poniższe zapytanie 16.

**Zapytanie 16.** Pobierz imiona i nazwiska wszystkich pracowników, którzy mają członków rodziny z wartościami atrybutów reprezentujących imię i płeć identycznymi jak ci pracownicy.

```
Z16: SELECT P.IMIĘ, P.NAZWISKO
      FROM PRACOWNIK AS P
      WHERE P.PESEL IN (SELECT C.PESELPRAC
                        FROM CZŁONEK_RODZINY AS C
                        WHERE P.IMIĘ = C.IMIĘ_CZŁONKA_RODZINY
                        AND P.PŁEĆ = C.PŁEĆ);
```

W zapytaniu zagnieżdżonym w Z16 musimy użyć kwalifikowanej nazwy P.PŁEĆ, ponieważ chcemy się odwołać do atrybutu PŁEĆ relacji PRACOWNIK z zewnętrznego zapytania, a relacja CZŁONEK\_RODZINY obejmuje atrybut o takiej samej nazwie. Gdyby w zagnieżdżonym zapytaniu znalazły się nieopatrzone kwalifikatorem odwołania do atrybutu PŁEĆ, dotyczyłyby atrybutu z relacji CZŁONEK\_RODZINY. Łatwo zauważyć, że *nie musimy* stosować tego kwalifikatora dla atrybutów IMIĘ i PESEL, ponieważ relacja CZŁONEK\_RODZINY nie zawiera atrybutów IMIĘ i PESEL, zatem w tym przypadku nie mamy do czynienia z niejednoznacznością.

Ogólnie rzecz biorąc, zaleca się tworzenie zmiennych krotek (aliasów) dla *wszystkich* tabel, do których odwołujemy się wewnątrz zapytań języka SQL, ponieważ tylko w ten sposób możemy całkowicie wyeliminować zagrożenie związane z błędami niejednoznaczności (patrz zapytanie Z16).

### 7.1.3. Zagnieżdżone zapytania skorelowane

Za każdym razem, gdy warunek określony w klauzuli WHERE zagnieżdżonego zapytania odwołuje się do któregoś z atrybutu relacji zadeklarowanej w zapytaniu zewnętrznym, mówimy, że oba zapytania są **skorelowane**. Zrozumienie koncepcji skorelowanych zapytań jest łatwiejsze, jeśli przyjmiemy, że *zagnieżdżone zapytanie jest wykonywane dokładnie raz dla każdej krotki (lub każdej kombinacji krotek) w zewnętrznym zapytaniu*. Przykładowo, przedstawione powyżej zapytanie Z16 możemy interpretować w następujący sposób: dla *każdej* krotki relacji PRACOWNIK wykonaj zagnieżdżone zapytanie, które wyszukuje wartości atrybutu PESELPRAC wszystkich krotek relacji CZŁONEK\_RODZINY z imieniem i płcią identyczną jak imię i płeć danego pracownika (krotki relacji PRACOWNIK); jeśli wartość atrybutu PESEL tej krotki relacji PRACOWNIK znajduje się w wyniku zagnieżdżonego zapytania, krotka ta jest wybierana do wyniku całego zapytania.

Generalnie, zapytanie zawierające zagnieżdżone bloki select-from-where i wykorzystujące operator porównania = lub IN *zawsze* może być wyrażone w postaci pojedynczego bloku zapytania. Przykładowo, zapytanie Z16 możemy równie dobrze zapisać w następującej formie jako zapytanie Z16A:

```
Z16A: SELECT P.IMIĘ, P.NAZWISKO
      FROM PRACOWNIK AS P, CZŁONEK_RODZINY AS C
      WHERE P.PESEL = C.PESELPRAC AND P.PŁEĆ = C.PŁEĆ
      AND P.IMIĘ = C.IMIĘ_CZŁONKA_RODZINY;
```

### 7.1.4. Dostępne w języku SQL funkcje EXISTS i UNIQUE

EXISTS i UNIQUE to funkcje logiczne zwracające wartości TRUE i FALSE. Można więc używać ich w warunkach klauzuli WHERE. Funkcja EXISTS jest wykorzystywana w języku SQL do sprawdzania, czy wynik skorelowanego zapytania zagnieżdżonego jest *pusty* (nie zawiera żadnych krotek), czy nie. Wynik działania tej funkcji to wartość logiczna TRUE, jeśli wynik zapytania zagnieżdżonego obejmuje choć jedną krotkę. Jeżeli taki wynik nie zawiera krotek, funkcja zwraca FALSE. Przykładowe zastosowania funkcji EXISTS (i NOT EXISTS) zilustrujemy za pomocą kilku zapytań. W pierwszej kolejności zmodyfikujemy zapytanie 16. w taki sposób, aby wykorzystywało właśnie funkcję EXISTS (patrz zapytanie Z16B).

```
Z16B: SELECT P.IMIĘ, P.NAZWISKO
       FROM PRACOWNIK AS P
       WHERE EXISTS (SELECT *
                     FROM CZŁONEK_RODZINY AS C
                     WHERE P.PESEL = C.PESELPRAC AND P.PŁEĆ = C.PŁEĆ
                     AND P.IMIĘ = C.IMIĘ_CZŁONKA_RODZINY);
```

Funkcje EXISTS i NOT EXISTS są zwykle wykorzystywane w połączeniu ze *skorelowanymi* zapytaniami zagnieżdżonymi. W zapytaniu Z16B zapytanie zagnieżdżone odwołuje się do atrybutów PESEL, IMIĘ i PŁEĆ relacji PRACOWNIK zadeklarowanej w zapytaniu zewnętrznym. Całe zapytanie Z16B możemy więc interpretować w następujący sposób: dla każdej krotki relacji PRACOWNIK wyznacz wynik zapytania zagnieżdżonego (które wyszukuje wszystkie krotki relacji CZŁONEK\_RODZINY z takim samym numerem PESEL, płcią oraz imieniem jak w danej krotce relacji PRACOWNIK); jeśli w wyniku wygenerowanym przez zapytanie zagnieżdżone istnieje (EXISTS) przynajmniej jedna krotka, wówczas taka krotka relacji PRACOWNIK jest kwalifikowana do wyniku całego zapytania. Zasadniczo, funkcja EXISTS(Z) zwraca wartość TRUE, jeśli w relacji wynikowej zagnieżdżonego zapytania Z istnieje *przynajmniej jedna krotka*; w przeciwnym przypadku funkcja EXISTS(Z) zwraca wartość FALSE. Z drugiej strony, funkcja NOT EXISTS(Z) zwraca wartość TRUE, jeśli relacja wynikowa zagnieżdżonego zapytania Z nie zawiera *żadnej krotki*; w przeciwnym przypadku funkcja NOT EXISTS(Z) zwraca wartość FALSE. Poniższe zapytanie ilustruje zastosowanie funkcji NOT EXISTS.

**Zapytanie 6.** Pobierz imiona i nazwiska wszystkich pracowników, którzy nie mają żadnych członków rodziny.

```
Z6: SELECT IMIĘ, NAZWISKO
     FROM PRACOWNIK
     WHERE NOT EXISTS (SELECT *
                       FROM CZŁONEK_RODZINY
                       WHERE PESEL = PESELPRAC);
```

W zapytaniu Z6 skorelowane zapytanie zagnieżdżone pobiera wszystkie krotki relacji CZŁONEK\_RODZINY powiązane z konkretną krotką relacji PRACOWNIK. Jeśli takie powiązane krotki *nie istnieją*, dana krotka relacji PRACOWNIK jest pobierana, ponieważ warunek klauzuli WHERE ma wtedy wartość TRUE. Zapytanie Z6 możemy wyjaśnić w następujący sposób: dla *każdej* krotki relacji PRACOWNIK skorelowane zapytanie zagnieżdżone znajduje wszystkie te krotki relacji CZŁONEK\_RODZINY, w których wartość atrybutu PESELPRAC pasuje do wartości atrybutu PESEL danego pracownika; jeśli wynik tego zagnieżdżonego zapytania jest

pusty, wiemy, że dany pracownik nie ma żadnych członków rodziny, zatem reprezentująca go krotka relacji PRACOWNIK jest wybierana do relacji wynikowej (gdzie umieszczamy wartości atrybutów IMIĘ i NAZWISKO tej krotki).

**Zapytanie 7.** Wymień imiona i nazwiska wszystkich kierowników, którzy mają przynajmniej po jednym członku rodziny.

```
Z7:  SELECT IMIĘ, NAZWISKO
      FROM   PRACOWNIK
      WHERE  EXISTS (SELECT *
                     FROM   CZŁONEK_RODZINY
                     WHERE  PESEL = PESELPRAC);

      AND
      EXISTS (SELECT *
              FROM   DZIAŁ
              WHERE  PESEL = PESELKIEROWNIKA);
```

Zapytanie Z7 jest tylko jedną wersją zapytania generującego szukany wynik — użyliśmy dwóch skorelowanych zapytań zagnieżdżonych; pierwsze z nich znajduje wszystkich członków rodziny (krotki relacji CZŁONEK\_RODZINY) powiązane z pracownikami (krotkami relacji PRACOWNIK), drugie wybiera wszystkie działy (krotki relacji DZIAŁ) zarządzane przez danego pracownika. Jeśli istnieje przynajmniej jedna taka krotka w wyniku pierwszego zagnieżdżonego zapytania i jedna krotka w wyniku drugiego zapytania, dana krotka relacji PRACOWNIK jest wybierana. Czy potrafisz zapisać powyższe zapytanie, stosując tylko jedno zapytanie zagnieżdżone lub w ogóle bez takich zapytań.

Zapytanie 3. *(znajdź imiona i nazwiska wszystkich pracowników, którzy bezpośrednio realizują wszystkie projekty nadzorowane przez dział piąty)* można w języku SQL wyrazić z wykorzystaniem funkcji EXISTS oraz NOT EXISTS. Pokażemy tu dwa sposoby zapisu zapytania Z3 w języku SQL: w postaci zapytań Z3A i Z3B. Jest to przykład zapytań wymagających *kwantyfikatora ogólnego* (patrz punkt 8.6.7). Jedną z możliwości to zastosowanie opisanego dalej wyrażenia ( $S2 \text{ EXCEPT } S1$ ) i sprawdzenie, czy wynik jest pusty<sup>1</sup>. To rozwiązanie przedstawiono w zapytaniu Z3A.

```
Z3A:  SELECT IMIĘ, NAZWISKO
      FROM   PRACOWNIK
      WHERE  NOT EXISTS ((SELECT NUMERPROJ
                          FROM   PROJEKT
                          WHERE  NR_DZ = 5)
                        EXCEPT (SELECT NR_PROJ
                                   FROM   PRACUJE_NAD
                                   WHERE  PESEL = PESELPRAC));
```

Pierwsze (nieskorelowane) podzapytanie zapytania Z3A wybiera wszystkie krotki reprezentujące projekty nadzorowane przez dział 5., natomiast drugie (skorelowane) podzapytanie wybiera wszystkie projekty, które są bezpośrednio realizowane przez konkretnego pracownika. Jeśli różnica zbiorów (wyznaczana przez operator MINUS lub EXCEPT) wygenerowanych przez pierwsze i drugie podzapytanie jest pusta, wiadomo, że dany

<sup>1</sup> Pamiętaj, że EXCEPT to operator różnicy zbiorów. Czasem (np. w bazach Oracle) używane jest też słowo kluczowe MINUS.



pracownik bezpośrednio realizuje wszystkie uwzględniane projekty, zatem krotka reprezentująca tego pracownika jest pobierana.

Drugą z możliwości przedstawiono w poniższym zapytaniu Z3B. Warto zauważyć, że potrzebowaliśmy aż dwupoziomowego mechanizmu zagnieżdżenia i że taka postać zapytania jest nieco bardziej skomplikowana od zapytania Z3A.

```
Z3B:  SELECT IMIĘ, NAZWISKO
        FROM PRACOWNIK
        WHERE NOT EXISTS (SELECT *
                           FROM PRACUJE_NAD P
                           WHERE (P.NR_PROJ IN (SELECT NUMERPROJ
                                                  FROM PROJEKT
                                                  WHERE NR_DZ = 5))
                           AND
                           NOT EXISTS (SELECT *
                                       FROM PRACUJE_NAD Q
                                       WHERE Q.PESELPRAC = PESEL
                                       AND Q.NR_PROJEKTU = P.NR_PROJ));
```

Zewnętrzne zapytanie zagnieżdżone zdefiniowane w zapytaniu Z3B wybiera wszystkie krotki relacji PRACUJE\_NAD (P), których wartość atrybutu NR\_PROJ wskazuje na projekt nadzorowany przez dział 5., *pod warunkiem*, że nie istnieje żadna krotka relacji PRACUJE\_NAD (C) z taką samą wartością atrybutu NR\_PROJ i numerem PESEL pracownika reprezentowanego przez daną krotkę skrajnie zewnętrznego zapytania. Przedstawiona postać zapytania Z3B odpowiada następującej interpretacji oryginalnego zapytania 3.: wybierz każdego takiego pracownika, żeby nie istniał projekt nadzorowany przez dział 5., który nie jest przez tego pracownika bezpośrednio realizowany. Takie podejście jest zgodne z techniką stosowaną podczas tworzenia zapytań w relacyjnym rachunku krotek (patrz punkt 8.6.7).

Istnieje inna przydatna funkcja języka SQL, UNIQUE(Z), która zwraca wartość TRUE, jeśli w relacji wynikowej wygenerowanej przez zapytanie Z nie istnieją duplikaty krotek; w przeciwnym przypadku funkcja UNIQUE(Z) zwraca wartość FALSE. Funkcja ta może być wykorzystywana do sprawdzania, czy wynik zagnieżdżonego zapytania jest zbiorem (bez powtórzeń), czy wielozbiorem (z duplikatami).

## 7.1.5. Jawne deklarowanie zbiorów i zmienianie nazw atrybutów w języku SQL

Mieliśmy już okazję przeanalizować wiele zapytań zawierających zapytania zagnieżdżone w klauzuli WHERE. Okazuje się, że w klauzuli WHERE istnieje także możliwość wykorzystywania (zamiast zagnieżdżonych zapytań) **jawnie deklarowanych zbiorów wartości**. W języku SQL taki zbiór należy umieścić w nawiasach.

**Zapytanie 17.** Pobierz numery PESEL wszystkich pracowników, którzy bezpośrednio realizują projekty oznaczone numerami 1, 2 lub 3.

```
Z17:  SELECT DISTINCT PESELPRAC
        FROM PRACUJ_NAD
        WHERE NR_PROJ IN (1, 2, 3);
```

W języku SQL istnieje możliwość **zmiany nazw** atrybutów występujących w relacji wynikowej zapytania przez dodanie kwalifikatora AS bezpośrednio poprzedzającego nową, docelową nazwę. Oznacza to, że wspomniana już konstrukcja AS może być wykorzystywana do deklarowania nazw zastępczych (tzw. aliasów) zarówno dla nazw atrybutów, jak i nazw relacji, oraz że może być stosowana w odpowiednich klauzulach zapytania. Przykładowo, zapytanie Z8A jest nową wersją zapytania Z8 z punktu 4.3.2. Wprowadzając niewielkie zmiany możemy stworzyć zapytanie pobierające nazwiska wszystkich pracowników wraz z ich bezpośrednimi przełożonymi i jednocześnie zmienić wynikowe nazwy atrybutów na odpowiednio NAZWISKO\_PRACOWNIKA oraz NAZWISKO\_PRZEŁOŻONEGO. Nowe nazwy atrybutów będą wyświetlane w postaci nagłówków kolumn relacji wynikowej wygenerowanej przez to zapytanie.

```
Z8A: SELECT E.NAZWISKO AS NAZWISKO_PRACOWNIKA, P.NAZWISKO AS NAZWISKO_PRZEŁOŻONEGO
      FROM PRACOWNIK AS E, PRACOWNIK AS P
      WHERE E.PESELPRZEŁOŻ = P.PESEL;
```

## 7.1.6. Tabele połączone w języku SQL

**Tabele połączone** (lub **relacje połączone**) było początkowo implementowane w języku SQL z myślą o możliwości określania w ramach *klauzuli FROM zapytania* tabeli (relacji) wynikowej generowanej przez operację złączenia. Taka konstrukcja może być łatwiejsza do zrozumienia od prezentowanych do tej pory kombinacji wszystkich warunków selekcji i złączenia wewnątrz klauzuli WHERE. Przykładowo, przeanalizujmy zapytanie Z1, które pobiera imiona i nazwiska oraz adresy wszystkich pracowników zatrudnionych w dziale 'Badania'. Rozwiązanie polegające na określeniu w pierwszej kolejności warunku złączenia relacji PRACOWNIK oraz DZIAŁ i późniejszym zdefiniowaniu warunku selekcji oczekiwanych krotek i atrybutów może się okazać w tym wyrażeniu dużo łatwiejsze. W języku SQL możemy tak zmodyfikowane zapytanie zapisać jako Z1A:

```
Z1A: SELECT IMIĘ, NAZWISKO, ADRES
      FROM (PRACOWNIK JOIN DZIAŁ ON NRDZ = NUMERDZ)
      WHERE NAZWADZ = 'Badania';
```

Użyta w zapytaniu Z1A klauzula FROM zawiera pojedynczą *tabelę połączoną*. Atrybutami takiej tabeli są wszystkie atrybuty pierwszej tabeli (w tym przypadku PRACOWNIK), które poprzedzają wszystkie atrybuty drugiej tabeli (w tym przypadku DZIAŁ). Koncepcja tabeli połączonej dodatkowo daje użytkownikowi możliwość określania różnych typów złączenia, w tym złączenia naturalnego oraz rozmaitych rodzajów złączenia zewnętrznego. W przypadku **złączenia naturalnego** pary relacji *R* i *S* nie określamy dodatkowego warunku złączenia — język SQL niejawnie tworzy *warunek równozłączenia dla każdej pary atrybutów relacji R i S, które mają takie same nazwy*. Każda taka para atrybutów jest dołączana do relacji wynikowej *tylko raz* (więcej informacji o różnych typach złączeń w algebrze relacyjnej znajdziesz w punktach 8.3.2 i 8.4.4).

Jeśli nazwy atrybutów złączenia w relacjach bazowych nie są ze sobą zgodne, istnieje możliwość odpowiedniego dostosowania (zmiany) nazw atrybutów i późniejszego zastosowania operacji złączenia naturalnego. W takim przypadku do zmiany nazw relacji i wszystkich jej atrybutów w ramach klauzuli FROM można wykorzystać konstrukcję AS. Praktyczne zastosowanie takiego rozwiązania zilustrowano w zapytaniu Z1B, gdzie nazwę

relacji DZIAŁ zmieniono na DZ, a nazwy jej atrybutów zmieniono na NAZWADZ, NRDZ (w ten sposób dostosowano tę nazwę do użytego w zapytaniu atrybutu złączenia NRDZ relacji PRACOWNIK), MPESEL oraz MPDATA. Efektem tych zmian jest możliwość zastosowania operacji złączenia naturalnego z warunkiem `PRACOWNIK.NRDZ = DZ.NRDZ`, ponieważ po uwzględnieniu wymienionych powyżej zmian jest to jedyna para atrybutów oznaczonych taką samą nazwą.

```
Z1B: SELECT PIMIEŃ, NAZWISKO, ADRES
      FROM   (PRACOWNIK NATURAL JOIN
              (DZIAŁ AS DZ(NAZWADZ, NRDZ, MPESEL, MPDATA)))
      WHERE  NAZWADZ = 'Badania';
```

Domyślnym typem złączenia danych w postaci tabeli połączonej jest tzw. **złączenie wewnętrzne**, które przewiduje umieszczanie w relacjach wynikowych tylko tych krotek, dla których istnieją odpowiedniki (pasujące krotki) w drugiej relacji złączenia. Przykładowo, w zapytaniu Z8A do relacji wynikowej zostaną zakwalifikowane krotki reprezentujące tylko tych pracowników, którzy *mają przełożonych*; krotki relacji PRACOWNIK zawierające wartość NULL w atrybucie PESELPRZEŁOŻ zostaną w procesie złączania odrzucone. Jeśli użytkownik oczekuje, że w relacji wynikowej zostaną uwzględnione krotki wszystkich pracowników, musi jawnie użyć operacji **złączenia zewnętrznego** (patrz punkt 8.4.4, gdzie przedstawiono definicję tej operacji z algebry relacyjnej). W języku SQL można ten cel osiągnąć, stosując operator OUTER JOIN w tabeli połączonej — patrz poniższe zapytanie Z8B:

```
Z8A: SELECT E.NAZWISKO AS NAZWISKO_PRACOWNIKA,
           P.NAZWISKO AS NAZWISKO_PRZEŁOŻONEGO
      FROM   (PRACOWNIK AS E LEFT OUTER JOIN PRACOWNIK AS P
              ON E.PESELPRZEŁOŻ = P.PESEL);
```

Do dostępnych w języku SQL możliwości w zakresie określania tabel połączonych należy stosowanie następujących operatorów: INNER JOIN (złączenie wewnętrzne, pobierane są tylko pary krotek pasujących do warunku złączenia; w języku SQL równoważne operacji JOIN), LEFT OUTER JOIN (lewe złączenie zewnętrzne, w wyniku znajduje się każda krotka z tabeli podanej po lewej stronie; jeśli nie istnieje pasująca krotka tabeli podanej po prawej stronie, atrybuty z prawej tabeli są uzupełniane wartościami NULL), RIGHT OUTER JOIN (prawe złączenie zewnętrzne, w wyniku znajduje się każda krotka z tabeli podanej po prawej stronie; jeśli nie istnieje pasująca krotka tabeli podanej po lewej stronie, atrybuty z lewej tabeli są uzupełniane wartościami NULL) oraz FULL OUTER JOIN (pełne złączenie zewnętrzne). W przypadku trzech ostatnich opcji słowo kluczowe OUTER może być pomijane. Jeśli atrybuty złączenia są oznaczone takimi samymi nazwami, możemy dodatkowo zastosować złączenie naturalne jako jedną z wersji złączenia zewnętrznego — wystarczy użyć słowa kluczowego NATURAL bezpośrednio przed odpowiednim operatorem (np. NATURAL LEFT OUTER JOIN). Do określania operacji iloczynu kartezjańskiego (patrz punkt 8.2.2) w języku SQL wykorzystuje się słowo kluczowe CROSS JOIN, umieszczając to słowo kluczowe w zapytaniu należy jednak zachować szczególną ostrożność, ponieważ wygenerowana relacja wynikowa będzie zawierała wszystkie możliwe kombinacje krotek.

W języku SQL mamy także możliwość *zagnieżdżania* operacji złączenia tabel — oznacza to, że jedna ze złączonych tabel sama może być tabelą połączoną. W ten sposób można zdefiniować złączenie trzech lub więcej tabel tworzące jedną tabelę połączoną. Jest to **złączenie z udziałem wielu tabel**. Zapytanie Z2A jest jeszcze jedną wersją przedstawionego wcześniej zapytania Z2 z punktu 6.3.1 (zbudowaną właśnie za pomocą połączonej tabeli):

```
Z2A: SELECT NUMERPROJ, NR_DZ, NAZWISKO, ADRES, DATAUR
      FROM ((PROJEKT JOIN DZIAŁ ON NR_DZ = NUMERDZ)
            JOIN PRACOWNIK ON PESELKIEROWNIKA = PESEL)
      WHERE LOKALIZACJAPROJ = 'Gliwice';
```

Nie we wszystkich implementacjach języka SQL zaimplementowana jest nowa składnia tabel połączonych. W niektórych systemach do tworzenia złączeń zewnętrznych stosowana była inna składnia, gdzie do tworzenia lewych, prawych i pełnych złączeń zewnętrznych wykorzystywano w warunku złączenia operatory  $+=$ ,  $=+$  i  $+=+$ . Ta składnia jest dostępna np. w bazach Oracle. Aby za pomocą tej składni zapisać lewe złączenie zewnętrzne z zapytania Z8B, można zmodyfikować je do postaci zapytania Z8C:

```
Z8C: SELECT P.Nazwisko, S.Nazwisko
      FROM PRACOWNIK P, PRACOWNIK S
      WHERE P.PESELPRZEŁOŻ += S.PESEL;
```

### 7.1.7. Funkcje agregujące w języku SQL

**Funkcje agregujące** służą do podsumowywania informacji z wielu krotek za pomocą jednej krotki. **Grupowanie** służy do tworzenia podgrup krotek przed wygenerowaniem podsumowania. Grupowanie i agregacja są potrzebne w wielu aplikacjach baz danych, a w przykładach pokażemy, jak stosować te mechanizmy w języku SQL. Dostępnych jest wiele wbudowanych funkcji, w tym COUNT, SUM, MAX, MIN oraz AVG<sup>2</sup>. Funkcja COUNT zwraca *liczbę krotek lub wartości* znalezionych podczas wykonywania zapytania. Funkcje SUM, MAX, MIN i AVG są stosowane dla zbiorów lub wielozbiorów wartości numerycznych i zwracają odpowiednio sumę, wartość maksymalną, wartość minimalną i średnią arytmetyczną tych wartości. Wymienione funkcje można stosować zarówno w klauzuli SELECT, jak i w klauzuli HAVING (którą wprowadzimy w dalszej części tego rozdziału). Funkcje MAX i MIN mogą być stosowane także dla atrybutów, których wartości należą do dziedzin nienumerycznych, jeśli tylko pomiędzy wartościami tych dziedzin spełniony jest warunek *całkowitego porządku*<sup>3</sup>. Przykładowe zastosowanie tych funkcji zostało zilustrowane w przedstawionych poniżej przykładowych zapytaniach.

**Zapytanie 19.** Wyznacz sumę wynagrodzeń otrzymywanych przez wszystkich pracowników; określ także pensję maksymalną, pensję minimalną i średnie wynagrodzenie w firmie.

```
Z19: SELECT SUM(PENSJA), MAX(PENSJA), MIN(PENSJA), AVG(PENSJA)
      FROM PRACOWNIK;
```

To zapytanie zwraca *jednowierszowe* podsumowanie dotyczące wszystkich wierszy tabeli PRACOWNIK. Można zastosować słowo kluczowe AS, aby zmienić nazwy kolumn w wynikowej jednowierszowej tabeli — np. tak jak w zapytaniu Z19A.

<sup>2</sup> Standard SQL-99 został rozszerzony o dodatkowe funkcje agregujące zaprojektowane z myślą o bardziej zaawansowanych obliczeniach statystycznych.

<sup>3</sup> Całkowity porządek oznacza, że dla każdej pary wartości należących do danej dziedziny można wskazać jedną wartość, która zgodnie ze zdefiniowanym porządkiem występuje przed drugą wartością; przykładowo, warunek całkowitego porządku spełniają zarówno wartości należące do dziedzin DATE, TIME oraz TIMESTAMP, jak i ciągi znakowe (gdzie kolejność zależy od porządku alfabetycznego).

```
Z19A: SELECT SUM (Pensja) AS Suma_pensji, MAX (Pensja) AS Najwyższa_pensja,  
           MIN (Pensja) AS Najniższa_pensja, AVG (Pensja) AS Średnia_pensja  
FROM     PRACOWNIK;
```

Gdybyśmy chcieli użyć powyższych funkcji tylko dla pracowników określonego działu — powiedzmy działu 'Badania' — moglibyśmy zastosować zapytanie 20., w którym zbiór uwzględnianych krotek relacji PRACOWNIK ograniczono za pomocą odpowiedniego warunku klauzuli WHERE wyłącznie do tych reprezentujących pracowników zatrudnionych w dziale 'Badania'.

**Zapytanie 20.** Wyznacz sumę wynagrodzeń otrzymywanych przez wszystkich pracowników działu 'Badania'; oblicz także pensję maksymalną, pensję minimalną i średnie wynagrodzenie osób zatrudnionych w tym dziale.

```
Z20: SELECT SUM(PENSJA), MAX(PENSJA), MIN(PENSJA), AVG(PENSJA)  
FROM      (PRACOWNIK JOIN DZIAŁ ON NRDZ = NUMERDZ)  
WHERE     NAZWADZ = 'Badania';
```

**Zapytanie 21. i 22.** Wyznacz łączną liczbę pracowników zatrudnionych w firmie (Z21) oraz liczbę pracowników zatrudnionych w dziale 'Badania' (Z22).

```
Z21: SELECT COUNT(*)  
FROM    PRACOWNIK;  
Z22: SELECT COUNT(*)  
FROM    PRACOWNIK, DZIAŁ  
WHERE   NRDZ = NUMERDZ AND NAZWADZ = 'Badania';
```

W powyższym przykładzie użyliśmy symbolu gwiazdki (\*), który w tym przypadku reprezentuje wszystkie *wiersze* (krotki), zatem funkcja COUNT(\*) zwraca liczbę wierszy z wyniku zapytania. Funkcja COUNT może posłużyć do zliczenia unikatowych wartości w poszczególnych kolumnach (patrz poniższy przykład), zamiast do zliczania wszystkich krotek.

**Zapytanie 23.** Wyznacz liczbę różnych wartości pensji wypłacanych pracownikom firmy.

```
Z23: SELECT COUNT(DISTINCT PENSJA)  
FROM    PRACOWNIK;
```

Gdybyśmy w zapytaniu Z23 użyli konstrukcji COUNT(PENSJA) zamiast COUNT(DISTINCT PENSJA), powtórzone wartości (wysokości wynagrodzeń) nie zostałyby wyeliminowane. W obu przypadkach krotki z wartością pustą (NULL) atrybutu PENSJA nie zostaną policzone. Wartości NULL są **odrzucone** podczas stosowania dla danej kolumny (atrybutu) funkcji agregujących. Jedynym wyjątkiem jest wyrażenie COUNT(\*), ponieważ zlicza ono krotki, a nie wartości. W poprzednich przykładach wartości pensji równe NULL nie są uwzględniane w obliczeniach w funkcjach agregujących. Gdy funkcja agregująca działa dla kolekcji wartości, zgodnie z ogólną regułą wartości NULL są eliminowane przed wykonaniem obliczeń. Jeśli kolekcja stanie się pusta (ponieważ wszystkie wartości to NULL), funkcja agregująca zwróci NULL. Wyjątkiem jest funkcja COUNT, która dla pustej kolekcji wartości zwraca 0.

Przedstawione powyżej przykłady zapytań podsumowują *całą relację* (zapytania Z19, Z21 i Z23) lub tylko wybrany podzbiór krotek (zapytania Z20 i Z22), zatem każde z tych zapytań zwraca pojedynczą krotkę lub pojedynczą wartość. Przykłady te ilustrują sposób wykorzystywania tych funkcji w celu uzyskiwania wartości lub krotki stanowiącej podsumowanie zawartości wybranego fragmentu bazy danych. Zaprezentowane funkcje

mogą być stosowane także w warunkach selekcji definiowanych dla zagnieżdżonych zapytań. Możemy zdefiniować skorelowane zapytanie zagnieżdżone z funkcją agregującą i następnie użyć tego zagnieżdżonego zapytania w klauzuli WHERE zapytania zewnętrznego. Przykładowo, aby uzyskać imiona i nazwiska wszystkich pracowników mających na utrzymaniu co najmniej dwóch członków rodziny, możemy skonstruować następujące zapytanie:

```
Z5:  SELECT IMIĘ, NAZWISKO
      FROM PRACOWNIK
      WHERE (SELECT COUNT(*)
             FROM CZŁONEK_RODZINY
             WHERE PESEL = PESELPRAC) >= 2;
```

Skorelowane zapytanie zagnieżdżone określa liczbę członków rodziny każdego z pracowników; jeśli uzyskana wartość jest większa lub równa liczbie 2, dana krotka pracownika zostaje uwzględniona w relacji wynikowej.

Język SQL obejmuje też funkcje agregujące SOME i ALL, które można zastosować do kolekcji wartości logicznych. Funkcja SOME zwraca wartość TRUE, jeśli przynajmniej jeden element kolekcji ma wartość TRUE. Funkcja ALL zwraca TRUE, jeżeli wszystkie elementy kolekcji mają tę wartość.

## 7.1.8. Grupowanie: klauzule GROUP BY i HAVING

W wielu przypadkach niezbędne jest zastosowanie funkcji agregujących dla *podgrup krotek danej relacji*, gdzie same podgrupy wyznacza się w oparciu o wartości atrybutów. Przykładowo, sytuacja może wymagać wyznaczenia średniego poziomu wynagrodzeń pracowników *każdego z działów firmy* lub liczby pracowników, którzy bezpośrednio są zaangażowani w realizację *każdego z projektów*. W takich przypadkach konieczne jest **podzielenie** relacji na rozłączne podzbiory (**grupy**) krotek. Każda taka grupa (część) będzie się składała z krotek zawierających identyczną wartość w jednym lub kilku atrybutach, nazywanych **atrybutami** (lub **atrybutem**) **grupowania**. Rozwiązanie oparte na grupowaniu umożliwia nam zastosowanie funkcji agregującej dla każdej z tych grup niezależnie. Język SQL oferuje w tym celu klauzulę GROUP BY. Klauzula ta określa atrybuty grupowania, które powinny występować także w klauzuli SELECT — tylko takie rozwiązanie umożliwi prezentowanie wartości wynikowych wygenerowanych przez funkcję agregującą wraz z wartościami atrybutu lub atrybutów grupujących.

**Zapytanie 24.** Dla każdego działu znajdź jego numer, liczbę zatrudnionych w nim pracowników oraz średni poziom ich wynagrodzeń.

```
Z24: SELECT NRDZ, COUNT(*), AVG(PENSJA)
      FROM PRACOWNIK
      GROUP BY NRDZ;
```

W zapytaniu Z24 krotki relacji PRACOWNIK zostały podzielone na grupy — każda z nowych grup gromadzi krotki z taką samą wartością atrybutu NRDZ. Każda grupa obejmuje więc pracowników z jednego działu. Funkcje COUNT i AVG są stosowane osobno dla każdej wyznaczonej w ten sposób grupy krotek. Łatwo zauważyć, że klauzula SELECT zawiera wyłącznie atrybut grupujący oraz funkcje stosowane dla poszczególnych grup krotek. Efekt wykorzystania mechanizmu grupowania z zapytania Z24 przedstawiono na rysunku 7.1(a) (gdzie zaprezentowano także sam wynik tego zapytania).



(a)

IMIĘ	INICJAŁDRIMIENIA	NAZWISKO	PESEL	...	PENSJA	PESELPRZEŁOŻ	NRDZ
Jan	B	Szewczyk	65010912345		3000	55120834598	5
Franciszek	T	Wieszczycki	55120834598		4000	37111045873	5
Alicja	J	Zalewska	68011932514		2500	41062013258	4
Janina	S	Wojtczak	41062013258	...	4300	37111045873	4
Robert	K	Napierski	62091502054		3800	55120834598	5
Joanna	A	Englert	72073110039		2500	55120834598	5
Albert	W	Janiszewski	69032923149		2500	41062013258	4
Józef	E	Bąk	37111045873		5500	brak	1

NRDZ	COUNT(*)	AVG(PENSJA)
5	4	3325
4	3	3100
1	1	5500

Wyniki zapytania Z24

Grupowanie krotek typu PRACOWNIK według wartości atrybutu NRDZ

(b)

NAZWAPROJ	NUMERPROJ	...	PESELPRAC	NR_PROJ	GODZINY
ProduktX	1		123456789	1	32.5
ProduktX	1		453453453	1	20.0
ProduktY	2		123456789	2	7.5
ProduktY	2		453453453	2	20.0
ProduktY	2		333445555	2	10.0
ProduktZ	3		666884444	3	40.0
ProduktZ	3		333445555	3	10.0
Komputeryzacja	10	...	333445555	10	10.0
Komputeryzacja	10		999887777	10	10.0
Komputeryzacja	10		987987987	10	35.0
Reorganizacja	20		333445555	20	10.0
Reorganizacja	20		987654321	20	15.0
Reorganizacja	20		888665555	20	NULL
ZwiększenieZysków	30		987987987	30	5.0
ZwiększenieZysków	30		987654321	30	20.0
ZwiększenieZysków	30		999887777	30	30.0

Te grupy nie są wybierane na podstawie warunku HAVING z zapytania Z26

Po zastosowaniu klauzuli WHERE, ale przed użyciem klauzuli HAVING

NAZWAPROJ	NUMERPROJ	...	PESELPRAC	NR_PROJ	GODZINY
ProduktY	2		123456789	2	7.5
ProduktY	2		453453453	2	20.0
ProduktY	2		333445555	2	10.0
Komputeryzacja	10		333445555	10	10.0
Komputeryzacja	10	...	999887777	10	10.0
Komputeryzacja	10		987987987	10	35.0
Reorganizacja	20		333445555	20	10.0
Reorganizacja	20		987654321	20	15.0
Reorganizacja	20		888665555	20	NULL
ZwiększenieZysków	30		987987987	30	5.0
ZwiększenieZysków	30		987654321	30	20.0
ZwiększenieZysków	30		999887777	30	30.0

NAZWAPROJ	COUNT(*)
ProduktY	3
Komputeryzacja	3
Reorganizacja	3
ZwiększenieZysków	3

Wynik zapytania Z26 (atrybut NUMERPROJ został pominięty)

Po zastosowaniu warunku z klauzuli HAVING

RYSUNEK 7.1. Wyniki zastosowania klauzul GROUP BY i HAVING: (a) Z24, (b) Z26

Jeśli okaże się, że atrybut grupujący zawiera wartości puste (NULL), wówczas zostanie utworzona **osobna grupa** dla wszystkich krotek zawierających *wartość pustą w atrybucie grupującym*. Przykładowo, gdyby tabela PRACOWNIK zawierała jakieś krotki z wartością NULL dla atrybutu grupującego NRDZ, w wyniku zapytania Z24 zostałyby dla tych krotek stworzona osobna grupa.

**Zapytanie 25.** Dla każdego projektu znajdź jego numer, nazwę oraz liczbę pracowników, którzy są bezpośrednio zaangażowani w jego realizację.



```

Z25:  SELECT  NUMERPROJ, NAZWAPROJ, COUNT(*)
      FROM    PROJEKT, PRACUJE_NAD
      WHERE   NUMERPROJ = NR_PROJ
      GROUP BY NUMERPROJ, NAZWAPROJ;

```

Zapytanie Z25 jest przykładem użycia warunku złączenia razem z klauzulą grupującą GROUP BY. W tym przypadku proces grupowania i wykonania funkcji agregującej odbywa się już *po* złączeniu pary relacji z klauzuli WHERE.

Istnieją jednak sytuacje, w których niezbędne jest pobranie wartości funkcji tylko i wyłącznie dla *grup, które spełniają określone warunki*. Przykładowo, przypuśćmy, że chcemy tak zmodyfikować nasze zapytanie 25., aby w ostatecznym wyniku zostały uwzględnione tylko projekty realizowane przynajmniej przez trzech pracowników. Język SQL oferuje w tym celu klauzulę HAVING, która może występować w połączeniu z klauzulą GROUP BY. Klauzula HAVING powinna zawierać warunki mające zastosowanie dla grupy krotek powiązanych ze sobą w oparciu o każdą wartość atrybutów grupujących. W wyniku zapytania są umieszczane tylko te grupy krotek, które spełniają warunek zdefiniowany w klauzuli HAVING. Przykładowe zastosowanie tego mechanizmu zilustrowano w zapytaniu 26.

**Zapytanie 26.** Dla każdego projektu, *który jest realizowany przez więcej niż dwóch pracowników*, znajdź jego numer, nazwę oraz liczbę pracowników, którzy są bezpośrednio zaangażowani w jego realizację.

```

Z26:  SELECT  NUMERPROJ, NAZWAPROJ, COUNT(*)
      FROM    PROJEKT, PRACUJE_NAD
      WHERE   NUMERPROJ = NR_PROJ
      GROUP BY NUMERPROJ, NAZWAPROJ
      HAVING  COUNT(*) > 2;

```

Nietrudno zauważyć, że o ile zdefiniowane w klauzuli WHERE warunki selekcji ograniczają zbiór *krotek*, dla których zadeklarowane funkcje agregujące są stosowane, o tyle klauzula HAVING pełni podobną rolę w fazie ograniczania zbioru *całych grup*. Na rysunku 7.1(b) przedstawiono zarówno zastosowanie klauzuli HAVING, jak i wynik otrzymany w wyniku zapytania Z26.

**Zapytanie 27.** Dla każdego projektu znajdź jego numer, nazwę oraz liczbę pracowników zatrudnionych w dziale 5., którzy są bezpośrednio zaangażowani w jego realizację.

```

Z27:  SELECT  NUMERPROJ, NAZWAPROJ, COUNT(*)
      FROM    PROJEKT, PRACUJE_NAD, PRACOWNIK
      WHERE   NUMERPROJ = NR_PROJ AND PESEL = PESELPRAC AND NRDZ = 5
      GROUP BY NUMERPROJ, NAZWAPROJ;

```

W zapytaniu 27. ograniczyliśmy zbiór przeszukiwanych krotek (a więc także zbiór krotek przydzielanych do poszczególnych grup) do tych wierszy, które spełniają warunek określony w klauzuli WHERE — czyli wyłącznie do pracowników zatrudnionych w dziale 5. Warto pamiętać, że podczas wprowadzania dwóch różnych warunków (jednego dla funkcji zadeklarowanej w klauzuli SELECT i drugiego dla funkcji określonej w klauzuli HAVING) musimy zachować szczególną ostrożność. Przykładowo, przypuśćmy, że dla poszczególnych działów firmy chcemy wyznaczyć *łącną* liczbę pracowników, których miesięczna pensja przekracza 4000 złotych, ale nasze obliczenia mają dotyczyć tylko działów zatrudniających więcej niż pięciu pracowników. W takim przypadku warunek (PENSJA >

4000) ma zastosowanie wyłącznie dla funkcji COUNT użytej w klauzuli SELECT. Przypuśćmy, że spróbujemy zrealizować to zadanie za pomocą poniższego, *błędnego* zapytania:

```
SELECT  NRDZ, COUNT(*)
FROM    PRACOWNIK
WHERE   PENSJA > 4000
GROUP BY NRDZ
HAVING  COUNT(*) > 5;
```

Powyższe zapytanie jest nieprawidłowe, ponieważ na jego podstawie do relacji wynikowej zostaną zakwalifikowane tylko te działy, które zatrudniają więcej niż pięciu pracowników *zarabiających miesięcznie ponad 4000 złotych*. Wynika to z faktu, że klauzula WHERE, która selekcjonuje poszczególne krotki, jest wykonywana jako pierwsza; natomiast klauzula HAVING, która selekcjonuje poszczególne grupy krotek, jest stosowana później. Oznacza to, że przeszukiwany zbiór krotek jest ograniczany do pracowników zarabiających ponad 4000 złotych, *zanim* zostanie zastosowana funkcja w klauzuli HAVING. Jednym ze sposobów rozwiązania tego problemu i skonstruowania prawidłowego zapytania jest użycie zapytania zagnieżdżonego (patrz zapytanie 28.).

**Zapytanie 28.** Dla każdego działu, który zatrudnia więcej niż pięciu pracowników, znajdź numer działu i liczbę jego pracowników, którzy zarabiają miesięcznie więcej niż 4000 złotych.

```
Z28:  SELECT  NRDZ, COUNT(*)
        FROM    PRACOWNIK
        WHERE   PENSJA > 4000 AND NRDZ IN
              (SELECT  NRDZ
               FROM    PRACOWNIK
               GROUP BY NRDZ
               HAVING  COUNT(*) > 5)
        GROUP BY NRDZ;
```

### 7.1.9. Inne konstrukcje języka SQL: WITH i CASE

W tym punkcie przedstawimy dwie dodatkowe konstrukcje języka SQL. Klauzula WITH umożliwia zdefiniowanie tabeli używanej tylko na potrzeby konkretnego zapytania. Ten proces przypomina nieco tworzenie perspektyw (patrz podrozdział 7.3) używanych tylko w jednym zapytaniu, a następnie usuwanych. Ta klauzula została wprowadzona jako udogodnienie w standardzie SQL:99 i nie musi być dostępna we wszystkich SZBD opartych na języku SQL. Zapytania z klauzulą WITH zwykle można zapisać także za pomocą innych konstrukcji języka SQL. Aby pokazać przykład użycia klauzuli WITH, zmodyfikujmy zapytanie Z28 do postaci zapytania Z28a:

```
Z28a:  WITH  DUZEDZIAŁY (NRDZ) AS
        (SELECT  NRDZ
         FROM    PRACOWNIK
         GROUP BY NRDZ
         HAVING  COUNT(*) >5)
        SELECT  NRDZ, COUNT(*)
        FROM    PRACOWNIK
        WHERE   Pensja>40000 AND NRDZ IN DUZEDZIAŁY
        GROUP BY NRDZ;
```

W zapytaniu Z28' za pomocą klauzuli WITH zdefiniowaliśmy tymczasową tabelę DUŻEDZIAŁY, zawierającą numery działów zatrudniających więcej niż pięciu pracowników. Ta tabela jest używana w dalszym zapytaniu, a po jego wykonaniu zostaje usunięta.

W języku SQL dostępna jest też konstrukcja CASE, której można użyć, jeśli wartość powinna zależeć od określonych warunków. Tę konstrukcję można zastosować w dowolnej części zapytania w języku SQL, gdzie oczekiwana jest wartość — w tym w zapytaniach i do wstawiania lub aktualizowania krotek. Pokażemy to na przykładzie. Założmy, że chcemy przyznać pracownikom różne podwyżki w zależności od tego, dla którego działu pracują. Przykładowo, pracownicy z działu piątego otrzymają po 200 złotych podwyżki, osoby z działu czwartego — po 150 złotych, a z działu pierwszego — po 300 złotych (krotki reprezentujące pracowników przedstawia rysunek 5.6). Następnie można zmodyfikować operację aktualizacji A6 z punktu 6.4.3 do postaci instrukcji A6':

```
A60:  UPDATE  PRACOWNIK
      SET     PENSJA =
      CASE    WHEN  NRDZ =5  THEN PENSJA + 200
              WHEN  NRDZ =4  THEN PENSJA + 150
              WHEN  NRDZ =1  THEN PENSJA + 300
              ELSE  PENSJA + 0;
```

W zapytaniu A6' wysokość podwyżki jest określana za pomocą konstrukcji CASE na podstawie numerów działów, w których są zatrudnieni pracownicy. Konstrukcję CASE można stosować także do wstawiania krotek, w których różne atrybuty mogą mieć wartość NULL w zależności od typu rekordu zapisywanego w tabeli — np. w sytuacji, gdy zbiór podklas (patrz rozdział 4.) jest odwzorowywany do postaci jednej tabeli (patrz rozdział 9.) lub gdy typ unii jest odwzorowywany do postaci relacji.

## 7.1.10. Zapytania rekurencyjne w języku SQL

W tym punkcie pokażemy, jak pisać zapytania rekurencyjne w języku SQL. Ta składnia została dodana w standardzie SQL:99, aby umożliwić użytkownikom tworzenie zapytań rekurencyjnych w deklaracyjny sposób. Przykładem **rekurencyjnej relacji** między krotkami tego samego typu jest relacja między pracownikiem a przełożonym. Na rysunkach 5.5 i 5.6 jest ona przedstawiona za pomocą klucza obcego PESELPRZEŁOŻ. Relacja ta łączy każdą krotkę pracownika (w roli podwładnego) z inną krotką pracownika (w roli przełożonego). Przykładem operacji rekurencyjnej jest pobranie wszystkich podwładnych przełożonego *e* z wszystkich poziomów, a więc wszystkich pracowników *e'* bezpośrednio nadzorowanych przez *e*, wszystkich pracowników *e''* nadzorowanych przez *e'*, wszystkich pracowników *e'''* nadzorowanych przez *e''* itd. W standardzie SQL:99 to zapytanie można zapisać tak:

```
Z29:  WITH RECURSIVE  PODWŁ_PRZEŁ (PESELPRZEŁ, PESELPRAC) AS
      (SELECT          PESELPRZEŁOŻ, PESEL
      FROM            PRACOWNIK
      UNION
      SELECT          E.PESEL, S.PESELPRZEŁ
      FROM            PRACOWNIK AS E, PODWŁ_PRZEŁ AS S
      WHERE           E.PESELPRZEŁOŻ = S.PESELPRAC)
      SELECT*
      FROM            PODWŁ_PRZEŁ;
```

W zapytaniu Z29 zdefiniowana została perspektywa `PODWŁ_PRZEŁ`, gdzie zapisywany jest wynik przedstawionego rekurencyjnego zapytania. Początkowo ta perspektywa jest pusta. Najpierw zostaje zapełniona kombinacjami numerów PESEL z pierwszego poziomu par (przełożony, podwładny) za pomocą pierwszej części zapytania (`SELECT PESELPRZEŁOŻ, PESEL FROM PRACOWNIK`), nazywanej **zapytaniem bazowym**. Te dane są łączone za pomocą instrukcji `UNION` z danymi podwładnych z następnych poziomów za pomocą drugiej części zapytania, gdzie zawartość perspektywy jest ponownie złączana z wartościami bazowymi, aby uzyskać kombinacje z drugiego poziomu. Te dane są sumowane z danymi z pierwszego poziomu (za pomocą instrukcji `UNION`). Proces ten jest powtarzany na następnych poziomach do czasu osiągnięcia punktu stałego, kiedy do perspektywy nie są już dodawane dalsze krotki. Na tym etapie wynik tego rekurencyjnego zapytania jest zapisany w perspektywie `PODWŁ_PRZEŁ`.

### 7.1.11. Omówienie i podsumowanie zapytań języka SQL

Zapytanie zdefiniowane w języku SQL może się składać z maksymalnie sześciu klauzul, ale tylko dwie pierwsze (`SELECT` i `FROM`) są wymagane. Zapytanie może obejmować wiele wierszy i kończy się średnikiem. Elementy zapytania są rozdzielone odstępami, a do grupowania odpowiednich fragmentów zapytania można stosować nawiasy. Klauzule są deklarowane w następującej kolejności (nawiasami kwadratowymi oznaczono klauzule opcjonalne):

```
SELECT    <LISTA ATRYBUTÓW I FUNKCJI>
FROM      <LISTA TABEL>
[WHERE    <WARUNEK>]
[GROUP BY <ATRYBUT (LUB ATRYBUTY) GRUPOWANIA>]
[HAVING   <WARUNEK GRUPOWANIA>]
[ORDER BY <LISTA ATRYBUTÓW>]
```

Klauzula `SELECT` zawiera listę atrybutów lub funkcji, które mają się znaleźć w wyniku zapytania. Klauzula `FROM` określa wszystkie relacje (tabele) niezbędne do wykonania danego zapytania, włącznie z relacjami połączonymi, ale z wyłączeniem relacji wykorzystywanych w zapytaniach zagnieżdżonych. Klauzula `WHERE` określa warunki selekcji krotek z relacji wymienionych w klauzuli `FROM`, włącznie z warunkami złączenia, jeśli są potrzebne. Klauzula `GROUP BY` określa atrybuty grupowania, natomiast klauzula `HAVING` definiuje warunek dla wyselekcjonowanych grup (zamiast dla pojedynczych krotek). Wbudowane funkcje agregujące `COUNT`, `SUM`, `MIN`, `MAX` i `AVG` nie tylko są wykorzystywane w połączeniu z grupowaniem, ale także mogą być stosowane dla wszystkich krotek wyselekcjonowanych przez zapytanie bez klauzuli `GROUP BY`. I wreszcie klauzula `ORDER BY` określa kolejność wyświetlania krotek w relacji wynikowej zapytania.

Aby poprawnie formułować zapytania, warto rozważyć kroki definiujące *znaczenie* (*semantykę*) każdego zapytania. *Koncepcyjnie*<sup>4</sup> wynik zapytania jest wyznaczany przez zastosowanie w pierwszej kolejności klauzuli `FROM` (celem zidentyfikowania wszystkich tabel niezbędnych do wykonania zapytania lub zmaterializowania ewentualnych tabel połączonych), bezpośrednio potem stosowane są warunki selekcji i złączania krotek

---

<sup>4</sup> Faktyczna kolejność wykonywania poszczególnych operacji składających się na przetwarzanie zapytania zależy od implementacji; w tym miejscu prezentujemy jedynie metodę koncepcyjnej analizy zapytań, która może ułatwić ich prawidłowe konstruowanie.

zadeklarowane w klauzuli `WHERE`, w następnej klauzuli `GROUP BY` i `HAVING`. Konceptyjnie klauzula `ORDER BY` jest stosowana na samym końcu, kiedy ostateczny wynik zapytania podlega sortowaniu. Jeśli w zapytaniu nie zostanie użyta żadna z trzech ostatnich klauzul (`GROUP BY`, `HAVING` oraz `ORDER BY`), możemy przyjąć, że takie zapytanie jest *na poziomie koncepcyjnym* wykonywane zgodnie z następującą procedurą: Dla *każdej kombinacji krotek* — łączącej po jednej krotce z każdej z relacji wymienionych w klauzuli `FROM` — wyznaczana jest klauzula `WHERE`; jeśli jej wynik jest prawdziwy (otrzymaliśmy wartość `TRUE`), przechowywane w tej krotce wartości atrybutów określonych w klauzuli `SELECT` są umieszczane w relacji wynikowej tego zapytania. Nie jest to oczywiście najbardziej efektywny sposób implementacji zapytań w rzeczywistych systemach — każdy system zarządzania bazą danych oferuje specjalne metody optymalizacji zapytań, które pozwalają wybrać efektywny plan wykonywania zapytania. Problemy przetwarzania zapytań i optymalizacji tego procesu omówimy w rozdziałach 18. i 19.

Ogólnie rzecz biorąc, w języku SQL istnieje wiele sposobów wyrażania tego samego zapytania. Ta elastyczność w definiowaniu zapytań ma zarówno swoje zalety, jak i wady. Główną zaletą jest możliwość wyboru przez użytkownika takiej techniki konstruowania zapytań, w której jest najbardziej biegły. Przykładowo, wiele zapytań może być wyrażanych z warunkiem złączenia definiowanym w klauzuli `WHERE`, przez zastosowanie relacji (tabel) połączonych w klauzuli `FROM` lub z użyciem jakiejś postaci zagnieżdżonych zapytań i operatora porównania `IN`. Niektórzy użytkownicy preferują jedną technikę, inni wolą budować swoje zapytania stosując zupełnie inne rozwiązania. Z punktu widzenia programisty i samego systemu bazy danych lepszym rozwiązaniem jest generalnie pisanie zapytań zawierających jak najmniej zagnieżdżonych konstrukcji i operacji sortujących — optymalizacja zapytania jest w takim przypadku ułatwiona.

Negatywnym efektem możliwości wyrażania tego samego zapytania na wiele sposobów jest możliwe zagubienie użytkowników, którzy mogą nie wiedzieć, która technika jest najwłaściwsza w przypadku konkretnych typów zapytań. Kolejnym problemem jest fakt, że wykonanie tego samego zapytania wyrażonego w jeden sposób może się okazać bardziej efektywne od wykonania jego alternatywnej postaci. Idealnym rozwiązaniem byłaby oczywiście sytuacja, w której system zarządzania bazą danych wykonywałby to samo zapytanie w taki sam sposób, niezależnie od techniki zastosowanej podczas jego definiowania. W praktyce jest to trudne, ponieważ każdy system zarządzania bazą danych stosuje inne metody przetwarzania zapytań konstruowanych w różny sposób. Oznacza to, że na użytkowników tych systemów spada dodatkowy ciężar określania, która z dostępnych specyfikacji zapytań jest najbardziej efektywna. Byłoby oczywiście najlepiej, gdyby użytkownik musiał się martwić wyłącznie o prawidłowość tworzonych przez siebie zapytań — odpowiedzialność za ich efektywne wykonywanie spadłaby wówczas na system zarządzania bazą danych. W praktyce jednak wiedza użytkownika na temat rodzajów konstrukcji, których przetwarzanie jest bardziej kosztowne od pozostałych, często okazuje się przydatna.

## 7.2. Definiowanie ograniczeń w postaci asercji i działań w postaci wyzwalaczy

W tym podrozdziale przedstawiamy dwie dodatkowe funkcje języka SQL: instrukcje `CREATE ASSERTION` i `CREATE TRIGGER`. W punkcie 7.2.1 opisana jest instrukcja `CREATE ASSERTION`, którą można zastosować do tworzenia dodatkowych rodzajów ograniczeń wykraczających poza *wbudowane ograniczenia modelu relacyjnego* (kluczy głównych i unikatowych, integralności encji oraz integralności odwołań) przedstawione w podrozdziale 5.2. Ograniczenia wbudowane można tworzyć w instrukcji `CREATE TABLE` języka SQL (patrz podrozdział 6.1 i 6.2).

W punkcie 7.2.2 opisujemy instrukcję `CREATE TRIGGER`, stosowaną do tworzenia automatycznych działań, które system bazy danych wykonuje w reakcji na określone zdarzenia i warunki. Mechanizmy tego rodzaju są nazywane **aktywnymi bazami danych**. W tym rozdziale przedstawiamy tylko podstawy **wyzwalaczy**, a bardziej szczegółowe omówienie aktywnych baz danych znajdziesz w podrozdziale 26.1.

### 7.2.1. Definiowanie ogólnych ograniczeń w postaci asercji w języku SQL

W języku SQL użytkownicy mogą definiować ogólne ograniczenia, czyli takie, których nie można zakwalifikować do żadnej z kategorii opisanych w podrozdziale 6.1 lub 6.2 — służą do tego tzw. **deklaratywne asercje** tworzone za pomocą polecenia `CREATE ASSERTION`. Każda asercja ma przypisaną nazwę ograniczenia i zawiera warunek podobny do tych, które stosowaliśmy w klauzuli `WHERE` zapytań języka SQL. Przykładowo, do zdefiniowania w języku SQL ograniczenia w postaci: *pensja pracownika nie może być większa od pensji kierownika działu, w którym ten pracownik jest zatrudniony* możemy użyć następującej asercji:

```
CREATE ASSERTION OGRANICZENIE_PENSJI
CHECK (NOT EXISTS (SELECT *
                    FROM   PRACOWNIK E, PRACOWNIK M,
                           DZIAŁ D
                    WHERE  E.PENSJA > M.PENSJA
                           AND E.NRDZ = D.NUMERDZ
                           AND D.PESELKIEROWNIKA = M.PESEL));
```

Bezpośrednio za zdefiniowaną powyżej nazwą ograniczenia `OGRANICZENIE_PENSJI` umieszczono słowo kluczowe `CHECK`, po którym (w nawiasach) skonstruowano właściwy **warunek**, który musi być prawdziwy dla każdego stanu bazy danych, w którym asercja ma być spełniona. Nazwa ograniczenia może być wykorzystywana później do wyłączenia go, modyfikowania lub usuwania. System zarządzania bazą danych odpowiada za zapewnienie, że taki warunek nie zostanie naruszony w żadnym stanie bazy danych. W wyrażeniach podobnych do powyższego można stosować dowolne klauzule `WHERE`, jednak w wielu przypadkach wystarczy zastosowanie warunków języka SQL opartych na funkcjach `EXISTS` i `NOT EXISTS`. Gdy jakieś krotki w bazie danych powodują, że wyznaczona wartość warunku wyrażenia `ASSERTION` jest równa `FALSE`, ograniczenie zostaje **naruszone**. To samo ograniczenie jest **spełnione** przez stan bazy danych, jeśli w tym stanie bazy danych *nie istnieje kombinacja krotek*, która narusza to ograniczenie.



Podstawowa technika stosowana podczas tworzenia tego typu asercji polega na skonstruowaniu zapytania, które wybierze wszystkie krotki *naruszające docelowy warunek* ograniczenia. Umieszczając to zapytanie wewnątrz klauzuli NOT EXISTS powodujemy, że wynik zapytania musi być pusty, aby warunek miał wartość TRUE. Oznacza to, że dana asercja zostanie naruszona, jeśli wynik tego zapytania nie będzie pusty. W przykładzie zbudowane zapytanie wybiera wszystkich pracowników, których pensje przewyższają wynagrodzenia kierowników zatrudniających ich działów. Jeśli wynik tego zapytania nie będzie pusty, asercja będzie naruszona.

Warto pamiętać, że klauzula CHECK i warunek ograniczenia mogą być stosowane także do definiowania więzów dla *poszczególnych* atrybutów i dziedzin wartości (patrz punkt 6.2.1) oraz *poszczególnych* krotek (patrz punkt 6.2.4). Główna różnica pomiędzy poleceniem CREATE ASSERTION a pozostałymi dwoma rodzajami ograniczeń wynika z faktu, że klauzule CHECK nałożone na pojedyncze atrybuty, dziedziny i krotki są sprawdzane w języku SQL *tylko w chwili wstawiania lub aktualizowania krotek*. Oznacza to, że w przypadku tak zdefiniowanych ograniczeń dla atrybutów, dziedzin i krotek mechanizm sprawdzania może być znacznie wydajniej zaimplementowany w systemie zarządzania bazą danych. Projektant schematu powinien stosować klauzulę CHECK dla atrybutów, dziedzin i krotek tylko wtedy, gdy jest absolutnie pewien, że zdefiniowane w ten sposób ograniczenie *może być naruszane tylko podczas wstawiania lub aktualizacji krotek*. Z drugiej strony projektant schematu powinien stosować polecenie CREATE ASSERTION tylko w sytuacjach, kiedy skuteczne użycie klauzuli CHECK dla atrybutów, dziedzin lub krotek (i tym samym wykorzystanie mechanizmów wydajniej implementowanych przez system zarządzania bazą danych) nie jest możliwe.

## 7.2.2. Wprowadzenie do wyzwalaczy w języku SQL

Innym ważnym poleceniem języka SQL jest instrukcja CREATE TRIGGER. W wielu przypadkach wygodnym rozwiązaniem jest zdefiniowanie typu akcji (działania) podejmowanej w razie wystąpienia określonych zdarzeń lub w przypadku spełnienia określonych warunków. Przykładowo, w niektórych sytuacjach korzystnym rozwiązaniem jest tylko poinformowanie użytkownika o zaistniałym naruszeniu ograniczeń. Kierownik działu może np. oczekiwać informacji o zdarzeniu polegającym na przekroczeniu przez jakiegoś użytkownika górnej granicy wydatków podczas podróży służbowej. W takim przypadku właściwym działaniem systemu zarządzania bazą danych powinno być wysłanie odpowiedniego komunikatu do zainteresowanego użytkownika. Zdefiniowany warunek pełni wówczas rolę **monitora** informacji zawartych w bazie danych. Warunkom można przypisywać także inne działania, w tym wykonywanie określonych *procedur składowanych* lub wyzwalanie innych operacji aktualizujących. Do implementowania takich działań (akcji) w języku SQL służy polecenie CREATE TRIGGER. Wyzwalacze omówimy bardziej szczegółowo w podrozdziale 26.1, kiedy będziemy opisywali *aktywne bazy danych*. Tu przedstawiamy tylko prosty przykład zastosowania wyzwalaczy.

Załóżmy, że chcemy sprawdzać, czy pensja pracownika jest większa od wynagrodzenia jego bezpośredniego przełożonego z bazy danych FIRMA (patrz rysunki 5.5 i 5.6). Ta reguła może zostać uruchomiona przez kilka zdarzeń: wstawienie nowego rekordu pracownika, modyfikację pensji pracownika lub zmianę przełożonego danej osoby. Załóżmy, że podejmowanym działaniem jest wywołanie zewnętrznej procedury składowanej



POWIADOM\_PRZEŁOŻ<sup>5</sup>, która prześle powiadomienie przełożonemu. Wyzwalacz można zapisać tak jak w instrukcji R5 poniżej. Zastosowaliśmy tu składnię z systemu bazy danych Oracle.

```
R5: CREATE TRIGGER NARUSZENIE_PENSJA
  BEFORE INSERT OR UPDATE OF PENSJA, PESEL_PRZEŁOŻ
  ON PRACOWNIK
  FOR EACH ROW
  WHEN (NEW.PENSJA > (SELECT PENSJA FROM PRACOWNIK
                        WHERE PESEL=NEW.PESEL_PRZEŁOŻ))
  POWIADOM_PRZEŁOŻ(NEW.PESEL_PRZEŁOŻ,
                    NEW.PESEL);
```

Wyzwalacz ma tu nazwę NARUSZENIE\_PENSJA, która pozwala później usunąć lub zdezaktywować ten wyzwalacz. Typowy wyzwalacz typu „zdarzenie, warunek, akcja” (ZWA) obejmuje trzy komponenty:

- (1) **Zdarzenia.** Są to zwykle operacje aktualizacji bezpośrednio przeprowadzane na bazie. W tym przykładzie zdarzeniami są: wstawienie nowego rekordu pracownika, zmiana wynagrodzenia pracownika lub zmiana przełożonego. Osoba pisząca wyzwalacz musi mieć pewność, że uwzględniła wszystkie możliwe zdarzenia. W niektórych sytuacjach konieczne jest napisanie kilku wyzwalaczy w celu ujęcia każdego scenariusza. W przykładzie zdarzenia są powiązane ze słowem kluczowym BEFORE, co oznacza, że wyzwalacz powinien zostać wykonany przed operacją wyzwalającą. Inna możliwość to zastosowanie słowa kluczowego AFTER, które powoduje, że wyzwalacz powinien zostać wykonany po zakończeniu operacji ze zdarzenia.
- (2) **Warunek.** Określa on, czy akcja reguły ma być wykonana — po zaistnieniu zdarzenia sprawdzany jest *opcjonalny* warunek. Jeżeli warunek *nie jest zdefiniowany*, to akcja zostanie wykonana bezwarunkowo. Jeżeli warunek jest zdefiniowany, to akcja zostanie wykonana tylko wtedy, gdy *jest on spełniony*. Warunek podawany jest w klauzuli WHEN wyzwalacza.
- (3) **Wykonywana akcja.** Akcja jest zazwyczaj ciągiem wyrażeń języka SQL, ale może to być również transakcja bazy danych lub wykonanie zewnętrznego programu, które ma nastąpić automatycznie. W przedstawionym przykładzie akcją jest wykonanie procedury składowanej POWIADOM\_PRZEŁOŻ.

Wyzwalacze mają wiele zastosowań, np.: w obszarze utrzymywania spójności bazy danych, monitorowania aktualizacji bazy i automatycznego aktualizowania danych pochodnych. Kompletnie omówienie wyzwalaczy zawiera podrozdział 26.1.

---

<sup>5</sup> Przy założeniu, że odpowiednia procedura zewnętrzna została zadeklarowana. Procedury składowane są opisane w rozdziale 10.

## 7.3. Perspektywy (tabele wirtualne) w języku SQL

W tym podrozdziale wprowadzimy stosowane w języku SQL pojęcie perspektywy. Następnie przedstawimy sposób definiowania perspektyw, omówimy problem aktualizowania perspektyw oraz metody ich implementowania w systemach zarządzania bazami danych.

### 7.3.1. Pojęcie perspektywy w języku SQL

W terminologii języka SQL **perspektywa** jest pojedynczą tabelą wywiedzioną z innych tabel<sup>6</sup>. Przez inne tabele rozumiemy zarówno *tabele bazowe*, jak i zdefiniowane wcześniej perspektywy. Perspektywa nie musi koniecznie mieć postaci fizycznej — często jest nawet nazywana **tabelą wirtualną** (a więc przeciwieństwem **tabel bazowych**, których krotki muszą być fizycznie składowane w bazie danych). Ta własność perspektyw ogranicza co prawda możliwość stosowania dla nich operacji aktualizujących, ale w żaden sposób nie wpływa na możliwości wykonywania zapytań na perspektywach.

Perspektywę możemy traktować jak mechanizm definiowania tabeli z informacjami, do których często musimy się odwoływać w naszych zapytaniach, chociaż taka tabela wcale nie musi fizycznie istnieć. Przykładowo, na bazie danych przedstawionej na rysunku 5.5 możemy często wykonywać zapytania dotyczące nazwisk pracowników i nazw realizowanych przez nich projektów. Zamiast wielokrotnego (w każdym takim zapytaniu) definiowania operacji złączenia tabel PRACOWNIK, PRACUJE\_NAD i PROJEKT, możemy stworzyć perspektywę zawierającą wynik takiego złączenia. Możemy wówczas kierować nasze zapytania do tej perspektywy — zapytania te będą w takim przypadku dotyczyły tylko jednej tabeli, zamiast trzech tabel wymagających wykonywania dwóch operacji złączenia. Relacje PRACOWNIK, PRACUJE\_NAD i PROJEKT nazywamy **tabelami definiującymi** naszą perspektywę.

### 7.3.2. Definiowanie perspektyw w języku SQL

Do tworzenia perspektyw w języku SQL służy polecenie `CREATE VIEW`. Każda nowa perspektywa ma nadawaną nazwę (wirtualnej) tabeli (nazwę perspektywy), listę nazw atrybutów oraz zapytanie określające zawartość perspektywy. Jeśli żaden z atrybutów perspektywy nie jest wynikiem stosowania funkcji ani operatorów arytmetycznych, nazwy atrybutów w poleceniu `CREATE VIEW` możemy pominąć, ponieważ w domyślnym przypadku nie różnią się one od nazw atrybutów tabel definiujących. Przedstawione poniżej polecenia P1 i P2 tworzą tabele wirtualne, których schematy zilustrowano na rysunku 7.2 — obie perspektywy wykorzystują schemat bazy danych z rysunku 5.5.

```
P1: CREATE VIEW PRACUJE_NAD1
    AS SELECT IMIĘ, NAZWISKO, NAZWAPROJ, GODZINY
    FROM PRACOWNIK, PROJEKT, PRACUJE_NAD
    WHERE PESEL = PESELPRAC AND NR_PROJ = NUMERPROJ;
```

---

<sup>6</sup> Stosowane w języku SQL pojęcie *perspektywy* jest nieco węższe od omówionego w rozdziałach 1. i 2. terminu *perspektywy użytkownika*, ponieważ perspektywa użytkownika może teoretycznie obejmować więcej niż jedną relację.

```
P2: CREATE VIEW INFO_O_DZIAŁACH(NAZWA_DZIAŁU, LICZBA_PRAC, SUMA_PENSJI)
AS SELECT NAZWADZ, COUNT(*), SUM(PENSJA)
FROM DZIAŁ, PRACOWNIK
WHERE NUMERDZ = NRDZ
GROUP BY NAZWADZ;
```

**PRACUJE\_NAD1**

IMIĘ	NAZWISKO	NAZWAPROJ	GODZINY
------	----------	-----------	---------

**INFO\_O\_DZIAŁACH**

NAZWA_DZIAŁU	LICZBA_PRAC	SUMA_PENSJI
--------------	-------------	-------------

RYСУNEK 7.2. Dwie perspektywy stworzone dla schematu bazy danych z rysunku 5.5

W poleceniu P1 nie określamy (choć moglibyśmy to zrobić) żadnych nowych nazw atrybutów dla tworzonej perspektywy `PRACUJE_NAD1`; w tym przypadku perspektywa `PRACUJE_NAD1` *dziedziczy* nazwy atrybutów ze swoich tabel definiujących `PRACOWNIK`, `PROJEKT` i `PRACUJE_NAD`. W poleceniu P2 tworzącym perspektywę `INFO_O_DZIAŁACH` jawnie określono nowe nazwy atrybutów, stosując zależność jeden-do-jednego pomiędzy atrybutami wymienionymi w klauzuli `CREATE VIEW` oraz atrybutami wyszczególnionymi w klauzuli `SELECT` zapytania definiującego tę perspektywę.

Możemy teraz budować zapytania języka SQL dla tych perspektyw (wirtualnych tabel) — robimy to w taki sam sposób, jak konstruowaliśmy zapytania dla tabel bazowych. Przykładowo, do uzyskania imion i nazwisk wszystkich pracowników, którzy są bezpośrednio zaangażowani w realizację projektu nazwanego 'ProjektX', możemy użyć następującego zapytania ZP1 wykonywanego na naszej nowej perspektywie `PRACUJE_NAD1`:

```
ZP1: SELECT IMIĘ, NAZWISKO
FROM PRACUJE_NAD1
WHERE NAZWAPROJ = 'ProjektX';
```

To samo zapytanie wymagałoby określenia aż dwóch operacji złączenia tabel, gdybyśmy chcieli je wykonać na odpowiednich relacjach bazowych — jedną z głównych zalet perspektyw jest właśnie znaczne uproszczenie pewnych zapytań. Perspektywy są także wykorzystywane w roli mechanizmu zabezpieczającego i ułatwiającego autoryzację (patrz punkt 7.3.4 i rozdział 30.).

Użytkownicy oczekują, że informacje zawarte w wykorzystywanych przez nich perspektywach *zawsze będą aktualne*; jeśli zmodyfikujemy krotki w tabelach bazowych, na podstawie których zdefiniowano perspektywy, także zawartość tych perspektyw powinna zostać automatycznie zaktualizowana. Oznacza to, że perspektywa nie jest realizowana w czasie wykonywania *polecenia, które ją definiuje*, ale w czasie wykonywania *zapytania* wykorzystującego zawarte w niej dane. Za zagwarantowanie zgodności danych zawartych w perspektywie z danymi przechowywanymi w tabelach bazowych zawsze odpowiada system zarządzania bazą danych, nie użytkownik. Różne sposoby zapewniania aktualności perspektyw przez SZBD omawiamy w następnym punkcie.

Jeśli jakaś perspektywa nie jest nam już potrzebna, możemy użyć polecenia `DROP VIEW` do jej usunięcia. Przykładowo, do usunięcia z bazy danych perspektywy utworzonej za pomocą polecenia `P1A` możemy użyć zdefiniowanego w języku SQL polecenia `P1A`:

```
P1A: DROP VIEW PRACUJE_NAD1;
```

### 7.3.3. Implementacja perspektyw i mechanizm ich aktualizowania

Okazuje się, że problem efektywnego implementowania perspektyw wykorzystywanych w zapytaniach jest dosyć skomplikowany. Do tej pory zaproponowano dwa główne rozwiązania w tym zakresie. Pierwsza strategia (nazwana **modyfikacją zapytania**) przewiduje przekształcanie zapytań kierowanych przez użytkownika do perspektyw w zapytania wykonywane na odpowiednich tabelach bazowych. Przykładowo, zgodnie z tym podejściem zapytanie `ZP1` zostałoby przez system zarządzania bazą danych automatycznie zmodyfikowane do następującej postaci:

```
SELECT IMIĘ, NAZWISKO
FROM   PRACOWNIK, PROJEKT, PRACUJE_NAD
WHERE  PESEL = PESELPRAC AND NR_PROJ = NUMERPROJ
       AND NAZWAPROJ = 'ProjektX';
```

Wadą tego rozwiązania jest jego nieefektywność w przypadku perspektyw zdefiniowanych za pomocą skomplikowanych zapytań, których wykonywanie jest kosztowne czasowo, szczególnie w przypadku, gdy w krótkim okresie czasu użytkownicy spróbują uruchomić stosunkowo dużą liczbę zapytań kierowanych do jednej perspektywy. Druga koncepcja (nazwana **materializacją perspektywy**) przewiduje fizyczne tworzenie tymczasowej tabeli dla perspektywy w momencie próby wykonania pierwszego kierowanego do niej zapytania przy założeniu, że pozostałe zapytania kierowane do tej samej perspektywy będą korzystały z tej samej, już istniejącej tabeli. W takim przypadku niezbędne jest zastosowanie efektywnego mechanizmu automatycznego aktualizowania tabeli perspektywy w odpowiedzi na każdą aktualizację tabel bazowych — tylko w ten sposób można zagwarantować aktualność danych przechowywanych w tymczasowej tabeli zmateriaлизованej perspektywy. W tym celu opracowano metody wykorzystujące technikę **przyrostowej aktualizacji**, która przewiduje konieczność wstawiania, usuwania i modyfikowania krotek w *zmateriaлизованej tabeli perspektywy* w przypadku zmian wprowadzonych w *jednej z tabel bazowych definiujących tę perspektywę*. Perspektywa jest fizycznie utrzymywana w bazie danych tak długo, jak długo użytkownicy wywołują skierowane do niej zapytania. Jeśli dana perspektywa nie jest przedmiotem zapytań przez określony okres czasu, system może automatycznie usunąć fizyczną tabelę perspektywy i w przyszłości skonstruować ją od podstaw w momencie pojawienia się kolejnych zapytań, które będą się do danej perspektywy odwoływały.

Stosowane są różne strategie dotyczące czasu aktualizowania perspektywy zmateriaлизованej. Metoda **natychmiastowej aktualizacji** polega na aktualizowaniu perspektywy bezpośrednio po zmodyfikowaniu tabel bazowych. Zgodnie ze strategią **leniwej aktualizacji** perspektywa jest modyfikowana wtedy, gdy jest potrzebna w zapytaniu. Metoda **okresowej aktualizacji** wymaga okresowego aktualizowania perspektywy. W tej ostatniej strategii wynik zapytania kierowanego do perspektywy może być nieaktualny.

Użytkownik zawsze może wywołać zapytanie pobierające dane z dowolnych perspektyw. Jednak wywołanie instrukcji INSERT, DELETE i UPDATE dla tabel perspektyw często jest niemożliwe. Zasadniczo, aktualizację perspektywy zdefiniowanej na podstawie zawartości *pojedynczej tabeli bazowej* i bez żadnych *funkcji agregujących* można (pod pewnymi warunkami) w prosty sposób odwzorować w odpowiednie operacje na tabeli bazowej. W przypadku perspektywy stanowiącej złączenie tabel, proces odwzorowywania operacji aktualizacji perspektywy w odpowiednią operację na tabelach bazowych może przebiegać na *wiele różnych sposobów*. Dlatego SZBD często nie potrafi ustalić, jakie aktualizacje są potrzebne. Aby zilustrować potencjalne problemy związane z aktualizowaniem perspektyw zdefiniowanych dla więcej niż jednej tabeli, przyjrzyjmy się utworzonej w poprzednim punkcie perspektywie PRACUJE\_NAD1 i przypuśćmy, że w przypadku pracownika 'Jan Nowak' zmieniono oryginalną nazwę projektu (wartość atrybutu NAZWAPROJ) 'ProduktX' na nazwę 'ProduktY'. Odpowiednią operację aktualizacji perspektywy można zrealizować za pomocą poniższego polecenia AP1:

```
AP1: UPDATE PRACUJE_NAD1
      SET   NAZWAPROJ = 'ProduktY'
      WHERE NAZWISKO = 'Nowak' AND IMIĘ = 'Jan'
            AND NAZWAPROJ = 'ProduktX';
```

Powyższe polecenie można odwzorować do różnych operacji aktualizujących relacje bazowe, które doprowadzą do oczekiwanego efektu aktualizacji perspektywy. Ponadto niektóre z tych operacji spowodują efekty uboczne wpływające na wyniki innych zapytań. Poleceniu AP1 odpowiadają dwie możliwe operacje aktualizacji, (a) i (b), na relacjach bazowych; obie operacje przedstawiono poniżej:

```
(a): UPDATE PRACUJE_NAD
      SET   NR_PROJ = (SELECT NUMERPROJ
                       FROM   PROJEKT
                       WHERE  NAZWAPROJ = 'ProduktY')
      WHERE PESELPRAC IN (SELECT PESEL
                          FROM   PRACOWNIK
                          WHERE  NAZWISKO = 'Nowak' AND IMIĘ = 'Jan')
      AND
      NR_PROJ = (SELECT NUMERPROJ
                FROM   PROJEKT
                WHERE  NAZWAPROJ = 'ProduktX');
(b): UPDATE PROJEKT SET NAZWAPROJ = 'ProduktY'
      WHERE NAZWAPROJ = 'ProduktX';
```

Operacja aktualizacji (a) łączy pracownika z imieniem i nazwiskiem 'Jan Nowak' z krótką relacją PROJEKT reprezentującą projekt oznaczony nazwą 'ProduktY', zamiast z dotychczasową krótką reprezentującą projekt 'ProduktX' — taka zmiana jest w tym przypadku jak najbardziej pożądana. Okazuje się jednak, że także operacja (b) przyniosłaby oczekiwany efekt na poziomie perspektywy, chociaż zmienia jedynie dotychczasową nazwę projektu reprezentowanego przez odpowiednią krótką relacji PROJEKT (z 'ProduktX' na 'ProduktY'). Jest mało prawdopodobne, by użytkownicy, którzy określili taką aktualizację perspektywy (zrealizowaną za pomocą polecenia AP1), chcieli, by ich działanie było interpretowane w sposób odpowiadający operacji (b), ponieważ efektem takiego działania byłaby konieczność zmodyfikowania wszystkich krotek perspektywy z wartością 'ProduktX' atrybutu NAZWAPROJ.

Niektóre aktualizacje perspektyw w ogóle nie mają sensu. Przykładowo, zmodyfikowanie atrybutu `SUMA_PENSJI` perspektywy `INFO_O_DZIAŁACH` jest logicznie nieuzasadnione, ponieważ atrybut ten reprezentuje sumę wynagrodzeń pojedynczych pracowników (odpowiednie żądanie przedstawiono poniżej w postaci polecenia AP2):

```
AP2: UPDATE INFO_O_DZIAŁACH
      SET     SUMA_PENSJI = 10000
      WHERE   NAZWADZ = 'Badania';
```

Aktualizacja perspektywy jest wykonywalna w sytuacji, gdy *tylko jedna możliwa operacja aktualizacji* relacji bazowych może prowadzić do otrzymania oczekiwanego efektu na poziomie danej perspektywy. Za każdym razem, gdy aktualizacja perspektywy może być odwzorowana w *więcej niż jedną* taką operację, zmiany są zwykle niedozwolone. Część badaczy uważa, że SZBD powinny stosować określoną procedurę wyboru najbardziej prawdopodobnego sposobu aktualizacji. Niektórzy badacze podjęli próby opracowania procedur wyboru najwłaściwszych metod aktualizacji, inni uznali, że lepszym rozwiązaniem jest pozostawienie decyzji w tej sprawie użytkownikowi (podczas definiowania perspektywy). Jednak w większości komercyjnych SZBD użytkownicy nie mają możliwości wyboru.

Podsumowując, możemy sformułować następujące wnioski:

- Perspektywa zbudowana w oparciu o dane zawarte w pojedynczej tabeli bazowej może być aktualizowana, jeśli jej atrybuty obejmują klucz główny relacji bazowej oraz jeśli zawiera wszystkie te atrybuty zadeklarowane z ograniczeniem `NOT NULL`, które nie mają określonych wartości domyślnych.
- Ogólnie rzecz biorąc, perspektywy zdefiniowane na bazie wielu tabel (stanowiące ich złączenie) nie mogą być aktualizowane.
- Perspektywy zdefiniowane z wykorzystaniem techniki grupowania lub funkcji agregujących także nie mogą być aktualizowane.

Jeśli dana perspektywa *ma być aktualizowana* za pomocą instrukcji `INSERT`, `DELETE` lub `UPDATE`, na końcu jej definicji należy użyć dostępnej w języku SQL klauzuli `WITH CHECK OPTION`. Dzięki niej system może odrzucać operacje, które naruszają reguły języka SQL związane z aktualizowaniem perspektyw. Kompletny zestaw reguł języka SQL dotyczących modyfikowania perspektyw przez użytkownika jest bardziej złożony niż wcześniej przedstawiona lista.

Perspektywy można też definiować w **klauzuli FROM** zapytań języka SQL. Są to tzw. **perspektywy wewnętrzzwierszowe**. W tym podejściu perspektywa jest definiowana w samym zapytaniu.

### 7.3.4. Perspektywy jako mechanizm uwierzytelniania

Stosowane w zapytaniach języka SQL instrukcje uwierzytelniania (`GRANT` i `REVOKE`) opisujemy szczegółowo w rozdziale 30., gdzie omawiamy zabezpieczenia baz danych i mechanizmy uwierzytelniania. Tu przedstawiamy tylko kilka prostych przykładów ilustrujących, jak stosować perspektywy do ukrywania określonych atrybutów lub krotek przed nieuprawnionymi użytkownikami. Załóżmy, że dany użytkownik może wyświetlać tylko informacje o osobach pracujących dla działu piątego. Możesz wtedy utworzyć perspektywę `PRACWYD5` i przyznać temu użytkownikowi uprawnienia do kierowania zapytań do tej perspektywy,

a nie do tabeli bazowej PRACOWNIK. Użytkownik będzie mógł pobierać tylko informacje z tych krotek z danymi pracowników, gdzie NRDZ = 5; zapytania kierowane do tej perspektywy nie pozwalają zobaczyć innych krotek pracowników.

```
CREATE VIEW PRACWYD5 AS
SELECT *
FROM PRACOWNIK
WHERE NRDZ = 5;
```

W podobny sposób perspektywa pozwala udostępnić użytkownikowi wyłącznie wybrane kolumny. Widoczne mogą być np. tylko imię, nazwisko i adres pracownika:

```
CREATE VIEW PODSTAWOWE_DANE_PRAC AS
SELECT NAZWISKO, IMIĘ, ADRES
FROM PRACOWNIK;
```

Tak więc utworzenie odpowiedniej perspektywy i przyznanie wybranym użytkownikom dostępu do niej, a nie do tabel bazowych, pozwala pobrać tylko dane zdefiniowane w tej perspektywie. W rozdziale 30. szczegółowo opisano bezpieczeństwo i uwierzytelnianie, w tym instrukcje GRANT i REVOKE języka SQL.

## 7.4. Dostępne w języku SQL polecenia zmiany schematu

W tym podrozdziale przedstawimy przegląd dostępnych w języku SQL tzw. **poleceń ewolucji schematu**, które mogą być wykorzystywane do modyfikowania schematów baz danych polegającego na dodawaniu i usuwaniu tabel, atrybutów, ograniczeń i innych elementów składających się na schematy. Operacje te można wykonywać w trakcie działania bazy danych; nie wymagają one ponownej kompilacji schematu. SZBD musi wykonywać niektóre testy, aby mieć pewność, że zmiany nie wpływają na resztę bazy i nie powodują utraty jej spójności.

### 7.4.1. Polecenie DROP

Polecenie DROP może być wykorzystywane do usuwania *nazwanych* elementów schematów, a więc tabel, dziedzin wartości, typów lub ograniczeń. Istnieje także możliwość usuwania całych schematów. Przykładowo, jeśli schemat nie jest już potrzebny, można użyć polecenia DROP SCHEMA. Istnieją dwie opcje definiujące *zachowanie podczas usuwania*: CASCADE i RESTRICT (odpowiadające odpowiednio kaskadowemu przekazywaniu tego działania oraz jego ograniczonemu zasięgowi). Przykładowo, aby usunąć schemat bazy danych FIRMA wraz ze wszystkimi należącymi do niego tabelami, dziedzinami wartościami i innymi elementami, należy użyć opcji CASCADE w następujący sposób:

```
DROP SCHEMA FIRMA CASCADE;
```

Gdybyśmy zamiast opcji CASCADE użyli opcji RESTRICT, schemat bazy danych FIRMA zostałby usunięty pod warunkiem, że nie zawierałby w sobie *żadnych* elementów; w przeciwnym przypadku polecenie DROP zostałoby odrzucone. Aby zastosować opcję RESTRICT, użytkownik musi najpierw pojedynczo usunąć wszystkie elementy schematu; dopiero potem może usunąć sam schemat.



Jeśli wybrana relacja bazowa wewnątrz schematu nie jest już potrzebna, można tę relację (wraz z jej definicją) usunąć za pomocą polecenia `DROP TABLE`. Przykładowo, jeśli uznamy, że nie ma potrzeby dalszego utrzymywania informacji o członkach rodzin pracowników firmy w bazie danych `FIRMA` (patrz rysunek 6.1), możemy usunąć relację `CZŁONEK_↪RODZINY` przez wykonanie następującego polecenia:

```
DROP TABLE CZŁONEK_RODZINY CASCADE;
```

Gdybyśmy zamiast opcji `CASCADE` użyli opcji `RESTRICT`, wybrana tabela mogłaby zostać usunięta tylko w sytuacji, gdyby nie istniały żadne wskazujące na nią *odwołania* w żadnym ze zdefiniowanych ograniczeń (mające np. postać deklaracji kluczy obcych w innej relacji), ani w żadnej z perspektyw (patrz podrozdział 7.3). Użycie opcji `CASCADE` spowoduje, że wszystkie ograniczenia i perspektywy, które odwołują się do danej tabeli, zostaną automatycznie usunięte ze schematu wraz z tą tabelą.

Warto zauważyć, że polecenie `DROP TABLE` (jeśli zostało z powodzeniem wykonane) usuwa nie tylko wszystkie rekordy tabeli, ale też *definicję tabeli* z katalogu. Jeżeli celem jest usunięcie rekordów, ale pozostawienie definicji tabeli (aby móc ją później wykorzystać), należy zastosować polecenie `DELETE` (patrz punkt 6.4.2) zamiast `DROP TABLE`.

Polecenie `DROP` może być wykorzystywane także do usuwania innych rodzajów nazwanych elementów schematu bazy danych, w tym ograniczeń i dziedzin wartości.

## 7.4.2. Polecenie ALTER

Definicja tabeli bazowej lub dowolnego innego nazwanego elementu należącego do schematu bazy danych może zostać zmieniona za pomocą polecenia `ALTER`. W przypadku tabel możliwe *operacje zmiany tabel* obejmują nie tylko dodawanie i usuwanie kolumn (atrybutów) oraz modyfikowanie ich definicji, ale także dodawanie i usuwanie ograniczeń nakładanych na tabelę. Przykładowo, aby do tabeli `PRACOWNIK` schematu `FIRMA` (patrz rysunek 6.1) dodać atrybut umożliwiający przechowywanie informacji o stanowiskach zajmowanych przez pracowników, możemy użyć następującego polecenia:

```
ALTER TABLE FIRMA.PRACOWNIK ADD COLUMN STANOWISKO VARCHAR(12);
```

Musimy jeszcze wpisać wartości nowego atrybutu `STANOWISKO` dla wszystkich istniejących krotek tabeli `PRACOWNIK`. Możemy to zrobić albo za pomocą dopisanej klauzuli wartości domyślnych (`DEFAULT`), albo wykorzystując polecenie `UPDATE` do każdej krotki (patrz podrozdział 6.4.3). Jeśli nie zdefiniujemy klauzuli `DEFAULT`, bezpośrednio po wykonaniu powyższego polecenia nowy atrybut będzie przechowywał wartość pustą (`NULL`) we wszystkich krotkach zmodyfikowanej relacji; oznacza to, że w takim przypadku *nie możemy* użyć klauzuli `NOT NULL`.

Aby usunąć kolumnę, musimy wybrać jedno z pary dostępnych zachowań: `CASCADE` lub `RESTRICT`. Jeśli zdecydujemy się na użycie opcji `CASCADE`, wszystkie ograniczenia i perspektywy odwołujące się do danej kolumny zostaną automatycznie usunięte ze schematu bazy danych (wraz ze wskazaną kolumną). Jeśli natomiast zastosujemy opcję `RESTRICT`, polecenie zostanie zaakceptowane i wykonane przez system zarządzania bazą danych pod warunkiem, że nie zostaną znalezione perspektywy lub ograniczenia (lub inne elementy) odwołujące się do usuwanej kolumny. Przykładowo, poniższe polecenie usuwa atrybut `ADRES` z tabeli bazowej `PRACOWNIK`:

```
ALTER TABLE FIRMA.PRACOWNIK DROP COLUMN ADRES CASCADE;
```

Możliwe jest też modyfikowanie definicji kolumn przez usuwanie istniejącej klauzuli DEFAULT lub przez zdefiniowanie nowej klauzuli tego typu. Poniższe przykłady ilustrują takie działania:

```
ALTER TABLE FIRMA.DZIAŁ ALTER COLUMN PESELKIEROWNIKA  
DROP DEFAULT;  
ALTER TABLE FIRMA.DZIAŁ ALTER COLUMN PESELKIEROWNIKA  
SET DEFAULT 55120834598;
```

Można także zmieniać (dodawać lub usuwać) nazwane ograniczenia zdefiniowane dla tabeli. Usunięcie ograniczenia jest możliwe tylko wtedy, gdy podczas tworzenia zostało ono opatrzone nazwą. Przykładowo, aby usunąć ograniczenie nazwane PRACPRZEŁOŻKO (patrz rysunek 6.2) z relacji PRACOWNIK, możemy użyć następującego polecenia:

```
ALTER TABLE FIRMA.PRACOWNIK  
DROP CONSTRAINT PRACPRZEŁOŻKO CASCADE;
```

Kiedy powyższe polecenie zostanie prawidłowo wykonane, możemy — w razie potrzeby — ponownie zdefiniować (dodać) inne ograniczenie tego typu. Dodanie nowego ograniczenia wymaga zastosowania wewnątrz polecenia ALTER TABLE słowa kluczowego ADD CONSTRAINT poprzedzającego definicję nowej struktury (może to być ograniczenie nazwane lub nienazwane; definiowane w ten sposób polecenie może być dowolnego typu spośród omówionych do tej pory rodzajów ograniczeń tabel).

W powyższych punktach omówiliśmy w skrócie dostępne w języku SQL polecenia ewolucji schematu. Za pomocą odpowiednich poleceń można tworzyć nowe tabele i perspektywy w schematach baz danych. Istnieje wiele innych rozwiązań szczegółowych i opcji — zainteresowanych Czytelników odsyłamy do wymienionych w bibliografii (na końcu tego rozdziału) materiałów i dokumentów poświęconych językowi SQL.

## 7.5. Podsumowanie

W tym rozdziale zaprezentowaliśmy dodatkowe właściwości stosowanego w bazach danych języka SQL. Zaczęliśmy od przedstawienia (w podrozdziale 7.1) złożonych mechanizmów zapytań pobierających dane, w tym zapytań zagnieżdżonych, tabel połączonych, złączeń zewnętrznych, funkcji agregujących i grupowania. W podrozdziale 7.2 przedstawiliśmy instrukcję CREATE ASSERTION, umożliwiającą tworzenie bardziej ogólnych ograniczeń bazy danych, oraz omówiliśmy wyzwalacze i instrukcję CREATE TRIGGER. Następnie (w podrozdziale 7.3) omówiliśmy dostępny w języku SQL mechanizm perspektyw. Perspektywy są też nazywane *tabelami wirtualnymi* lub *pochodnymi*, ponieważ są dostępne użytkownikom w formie podobnej do tabel. Jednak informacje w tych „tabelach” są tworzone na podstawie wcześniej zdefiniowanych tabel. W podrozdziale 7.4 przedstawiliśmy instrukcję ALTER TABLE języka SQL, służącą do modyfikowania tabel i ograniczeń baz danych.

Tabela 7.2 zawiera podsumowanie składni (struktury) różnych instrukcji języka SQL. To podsumowanie nie jest kompletne i nie opisuje wszystkich możliwych konstrukcji języka SQL. Ma to być krótki przegląd najważniejszych typów konstrukcji dostępnych w języku SQL.

TABELA 7.2. Podsumowanie składni języka SQL

CREATE TABLE <nazwa tabeli> ( <nazwa kolumny> <typ kolumny> [ <ograniczenie atrybutu> ] { , <nazwa kolumny> <typ kolumny> [ <ograniczenie atrybutu> ] } [ <ograniczenie tabeli> { , <ograniczenie tabeli> } ] )
DROP TABLE <nazwa tabeli>
ALTER TABLE <nazwa tabeli> ADD <nazwa kolumny> <typ kolumny>
SELECT [ DISTINCT ] <lista atrybutów> FROM ( <nazwa tabeli> { <alias> }   <tabela połączona> ) { , ( <nazwa tabeli> { <alias> }   ↳ <tabela połączona> ) } [ WHERE <warunek> ] [ GROUP BY <atrybuty grupujące> [ HAVING <warunek wybierania grup> ] ] [ ORDER BY <nazwa kolumny> [ <uporządkowanie> ] { , <nazwa kolumny> [ <uporządkowanie> } ] ]
<lista atrybutów> ::= ( *   ( <nazwa kolumny>   <funkcja> ( ( [ DISTINCT ] <nazwa ↳ kolumny>   * ) ) ) { , ( <nazwa kolumny>   <funkcja> ( ( [ DISTINCT ] <nazwa kolumny>   * ) ) ) } )
<atrybuty grupujące> ::= <nazwa kolumny> { , <nazwa kolumny> }
<uporządkowanie> ::= ( ASC   DESC )
INSERT INTO <nazwa tabeli> [ ( <nazwa kolumny> { , <nazwa kolumny> } ) ] ( VALUES ( <stała wartość> , { <stała wartość> } ) { , ( <stała wartość> { , <stała ↳ wartość> } ) }   <instrukcja select> )
DELETE FROM <nazwa tabeli> [ WHERE <warunek wybierania> ]
UPDATE <nazwa tabeli> SET <nazwa kolumny> = <wyrażenie określające wartość> { , <nazwa kolumny> = <wyrażenie ↳ określające wartość> } [ WHERE <warunek wybierania> ]
CREATE [ UNIQUE ] INDEX <nazwa indeksu> ON <nazwa tabeli> ( <nazwa kolumny> [ <uporządkowanie> ] { , <nazwa kolumny> [ <uporządkowanie> ] } ) [ CLUSTER ]
DROP INDEX <nazwa indeksu>
CREATE VIEW <nazwa perspektywy> [ ( <nazwa kolumny> { , <nazwa kolumny> } ) ] AS <instrukcja select>
DROP VIEW <nazwa perspektywy>

UWAGA: polecenia służące do tworzenia i usuwania indeksów nie są częścią standardu SQL.

Stosujemy tu notację BNF, w której symbole są zapisywane w nawiasach ostrych <...>, elementy opcjonalne są umieszczane w nawiasach kwadratowych [...], powtarzające się części znajdują się w nawiasach klamrowych {...}, a alternatywy są wymieniane w zwykłych nawiasach (...|...|...)7.

7 Kompletna składnia języka SQL jest opisana w wielu obszernych dokumentach liczących setki stron.

## Pytania powtórkowe

- 7.1. Opisz sześć klauzul stosowanych w składni zapytań języka SQL i przedstaw rodzaje konstrukcji, które można deklarować w każdej z nich. Które z tych klauzul są wymagane, a które mają charakter opcjonalny?
- 7.2. Spróbuj na poziomie koncepcyjnym opisać sposób, w jaki wykonywane są zapytania SQL — przedstaw koncepcyjną kolejność wykonywania każdej z sześciu klauzul składających się na zapytanie.
- 7.3. Omów sposób interpretowania wartości pustych (NULL) w operatorach porównania języka SQL. Jak wartości NULL są traktowane w zapytaniach języka SQL, w których zastosowano funkcje agregujące? Jak traktowane są wartości NULL występujące w atrybutach grupujących?
- 7.4. W jaki sposób poniższe konstrukcje są używane w języku SQL? Opisz opcje dostępne w każdej z nich. Określ, do czego przydatne są poszczególne konstrukcje.
  - a) Zapytania zagnieżdżone.
  - b) Tabele połączone i złączenia zewnętrzne.
  - c) Funkcje agregujące i grupowanie.
  - d) Wyzwalacze.
  - e) Asercje (wyjaśnij, czym różnią się one od wyzwalaczy).
  - f) Klauzula WITH języka SQL.
  - g) Konstrukcja CASE języka SQL.
  - h) Perspektywy (omów możliwość ich aktualizowania).
  - i) Polecenia zmiany schematu.

## Ćwiczenia

- 7.5. Zdefiniuj w języku SQL poniższe zapytania dla bazy danych z rysunku 5.5. Przedstaw wyniki tych zapytań przy założeniu, że baza danych FIRMA znajduje się w stanie zaprezentowanym na rysunku 5.6.
  - a) Dla każdego działu, którego pracownicy zarabiają średnio więcej niż 3000 złotych, pobierz nazwę działu i liczbę zatrudnionych w nim pracowników.
  - b) Przypuśćmy, że dla każdego działu chcemy znaleźć liczbę zatrudnionych w nim pracowników płci *męskiej*, zamiast łącznej liczby pracowników (jak w ćwiczeniu 7.5(a)). Czy takie zapytanie można zdefiniować w języku SQL? Odpowiedź uzasadnij.
- 7.6. Zdefiniuj w języku SQL następujące zapytania dla schematu bazy danych przedstawionego na rysunku 1.2.
  - a) Pobierz nazwiska wszystkich studentów, którzy wszystkie przedmioty zaliczyli na ocenę 5, i nazwy wydziałów, na których studiują.
  - b) Pobierz nazwiska wszystkich studentów, którzy nie zaliczyli żadnego przedmiotu na ocenę 5, i nazwy wydziałów, na których studiują.

- 7.7. Przedstaw w języku SQL opisane zapytania dotyczące bazy z rysunku 5.5. Wykorzystaj zapytania zagnieżdżone i inne techniki omówione w tym rozdziale.
- Pobierz nazwiska wszystkich pracowników z działu, do którego przypisana jest osoba o najwyższej pensji w firmie.
  - Pobierz nazwiska wszystkich pracowników mających przełożonego, którego przełożony ma numer PESEL równy '37111045873'.
  - Pobierz nazwiska pracowników zarabiających przynajmniej 1000 złotych więcej niż osoba, która ma najniższą pensję w firmie.
- 7.8. Zdefiniuj w języku SQL następujące perspektywy dla przedstawionego na rysunku 5.5 schematu relacyjnej bazy danych FIRMA:
- Perspektywa zawierająca dla każdego działu jego nazwę oraz nazwisko i wysokość pensji kierownika tego działu.
  - Perspektywa zawierająca dla każdego zatrudnionego w dziale 'Badania' nazwisko pracownika, nazwisko jego bezpośredniego przełożonego oraz wysokość pensji pracownika.
  - Perspektywa zawierająca dla każdego projektu jego nazwę, nazwę działu kontrolującego, liczbę pracowników bezpośrednio zaangażowanych w jego realizację oraz łączną liczbę godzin poświęcanych temu projektowi przez wszystkich pracowników.
  - Perspektywa zawierająca dla każdego projektu *realizowanego przez więcej niż jednego pracownika* nazwę projektu, nazwę działu kontrolującego, liczbę pracowników zaangażowanych w jego realizację oraz łączną liczbę godzin poświęcanych temu projektowi przez wszystkich pracowników.
- 7.9. Przeanalizuj poniższą perspektywę PODSUMOWANIE\_DZIAŁÓW zdefiniowaną na bazie danych FIRMA z rysunku 5.6:

```
CREATE VIEW PODSUMOWANIE_DZIAŁÓW (D, L, ŁĄCZNA_P, ŚREDNIA_P)
AS SELECT   NRDZ, COUNT(*), SUM(PENSJA), AVG(PENSJA)
FROM        PRACOWNIK
GROUP BY    NRDZ;
```

Spróbuj określić, które z wymienionych poniżej zapytań i operacji aktualizujących będzie można wykonać na tak zdefiniowanej perspektywie. Jeśli uznasz, że wykonanie któregoś z zapytań lub operacji aktualizacji jest możliwe, spróbuj skonstruować odpowiednie zapytanie lub operację aktualizacji właściwych relacji bazowych i podaj wynik zastosowania swojego polecenia dla stanu bazy danych z rysunku 5.6.

- SELECT \*  
FROM PODSUMOWANIE\_DZIAŁÓW;
- SELECT D, L  
FROM PODSUMOWANIE\_DZIAŁÓW  
WHERE ŁĄCZNA\_P > 10000;
- SELECT D, ŚREDNIA\_P  
FROM PODSUMOWANIE\_DZIAŁÓW  
WHERE L > (SELECT L FROM PODSUMOWANIE\_DZIAŁÓW WHERE D = 4);

```
d) UPDATE PODSUMOWANIE_DZIAŁÓW
   SET    D = 3
   WHERE  D = 4;

e) DELETE FROM PODSUMOWANIE_DZIAŁÓW
   WHERE  L > 4;
```

## Wybrane publikacje

Reisner (1977) opisała analizę czynników ludzkich w języku SEQUEL (poprzedniku języka SQL); odkryła, że użytkownicy mają trudności z poprawnym tworzeniem warunków złączenia i z grupowaniem. Date (1984) przedstawił krytykę języka SQL oraz opisał jego wady i zalety. Date i Darwen (1993) omówili język SQL2. W ANSI (1986) zaprezentowano pierwszy standard SQL. W podręcznikach różnych producentów baz opisano cechy implementacji języka SQL w systemach DB2, SQL/DS, Oracle, INGRES, Informix i innych komercyjnych produktach SZBD. Melton i Simon (1993) szczegółowo opisali standard ANSI 1992 (SQL2). Horowitz (1992) przedstawił wybrane problemy związane z integralnością odwołań i propagacją aktualizacji w języku SQL2.

Kwestię aktualizacji perspektyw podjęli w swoich publikacjach między innymi Dayal i Bernstein (1978), Keller (1982) oraz Langerak (1990). Blakeley i in. (1989) poświęcił swoją książkę implementacji mechanizmu perspektyw. Negri i in. (1991) opisał formalną semantykę zapytań języka SQL.

Dostępnych jest wiele książek opisujących różne aspekty języka SQL. Przykładowo, dwoma źródłami wiedzy o języku SQL-99 są Melton i Simon (2002) oraz Melton (2003). Późniejsze standardy języka SQL (SQL 2006 i SQL 2008) są opisane w różnych raportach technicznych, jednak nie istnieją ich opisy wzorcowe.





## Algebra relacyjna i rachunek relacji

W tym rozdziale omówimy dwa *języki formalne* dla relacyjnego modelu danych: algebrę relacyjną oraz rachunek relacji. Natomiast w rozdziałach 6. i 7. przedstawiliśmy *język praktyczny* relacyjnego modelu danych — standard SQL. Algebra relacyjna i rachunek relacji zostały opracowane przed językiem SQL. SQL jest oparty przede wszystkim na aspektach rachunku relacji i został wzbogacony o elementy z algebry relacyjnej. Ponieważ większość relacyjnych SZBD korzysta z języka SQL, najpierw przedstawiliśmy właśnie jego.

Zgodnie z naszymi spostrzeżeniami z rozdziału 2. model danych musi (poza podstawowymi elementami odpowiadającymi za definiowanie struktury i ograniczeń baz danych) obejmować zbiór operacji, które umożliwią manipulowanie informacjami zawartymi w bazie danych. Struktury i ograniczenia z formalnego modelu relacyjnego przedstawiliśmy w rozdziale 5. Podstawowy zbiór operacji dla modelu relacyjnego jest nazywany **algebrą relacyjną**. Operacje należące do tego zbioru umożliwiają użytkownikowi określanie podstawowych żądań wyszukiwania jako *wyrażeń algebry relacyjnej*. Wynikiem każdej z takich operacji jest nowa relacja. Oznacza to, że operacje algebry relacyjnej generują nowe relacje, które dalej mogą być manipulowane za pomocą operacji tej samej algebry. Sekwencja operacji tej algebry tworzy **wyrażenie algebry relacyjnej**, którego wynikiem także jest relacja reprezentująca efekt wykonania zapytania do bazy danych (lub żądania wyszukiwania danych).

Algebra relacyjna jest bardzo ważna z kilku powodów. Po pierwsze, stanowi formalną podstawę dla operacji modelu relacyjnego. Po drugie (i ten powód wydaje się jeszcze ważniejszy), jest wykorzystywana podczas implementowania i optymalizowania zapytań w systemach zarządzania relacyjnymi bazami danych (ang. *Relational Database Management System* — *RDBMS*); patrz rozdziały 18. i 19. Po trzecie, niektóre z elementów tej algebry zostały włączone do języka zapytań SQL (który jest standardem w systemach zarządzania relacyjnymi bazami danych). Choć większość używanych obecnie komercyjnych relacyjnych SZBD nie udostępnia interfejsu do zgłaszania zapytań algebry relacyjnej, podstawowe operacje i funkcje wewnętrznych modułów większości systemów relacyjnych są oparte na operacjach algebry relacyjnej. Te operacje są szczegółowo zdefiniowane w podrozdziałach od 8.1 do 8.4.

O ile algebra relacyjna definiuje zbiór operacji dla relacyjnego modelu danych, o tyle **rachunek relacji** zapewnia *deklaratywną* notację, która umożliwia definiowanie relacyjnych zapytań na nieco wyższym poziomie abstrakcji. W wyrażeniach rachunku relacji nie istnieje pojęcie *kolejności operacji*, która mogłaby wpływać na sposób uzyskiwania rezultatów zapytania — wyrażenie określa jedynie informacje, które powinny się znaleźć w wygenerowanym wyniku. Na tym właśnie polega główna różnica pomiędzy algebrą relacyjną a rachunkiem relacji. Znaczenie rachunku relacji wynika z jego solidnych podstaw

matematycznych oraz z powszechnego stosowania niektórych z jego elementów w relacyjnym rachunku krotek<sup>1</sup> w wykorzystywanym w systemach zarządzania relacyjnymi bazami danych języku SQL (standardowym języku zapytań).

Algebra relacyjna jest często uważana za integralną część relacyjnego modelu danych; operacje składające się na nią można podzielić na dwie grupy. Jedna grupa obejmuje operacje na zbiorach pochodzących z matematycznej teorii zbiorów — operacje tego typu mają w tym przypadku zastosowanie, ponieważ w *formalnym* modelu relacyjnym każda relacja jest (zgodnie z definicją) zbiorem krotek (patrz podrozdział 5.1). Operacje na zbiorach obejmują sumę, część wspólną (iloczyn zbiorów), różnicę zbiorów oraz iloczyn kartezjański. Druga grupa składa się z operacji opracowanych specjalnie z myślą o relacyjnych bazach danych — są to między innymi operacje selekcji, projekcji i złączenia. W pierwszej kolejności (w podrozdziale 8.1) opiszemy operacje selekcji i projekcji, ponieważ są to **operacje unarne**, a więc takie, które operują na pojedynczych relacjach. Następnie (w podrozdziale 8.2) omówimy operacje na zbiorach. W podrozdziale 8.3 przedstawimy operację złączenia i inne skomplikowane **operacje binarne**, czyli takie, które operują na dwóch tabelach, łącząc powiązane krotki (rekordy) na podstawie *warunków złączenia*. W naszych przykładach będziemy wykorzystywali przedstawioną na rysunku 5.6 relacyjną bazę danych FIRMA.

Niektóre popularne żądania dotyczące informacji zawartych w bazach danych nie mogą być wykonywane wyłącznie w oparciu o operacje oryginalnej algebry relacyjnej, zatem do ich wyrażania stworzono dodatkowe operacje, które rozszerzają tę algebrę. Należą do nich między innymi **funkcje agregacji**, które umożliwiają generowanie *podsumowań* dla danych zawartych w tabelach, oraz dodatkowe typy operacji złączenia i sumowania (złączenia i sumy zewnętrzne). Wspomniane operacje (opisane w podrozdziale 8.4) zostały dodane do oryginalnej algebry relacyjnej z uwagi na ich znaczenie dla wielu popularnych zastosowań baz danych. W podrozdziale 8.5 przedstawiliśmy przykłady definiowania zapytań opartych na operacjach algebry relacyjnej. Niektóre z prezentowanych zapytań posłużyły w rozdziałach 6. i 7. Możesz porównać, jak te same zapytania wyglądają w różnych językach zapytań.

W podrozdziałach 8.6 i 8.7 opisaliśmy inny ważny język formalny stworzony z myślą o relacyjnych bazach danych — **rachunek relacji**. Istnieją dwie odmiany tego rachunku. W podrozdziale 8.6 przedstawiliśmy relacyjny rachunek *krotek*, natomiast w podrozdziale 8.7 omówiliśmy relacyjny rachunek *dziedzin*. Niektóre z prezentowanych w rozdziałach 6. i 7. konstrukcji języka SQL opierają się na relacyjnym rachunku krotek. Rachunek relacji jest językiem formalnym opartym na gałęzi logiki matematycznej nazywanej rachunkiem predykatów<sup>2</sup>. W relacyjnym rachunku krotek zmienne obejmują wybrane *krotki*, natomiast w relacyjnym rachunku dziedzin — wybrane *dziedziny* (wartości) atrybutów. W dodatku C przedstawiliśmy przegląd języka zapytań przez przykłady (ang. *Query-by-Example* — *QBE*), który jest graficznym, przyjaznym dla użytkownika językiem relacyjnym opartym na rachunku dziedzin. W podrozdziale 8.8 zawarto podsumowanie treści tego rozdziału.

---

<sup>1</sup> Język SQL opiera się na relacyjnym rachunku krotek, ale obejmuje także niektóre operacje podstawowej algebry relacyjnej i jej rozszerzeń (patrz rozdziały 8. i 9.).

<sup>2</sup> W tym rozdziale nie jest wymagana znajomość rachunku predykatów pierwszego rzędu (nazywanego także rachunkiem kwantyfikatorów).

Czytelnicy, którzy nie są zainteresowani szczegółowym omawianiem wymienionych przed chwilą zagadnień, a jedynie wprowadzeniem do relacyjnych języków formalnych, mogą pominąć podrozdziały 8.4, 8.6 i 8.7.

## 8.1. Relacyjne operacje unarne: selekcja i projekcja

### 8.1.1. Operacja selekcji

Operacja **selekcji** (ang. SELECT) jest niezbędna do wyznaczania takiego *podzbioru* krotek z relacji, który spełni **warunek selekcji**. Operację selekcji można traktować jak pewnego rodzaju *filtr*, który przepuszcza tylko krotki spełniające zdefiniowany warunek. Można też przyjąć, że selekcja *ogranicza* listę krotek relacji do tych, które spełniają warunek. Operacja selekcji może także być przedmiotem wizualizacji przez *poziomy podział* relacji na dwa zbiory krotek — pierwszy składający się z krotek spełniających warunek selekcji i przekazanych na wyjściu operacji oraz drugi zawierający krotki, które nie spełniają tego warunku i zostały odrzucone przez operację selekcji. Przykładowo, aby wybrać tylko te krotki relacji PRACOWNIK, które reprezentują pracowników zatrudnionych w dziale 4., lub których pensja przekracza 3000 zł, możemy pojedynczo określić każdy z tych warunków w osobnych operacjach selekcji:

$$\sigma_{\text{NRDZ} = 4}(\text{PRACOWNIK})$$

$$\sigma_{\text{PENSJA} > 3000}(\text{PRACOWNIK})$$

Ogólna postać operacji selekcji jest następująca:

$$\sigma_{\langle \text{warunek selekcji} \rangle}(R)$$

Symbol  $\sigma$  (sigma) jest wykorzystywany do zapisywania operatora selekcji, a warunek selekcji jest wyrażeniem logicznym zdefiniowanym dla (wartości) atrybutów relacji  $R$ . Warto zauważyć, że generalnie  $R$  może być wyrażeniem *algebry relacyjnej*, którego wynikiem jest relacja — najprostszym takim wyrażeniem jest oczywiście nazwa relacji składowanej w bazie danych. Relacja wygenerowana na wyjściu operacji selekcji zawiera *niektóre atrybuty* relacji  $R$ .

Wyrażenie logiczne zdefiniowane w *<warunku selekcji>* może się składać z wielu **klauzul** w postaci:

$$\langle \text{nazwa atrybutu} \rangle \langle \text{operacja porównania} \rangle \langle \text{wartość stała} \rangle$$

lub

$$\langle \text{nazwa atrybutu} \rangle \langle \text{operacja porównania} \rangle \langle \text{nazwa atrybutu} \rangle$$

gdzie *<nazwa atrybutu>* jest nazwą jednego z atrybutów relacji  $R$ , *<operacja porównania>* jest zwykle jednym z operatorów należących do zbioru  $\{=, <, >, \geq, \leq, \neq\}$ , natomiast *<wartość stała>* jest stałą z dziedziny wartości danego atrybutu. Klauzule mogą być dowolnie łączone za pomocą operatorów logicznych *I*, *LUB* oraz *NIE*, które tworzą ogólny warunek selekcji.

Przykładowo, aby wyszukać krotki reprezentujące wszystkich pracowników, którzy albo są zatrudnieni w dziale 4. i zarabiają miesięcznie ponad 2500 zł, albo pracują w dziale 5. i zarabiają miesięcznie ponad 3000 zł, możemy użyć następującej operacji selekcji:

$\sigma_{(NRDZ = 4 \wedge PENSJA > 2500) \vee (NRDZ = 5 \wedge PENSJA > 3000)}(PRACOWNIK)$

Wynik tej operacji przedstawiono na rysunku 8.1(a).

Warto pamiętać, że operatory porównania należące do zbioru  $\{=, <, \leq, >, \geq, \neq\}$  mają zastosowanie wyłącznie dla atrybutów, których dziedziny są uporządkowanymi zbiorami wartości, a więc dotyczy to np. dziedzin numerycznych lub dat. Także dziedziny ciągów znaków są traktowane jako zbiory uporządkowane zgodnie z kolejnością sortowania. Jeśli dziedzina danego atrybutu jest nieuporządkowanym zbiorem wartości, możemy stosować wyłącznie operatory porównania z dwuelementowego zbioru  $\{=, \neq\}$ . Przykładem takiej nieuporządkowanej dziedziny jest *Kolor* = {czerwony, niebieski, zielony, biały, żółty, ...}, gdzie pomiędzy różnymi kolorami nie ma ustalonego porządku. Niektóre dziedziny umożliwiają stosowanie dodatkowych typów operatorów porównania; przykładowo, dziedzina ciągów znaków może dopuszczać stosowanie operatora porównania PODCIĄG.

(a)

PIMIĘ	DIMIĘ	NAZWISKO	PESEL		ADRES	PLEĆ	PENSJA	PESELSZEFA	NRDZIAŁU
Franciszek	Teodor	Wieszczycki	55120834598	1955-12-08	os. T. Kościuszki 4a/11, Szczecin	M	4000	37111045873	5
Janina		Wojtczak	41062013258	1941-06-20	ul. Kwiatowa 78, Mosina	K	4300	37111045873	4
Robert		Napierski	62091502054	1962-09-15	ul. Podgórna 7, Police	M	3800	55120834598	5

(b)

NAZWISKO	PIMIĘ	PENSJA
Szewczyk	Jan	3000
Wieszczycki	Franciszek	4000
Zalewska	Alicja	2500
Wojtczak	Janina	4300
Napierski	Robert	3800
Englert	Joanna	2500
Janiszewski	Albert	2500
Bąk	Józef	5500

(c)

PLEĆ	PENSJA
M	3000
M	4000
K	2500
K	4300
M	3800
M	2500
M	5500

RYСУNEK 8.1. Wynik operacji selekcji i projekcji. (a)  $\sigma_{(NRDZ = 4 \wedge PENSJA > 2500) \vee (NRDZ = 5 \wedge PENSJA > 3000)}(PRACOWNIK)$ . (b)  $\pi_{NAZWISKO, IMIĘ, PENSJA}(PRACOWNIK)$ . (c)  $\pi_{PLEĆ, PENSJA}(PRACOWNIK)$

Zasadniczo, wynik operacji selekcji może być określany w następujący sposób. Warunek selekcji jest stosowany niezależnie dla każdej *pojedynczej krotki* *t* w relacji *R*. Odbywa się to przez zastępowanie każdego wystąpienia atrybutu *A<sub>i</sub>* w warunku selekcji jego wartością w krotce *t*[*A<sub>i</sub>*]. Jeśli wynikiem warunku będzie wówczas PRAWDA (warunek selekcji będzie spełniony), krotka *t* zostanie **włączona** do relacji wynikowej. Wszystkie takie krotki będą tworzyły łączny wynik operacji selekcji. Logiczne warunki I, LUB i NIE są w wyrażeniach warunkowych interpretowane w tradycyjny sposób, co oznacza, że:

- wyrażenie (*warunek1* I *warunek2*) jest prawdziwe, jeśli zarówno *warunek1*, jak i *warunek2* jest prawdziwy; w przeciwnym przypadku wyrażenie (*warunek1* I *warunek2*) jest fałszywe;

- wyrażenie (*warunek1* LUB *warunek2*) jest prawdziwe, jeśli prawdziwy jest albo *warunek1*, albo *warunek2*, albo oba warunki jednocześnie; w przeciwnym przypadku wyrażenie (*warunek1* LUB *warunek2*) jest fałszywe;
- wyrażenie (NIE *warunek*) jest prawdziwe, jeśli *warunek* jest fałszywy; w przeciwnym przypadku wyrażenie (NIE *warunek*) jest fałszywe.

Operator selekcji jest **unarny**, co oznacza, że stosuje się go dla pojedynczej relacji. Co więcej, operator selekcji jest *osobno* stosowany dla *każdej krotki*, zatem wykorzystywane warunki selekcji nie mogą obejmować swoim zasięgiem więcej niż jedną krotkę. **Stopień** relacji wynikowej generowanej przez operację selekcji (a więc liczba atrybutów tej relacji) musi być taka sama jak stopień relacji *R*. Liczba krotek w relacji wynikowej zawsze jest *mniej lub równa* liczbie krotek w relacji *R*. Oznacza to, że dla każdego warunku *C* spełniona jest nierówność  $|\sigma_C(R)| \leq |R|$ . Odsetek krotek wybranych przez warunek selekcji jest nazywany **selektywnością** tego warunku.

Warto pamiętać, że operacja selekcji jest **przemienna**, co oznacza, że:

$$\sigma_{\langle \text{warunek1} \rangle}(\sigma_{\langle \text{warunek2} \rangle}(R)) = \sigma_{\langle \text{warunek2} \rangle}(\sigma_{\langle \text{warunek1} \rangle}(R))$$

Sekwencja operacji selekcji może więc być stosowana w dowolnej kolejności. Dodatkowo, zawsze możemy **kaskadowo łączyć** operacje selekcji w pojedynczą operację tego typu, wykorzystując warunek koniunkcyjny (połączony operatorem I):

$$\sigma_{\langle \text{warunek1} \rangle}(\sigma_{\langle \text{warunek2} \rangle}(\dots (\sigma_{\langle \text{warunekn} \rangle}(R)) \dots)) = \sigma_{\langle \text{warunek1} \rangle \wedge \langle \text{warunek2} \rangle \wedge \dots \wedge \langle \text{warunekn} \rangle}(R)$$

W języku SQL warunek selekcji jest zwykle podawany w *klauzuli WHERE* zapytania. Przykładowo, poniższa operacja:

```
NRDZ = 4 I PENSJA > 2500 (PRACOWNIK)
```

to odpowiednik następującego zapytania języka SQL:

```
SELECT *
FROM   PRACOWNIK
WHERE  NRDZ=4 AND PENSJA>2500;
```

## 8.1.2. Operacja projekcji

Jeśli przyjmiemy, że relacja jest tabelą, operacja selekcji wybiera niektóre z *wierszy* tabeli wejściowej, jednocześnie odrzucając pozostałe. Zupełnie inaczej przebiega realizacja operacji **projekcji**, która wybiera pewne *kolumny* z tabeli (także odrzucając pozostałe). Jeśli jesteśmy zainteresowani tylko niektórymi atrybutami relacji, wykorzystujemy operację projekcji do *rzutowania* relacji wejściowej do postaci relacji złożonej wyłącznie z wybranych atrybutów. Wynik operacji projekcji może więc być wizualnie przedstawiany w formie *piónowego podziału* relacji na dwie relacje — jedną złożoną wyłącznie z potrzebnych kolumn (atrybutów), która zawiera wynik tej operacji, i drugą zawierającą kolumny odrzucone. Przykładowo, do uzyskania listy zawartych w relacji PRACOWNIK imion, nazwisk i wysokości pensji dla wszystkich pracowników, możemy użyć następującej operacji projekcji:

```
 $\pi_{\text{NAZWISKO} \cdot \text{IMIE} \cdot \text{PENSJA}}(\text{PRACOWNIK})$ 
```

Relację otrzymaną po wykonaniu tej operacji przedstawiono na rysunku 8.1(b). Poniżej zaprezentowano ogólną postać relacji projekcji:

$$\pi_{\langle \text{lista atrybutów} \rangle}(R)$$

$\pi$  (pi) jest symbolem wykorzystywanym do reprezentowania operacji projekcji, natomiast  $\langle \text{lista atrybutów} \rangle$  określa oczekiwany zbiór kolumn wybranych z pełnego zbioru atrybutów relacji  $R$ . Także w tym przypadku warto pamiętać, że  $R$  jest generalnie dowolnym wyrażeniem algebry relacyjnej, którego wynikiem jest relacja (w najprostszym przypadku jest to po prostu nazwa relacji składowanej w bazie danych). Relacja wynikowa operacji projekcji zawiera wyłącznie atrybuty wymienione na  $\langle \text{liście atrybutów} \rangle$ . Porządek tych atrybutów w wygenerowanej relacji jest zgodny z ich kolejnością na przekazanej na wejściu liście. Oznacza to, że **stopień** relacji wynikowej jest równy liczbie atrybutów na tej liście.

Jeśli dana na wejściu lista atrybutów zawiera wyłącznie atrybuty, które nie tworzą klucza oryginalnej relacji  $R$ , w relacji wynikowej mogą wystąpić powtórzenia krotek. Operacja projekcji *usuwa wszystkie powtórzenia krotek*, zatem wynikiem tej operacji zawsze jest zbiór różnych krotek, a więc prawidłowa relacja. Takie rozwiązanie jest nazywane mechanizmem **eliminowania powtórzeń**. Przykładowo, przeanalizujemy poniższą operację projekcji:

$$\pi_{\text{PŁEĆ, PENSJA}}(\text{PRACOWNIK})$$

Wynik tej operacji przedstawiono na rysunku 8.1(c). Łatwo zauważyć, że krotka  $\langle K, 2500 \rangle$  występuje w relacji przedstawionej na rysunku 8.1(c) tylko raz, chociaż taka kombinacja wartości pojawia się w relacji PRACOWNIK dwukrotnie. Eliminowanie powtórzeń obejmuje zastosowanie sortowania lub innej techniki wykrywania powtórzeń, dlatego wymaga dodatkowego przetwarzania. Jeśli powtórzenia nie są eliminowane, wynikiem jest **wielozbiór** krotek, a nie ich zbiór. W formalnym modelu relacyjnym nie jest to dozwolone, jest to jednak dopuszczalne w języku SQL (patrz podrozdział 6.3).

Liczba krotek w relacji będącej wynikiem wykonania operacji projekcji jest zawsze mniejsza lub równa liczbie krotek w oryginalnej relacji  $R$ . Gdyby lista atrybutów operacji projekcji była nadkluczem (tj. gdyby zawierała jeden z kluczy relacji  $R$ ), relacja wynikowa zawierałaby *taką samą* liczbę krotek jak oryginalna relacja  $R$ . Co więcej, dopóki lista  $\langle \text{lista1} \rangle$  zawiera atrybuty z listy  $\langle \text{lista2} \rangle$ , spełniona jest równość:

$$\pi_{\langle \text{lista1} \rangle}(\pi_{\langle \text{lista2} \rangle}(R)) = \pi_{\langle \text{lista1} \rangle}(R)$$

W przeciwnym przypadku lewa strona powyższego równania nie jest prawidłowym wyrażeniem algebry relacyjnej. Warto także pamiętać, że operacja projekcji *nie spełnia* warunków operacji przechodniej.

W języku SQL lista atrybutów projekcji jest podawana w *klauzuli SELECT* zapytania. Przykładowo, poniższa operacja:

$$\pi_{\text{PŁEĆ, PENSJA}}(\text{PRACOWNIK})$$

jest odpowiednikiem następującego zapytania języka SQL:

```
SELECT DISTINCT PŁEĆ, PENSJA
FROM PRACOWNIK
```



Zauważ, że jeśli z tego zapytania SQL usuniesz słowo kluczowe DISTINCT, powtórzenia nie będą eliminowane. W formalnej algebrze relacyjnej nie jest to możliwe, jednak można ją rozszerzyć o tę operację i dopuścić tworzenie relacji w postaci wielozbiorów. Tu nie omawiamy takiego rozszerzenia.

8.1.3. Sekwencje operacji i operacja ZMIANA NAZWY

Relacje zaprezentowane na rysunku 8.1 nie mają przydzielonych żadnych nazw. Zasadniczo możemy kolejno stosować wiele operacji algebry relacyjnej, tworząc tym samym sekwencję takich operacji. Mamy możliwość zapisania tych operacji albo w postaci pojedynczego **wyrażenia algebry relacyjnej** (przez odpowiednie zagnieżdżenie operacji), albo przez sekwencyjne stosowanie pojedynczych operacji i tworzenie pośrednich relacji wynikowych. W tej drugiej sytuacji musimy przypisać nazwy relacjom, które przechowują wyniki pośrednie. Przykładowo, aby pobrać imię, nazwisko i pensję wszystkich pracowników z działu piątego, trzeba zastosować operacje selekcji i projekcji. Pojedyncze wyrażenie algebry relacyjnej (**wyrażenie wewnętrzne**) możemy zapisać w następujący sposób:

$\pi_{IMIE, NAZWISKO, PENSJA}(\sigma_{NRDZ = 5}(PRACOWNIK))$

Na rysunku 8.2(a) przedstawiono wynik tego wyrażenia. Alternatywnym rozwiązaniem jest zastosowanie sekwencji operacji algebry relacyjnej i nadawanie nazw każdej z otrzymywanych relacji pośrednich oraz zastosowanie **operacji przypisania** reprezentowanej za pomocą symbolu  $\leftarrow$  (strzałki skierowanej w lewo):

$PRAC\_DZIAŁU\_5 \leftarrow \sigma_{NRDZ = 5}(PRACOWNIK)$

$WYNIK \leftarrow \pi_{IMIE, NAZWISKO, PENSJA}(PRAC\_DZIAŁU\_5)$

(a)

PIMIE	NAZWISKO	PENSJA
Jan	Szewczyk	3000
Franciszek	Wieszczycki	4000
Robert	Napierski	3800
Joanna	Englert	2500

(b)

TYMCZASOWA

PIMIE	DIMIE	NAZWISKO	PESEL	DATAUR	ADRES	PLEC	PENSJA	PESELSZEFA	NRDZIAŁU
Jan	Bartłomiej	Szewczyk	65010912345	1965-01-09	ul. Kręta 4/3, Szczecin	M	3000	333445555	5
Franciszek	Teodor	Wieszczycki	55120834598	1955-12-08	os. T. Kościuszki 4a/11, Szczecin	M	4000	888665555	5
Robert	Krzysztof	Napierski	62091502054	1962-09-15	ul. Podgórna 7, Police	M	3800	333445555	5
Joanna	Aldona	Englert	72073110039	1972-07-31	ul. Wielka 40, Szczecin	K	2500	333445555	5

R

PIERWSZEIMIE	NAZWISKO	PENSJA
Jan	Szewczyk	3000
Franciszek	Wieszczycki	4000
Robert	Napierski	3800
Joanna	Englert	2500

RYSUNEK 8.2. Wynik wykonania sekwencji operacji. (a)  $\pi_{IMIE, NAZWISKO, PENSJA}(\sigma_{NRDZ = 5}(PRACOWNIK))$ .

(b) Zastosowanie relacji pośrednich i modyfikowanie nazw atrybutów



Rozbicie skomplikowanej sekwencji operacji algebry relacyjnej przez określenie pośrednich relacji wynikowych często jest prostszym rozwiązaniem niż napisanie jednego, złożonego wyrażenia tej algebry. Tę samą technikę możemy wykorzystać do **zmieniania nazw** (ang. *rename*) atrybutów w relacjach pośrednich i wynikowych. Jak się niedługo przekonamy, takie rozwiązanie może być przydatne w bardziej skomplikowanych operacjach, np. sumowania lub złączenia. Aby zmienić nazwy atrybutów w relacji, musimy jedynie wymienić nowe nazwy atrybutów na liście w nawiasach (patrz poniższy przykład):

$$\text{Tymczasowa} \leftarrow \sigma_{\text{NRDZ}=5}(\text{Pracownik})$$

$$R(\text{Imię, Nazwisko, Pensja}) \leftarrow \pi_{\text{Imię, Nazwisko, Pensja}}(\text{Tymczasowa})$$

Wynik wykonania powyższej pary operacji algebry relacyjnej przedstawiono na rysunku 8.2(b).

Gdybyśmy nie zastosowali operacji zmiany nazw, nazwy atrybutów w relacji wynikowej operacji selekcji byłyby takie same jak nazwy odpowiednich atrybutów w oryginalnej relacji i nie zmieniłaby się także kolejność tych atrybutów. W przypadku operacji projekcji bez zmiany nazw relacja wynikowa zawierałaby te same nazwy atrybutów, które umieszczono na liście projekcji (w takiej samej kolejności, w jakiej zostały wymienione na tej liście).

Możemy także zdefiniować formalną operację **zmiana nazwy** (która może zmieniać albo nazwę relacji, albo nazwy atrybutów, albo nazwy relacji i atrybutów jednocześnie) — jako operator unarny. Zastosowaną dla relacji  $R$  stopnia  $n$  operację zmiany nazwy można zapisać w jednej z wymienionych poniżej postaci:

$$\rho_{S(B_1, B_2, \dots, B_n)}(R) \text{ lub } \rho_S(R) \text{ lub } \rho_{(B_1, B_2, \dots, B_n)}(R)$$

Symbol  $\rho$  (ro) jest wykorzystywany do oznaczania operatora zmiany nazwy,  $S$  jest nową nazwą relacji, natomiast sekwencja  $B_1, B_2, \dots, B_n$  jest listą nowych nazw atrybutów. Pierwsze wyrażenie zmienia zarówno nazwę relacji, jak i nazwy jej atrybutów; drugie wyrażenie zmienia tylko nazwę relacji; natomiast trzecie zmienia wyłącznie nazwy atrybutów. Gdyby atrybutami relacji  $R$  były  $(A_1, A_2, \dots, A_n)$  w takiej kolejności, wówczas każda oryginalna relacja  $A_i$  miałaby nazwę zmienioną na  $B_i$ .

W języku SQL jedno zapytanie zwykle reprezentuje złożone wyrażenie algebry relacyjnej. Do zmiany nazwy w języku SQL służy alias tworzony za pomocą słowa AS:

```
SELECT  P.IMIĘ AS PIERWSZE_IMIĘ, P.NAZWISKO AS NAZWISKO, P.PENSJA AS PENSJA
FROM    PRACOWNIK AS P
WHERE   P.NRDZ=5.
```

## 8.2. Operacje algebry relacyjnej pochodzące z teorii zbiorów

### 8.2.1. Operacje sumy, części wspólnej i różnicy

Kolejną grupę operacji algebry relacyjnej stanowią standardowe matematyczne działania na zbiorach. Przykładowo, aby uzyskać numery PESEL wszystkich pracowników, którzy są zatrudnieni w piątym dziale firmy lub którzy są bezpośrednimi przełożonymi pracowników tego działu, możemy w następujący sposób użyć operacji sumy<sup>3</sup>:

PRAC\_DZIAŁU\_5  $\leftarrow \sigma_{NRDZ = 5}(PRACOWNIK)$

WYNIK1  $\leftarrow \pi_{PESEL}(PRAC\_DZIAŁU\_5)$

WYNIK2  $\leftarrow \pi_{PESELPRZEŁOZ}(PRAC\_DZIAŁU\_5)$

WYNIK  $\leftarrow WYNIK1 \cup WYNIK2$

Relacja WYNIK1 zawiera numery PESEL wszystkich pracowników zatrudnionych w dziale 5., natomiast relacja WYNIK2 przechowuje numery PESEL wszystkich pracowników, którzy są bezpośrednimi przełożonymi pracowników działu 5. Operator sumy zwraca krotki, które należą albo do relacji WYNIK1, albo do relacji WYNIK2, albo do obu relacji jednocześnie (patrz rysunek 8.3). Oznacza to, że wartość atrybutu PESEL równa 55120834598 będzie występowała w relacji wynikowej tylko raz.

WYNIK1	WYNIK2	WYNIK														
<table><tr><th>PESEL</th></tr><tr><td>65010912345</td></tr><tr><td>55120834598</td></tr><tr><td>62091502054</td></tr><tr><td>72073110039</td></tr></table>	PESEL	65010912345	55120834598	62091502054	72073110039	<table><tr><th>PESEL</th></tr><tr><td>55120834598</td></tr><tr><td>37111045873</td></tr></table>	PESEL	55120834598	37111045873	<table><tr><th>PESEL</th></tr><tr><td>65010912345</td></tr><tr><td>55120834598</td></tr><tr><td>62091502054</td></tr><tr><td>72073110039</td></tr><tr><td>37111045873</td></tr></table>	PESEL	65010912345	55120834598	62091502054	72073110039	37111045873
PESEL																
65010912345																
55120834598																
62091502054																
72073110039																
PESEL																
55120834598																
37111045873																
PESEL																
65010912345																
55120834598																
62091502054																
72073110039																
37111045873																

RYСУNEK 8.3. Wynik wykonania operacji sumy: WYNIK  $\leftarrow$  WYNIK1  $\cup$  WYNIK2

Do scalania elementów należących do dwóch zbiorów wykorzystuje się wiele różnych operacji teorii zbiorów, w tym takie operacje jak **suma**, **część wspólna** czy **różnica**. Są to operacje **binarne**, co oznacza, że każda z nich jest stosowana dla dwóch zbiorów (składających się z krotek). Kiedy wymienione operacje są wykorzystywane w relacyjnych bazach danych, para relacji, dla których zastosujemy którąkolwiek z tych operacji, musi zawierać **krotki tych samych typów**; warunek ten nosi nazwę **zgodności złączenia**. Mówimy, że między dwiema relacjami  $R(A_1, A_2, \dots, A_n)$  oraz  $S(B_1, B_2, \dots, B_n)$  istnieje **zgodność złączenia**, jeśli obie relacje są tego samego stopnia  $n$ , oraz jeśli  $\text{dom}(A_i) = \text{dom}(B_i)$  dla  $1 \leq i \leq n$ . Oznacza to, że wspomniana para relacji zawiera taką samą liczbę atrybutów oraz że każda para odpowiadających sobie atrybutów ma tę samą dziedzinę wartości.

<sup>3</sup> W postaci pojedynczego wyrażenia algebry relacyjnej te operacje wyglądają tak:  
WYNIK  $\leftarrow \pi_{PESEL}(\sigma_{NRDZ = 5}(PRACOWNIK)) \cup \pi_{PESELPRZEŁOZ}(\sigma_{NRDZ = 5}(PRACOWNIK))$ .

Dla dwóch relacji  $R$  i  $S$ , między którymi istnieje zgodność relacji, możemy w następujący sposób zdefiniować trzy operacje: suma, część wspólna czy różnica:

- **suma:** Wynikiem tej operacji (którą zapisujemy w postaci  $R \cup S$ ) jest relacja zawierająca wszystkie krotki należące do jednej z pary relacji  $R$  i  $S$  lub do obu tych relacji jednocześnie. Wszelkie powtórzenia krotek są w tej operacji automatycznie eliminowane.
- **część wspólna:** Wynikiem tej operacji (którą zapisujemy w postaci  $R \cap S$ ) jest relacja zawierająca wszystkie krotki, które należą zarówno do relacji  $R$ , jak i do relacji  $S$ .
- **różnica zbiorów:** Wynikiem tej operacji (którą zapisujemy w postaci  $R - S$ ) jest relacja zawierająca wszystkie krotki, które należą do relacji  $R$ , ale nie należą do relacji  $S$ .

W naszych rozważaniach przyjmiemy konwencję, która zakłada, że nazwy atrybutów w relacji wynikowej są identyczne jak w *pierwszej* relacji  $R$ . Oczywiście zawsze istnieje możliwość zmiany nazwy atrybutów w tej relacji za pomocą zaprezentowanego w poprzednim podrozdziale operatora zmiany nazwy.

Przykładowe zastosowanie operacji sumy, części wspólnej oraz różnicy przedstawiono na rysunku 8.4. Pomiędzy zaprezentowanymi na rysunku 8.4(a) relacjami STUDENT i WYKŁADOWCA istnieje zgodność złączenia, a ich krotki reprezentują odpowiednio nazwiska studentów i nazwiska wykładowców. Widoczna na rysunku 8.4(b) relacja wynikowa operacji sumy zawiera nazwiska wszystkich studentów i wykładowców. Nietrudno zauważyć, że powtarzające się krotki w obu relacjach występujących w roli operandów sumy zbiorów są umieszczane w relacji wynikowej tylko raz. Przedstawiona na rysunku 8.4(c) relacja wynikowa operacji części wspólnej zawiera tylko te krotki, które reprezentują osoby będące jednocześnie studentami i wykładowcami.

Łatwo zauważyć, że zarówno operacja suma, jak i operacja część wspólna jest *przemienne*; oznacza to, że:

$$R \cup S = S \cup R \text{ oraz } R \cap S = S \cap R$$

Operacje **sumy** i **części wspólnej** można traktować jak operacje  $n$ -składnikowe, które mają zastosowanie dla dowolnej liczby relacji, ponieważ obie te operacje są *łączne*; oznacza to, że:

$$R \cup (S \cap T) = (R \cup S) \cap T \text{ oraz } R \cap (S \cup T) = (R \cap S) \cup T$$

Operator **różnicy** nie jest przemienne, zatem:

$$R - S \neq S - R$$

Na rysunku 8.4(d) przedstawiono relację wynikową zawierającą nazwiska studentów, którzy nie są wykładowcami, natomiast na rysunku 8.4(e) — relację wynikową zawierającą nazwiska wykładowców, którzy nie są studentami.

Warto zauważyć, że część wspólną można zapisać za pomocą sumy i różnicy zbiorów w następujący sposób:

$$R \cap S = ((R \cup S) - (R - S)) - (S - R)$$

(a) STUDENT

I	N
Zuzanna	Zborowska
Robert	Słowiński
Jacek	Kostera
Barbara	Janda
Anna	Florek
Janusz	Wrona
Euzebiusz	Góra

WYKŁADOWCA

PIMIĘ	NAZWISKO
Jan	Szewczyk
Ryszard	Borowski
Zuzanna	Zborowska
Franciszek	Janson
Robert	Słowiński

(b)

I	N
Zuzanna	Zborowska
Robert	Słowiński
Jacek	Kostera
Barbara	Janda
Anna	Florek
Janusz	Wrona
Euzebiusz	Góra
Jan	Szewczyk
Ryszard	Borowski
Franciszek	Janson

(c)

I	N
Zuzanna	Zborowska
Robert	Słowiński

(d)

I	N
Jacek	Kostera
Barbara	Janda
Anna	Florek
Janusz	Wrona
Euzebiusz	Góra

(e)

PIMIĘ	NAZWISKO
Jan	Szewczyk
Ryszard	Borowski
Franciszek	Janson

RYSUNEK 8.4. Trzy przykładowe operacje na zbiorach: suma, część wspólna oraz różnica.  
(a) Dwie relacje, między którymi istnieje zgodność złączenia. (b) STUDENT U WYKŁADOWCA.  
(c) STUDENT∩WYKŁADOWCA. (d) STUDENT – WYKŁADOWCA. (e) WYKŁADOWCA – STUDENT

W języku SQL istnieją trzy operacje (UNION, INTERSECT i EXCEPT) odpowiadające opisanym tu operacjom na zbiorach. Ponadto dostępne są operacje dla wielozbiorów (UNION ALL, INTERSECT ALL i EXCEPT ALL) nieusuwaające powtórzeń (patrz punkt 6.3.4).

### 8.2.2. Operacja iloczynu (produktu) kartezjańskiego

W tym punkcie omówimy operację **iloczynu kartezjańskiego** (znaną także pod nazwą **produktu kartezjańskiego** lub **złączenia krzyżowego**), którą oznaczamy za pomocą symbolu  $\times$ . Iloczyn kartezjański — podobnie jak omówione wcześniej operacje — jest binarną operacją na zbiorach, ale *nie* wymaga od relacji zgodności złączenia. W wersji binarnej ta operacja zwraca nowe elementy utworzone w wyniku połączenia każdego elementu (krotki) z jednej relacji (zbioru) z każdym elementem (krotką) drugiej relacji (zbioru). Ogólnie rzecz biorąc, wynikiem wyrażenia  $R(A_1, A_2, \dots, A_n) \times S(B_1, B_2, \dots, B_m)$  jest relacja  $Q$  stopnia  $n+m$  (z taką liczbą atrybutów) w postaci:  $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$  — atrybuty są uporządkowane właśnie w taki sposób. Relacja wynikowa  $Q$  zawiera po jednej krotce dla każdej kombinacji krotek (jednej pochodzącej z relacji  $R$  i jednej z relacji  $S$ ). Oznacza to, że jeśli relacja  $R$  składa się z  $n_R$  krotek (co zapisujemy w postaci  $|R| = n_R$ ) oraz relacja  $S$  składa się z  $n_S$  krotek, wówczas relacja wynikowa  $R \times S$  będzie zawierała  $n_R \cdot n_S$  krotek.

$N$ -arna operacja iloczynu kartezjańskiego jest rozwinięciem opisanego procesu generującym nowe krotki w formie wszystkich możliwych kombinacji krotek z  $n$  bazowych relacji. Operacja iloczynu kartezjańskiego sama w sobie nie ma konkretnego znaczenia. Przydatność tej operacji ujawnia się dopiero w momencie, gdy dla otrzymanej relacji wynikowej zastosujemy operację selekcji, która odpowiednio skojarzy wartości atrybutów pochodzących z relacji składowych. Przykładowo, przypuśćmy, że chcemy otrzymać listę nazwisk wszystkich członków rodzin pracowników płci żeńskiej. Możemy ten cel osiągnąć za pomocą następującej sekwencji operacji:

$$\text{PRAC\_K} \leftarrow \sigma_{\text{PLEC} = 'K'}(\text{PRACOWNIK})$$

$$\text{NAZWISKA\_PRAC} \leftarrow \pi_{\text{IMIE}, \text{NAZWISKO}, \text{PESEL}}(\text{PRAC\_K})$$

$$\text{CZŁONKOWIE\_RODZINY\_PRAC} \leftarrow \text{NAZWISKA\_PRAC} \times \text{CZŁONEK\_RODZINY}$$

$$\text{CZŁONKOWIE\_RODZINY\_DANEGO\_PRAC} \leftarrow \sigma_{\text{PESEL} = \text{PESELPRAC}}(\text{CZŁONKOWIE\_RODZINY\_PRAC})$$

$$\text{WYNIK} \leftarrow \pi_{\text{IMIE}, \text{NAZWISKO}, \text{IMIE\_CZŁONKA\_RODZINY}}(\text{CZŁONKOWIE\_RODZINY\_DANEGO\_PRAC})$$

Na rysunku 8.5 przedstawiono relacje wynikowe wygenerowane za pomocą powyższej sekwencji operacji. Relacja CZŁONKOWIE\_RODZINY\_PRAC jest wynikiem zastosowania operacji iloczynu kartezjańskiego dla relacji NAZWISKA\_PRAC z rysunku 8.5 i relacji CZŁONEK\_RODZINY z rysunku 5.6. W relacji CZŁONKOWIE\_RODZINY\_PRAC każda krotka pochodząca z relacji NAZWISKA\_PRAC jest łączona z każdą krotką z relacji CZŁONEK\_RODZINY — w ten sposób otrzymujemy wynik, który sam w sobie niewiele oznacza (ponieważ każdy członek rodziny jest tu łączony z *każdym* pracownikiem płci żeńskiej). Chcemy przecież połączyć krotki reprezentujące pracowników płci żeńskiej wyłącznie z krotkami reprezentującymi ich członków rodziny, a więc krotki relacji CZŁONEK\_RODZINY z wartościami atrybutu PESELPRAC odpowiadającymi wartościom atrybutu PESEL wybranych krotek z relacji PRACOWNIK. Oczekiwany efekt otrzymujemy dopiero w relacji CZŁONKOWIE\_RODZINY\_DANEGO\_PRAC. Relacja CZŁONKOWIE\_RODZINY\_PRAC jest dobrym przykładem sytuacji, w której algebra relacyjna może być prawidłowo wykorzystywana do otrzymywania wyników, które bez odpowiedniego przetworzenia nie przedstawiają żadnej wartości. W takich przypadkach to użytkownik odpowiada za stosowanie tylko takich operacji na relacjach, które doprowadzą do uzyskania oczekiwanych informacji.

Operacja iloczynu kartezjańskiego tworzy krotki z połączonymi atrybutami dwóch relacji. Możemy następnie zastosować operację selekcji do wyselekcjonowania *tylko tych krotek z obu relacji, które są ze sobą powiązane* — tak jak w powyższym przykładzie, powodzenie takiego rozwiązania zależy od użycia właściwego warunku selekcji. Ponieważ sekwencja operacji iloczynu kartezjańskiego i selekcji jest powszechnie wykorzystywana do identyfikowania i selekcji powiązanych ze sobą krotek, które są przechowywane w dwóch różnych relacjach, opracowano specjalną operację złączenia, która definiuje tę sekwencję w postaci pojedynczej operacji. Omówimy tę operację w następnym podrozdziale.

W języku SQL iloczyn kartezjański można uzyskać za pomocą opcji CROSS JOIN przy tworzeniu połączonych tabel (patrz punkt 7.1.6). Ponadto jeśli w klauzuli FROM występują dwie tabele, a w klauzuli WHERE zapytania języka SQL nie ma dotyczącego je warunku złączenia, wynikiem jest iloczyn kartezjański tych dwóch tabel (patrz zapytanie Z10 w punkcie 6.3.3).

PRACOWNIK\_PŁCI\_ŻENSKIEJ

PIMIĘ	DIMIĘ	NAZWISKO	PESEL	DATAUR	ADRES	PŁEĆ	PENSJA	PESELSZEFA	DRDZIAŁU
Alicja	Joanna	Zalewska	68011932514	1968-07-19	os. Piastowskie 2/23, Swarzędz	K	2500	41062013258	4
Janina	Sylvia	Wojtczak	41062013258	1941-06-20	ul. Kwiatowa 78, Mosina	K	4300	37111045873	4
Joanna	Aldona	Englert	72073110039	1972-07-31	ul. Wielka 40, Szczecin	K	2500	55120834598	5

NAZWISKA\_PRAC

PIMIĘ	NAZWISKO	PESEL
Alicja	Zalewska	68011932514
Janina	Wojtczak	41062013258
Joanna	Englert	72073110039

CZŁONKOWIE\_RODZINY\_PRAC

PIMIĘ	NAZWISKO	PESEL	PPESEL	IMIĘ_CZŁONKA_RODZINY	PŁEĆ	DATAUR	...
Alicja	Zalewska	68011932514	55120834598	Alicja	K	1986-04-05	...
Alicja	Zalewska	68011932514	55120834598	Tomasz	M	1983-10-25	...
Alicja	Zalewska	68011932514	55120834598	Janusz	K	1958-05-03	...
Alicja	Zalewska	68011932514	41062013258	Anna	M	1942-02-28	...
Alicja	Zalewska	68011932514	65010912345	Michał	M	1988-01-04	...
Alicja	Zalewska	68011932514	65010912345	Alice	K	1988-12-30	...
Alicja	Zalewska	68011932514	65010912345	Elżbieta	K	1967-05-05	...
Janina	Wojtczak	41062013258	55120834598	Alicja	K	1986-04-05	...
Janina	Wojtczak	41062013258	55120834598	Tomasz	M	1983-10-25	...
Janina	Wojtczak	41062013258	55120834598	Janusz	K	1958-05-03	...
Janina	Wojtczak	41062013258	41062013258	Anna	M	1942-02-28	...
Janina	Wojtczak	41062013258	65010912345	Michał	M	1988-01-04	...
Janina	Wojtczak	41062013258	65010912345	Alicja	K	1988-12-30	...
Janina	Wojtczak	41062013258	65010912345	Elżbieta	K	1967-05-05	...
Joanna	Englert	72073110039	55120834598	Alicja	K	1986-04-05	...
Joanna	Englert	72073110039	55120834598	Tomasz	M	1983-10-25	...
Joanna	Englert	72073110039	55120834598	Janusz	K	1958-05-03	...
Joanna	Englert	72073110039	41062013258	Anna	M	1942-02-28	...
Joanna	Englert	72073110039	65010912345	Michał	M	1988-01-04	...
Joanna	Englert	72073110039	65010912345	Alicja	K	1988-12-30	...
Joanna	Englert	72073110039	65010912345	Elżbieta	K	1967-05-05	...

CZŁONKOWIE\_RODZINY\_DANEGO\_PRAC

PIMIĘ	NAZWISKO	PESEL	PPESEL	IMIĘ_CZŁONKA_RODZINY	PŁEĆ	DATAUR	...
Janina	Wojtczak	41062013258	41062013258	Anna	M	1942-02-28	...

WYNIK

PIMIĘ	NAZWISKO	IMIĘ_CZŁONKA_RODZINY
Janina	Wojtczak	Anna

RYSUNEK 8.5. Operacja iloczynu (produktu) kartezjańskiego

## 8.3. Binarne operacje na relacjach: złączenie i dzielenie

### 8.3.1. Operacja złączenia

Operacja złączenia (oznaczana za pomocą symbolu  $\bowtie$ ) jest wykorzystywana do złączenia występujących w dwóch różnych relacjach *powiązanych ze sobą krotek* w pojedyncze, „dłuższe” krotki. Ta operacja jest bardzo ważna w przypadku wszystkich relacyjnych baz

danych, które zawierają więcej niż jedną relację, ponieważ umożliwia przetwarzanie istniejących związków pomiędzy relacjami. Aby lepiej zrozumieć znaczenie operacji złączenia, przypuśćmy, że chcemy uzyskać nazwiska kierowników każdego z działów. Aby uzyskać nazwisko kierownika, musimy połączyć każdą krotkę reprezentującą dział z krotką reprezentującą pracownika, której wartość atrybutu PESEL jest równa wartości atrybutu PESEL  $\hookrightarrow$  KIEROWNIKA w krotce działu. Możemy to zrobić za pomocą prostej sekwencji operacji złączenia obu relacji i operacji projekcji wyniku do niezbędnych atrybutów:

$$\text{KIEROWNIK\_DZIAŁU} \leftarrow \text{DZIAŁ} \bowtie_{\text{PESELKIEROWNIKA} = \text{PESELPRACOWNIK}}$$

$$\text{WYNIK} \leftarrow \pi_{\text{NAZWADZ, NAZWISKO, IMIĘ}}(\text{KIEROWNIK\_DZIAŁU})$$

Pierwszą z powyższej sekwencji operacji przedstawiono na rysunku 8.6. Łatwo zauważyć, że atrybut PESELKIEROWNIKA jest kluczem obcym relacji DZIAŁ powiązanym z atrybutem PESEL, który jest kluczem głównym relacji PRACOWNIK. Ponadto dużą rolę w dopasowywaniu krotek przechowywanych we wskazywanej relacji PRACOWNIK odgrywają więzy integralności odwołań.

#### MENADŻER\_DZIAŁU

DNAZWA	DNUMER	PESELMENADŻERA	...	PIMIĘ	DIMIĘ	NAZWISKO	PESEL	...
Badania	5	55120834598	...	Franciszek	Teodor	Wieszczycki	55120834598	...
Administracja	4	41062013258	...	Janina	Sylwia	Wojtczak	41062013258	...
Centrala	1	37111045873	...	Józef	Edward	Bąk	37111045873	...

RYСУNEK 8.6. Wynik operacji złączenia:  $\text{KIEROWNIK\_DZIAŁU} \leftarrow \text{DZIAŁ} \bowtie_{\text{PESELKIEROWNIKA} = \text{PESEL}} \text{PRACOWNIK}$

Operacja złączenia może być łatwo wyrażona za pomocą sekwencji operacji iloczynu kartezjańskiego i selekcji. Sama operacja złączenia jest jednak bardzo ważna, ponieważ wyjątkowo często stosuje się ją do określania zapytań do relacyjnych baz danych. Przeanalizujemy raz jeszcze zaprezentowany w poprzednim podrozdziale przykład zastosowania operacji iloczynu kartezjańskiego, który zawierał następującą sekwencję operacji:

$$\text{CZŁONKOWIE\_RODZINY\_PRAC} \leftarrow \text{NAZWISKA\_PRAC} \times \text{CZŁONEK\_RODZINY}$$

$$\text{CZŁONKOWIE\_RODZINY\_DANEGO\_PRAC} \leftarrow \sigma_{\text{PESEL} = \text{PESELPRAC}}(\text{CZŁONKOWIE\_RODZINY\_PRAC})$$

Powyższą sekwencję operacji iloczynu kartezjańskiego i selekcji można bez trudu zastąpić jedną operacją złączenia:

$$\text{CZŁONKOWIE\_RODZINY\_DANEGO\_PRAC} \leftarrow \text{NAZWISKA\_PRAC} \bowtie_{\text{PESEL} = \text{PESELPRAC}} \text{CZŁONEK\_RODZINY}$$

Ogólna postać operacji złączenia dla pary relacji<sup>4</sup>  $R(A_1, A_2, \dots, A_n)$  i  $S(B_1, B_2, \dots, B_m)$  jest następująca:

$$R \bowtie_{\langle \text{warunek złączenia} \rangle} S$$

Wynikiem operacji złączenia jest relacja  $Q$  zawierająca  $n+m$  atrybutów uporządkowanych według następującej kolejności:  $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$ ; relacja  $Q$  zawiera po jednej krotce dla każdej kombinacji krotek relacji  $R$  i  $S$ , pod warunkiem, że ta kombinacja

<sup>4</sup> Także w tym przypadku  $R$  i  $S$  mogą być dowolnymi relacjami otrzymanymi z ogólnego wyrażenia algebry relacyjnej.



*spełnia warunek złączenia*. To dodatkowe obostrzenie jest głównym czynnikiem odróżniającym operację złączenia od operacji iloczynu kartezjańskiego. W przypadku operacji złączenia, w relacji wynikowej są umieszczane tylko te kombinacje krotek, *które spełniają warunek złączenia*, natomiast w przypadku operacji iloczynu kartezjańskiego relacja wynikowa zawiera *wszystkie* możliwe kombinacje krotek. Warunek złączenia może dotyczyć wartości dowolnych atrybutów z pary relacji  $R$  i  $S$ , jego wartość jest wyznaczana dla każdej możliwej kombinacji krotek obu relacji. Każda kombinacja krotek, dla której warunek złączenia ma wyznaczoną wartość PRAWDA (jest spełniony), jest umieszczana w relacji wynikowej  $Q$  w postaci *jednej połączonej krotki*.

Ogólna postać warunku złączenia jest następująca:

$$\langle \text{warunek} \rangle \text{ I } \langle \text{warunek} \rangle \text{ I } \dots \text{ I } \langle \text{warunek} \rangle$$

gdzie każdy warunek ma postać  $A_i \theta B_j$  —  $A_i$  jest atrybutem relacji  $R$ ,  $B_j$  jest atrybutem relacji  $S$ , wartości atrybutów  $A_i$  i  $B_j$  należą do tej samej dziedziny oraz  $\theta$  (theta) jest jednym z operatorów porównania należących do zbioru  $\{=, <, \leq, >, \geq, \neq\}$ . Operacja złączenia z tak zdefiniowanym ogólnym warunkiem złączenia jest nazywana operacją **złączenia theta**. Krotki, które przechowują wartości puste w atrybutach złączenia, *nie są umieszczane* w relacji wynikowej. W związku z tym można przyjąć, że operacja złączenia wcale *nie* musi obejmować wszystkich informacji zawartych w relacjach występujących w roli jej operandów (ponieważ krotki niedopasowane do krotek z drugiej relacji nie występują w wyniku).

### 8.3.2. Odmiany operacji złączenia: operacje równo-złączenia i złączenia naturalnego

Większość współczesnych zastosowań operacji złączenia opiera się na warunkach złączenia zawierających wyłącznie porównania równościowe. Takie operacje złączenia, w których jedynym operatorem porównania jest znak równości, nazywamy operacjami **równozłączenia**. Oba wcześniejsze przykłady dotyczą równo-złączeń. Łatwo zauważyć, że relacja wynikowa wygenerowana przez operację równo-złączenia zawsze zawiera jedną lub więcej par atrybutów z *identycznymi wartościami* we wszystkich krotkach. Przykładowo, wartości atrybutów PESELKIEROWNIKA i PESEL we wszystkich krotkach przedstawionej na rysunku 8.6 relacji KIEROWNIK\_DZIAŁU są identyczne, ponieważ określony dla tych dwóch atrybutów równościowy warunek złączenia *wymaga, by wartości tych atrybutów były identyczne* w każdej wynikowej krotce. Ponieważ jeden z pary atrybutów zawierających identyczne wartości zawsze jest nadmiarowy, stworzono operację **złączenia naturalnego** (oznaczaną symbolem  $*$ ), która usuwa z relacji wynikowej drugi (nadmiarowy) atrybut warunku operacji równo-złączenia<sup>5</sup>. Standardowa definicja operacji złączenia naturalnego wymaga, aby dwa atrybuty złączenia (lub, jeśli jest ich więcej, każda z par atrybutów złączenia) występowała w obu relacjach pod taką samą nazwą. Jeśli jest inaczej, najpierw stosuje się operację zmiany nazwy odpowiednich atrybutów.

Założmy, że chcemy połączyć każdą krotkę z relacji PROJEKT z krotką działu (relacji DZIAŁ) nadzorującego dany projekt. W poniższym przykładzie najpierw zmienimy nazwę atrybutu NUMERDZ relacji DZIAŁ na NR\_DZ (w ten sposób uzyskamy zgodność nazw z odpowiednim atrybutem relacji PROJEKT), po czym zastosujemy operację złączenia naturalnego:

<sup>5</sup> Operacja ZŁĄCZENIA NATURALNEGO jest po prostu sekwencją operacji RÓWNO-ZŁĄCZENIA i operacji usuwania nadmiarowych atrybutów.

$DZIAŁ\_PROJ \leftarrow PROJEKT * \rho_{(NAZWAZ, NUMERDZ, PESELKIEROWNIKA, DATAPRZEJKIEROWNICTWA)}(DZIAŁ)$

To samo zapytanie można zrealizować w dwóch krokach — tworząc pośrednią tabelę nazywaną np. DZ, patrz poniżej:

$DZ \leftarrow \rho_{(NAZWAZ, NUMERDZ, PESELKIEROWNIKA, DATAPRZEJKIEROWNICTWA)}(DZIAŁ)$

$DZIAŁ\_PROJ \leftarrow PROJEKT * DZ$

Atrybut NUMERDZ jest w takim przypadku nazywany **atrybutem złączenia** w operacji złączenia naturalnego, ponieważ stanowi jedyny atrybut o tej samej nazwie występujący w obu relacjach. Relację wynikową tej operacji przedstawiono na rysunku 8.7(a). Każda krotka w relacji DZIAŁ\_PROJ jest kombinacją krotki relacji PROJEKT z odpowiednią krotką relacji DZIAŁ (reprezentującą dział firmy nadzorujący dany projekt), jednak zawiera *tylko jeden atrybut złączenia*.

(a)

DZIAŁ\_PROJ

PNAZWA	PNUMER	PLOKALIZACJA	NR_DZIAŁU	DNAZWA	PESELMENADŻERA	DATAPOCZMENADŻERA
ProduktX	1	Bydgoszcz	5	Badanie	55120834509	1988-05-22
ProduktY	2	Świebodzin	5	Badanie	55120834509	1988-05-22
ProduktZ	3	Szczecin	5	Badanie	55120834509	1988-05-22
Komputeryzacja	10	Poznań	4	Administracja	41062013258	1995-01-01
Reorganizacja	20	Szczecin	1	Centrala	37111045873	1981-06-19
ZwiększenieZysków	30	Poznań	4	Administracja	41062013258	1995-01-01

(b)

LOKALIZACJE\_DZIAŁÓW

DNAZWA	DNUMER	PESELMENADŻERA	DATAPOCZMENADŻERA	LOKALIZACJA
Centrala	1	37111045873	1981-06-19	Szczecin
Administracja	4	41062013258	1995-01-01	Poznań
Badanie	5	55120834509	1988-05-22	Bydgoszcz
Badanie	5	55120834509	1988-05-22	Świebodzin
Badanie	5	55120834509	1988-05-22	Szczecin

RYSUNEK 8.7. Wyniki dwóch operacji złączenia naturalnego. (a)  $DZIAŁ\_PROJ \leftarrow PROJEKT * DZ$ .

(b)  $LOK\_DZIAŁÓW \leftarrow DZIAŁ * LOKALIZACJE\_DZIAŁÓW$

Gdyby atrybuty, na których zdefiniowano operację złączenia naturalnego, miały już *takie same nazwy w obu relacjach*, zmiana nazwy byłaby oczywiście zbędnym zabiegiem. Przykładowo, aby zastosować tę operację na atrybutach relacji DZIAŁ i LOKALIZACJE\_DZIAŁÓW, wystarczyłoby użyć wyrażenia:

$LOK\_DZIAŁÓW \leftarrow DZIAŁ * LOKALIZACJE\_DZIAŁÓW$

Relację wynikową tej operacji przedstawiono na rysunku 8.7(b). Otrzymana relacja zawiera kombinacje krotek reprezentujących działu firmy z krotkami reprezentującymi ich lokalizacje — dla każdej krotki reprezentującej jedną lokalizację istnieje w tej relacji jedna krotka. Ogólnie rzecz biorąc, złączenie naturalne jest wykonywane przez porównywanie *wszystkich par atrybutów*, które w obu relacjach mają takie same nazwy, i połączenie tych warunków za pomocą operatora I. Może istnieć lista atrybutów złączenia z każdej z łączonych relacji, a każda z odpowiadających sobie par musi mieć taką samą nazwę.

Warto pamiętać, że jeśli żadna z wygenerowanych kombinacji krotek nie spełni warunku złączenia, wynikiem operacji złączenia będzie pusta relacja (czyli taka, która zawiera zero krotek). Generalnie, jeśli relacja  $R$  zawiera  $n_R$  krotek oraz relacja  $S$  zawiera  $n_S$  krotek, relacja wynikowa operacji złączenia w postaci  $R \bowtie_{\langle \text{warunek złączenia} \rangle} S$  będzie zawierała minimalnie 0 i maksymalnie  $n_R * n_S$  krotek. Oczekiwany rozmiar relacji wynikowej uzyskanej w procesie złączenia podzielony przez rozmiar maksymalny ( $n_R * n_S$ ) daje współczynnik nazywany **selektywnością złączenia**, który jest własnością każdego warunku złączenia. Jeśli dla operacji złączenia nie określono żadnego warunku złączenia, do relacji wynikowej zostaną zakwalifikowane wszystkie kombinacje krotek, a operacja złączenia zostanie przekształcona w operację iloczynu kartezjańskiego (nazywaną także produktem kartezjańskim lub złączeniem krzyżowym).

Nietrudno zauważyć, że operacja złączenia jest wykorzystywana do takiego złączenia danych pochodzących z wielu relacji, które umożliwi prezentowanie powiązanych ze sobą informacji w postaci pojedynczej tabeli. Operacje tego typu są także nazywane **złączeniami wewnętrznymi** — to określenie ma na celu ich odróżnienie od innej odmiany operacji złączenia, którą nazywa się **złączeniem zewnętrznym** (patrz punkt 8.4.4). Nieformalnie można stwierdzić, że **złączenie wewnętrzne** to operacja „dopasuj i połącz” obejmująca iloczyn kartezjański i selekcję. Warto pamiętać, że operacja złączenia może zostać użyta pomiędzy dwoma odwołaniami do tej samej relacji — takie rozwiązanie zaprezentowano w punkcie 8.4.3. Co więcej, okazuje się, że operacje złączenia naturalnego i równozłączenia mogą dotyczyć wielu tabel i prowadzić tym samym do operacji **złączenia  $n$ -krotnego**. Przykładowo, przeanalizujmy następującą operację złączenia 3-krotnego:

$$((\text{PROJEKT} \bowtie_{\text{NR\_DZ} = \text{NUMERDZ}} \text{DZIAŁ}) \bowtie_{\text{PESELKIEROWNIKA} = \text{PESEL}} \text{PRACOWNIK})$$

W ten sposób wiążemy każdy projekt z odpowiednim działem nadzorującym, po czym wiążemy ten dział z pracownikiem pełniącym rolę jego kierownika. Ostateczny wynik ma postać skonsolidowanej relacji, w której każda krotka zawiera takie interesujące zestawienie projektu-działu-kierownika.

W języku SQL złączenia można wykonywać na kilka różnych sposobów. Pierwsza metoda polega na podaniu warunków złączenia w klauzuli `WHERE` (obok innych warunków selekcji). Jest to bardzo często stosowana technika, zilustrowana w zapytaniach Z1, Z1A, Z1B, Z2 i Z8 w punktach 6.3.1 i 6.3.2, a także w wielu innych przykładowych zapytaniach z rozdziałów 6. i 7. Drugi sposób to zastosowanie relacji zagnieżdżonej, tak jak w zapytaniach Z4A i Z16 w punkcie 7.1.2. Jeszcze inna metoda to wykorzystanie tabel połączonych (zapytania Z1A, Z1B, Z8B i Z2A w punkcie 7.1.6). Mechanizm tabel połączonych został dodany do języka SQL2, aby umożliwić użytkownikom bezpośrednie tworzenie wszystkich rodzajów złączeń (inne techniki mają większe ograniczenia). Dzięki takim tabelom użytkownik może też jednoznacznie odróżnić warunki złączenia od warunków selekcji z klauzuli `WHERE`.

### 8.3.3. Kompletny zbiór operacji algebry relacyjnej

Udowodniono, że zaprezentowany w tym rozdziale zbiór operacji algebry relacyjnej  $\{\sigma, \pi, \cup, \rho, -, \times\}$  jest zbiorem **kompletnym**; oznacza to, że wszystkie pozostałe operacje oryginalnej algebry relacyjnej można wyrazić w postaci *sekwencji operacji należących do tego zbioru*. Przykładowo, operację części wspólnej można wyrazić za pomocą odpowiedniej kombinacji operacji sumy i różnicy relacji:

$$R \cap S \equiv (R \cup S) - ((R - S) \cup (S - R))$$

Chociaż — ściśle mówiąc — operacja części wspólnej nie jest niezbędna, stosowanie tak skomplikowanych wyrażeń za każdym razem, gdy chcemy otrzymać część wspólną dwóch relacji, byłoby niewygodne. Innym przykładem podobnego rozwiązania jest wspominana już operacja złączenia, którą można zastąpić operacją selekcji poprzedzoną operacją iloczynu kartezjańskiego:

$$R \bowtie_{\langle \text{warunek} \rangle} S \equiv \sigma_{\langle \text{warunek} \rangle} (R \times S)$$

Podobnie, operacja złączenia naturalnego może być wyrażana za pomocą operacji iloczynu kartezjańskiego poprzedzonej operacją zmiany nazwy i poprzedzającej odpowiednie operacje selekcji i projekcji. Oznacza to, że różne operacje złączenia także *nie są ściśle wymagane* z punktu widzenia możliwości wyrażania zapytań w oparciu o algebrę relacyjną. Złączenie w postaci wyodrębnionych operacji jest jednak bardzo ważne z uwagi na wygodę stosowania i ogromną popularność tego typu zapytań w wielu aplikacjach baz danych. Pozostałe operacje zostały włączone do algebry relacyjnej raczej dla wygody niż z konieczności. Jedną z nich (operację dzielenia) omówimy w kolejnym punkcie.

### 8.3.4. Operacja dzielenia

Operacja dzielenia (oznaczana za pomocą symbolu  $\div$ ) jest przydatna podczas definiowania specjalnych rodzajów zapytań w aplikacjach baz danych. Przykład takiego zapytania może mieć postać: *wyszukaj nazwiska pracowników, którzy pracują nad **wszystkimi** projektami, nad którymi pracuje także Jan Nowak*. Aby wyrazić to zapytanie za pomocą operacji dzielenia, musimy postępować zgodnie z następującą procedurą. Najpierw musimy stworzyć listę numerów projektów, nad którymi pracuje Jan Nowak — umieścimy ją w pośredniej relacji nazwanej PROJEKTY\_NOWAKA:

$$\text{NOWAK} \leftarrow \sigma_{\text{IMIĘ} = \text{'Jan'} \wedge \text{NAZWISKO} = \text{'Nowak'}} (\text{PRACOWNIK})$$

$$\text{PROJEKTY\_NOWAKA} \leftarrow \pi_{\text{NR\_PROJ}} (\text{PRACUJE\_NAD} \bowtie_{\text{PESELPRAC} = \text{PESEL NOWAK}})$$

Następnie musimy stworzyć relację zawierającą krotki w postaci  $\langle \text{NR\_PROJ}, \text{PESELPRAC} \rangle$ , które będą reprezentowały fakt pracy właściciela numeru PESEL reprezentowanego przez atrybut PESELPRAC nad projektem oznaczonym numerem NR\_PROJ — zadanie to zrealizujemy za pomocą pośredniej relacji PESEL\_NR\_PROJEKTÓW:

$$\text{PESEL\_NR\_PROJEKTÓW} \leftarrow \pi_{\text{PESELPRAC}, \text{NR\_PROJ}} (\text{PRACUJE\_NAD})$$

I wreszcie musimy zastosować dla obu relacji pośrednich operację dzielenia, która umożliwi nam wygenerowanie relacji pośredniej z interesującymi nas numerami PESEL:

$$\text{NUMERY\_PESEL}(\text{PESEL}) \leftarrow \text{PESEL\_NR\_PROJEKTÓW} \div \text{PROJEKTY\_NOWAKA}$$

$$\text{WYNIK} \leftarrow \pi_{\text{IMIĘ}, \text{NAZWISKO}} (\text{NUMERY\_PESEL} * \text{PRACOWNIK})$$

Wyniki wykonywania powyższych operacji na przykładowych danych przedstawiono na rysunku 8.8(a).

Operacja dzielenia jest stosowana dla dwóch relacji, zatem ma postać  $R(Z) \div S(X)$ , gdzie atrybuty  $S$  są podzbiorem atrybutów  $R$  (a więc  $X \subseteq Z$ ). Niech  $Y = Z - X$  (zatem niech  $Z = XY$ ); oznaczałoby to, że  $Y$  ma być zbiorem tych atrybutów relacji  $R$ , które nie należą do relacji  $S$ .

Wynikiem operacji dzielenia jest relacja  $T(Y)$ , która zawiera krotkę  $t$ , jeśli w relacji  $R$  istnieją takie krotki  $t_R$ , że  $t_R[Y] = t$ , oraz jeśli spełnione jest równanie  $t_R[X] = t_S$  dla każdej krotki  $t_S$  w relacji  $S$ . Oznacza to, że aby krotka  $t$  mogła się znaleźć w relacji wynikowej  $T$  operacji dzielenia, wszystkie wartości w tej krotce muszą występować w relacji  $R$  w kombinacji z *każdą* krotką  $S$ . Warto pamiętać, że zgodnie z definicją operacji dzielenia, krotki w relacji występującej w roli mianownika ograniczają relację występującą w roli licznika przez dopuszczanie do relacji wynikowej tylko tych krotek, które odpowiadają wszystkim wartościom istniejącym w mianowniku operacji. W zasadzie, znajomość tych wartości nie jest niezbędna, ponieważ można je wyznaczyć za pomocą innej operacji — tak jak w relacji PROJEKTY\_NOWAKA we wcześniejszym przykładzie.

Na rysunku 8.8(b) przedstawiono operację dzielenia, w której  $X = \{A\}$ ,  $Y = \{B\}$  oraz  $Z = \{A, B\}$ . Łatwo zauważyć, że krotki (wartości)  $b_1$  i  $b_4$  występują w relacji  $R$  w kombinacjach ze wszystkimi trzema krotkami relacji  $S$ , dlatego właśnie te dwie krotki relacji  $R$  znalazły się w relacji wynikowej  $T$ . Żadna z pozostałych wartości atrybutu  $B$  w relacji  $R$  nie występuje w kombinacji ze wszystkimi krotkami relacji  $S$  i w związku z tym nie jest wybierana ( $b_2$  nie występuje w kombinacji z wartością  $a_2$ , natomiast  $b_3$  nie występuje w kombinacji z wartością  $a_1$ ).

(a)

PESEL_NR_PROJEKTÓW	
PPESEL	NR_PROJEKTU
65010912345	1
65010912345	2
62091502054	3
72073110039	1
72073110039	2
55120834509	2
55120834509	3
55120834509	10
55120834509	20
68011932514	30
68011932514	10
69032923149	10
69032923149	30
41062013258	30
41062013258	20
37111045873	20

PROJEKTY\_SZEWCZYKA

NR_PROJEKTU
1
2

NUMERY\_PESEL

PESEL
65010912345
72073110039

(b)

R	
A	B
a1	b1
a2	b1
a3	b1
a4	b1
a1	b2
a3	b2
a2	b3
a3	b3
a4	b3
a1	b4
a2	b4
a3	b4

S

A
a1
a2
a3

T

B
b1
b4

RYSUNEK 8.8. Operacja dzielenia. (a) Dzielenie relacji PESEL\_NR\_PROJEKTÓW przez relację PROJEKTY\_NOWAKA. (b)  $T \leftarrow R \div S$

Operacja dzielenia może być wyrażona za pomocą sekwencji operacji projekcji, iloczynu kartezjańskiego i różnicy:

$$\begin{aligned} T_1 &\leftarrow \pi_r(R) \\ T_2 &\leftarrow \pi_r((S \times T_1) - R) \\ T &\leftarrow T_1 - T_2 \end{aligned}$$

Operacja dzielenia została zdefiniowana dla wygody użytkowników, którzy często muszą wykonywać zapytania oparte na *uniwersalnej kwantyfikacji* (patrz punkt 8.6.7) lub warunku występowania *wszystkich* wartości. Większość implementacji relacyjnych systemów zarządzania relacyjnymi bazami danych obsługujących standard SQL jako główny język zapytań nie implementuje bezpośrednio operacji dzielenia. Standard SQL oferuje nieco inny sposób obsługi tego typu zapytań (patrz zapytania Z3A i Z4B z punktu 7.1.4). W tabeli 8.1 przedstawiono listę różnych operacji podstawowej algebry relacyjnej, które omówiliśmy w tym podrozdziale.

TABELA 8.1. Operacje algebry relacyjnej

Operacja	Przeznaczenie	Notacja
selekcja	Wybiera z relacji $R$ wszystkie krotki, które spełniają warunek selekcji	$\sigma_{\langle \text{WARUNEK SELEKCJI} \rangle}(R)$
projekcja	Zwraca nową relację, która zawiera tylko niektóre atrybuty relacji $R$ ; usuwa ewentualne powtórzenia krotek	$\pi_{\langle \text{LISTA ATRYBUTÓW} \rangle}(R)$
złączenie theta	Zwraca wszystkie te kombinacje krotek z relacji $R_1$ i $R_2$ , które spełniają warunek złączenia	$R_1 \bowtie_{\langle \text{WARUNEK ZŁĄCZENIA} \rangle} R_2$
równo-złączenie	Zwraca wszystkie te kombinacje krotek z relacji $R_1$ i $R_2$ , które spełniają warunek złączenia (dopuszczalne są wyłącznie warunki równościowe)	$R_1 \bowtie_{\langle \text{WARUNEK ZŁĄCZENIA} \rangle} R_2$ LUB $R_1 \bowtie_{\langle \text{ATRYBUTY ZŁĄCZENIA 1} \rangle, \langle \text{ATRYBUTY ZŁĄCZENIA 2} \rangle} R_2$
złączenie naturalne	Działa tak samo jak operacja równo-złączenia z tą różnicą, że atrybuty złączenia pochodzące z relacji $R_2$ nie są umieszczane w relacji wynikowej; jeśli atrybuty złączenia mają w obu relacjach takie same nazwy, w ogóle nie musimy ich wymienić w warunku złączenia	$R_1^*_{\langle \text{WARUNEK ZŁĄCZENIA} \rangle} R_2$ LUB $R_1^*_{\langle \text{ATRYBUTY ZŁĄCZENIA 1} \rangle, \langle \text{ATRYBUTY ZŁĄCZENIA 2} \rangle} R_2$
suma	Zwraca relację zawierającą wszystkie krotki występujące w relacji $R_1$ , w relacji $R_2$ lub w obu relacjach jednocześnie; relacje $R_1$ i $R_2$ muszą spełniać warunek zgodności złączenia	$R_1 \cup R_2$
część wspólna	Zwraca relację, która zawiera wszystkie krotki występujące jednocześnie w relacji $R_1$ i relacji $R_2$ ; relacje $R_1$ i $R_2$ muszą spełniać warunek zgodności złączenia	$R_1 \cap R_2$
różnica	Zwraca relację zawierającą wszystkie krotki, które występują w relacji $R_1$ , ale nie występują w relacji $R_2$ ; relacje $R_1$ i $R_2$ muszą spełniać warunek zgodności złączenia	$R_1 \setminus R_2$
iloczyn kartezyjański	Zwraca relację zawierającą wszystkie atrybuty relacji $R_1$ i $R_2$ , w której są zawarte wszystkie możliwe kombinacje krotek pochodzących z obu relacji wejściowych	$R_1 \times R_2$
dzielenie	Zwraca relację $R(X)$ zawierającą wszystkie krotki $t[X]$ z relacji $R_1[Z]$ , które występują w relacji $R_1$ w kombinacji ze wszystkimi krotkami relacji $R_2(Y)$ , gdzie $Z = Z \cup Y$	$R_1(Z) \div R_2(Y)$

### 8.3.5. Notacja drzew zapytań

W tym punkcie przedstawiamy notację standardowo używaną w relacyjnych SZBD do wewnętrznego reprezentowania zapytań. Jest to notacja *drzew zapytań* (nazywanych czasem *drzewami wykonywania* lub *przetwarzania zapytań*). Takie drzewa obejmują operacje algebry relacyjnej i są używane jako struktura danych do wewnętrznego reprezentowania zapytań w relacyjnych SZBD.

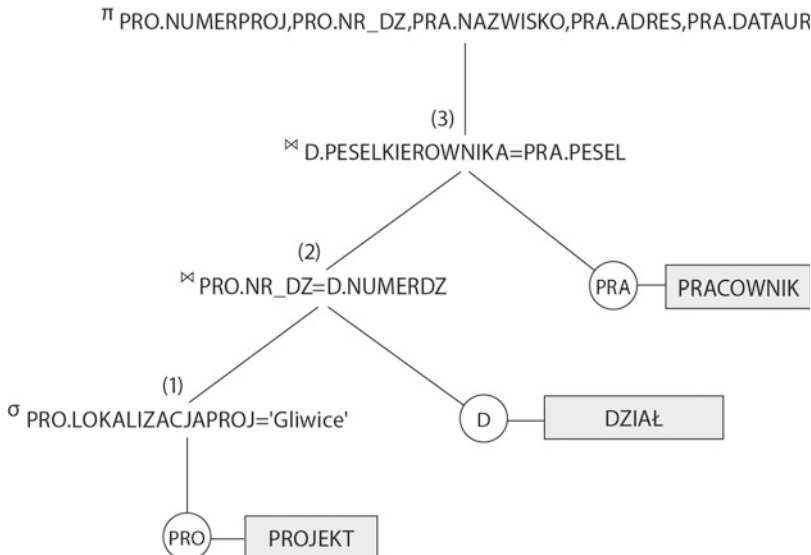
**Drzewo zapytań** to drzewiasta struktura danych odpowiadająca wyrażeniom algebry relacyjnej. Relacje wejściowe zapytania są reprezentowane jako *liście* drzewa, a operacje algebry relacyjnej — jako węzły wewnętrzne. Przetwarzanie drzewa zapytań polega na wykonywaniu operacji z wewnętrznych węzłów, gdy dostępne są ich operandy (reprezentowane przez węzły podrzędne). Węzeł wewnętrzny jest następnie zastępowany relacją wynikową z wykonanej operacji. Przetwarzanie kończy się po wykonaniu operacji z korzenia i uzyskaniu wynikowej relacji dla zapytania.

Na rysunku 8.9 pokazane jest drzewo zapytań dla zapytania 2. (patrz punkt 6.3.1): *dla każdego projektu zlokalizowanego w Gliwicach podaj numer projektu, numer działu zarządzającego oraz nazwisko, adres i datę urodzenia kierownika tego działu*. To zapytanie jest określone dla schematu relacyjnego z rysunku 5.5 i odpowiada następującemu wyrażeniu algebry relacyjnej:

```

 $\pi$ NUMERPROJ, NR_DZ, NAZWISKO, ADRES,
 $\rightarrow$ DATAUR((( $\sigma$ LOKALIZACJAPROJ='Gliwice')(PROJEKT))
 $\bowtie$ NR_DZ=NUMERDZ(DZIAŁ))  $\bowtie$ PESELKIEROWNIKA=PESEL(PRACOWNIK))

```



RYSunEK 8.9. Drzewo zapytań odpowiadające wyrażeniu algebry relacyjnej dla zapytania Z2

Na rysunku 8.9 trzy liście: PRO, D i PRA reprezentują trzy relacje: PROJEKT, DZIAŁ i PRACOWNIK. Operacje algebry relacyjnej z tego wyrażenia są reprezentowane jako wewnętrzne węzły drzewa. Drzewo zapytań bezpośrednio wyznacza kolejność wykonywania operacji.



Aby wykonać Z2, węzeł oznaczony na rysunku 8.9 jako (1) musi rozpocząć przetwarzanie przed węzłem (2), ponieważ niektóre wynikowe krotki z operacji (1) muszą być dostępne przed rozpoczęciem wykonywania operacji (2). Podobnie węzeł (2) musi rozpocząć przetwarzanie i wygenerować wyniki przed rozpoczęciem wykonywania węzła (3) itd. Drzewo zapytań zapewnia dobrą wizualną reprezentację zapytania i pomaga je zrozumieć w kategoriach używanych operacji relacyjnych. Warto stosować takie drzewa jako dodatkowy sposób przedstawiania zapytań algebry relacyjnej. Do drzew zapytań wrócimy w trakcie omawiania przetwarzania i optymalizacji zapytań w rozdziałach 18. i 19.

## 8.4. Dodatkowe operacje relacyjne

Niektóre popularne żądania realizowane w bazach danych, których obsługa jest niezbędna w komercyjnych systemach zarządzania relacyjnymi bazami danych, nie mogą być realizowane wyłącznie w oparciu o operacje oryginalnej algebry relacyjnej (patrz podrozdziały 8.1, 8.2 i 8.3). W tym podrozdziale zdefiniujemy dodatkowe operacje, które umożliwiają wyrażanie tego typu żądań. Prezentowane operacje znacznie rozszerzają możliwości wyrażu algebry relacyjnej.

### 8.4.1. Uogólniona projekcja

Operacja uogólnionej projekcji to rozwinięcie umożliwiające umieszczanie na liście projekcji funkcji zależnych od atrybutów. Ogólna postać tej operacji to:

$$\pi_{F1, F2, \dots, Fn}(R)$$

$F1, F2, \dots, Fn$  to funkcje zależne od atrybutów z relacji  $R$ . Mogą one obejmować operacje arytmetyczne i stałe. Ta operacja jest przydatna w trakcie tworzenia raportów, gdzie w kolumnach wyników zapytania trzeba umieścić obliczane wartości.

Przyjrzyj się np. następującej relacji:

PRACOWNIK(PESEL, PENSJA, POTRĄCENIA, LATAPRACY)

Możliwe, że w raporcie trzeba wyświetlić następujące informacje:

PENSJA NETTO = PENSJA - POTRĄCENIA  
PREMIA = 2000 \* LATAPRACY  
PODATKI = 0,25 \* PENSJA

W tej sytuacji można zastosować uogólnioną projekcję w połączeniu ze zmianami nazw:

RAPORT  $\leftarrow \rho(\text{PESEL, PENSJANETTO, PREMIA, PODATKI})$   
( $\pi_{\text{PESEL, PENSJA-POTRĄCENIA, 2000*LATAPRACY, 0.25*PENSJA}}(\text{PRACOWNIK})$ )

### 8.4.2. Funkcje agregujące i mechanizm grupowania

Inny typ żądań, których nie można wyrażać za pomocą podstawowych operacji algebry relacyjnej, obejmuje określanie matematycznych **funkcji agregujących** na zbiorach wartości uzyskanych w bazie danych. Do przykładów tego rodzaju funkcji należy wyznaczanie śred-

niego i łącznego wynagrodzenia wszystkich pracowników firmy, a także łącznej liczby tych pracowników. Funkcje agregujące są wykorzystywane w prostych zapytaniach statystycznych, które umożliwiają generowanie podsumowań dla krotek przechowywanych w bazie danych. Do funkcji, które stosuje się dla zbiorów wartości numerycznych najczęściej, należą: *suma*, *średnia*, *wartość maksymalna* oraz *wartość minimalna*. Do zliczania krotek lub wartości wykorzystuje się funkcję *liczba*.

Innym często spotykanym elementem zapytań jest żądanie grupowania krotek relacji w oparciu o wartość któregoś z jej atrybutów oraz zastosowania funkcji agregacji *niezależnie dla każdej z utworzonych grup*. Przykładem takiego rozwiązania może być grupowanie krotek reprezentujących pracowników (w relacji PRACOWNIK) w oparciu o wartość atrybutu NRDZ — każda grupa będzie wówczas zawierała krotki reprezentujące pracowników zatrudnionych w tym samym dziale firmy. Możemy następnie utworzyć listę, na której dla każdej wartości atrybutu NRDZ znajdzie się np. średnia pensja lub liczba pracowników zatrudnionych w danym dziale.

Możemy zdefiniować operację funkcji agregacji, wykorzystując symbol  $\mathfrak{S}$  (nazywany pisanym „F”)<sup>6</sup> do określania żądań tego typu w postaci następujących wyrażeń:

$$\langle \text{atrybuty grupowania} \rangle \mathfrak{S} \langle \text{lista funkcji} \rangle (R)$$

gdzie  $\langle \text{atrybuty grupowania} \rangle$  mają postać listy wyznaczonych atrybutów relacji  $R$ , natomiast  $\langle \text{lista funkcji} \rangle$  składa się z par w postaci  $(\langle \text{funkcja} \rangle \langle \text{atrybut} \rangle)$ . W każdej z takich par element  $\langle \text{funkcja} \rangle$  reprezentuje jedną z dostępnych funkcji (takich jak suma, średnia, maksimum, minimum czy liczba), zaś element  $\langle \text{atrybut} \rangle$  jest nazwą jednego z atrybutów relacji wejściowej  $R$ . Relacja wynikowa zawiera nie tylko atrybuty grupowania (atrybuty grupujące), ale także po jednym atrybucie dla każdego elementu na liście funkcji. Przykładowo, aby wyszukać wszystkie numery działów firmy wraz z liczbami zatrudnionych w nich pracowników oraz ich średnimi pensjami, zmieniając jednocześnie nazwy atrybutów wynikowych, możemy użyć następującego wyrażenia:

$$PR(\text{NRDZIALU, LICZBA\_PRACOWNIKÓW, ŚREDNIA\_PENSJA})(\text{NRDZ } \mathfrak{S} \text{ liczba PESEL, średnia PENSJA}(\text{PRACOWNIK}))$$

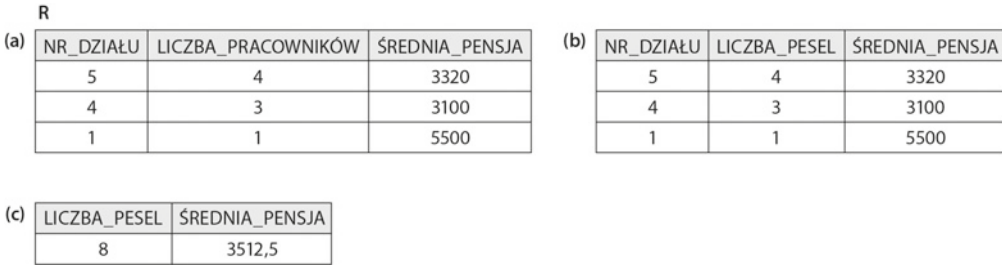
Relację wynikową otrzymaną po wykonaniu tego wyrażenia na relacji PRACOWNIK z rysunku 5.6 przedstawiono na rysunku 8.10(a).

W powyższym przykładzie określiliśmy listę nazw atrybutów (w nawiasach w operacji zmiany nazwy) dla wynikowej relacji  $R$ . Gdybyśmy nie zastosowali operacji zmiany nazwy, atrybuty w relacji wynikowej, które odpowiadają liście funkcji, miałyby postać konkatenacji nazw funkcji z nazwami atrybutów, a więc byłyby nazywane zgodnie ze schematem:  $\langle \text{funkcja} \rangle \_ \langle \text{atrybut} \rangle$ <sup>7</sup>. Przykładowo, na rysunku 8.10(b) przedstawiono wynik wykonania następującej operacji:

$$\text{NRDZ } \mathfrak{S} \text{ liczba PESEL, średnia PENSJA}(\text{PRACOWNIK})$$

<sup>6</sup> Nie ma jednej uzgodnionej notacji do określania funkcji agregujących. W niektórych przypadkach stosuje się np. pisaną literę „A”.

<sup>7</sup> Jest to arbitralnie wybrana notacja spójna z językiem SQL.



RYСУNEK 8.10. OPERACJA FUNKCJI AGREGUJĄCEJ.

- (A)  $P_{R(NR\_DZIAŁU, LICZBA\_PRACOWNIKÓW, ŚREDNIA\_PENSJA)}(NRDZ \mathcal{S}_{LICZBA\_PESEL, ŚREDNIA\_PENSJA}(PRACOWNIK))$ .
- (B)  $NRDZ \mathcal{S}_{LICZBA\_PESEL, ŚREDNIA\_PENSJA}(PRACOWNIK)$ .
- (C)  $\mathcal{S}_{LICZBA\_PESEL, ŚREDNIA\_PENSJA}(PRACOWNIK)$

Gdybyśmy nie określili żadnych atrybutów grupowania, wymienione funkcje zostałyby zastosowane dla *wszystkich krotek* w danej relacji, a więc relacja wynikowa składałaby się *tylko z jednej krotki*. Przykładowo, na rysunku 8.10(c) przedstawiono wynik następującej operacji:

$$\mathcal{S}_{liczba\_PESEL, średnia\_PENSJA}(PRACOWNIK)$$

Warto pamiętać, że ewentualne powtórzenia *nie są eliminowane* w czasie stosowania funkcji agregującej, co oznacza, że wyznaczane wartości funkcji suma lub średnia są zgodne ze standardową interpretacją odpowiednich działań<sup>8</sup>. W agregacji nie są tu jednak uwzględniane wartości puste (patrz punkt 7.1.7). Należy przy tym podkreślić, że wynikiem zastosowania funkcji agregacji jest relacja, nie liczba skalarna — dotyczy to także sytuacji, w których tym wynikiem jest jedna wartość. Ta właściwość czyni z algebry relacyjnej system zamknięty.

### 8.4.3. Rekurencyjne operacje domknięcia

Jeszcze innym typem operacji, która generalnie nie może być zdefiniowana w oryginalnej, podstawowej wersji algebry relacyjnej, jest **rekurencyjne domknięcie**. Operacja ta jest stosowana dla **rekurencyjnego powiązania** łączącego krotki tego samego typu, np. dla związku pomiędzy pracownikiem i jego przełożonym. Tego typu związki są opisywane za pomocą kluczy obcych — w tym przypadku klucza PESELPRZEŁOŻ relacji PRACOWNIK (patrz rysunki 5.5 i 5.6), który łączy krotkę reprezentującą pracownika (w roli podwładnego) z krotką reprezentującą innego pracownika (występującego w roli przełożonego). Przykładem rekurencyjnej operacji może być wyszukiwanie wszystkich podwładnych danego pracownika  $p$  na wszystkich poziomach — a więc wszystkich pracowników  $p'$  będących bezpośrednimi podwładnymi pracownika  $p$ , wszystkich pracowników  $p''$  występujących w roli bezpośrednich podwładnych pracownika  $p'$ , wszystkich pracowników  $p'''$  będących bezpośrednimi podwładnymi pracownika  $p''$  itd.

<sup>8</sup> W języku SQL istnieje specjalna opcja eliminowania powtórzeń przed zastosowaniem funkcji agregującej (wystarczy w zapytaniu użyć słowa kluczowego DISTINCT — patrz punkt 4.4.4).

Chociaż opisanie za pomocą wyrażenia algebry relacyjnej wszystkich pracowników występujących w roli podwładnych pracownika  $p$  na *określonym poziomie* jest stosunkowo łatwe (dzięki łączeniu tabeli z nią samą), znacznie trudniejsze okazuje się wyszukiwanie wszystkich podwładnych na *wszystkich* poziomach. Przykładowo, aby uzyskać numery PESEL wszystkich pracowników  $p'$ , którzy są bezpośrednimi podwładnymi pracownika  $p$  (a więc znajdują się *na pierwszym poziomie*), np. Józefa Bąka (patrz rysunek 5.6), możemy użyć następującej sekwencji operacji:

$$\text{PESEL\_BĄKA} \leftarrow \pi_{\text{PESEL}} (\sigma_{\text{IMIĘ} = \text{'Józef'} \wedge \text{NAZWISKO} = \text{'Bąk'}} (\text{PRACOWNIK}))$$

$$\text{KIEROWNICTWO}(\text{PESEL1}, \text{PESEL2}) \leftarrow \pi_{\text{PESEL}, \text{PESELSZEF}} (\text{PRACOWNIK})$$

$$\text{WYNIK1}(\text{PESEL}) \leftarrow \pi_{\text{PESEL1}} (\text{KIEROWNICTWO} \bowtie_{\text{PESEL2} = \text{PESEL}} \text{PESEL\_BĄKA})$$

Aby otrzymać listę wszystkich pracowników podlegających Józefowi Bąkowi na drugim poziomie — czyli wszystkich pracowników  $p''$  będących bezpośrednimi podwładnymi pewnego pracownika  $p'$ , który z kolei podlega bezpośrednio Józefowi Bąkowi — możemy użyć jeszcze jednej operacji **złączenia** dla wyniku pierwszego zapytania:

$$\text{WYNIK2}(\text{PESEL}) \leftarrow \pi_{\text{PESEL1}} (\text{KIEROWNICTWO} \bowtie_{\text{PESEL2} = \text{PESEL}} \text{WYNIK1})$$

Aby otrzymać zarówno zbiór pracowników będących podwładnymi Józefa Bąka na pierwszym poziomie, jak i zbiór tych pracowników, dla których Józef Bąk jest pośrednim przełożonym (czyli podwładnych drugiego poziomu), możemy zastosować operację sumy dla dwóch otrzymanych przed chwilą relacji wynikowych:

$$\text{WYNIK} \leftarrow \text{WYNIK1} \cup \text{WYNIK2}$$

Wyniki tych zapytań przedstawiono na rysunku 8.11. Chociaż istnieje możliwość wyszukania pracowników na każdym poziomie hierarchii kierowniczej i zastosowania dla tych relacji pośrednich operacji sumy, nie możemy zdefiniować uniwersalnego zapytania w postaci „wyszukaj podwładnych Józefa Bąka na wszystkich poziomach” bez zastosowania odpowiedniego mechanizmu pętli<sup>9</sup>. Z myślą o wyznaczaniu relacji rekurencyjnych na wszystkich poziomach rekurencji zaprojektowano operację nazwaną *domknięciem tranzytywnym*.

#### 8.4.4. Operacje złączenia zewnętrznego

Omówimy teraz niektóre z istniejących rozszerzeń podstawowej operacji złączenia, które są niezbędne do definiowania pewnych rodzajów zapytań. Opisane do tej pory operacje złączenia dopasowują krotki spełniające zdefiniowany warunek złączenia. Przykładowo, w przypadku operacji złączenia naturalnego mającej postać  $R * S$ , w relacji wynikowej znajdują się tylko te krotki relacji  $R$ , dla których istnieją odpowiednie krotki w relacji  $S$ , oraz tylko te krotki relacji  $S$ , dla których istnieją odpowiednie krotki w relacji  $R$ . Oznacza to, że krotki, dla których nie istnieją *pasujące (powiązane)* krotki w drugiej relacji, zostaną wyeliminowane z wyniku operacji złączenia. Wyeliminowane zostaną także krotki z wartością pustą w atrybutach złączenia. Tego typu złączenia, w których usuwane są niedopasowane krotki,

<sup>9</sup> Specjalna składnia dla operacji rekurencyjnego domknięcia jest oferowana w standardzie SQL3.

KIEROWNICTWO  
(Numer PESEL Bąka to 37111045873)  
(PESEL) (PESELSZEFA)

PESEL1	PESEL2
65010912345	55120834598
55120834598	37111045873
68011932514	41062013258
41062013258	37111045873
62091502054	55120834598
72073110039	55120834598
69032923149	41062013258
37111045873	brak

WYNIK1

PESEL
55120834598
41062013258

(Podwładni Bąka)

WYNIK2

PESEL
65010912345
68011932514
62091502054
72073110039
69032923149

(Podwładni  
podwładnych Bąka)

WYNIK

PESEL
65010912345
68011932514
62091502054
72073110039
69032923149
55120834598
41062013258

(WYNIK1  $\cup$  WYNIK2)

RYSUNEK 8.11. Dwupoziomowe zapytanie rekurencyjne

to **złączenia wewnętrzne**. Złączenia opisane wcześniej w podrozdziale 8.3 to właśnie złączenia wewnętrzne. Takie złączenia skutkują utratą informacji, gdy użytkownik chce, aby wynik obejmował wszystkie krotki z jednej lub kilku uwzględnianych relacji.

Istnieje zbiór operacji, nazywanych **złączeniami zewnętrznymi**, które mogą być stosowane w wyrażeniach, których relacje wynikowe mają uwzględniać informacje zawarte we wszystkich krotkach  $R$ , we wszystkich krotkach  $S$  lub we wszystkich krotkach obu tych relacji niezależnie od istnienia pasujących krotek w drugiej relacji operacji złączenia. W ten sposób można definiować zapytania, w których krotki pochodzące z obu tabel są łączone przez dopasowania odpowiednich wierszy, ale bez jednoczesnej utraty jakichkolwiek krotek spowodowanej brakiem pasujących wartości. Przykładowo, przypuśćmy, że chcemy wygenerować listę nazwisk wszystkich pracowników wraz z nazwami zarządzanych przez nich działów, *jeśli akurat kierują działami*; w przypadku pracowników, którzy nie zarządzają żadnymi działami, możemy ten fakt oznaczyć wartością pustą. Możemy w tym celu zastosować operację **lewego złączenia zewnętrznego** (oznaczanego symbolem  $\bowtie$ ) w następujących wyrażeniach:

TYMCZASOWY  $\leftarrow$  (PRACOWNIK  $\bowtie_{\text{PESEL} = \text{PESELPRZEŁOZ}}$  DZIAŁ)

WYNIK  $\leftarrow \pi_{\text{IMIE, INICJALDRIMIENTA, NAZWISKO, NAZWADZ}}(\text{TYMCZASOWY})$

Operacja lewego złączenia zewnętrznego kwalifikuje do relacji wynikowej wszystkie krotki z *pierwszej* (lub *lewej*) relacji  $R$  wyrażenia  $R \bowtie S$ ; następnie atrybuty relacji  $S$  w generowanej relacji wynikowej są wypełniane (lub *dopełniane*) wartościami pustymi. Wynik opisanej powyżej operacji lewego złączenia zewnętrznego przedstawiono na rysunku 8.12.

WYNIK

PIMIE	DIMIE	NAZWISKO	DNAZWA
Jan	Bartłomiej	Szewczyk	brak
Franciszek	Teodor	Wieszczyski	Badania
Alicja	Joanna	Zalewska	brak
Janina	Sylvia	Wojtczak	Administracja
Robert	Krzysztof	Napierski	brak
Joanna	Aldona	Englert	brak
Albert	Wojciech	Janiszewski	brak
Józef	Edward	Bąk	Centrala

RYSUNEK 8.12. Wynik operacji lewego złączenia zewnętrznego

Podobna operacja, nazywana **prawym złączeniem zewnętrznym** i oznaczana za pomocą symbolu  $\bowtie$ , kwalifikuje do relacji wynikowej wszystkie krotki z *drugiej* (lub *prawej*) relacji  $S$  wyrażenia  $R \bowtie S$ . Istnieje także trzecia operacja, tzw. **pełne złączenie zewnętrzne** (oznaczane symbolem  $\bowtie$ ), która kwalifikuje do relacji wynikowej wszystkie krotki występujące zarówno w lewej, jak i w prawej relacji, niezależnie od tego, czy w relacji znajdującej się po przeciwnej stronie operatora złączenia występują ich odpowiedniki (jeśli nie, odpowiednie atrybuty są dopełniane wartością pustą). Wymienione w tym punkcie trzy operacje złączania są częścią standardu SQL2 (patrz punkt 7.1.6). Te operacje zostały udostępnione później jako rozszerzenie algebry relacyjnej w reakcji na typowe wymogi użytkowników aplikacji biznesowych, aby wyświetlać kompletne informacje z wielu tabel. Czasem potrzebna jest pełna lista danych z wielu tabel niezależnie od tego, czy występują w nich pasujące wartości.

### 8.4.5. Operacja sumy zewnętrznej

Operacja **sumy zewnętrznej** została opracowana z myślą o wyznaczaniu sumy krotek pochodzących z pary relacji wejściowych w sytuacji, gdy relacje te mają niektóre wspólne atrybuty, ale *nie spełniają warunku zgodności złączenia*. Operacja tego typu pobiera sumę krotek występujących w parze relacji  $R(X, Y)$  oraz  $S(X, Z)$ , które są ze sobą **częściowo zgodne**, co oznacza, że tylko niektóre z ich atrybutów (przyjmijmy, że należące do zbioru  $X$ ) spełniają warunek zgodności złączenia. Atrybuty spełniające ten warunek są reprezentowane w relacji wynikowej tylko raz, natomiast atrybuty, które tego warunku nie spełniają, są przechowywane w relacji wyjściowej  $T(X, Y, Z)$  dwukrotnie. Operacja ta działa więc jak pełne złączenie zewnętrzne oparte na wspólnych atrybutach.

Mówimy, że para krotek  $t_1$  z relacji  $R$  i  $t_2$  z relacji  $S$  **pasuje do siebie**, jeśli  $t_1[X] = t_2[X]$ . Takie krotki są łączone (sumowane) w jedną krotkę wynikowej relacji  $T$ . Te krotki w relacjach wejściowych, które nie mają odpowiadających sobie krotek w relacji po drugiej stronie operatora sumowania, są dopełniane wartościami pustymi. Przykładowo, operację sumy zewnętrznej można zastosować dla dwóch relacji, których schematy mają następującą postać: STUDENT(Nazwisko, PESEL, Wydział, Promotor) oraz WYKŁADOWCA(Nazwisko, PESEL, Wydział, Tytuł). Krotki pochodzące z tych dwóch relacji są dopasowywane w oparciu o występujące w nich identyczne kombinacje wartości wspólnych atrybutów Nazwisko, PESEL i Wydział. Relacja wynikowa (STUDENT\_LUB\_WYKŁADOWCA) będzie więc zawierała następujące atrybuty:

STUDENT\_LUB\_WYKŁADOWCA(Nazwisko, PESEL, Wydział, Promotor, Tytuł)

W relacji wynikowej są umieszczane wszystkie krotki z obu relacji wejściowych, jednak krotki z takimi samymi kombinacjami wartości (Nazwisko, PESEL, Wydział) będą w tej relacji występowały tylko raz. Krotki występujące tylko w relacji STUDENT będą miały wartość pustą w atrybucie Tytuł, natomiast krotki występujące tylko w relacji WYKŁADOWCA będą miały wartość pustą w atrybucie Promotor. Krotki występujące w obu relacjach (reprezentujące studentów prowadzących wykłady) będą zawierały poprawne (niepuste) wartości we wszystkich atrybutach relacji wynikowej<sup>10</sup>.

Warto pamiętać, że ta sama osoba nadal może być reprezentowana w relacji wynikowej dwukrotnie (przez dwie różne krotki). Przykładowo, nasza prosta baza danych może zawierać informacje o studencie piątego roku na Wydziale Matematyki, który jest jednocześnie wykładowcą na Wydziale Informatyki. Chociaż taka osoba jest reprezentowana w relacjach STUDENT i WYKŁADOWCA przez dwie kroki z takimi samymi wartościami atrybutów (Nazwisko, PESEL), obie relacje nie będą zgodne z powodu różnych wartości atrybutu Wydział, zatem nie zostaną w relacji wynikowej scalone w jedną krotkę. Wynika to z faktu, że atrybut Wydział występuje w obu relacjach w dwóch różnych znaczeniach: w relacji STUDENT reprezentuje wydział, na którym dana osoba studiuje, natomiast w relacji WYKŁADOWCA reprezentuje wydział, na którym dana osoba jest zatrudniona w charakterze wykładowcy. Gdybyśmy chcieli sumować dane o studentach i wykładowcach wyłącznie w oparciu o kombinację takich samych wartości atrybutów (Nazwisko, PESEL), powinniśmy zmienić nazwę atrybutów Wydział w każdej z tabel wejściowych, aby prawidłowo odzwierciedlały ich zróżnicowane znaczenia, oraz wykluczyć je ze zbioru atrybutów spełniających warunek zgodnościłączenia. Przykładowo, można zmienić nazwy tych atrybutów na WydziałKierunku w relacji STUDENT i WydziałZatrudnienia w relacji WYKŁADOWCA.

## 8.5. Przykłady zapytań w algebrze relacyjnej

Przedstawimy teraz kilka dodatkowych przykładów ilustrujących stosowanie zaprezentowanych w tym rozdziale operacji algebry relacyjnej. Wszystkie zapytania odwołują się do bazy danych z rysunku 5.6. Ogólnie rzecz biorąc, to samo zapytanie może być wyrażane na

<sup>10</sup> Łatwo zauważyć, że jeśli atrybutamiłączenia są wszystkie wspólne atrybuty obu relacji sumowanych, operacja SUMY ZEWNĘTRZNEJ faktycznie jest równoważna operacji PEŁNEGO ZŁĄCZENIA ZEWNĘTRZNEGO.



wiele sposobów i w oparciu o wiele różnych operacji. W poniższych przykładach zaprezentujemy tylko po jednej wersji każdego z zapytań, a ewentualne konstruowanie równoważnych wyrażeń pozostawimy Czytelnikowi.

**Zapytanie 1.** Wyszukaj imiona i nazwiska oraz adresy wszystkich pracowników zatrudnionych w Dziale Badawczym.

$$\begin{aligned} \text{DZIAŁ\_BADAWCZY} &\leftarrow \sigma_{\text{NAZWADZ} = \text{'Badania'}}(\text{DZIAŁ}) \\ \text{PRACOWNICY\_DZIAŁU\_BADAWCZEGO} &\leftarrow (\text{DZIAŁ\_BADAWCZY} \bowtie_{\text{NUMERDZ} = \text{NRDZ}} \text{PRACOWNIK}) \\ \text{WYNIK} &\leftarrow \pi_{\text{IMIĘ, NAZWISKO, ADRES}}(\text{PRACOWNICY\_DZIAŁU\_BADAWCZEGO}) \end{aligned}$$

W formie jednowierszowej to zapytanie wygląda tak:

$$\pi_{\text{IMIĘ, NAZWISKO, ADRES}}(\sigma_{\text{NAZWADZ} = \text{'Badania'}}(\text{DZIAŁ}) \bowtie_{\text{NUMERDZ} = \text{NRDZ}} (\text{PRACOWNIK}))$$

To zapytanie można zdefiniować na wiele różnych sposobów; przykładowo, kolejność operacji złączenia i selekcji można odwrócić, można także zastąpić operację złączenia operacją złączenia naturalnego po zmianie nazwy jednego z atrybutów złączenia.

**Zapytanie 2.** Dla każdego projektu realizowanego w Gliwicach znajdź numer projektu, numer działu nadzorującego oraz nazwisko, adres i datę urodzenia kierownika zarządzającego danym działem.

$$\begin{aligned} \text{PROJEKTY\_REALIZOWANE\_W\_GLIWICACH} &\leftarrow \sigma_{\text{LOKALIZACJAPROJ} = \text{'Gliwice'}}(\text{PROJEKT}) \\ \text{DZIAŁ\_NADZORUJĄCY} &\leftarrow (\text{PROJEKTY\_REALIZOWANE\_W\_GLIWICACH} \bowtie_{\text{NR\_DZIAŁU} = \text{NRDZ}} \text{DZIAŁ}) \\ \text{KIEROWNIK\_DZIAŁU\_NADZORUJĄCEGO} &\leftarrow (\text{DZIAŁ\_NADZORUJĄCY} \bowtie_{\text{PESELKIEROWNIKA} = \text{PESEL}} \text{PRACOWNIK}) \\ \text{WYNIK} &\leftarrow \pi_{\text{NUMERPROJ, NR\_DZIAŁU, NAZWISKO, ADRES, DATAUR}}(\text{KIEROWNIK\_DZIAŁU\_NADZORUJĄCEGO}) \end{aligned}$$

W tym przykładzie najpierw pobierane są projekty zlokalizowane w Gliwicach, które następnie są złączane z działami nadzorującymi. Potem wynik złączany jest z kierownikami działów. Ostatni krok to operacja projekcji z użyciem potrzebnych atrybutów.

**Zapytanie 3.** Znajdź nazwiska pracowników przydzielonych do *wszystkich* projektów nadzorowanych przez dział oznaczony numerem 5.

$$\begin{aligned} \text{PROJEKTY\_DZIAŁU5} &\leftarrow \pi_{\text{NUMERPROJ}}(\sigma_{\text{NR\_DZIAŁU} = 5}(\text{PROJEKT})) \\ \text{PRACOWNICY\_PROJEKTY}(\text{PESELPRAC, NR\_PROJEKTU}) &\leftarrow \pi_{\text{PESELPRAC, NR\_PROJ}}(\text{PRACUJE\_NAD}) \\ \text{NUMERY\_PESEL\_PRACOWNIKÓW} &\leftarrow \text{PRACOWNICY\_PROJEKTY} \div \text{PROJEKTY\_DZIAŁU5} \\ \text{WYNIK} &\leftarrow \pi_{\text{NAZWISKO, IMIĘ}}(\text{NUMERY\_PESEL\_PRACOWNIKÓW} * \text{PRACOWNIK}) \end{aligned}$$

W tym zapytaniu najpierw tworzymy tabelę PROJEKTY\_DZIAŁU5 zawierającą numery wszystkich projektów nadzorowanych przez dział piąty. Potem tworzymy tabelę PRACOWNICY\_PROJEKTY z krotkami (PESEL, NR\_PROJEKTU) i stosujemy operację dzielenia. Zauważ, że nazwy atrybutów są zmieniane, dzięki czemu zostaną prawidłowo użyte w operacji dzielenia. W ostatnim kroku złączamy wynik dzielenia (obejmujący tylko numery PESEL) z tabelą PRACOWNIK, aby pobrać z niej atrybuty IMIĘ i NAZWISKO.

**Zapytanie 4.** Stwórz listę numerów projektów, w których realizację jest zaangażowany pracownik nazwiskiem 'Nowak' (niezależnie od tego, czy jego udział polega na pracy nad poszczególnymi projektami, czy na zarządzaniu działem nadzorującym te projekty).

$$\begin{aligned} \text{NOWAK}(\text{PESELPRAC}) &\leftarrow \pi_{\text{PESEL}}(\sigma_{\text{NAZWISKO} = \text{'Nowak'}}(\text{PRACOWNIK})) \\ \text{PROJEKTY\_OPRACOWYWANE\_PRZEZ\_NOWAKA} &\leftarrow \pi_{\text{NR\_PROJ}}(\text{PRACUJE\_NAD} * \text{SZEWCZYK}) \\ \text{KIEROWNICY} &\leftarrow \pi_{\text{NAZWISKO, NUMERDZ}}(\text{PRACOWNIK} \bowtie_{\text{PESEL} = \text{PESELKIEROWNIKA}} \text{DZIAŁ}) \\ \text{DZIAŁY\_KIEROWANE\_PRZEZ\_NOWAKA}(\text{NR\_DZIAŁU}) &\leftarrow \pi_{\text{NUMERDZ}}(\sigma_{\text{NAZWISKO} = \text{'Nowak'}}(\text{KIEROWNICY})) \\ \text{PROJEKTY\_ZARZĄDZANE\_PRZEZ\_NOWAKA} &\leftarrow \pi_{\text{NUMERPROJ}}(\text{DZIAŁY\_KIEROWANE\_PRZEZ\_NOWAKA} * \text{PROJEKT}) \\ \text{WYNIK} &\leftarrow (\text{PROJEKTY\_OPRACOWYWANE\_PRZEZ\_NOWAKA} \cup \text{PROJEKTY\_KIEROWANE\_PRZEZ\_NOWAKA}) \end{aligned}$$

W tym zapytaniu pobraliśmy numery projektów, nad którymi pracuje Nowak ( $\text{PROJEKTY\_OPRACOWYWANE\_PRZEZ\_NOWAKA}$ ). Następnie wczytaliśmy numery projektów, dla których Nowak jest kierownikiem działu nadzorującego ( $\text{PROJEKTY\_ZARZĄDZANE\_PRZEZ\_NOWAKA}$ ). W ostatnim kroku wyznaczyliśmy sumę relacji  $\text{PROJEKTY\_OPRACOWYWANE\_PRZEZ\_NOWAKA}$  i  $\text{PROJEKTY\_ZARZĄDZANE\_PRZEZ\_NOWAKA}$ . W postaci jednego wyrażenia zapytanie to wygląda tak:

$$\begin{aligned} &\pi_{\text{NUMERPROJ}}(\text{PRACUJE\_NAD} \bowtie_{\text{PESELPRAC} = \text{PESEL}} (\pi_{\text{PESEL}}(\sigma_{\text{NAZWISKO} = \text{'Nowak'}}(\text{PRACOWNIK}))) \cup \pi_{\text{NUMERPROJ}} \\ &((\pi_{\text{NUMERDZ}}(\sigma_{\text{NAZWISKO} = \text{'Nowak'}}(\pi_{\text{NAZWISKO, NUMERDZ}}(\text{PRACOWNIK}))) \bowtie_{\text{PESEL} = \text{PESELKIEROWNIKA}} \text{DZIAŁ})) \bowtie_{\text{NUMERDZ} = \text{NRDZ}} \text{PROJEKT}) \end{aligned}$$

**Zapytanie 5.** Wymień nazwiska wszystkich pracowników, dla których istnieją w bazie danych informacje o co najmniej dwóch członkach rodziny.

Mówiąc precyzyjnie, takiego zapytania nie da się skonstruować w oparciu o *podstawową (oryginalną) algebrę relacyjną*. Musimy użyć operacji funkcji agregacji z popularną funkcją agregacji licznik (ang. *COUNT*). Zakładamy, że członkowie rodziny *tego samego* pracownika mają *różne* wartości atrybutu  $\text{IMIĘ\_CZŁONKA\_RODZINY}$ .

$$\begin{aligned} T1(\text{PESEL, LICZBA\_CZŁONKÓW\_RODZINY}) &\leftarrow \\ &\text{PESELPRAC } \mathcal{S}_{\text{LICZNIK}} \text{ IMIĘ\_CZŁONKA\_RODZINY}(\text{CZŁONEK\_RODZINY}) \\ T2 &\leftarrow \sigma_{\text{LICZBA\_CZŁONKÓW\_RODZINY} \geq 2}(T1) \\ \text{WYNIK} &\leftarrow \pi_{\text{NAZWISKO, IMIĘ}}(T2 * \text{PRACOWNIK}) \end{aligned}$$

**Zapytanie 6.** Stwórz listę nazwisk wszystkich pracowników, którzy nie mają na utrzymaniu żadnych członków rodziny.

Jest to typowy przykład zapytania wykorzystującego operację odejmowania (różnicy zbiorów).

$$\begin{aligned} \text{WSZYSCY\_PRACOWNICY} &\leftarrow \pi_{\text{PESEL}}(\text{PRACOWNIK}) \\ \text{PRACOWNICY\_Z\_CZŁONKAMI\_RODZINY}(\text{PESEL}) &\leftarrow \pi_{\text{PESELPRAC}}(\text{CZŁONEK\_RODZINY}) \\ \text{PRACOWNICY\_BEZ\_CZŁONKÓW\_RODZINY} &\leftarrow (\text{WSZYSCY\_PRACOWNICY} - \text{PRACOWNICY\_Z\_CZŁONKAMI\_RODZINY}) \\ \text{WYNIK} &\leftarrow \pi_{\text{NAZWISKO, IMIĘ}}(\text{PRACOWNICY\_BEZ\_CZŁONKÓW\_RODZINY} * \text{PRACOWNIK}) \end{aligned}$$

Najpierw tworzona jest relacja WSZYSCY\_PRACOWNICY z wszystkimi numerami PESEL pracowników. Następnie tworzona jest tabela PRACOWNICY\_Z\_CZŁONKAMI\_RODZINY z numerami PESEL pracowników mających przynajmniej jednego członka rodziny. Potem stosujemy operację różnicy zbiorów, aby uzyskać relację PRACOWNICY\_BEZ\_CZŁONKÓW\_RODZINY z numerami PESEL pracowników niemających członków rodziny. Ta relacja jest łączana z relacją PRACOWNIK w celu pobrania potrzebnych atrybutów. Oto to zapytanie w postaci wyrażenia jednowierszowego:

$$\pi_{\text{NAZWISKO, IMIE}}((\pi_{\text{PESEL}}(\text{PRACOWNIK}) - \rho_{\text{PESEL}}(\pi_{\text{PESELPRAC}}(\text{CZŁONEK\_RODZINY}))) * \text{PRACOWNIK})$$

**Zapytanie 7.** Wymień nazwiska kierowników, którzy mają na utrzymaniu przynajmniej po jednym członku rodziny.

$$\text{KIEROWNICY}(\text{PESEL}) \leftarrow \pi_{\text{PESELKIEROWNIKA}}(\text{DZIAŁ})$$

$$\text{PRACOWNICY\_Z\_CZŁONKAMI\_RODZINY}(\text{PESEL}) \leftarrow \pi_{\text{PESELPRAC}}(\text{CZŁONEK\_RODZINY})$$

$$\text{KIEROWNICY\_Z\_CZŁONKAMI\_RODZINY} \leftarrow (\text{KIEROWNICY} \cap \text{PRACOWNICY\_Z\_CZŁONKAMI\_RODZINY})$$

$$\text{WYNIK} \leftarrow \pi_{\text{NAZWISKO, IMIE}}(\text{KIEROWNICY\_Z\_CZŁONKAMI\_RODZINY} * \text{PRACOWNIK})$$

W tym zapytaniu pobieramy numery PESEL kierowników (relacja KIEROWNICY) i numery PESEL pracowników mających przynajmniej jednego członka rodziny (relacja PRACOWNICY\_Z\_CZŁONKAMI\_RODZINY). Następnie wyznaczamy część wspólną zbiorów, aby otrzymać numery PESEL kierowników mających przynajmniej jednego członka rodziny.

Jak już wspominaliśmy, to samo zapytanie można w algebrze relacyjnej zdefiniować na wiele różnych sposobów. Przykładowo, te same operacje można w wielu przypadkach stosować w różnej kolejności. Co więcej, niektóre operacje można zastępować innymi — przykładowo, operację części wspólnej z zapytania 7. można z powodzeniem zastąpić operacjąłączenia naturalnego. Spróbuj zrealizować wszystkie przedstawione powyżej przykładowe zapytania za pomocą innych operacji<sup>11</sup>. Dla zapytań Z1, Z4 i Z6 pokazaliśmy, jak zapisać zapytania w formie pojedynczych wyrażeń algebry relacyjnej. Spróbuj zrobić to samo dla pozostałych zapytań. W rozdziałach 6. i 7. oraz podrozdziałach 8.6 i 8.7 prezentujemy sposób tworzenia zapytań w innych językach relacyjnych.

## 8.6. Relacyjny rachunek krotek

W tym i kolejnym podrozdziale wprowadzimy nieco inny formalny język zapytań dla modelu relacyjnego nazywanego **rachunkiem krotek**. W tym podrozdziale przedstawiamy **relacyjny rachunek krotek**, a w podrozdziale 8.7 — jego odmianę nazywaną **relacyjnym rachunkiem dziedzin**. W obu wersjach zdefiniowanie pojedynczego żądania wyszukania danych wymaga stworzenia jednego **deklaratywnego** wyrażenia, zatem użytkownik nie ma możliwości opisywania sposobu wykonywania (*kolejności operacji*) tak skonstruowanego zapytania. Wyrażenie tego rachunku określa, *co* ma być wynikiem zapytania, natomiast nic nie mówi o *sposobie* dojścia do tego wyniku. Oznacza to, że rachunek relacji należy traktować jako język **nieproceduralny**. Ta kluczowa własność odróżnia ten rachunek od algebry

<sup>11</sup> W czasie optymalizowania zapytań (patrz rozdziały 18. i 19.) system wybierze taką sekwencję operacji, która będzie odpowiadała przyjętej strategii efektywnego wykonywania zapytań.

relacyjnej, w której opis żądania wyszukiwania danych musi mieć postać odpowiedniej *sekwencji operacji* — można więc przyjąć, że algebra relacyjna jest **proceduralnym** sposobem wyrażania zapytań. Istnieje możliwość zagnieżdżania operacji tej algebry i tworzenia tym samym zapytań mających postać pojedynczych wyrażeń, jednak pomiędzy tymi operacjami składowymi zawsze będziemy musieli jawnie określać kolejność wykonywania. Porządek przetwarzania operacji dodatkowo wpływa na strategię wyznaczania wyniku. Pojedyncze wyrażenie rachunku krotek może być zapisywane na wiele różnych sposobów, jednak sam sposób zapisywania tego typu wyrażeń nie wpływa na sposób wykonywania zapytań.

Udowodniono, że każda operacja wyszukiwania danych, którą można zdefiniować w oparciu o podstawową algebrę relacyjną, może być określona także w postaci wyrażenia rachunku relacji (i odwrotnie); innymi słowy, **moc wyrażania** obu języków jest *identyczna*. W ten sposób doszliśmy do definicji pojęcia relacyjnie kompletnego języka. Relacyjny język zapytań  $L$  nazywamy językiem **relacyjnie kompletnym**, jeśli możemy w nim wyrazić dowolne zapytanie, które może być wyrażone w rachunku relacji. Relacyjna kompletność stała się istotnym elementem w procesie porównywania mocy wyrażania wysokopoziomowych języków zapytań. Jak się jednak przekonaliśmy w podrozdziale 8.4, niektóre zapytania, które często stosuje się w popularnych aplikacjach baz danych, nie mogą być wyrażane wyłącznie w oparciu o algebrę relacyjną lub rachunek relacji. Większość relacyjnych języków zapytań jest co prawda relacyjnie kompletna, ale oferuje *większą moc wyrażania* niż algebra relacyjna czy rachunek relacji, przede wszystkim z uwagi na dodatkową konieczność obsługi takich operacji jak funkcje agregacji, grupowanie czy porządkowanie. We wprowadzeniu do tego rozdziału wspomnieliśmy, że rachunek relacyjny jest ważny z dwóch powodów. Po pierwsze, ma solidne podstawy w postaci logiki matematycznej. Po drugie, standardowy język zapytań dla relacyjnych SZBD (SQL) jest oparty na rachunku relacyjnym krotek.

Wszystkie przykłady prezentowane w tym i następnym podrozdziale odwołują się do bazy danych przedstawionej na rysunkach 5.6 i 5.7. W naszych rozważaniach będziemy wykorzystywali te same zapytania, które demonstrowaliśmy w podrozdziale 8.5. W punktach 8.6.6, 8.6.7 i 8.6.8 omówimy zastosowania uniwersalnych kwantyfikatorów — punkty te mogą pominąć ci Czytelnicy, którzy są zainteresowani jedynie ogólnym wprowadzeniem do rachunku krotek.

### 8.6.1. Zmienne krotek i relacje zakresowe

Relacyjny rachunek krotek opiera się na określaniu pewnej liczby tzw. **zmiennych krotek** (ang. *tuple variables*). Każda z takich zmiennych *obejmuje* zwykle określoną relację składowaną w bazie danych, co oznacza, że wartością tej zmiennej może być dowolna krotka wybrana z takiej relacji. Proste zapytanie relacyjnego rachunku krotek ma postać:

$$\{t \mid \text{WARUNEK}(t)\}$$

gdzie  $t$  jest zmienną krotki, natomiast  $\text{WARUNEK}(t)$  jest wyrażeniem warunkowym (logicznym) o wartości TRUE lub FALSE dla różnych krotek podstawianych pod zmienną  $t$ . Wynikiem takiego zapytania jest zbiór wszystkich krotek  $t$ , które **spełniają** warunek  $\text{WARUNEK}(t)$  (dla których to wyrażenie warunkowe jest prawdziwe). Przykładowo, aby wyszukać wszystkich pracowników, których pensja przekracza 5000 złotych, możemy użyć następującego wyrażenia rachunku krotek:

$$\{t \mid \text{PRACOWNIK}(t) \wedge t.\text{PENSJA} > 5000\}$$

Warunek  $\text{PRACOWNIK}(t)$  określa, że **relacją zakresową** zmiennej krotki  $t$  jest relacja  $\text{PRACOWNIK}$ . Każda krotka  $t$  relacji  $\text{PRACOWNIK}$ , która spełnia warunek  $t.\text{PENSJA} > 5000$ , zostanie zakwalifikowana do wyniku powyższego wyrażenia. Warto pamiętać, że wyrażenie  $t.\text{PENSJA}$  odwołuje się do atrybutu  $\text{PENSJA}$  zmiennej krotki  $t$  — użyta notacja przypomina sposób oznaczania nazw atrybutów poprzedzającymi nazwami relacji lub aliasów w języku SQL (patrz rozdział 6.). Odpowiednikiem odwołania  $t.\text{PENSJA}$  w notacji prezentowanej w rozdziale 5. byłoby wyrażenie  $t[\text{PENSJA}]$ .

Powyższe zapytanie zwraca wartości wszystkich atrybutów dla każdej z wybranych krotek  $t$  relacji  $\text{PRACOWNIK}$ . Aby otrzymać tylko *niektóre* z tych atrybutów (np. tylko imiona i nazwiska), możemy użyć zapytania w postaci:

$$\{t.\text{IMIE}, t.\text{NAZWISKO} \mid \text{PRACOWNIK}(t) \text{ I } t.\text{PENSJA} > 5000\}$$

Mówiąc nieformalnie, w każdym wyrażeniu rachunku krotek musimy określić następujące informacje:

- Dla każdej zmiennej krotki  $t$  musimy wskazać **relację zakresową**  $R$  z krotkami  $t$ . Wartość jest definiowana za pomocą wyrażenia warunkowego w postaci  $R(t)$ . Jeśli nie zdefiniujesz relacji zakresowej, zmienna  $t$  będzie przyjmować wszystkie krotki „we wszechświecie”, ponieważ nie będzie ograniczona do jednej relacji.
- Musimy zdefiniować warunek, który pozwoli zakwalifikować do wyniku wyrażenia tylko określone kombinacje krotek. O ile zmienne krotek obejmują swoje relacje zakresowe, wartość wyrażenia warunkowego jest wyznaczana dla każdej możliwej kombinacji krotek — w ten sposób można **wyselekcjonować kombinacje**, dla których dany warunek jest spełniony (jest prawdziwy).
- Należy zdefiniować zbiór atrybutów, które mają zostać umieszczone w wyniku, tzw. **żądane atrybuty**. Wartości tych atrybutów będą przekazywane do relacji wynikowej dla każdej z zakwalifikowanych kombinacji krotek.

Zanim przystąpimy do omawiania formalnej składni relacyjnego rachunku krotek, przeanalizujemy jeszcze jedno zapytanie:

**Zapytanie 0.** Wyszukaj datę urodzenia i adres pracownika (lub pracowników), który nazywa się 'Jan B. Nowak'.

$$\text{Q0: } \{t.\text{DATAUR}, t.\text{ADRES} \mid \text{PRACOWNIK}(t) \text{ I } t.\text{IMIE} = \text{'Jan'} \text{ I } t.\text{INICJAŁDRIMIENIA} = \text{'B'} \text{ I } t.\text{NAZWISKO} = \text{'Nowak'}\}$$

W relacyjnym rachunku krotek najpierw określamy interesujące nas (żądane) atrybuty  $t.\text{DATAUR}$  i  $t.\text{ADRES}$  dla każdej z wybieranych krotek  $t$ . Następnie definiujemy warunek selekcji krotki (bezpośrednio za znakiem  $\mid$ ) — w tym przypadku określamy, że zmienna  $t$  musi być krotką relacji  $\text{PRACOWNIK}$  z wartościami atrybutów  $\text{IMIE}$ ,  $\text{INICJAŁDRIMIENIA}$  oraz  $\text{NAZWISKO}$  równymi odpowiednio 'Jan', 'B' i 'Nowak'.

## 8.6.2. Wyrażenia i wzory w relacyjnym rachunku krotek

Ogólne **wyrażenie** relacyjnego rachunku krotek ma następującą postać:

$$\{t_1.A_j, t_2.A_k, \dots, t_n.A_m \mid \text{WARUNEK}(t_1, t_2, \dots, t_n, t_{n+1}, t_{n+2}, \dots, t_{n+m})\}$$

gdzie  $t_1, t_2, \dots, t_n, t_{n+1}, t_{n+2}, \dots, t_{n+m}$  są zmiennymi krotek, każde  $A_i$  jest atrybutem relacji objętej przez zmienną  $t_i$ , natomiast WARUNEK jest **warunkiem** lub **wzorem**<sup>12</sup> relacyjnego rachunku krotek. Takie wzory składają się z **atomów** rachunku predykatów, z których każdy może mieć jedną z następujących postaci:

- (1) Atom w postaci  $R(t_i)$ , gdzie  $R$  jest nazwą relacji, natomiast  $t_i$  jest zmienną krotki. Atom w takiej postaci identyfikuje zakres zmiennej krotki  $t_i$  jako relację oznaczoną nazwą  $R$ . Jeśli krotka  $t_i$  należy do relacji  $R$ , wartość atomu to PRAWDA. W przeciwnym razie ta wartość to FAŁSZ.
- (2) Atom w postaci  $t_i A \text{ op } t_j B$ , gdzie **op** jest jednym z operatorów porównania należących do zbioru  $\{=, <, \leq, >, \geq, \neq\}$ ,  $t_i$  oraz  $t_j$  są zmiennymi krotek,  $A$  jest atrybutem relacji należącej do zakresu zmiennej  $t_i$ , natomiast  $B$  jest atrybutem relacji należącej do zakresu zmiennej  $t_j$ .
- (3) Atom w postaci  $t_i A \text{ op } c$  lub  $c \text{ op } t_j B$ , gdzie **op** jest jednym z operatorów porównania należących do zbioru  $\{=, <, \leq, >, \geq, \neq\}$ ,  $t_i$  oraz  $t_j$  są zmiennymi krotek,  $A$  jest atrybutem relacji należącej do zakresu zmiennej  $t_i$ ,  $B$  jest atrybutem relacji należącej do zakresu zmiennej  $t_j$ , natomiast  $c$  jest wartością stałą.

Każda z wymienionych powyżej postaci atomów może mieć albo wartość PRAWDA, albo wartość FAŁSZ dla określonej kombinacji krotek; w tym kontekście często używa się określenia **wartości prawdziwości** atomu. Generalnie, zmienna krotki  $t$  obejmuje wszystkie możliwe krotki *we wszechświecie*. W przypadku atomów mających postać  $R(t)$ , jeśli zmienna  $t$  będzie reprezentowała krotkę *należącą do określonej relacji*  $R$ , takie atomy będą miały wartość PRAWDA; w przeciwnym przypadku atomy w tej postaci będą miały wartość FAŁSZ. W przypadku atomów 2. i 3. typu, jeśli zmienne krotek będą reprezentowały takie krotki, w których wartości określonych atrybutów będą spełniały dany warunek, takie atomy będą miały wartość PRAWDA.

**Wzór** (lub warunek logiczny) składa się z jednego lub więcej atomów połączonych ze sobą za pomocą logicznych operatorów **I**, **LUB** oraz **NIE** — jego rekurencyjna definicja składa się z reguł 1. i 2.:

- *Reguła 1.* Każdy atom jest wzorem.
- *Reguła 2.* Jeśli  $F_1$  i  $F_2$  są wzorami, wzorami są także wyrażenia  $(F_1 \text{ I } F_2)$ ,  $(F_1 \text{ LUB } F_2)$ , **NIE**( $F_1$ ) oraz **NIE**( $F_2$ ). Wartości prawdziwości tych wzorów wynikają oczywiście z wartości prawdziwości ich wzorów składowych  $F_1$  i  $F_2$  — są wyznaczone według następującego schematu:
  - a)  $(F_1 \text{ I } F_2)$  jest PRAWDZIWY, jeśli zarówno wzór  $F_1$ , jak i wzór  $F_2$  jest PRAWDZIWY; w przeciwnym przypadku wzór ten jest FAŁSZYWY.
  - b)  $(F_1 \text{ LUB } F_2)$  jest FAŁSZYWY, jeśli FAŁSZYWY jest zarówno wzór  $F_1$ , jak wzór  $F_2$ ; w przeciwnym przypadku wzór ten jest PRAWDZIWY.
  - c) **NIE**( $F_1$ ) jest PRAWDZIWY, jeśli wzór  $F_1$  jest FAŁSZYWY; jeśli natomiast wzór  $F_1$  jest PRAWDZIWY, wzór **NIE**( $F_1$ ) jest FAŁSZYWY.
  - d) **NIE**( $F_2$ ) jest PRAWDZIWY, jeśli wzór  $F_2$  jest FAŁSZYWY; jeśli natomiast wzór  $F_2$  jest PRAWDZIWY, wzór **NIE**( $F_2$ ) jest FAŁSZYWY.

<sup>12</sup> W logice matematycznej nazywanym **poprawnie zbudowaną formułą** (ang. *well-formed formula* — WFF).



### 8.6.3. Kwantyfikatory uniwersalne i egzystencjalne

We wzorach można dodatkowo wykorzystywać dwa specjalne symbole nazywane **kwantifikatorami** — są to **kwantyfikatory uniwersalne** (nazywane także **ogólnymi** i oznaczane symbolem  $\forall$ ) oraz **kwantyfikatory egzystencjalne** (nazywane także **szczegółowymi** i oznaczane symbolem  $\exists$ ). Wartości prawdziwości dla wzorów z kwantifikatorami są opisywane przez dwie przedstawione poniżej reguły (oznaczone numerami 3 i 4); najpierw jednak musimy zdefiniować pojęcia wolnych i związanych zmiennych krotek wewnątrz wzorów. Mówiąc nieformalnie, zmienna krotki  $t$  jest związana, jeśli została poprzedzona kwantyfikatorem, a więc jeśli występuje w klauzuli  $(\exists t)$  lub  $(\forall t)$ ; w przeciwnym przypadku mówimy, że zmienna krotki  $t$  jest wolna. Mówiąc formalnie, zmienną krotki definiujemy we wzorze jako **wolną** lub **związaną** w oparciu o następujące reguły:

- Wystąpienie zmiennej krotki we wzorze  $F$ , który *jest atomem*, jest wolne w tym wzorze.
- Wystąpienie zmiennej krotki  $t$  jest wolne lub związane we wzorze składającym się z operatorów logicznych — a więc mającym postać  $(F_1 \text{ I } F_2)$ ,  $(F_1 \text{ LUB } F_2)$ ,  $\text{NIE}(F_1)$  lub  $\text{NIE}(F_2)$  — w zależności od tego, czy jest wolne lub związane we wzorze  $F_1$  lub  $F_2$  (oczywiście, jeśli występuje w którymś z tych wzorów). Warto pamiętać, że we wzorze mającym postać  $F = (F_1 \text{ I } F_2)$  lub  $F = (F_1 \text{ LUB } F_2)$  zmienna krotki może być wolna we wzorze składowym  $F_1$  oraz związana we wzorze składowym  $F_2$  (lub odwrotnie); w takim przypadku jedno wystąpienie zmiennej krotki we wzorze  $F$  jest związane, drugie jest wolne.
- Wszystkie *wolne* wystąpienia zmiennej krotki  $t$  we wzorze  $F$  są **związane** we wzorze  $F'$  mającym postać  $F' = (\exists t)(F)$  lub  $F' = (\forall t)(F)$ . Zmienna krotki jest w takim przypadku związana kwantyfikatorem użytym we wzorze  $F'$ . Przykładowo, przeanalizujmy następujące wzory:

$F_1 : d. \text{NAZWADZ} = \text{'Badania'}$

$F_2 : (\exists t)(d. \text{NUMERDZ} = t. \text{NRDZ})$

$F_3 : (\forall d)(d. \text{PESELKIEROWNIKA} = \text{'55120834598'})$

Zmienna krotki  $d$  jest *wolna* zarówno we wzorze  $F_1$ , jak i we wzorze  $F_2$ , ale jest związana kwantyfikatorem uniwersalnym ( $\forall$ ) we wzorze  $F_3$ . Zmienna  $t$  jest związana kwantyfikatorem egzystencjalnym ( $\exists$ ) we wzorze  $F_2$ .

Możemy teraz przedstawić formalne reguły 3. i 4., które stanowią dopełnienie rozpoczętej wcześniej definicji wzoru:

- *Reguła 3.* Jeśli  $F$  jest wzorem, wówczas wzorem jest także wyrażenie  $(\exists t)(F)$ , gdzie  $t$  jest zmienną krotki. Wzór  $(\exists t)(F)$  jest PRAWDZIWY, jeśli wzór  $F$  jest PRAWDZIWY dla  *pewnych*  (lub przynajmniej jednej z) krotek przypisanych do wolnych wystąpień zmiennej  $t$  we wzorze  $F$ ; w przeciwnym przypadku wzór  $(\exists t)(F)$  jest FAŁSZYWY.
- *Reguła 4.* Jeśli  $F$  jest wzorem, wówczas wzorem jest także wyrażenie  $(\forall t)(F)$ , gdzie  $t$  jest zmienną krotki. Wzór  $(\forall t)(F)$  jest PRAWDZIWY, jeśli wzór  $F$  jest PRAWDZIWY dla *każdej krotki* (we wszechświecie) przypisanej do wolnych wystąpień zmiennej  $t$  we wzorze  $F$ ; w przeciwnym przypadku wzór  $(\forall t)(F)$  jest FAŁSZYWY.



Kwantyfikator  $(\exists)$  jest nazywany egzystencjalnym, ponieważ wzór  $(\exists t)(F)$  jest PRAWDZIWIY, jeśli „istnieje” pewna krotka, dla której wzór  $F$  jest PRAWDZIWIY. W przypadku kwantyfikatora uniwersalnego, wzór  $(\forall t)(F)$  jest PRAWDZIWIY, jeśli każda możliwa krotka, która może być przypisana do wolnego wystąpienia zmiennej krotki  $t$  we wzorze  $F$ , zastępuje to wystąpienie  $t$ , oraz jeśli wzór  $F$  jest PRAWDZIWIY dla *każdego takiego zastąpienia*. Kwantyfikator uniwersalny jest niekiedy nazywany kwantyfikatorem ogólnym lub kwantyfikatorem *dla każdego*, ponieważ każda krotka we *wszechświecie* krotek musi zapewniać PRAWDZIWOŚĆ wzoru  $F$ , aby cały wzór opatrzony kwantyfikatorem był PRAWDZIWIY.

#### 8.6.4. Przykładowe zapytania w relacyjnym rachunku krotek

Wykorzystamy teraz niektóre z zapytań zaprezentowanych w podrozdziale 8.5 do zdemonstrowania metod konstruowania równoważnych konstrukcji w algebrze relacyjnej i w rachunku relacji. Warto zauważyć, że niektóre zapytania są znacznie łatwiejsze do wyrażenia w algebrze relacyjnej niż w rachunku relacji, natomiast inne wprost przeciwnie — można łatwiej zapisywać je w rachunku relacji niż w oparciu o operacje algebry relacyjnej.

**Zapytanie 1.** Wyszukaj nazwiska i adresy wszystkich pracowników zatrudnionych w dziale badawczym.

**Z1:**  $\{t.IMIĘ, t.NAZWISKO, t.ADRES \mid \text{PRACOWNIK}(t) \wedge (\exists d)(\text{DZIAŁ}(d) \wedge d.NAZWADZ = \text{'Badania'} \wedge d.NUMERDZ = t.NRDZ)\}$

Jedynymi wolnymi zmiennymi krotek w wyrażeniu algebry relacyjnej zawsze powinny być te zmienne, które znajdują się po lewej stronie znaku ( $\mid$ ). W zapytaniu Z1 jedyną wolną zmienną jest  $t$ ; zmienna ta jest *kolejno wiązana* z każdą krotką relacji PRACOWNIK. Jeśli istnieją krotki *spełniające warunki* zdefiniowane w zapytaniu Z1, wynikiem wykonania tego zapytania będzie relacja zawierająca atrybuty IMIĘ, NAZWISKO oraz ADRES dla każdej z takich krotek. Warunki PRACOWNIK( $t$ ) oraz DZIAŁ( $d$ ) określają relacje należące do zakresów zmiennych  $t$  i  $d$ . Wyrażenie  $d.NAZWADZ = \text{'Badania'}$  jest **warunkiem selekcji** i odpowiada operacji selekcji z algebry relacyjnej, natomiast wyrażenie  $d.NUMERDZ = t.NRDZ$  jest **warunkiem złączenia** i pełni rolę podobną do stosowanej w algebrze relacyjnej operacji złączenia wewnętrznego (patrz podrozdział 8.3).

**Zapytanie 2.** Dla każdego projektu realizowanego w Gliwicach wymień numer projektu, numer działu nadzorującego oraz nazwisko, datę urodzenia i adres kierownika kierującego tym działem.

**Z2:**  $\{p.NUMERPROJ, p.NR\_DZ, m.NAZWISKO, m.DATAUR, m.ADRES \mid \text{PROJEKT}(p) \wedge \text{PRACOWNIK}(m) \wedge p.LOKALIZACJAPROJ = \text{'Gliwice'} \wedge ((\exists d)(\text{DZIAŁ}(d) \wedge p.NR\_DZ = d.NUMERDZ \wedge d.PESELKIEROWNIKA = m.PESEL))\}$

W zapytaniu Z2 istnieją dwie wolne zmienne krotek:  $p$  i  $m$ . Zmienna krotek  $d$  jest związana z kwantyfikatorem egzystencjalnym (szczegółowym). Warunek tego zapytania jest wyznaczany osobno dla każdej kombinacji krotek przypisanych do zmiennych  $p$  i  $m$  — ze wszystkich możliwych kombinacji krotek, z którymi ta para zmiennych jest związana, do relacji wynikowej są kwalifikowane tylko te kombinacje, które spełniają warunek zapytania.

Wiele zmiennych krotek w jednym zapytaniu może obejmować swoim zasięgiem tę samą relację. Przykładowo, aby określić zapytanie Z8 (dla każdego pracownika wyszukaj imię i nazwisko oraz imię i nazwisko jego bezpośredniego przełożonego), musimy zdefiniować dwie zmienne krotek,  $p$  i  $s$  (reprezentujące odpowiednio krotki pracowników i przełożonych), które obejmują tę samą relację PRACOWNIK:

**Z8:**  $\{p.IMIE, p.NAZWISKO, s.IMIE, s.NAZWISKO \mid PRACOWNIK(p) \text{ I } PRACOWNIK(s) \text{ I } p.PESELPRZEŁOŻ = s.PESEL\}$

**Zapytanie 3'.** Znajdź nazwiska wszystkich pracowników, którzy pracują nad  *pewnym*  projektem nadzorowanym przez dział oznaczony numerem piątym. To zapytanie jest pewną odmianą wcześniejszego zapytania 3., w której słowo *wszystkich* zostało zastąpione słowem *pewnym*. W tym przypadku musimy zdefiniować dwa warunki złączenia i dwa kwantyfikatory egzystencjalne (szczegółowe).

**Z3':**  $\{p.NAZWISKO, p.IMIE \mid PRACOWNIK(p) \text{ I } ((\exists x)(\exists w)(PROJEKT(x) \text{ I } PRACUJE\_NAD(w) \text{ I } x.NR\_DZ = 5 \text{ I } w.PESELPRAC = p.PESEL \text{ I } x.NUMERPROJ = w.NUMERPROJ)))\}$

**Zapytanie 4.** Stwórz listę numerów projektów, w których realizację jest zaangażowany (albo w roli osoby opracowującej, albo w roli kierownika działu nadzorującego dany projekt) pracownik nazwiskiem 'Nowak'.

**Z4:**  $\{p.NUMERPROJ \mid PROJEKT(p) \text{ I } (((\exists e)(\exists w)(PRACOWNIK(e) \text{ I } PRACUJE\_NAD(w) \text{ I } w.NR\_PROJ = p.NUMERPROJ \text{ I } e.NAZWISKO = 'Nowak' \text{ I } e.PESEL = w.PESELPRAC)))\}$

**LUB**

$((\exists m)(\exists d)(PRACOWNIK(m) \text{ I } DZIAŁ(d) \text{ I } p.NR\_DZ = d.NUMERDZ \text{ I } d.PESELKIEROWNIKA = m.PESEL \text{ I } m.NAZWISKO = 'Nowak' ))))\}$

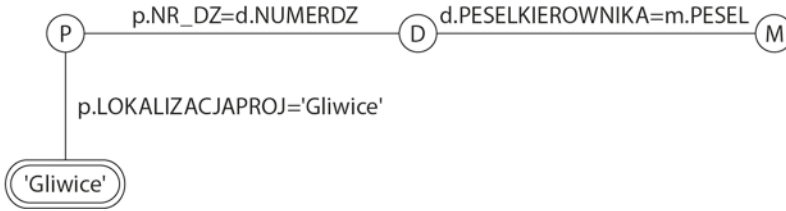
Porównaj powyższe wyrażenia z zaprezentowaną w podrozdziale 8.5 wersją tego samego zapytania zbudowaną w oparciu o operacje algebry relacyjnej. Stosowaną w algebrze relacyjnej operację sumy w większości przypadków można zastąpić operatorem logicznym LUB w rachunku relacji.

### 8.6.5. Notacja używana dla grafów zapytań

W tym punkcie opiszemy notację zaproponowaną do przedstawiania zapytań rachunku relacji w formie graficznej i bez skomplikowanych kwantyfikatorów. Uwzględniane tu zapytania to **zapytania selekcji-projekcji-złączenia**, ponieważ obejmują tylko te trzy operacje algebry relacyjnej. Tę notację można rozszerzyć o ogólniejsze zapytania, jednak nie omawiamy tu tego zagadnienia. Opisywana reprezentacja graficzna jest nazywana **grafem zapytań**. Na rysunku 8.13 pokazany jest graf zapytania Z2. Relacje z zapytania są reprezentowane przez **węzły relacji** przedstawiane jako pojedyncze okręgi. Wartości stałe, zwykle z warunków selekcji, mają postać **węzłów stałych** (podwójne okręgi lub owale). Warunki selekcji i złączenia są reprezentowane za pomocą **krawędzi** grafu (linii łączących węzły). Atrybuty pobierane z relacji są wymienione w nawiasie kwadratowym nad każdą relacją.

[p.NUMERPROJ,p.NR\_DZ]

[m.NAZWISKO,m.ADRES,m.DATAUR]



RYSUNEK 8.13. Graf zapytania dla Z2

W grafie zapytania nie jest określona kolejność wykonywania operacji. Dlatego jest to bardziej neutralna reprezentacja zapytania selekcja-projekcja-złączenie (w porównaniu z drzewem zapytań, gdzie kolejność przetwarzania jest pośrednio określona; patrz punkt 8.3.5). Każdemu zapytaniu odpowiada tylko jeden graf. Choć niektóre techniki optymalizowania zapytań były oparte na grafach, obecnie powszechnie przyjmuje się, że lepszym rozwiązaniem są drzewa zapytań, ponieważ w praktyce optymalizator zapytań musi wyświetlić kolejność operacji na potrzeby wykonywania zapytania, a graf na to nie pozwala.

W kolejnym punkcie omówimy związki pomiędzy kwantyfikatorami uniwersalnymi i egzystencjalnymi, a także przedstawimy sposób, w jaki można je wzajemnie przekształcać.

### 8.6.6. Wzajemne przekształcanie kwantyfikatorów uniwersalnych i egzystencjalnych

Wprowadzimy teraz kilka dobrze znanych z logiki matematycznej przekształceń związanych z kwantyfikatorami uniwersalnymi (ogólnymi) i egzystencjalnymi (szczegółowymi). Istnieje możliwość przekształcania kwantyfikatorów uniwersalnych w kwantyfikatory egzystencjalne i odwrotnie — oba przekształcenia prowadzą do uzyskania wyrażeń równoważnych. Procedurę pojedynczego przekształcenia można nieformalnie opisać w następujący sposób: przekształć jeden typ kwantyfikatora w drugi z symbolem negacji (poprzedź go operatorem **NIE**); zastąp operator **I** operatorem **LUB** i odwrotnie; zanegowany wzór przekształć we wzór niezanegowany; niezanegowany wzór przekształć we wzór zanegowany. Poniżej przedstawiono niektóre specjalne przypadki tego typu przekształceń (symbol  $\equiv$  oznacza **równoważność**):

$$(\forall x)(P(x)) \equiv \text{NIE}(\exists x)(\text{NIE}(P(x)))$$

$$(\exists x)(P(x)) \equiv \text{NIE}(\forall x)(\text{NIE}(P(x)))$$

$$(\forall x)(P(x) \text{ I } Q(x)) \equiv \text{NIE}(\exists x)(\text{NIE}(P(x)) \text{ LUB } \text{NIE}(Q(x)))$$

$$(\forall x)(P(x) \text{ LUB } Q(x)) \equiv \text{NIE}(\exists x)(\text{NIE}(P(x)) \text{ I } \text{NIE}(Q(x)))$$

$$(\exists x)(P(x) \text{ LUB } Q(x)) \equiv \text{NIE}(\forall x)(\text{NIE}(P(x)) \text{ I } \text{NIE}(Q(x)))$$

$$(\exists x)(P(x) \text{ I } Q(x)) \equiv \text{NIE}(\forall x)(\text{NIE}(P(x)) \text{ LUB } \text{NIE}(Q(x)))$$

Warto także pamiętać, że poniższe wyrażenia są PRAWDZIWE (symbol  $\Rightarrow$  oznacza **implikację**):

$$(\forall x)(P(x)) \Rightarrow (\exists x)(P(x))$$

$$\text{NIE}(\exists x)(P(x)) \Rightarrow \text{NIE}(\forall x)(P(x))$$

### 8.6.7. Stosowanie kwantyfikatorów uniwersalnych w zapytaniach

Za każdym razem, gdy decydujemy się na użycie kwantyfikatora uniwersalnego, rozsądnym rozwiązaniem jest postępowanie zgodnie z kilkoma istotnymi regułami, które dadzą nam pewność, że konstruowane wyrażenie jest sensowne. Omówimy te zasady na przykładzie zapytania 3.

**Zapytanie 3.** Znajdź nazwiska pracowników przydzielonych do *wszystkich* projektów nadzorowanych przez dział oznaczony numerem 5. Jednym ze sposobów zdefiniowania tego zapytania jest zastosowanie kwantyfikatora uniwersalnego (patrz poniżej).

**Z3:**  $\{p.\text{NAZWISKO}, p.\text{IMIĘ} \mid \text{PRACOWNIK}(p) \text{ I } ((\forall x)(\text{NIE}(\text{PROJEKT}(x)) \text{ LUB NIE}$

$(x.\text{NR\_DZ} = 5) \text{ LUB } ((\exists w)(\text{PRACUJE\_NAD}(w) \text{ I } w.\text{PESELPRAC} = p.\text{PESEL}$

$\text{I } x.\text{NUMERPROJ} = w.\text{NR\_PROJ}))\}$

Możemy oczywiście rozłożyć zapytanie Z3 na jego podstawowe składniki:

**Z3:**  $\{p.\text{NAZWISKO}, p.\text{IMIĘ} \mid \text{PRACOWNIK}(p) \text{ I } F'\}$

$F' = ((\forall x)(\text{NIE}(\text{PROJEKT}(x)) \text{ LUB } F_1))$

$F_1 = \text{NIE}(x.\text{NR\_DZ} = 5) \text{ LUB } F_2$

$F_2 = ((\exists w)(\text{PRACUJE\_NAD}(w) \text{ I } w.\text{PESELPRAC} = p.\text{PESEL}$

$\text{I } x.\text{NUMERPROJ} = w.\text{NR\_PRJ}))$

Chcemy się upewnić, że wybrany pracownik  $p$  pracuje *nad wszystkimi projektami* nadzorowanymi przez dział oznaczony numerem 5., jednak definicja uniwersalnego kwantyfikatora mówi, że aby cały kwantyfikowany wzór był PRAWDZIWY, *wzór wewnętrzny* musi być PRAWDZIWY *dla wszystkich krotek we wszechświecie*. Rozwiązanie polega w tym przypadku na wyłączeniu z obszaru uniwersalnej kwantyfikacji *wszystkich krotek*, którymi nie jesteśmy zainteresowani, przez zapewnienie PRAWDZIWOŚCI danego warunku *dla wszystkich tych krotek*. Taki zabieg jest konieczny, ponieważ PRAWDZIWOŚĆ kwantyfikowanego wzoru wymaga, aby zmienna krotki objęta uniwersalnym kwantyfikatorem była PRAWDZIWA *dla każdej krotki*, która może zostać jej przypisana.

Pierwszymi krotkami przeznaczonymi do wyłączenia (przez ich automatyczne wartościowanie jako PRAWDZIWE) są te krotki, które nie należą do interesującej nas relacji  $R$ . W zapytaniu Z3 zastosowanie wyrażenia  $\text{NIE}(\text{PROJEKT}(x))$  wewnątrz wzoru objętego uniwersalną kwantyfikacją powoduje, że otrzymujemy wartość PRAWDA dla wszystkich krotek  $x$ , które nie należą do relacji  $\text{PROJEKT}$ . W dalszej kolejności wyłączamy krotki nie będące przedmiotem naszego zainteresowania, które należą do samej relacji  $R$ . Użycie

w zapytaniu Z3 wyrażenia  $\text{NIE}(x.\text{NR\_DZ} = 5)$  powoduje, że otrzymujemy wartość PRAWDA dla wszystkich krotek  $x$ , które należą do relacji PROJEKT, ale nie są nadzorowane przez dział oznaczony numerem 5. Na końcu definiujemy warunek  $F_2$ , który musi być spełniony dla wszystkich pozostałych krotek w relacji  $R$ . Oznacza to, że całe zapytanie Z3 możemy wyjaśnić w następujący sposób:

- (1) Aby wzór  $F' = (\forall x)(F)$  był PRAWDZIWY, musimy zapewnić PRAWDZIWOŚĆ wzoru  $F$  dla wszystkich krotek we wszechświecie, które mogą być przypisane do zmiennej  $x$ . W zapytaniu Z3 interesuje nas jednak wyłącznie PRAWDZIWOŚĆ wzoru  $F$  dla wszystkich krotek reprezentujących (w relacji PROJEKT) projekty, które są nadzorowane przez dział 5. Oznacza to, że wzór  $F$  powinien mieć postać  $(\text{NIE}(\text{PROJEKT}(x) \text{ LUB } F_1))$ . Sam warunek  $(\text{NIE}(\text{PROJEKT}(x) \text{ LUB } \dots))$  jest PRAWDZIWY dla wszystkich krotek *nienależących* do relacji PROJEKT, zatem pełni rolę elementu eliminującego te krotki z dalszych rozważań odnośnie do prawdziwości wzoru  $F_1$ . Dla każdej krotki w relacji PROJEKT wzór  $F_1$  musi być PRAWDZIWY, jeśli PRAWDZIWY jest wzór  $F'$ .
- (2) Zgodnie z tym samym tokiem rozumowania nie chcemy rozważać tych krotek (projektów) należących do relacji PROJEKT, które nie są nadzorowane przez dział oznaczony numerem 5., ponieważ interesują nas wyłącznie krotki tej relacji z wartością  $\text{NR\_DZ} = 5$ . Możemy więc zapisać następujące wyrażenie:

**JĘŚLI**  $(x.\text{NR\_DZ} = 5)$ , **TO**  $F_2$

które jest równoważne wyrażeniu:

**(NIE** $(x.\text{NR\_DZ} = 5)$  **LUB**  $F_2)$

- (3) Wzór  $F_1$  ma w tej sytuacji postać  $\text{NIE}(x.\text{NR\_DZ} = 5) \text{ LUB } F_2$ . W kontekście zapytania Z3 oznacza to, że dla krotki  $x$  w relacji PROJEKT albo musi być spełniony warunek  $\text{NR\_DZ} \neq 5$ , albo PRAWDZIWY musi być wzór  $F_2$ .
- (4) I wreszcie, wzór  $F_2$  zawiera warunek, który zgodnie z naszymi zamierzeniami musi być spełniony dla wybieranych krotek relacji PRACOWNIK — z tego warunku wynika, że dany pracownik musi pracować *nad każdym projektem* (krotką relacji PROJEKT), który nie został jeszcze wyłączony z rozważań dotyczących prawdziwości. Właśnie takie krotki relacji PRACOWNIK są wybierane przez zapytanie.

Wyrażając zapytanie Z3 w języku naturalnym, możemy użyć następującego warunku selekcji krotki  $p$  z relacji PRACOWNIK: dla każdej krotki  $x$  w relacji PROJEKT z wartością  $x.\text{NR\_DZ} = 5$  musi istnieć taka krotka  $w$  w relacji PRACUJE\_NAD, że  $w.\text{PESELPRAC} = p.\text{PESEL}$  oraz  $w.\text{NR\_PROJ} = x.\text{NUMERPROJ}$ . Tak zbudowane zdanie jest równoważne wymaganiu sformułowanemu w następującej postaci: PRACOWNIK  $p$  pracuje nad każdym projektem (relacja PROJEKT)  $x$  w dziale (relacja DZIAŁ) oznaczonym numerem 5 (u!).

Stosując zaprezentowane w punkcie 8.6.6 ogólne reguły przekształcania kwantyfikatorów uniwersalnych w kwantyfikatory egzystencjalne, możemy raz jeszcze wyrazić zapytanie Z3 w postaci przedstawionego poniżej zapytania Z3A:

**Z3A:**  $\{p.\text{NAZWISKO}, p.\text{IMIĘ} \mid \text{PRACOWNIK}(p) \text{ I } (\text{NIE}(\exists x)(\text{PROJEKT}(x) \text{ I }$

$(x.\text{NR\_DZ} = 5 \text{ I } (\text{NIE}(\exists w)(\text{PRACUJE\_NAD}(w) \text{ I } w.\text{PESELPRAC} = p.\text{PESEL}$

$\text{I } x.\text{NUMERPROJ} = w.\text{NR\_PROJ}))))\}$

Przedstawimy teraz kilka dodatkowych przykładów zapytań wykorzystujących kwantyfikatory.

**Zapytanie 6.** Znajdź imiona i nazwiska pracowników, którzy nie mają na utrzymaniu członków rodziny.

$$\mathbf{Z6: \{p.IMIE, p.NAZWISKO \mid PRACOWNIK(p) \wedge (\neg(\exists d)(CZŁONEK\_RODZINY(d) \wedge p.PESEL = d.PESELPRAC))\}}$$

Stosując ogólne reguły przekształceń możemy wyrazić zapytanie Z6 w nieco innej postaci:

$$\mathbf{Z6A: \{p.IMIE, p.NAZWISKO \mid PRACOWNIK(p) \wedge ((\forall d)(\neg(CZŁONEK\_RODZINY(d) \wedge p.PESEL = d.PESELPRAC)))\}}$$

**Zapytanie 7.** Wymień imiona i nazwiska kierowników, którzy mają na utrzymaniu przynajmniej po jednym członku rodziny.

$$\mathbf{Z7: \{e.NAZWISKO, e.IMIE \mid PRACOWNIK(e) \wedge ((\exists d)(\exists p)(DZIAŁ(d) \wedge CZŁONEK\_RODZINY(p) \wedge e.PESELPRAC = d.PESELKIEROWNIKA \wedge p.PESELPRAC = e.PESEL))\}}$$

Konstrukcja tego zapytania oznacza, że wyrażenie *kierowników, którzy mają na utrzymaniu przynajmniej po jednym członku rodziny* jest interpretowane tak: *kierowników, dla których w bazie danych istnieją informacje o jakimś członku rodziny*.

### 8.6.8. Bezpieczne wyrażenia

Za każdym razem, gdy stosujemy kwantyfikatory uniwersalne, kwantyfikatory egzystencjalne lub negację predykatów w wyrażeniach rachunku relacji, musimy się upewnić, że otrzymane wyrażenie wynikowe będzie miało sens. **Bezpieczne wyrażenie** rachunku relacji to takie, które gwarantuje nam uzyskanie po jego obliczeniu *skończonej liczby krotek*; w przeciwnym przypadku wyrażenie uważa się za **niebezpieczne**. Przykładowo, wyrażenie:

$$\{t \mid \neg(\exists x)(PRACOWNIK(x) \wedge t = x)\}$$

jest *niebezpieczne*, ponieważ reprezentuje wszystkie krotki we wszechświecie, które *nie* są krotkami relacji PRACOWNIK, a takich krotek jest nieskończenie wiele. Jeśli stosując uniwersalne kwantyfikatory będziemy postępowali zgodnie z regułami zdefiniowanymi podczas omawiania zapytania Z3, zawsze będziemy otrzymywali bezpieczne wyrażenia. Możemy stworzyć bardziej precyzyjną definicję bezpiecznych wyrażeń przez wprowadzenie pojęcia *dziedziny wyrażenia relacyjnego rachunku krotek* — jest to zbiór wszystkich wartości, które występują albo w danym wyrażeniu postaci wartości stałych, albo w którejkolwiek z krotek relacji, do których to wyrażenie się odwołuje. Dziedziną wyrażenia  $\{t \mid \neg(\exists x)(PRACOWNIK(x) \wedge t = x)\}$  jest zbiór wszystkich wartości atrybutów występujących w pewnej krotce relacji PRACOWNIK (w którymkolwiek z jej atrybutów). Zgodnie z przedstawioną przed chwilą definicją, dziedzina wyrażenia Z3A obejmowałaby wszystkie wartości występujące w relacjach PRACOWNIK, PROJEKT oraz PRACUJE\_NAD (zbiór należałoby rozszerzyć o występującą w samym zapytaniu wartość 5).

O wyrażeniu rachunku relacji mówimy, że jest **bezpieczne**, jeśli wszystkie wartości w jego wyniku pochodzą z dziedziny tego wyrażenia. Łatwo zauważyć, że wynik wyrażenia  $\{t \mid \neg(\exists x)(PRACOWNIK(x) \wedge t = x)\}$  jest *niebezpieczny*, ponieważ będzie zawierał wszystkie krotki (a więc także wszystkie wartości) spoza relacji PRACOWNIK, a takie wartości oczywiście nie należą do dziedziny tego wyrażenia. Wszystkie pozostałe przykłady wyrażeń zaprezentowane w tym podrozdziale są w tym znaczeniu wyrażeniami bezpiecznymi.

## 8.7. Relacyjny rachunek dziedzin

Istnieje jeszcze jeden typ rachunku relacyjnego, który często nazywa się relacyjnym rachunkiem dziedzin lub po prostu **rachunkiem dziedzin**. Niemal w tym samym czasie, w którym dział IBM Research w San Jose, w Kalifornii, opracowywał standard SQL (języka opartego na relacyjnym rachunku krotek; patrz rozdziały 6. i 7.), w dziale IBM T. J. Watson Research Center w Yorktown Heights w stanie Nowy Jork projektowano język nazwany QBE (od ang. *Query-By-Example*, czyli „zapytanie przez przykład”). Formalną specyfikację rachunku dziedzin zaproponowano bezpośrednio po opracowaniu systemu QBE.

Rachunek dziedzin różni się od rachunku krotek *typem zmiennych* wykorzystywanych w konstruowanych wzorach. Zamiast stosowania zmiennych obejmujących krotki, zmienne używane w wyrażeniach rachunku dziedzin reprezentują pojedyncze wartości należące do dziedzin atrybutów. Aby zbudować relację stopnia  $n$  dla wyniku zapytania, musimy mieć  $n$  takich **zmiennych dziedzin** — po jednej dla każdego atrybutu. Wyrażenie rachunku dziedzin ma postać:

$$\{x_1, x_2, \dots, x_n \mid \text{WARUNEK}(x_1, x_2, \dots, x_n, x_{n+1}, x_{n+2}, \dots, x_{n+m})\}$$

gdzie  $x_1, x_2, \dots, x_n, x_{n+1}, x_{n+2}, \dots, x_{n+m}$  są zmiennymi dziedzin obejmującymi dziedziny poszczególnych atrybutów, natomiast WARUNEK jest **warunkiem** lub **wzorem** relacyjnego rachunku dziedzin.

Wzór składa się z **atomów**. Atomy we wzorach relacyjnego rachunku dziedzin nieznacznie się różnią od atomów stosowanych w wyrażeniach rachunku krotek. Każdy z atomów może mieć jedną z następujących postaci:

- (1) Atom w postaci  $R(x_1, x_2, \dots, x_j)$ , gdzie  $R$  jest nazwą relacji stopnia  $j$ , natomiast każde  $x_i$  (gdzie  $1 \leq i \leq j$ ) jest zmienną dziedziny. Atom w takiej formie określa, że lista wartości zmiennych  $\langle x_1, x_2, \dots, x_j \rangle$  musi być krotką w relacji, której nazwą jest  $R$ , gdzie  $x_i$  jest wartością  $i$ -tego atrybutu tej krotki. Aby uczynić wyrażenie relacyjnego rachunku dziedzin bardziej zwięzłym, możemy *pominąć przecinki* na liście zmiennych atomu, zatem zaprezentowaną wcześniej postać wyrażenia możemy zapisać w sposób następujący:

$$\{x_1, x_2, \dots, x_n \mid R(x_1 x_2 \dots x_3) \mathbf{I} \dots\}$$

zamiast

$$\{x_1, x_2, \dots, x_n \mid R(x_1, x_2, \dots, x_3) \mathbf{I} \dots\}$$

- (2) Atom w postaci  $x_i \mathbf{op} x_j$ , gdzie **op** jest jednym z operatorów porównania należących do zbioru  $\{=, <, \leq, >, \geq, \neq\}$ , natomiast  $x_i$  oraz  $x_j$  są zmiennymi dziedzin.
- (3) Atom w postaci  $x_i \mathbf{op} c$  lub  $c \mathbf{op} x_j$ , gdzie **op** jest jednym z operatorów porównania należących do zbioru  $\{=, <, \leq, >, \geq, \neq\}$ ,  $x_i$  oraz  $x_j$  są zmiennymi dziedzin, natomiast  $c$  jest wartością stałą.

Podobnie jak w relacyjnym rachunku krotek atomy mogą mieć dla określonego zbioru wartości albo wartość PRAWDZA, albo wartość FAŁSZ — są to tzw. **wartości prawdziwości** atomów. W przypadku pierwszej postaci atomu, jeśli zmienne dziedzin mają przypisywane wartości odpowiadające krotce określonej relacji  $R$ , wówczas taki atom jest PRAWDZIWY. W przypadku drugiej i trzeciej postaci atomu, atom jest PRAWDZIWY, jeśli zmienne dziedzin mają przypisywane wartości, które spełniają zdefiniowany warunek.



Podobnie jak w relacyjnym rachunku krotek wzory składają się z atomów, zmiennych i kwantyfikatorów, zatem nie będziemy w tym miejscu powtarzali formalnej specyfikacji wzorów. Poniżej prezentujemy kilka przykładowych zapytań zdefiniowanych właśnie w rachunku dziedzin. Do oznaczania zmiennych dziedzin będziemy wykorzystywali małe litery:  $l, m, n, \dots, x, y, z$ .

**Zapytanie 0.** Wyszukaj datę urodzenia i adres pracownika (lub pracowników), który nazywa się 'Jan B. Nowak'.

**Z0:**  $\{uv \mid (\exists q) (\exists r) (\exists s) (\exists t) (\exists w) (\exists x) (\exists y) (\exists z)$

$(\text{PRACOWNIK}(qrstuvwxyz) \wedge q = \text{'Jan'} \wedge r = \text{'B'} \wedge s = \text{'Nowak'})\}$

Potrzebujemy aż dziesięciu zmiennych dla relacji PRACOWNIK, po jednej dla dziedziny każdego z kolejnych atrybutów tej relacji. Ze wspomnianej dziesiątki zmiennych ( $q, r, s, \dots, z$ ) tylko para zmiennych  $u$  i  $v$  jest wolna (ponieważ znajduje się po lewej stronie znaku  $\mid$  i nie podlega kwantyfikacji). Najpierw określamy **żądane atrybuty** (w tym przypadku DATAUR i ADRES), które będą reprezentowane za pomocą pary wolnych zmiennych dziedzin, odpowiednio  $u$  dla atrybutu DATAUR i  $v$  dla atrybutu ADRES. Następnie (za znakiem  $\mid$ ) określamy warunek selekcji krotek — w tym przypadku warunek ten ma postać wymogu odnośnie do sekwencji wartości przypisanych do zmiennych  $qrstuvwxyz$ , która ma stanowić krotkę relacji PRACOWNIK, i w której zmienne  $q$  (IMIE),  $r$  (INICJAŁDRIMIENTA) oraz  $s$  (NAZWISKO) mają reprezentować odpowiednio wartości 'Jan', 'B' i 'Nowak'. Dla wygody w kolejnych prezentowanych przykładach będziemy umieszczali w zasięgu kwantyfikatorów tylko te zmienne, które *rzeczywiście występują w danym warunku* (w przypadku zapytania Z0 byłoby to zmienne  $q, r$  i  $s$ )<sup>13</sup>.

W zapytaniu Z0A przedstawiono alternatywną skróconą notację, którą wykorzystuje się w systemie QBE do zapisywania zapytań — w tym przypadku stosujemy bezpośredni zapis stałych 'Jan', 'B' i 'Nowak'. Wszystkie zmienne są umieszczane po lewej stronie symbolu  $\mid$  i znajdują się w obszarze niejawnego kwantyfikatora egzystencjalnego<sup>14</sup>.

**Z0A:**  $\{uv \mid \text{PRACOWNIK}(\text{'Jan'}, \text{'B'}, \text{'Nowak'}, t, u, v, w, x, y, z)\}$

**Zapytanie 1.** Wyszukaj imiona i nazwiska oraz adresy wszystkich pracowników zatrudnionych w Dziale Badawczym.

**Z1:**  $\{qsv \mid (\exists z) (\exists l) (\exists m) (\text{PRACOWNIK}(qrstuvwxyz) \wedge$

$\text{DZIAŁ}(lmno) \wedge l = \text{'Badania'} \wedge m = z)\}$

Warunek wiążący dwie zmienne dziedzin, które obejmują atrybuty pochodzące z dwóch różnych relacji (w przypadku zapytania Z1 chodzi o warunek  $m = z$ ), jest **warunkiem złączenia**; natomiast warunek wiążący zmienną dziedzin z wartością stałą (w przypadku zapytania Z1 chodzi o warunek  $l = \text{'Badania'}$ ) jest **warunkiem selekcji**.

<sup>13</sup> Notacja przewidująca oznaczanie kwantyfikatorami tylko tych zmiennych dziedzin, które faktycznie są wykorzystywane w warunkach, i zapisywania takich predykatów jak  $\text{PRACOWNIK}(qrstuvwxyz)$  bez oddzielania zmiennych dziedzin za pomocą przecinków jest oczywiście skróconą wersją stosowaną dla wygody i prostoty; z formalnego punktu widzenia taka notacja nie jest prawidłowa.

<sup>14</sup> Także ta notacja nie jest poprawna z formalnego punktu widzenia.

**Zapytanie 2.** Dla każdego projektu realizowanego w Gliwicach znajdź numer projektu, numer działu nadzorującego oraz nazwisko, adres i datę urodzenia kierownika zarządzającego danym działem.

$$\begin{aligned} \mathbf{Z2:} \{ & iksuv \mid (\exists j) (\exists m) (\exists n) (\exists t) (\text{PROJEKT}(hijk) \mathbf{I} \\ & \text{PRACOWNIK}(qrstuvwxyz) \mathbf{I} \text{ DZIAŁ}(lmno) \mathbf{I} k = m \mathbf{I} \\ & n = t \mathbf{I} j = \text{'Gliwice'} ) \} \end{aligned}$$

**Zapytanie 6.** Znajdź nazwiska pracowników, którzy nie mają na utrzymaniu żadnych członków rodziny.

$$\begin{aligned} \mathbf{Z6:} \{ & qs \mid (\exists t) (\text{PRACOWNIK}(qrstuvwxyz) \mathbf{I} \\ & (\mathbf{NIE}(\exists l)(\text{CZŁONEK\_RODZINY}(lmnop) \mathbf{I} t = l))) \} \end{aligned}$$

Zapytanie 6. można także wyrazić w oparciu o kwantyfikatory uniwersalne, zamiast z wykorzystaniem kwantyfikatorów egzystencjalnych — patrz poniższe zapytanie Z6A:

$$\begin{aligned} \mathbf{Z6A:} \{ & q, s \mid (\exists t) (\text{PRACOWNIK}(qrstuvwxyz) \mathbf{I} \\ & ((\forall l)(\mathbf{NIE}(\text{CZŁONEK\_RODZINY}(lmnop)) \mathbf{LUB} \mathbf{NIE}(t = l)))) \} \end{aligned}$$

**Zapytanie 7.** Wymień nazwiska kierowników, którzy mają na utrzymaniu przynajmniej po jednym członku rodziny.

$$\begin{aligned} \mathbf{Z7:} \{ & s, q \mid (\exists t) (\exists j) (\exists l) (\text{PRACOWNIK}(qrstuvwxyz) \mathbf{I} \text{ DZIAŁ}(hijk) \\ & \mathbf{I} \text{ CZŁONEK\_RODZINY}(lmnop) \mathbf{I} t = j \mathbf{I} l = t) \} \end{aligned}$$

Jak już wspominaliśmy, można wykazać, że każde zapytanie, które może być wyrażone w postaci wyrażenia algebry relacyjnej, może być także wyrażone w relacyjnym rachunku dziedzin lub w relacyjnym rachunku krotek. Okazuje się także, że każde *bezpieczne wyrażenie* relacyjnego rachunku dziedzin lub relacyjnego rachunku krotek można wyrazić w algebrze relacyjnej.

Język *QBE* (ang. *Query-By-Example*) opiera się na relacyjnym rachunku dziedzin — ta własność systemu QBE została jednak odkryta dopiero w momencie, w którym zaprezentowano formalny opis tego rachunku. QBE był jednym z pierwszych graficznych języków zapytań z minimalną liczbą reguł składniowych opracowanych dla systemów baz danych. Język ten został zaprojektowany w dziale IBM Research i jest obecnie udostępniany w postaci komercyjnych produktów firmy IBM — stanowi część opcjonalnego interfejsu *QMF* (ang. *Query Management Facility*) dla systemu DB2. Język QBE znalazł wielu naśladowców w innych produktach komercyjnych. Z uwagi na miejsce tego rozwiązania wśród języków relacyjnych, wprowadziliśmy przegląd jego możliwości w dodatku C.

## 8.8. Podsumowanie

W tym rozdziale zaprezentowaliśmy dwa języki formalne dla relacyjnego modelu danych. Języki te są wykorzystywane do manipulowania relacjami i generowania nowych relacji stanowiących wyniki wykonywanych zapytań. Omówiliśmy algebrę relacyjną wraz z jej operacjami, które stosuje się do definiowania sekwencji działań tworzących zapytania. Następnie

wprowadziliśmy dwa typy relacyjnych rachunków, nazywanych rachunkiem krotek i rachunkiem dziedzin.

W podrozdziałach od 8.1 do 8.3 wprowadziliśmy podstawowe operacje algebry relacyjnej i przedstawiliśmy typy zapytań, w których prezentowane operacje mają zastosowanie. W pierwszej kolejności omówiliśmy takie unarne operatory relacyjne jak selekcja i projekcja, a także operację zmiany nazwy. Następnie przystąpiliśmy do omawiania operacji znanych z teorii zbiorów, które wymagały od relacji występujących w roli operandów spełnienia warunku zgodności złączenia — przedstawiliśmy operacje sumy, części wspólnej oraz różnicy zbiorów. Operacja iloczynu kartezjańskiego może być wykorzystywana do złączenia krotek pochodzących z dwóch różnych relacji i uzyskiwania w jego wyniku wszystkich możliwych kombinacji. Operacja ta rzadko jednak jest stosowana w praktyce — użyliśmy jej do pokazania, jak w oparciu o połączenie iloczynu kartezjańskiego z operacją selekcji można zdefiniować zbiór zawierający wyłącznie pasujące do siebie krotki, co prowadzi do operacji złączenia. W dalszej części tego rozdziału opisaliśmy różne wersje operacji złączenia nazywane złączeniem theta, równo-złączeniem i złączeniem naturalnym. Przedstawiliśmy też drzewa zapytań. Są one graficzną reprezentacją zapytań algebry relacyjnej i można je stosować także do tworzenia wewnętrznych struktur danych używanych do reprezentowania zapytań w SZBD.

Omówiliśmy następnie kilka ważnych rodzajów zapytań, których *nie można* wyrazić za pomocą podstawowych operacji algebry relacyjnej, ale które są jednocześnie bardzo istotne w praktycznych zastosowaniach relacyjnego modelu danych. Wprowadziliśmy uogólnioną projekcję pozwalającą używać na liście projekcji funkcji zależnych od atrybutów. Opisaliśmy również funkcje agregacji (nazywane też funkcjami agregującymi), które stosuje się do obsługi żądań statystycznych wymagających agregowania danych w celu podsumowywania informacji z tabel. Omówiliśmy także zapytania rekurencyjne, których nie można bezpośrednio definiować za pomocą operacji algebry relacyjnej, ale które można wykonywać krok po kroku w przedstawiony tu sposób. W dalszej części tego rozdziału przedstawiliśmy operacje złączenia zewnętrznego i sumy zewnętrznej, które stanowią przydatne rozszerzenie podstawowych operacji złączenia oraz sumy i pozwalają zachować w wynikach wszystkie informacje z relacji źródłowych.

W ostatnich dwóch podrozdziałach opisaliśmy podstawowe pojęcia związane z rachunkiem relacji, który opiera się na jednym z działów logiki matematycznej nazywanym rachunkiem predykatów. Istnieją dwa typy rachunków relacji: (1) relacyjny rachunek krotek, który wykorzystuje zmienne krotek reprezentujące krotki (wiersze) relacji, oraz (2) relacyjny rachunek dziedzin, który wykorzystuje zmienne dziedzin obejmujące dziedziny (kolumny) relacji. W rachunku relacji zapytania są definiowane w postaci pojedynczych, deklaratywnych wyrażeń i nie mogą określać żadnej kolejności ani żadnych konkretnych metod dochodzenia do oczekiwanych wyników. Oznacza to, że rachunek relacji jest często uznawany za *deklaratywny* język wyższego poziomu niż np. algebra relacyjna, ponieważ wyrażenie rachunku relacji stwierdza jedynie, *co* chcemy uzyskać, bez określania, *w jaki sposób* dane zapytanie ma być wykonywane.

Przedstawiliśmy grafy zapytań będące wewnętrzną reprezentacją zapytań w rachunku relacji. Zaprezentowaliśmy także znaczenie kwantyfikatora egzystencjalnego ( $\exists$ ) oraz kwantyfikatora uniwersalnego ( $\forall$ ). Opisaliśmy także problem określania bezpiecznych zapytań, które gwarantują spełnienie warunku skończoności wyników. Omówiliśmy także

reguły przekształcania kwantyfikatorów uniwersalnych w kwantyfikatory egzystencjalne i odwrotnie. To właśnie kwantyfikatory zapewniają rachunkowi relacji taką moc wyrażania, która czyni z niej język równoważny algebrze relacyjnej. Podstawowy rachunek relacji nie zawiera mechanizmów analogicznych do grupowania i agregowania danych — odpowiednie elementy są dostępne w rozszerzeniach tego rachunku.

## Pytania powtórkowe

- 8.1. Wymień operacje należące do algebry relacyjnej i krótko opisz ich przeznaczenie.
- 8.2. Czym jest zgodność złączenia? Dlaczego operacje sumy, części wspólnej i różnicy wymagają od relacji występujących w roli operandów właśnie zgodności złączenia?
- 8.3. Omów kilka rodzajów zapytań, w których zmiana nazw atrybutów jest niezbędnym krokiem w stronę zapewnienia jednoznaczności ich interpretacji.
- 8.4. Omów różne typy operacji *złączenia wewnętrznego*. W jakich sytuacjach niezbędne jest stosowanie złączenia theta?
- 8.5. Jaką rolę odgrywa pojęcie *klucza obcego* w procesie definiowania najbardziej popularnych rodzajów operacji złączenia?
- 8.6. Czym jest operacja funkcji agregującej? Do czego jest wykorzystywana?
- 8.7. Co odróżnia operację złączenia zewnętrznego od operacji złączenia wewnętrznego? Czym różni się operacja sumy zewnętrznej od operacji sumy?
- 8.8. Na jakich polach rachunek relacji różni się od algebry relacyjnej, a na jakich oba te języki są do siebie podobne?
- 8.9. Co odróżnia relacyjny rachunek krotek od relacyjnego rachunku dziedzin?
- 8.10. Omów znaczenie kwantyfikatora egzystencjalnego (oznaczanego symbolem  $\exists$ ) i kwantyfikatora uniwersalnego (oznaczanego symbolem  $\forall$ ).
- 8.11. Zdefiniuj następujące pojęcia w kontekście relacyjnego rachunku krotek: *zmienna krotki*, *relacja zakresowa*, *atom*, *wzór* i *wyrażenie*.
- 8.12. Zdefiniuj następujące pojęcia w kontekście relacyjnego rachunku dziedzin: *zmienna dziedziny*, *relacja zakresowa*, *atom*, *wzór* i *wyrażenie*.
- 8.13. Jak definiujemy *bezpieczne wyrażenie* w rachunku relacji?
- 8.14. Kiedy język zapytań nazywamy językiem relacyjnie kompletnym?

## Ćwiczenia

- 8.15. Przedstaw wynik dla każdego z przykładowych zapytań zaprezentowanych w podrozdziale 8.5 przy założeniu, że zapytania te zostaną wykonane na stanie bazy danych z rysunku 5.6.
- 8.16. Zdefiniuj następujące zapytania dla schematu bazy danych FIRMA zaprezentowanego na rysunku 5.5, wykorzystując omówione w tym rozdziale operacje relacyjne. Przedstaw także wyniki każdego z tych zapytań w przypadku jego zastosowania dla stanu bazy danych z rysunku 5.6.

- a) Znajdź nazwiska wszystkich pracowników działu 5., którzy na pracę nad projektem „ProjektX” poświęcają więcej niż 10 godzin tygodniowo.
  - b) Wymień nazwiska wszystkich pracowników, którzy mają na utrzymaniu członka lub członków rodziny noszących takie samo imię jak ci pracownicy.
  - c) Znajdź nazwiska wszystkich pracowników, którzy są bezpośrednimi podwładnymi Franciszka Wieszczyckiego.
  - d) Dla każdego projektu wymień nazwę i łączną liczbę godzin poświęcanych tygodniowo na jego realizację (przez wszystkich pracowników firmy zaangażowanych w pracę nad projektem).
  - e) Znajdź nazwiska wszystkich pracowników, którzy realizują jakiekolwiek projekty.
  - f) Znajdź nazwiska wszystkich pracowników, którzy nie realizują żadnych projektów.
  - g) Dla każdego działu znajdź jego nazwę oraz średnią wysokość pensji wypłacanych wszystkim pracownikom zatrudnionym w danym dziale.
  - h) Oblicz średnią pensję wszystkich pracowników firmy płci żeńskiej.
  - i) Znajdź nazwiska i adresy wszystkich pracowników, którzy realizują co najmniej po jednym projekcie zlokalizowanym w Warszawie, ale których działy zatrudniające mają swoje siedziby poza Warszawą.
  - j) Wymień nazwiska wszystkich kierowników działów, którzy nie mają na utrzymaniu żadnych członków rodziny.
- 8.17. Przeanalizuj przedstawiony na rysunku 5.8 przykładowy schemat relacyjnej bazy danych LINIE LOTNICZE, który opisano w ćwiczeniu 5.12. Zdefiniuj następujące zapytania, wykorzystując operacje algebry relacyjnej.
- a) Dla każdego lotu wymień jego numer, lotnisko wylotu dla pierwszego przelotu składającego się na dany lot oraz lotnisko przylotu dla ostatniego przelotu składającego się na ten lot.
  - b) Wymień numery i dni tygodnia dla wszystkich lotów lub przelotów, które rozpoczynają się w Międzykontynentalnym Porcie Lotniczym w Houston (kod lotniska 'IAH') i kończą w Międzykontynentalnym Porcie Lotniczym w Los Angeles (kod lotniska 'LAX').
  - c) Wymień numery lotów, kody lotnisk wylotów, planowane czasy wylotów, kody lotnisk przylotów, planowane czasy przylotów oraz dni tygodnia dla wszystkich lotów i przelotów, które rozpoczynają się na którymś z lotnisk w Houston i kończą się na którymś z lotnisk w Los Angeles.
  - d) Wymień wszystkie informacje o opłatach związanych z lotem oznaczonym numerem 'C0197'.
  - e) Znajdź listę wolnych miejsc w locie oznaczonym numerem 'C0197' i zaplanowanym na dzień '2009-10-09'.
- 8.18. Przeanalizuj zaprezentowany na rysunku 8.14 schemat relacyjnej bazy danych BIBLIOTEKA, która jest wykorzystywana do przechowywania informacji o książkach, czytelnikach i wypożyczeniach. Na rysunku 8.14 więzy integralności odwołań są reprezentowane za pomocą odpowiednich strzałek (zgodnie z notacją użytą na rysunku 5.7). Zapisz wyrażenia algebry relacyjnej dla następujących zapytań:

- Ile kopii książki pt. *Raj utracony* znajduje się w posiadaniu filii biblioteki, którą nazwano 'Poznań'?
- Ile kopii książki pt. *Raj utracony* znajduje się w posiadaniu wszystkich filii biblioteki?
- Znajdź nazwiska wszystkich czytelników, którzy aktualnie nie mają żadnych wypożyczonych książek.
- Dla każdej książki, która została wypożyczona z filii nazwanej 'Poznań', i której termin zwrotu (TerminZwrotu) wskazuje na dany dzień, znajdź tytuł, a także nazwisko i adres czytelnika, który aktualnie tę książkę przetrzymuje.
- Dla każdej filii biblioteki znajdź nazwę i łączną liczbę aktualnie wypożyczonych książek.
- Znajdź nazwiska, adresy i liczby wypożyczonych książek dla wszystkich czytelników, którzy aktualnie dysponują więcej niż pięcioma wypożyczonymi książkami.
- Dla każdej książki autorstwa (lub współautorstwa) 'Stephen Kinga' znajdź tytuł i liczbę kopii będących w posiadaniu filii nazwanej 'Centrala'.

**KSIĄŻKA**

IdentyfikatorKsiążki	Tytuł	NazwaWydawcy
----------------------	-------	--------------

**AUTORZY\_KSIĄŻEK**

IdentyfikatorKsiążki	NazwiskoAutora
----------------------	----------------

**WYDAWCA**

Nazwa	Adres	Telefon
-------	-------	---------

**KOPIE\_KSIĄŻEK**

IdentyfikatorKsiążki	IdentyfikatorFilii	LiczbaKopii
----------------------	--------------------	-------------

**WYPOŻYCZENIA\_KSIĄŻEK**

IdentyfikatorKsiążki	IdentyfikatorFilii	NumerKarty	DataWypożyczenia	TerminZwrotu
----------------------	--------------------	------------	------------------	--------------

**FILIA\_BIBLIOTEKI**

IdentyfikatorFilii	NazwaFilii	Adres
--------------------	------------	-------

**CZYTELNIK**

NumerKarty	Nazwisko	Adres	Telefon
------------	----------	-------	---------

RYSUNEK 8.14. Schemat relacyjnej bazy danych BIBLIOTEKA

- 8.19. Wykorzystując wyrażenia algebry relacyjnej, zdefiniuj następujące zapytania dla schematu relacyjnej bazy danych przedstawionego w ćwiczeniu 5.14:
- Wymień numery (atrybut `NrZamówienia`) i daty dostaw (atrybut `DataDostawy`) dla wszystkich zamówień z magazynu oznaczonego numerem 'W2'.
  - Wymień informacje o magazynach, które realizowały zamówienia złożone przez klienta nazwiskiem 'Jan Lipski'. Stwórz listę złożoną z pozycji w formacie `NrZamówienia`, `NrMagazynu`.
  - Opracuj listę nazwisk klientów, liczby zamówień oraz średnich wartości zamówień (a więc zawierającą kolumny `NazwiskoKlienta`, `LiczbaZamówień` i `ŚredniaWartośćZamówienia`), w której środkowa kolumna zawiera łączną liczbę zamówień zrealizowanych dla danego klienta, natomiast ostatnia kolumna zawiera ich średnią wartość.
  - Wymień zamówienia, które nie zostały zrealizowane w ciągu 30 dni od złożenia.
  - Wymień numery zamówień (atrybut `NrZamówienia`), których realizacja wymagała zaangażowania *wszystkich* magazynów utrzymywanych przez daną firmę w Warszawie.
- 8.20. Wykorzystując wyrażenia algebry relacyjnej, zdefiniuj następujące zapytania dla schematu relacyjnej bazy danych przedstawionego w ćwiczeniu 5.15:
- Podaj szczegóły (wszystkie atrybuty relacji `WYJAZD`) wszystkich wyjazdów, których koszt przekroczył 500 złotych.
  - Wymień numery PESEL przedstawicieli handlowych, którzy dotarli do miejscowości 'Leśmierz'.
  - Wymień łączne koszty wyjazdów wygenerowane przez przedstawiciela handlowego z wartością `PESEL = '76110532156'`.
- 8.21. Wykorzystując wyrażenia algebry relacyjnej, zdefiniuj następujące zapytania dla schematu relacyjnej bazy danych przedstawionego w ćwiczeniu 5.16:
- Wymień numery przedmiotów, na które w semestrze zimowym roku 2009 (a więc z wartością `Semestr = 'Z09'`) uczęszczali wszyscy studenci nazwiskiem 'Jan Nowak'.
  - Stwórz listę książek i opracowań (obejmującą atrybuty `NrPrzedmiotu`, `ISBNKsiążki` i `TytułKsiążki`) dla tych przedmiotów prowadzonych na wydziale 'INF', które wymagają więcej niż jednego podręcznika.
  - Wymień wszystkie wydziały, które wykorzystują *wyłącznie* książki wydane przez wydawnictwo 'Helion S.A.'.
- 8.22. Przeanalizuj dwie przedstawione na rysunku 8.15 tabele T1 i T2. Przedstaw wyniki następujących operacji na tych tabelach:
- $T1 \bowtie_{T1.P = T2.A} T2$
  - $T1 \bowtie_{T1.Q = T2.B} T2$
  - $T1 \Join_{T1.P = T2.A} T2$
  - $T1 \bowtie_{T1.Q = T2.B} T2$
  - $T1 \cup T2$
  - $T1 \bowtie_{(T1.P = T2.A \vee T1.R = T2.C)} T2$



TABELA T1

P	Q	R
10	a	5
15	b	8
25	a	6

TABELA T2

A	B	C
10	b	6
25	c	3
10	b	5

RYSUNEK 8.15. Stan bazy danych dla relacji T1 i T2

- 8.23. Wykorzystując wyrażenia algebry relacyjnej, zdefiniuj następujące zapytania dla schematu relacyjnej bazy danych przedstawionego w ćwiczeniu 5.17:
- Dla sprzedawcy nazwiskiem 'Janina Daniszewska' wymień następujące informacje dotyczące wszystkich sprzedanych przez nią samochodów: NrSeryjny, Producent i WartośćTransakcji.
  - Wymień numery seryjne i modele (atrybuty NrSeryjny i Model) samochodów, w których nie jest instalowane żadne wyposażenie dodatkowe.
  - Przeanalizuj możliwość zastosowania operacji złączenia naturalnego pomiędzy relacjami SPRZEDAWCA i SPRZEDAJE. Jakie byłoby znaczenie lewego złączenia zewnętrznego tych tabel (nie zmieniaj podanej przed chwilą kolejności relacji). Wyjaśnij swoje wnioski w oparciu o odpowiednie przykłady.
  - Zapisz w algebrze relacyjnej jakieś zapytanie wykorzystujące operację selekcji i jedną z operacji na zbiorach, po czym wyjaśnij własnymi słowami, co dane zapytanie oznacza.
- 8.24. Zdefiniuj zapytania a, b, c, e, f, i oraz j z ćwiczenia 8.16 zarówno w relacyjnym rachunku krotek, jak i w relacyjnym rachunku dziedzin.
- 8.25. Zdefiniuj zapytania a, b, c oraz d z ćwiczenia 8.17 zarówno w relacyjnym rachunku krotek, jak i w relacyjnym rachunku dziedzin.
- 8.26. Zdefiniuj zapytania c, d, f oraz g z ćwiczenia 8.18 zarówno w relacyjnym rachunku krotek, jak i w relacyjnym rachunku dziedzin.
- 8.27. Przyjmijmy, że mamy zapytanie wykorzystujące rachunek relacyjny i zawierające  $n$  zmiennych krotek. Jaka będzie typowa minimalna liczba warunków złączenia? Dlaczego? Jaki będzie skutek zastosowania mniejszej liczby warunków złączenia?
- 8.28. Napisz ponownie zapytania rachunku relacji, które przedstawiono w podrozdziale 8.7 po zapytaniu Z0; tym razem użyj skróconej notacji analogicznej do tej, którą zastosowano w zapytaniu Z0A (celem tej notacji jest zminimalizowanie liczby zmiennych dziedzin przez zastępowanie zmiennych stałymi we wszystkich miejscach, w których jest to możliwe).
- 8.29. Przeanalizuj następujące zapytanie: znajdź numery PESEL wszystkich pracowników, którzy są zaangażowani w realizację przynajmniej tych projektów, które realizuje także pracownik z numerem PESEL równym 65010912345. Takie zapytanie może mieć postać: (DLA KAŻDEGO  $x$ )(JEŚLI  $P$ , TO  $Q$ ), gdzie:
- $x$  jest zmienną krotki, która obejmuje relację PROJEKT;
  - $P$  reprezentuje fakt realizacji projektu  $x$  przez pracownika z numerem PESEL równym 65010912345;
  - $Q$  reprezentuje fakt realizacji projektu  $x$  przez pracownika  $e$ .

Wyraż to samo zapytanie w relacyjnym rachunku krotek — stosuj się do poniższych reguł:

- $(\forall x)(P(x)) \equiv \text{NIE}(\exists x)(\text{NIE}(P(x)))$
- $(\text{JEŚLI } P, \text{ TO } Q) \equiv (\text{NIE}(P) \text{ LUB } Q)$

8.30. Przedstaw sposób, w jaki można zdefiniować wymienione poniżej operacje algebry relacyjnej zarówno w relacyjnym rachunku krotek, jak i relacyjnym rachunku dziedzin.

- a)  $\sigma_A = c(R(A, B, C))$
- b)  $\pi_{\langle A, B \rangle}(R(A, B, C))$
- c)  $R(A, B, C) * S(D, E, F)$
- d)  $R(A, B, C) \cup S(A, B, C)$
- e)  $R(A, B, C) \cap S(A, B, C)$
- f)  $R(A, B, C) = S(A, B, C)$
- g)  $R(A, B, C) \times S(D, E, F)$
- h)  $R(A, B) \div S(A)$

8.31. Zasugeruj takie rozszerzenia dla rachunku relacji, które umożliwią wyrażanie w tym rachunku następujących rodzajów operacji (szczegółowo omówionych w podrozdziale 8.4): (a) funkcji agregujących i grupowania; (b) operacji złączenia zewnętrznego; (c) rekurencyjnych zapytań domknięcia.

8.32. Zapytanie zagnieżdżone to takie, które znajduje się w innym zapytaniu. Jest to umieszczone w nawiasie zapytanie, którego wynik można zastosować w różnych miejscach jako wartość (zamiast relacji). Zapisz opisane niżej zapytania dotyczące bazy z rysunku 5.5. Wykorzystaj zapytania zagnieżdżone i omówione w tym rozdziale operatory relacyjne. Przedstaw też wynik każdego zapytania dla stanu bazy danych z rysunku 5.6.

- a) Podaj nazwiska wszystkich pracowników z działu, który zatrudnia osobę o najwyższym wynagrodzeniu w całej firmie.
- b) Podaj nazwiska wszystkich pracowników mających przełożonego, którego szef ma PESEL 37111045873.
- c) Podaj nazwiska pracowników zarabiających przynajmniej 1000 złotych więcej niż pracownik zarabiający w firmie najmniej.

8.33. Określ, czy następujące wnioski są prawdziwe, czy fałszywe:

- a)  $\text{NIE}(P(x) \text{ LUB } Q(x)) \rightarrow (\text{NIE}(P(x)) \text{ I } (\text{NIE}(Q(x))))$ .
- b)  $\text{NIE}(\exists x)(P(x)) \rightarrow \forall x(\text{NIE}(P(x)))$ .
- c)  $(\exists x)(P(x)) \rightarrow \forall x((P(x)))$ .

## Ćwiczenia laboratoryjne

8.34. Zdefiniuj i wykonaj następujące zapytania za pomocą algebry relacyjnej, stosując interpreter tej algebry do schematu bazy danych FIRMA zaprezentowanego na rysunku 5.5.

- a) Znajdź nazwiska wszystkich pracowników działu piątego, którzy na pracę nad projektem „ProjektX” poświęcają więcej niż 10 godzin tygodniowo.
- b) Wymień nazwiska wszystkich pracowników, którzy mają na utrzymaniu członka lub członków rodziny noszących takie samo imię jak ci pracownicy.
- c) Znajdź nazwiska wszystkich pracowników, którzy są bezpośrednimi podwładnymi Franciszka Wieszczyckiego.
- d) Znajdź nazwiska pracowników, którzy pracują nad wszystkimi projektami.
- e) Znajdź nazwiska pracowników, którzy nie realizują żadnych projektów.
- f) Znajdź nazwiska i adresy wszystkich pracowników, którzy realizują co najmniej po jednym projekcie zlokalizowanym w Warszawie, ale których działy zatrudniające mają swoje siedziby poza Warszawą.
- g) Wymień nazwiska wszystkich kierowników działów, którzy nie mają na utrzymaniu żadnych członków rodziny.

8.35. Przyjrzyj się schematowi relacyjnemu ZAMÓWIENIAPOCZTOWE, opisującemu dane firmy realizującej zamówienia drogą pocztową.

CZĘŚCI(Nrcz, Nazwacz, Dstszt, Cena, Poziom)  
 KLIENCI(Nrk1, Nazwiskok1, Ulica, Kod, Telefon)  
 PRACOWNICY(Nrprac, Nazwiskoprac, Kod, Datazatr)  
 KODY\_POCZTOWE(Kod, Miasto)  
 ZAMÓWIENIA(Nrzam, Nrk1, Nrprac, Złożone, Wysłane)  
 SZCZEGÓŁYZAM(Nrzam, Nrcz, Sztuk)

Dstszt oznacza *liczbę dostępnych sztuk*. Nazwy pozostały atrybutów powinny być zrozumiałe. Zdefiniuj i wykonaj następujące zapytania, używając interpretera algebry relacyjnej do schematu bazy ZAMÓWIENIAPOCZTOWE.

- a) Pobierz nazwy części kosztujących mniej niż 20 złotych.
- b) Pobierz nazwiska i miasta zamieszkania pracowników, którzy przyjęli zamówienia na części kosztujące ponad 50 złotych.
- c) Pobierz pary numerów klientów mających ten sam kod pocztowy.
- d) Pobierz nazwiska klientów, którzy zamówili części od pracowników mieszkających we Wrocławiu.
- e) Pobierz nazwiska klientów, którzy zamówili części kosztujące mniej niż 20 złotych.
- f) Pobierz nazwiska klientów, którzy nie złożyli zamówienia.
- g) Pobierz nazwiska klientów, którzy złożyli dwa zamówienia.

8.36. Przyjrzyj się relacyjnemu schematowi DZIENNIKOCEN, opisującemu dane z dziennika ocen wykładowcy. (*Uwaga: atrybuty Pięć, Cztery i Trzy z relacji PRZEDMIOTY przechowują liczbę punktów potrzebną do uzyskania określonych ocen*).

KATALOG(Nrkat, Nazwakat)  
 STUDENCI(Nrind, Imię, Nazwisko, Inicjałdrugiegoimienia)  
 PRZEDMIOTY(Semestr, Nrdziału, Nrkat, Pięć, Cztery, Trzy)  
 ZAPISY(Nrind, Semestr, Nrdziału)

Zdefiniuj i wykonaj następujące zapytania, używając interpretera algebry relacyjnej do schematu bazy `DZIENNIKOCEN`.

- a) Pobierz nazwiska studentów zapisanych na zajęcia z automatów w jesiennym semestrze 2009 r.
- b) Pobierz wartości `Nrind` studentów zapisanych na przedmioty `INF226` i `INF227`.
- c) Pobierz wartości `Nrind` studentów zapisanych na przedmioty `INF226` lub `INF227`.
- d) Pobierz nazwiska studentów, którzy nie zapisali się na żadne zajęcia.
- e) Pobierz nazwiska studentów, którzy zapisali się na wszystkie przedmioty z tabeli `KATALOG`.

8.37. Przyjrzyj się bazie danych obejmującej następujące relacje:

`DOSTAWCA(Nrdost, Nazwadost)`

`CZĘŚĆ(Nrcz, Nazwacz)`

`PROJEKT(Nrproj, Nazwaproj)`

`DOSTAWA(Nrdost, Nrcz, Nrproj)`

Ta baza danych przechowuje informacje o dostawcach, częściach i projektach oraz obejmuje trójskładnikową relację między dostawcami, częściami i projektami. Jest to relacja typu wiele do wielu do wielu. Zdefiniuj i wykonaj następujące zapytania, posługując się interpreterem algebry relacyjnej:

- a) Pobierz numery części dostarczanych do dokładnie dwóch projektów.
- b) Pobierz nazwy dostawców, którzy dostarczają więcej niż dwie części do projektu 'J1'.
- c) Pobierz numery części dostarczanych przez każdego dostawcę.
- d) Pobierz nazwy projektów, do których części są dostarczane tylko przez dostawcę 'S1'.
- e) Pobierz nazwy dostawców, którzy dostarczają przynajmniej po dwie różne części do przynajmniej dwóch różnych projektów.

8.38. Zdefiniuj i wykonaj następujące zapytania dotyczące bazy z ćwiczenia 5.16, używając interpretera algebry relacyjnej.

- a) Pobierz nazwiska studentów zapisanych na kurs, na którym używany jest podręcznik opublikowany przez wydawnictwo Addison-Wesley-Longman.
- b) Pobierz nazwy kursów, na których przynajmniej raz zmieniono podręcznik.
- c) Pobierz nazwy wydziałów, na których używane są wyłącznie książki opublikowane przez wydawnictwo Addison-Wesley.
- d) Pobierz nazwy wydziałów, na których używane są podręczniki napisane przez Navathego i opublikowane przez wydawnictwo Addison-Wesley.
- e) Pobierz nazwiska studentów, którzy nigdy nie brali udziału w kursie, gdzie używano podręcznika napisanego przez Navathego i opublikowanego przez wydawnictwo Addison-Wesley.

8.39. Ponownie wykonaj ćwiczenia laboratoryjne od 8.34 do 8.38 dla relacyjnego rachunku dziedzin, używając interpretera tego rachunku.

## Wybrane publikacje

Podstawową algebrę relacyjną zdefiniował Codd (1970). Date (1983) omówił złączenia zewnętrzne. Carlis (1986) i Ozsoyoglu wraz ze współpracownikami (1985) omówili kierunki rozwoju operacji relacyjnych. Cammarata i in. (1989) rozszerzył więzy integralności i złączenia modelu relacyjnego.

Codd (1971) wprowadził język Alpha, który opiera się na elementach relacyjnego rachunku krotek. W języku Alpha zastosowano także mechanizm funkcji agregującej, który wykracza poza zakres tradycyjnego rachunku relacji. Codd (1972) zaproponował nie tylko oryginalną, formalną definicję tego rachunku, ale także przedstawił uniwersalny algorytm przekształcania wyrażeń relacyjnego rachunku krotek w wyrażenia algebry relacyjnej. Na relacyjnym rachunku krotek opiera się przedstawiony przez Stonebrakera i in. (1976) język QUEL, w którym pośrednio występują kwantyfikatory egzystencjalne, ale który nie umożliwia wykorzystywania w wyrażeniach kwantyfikatorów uniwersalnych. QUEL został zaimplementowany jako komercyjny język w systemie INGRES. Codd zdefiniował pojęcie relacyjnej kompletności języka zapytań, które oznacza, że jego moc wyrażania jest nie mniejsza od mocy wyrażania rachunku relacji. Ullman (1988) przedstawił formalny dowód równoważności algebry relacyjnej z bezpiecznymi wyrażeniami relacyjnego rachunku krotek i relacyjnego rachunku dziedzin. Abiteboul i in. (1995) oraz Atzeni i deAntonellis (1993) zaproponowali szczegółową interpretację formalnych języków relacyjnych.

Chociaż idea stosowania relacyjnego rachunku dziedzin została początkowo zaproponowana w postaci języka QBE (Zloof, 1975), formalny opis tego rachunku przedstawili Lacroix i Pirotte (1977a). Zloof (1975) opisał eksperymentalną wersję języka zapytań-przez-przykłady (QBE). Lacroix i Pirotte (1977b) opisali język ILL, który bazował na założeniach relacyjnego rachunku dziedzin. Whang i in. (1990) rozszerzył język QBE o możliwość stosowania kwantyfikatorów uniwersalnych. Przez lata proponowano wiele rozwiązań zmierzających do stworzenia wizualnych języków zapytań (QBE jest jednym z przykładów takiego języka); wiele ciekawych propozycji tego typu prezentowano na rozmaitych konferencjach, jak choćby Visual Database Systems Workshop, gdzie odpowiednie opracowania przedstawili Arisawa i Catarci (2000) oraz Zhou i Pu (2002).

# Projektowanie relacyjnych baz danych przez odwzorowywanie modelu ER i EER w model relacyjny

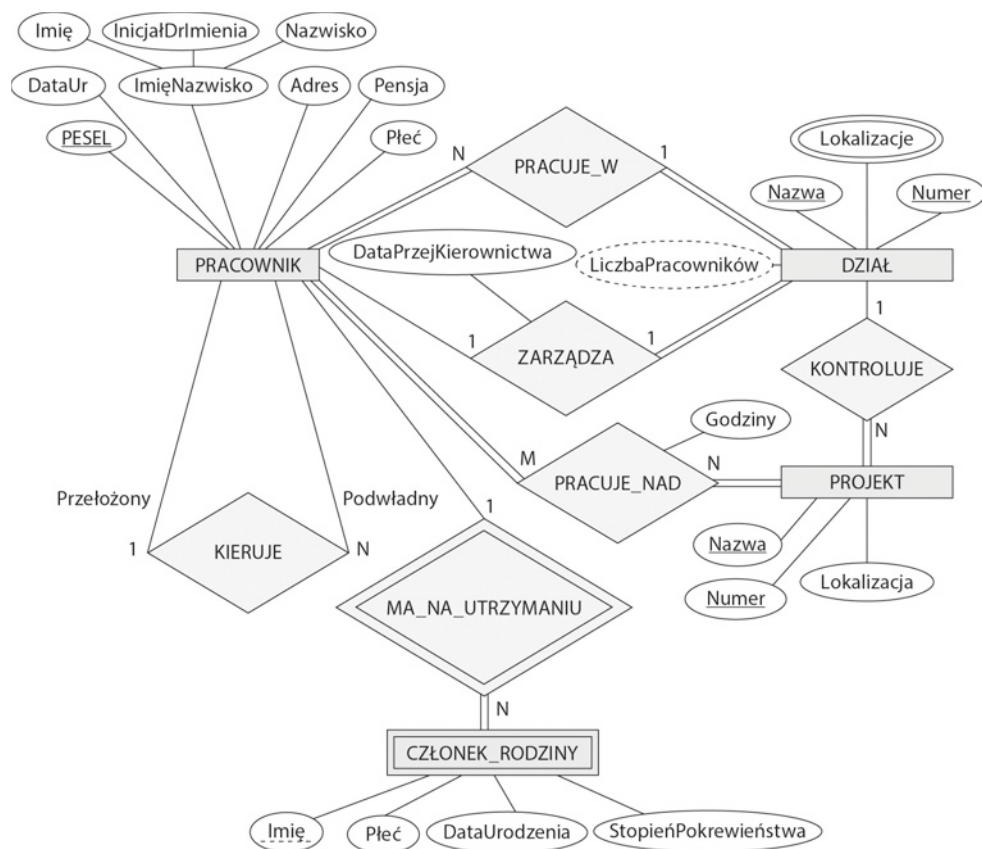
W tym rozdziale skoncentrujemy się na metodach *projektowania schematu relacyjnej bazy danych* w oparciu o koncepcyjny projekt schematu. Na rysunku 3.1 pokazano ogólne ujęcie tego procesu projektowania baz danych. W tym rozdziale koncentrujemy się na etapie tworzenia **logicznego projektu bazy danych** (ten etap nazywany jest też **odwzorowywaniem modelu danych**). Zaprezentujemy procedury tworzenia schematu relacyjnego na bazie istniejącego schematu związków encji (ER) lub rozszerzonego schematu ER (EER). Nasze rozważania będą dotyczyły nie tylko konstrukcji modeli ER i EER, które zaprezentowano w rozdziałach 3. i 4., ale także przedstawionych w rozdziałach od 5. do 8. konstrukcji modelu relacyjnego. Wiele narzędzi typu *CASE* (od ang. *Computer-Aided Software Engineering*, czyli wspomagana komputerowo inżynieria oprogramowania) opiera się na modelach ER, EER lub innych podobnych modelach (zgodnie z naszymi rozważaniami z rozdziałów 3. i 4.). Wiele z tych narzędzi wykorzystuje diagramy ER i EER lub ich odmiany do graficznego opracowywania schematów baz danych oraz do automatycznego konwertowania tych schematów w schematy relacyjnych baz danych (wyrażone w językach DDL konkretnych relacyjnych systemów zarządzania bazami danych) za pomocą algorytmów podobnych do tych, które prezentujemy w tym rozdziale.

W podrozdziale 9.1 przedstawimy siedmioetapowy algorytm konwertujący konstrukcje podstawowego modelu ER — typy encji (silne i słabe), związki binarne (z różnymi ograniczeniami strukturalnymi), związki  $n$ -składnikowe oraz atrybuty (proste, złożone i wielowartościowe) — w relacje. Następnie, w podrozdziale 9.2, dalej będziemy analizowali możliwości tego algorytmu, przedstawimy sposoby odwzorowywania w relacje takich konstrukcji modelu EER jak specjalizacja-generalizacja oraz typy unii (kategorie). Podrozdział 9.3 zawiera podsumowanie tego rozdziału.

## 9.1. Projektowanie relacyjnych baz danych w oparciu o odwzorowywanie modelu ER w model relacyjny

### 9.1.1. Algorytm odwzorowujący model ER w model relacyjny

Opiszemy teraz kolejne kroki algorytmu odwzorowującego schemat modelu ER w relacyjny schemat bazy danych. Do zilustrowania procedury odwzorowania wykorzystamy wielokrotnie omawianą przykładową bazę danych FIRMA. Na rysunku 9.1 raz jeszcze przedstawiono schemat ER tej bazy danych, natomiast odpowiedni schemat relacyjnej bazy danych FIRMA (który ilustruje wykonywane kroki odwzorowania) zaprezentowano na rysunku 9.2. Zakładamy, że odwzorowanie doprowadzi do utworzenia tabel z prostymi atrybutami jednowartościowymi. W wyniku procesu odwzorowywania zdefiniowane zostaną także ograniczenia modelu relacyjnego określone w rozdziale 5., obejmujące klucze główne, klucze unikatowe (jeśli takie występują) i więzy integralności odwołań.



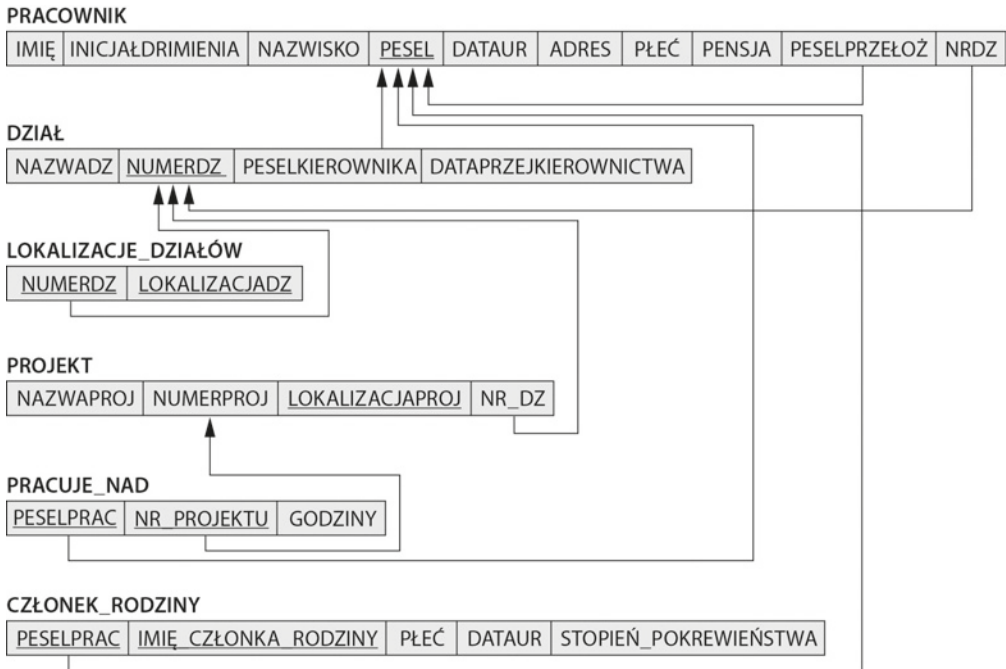
RYСУNEK 9.1. Diagram schematu koncepcyjnego dla bazy danych FIRMA



**Krok 1: Odwzorowanie zwykłych typów encji.** Dla każdego zwykłego (silnego) typu encji  $E$  w schemacie związków encji (ER) należy stworzyć relację  $R$ , która będzie zawierała wszystkie proste atrybuty typu encji  $E$ . W nowej relacji możemy umieścić tylko proste atrybuty składowe atrybutów złożonych. Musimy następnie wybrać jeden z atrybutów klucza typu encji  $E$  do roli klucza głównego relacji  $R$ . Jeśli wybrany klucz typu encji  $E$  jest atrybutem złożonym, zbiór atrybutów prostych składających się na ten atrybut będzie razem tworzył klucz główny nowej relacji  $R$ .

Jeśli podczas opracowywania projektu koncepcyjnego w typie encji  $E$  zidentyfikowano więcej niż jeden klucz, powinniśmy zachować informacje opisujące atrybuty tworzące każdy z dodatkowych kluczy, aby określić dodatkowe (unikatowe) klucze nowej relacji  $R$ . Wiedza na temat tych kluczy jest przydatna także podczas indeksowania danych i innych działań analitycznych.

W naszym przykładzie tworzymy relacje PRACOWNIK, DZIAŁ i PROJEKT (patrz rysunek 9.2), które odpowiadają zwykłym typom encji PRACOWNIK, DZIAŁ i PROJEKT z rysunku 9.1. Klucz obcy i atrybuty relacji (jeśli w ogóle istnieją) nie są na tym etapie uwzględniane; dodamy je w kolejnych krokach algorytmu. Do pominiętych w pierwszym kroku atrybutów należą: PESELPRZEŁOŻ i NR\_DZ z relacji PRACOWNIK, PESELKIEROWNIKA i DATAPRZEJKIEROWNICTWA z relacji DZIAŁ oraz NR\_DZ z relacji PROJEKT. W prezentowanym przykładzie wybieramy atrybuty PESEL, NUMERDZ i NUMERPROJ do roli kluczy głównych relacji PRACOWNIK, DZIAŁ i PROJEKT. Wiedza na temat występujących w roli kluczy wtórnych atrybutów NAZWADZ (dla relacji DZIAŁ) oraz NAZWAPROJ (dla relacji PROJEKT) jest zachowywana z myślą o późniejszym wykorzystaniu w trakcie projektowania.



RYSUNEK 9.2. Efekt odwzorowania schematu ER FIRMA w schemat relacyjnej bazy danych

Relacje utworzone w fazie odwzorowywania typów encji są niekiedy nazywane **relacjami encji**, ponieważ każda krotka (każdy wiersz) reprezentuje pojedynczy egzemplarz encji. Wynik tego etapu odwzorowywania jest pokazany na rysunku 9.3(a).

(a) PRACOWNIK

IMIĘ	INICJAŁDRIMIENIA	NAZWISKO	PESEL	DATAUR	ADRES	PŁEĆ	PENSJA
------	------------------	----------	-------	--------	-------	------	--------

DZIAŁ

NAZWADZ	NUMERDZ
---------	---------

PROJEKT

NAZWAPROJ	NUMERPROJ	LOKALIZACJAPROJ
-----------	-----------	-----------------

(b) CZŁONEK\_RODZINY

PESELPRAC	IMIĘ_CZŁONKA_RODZINY	PŁEĆ	DATAUR	STOPIEŃ_POKREWIEŃSTWA
-----------	----------------------	------	--------	-----------------------

(c) PRACUJE\_NAD

PESELPRAC	NR_PROJ	GODZINY
-----------	---------	---------

(d) LOKALIZACJE\_DZIAŁÓW

NUMERDZ	LOKALIZACJADZ
---------	---------------

RYSUNEK 9.3. Przedstawione wybrane kroki odwzorowywania. (a) Relacje encji po kroku 1.

(b) Dodatkowa relacja słabej encji po kroku 2. (c) Relacje związku po kroku 5.

(d) Relacja reprezentująca atrybut wielowartościowy po kroku 6

**Krok 2: Odwzorowanie słabych typów encji.** Dla każdego słabego typu encji  $W$  zdefiniowanego w schemacie związków encji (ER) z właścicielskim typem encji  $E$  należy stworzyć relację  $R$  zawierającą wszystkie proste atrybuty (lub proste składniki atrybutów złożonych) typu encji  $W$ . Dodatkowo nowa relacja  $R$  powinna zawierać — w postaci atrybutu (atrybutów) klucza obcego — atrybut (atrybuty) klucza głównego relacji odpowiadającej właścicielskiemu typowi encji (odpowiadających właścicielskim typom encji); w ten sposób uwzględniamy typ związku identyfikującego typu encji  $W$ . Kluczem głównym nowej relacji  $R$  powinna być kombinacja klucza głównego (kluczy głównych) właściciela (właścicieli) oraz klucza częściowego słabego typu encji  $W$  (jeśli taki klucz częściowy istnieje). Jeśli istnieje słaby typ encji  $E_2$ , którego właścicielem jest inny słaby typ encji  $E_1$ , typ encji  $E_1$  powinien być odwzorowany w odpowiednią relację przed typem encji  $E_2$ , aby możliwe było właściwe określenie jego klucza głównego.

W naszym przykładzie stworzymy w tym kroku relację CZŁONEK\_RODZINY, która będzie odpowiadała słabemu typowi encji CZŁONEK\_RODZINY (patrz rysunek 9.3(b)). Dołączymy do nowej relacji klucz główny PESEL z relacji PRACOWNIK, która w tym przypadku odpowiada właścicielskiemu typowi encji — atrybut ten będzie miał postać klucza obcego relacji CZŁONEK\_RODZINY. Dodatkowo (choć nie jest to konieczne) zmienimy nazwę tego atrybutu na PESELPRAC. Kluczem głównym nowej relacji CZŁONEK\_RODZINY będzie kombinacja atrybutów {PESELPRAC, IMIĘ\_CZŁONKA\_RODZINY}, ponieważ atrybut IMIĘ\_CZŁONKA\_RODZINY (jego nazwę zmieniono względem nazwy Imię z rysunku 9.1) jest kluczem częściowym typu encji CZŁONEK\_RODZINY.

Często stosuje się opcję propagacji (przekazywania kaskadowego) dla działań wywołanych za pośrednictwem odwołań (patrz podrozdział 6.2) — dotyczy to kluczy obcych w relacjach odpowiadających słabym typom encji, ponieważ istnienie egzemplarzy słabych typów encji jest uzależnione od istnienia właściwych egzemplarzy właścicielskich typów encji. Propagowanie akcji może dotyczyć operacji aktualizacji i usuwania (odpowiednio *ON UPDATE* oraz *ON DELETE*).

**Krok 3: Odwzorowanie binarnych typów związków 1:1.** Dla każdego binarnego typu związku *R* ze współczynnikiem liczności równym 1:1 w schemacie związków encji (ER) musimy zidentyfikować relacje *S* i *T*, które będą odpowiadały typom encji występującym w danym związku (typie relacji). Istnieją trzy możliwe podejścia do tej fazy odwzorowywania: (1) podejście oparte na wykorzystaniu klucza obcego, (2) podejście oparte na scalaniu związków oraz (3) podejście oparte na odwołaniach skrośnych lub relacjach związków. Pierwsze podejście jest najbardziej przydatne i powinno być stosowane we wszystkich przypadkach, chyba że spełnione są opisane poniżej warunki:

- (1) **Podejście oparte na wykorzystaniu klucza obcego:** Wybieramy jedną z relacji (przyjmijmy, że jest to relacja *S*) i dołączamy do zbioru jej atrybutów klucz obcy wskazujący na klucz główny relacji *T*. Lepszym rozwiązaniem jest w takim przypadku wybranie do roli *S* typu encji z *pełnym udziałem* w typie relacji *R*. Zbiór atrybutów relacji *S* powinien wówczas obejmować wszystkie atrybuty proste (lub proste składniki atrybutów złożonych) typu relacji *R* ze współczynnikiem liczności równym 1:1.

W naszym przykładzie odwzorowujemy typ związku 1:1 *KIERUJE* z rysunku 9.1 przez wybór występującego w związku typu encji *DZIAŁ* do roli relacji *S*, ponieważ udział tego typu encji w typie związku *KIERUJE* jest pełny (każdy dział ma swojego kierownika). Dołączamy do nowej relacji *DZIAŁ* klucz główny relacji *PRACOWNIK*, który w relacji *DZIAŁ* będzie miał status klucza obcego wskazującego na relację *PRACOWNIK* (z nazwą zmienioną na *PESELKIEROWNIKA*). Relacja *DZIAŁ* będzie także zawierała pochodzący z typu relacji *KIERUJE* prosty atrybut *DATAPRZEJKIEROWNICTWA* (z nazwą zmienioną na *DATAPOCZKIEROWNIKA*).

Łatwo zauważyć, że istnieje możliwość umieszczenia klucza głównego relacji *S* w roli klucza obcego w relacji *T*. W naszym przypadku wiązałoby się to z koniecznością stworzenia atrybutu klucza obcego (powiedzmy *KIEROWANY\_DZIAŁ*) w relacji *PRACOWNIK*, jednak taki klucz musiałby mieć wartości puste dla krotek reprezentujących tych pracowników, którzy nie kierują żadnym działem. Przykładowo, gdyby tylko 2 procent pracowników pełniło funkcje kierowników działów firmy, aż 98 procent takich kluczy obcych zawierałoby wartości puste. Jeszcze innym rozwiązaniem jest umieszczenie kluczy obcych zarówno w relacji *S*, jak i w relacji *T* — takie nadmiarowe podejście wiąże się jednak z utrudnieniami w utrzymywaniu spójności.

- (2) **Podejście oparte na scalaniu relacji:** Alternatywną możliwością w kwestii odwzorowywania typu związku 1:1 jest scalanie dwóch typów encji oraz łączącego je typu związku w pojedynczą relację modelu relacyjnego. Takie rozwiązanie jest możliwe w sytuacji, gdy udziały *obu typów encji w danej relacji są pełne*, ponieważ obie tabele zawsze mają wtedy tę samą liczbę krotek.
- (3) **Podejście oparte na odwołaniach skrośnych lub relacjach związków:** Trzecią możliwością jest stworzenie dodatkowej relacji *R*, której celem będzie utrzymywanie skrośnych odwołań pomiędzy kluczami głównymi obu relacji *S* i *T* reprezentujących

powiązane typy encji. W dalszej części tego rozdziału przekonamy się, że jest to jedyne dopuszczalne rozwiązanie w przypadku binarnych związków ze współczynnikiem licznosci równym  $M:N$ . Relacja  $R$  jest w takim przypadku nazywana **relacją związku** (a niekiedy także **tabelą wyszukiwania**), ponieważ każda krotka w relacji  $R$  reprezentuje egzemplarz związku łączącego pojedynczą krotkę relacji  $S$  z pojedynczą krotką relacji  $T$ . Relacja  $R$  obejmuje wtedy atrybuty klucza głównego z relacji  $S$  i  $T$  jako klucze obce dotyczące tych relacji. Kluczem głównym relacji  $R$  jest jeden z tych dwóch kluczy obcych; drugi z tych kluczy jest wtedy kluczem unikatowym relacji  $R$ . Wadą tego rozwiązania jest tworzenie dodatkowej relacji i wymóg dodatkowych operacji złączenia w trakcie łączenia powiązanych krotek z wymienionych tabel.

**Krok 4: Odwzorowanie binarnych typów związków  $1:N$ .** Możliwe są dwa podejścia: (1) oparte na kluczu obcym i (2) wykorzystujące odwołania skróśne (lub relację związku). Zwykle preferowana jest pierwsza z tych metod, ponieważ zmniejsza liczbę tabel.

- (1) **Podejście oparte na kluczu obcym:** Dla każdego zwykłego, binarnego typu związku  $R$  ze współczynnikiem licznosci równym  $1:N$ , należy zidentyfikować relację  $S$ , która reprezentuje typ encji występujący w danym związku po stronie licznosci  $N$ . Musimy dołączyć w roli klucza obcego relacji  $S$  klucz główny relacji  $T$ , która reprezentuje drugi typ encji występującej w związku (typie relacji)  $R$  — konieczność stworzenia klucza obcego po tej stronie wynika z faktu, że każdy egzemplarz encji po stronie licznosci  $N$  jest związany z najwyżej jednym egzemplarzem encji po stronie licznosci 1 danego typu związku. Do nowej relacji  $S$  musimy dodać wszystkie atrybuty proste (lub proste składniki atrybutów złożonych) odwzorowywanego typu związku  $1:N$ .

W naszym przykładzie możemy w ten sposób odwzorować następujące typy związków z licznoscią  $1:N$ : PRACUJE\_W, NADZORUJE oraz KIEROWNICTWO (patrz rysunek 9.1). W przypadku typu relacji PRACUJE\_W musimy dołączyć klucz główny NUMERDZ relacji DZIAŁ w postaci klucza obcego relacji PRACOWNIK i nadać mu nazwę NRDZ. W przypadku typu relacji KIEROWNICTWO dołączamy klucz główny relacji PRACOWNIK w postaci klucza obcego w tej samej relacji PRACOWNIK (co wynika z faktu rekurencyjnego powiązania różnych egzemplarzy tego samego typu encji) i nazywamy nowy atrybut PESELPRZEŁOŻ. Typ relacji NADZORUJE odwzorowujemy za pomocą atrybutu klucza obcego NR\_DZ w relacji PROJEKT, który odwołuje się do klucza głównego NUMERDZ relacji DZIAŁ. Te klucze obce są widoczne na rysunku 9.2.

- (2) **Podejście oparte na relacji związku.** Alternatywnym rozwiązaniem jest stworzenie **relacji związku** (czyli zastosowanie techniki odwołań skróśnych); jest to trzecia możliwość obsługi binarnych związków  $1:1$ . Możemy więc dodać do schematu nową relację  $R$ , której atrybutami będą klucze relacji  $S$  i  $T$ , i której klucz główny będzie identyczny jak klucz główny relacji  $S$ . Zastosowanie tego rozwiązania jest uzasadnione w sytuacji, gdy w odwzorowywanym typie związku występuje stosunkowo niewiele krotek relacji  $S$ , ponieważ w ten sposób możemy uniknąć nadmiernej liczby wartości pustych w kluczu obcym.

**Krok 5: Odwzorowanie binarnych typów związków  $M:N$ .** W tradycyjnym modelu relacyjnym bez atrybutów wielowartościowych jedyną metodą odwzorowywania związków  $M:N$  jest **relacja związku** (technika **odwołań skróśnych**). Dla każdego binarnego typu związku  $R$  ze współczynnikiem licznosci równym  $M:N$  należy stworzyć reprezentującą go nową relację  $S$ . Do relacji  $S$  musimy dołączyć atrybuty kluczy obcych wskazują-

ce na klucze główne relacji reprezentujących typy encji występujące w odwzorowywanym typie relacji; *kombinacja* tych atrybutów utworzy klucz główny relacji *S*. Relacja *S* powinna także zawierać (w postaci własnych atrybutów) wszystkie atrybuty proste (lub proste składniki atrybutów złożonych) typu związku *M:N*. Warto pamiętać, że typ związku ze współczynnikami *M:N* nie może być reprezentowany przez pojedynczy atrybut klucza obcego w jednej z powiązanych relacji (jak w przypadku typów związków 1:1 i 1:N) — z uwagi na współczynnik licznosci musimy stworzyć osobną *relację związku S*.

W naszym przykładzie odwzorowujemy typ związku `PRACUJE_NAD` ze współczynnikami licznosci *M:N* (patrz rysunek 9.1) przez stworzenie nowej relacji `PRACUJE_NAD` (patrz rysunek 9.2). Do nowej relacji dołączamy (w postaci atrybutów kluczy obcych) klucze główne relacji `PROJEKT` i `PRACOWNIK` — zmieniamy ich nazwy odpowiednio na `NR_PROJ` i `PESELPRAC` (zmiana nazwy *nie jest konieczna*; to efekt wyboru projektowego). Nowa relacja `PRACUJE_NAD` zawiera także atrybut `GODZINY`, który reprezentuje atrybut `GODZINY` odwzorowanego typu związku. Kluczem głównym relacji `PRACUJE_NAD` jest kombinacja atrybutów kluczy obcych, a więc para `{PESELPRAC, NR_PROJ}`. Utworzona **relacja związku** jest przedstawiona na rysunku 9.3(c).

Dla kluczy obcych w nowej relacji odpowiadającej typowi związku *R* powinniśmy określić opcję propagowania (przekazywania kaskadowego) działań wywoływanych za pośrednictwem odwołań (patrz podrozdział 4.2), ponieważ istnienie każdego egzemplarza tego typu związku jest uzależnione od istnienia egzemplarzy wiązanych przez siebie typów relacji. Propagowanie akcji może więc dotyczyć zarówno operacji aktualizacji, jak i operacji usuwania.

Nietrudno zauważyć, że tę samą technikę, którą zastosowaliśmy podczas odwzorowywania typów związków ze współczynnikami licznosci równym *M:N* (a więc opartą na odwołaniach skrośnych lub relacjach związków), można stosować także dla typów związków ze współczynnikami licznosci równymi 1:1 oraz 1:N. Takie rozwiązanie jest jednak zalecane tylko w sytuacjach, w których istnieje stosunkowo niewiele egzemplarzy odwzorowywanego typu związku — w ten sposób możemy uniknąć wartości pustych w atrybutach kluczy obcych. W takim przypadku klucz główny relacji związku będzie *tylko jedną* z kombinacji kluczy obcych odwołujących się do wzajemnie powiązanych relacji encji. W przypadku związku ze współczynnikami licznosci równym 1:N kluczem głównym relacji związku będzie klucz obcy wskazujący na relację, która reprezentuje typ encji występujący po stronie *N*. W przypadku związku ze współczynnikami licznosci równym 1:1 rolę klucza głównego relacji związku można przypisać dowolnemu z pary kluczy obcych.

**Krok 6: Odwzorowanie atrybutów wielowartościowych.** Dla każdego atrybutu wielowartościowego *A* należy stworzyć nową relację *R*. Relacja *R* będzie zawierała atrybuty odpowiadające wielowartościowemu atrybutowi *A* oraz atrybut klucza głównego *K* (w relacji *R* występujący w roli klucza obcego) relacji reprezentującej odwzorowywany typ encji lub typ relacji, który zawiera atrybut *A*. Klucz główny relacji *R* jest w tej sytuacji kombinacją atrybutu *A* i klucza obcego *K*. Jeśli odwzorowywany atrybut wielowartościowy jest jednocześnie atrybutem złożonym, w relacji *R* powinniśmy umieścić tylko jego proste składniki.

W naszym przykładzie musimy stworzyć relację `LOKALIZACJE_DZIAŁÓW` (rysunek 9.3(d)). Atrybut `LOKALIZACJADZ` reprezentuje wielowartościowy atrybut `LOKALIZACJE` oryginalnego typu encji `DZIAŁ`, natomiast atrybut `NUMERDZ` (w roli klucza obcego naszej relacji) repre-

zentuje klucz główny relacji DZIAŁ. Klucz główny relacji LOKALIZACJE\_DZIAŁÓW jest kombinacją pary atrybutów {NUMERDZ, LOKALIZACJADZ}. W relacji LOKALIZACJE\_DZIAŁÓW będzie istniała osobna krotka dla każdej lokalizacji poszczególnych działów. Warto zauważyć, że w nowszych wersjach modelu relacyjnego, gdzie dopuszczalne są tablicowe typy danych, atrybut wielowartościowy można odwzorować na atrybut tablicowy zamiast na odrębną tabelę.

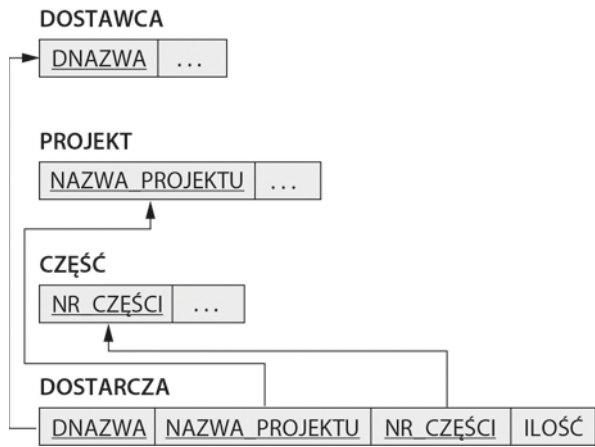
Dla kluczy obcych w nowej relacji  $R$  odpowiadającej atrybutowi wielowartościowemu powinniśmy określić opcję propagowania (przekazywania kaskadowego) działań wywoływanych za pośrednictwem odwołań (patrz podrozdział 6.2) — propagowanie działań powinno dotyczyć zarówno operacji aktualizacji, jak i operacji usuwania danych. Powinniśmy także pamiętać, że podczas odwzorowywania złożonego, wielowartościowego atrybutu dobór klucza dla relacji  $R$  wymaga przeprowadzenia analizy znaczenia poszczególnych atrybutów składowych. W niektórych przypadkach podczas odwzorowywania atrybutu wielowartościowego, który dodatkowo jest atrybutem złożonym, do klucza tworzonej relacji  $R$  można zaliczyć tylko niektóre atrybuty składowe — ich rola przypomina nieco znaczenie klucza częściowego w słabym typie encji, który odpowiada atrybutowi wielowartościowemu (patrz podrozdział 3.5).

Na rysunku 9.2 przedstawiono schemat relacyjnej bazy danych FIRMA uzyskany po wykonaniu kroków od 1. do 6., natomiast na rysunku 5.6 zaprezentowano przykładowy stan tej samej bazy danych. Łatwo zauważyć, że w naszych dotychczasowych rozważaniach pominęliśmy problem odwzorowywania typów związków  $n$ -składnikowych (gdzie  $n > 2$ ), ponieważ żaden taki związek nie istnieje w schemacie pokazanym na rysunku 9.1; tego rodzaju typy związków są odwzorowywane w podobny sposób jak typy związków ze współczynnikiem liczności równym  $M:N$ , czyli przez wykonanie opisanego poniżej, dodatkowego kroku algorytmu odwzorowującego.

**Krok 7: Odwzorowanie  $n$ -składnikowych typów związków.** Dla każdego typu  $n$ -składnikowego związku  $R$ , gdzie  $n > 2$ , tworzymy reprezentującą go nową relację  $S$ . Do nowej relacji  $S$  należy dołączyć (w postaci atrybutów klucza obcego) klucze główne relacji, które reprezentują typy encji występujące w odwzorowywanym  $n$ -składnikowym typie związku. Relacja  $S$  musi także zawierać wszystkie atrybuty proste (lub proste składniki atrybutów złożonych) danego  $n$ -składnikowego typu związku. Klucz główny relacji  $S$  jest zwykle kombinacją wszystkich kluczy obcych odwołujących się do relacji, które reprezentują typy encji występujące w odwzorowywanym typie związku. Jeśli jednak ograniczenie liczności dla któregośkolwiek z typów encji  $E$  występujących w typie związku  $R$  jest równe 1, klucz główny relacji  $S$  nie powinien obejmować atrybutu klucza obcego, który odwołuje się do relacji  $E'$  odpowiadającej typowi encji  $E$  (patrz punkt 3.9.2, poświęcony ograniczeniom związków  $n$ -składnikowych).

Przykładowo, rozważmy typ związku DOSTARCZA z rysunku 3.17, łączący dostawcę  $d$ , część  $c$  i projekt  $p$ , gdy  $d$  dostarcza  $c$  do  $p$ . Ten typ związku można odwzorować w przedstawioną na rysunku 9.4 relację DOSTARCZA, której kluczem głównym jest kombinacja trzech kluczy obcych {NAZWAD, NRCZĘŚCI, NAZWAPROJEKTU}.





RYSUNEK 9.4. Odwzorowanie n-składnikowego typu relacji DOSTARCZA z rysunku 3.17(a)

### 9.1.2. Omówienie i podsumowanie odwzorowania konstrukcji modelu ER w odpowiednie konstrukcje modelu relacyjnego

W tabeli 9.1 zawarto podsumowanie zgodności pomiędzy konstrukcjami i ograniczeniami (wiązaniami) stosowanymi w modelu związków encji (ER) oraz modelu relacyjnym.

TABELA 9.1. Zgodność poszczególnych konstrukcji modelu związków encji (ER) i modelu relacyjnego

MODEL ER	MODEL RELACYJNY
Typ encji	Relacja „encji”
Typ związku ze współczynnikiem liczności równym 1:1 lub 1:N	Klucz obcy (lub relacja „związku”)
Typ związku ze współczynnikiem liczności równym M:N	Relacja „związku” z dwoma kluczami obcymi
n-składnikowy typ związku	Relacja „związku” z n kluczami obcymi
Atrybut prosty	Atrybut
Atrybut złożony	Zbiór prostych atrybutów składowych
Atrybut wielowartościowy	Relacja i klucz obcy
Zbiór wartości	Dziedzina
Atrybut klucza	Klucz główny (lub klucz wtórny)

Jednym z najważniejszych elementów, na który warto zwrócić uwagę podczas analizy schematu relacyjnego, jest odmienny (w porównaniu ze schematem ER) sposób reprezentowania związków pomiędzy relacjami — w schematach relacyjnych nie stosuje się jawnych typów związków, a jedynie pary atrybutów *A* i *B*, z których jeden jest kluczem głównym, drugi jest kluczem obcym (oba zawierają wartości należące do tej samej dziedziny), dołączane do par relacji *S* i *T*. Dwie krotki w relacjach *S* i *T* są ze sobą powiązane, kiedy mają tę samą wartość odpowiednio atrybutów *A* i *B*. Stosując operację równozłączenia (lub złączenia naturalnego, jeśli para atrybutów łączenia ma taką samą nazwę) dla atrybutów *S.A* i *T.B*, możemy łatwo połączyć wszystkie pary powiązanych ze sobą krotek



z pary relacji  $S$  oraz  $T$  i tym samym zmaterializować tak reprezentowany związek. W sytuacji, gdy mamy do czynienia z typem związku binarnego ze współczynnikiem liczności równym 1:1 lub 1: $N$ , niezbędna jest tylko jedna operacja złączenia. W przypadku typu związku binarnego ze współczynnikiem liczności równym  $M:N$ , konieczne jest zastosowanie dwóch operacji złączenia, natomiast w przypadku typów związków  $n$ -składnikowych pełna materializacja egzemplarzy związku wymaga  $n$  operacji złączenia.

Przykładowo, aby stworzyć relację obejmującą nazwisko pracownika, nazwę projektu oraz liczbę godzin, jaką dany pracownik poświęca na realizację danego projektu, musimy połączyć każdą krotkę relacji PRACOWNIK z odpowiednią krotką relacji PROJEKT za pośrednictwem relacji PRACUJE\_NAD (patrz rysunek 9.2). Oznacza to, że musimy zastosować dla relacji PRACOWNIK i PRACUJE\_NAD operację równo-złączenia z warunkiem złączenia w postaci  $PESEL = PESELPRAC$ , po czym zastosować operację równo-złączenia dla otrzymanej relacji wynikowej i relacji PROJEKT, tym razem z warunkiem łączenia w postaci  $NR\_PROJ = NUMERPROJ$ . Ogólnie rzecz biorąc, kiedy niezbędne jest przeszukiwanie większej liczby relacji, konieczne jest zdefiniowanie wielu operacji złączenia. Użytkownik relacyjnej bazy danych zawsze musi mieć świadomość istnienia i znaczenia atrybutów kluczy obcych, aby mieć możliwość prawidłowego ich wykorzystywania i efektywnego łączenia krotek pochodzących z dwóch lub więcej relacji. Ta reguła jest niekiedy uważana za wadę relacyjnego modelu danych, ponieważ dopasowanie kluczy obcych i kluczy głównych nie zawsze jest oczywiste po samej analizie schematów relacyjnych. Jeśli operacja równo-złączenia jest wykonywana pomiędzy atrybutami dwóch relacji, które nie reprezentują związku klucza obcego-klucza głównego, otrzymany wynik często może być niezrozumiały i prowadzić do fałszywych (nieprawidłowych) danych. Przykładowo, Czytelnik może spróbować połączyć relacje PROJEKT i LOKALIZACJE\_DZIAŁÓW z warunkiem łączenia w postaci  $LOKALIZACJADZ = LOKALIZACJAPROJ$  i przeanalizować otrzymany rezultat.

Kolejnym elementem wartym zainteresowania w procesie projektowania schematu relacyjnego jest możliwość tworzenia osobnych relacji dla *każdego* atrybutu wielowartościowego. W przypadku konkretnej encji z określonym zbiorem wartości zdefiniowanym dla atrybutu wielowartościowego, w poszczególnych krotkach wartość atrybutu klucza może występować tylko raz dla każdej z wartości atrybutu wielowartościowego. Wynika to z faktu, że podstawowy model relacyjny *nie* przewiduje możliwości umieszczania wielu wartości (w postaci listy lub zbioru) w jednym atrybucie pojedynczej krotki. Przykładowo, ponieważ dział oznaczony numerem 5. ma trzy lokalizacje, w relacji LOKALIZACJE\_DZIAŁÓW występują trzy krotki (patrz rysunek 5.6), z których każda określa tylko jedną taką lokalizację. W analizowanym przez nas przykładzie zastosowaliśmy operację równo-złączenia dla relacji LOKALIZACJE\_DZIAŁÓW i DZIAŁ oraz atrybutu NUMERDZ, aby wraz z pozostałymi atrybutami relacji DZIAŁ otrzymać wartości wszystkich lokalizacji. Jak nie-trudno się domyślić, w relacji wynikowej wartości pozostałych atrybutów relacji DZIAŁ będą się powtarzały w różnych krotkach reprezentujących te same lokalizacje działów.

Podstawowa algebra relacyjna nie oferuje operacji zagnieżdżania ani kondensowania, których moglibyśmy użyć do wygenerowania w oparciu o zawartość relacji LOKALIZACJE\_DZIAŁÓW (patrz rysunek 5.6) zbioru krotek w postaci:  $\{<1, Szczecin>, <4, Poznań>, <5, \{Bydgoszcz, świebodzin, Szczecin\}>\}$ . Brak możliwości bezpośredniego otrzymania takiego wyniku jest poważną wadą znormalizowanej ( *płaskiej* ) wersji modelu relacyjnego. Model obiektowy i systemy obiektowo-relacyjne (patrz rozdział 12.) umożliwiają stosowanie atrybutów wielowartościowych za pomocą atrybutów typów tablicowych.

## 9.2. Odwzorowania konstrukcji modelu EER w relacje

Przystąpimy teraz do omawiania metod odwzorowywania konstrukcji rozszerzonego modelu ER (w skrócie EER) w odpowiednie relacje — w tym celu rozszerzymy zaprezentowany w punkcie 9.1.1 algorytm odwzorowujący konstrukcje modelu ER w elementy modelu relacyjnego.

### 9.2.1. Odwzorowywanie specjalizacji i generalizacji

Istnieje wiele możliwości w zakresie odwzorowywania zróżnicowanych podklas, które razem tworzą specjalizację (lub takich, które są generalizowane do jednej nadklasy), jak choćby zbiór podklas {SEKRETARKA, TECHNIK, INŻYNIER} nadklasy PRACOWNIK (patrz rysunek 4.4). Dwie podstawowe możliwości to odwzorowanie całej specjalizacji na **jedną tabelę** lub na **wiele tabel**. W ramach każdej z tych możliwości występują wersje zależne od ograniczeń dotyczących specjalizacji i generalizacji.

Możemy rozszerzyć przedstawiony w punkcie 9.1.1 algorytm odwzorowujący konstrukcje modelu ER w odpowiadające im elementy modelu relacyjnego przez dodanie do niego kolejnego kroku, którego zadaniem będzie obsługa odwzorowywania specjalizacji. Przedstawiony poniżej krok 8. algorytmu zapewnia jedynie najczęściej stosowane rozwiązania; w praktyce możliwe jest stosowanie także innych odwzorowań. W dalszej części tego punktu omówimy warunki, pod którymi można stosować poszczególne metody. *Atrybuty relacji  $R$*  będziemy zapisywać w postaci  $\text{Atr}(R)$ , natomiast *klucz główny relacji  $R$*  będziemy oznaczali za pomocą wyrażenia  $\text{KG}(R)$ . Najpierw opiszemy odwzorowania formalnie, a następnie przedstawimy przykłady.

**Krok 8: Metody odwzorowywania specjalizacji lub generalizacji.** Każdą specjalizację z  $m$  podklasami  $\{S_1, S_2, \dots, S_m\}$  oraz (stanowiącą ich generalizację) nadklasę  $C$  — gdzie  $\{k, a_1, a_2, \dots, a_n\}$  jest zbiorem atrybutów nadklasy  $C$  oraz  $k$  jest jej kluczem (głównym) — odwzorowujemy w odpowiednie schematy relacji, stosując jedną z poniższych metod:

- **Metoda 8A: Wiele relacji — dla nadklasy i podklas.** Tworzymy dla nadklasy  $C$  relację  $L$  z atrybutami  $\text{Atr}(L) = \{k, a_1, a_2, \dots, a_n\}$  oraz kluczem głównym  $\text{KG}(L) = k$ . Tworzymy relację  $L_i$  dla każdej podklasy  $S_i$ , gdzie  $1 \leq i \leq m$ , z atrybutami  $\text{Atr}(L_i) = \{k\} \cup \{\text{atrybuty podklasy } S_i\}$  oraz kluczem głównym  $\text{KG}(L_i) = k$ . To rozwiązanie sprawdza się w przypadku wszystkich rodzajów specjalizacji (pełnej i częściowej oraz rozłącznej i pokrywającej).
- **Metoda 8B: Wiele relacji — wyłącznie dla podklas.** Tworzymy po jednej relacji  $L_i$  dla każdej podklasy  $S_i$ , gdzie  $1 \leq i \leq m$ , z atrybutami  $\text{Atr}(L_i) = \{\text{atrybuty podklasy } S_i\} \cup \{k, a_1, a_2, \dots, a_n\}$  oraz kluczem głównym  $\text{KG}(L_i) = k$ . To rozwiązanie sprawdza się tylko w przypadku tych specjalizacji, w których podklasy są *całkowite* (a więc każda encja nadklasy musi należeć przynajmniej do jednej z jej podklas). Ponadto technika ta jest zalecana tylko dla specjalizacji z *ograniczeniem rozłączności* (patrz punkt 4.3.1). W specjalizacjach pokrywających ta sama encja może zostać powielona w kilku relacjach.
- **Metoda 8C: Pojedyncza relacja z jednym atrybutem typu.** Tworzymy pojedynczą relację  $L$  z atrybutami  $\text{Atr}(L) = \{k, a_1, a_2, \dots, a_n\} \cup \{\text{atrybuty podklasy } S_i\} \cup \dots \cup \{\text{atrybuty podklasy } S_m\} \cup \{t\}$  oraz kluczem głównym  $\text{KG}(L) = k$ . Atrybut  $t$  jest nazywany

atrybutem **typu** (lub atrybutem **dyskryminującym**) i reprezentuje informacje umożliwiające identyfikację podklas, do których należą poszczególne krotki. To rozwiązanie sprawdza się tylko w przypadku tych specjalizacji, w których podklasy są *rozłączne*, i może się wiązać z dużą liczbą wartości pustych, jeśli reprezentowane podklasy zawierają dużo specyficznych dla siebie atrybutów.

- **Metoda 8D: Pojedyncza relacja z wieloma atrybutami typów.** Tworzymy pojedynczy schemat relacji  $L$  z atrybutami  $\text{Attr}(L) = \{k, a_1, a_2, \dots, a_n\} \cup \{\text{atrybuty podklasy } S_i\} \cup \dots \cup \{\text{atrybuty podklasy } S_m\} \cup \{t_1, t_2, \dots, t_m\}$  oraz kluczem głównym  $\text{KG}(L) = k$ . Każdy atrybut  $t_i$ , gdzie  $1 \leq i \leq m$ , jest **logicznym atrybutem typu** i określa, czy dana krotka należy do podklasy  $S_i$ . To rozwiązanie sprawdza się tylko w przypadku tych specjalizacji, w których podklasy częściowo lub w całości się *pokrywają* (ale może być stosowane także w przypadku specjalizacji rozłącznych).

Metody 8A i 8B można nazwać **rozwiązaniami wielorelacyjnymi**, natomiast metody 8C i 8D — **rozwiązaniami jednorelacyjnymi**. Metoda 8A przewiduje tworzenie relacji  $L$  dla nadklasy  $C$  i jej atrybutów oraz po jednej relacji  $L_i$  dla każdej podklasy  $S_i$ ; każda relacja  $L_i$  zawiera nie tylko specyficzne (lokalne) atrybuty reprezentowanej podklasy  $S_i$ , ale także klucz główny nadklasy  $C$ , który jest przekazywany do wszystkich relacji  $L_i$  i staje się ich kluczem głównym. Operacja równo-złączenia oparta na wartościach kluczy głównych dowolnej pary relacji  $L_i$  i  $L$  zwraca wszystkie lokalne i odziedziczone atrybuty encji w podklasie  $S_i$ . Efekt zastosowania tego rozwiązania w praktyce przedstawiono na rysunku 9.5(a), gdzie odwzorowano schemat EER z rysunku 4.4. Rozwiązanie 8A może być stosowane niezależnie od wszelkich ograniczeń (więzów) nałożonych na daną specjalizację (rozłączna lub pokrywająca oraz pełna lub częściowa). Warto zauważyć, że dla każdej relacji  $L_i$  musi być spełniony warunek:

$$\pi_{\langle K \rangle}(L_i) \subseteq \pi_{\langle K \rangle}(L)$$

(a) PRACOWNIK



(b) SAMOCHÓD

NrKartyPojazdu	NumerRejestracyjny	Cena	PrędkośćMaksymalna	LiczbaPasażerów
----------------	--------------------	------	--------------------	-----------------

CIEŻARÓWKA

NrKartyPojazdu	NumerRejestracyjny	Cena	LiczbaOsi	Tonaż
----------------	--------------------	------	-----------	-------

(c) PRACOWNIK

PESEL	Płmię	Dłmię	Nazwisko	DataUr	Adres	TypPracy	SzybkośćMaszynopisania	StopieńTechniczny	WykształcenieInżynierskie
-------	-------	-------	----------	--------	-------	----------	------------------------	-------------------	---------------------------

(d) CZĘŚĆ

NumerCzęści	Opis	WFlaga	NumerProjektu	DataProdukcji	NumerSerii	ZFlaga	NazwaDostawcy	ListaCen
-------------	------	--------	---------------	---------------	------------	--------	---------------	----------

RYSUNEK 9.5. Dostępne rozwiązania w zakresie odwzorowywania specjalizacji lub generalizacji.

(a) Odwzorowanie schematu EER z rysunku 4.4 zgodnie z metodą 8.A. (b) Odwzorowanie schematu EER z rysunku 4.3(b) zgodnie z metodą 8.B. (c) Odwzorowanie schematu EER z rysunku 4.4 zgodnie z metodą 8.C. (d) Odwzorowanie schematu z rysunku 4.5 zgodnie z metodą 8.D z logicznymi polami typów WFlaga oraz ZFlaga

Ograniczenie to definiuje w każdej z relacji  $L_i$  klucz obcy, który wskazuje na relację  $L$ .

Operacja równo-złączenia w metodzie 8B jest *wbudowana* w schemat, co eliminuje konieczność stosowania dodatkowej relacji  $L$  — przykład takiego rozwiązania przedstawiono na rysunku 9.5(b), gdzie odwzorowano specjalizację ze schematu EER zaprezentowanego na rysunku 4.3(b). Stosowanie tego rozwiązania jest uzasadnione tylko w sytuacji, gdy spełniony jest *zarówno* warunek rozłączności, jak i warunek całkowitości. Jeśli odwzorowywana specjalizacja nie jest pełna (całkowita), encja, która nie należy do żadnej z podklas  $S_i$ , zostanie w procesie odwzorowywania utracona. Jeśli natomiast dana specjalizacja nie jest rozłączna, encja należąca do więcej niż jednej podklasy będzie nadmiarowo przechowywała atrybuty odziedziczone z nadklasy  $C$  w więcej niż jednej relacji  $L_i$ . Zastosowanie metody 8B oznacza, że żadna z relacji otrzymanych wskutek odwzorowania nie zawiera wszystkich encji z nadklasy  $C$ , zatem aby otrzymać kompletny zbiór tych encji, musimy zastosować operację złączenia zewnętrznego (lub pełnego złączenia zewnętrznego; patrz podrozdział 6.4) dla wszystkich operacji  $L_i$ . Wynik takiego złączenia zewnętrznego będzie podobny do relacji otrzymanych po odwzorowaniu specjalizacji zgodnie ze schematami 8C i 8D z tą różnicą, że nie będą zawierały pól typów. Za każdym razem, gdy szukamy dowolnej encji w nadklasie  $C$ , musimy przeszukać wszystkie  $m$  relacji  $L_i$ .

Rozwiązania 8C i 8D przewidują tworzenie pojedynczych relacji, które jednocześnie reprezentują nadklasę  $C$  oraz wszystkie jej podklasy. Encja nadklasy, która nie należy do żadnej podklasy, będzie miała wartości puste we wszystkich specyficznych (lokalnych) atrybutach podklas. Metody 8C i 8D nie są więc zalecane, jeśli dla podklas specjalizacji zdefiniowano wiele specyficznych (lokalnych) atrybutów. Jeśli jednak liczba specyficznych atrybutów w podklasach jest stosunkowo niewielka, odwzorowania oparte na rozwiązaniach 8C i 8D sprawdzają się lepiej niż metody 8A i 8B, ponieważ można w ten sposób wyeliminować konieczność wykonywania dodatkowych operacji równo-złączenia i złączenia zewnętrznego (a więc uzyskać bardziej efektywną implementację zapytań).

Metoda 8C jest stosowana przede wszystkim do obsługi rozłącznych podklas — odwzorowanie polega na dołączeniu pojedynczego **atrybutu typu** (**atrybutu obrazu** lub **atrybutu dyskryminującego**)  $t$ . Jego wartość określa, do których z  $m$  podklas należą poszczególne krotki; oznacza to, że dziedziną wartości atrybutu  $t$  może być zbiór  $\{1, 2, \dots, m\}$ . Jeśli odwzorowywana specjalizacja jest częściowa, atrybut  $t$  może zawierać wartości puste w tych krotkach, które nie należą do żadnej z reprezentowanych podklas. Jeśli odwzorowywana specjalizacja jest zdefiniowana przez atrybut, atrybut definiujący pełni jednocześnie rolę naszego atrybutu  $t$ , który w tym przypadku nie jest konieczny; na rysunku 9.5(c) przedstawiono efekt zastosowania tego rozwiązania dla specjalizacji EER zaprezentowanej na rysunku 4.4.

Metoda 8D została zaprojektowana z myślą o obsłudze częściowo pokrywających się podklas — odwzorowanie polega na dołączeniu  $m$  *logicznych* pól **typów**, po jednym dla *każdej* z odwzorowywanych podklas. Metoda 8D może być stosowana także do odwzorowywania rozłącznych podklas. Każde pole typu  $t_i$  może przechowywać wartość z dziedziny {tak, nie}, gdzie wartość „tak” oznacza, że dana krotka jest egzemplarzem podklasy  $S_i$ . Gdybyśmy zastosowali to rozwiązanie dla specjalizacji EER z rysunku 4.4, musielibyśmy dołączyć do relacji wynikowej trzy atrybuty typów (JestSekretarką, JestInżynierem oraz JestTechnikiem) zamiast — jak na rysunku 9.5(c) — jednego atrybutu TypPracy. Na rysunku 9.5(d) pokazano odwzorowanie specjalizacji z rysunku 4.5 za pomocą metody 8D.

W sytuacjach, gdy odwzorowanie ma dotyczyć wielopoziomowej hierarchii lub kraty specjalizacji (generalizacji), nie musimy koniecznie stosować tej samej metody dla wszystkich przekształcanych poziomów. Zamiast tego, dla części takiej hierarchii lub kraty możemy wykorzystać jedno z rozwiązań, a dla pozostałych elementów użyć pozostałych dostępnych metod. Na rysunku 9.6 przedstawiono jedno z możliwych odwzorowań w relacje kraty EER z rysunku 4.6. W tym przypadku zastosowaliśmy metodę 8A dla specjalizacji OSOBA/{PRACOWNIK, ABSOLWENT, STUDENT}, metodę 8C dla specjalizacji PRACOWNIK/{PRACOWNIK\_TECHNICZNY, PRACOWNIK\_NAUKOWY, ASYSTENT} (dodając atrybut typu TypPracownika). Następnie zastosowaliśmy metodę 8D z jedną tabelą dla specjalizacji ASYSTENT/{ASYSTENT\_BADAWCZY, ASYSTENT\_DYDAKTYCZNY}, dodając atrybuty typów ABFlaga i ADFlaga w relacji PRACOWNIK. Tę samą metodę zastosowaliśmy dla pary STUDENT/ASYSTENT, dodając atrybut typu AsystentFlaga w relacji STUDENT, a także dla specjalizacji STUDENT/{MAGISTRANT, STUDENT\_I\_IV}, umieszczając atrybuty typu MagistrantFlaga i StudentFlaga w relacji STUDENT. Wszystkie atrybuty użyte na rysunku 9.6, których nazwy zawierają słowa *Typ* i *Flaga*, są polami typów.



RYСУNEK 9.6. Odwzorowanie kraty specjalizacji modelu EER z rysunku 4.8 w oparciu o różne metody przekształceń

## 9.2.2. Odwzorowywanie współdzielonych podklas (konstrukcji dziedziczenia wielokrotnego)

Współdzielona podklasa (np. KIEROWNIK\_TECHNICZNY z rysunku 4.6) jest po prostu podklasą więcej niż jednej nadklasy, co musi się wiązać z mechanizmem tzw. wielokrotnego dziedziczenia. Wszystkie takie klasy muszą mieć ten sam atrybut klucza; w przeciwnym przypadku współdzielona podklasa byłaby modelowana w postaci kategorii, co opisano w podrozdziale 4.4. Podczas odwzorowywania współdzielonych podklas możemy stosować dowolną z metod omówionych w kroku 8. prezentowanego algorytmu, musimy jednak mieć na uwadze przedstawione ograniczenia algorytmu odwzorowującego. Przykładowo, w przypadku odwzorowanej na rysunku 9.6 współdzielonej podklasy ASYSTENT użyto zarówno metody 8C, jak i 8D. Podczas tworzenia relacji PRACOWNIK (z atrybutem TypPracownika) zastosowano metodę 8C, natomiast podczas tworzenia relacji STUDENT (z atrybutem FlagaAsystenta) wykorzystano metodę 8D.



### 9.2.3. Odwzorowywanie kategorii (typów unii)

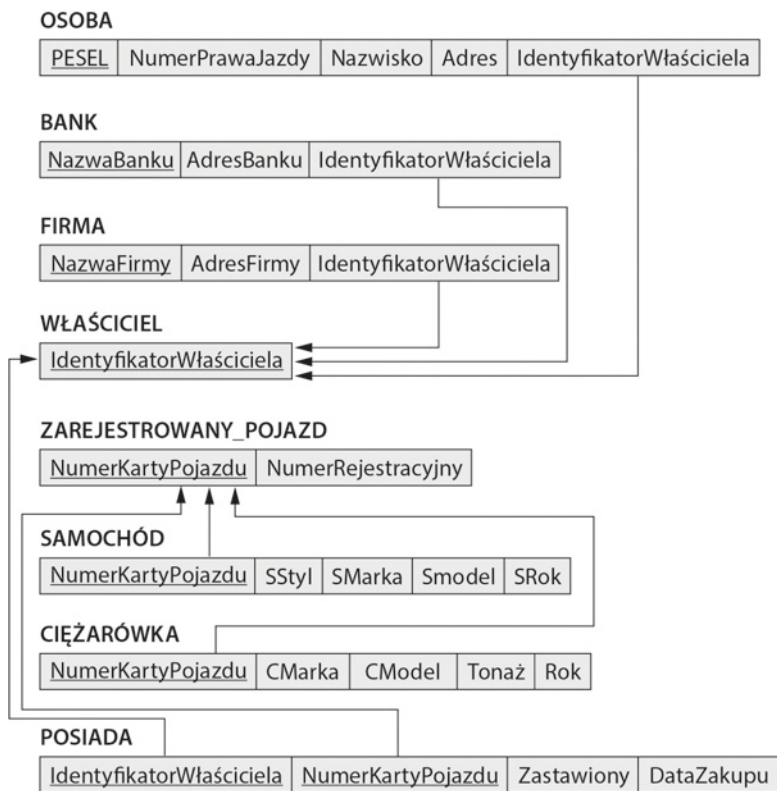
W tym punkcie dodamy kolejny (dziewiąty) krok do naszej procedury odwzorowywania konstrukcji modelu EER; w ten sposób zapewnimy właściwą obsługę kategorii. Kategoria (lub typ unii) jest podklasą *unii* dwóch lub większej liczby nadklas, które mogą mieć różne klucze, ponieważ mogą należeć do zupełnie innych typów encji (patrz podrozdział 4.4). Przykładem kategorii jest przedstawiony na rysunku 4.8 typ encji WŁAŚCICIEL, który jest podzbiorem unii trzech innych typów encji: OSOBA, BANK oraz FIRMA. Druga kategoria na rysunku 4.8 (ZAREJESTROWANY\_POJAZD) ma dwie nadklasy z tym samym atrybutem klucza.

**Krok 9: Odwzorowywanie typów unii (kategorii).** W przypadku kategorii, której nadklasy definiujące mają różne klucze, musimy w tradycyjny sposób określić nowy atrybut klucza (nazywany **kluczem sztucznym**) w trakcie tworzenia relacji będącej reprezentacją odwzorowywanej kategorii. Wynika to z faktu, że klucze klas definiujących są różne, zatem nie możemy wybrać tylko jednego z nich do roli unikatowego identyfikatora wszystkich encji kategorii. W analizowanym przez nas przykładzie z rysunku 4.8 możemy stworzyć relację WŁAŚCICIEL, która będzie odpowiadała kategorii WŁAŚCICIEL (patrz rysunek 9.7), i dołączyć do nowej relacji wszystkie atrybuty odwzorowywanej kategorii. Klucz główny relacji WŁAŚCICIEL jest tzw. kluczem sztucznym, nazwaną Identyfikator ↗ Właściciela. Atrybut klucza sztucznego (IdentyfikatorWłaściciela) musi także zostać dołączony (w postaci klucza obcego) do wszystkich relacji odpowiadających nadklasom odwzorowanej kategorii — w ten sposób zachowujemy informacje o powiązaniach wartości klucza sztucznego i kluczy poszczególnych nadklas. Łatwo zauważyć, że jeśli konkretna encja typu OSOBA (lub typu BANK albo FIRMA) nie należy do typu encji WŁAŚCICIEL, odpowiednia krotka w relacji OSOBA (lub relacji BANK albo FIRMA) będzie zawierała wartość pustą w atrybucie IdentyfikatorWłaściciela i nie będzie miała swojego odpowiednika w postaci krotki relacji WŁAŚCICIEL. Ponadto zalecane jest dodanie atrybutu typu (niepokazany na rysunku 9.7) do relacji WŁAŚCICIEL, aby określić konkretny typ encji, do którego należy każda krotka (OSOBA, BANK lub FIRMA).

W przypadku kategorii, których nadklasy mają zdefiniowany ten sam klucz (przykładem takiego rozwiązania jest typ encji POJAZD z rysunku 4.8), nie ma potrzeby dołączania do relacji wynikowej klucza sztucznego. Na rysunku 9.7 przedstawiono prawidłowe odwzorowanie kategorii ZAREJESTROWANY\_POJAZD, która jest przykładem nadklasy z identycznym kluczem.

## 9.3. Podsumowanie

W podrozdziale 9.1 przedstawiliśmy sposób odwzorowywania opracowanego w modelu związków encji (ER) projektu schematu koncepcyjnego w schemat relacyjnej bazy danych. Zaprezentowany algorytm odwzorowujący zilustrowaliśmy przykładami wyodrębnionymi z bazy danych FIRMA. W tabeli 9.1 przedstawiono podsumowanie odpowiadających sobie konstrukcji i ograniczeń (więzów) modelu ER i modelu relacyjnego. W podrozdziale 9.2 dołączyliśmy do omawianego algorytmu dodatkowe kroki, które umożliwiają odwzorowywanie konstrukcji typowych dla rozszerzonego modelu ER (EER) w odpowiednie elementy modelu relacyjnego. Podobne algorytmy stosuje się w graficznych narzędziach projektowania baz danych — dzięki nim można automatycznie tworzyć schematy relacyjne na bazie projektów schematów koncepcyjnych.



RYSUNEK 9.7. Odwzorowanie kategorii (typów unii) modelu EER z rysunku 4.8 w relacje

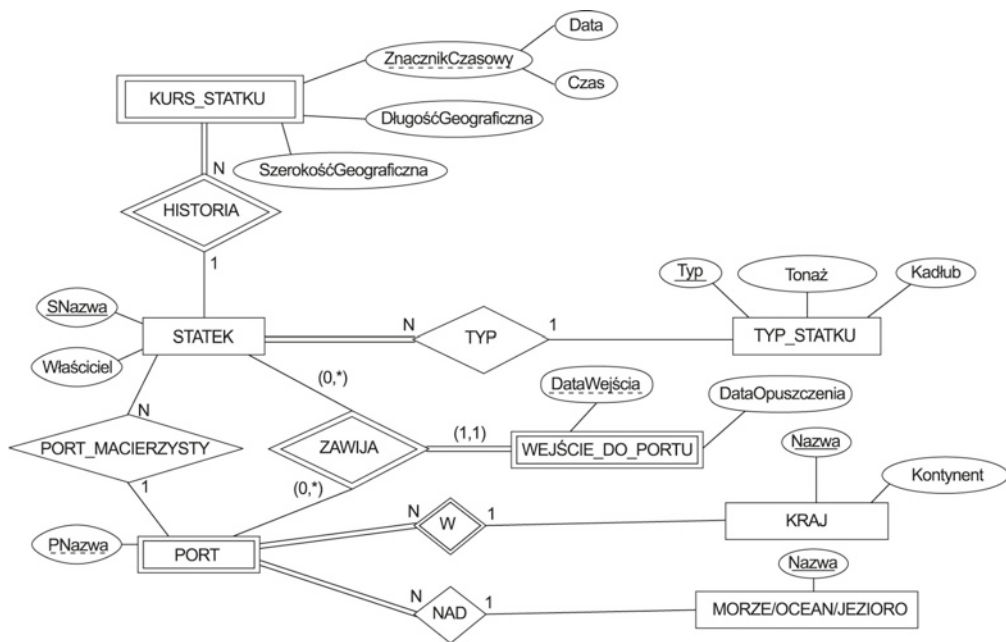
## Pytania powtórkowe

- 9.1. (a) Omów analogie pomiędzy konstrukcjami modelu związków encji (ER) a konstrukcjami modelu relacyjnego. Pokaż, jak poszczególne konstrukcje modelu ER można odwzorowywać w modelu relacyjnym, i omów ewentualne rozwiązania alternatywne w zakresie tych odwzorowań.
- (b) Omów metody odwzorowywania konstrukcji modelu EER w relacje, a także warunki, w jakich można stosować poszczególne z tych technik.

## Ćwiczenia

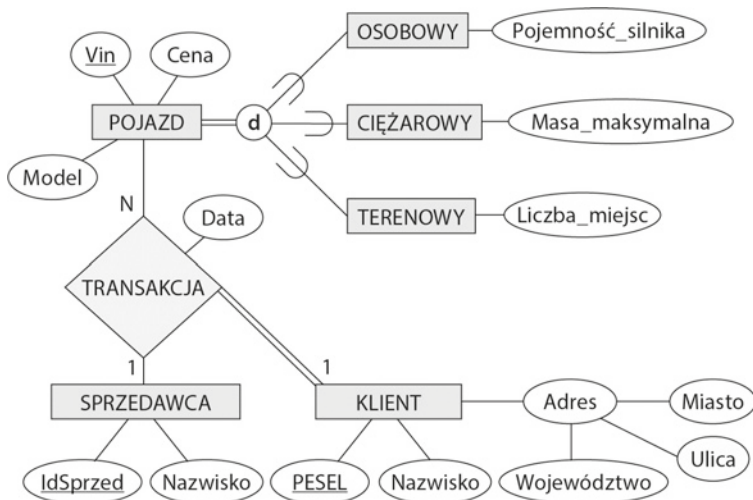
- 9.2. Odwzoruj schemat bazy danych UNIWERSYTET z rysunku 3.20 na schemat relacyjnej bazy danych.
- 9.3. Spróbuj odwzorować zaprezentowany na rysunku 6.14 schemat relacyjny w odpowiadający mu schemat związków encji. Działania tego typu mogą być częścią procesu nazywanego *inżynierią odwrotną* (*inżynierią wsteczną*), w którym schemat koncepcyjny jest tworzony w oparciu o istniejącą implementację bazy danych. Opisz wszystkie założenia przyjęte podczas realizacji tego zadania.





RYSUNEK 9.8. Schemat ER dla bazy danych ŚLEDZENIE\_STATKÓW

- 9.4. Na rysunku 9.8 przedstawiono schemat ER dla bazy danych, która może być wykorzystywana do przechowywania informacji o statkach transportowych, ich położeniu oraz o władzach morskich. Przekształć (odwzoruj) ten schemat w schemat relacyjny i wskaż wszystkie klucze główne oraz klucze obce.
- 9.5. Spróbuj odwzorować omówiony w ćwiczeniu 3.23 schemat związków encji dla bazy danych BANK (patrz rysunek 3.21) w schemat relacyjny. Wskaż wszystkie klucze główne i klucze obce nowego schematu. Powtórz to zadanie dla schematu LINIE\_LOTNICZE z ćwiczenia 3.19 (patrz rysunek 3.20) oraz dla pozostałych schematów omówionych w ćwiczeniach od 3.16 do 3.24.
- 9.6. Odwzoruj diagramy EER z rysunków 4.9 i 4.12 w odpowiednie schematy relacyjne. Wyjaśnij wszystkie podjęte decyzje dotyczące metod odwzorowywania.
- 9.7. Czy możliwe jest odwzorowanie binarnego związku M:N bez konieczności tworzenia nowej relacji? Odpowiedź uzasadnij.
- 9.8. Przyjrzyj się diagramowi EER z rysunku 9.9. Diagram ten ilustruje salon samochodowy.  
Odwzoruj ten schemat EER na zbiór relacji. Na potrzeby generalizacji relacji POJAZD na relacje OSOBOWY, CIĘŻAROWY, TERENOWY rozważ cztery metody opisane w punkcie 9.2.1 i przedstaw projekt schematu relacyjnego uzyskany za pomocą każdej z tych technik.
- 9.9. Używając atrybutów z diagramu EER z ćwiczenia 4.27, odwzoruj kompletny schemat na zbiór relacji. W celu odwzorowania generalizacji zastosuj odpowiednią metodę (z listy od 8A do 8D) z punktu 9.2.1 i uzasadnij dokonany wybór.



RYSUNEK 9.9. Diagram EER reprezentujący salon samochodowy

## Ćwiczenia laboratoryjne

- 9.10. Przyjrzyj się projektowi ER bazy danych `UNIwersytet`, której model zbudowano za pomocą narzędzi takich jak ERwin lub Rational Rose w ćwiczeniu laboratoryjnym 3.31. Za pomocą dostępnego w danym narzędziu mechanizmu generowania schematów w języku SQL utwórz taki schemat dla bazy danych Oracle.
- 9.11. Przyjrzyj się projektowi ER bazy danych `Zamówienia_Pocztowe`, której model zbudowano za pomocą narzędzi takich jak ERwin lub Rational Rose w ćwiczeniu laboratoryjnym 3.32. Za pomocą dostępnego w danym narzędziu mechanizmu generowania schematów w języku SQL utwórz taki schemat dla bazy danych Oracle.
- 9.12. Przyjrzyj się projektowi ER bazy danych `Recenzje_Prac_na_Konferencje`, której model zbudowano za pomocą narzędzi takich jak ERwin lub Rational Rose w ćwiczeniu laboratoryjnym 3.34. Za pomocą dostępnego w danym narzędziu mechanizmu generowania schematów w języku SQL utwórz taki schemat dla bazy danych Oracle.
- 9.13. Przyjrzyj się projektowi ER bazy danych `Dziennik_Ocen`, której model zbudowano za pomocą narzędzi takich jak ERwin lub Rational Rose w ćwiczeniu laboratoryjnym 4.28. Za pomocą dostępnego w danym narzędziu mechanizmu generowania schematów w języku SQL utwórz taki schemat dla bazy danych Oracle.
- 9.14. Przyjrzyj się projektowi ER bazy danych `Aukcje_Internetowe`, której model zbudowano za pomocą narzędzi takich jak ERwin lub Rational Rose w ćwiczeniu laboratoryjnym 4.29. Za pomocą dostępnego w danym narzędziu mechanizmu generowania schematów w języku SQL utwórz taki schemat dla bazy danych Oracle.

## Wybrane publikacje

Oryginalny algorytm odwzorowujący konstrukcję modelu ER w elementy modelu relacyjnego został opisany w zaliczanej do klasyki publikacji Chena (1976). Batini i in. (1992) opisali różne algorytmy odwzorowywania modeli ER i EER na starsze modele oraz dokonywania przekształceń odwrotnych.



# IV

---

## Techniki programowania baz danych





# Wprowadzenie do technik programowania w języku SQL

W rozdziałach 6. i 7. opisaliśmy kilka aspektów języka SQL, który jest standardowym rozwiązaniem we współczesnych systemach zarządzania relacyjnymi bazami danych. Przedstawiliśmy polecenia tego języka wykorzystywane do definiowania danych, modyfikowania schematu oraz konstruowania zapytań i operacji aktualizujących. Zaprezentowaliśmy także sposób definiowania najbardziej popularnych ograniczeń, w tym ograniczeń klucza i więzów integralności odwołań.

W kilku kolejnych podrozdziałach tego rozdziału omówimy rozmaite techniki programowania, które umożliwiają uzyskiwanie dostępu do informacji zawartych w bazie danych z poziomu programów. W praktyce, większość operacji dostępu do baz danych odbywa się właśnie za pośrednictwem zewnętrznych programów, które implementują **aplikacje baz danych**. Tego typu oprogramowanie jest zwykle tworzone w takich językach programowania jak Java, C/C++/C# i (kiedyś) COBOL. Ponadto do programowania dostępu do baz danych w aplikacjach internetowych używane są liczne języki skryptowe takie jak PHP, Python i JavaScript. W niniejszym rozdziale koncentrujemy się na tym, jak uzyskać dostęp do baz danych za pomocą tradycyjnych języków programowania (C, C++ i Javy); w następnym rozdziale znajdziesz wprowadzenie do tego, jak korzystać z baz za pomocą języków skryptowych takich jak PHP. W punkcie 2.3.1 stwierdziliśmy, że kiedy polecenia wykonywane na bazie danych są dołączane do programu, uniwersalny język programowania jest nazywany *językiem nadrzędnym*, natomiast język samej bazy danych (w naszym przypadku SQL) jest nazywany *podjęzykiem danych*. W niektórych przypadkach tworzone są *specjalne języki programowania baz danych* przeznaczone do pisania aplikacji baz danych. Chociaż wiele tego typu rozwiązań doczekało się jedynie postaci prototypów badawczych, niektóre zaprojektowane w ten sposób języki zyskały sporą popularność, jak choćby język *PL/SQL* (ang. *Programming Language/SQL*) firmy Oracle.

Warto zauważyć, że programowanie baz danych to bardzo obszerne zagadnienie. Istnieją całe książki poświęcone poszczególnym technikom programowania baz danych i temu, jak metody te są realizowane w konkretnych systemach. Cały czas rozwijane są też nowe techniki, a zmiany w istniejących metodach są włączane w nowsze wersje systemów i języki. Dodatkową trudnością w prezentowaniu tego zagadnienia jest to, że choć istnieją standardy języka SQL, podlegają one ciągłym zmianom, a każdy producent SZBD może wprowadzać modyfikacje względem standardu. Z tego powodu zdecydowaliśmy się przedstawić wprowadzenie do wybranych podstawowych technik programowania baz danych i porównać je ze sobą, zamiast szczegółowo analizować jedną konkretną metodę



lub jeden system. Przedstawione tu przykłady ilustrują główne różnice, z którymi programista zetknie się, korzystając z poszczególnych technik. W przykładach staramy się posługiwać standardami języka SQL, zamiast opisywać konkretny system. Jeśli używasz określonego systemu, materiały z tego rozdziału mogą posłużyć za wprowadzenie, jednak należy je wzbogacić o podręcznik lub książki dotyczące konkretnego narzędzia.

Naszą prezentację programowania baz danych rozpoczniemy w podrozdziale 10.1 od przeglądu różnych technik, które zaprojektowano z myślą o dostępie do baz danych z poziomu programów. Następnie (w podrozdziale 10.2) omówimy reguły osadzania poleceń języka SQL w kodzie uniwersalnego języka programowania (tak zastosowany język SQL jest nazywany *osadzonym SQL-em*). W tym samym podrozdziale przedstawimy krótko tzw. *dynamiczny język SQL*, w którym zapytania mogą być konstruowane dynamicznie w czasie działania aplikacji, i zaprezentujemy podstawy osadzania poleceń języka SQL w oparciu o technologię SQLJ, którą zaprojektowano specjalnie dla języka programowania Java. W podrozdziale 10.3 omówimy technikę znaną jako *SQL/CLI* (ang. *Call Level Interface*), w której do uzyskiwania dostępu do bazy danych wykorzystywana jest specjalna biblioteka procedur i funkcji. Przez lata zaproponowano wiele różnych bibliotek oferujących funkcje przeznaczone do realizacji podobnych zadań. Zestaw funkcji SQL/CLI został zatwierdzony jako część standardu SQL. Inną popularną biblioteką funkcji jest *ODBC* (ang. *Open DataBase Connectivity*), podobna pod wieloma względami do technologii SQL/CLI (SQL/CLI można traktować jak ustandaryzowaną wersję ODBC). Trzecią tego typu biblioteką (którą opiszemy) jest *JDBC*, czyli zestaw funkcji opracowanych specjalnie z myślą o dostępie do baz danych z poziomu oprogramowania tworzonego w obiektowym języku Java. W językach obiektowych zamiast bibliotek funkcji lub procedur używane są klasy. Każda klasa obejmuje własne operacje i funkcje. W podrozdziale 10.4 omówimy technikę *SQL/PSM* (ang. *Persistent Stored Modules*), która jest częścią standardu SQL i umożliwia składowanie modułów programowym (procedur i funkcji) w systemie zarządzania bazą danych oraz uzyskiwanie dostępu do tych modułów za pomocą języka SQL. Przedstawimy też proceduralny *język programowania baz danych* służący do pisania trwałych modułów składowanych. W podrozdziale 10.5 pokrótce porównamy trzy opisane sposoby programowania baz danych, a w podrozdziale 10.6 zawarliśmy podsumowanie tego rozdziału.

## 10.1. Przegląd technik i zagadnień z obszaru programowania baz danych

W tym podrozdziale skupimy się na technikach opracowanych z myślą o możliwości uzyskiwania dostępu do baz danych z poziomu innych programów oraz, w szczególności, na konkretnych metodach wywoływania poleceń języka SQL w kodzie tych aplikacji. W prezentacji języka SQL w rozdziałach 6. i 7. skupialiśmy się na konstrukcjach językowych wykorzystywanych do wykonywania rozmaitych operacji na bazie danych — począwszy od definiowania schematu i ograniczeń, a skończywszy na budowaniu zapytań, operacji aktualizujących oraz tworzeniu perspektyw. Większość systemów baz danych oferuje specjalne **interaktywne interfejsy**, za pośrednictwem których można wykonywać polecenia języka SQL wpisywane bezpośrednio przez użytkowników tego typu systemów. Przykładowo, polecenie SQLPLUS w systemie komputerowym z zainstalowanym relacyjnym systemem zarzą-

dzania bazą danych firmy Oracle powoduje uruchomienie takiego interaktywnego interfejsu. Użytkownik tego interfejsu może wpisywać polecenia i zapytania języka SQL bezpośrednio w oknie programu — wystarczy, że po wpisaniu niezbędnej liczby wierszy zakończy kod języka SQL średnikiem i naciśnie klawisz *Enter* ("`<cr>`"). Alternatywnym rozwiązaniem jest stworzenie i wykonanie za pośrednictwem interaktywnego interfejsu całego **pliku poleceń** — w tym celu należy wpisać polecenie `@<nazwa pliku>`. System bazy danych wykona wszystkie polecenia zapisane we wskazanym pliku i wyświetli ewentualne wyniki tych operacji.

Interaktywny interfejs jest dosyć wygodnym narzędziem do tworzenia schematów i ograniczeń, a także do okazjonalnego wykonywania zapytań tworzonych *ad hoc*. W praktyce jednak większość interaktywnych działań na bazie danych odbywa się za pośrednictwem precyzyjnie zaprojektowanych i przetestowanych programów. Programy te są nazywane **programami aplikacji** lub **aplikacjami baz danych** — z punktu widzenia użytkownika końcowego pełnią one rolę tzw. *zapuszkowanych transakcji* (patrz punkt 1.4.3). Innym bardzo popularnym sposobem wykorzystywania oprogramowania bazy danych jest stosowanie programów aplikacji opartych na **interfejsie WWW**; przykładowo, taki interfejs może być wykorzystany do stworzenia systemu rezerwacji biletów lotniczych lub sklepu internetowego. W praktyce większość aplikacji handlu elektronicznego wykorzystuje właśnie tego typu mechanizmy przekazywania poleceń do baz danych. W rozdziale 11. znajdziesz przegląd programowania internetowych baz danych z użyciem PHP — popularnego ostatnio języka skryptowego.

W tym podrozdziale w pierwszej kolejności przedstawimy przegląd głównych strategii w zakresie programowania baz danych. Następnie omówimy niektóre problemy związane z dostępem do baz danych w kodzie uniwersalnych języków programowania. Na końcu przedstawimy typową sekwencję poleceń stanowiących interakcję bazy danych i tak opracowanego oprogramowania.

### 10.1.1. Strategie programowania baz danych

Istnieje wiele technik w zakresie dołączania do programów aplikacji elementów interakcji z bazami danych. Do najważniejszych ze stosowanych w tym celu strategii należą:

#### (1) **Osadzanie poleceń bazy danych w uniwersalnym języku programowania.**

Ta strategia przewiduje **osadzanie** wyrażeń języka bazy danych w kodzie nadrzędnego języka programowania — takie wyrażenia są jednak oznaczane specjalnym prefiksem. Przykładowo, takim prefiksem dla osadzonych poleceń języka SQL może być ciąg znaków `EXEC SQL`, który poprzedza wszystkie polecenia języka SQL występujące w kodzie nadrzędnego języka programowania<sup>1</sup>. **Prekompilator** lub **preprocesor** najpierw skanuje kod źródłowy programu, aby już na tym etapie zidentyfikować wszystkie polecenia skierowane do bazy danych i przekazać je do przetworzenia przez system zarządzania bazą danych. Takie polecenia są w kodzie programu zastępowane odpowiednimi wywołaniami funkcji udostępnianych w kodzie wygenerowanym przez system zarządzania bazą danych. Ta technika jest zwykle nazywana **osadzonym kodem języka SQL**.

---

<sup>1</sup> Niekiedy stosuje się także inne prefiksy, jednak właśnie prefiks `EXEC SQL` jest wykorzystywany najczęściej.

- (2) **Stosowanie biblioteki funkcji lub klas bazy danych. Biblioteka funkcji** jest udostępniana programistom nadrzędnego języka programowania z myślą o wywołaniach operacji na bazach danych. Przykładowo, taka biblioteka może zawierać funkcje umożliwiające nawiązywanie połączeń z bazą danych, przygotowywanie zapytań, wykonywanie zapytań, aktualizowanie danych, przetwarzanie w pętli wyników zapytania rekord po rekordzie itp. Rzeczywiste zapytania i polecenia aktualizacji bazy danych (a także wszystkie inne potrzebne informacje) są przekazywane w postaci parametrów wywołań tych funkcji. Ta strategia programowania baz danych opiera się na czymś, co jest często nazywane **interfejsem programowym aplikacji** (ang. *Application Programming Interface*, w skrócie **API**), za pośrednictwem którego można uzyskiwać dostęp do baz danych z poziomu programów aplikacji. W **obiektowych językach programowania** używane są **biblioteki klas**. Przykładowo, w języku Java dostępna jest biblioteka klas JDBC, która umożliwia generowanie różnych typów obiektów, takich jak obiekty połączeń z konkretną bazą danych, obiekty zapytań i obiekty wyników zapytań. Obiekty każdego typu udostępniają zestaw operacji powiązanych z klasą danego obiektu.
- (3) **Projektowanie zupełnie nowych języków. Język programowania bazy danych** jest projektowany od podstaw w taki sposób, aby zapewnić zgodność zarówno z modelem bazy danych, jak i ze stosowanym w niej językiem zapytań. Do języka bazy danych dodawane są dodatkowe struktury programowania, takie jak pętle czy instrukcje warunkowe, dzięki czemu język ten staje się kompletnym językiem programowania. Przykładem zastosowania tego podejścia jest język PL/SQL z baz Oracle. Standard SQL obejmuje język SQL/PSM przeznaczony do tworzenia procedur składowanych.

W praktyce dwa pierwsze podejścia są bardziej popularne, ponieważ często okazuje się, że wiele już napisanych (w uniwersalnych językach programowania) aplikacji wymaga dostępu do bazy danych. Zastosowanie trzeciej strategii jest uzasadnione w przypadku aplikacji, których działanie wymaga intensywnej współpracy z bazą danych. Jednym z głównych problemów, z którym borykają się użytkownicy stosujący pierwszą strategię, jest tzw. *niezgodność impedancji*, która nie występuje w rozwiązaniach opartych na trzeciej strategii.

### 10.1.2. Niezgodność impedancji

**Niezgodność impedancji** jest terminem wykorzystywanym do określania problemów wynikających z różnic pomiędzy modelem bazy danych a modelem języka programowania. Przykładowo, praktyczny model relacyjny obejmuje trzy główne konstrukcje: kolumny (atrybuty) i ich typy danych, wiersze (inaczej krotki lub rekordy) oraz tabele (zbiory lub wielozbiory rekordów). Pierwszym potencjalnym problemem jest fakt, iż *typy danych wykorzystywane w językach programowania* różnią się od *typów danych atrybutów* w relacyjnym modelu danych. Oznacza to, że dla każdego języka programowania istnieje konieczność stworzenia **powiązań** pomiędzy jego typami danych a odpowiednimi typami atrybutów modelu bazy danych. Opracowanie tych powiązań jest konieczne dla *każdego języka programowania* z osobna, ponieważ każdy z tych języków wykorzystuje inne typy danych; przykładowo, typy danych dostępne w językach C, C++ i Java różnią się nie tylko od siebie,

ale także od typów danych stosowanych w języku SQL (które są standardowymi typami dla relacyjnych baz danych).

Kolejny problem wynika z faktu, że rezultatami większości zapytań są zbiory lub wielozbiory krotek (wierszy), a każda krotka ma postać sekwencji wartości atrybutów. Programista często staje przed koniecznością uzyskania dostępu do konkretnych wartości danych przechowywanych w pojedynczych krotkach (np. w celu ich wyświetlenia lub dalszego przetworzenia). Oznacza to, że niezbędne jest stworzenie powiązań, które umożliwią odwzorowywanie *struktury danych wyniku zapytania* (mającej zwykle postać tabeli) w odpowiednią strukturę danych języka programowania. Potrzebny jest także mechanizm przeglądania w pętli kolejnych krotek **wyniku zapytania**, dzięki czemu będzie mógł kolejno uzyskiwać dostęp do pojedynczych krotek i odczytywać ich wartości. Pobrane wartości atrybutów są zwykle kopiowane do odpowiednich zmiennych programu na potrzeby dalszego przetwarzania danych przez program. Do przeglądania krotek wyniku zapytania w pętli wykorzystuje się tzw. **kursor** lub **zmienną iteratora**. Pojedyncze wartości wewnątrz każdej z krotek są zwykle zapisywane w osobnych zmiennych należących do właściwego typu.

Niezgodność impedancji nie stanowi większego problemu w sytuacjach, gdy wykorzystuje się specjalny język programowania baz danych, który został zaprojektowany w sposób umożliwiający stosowanie tego samego modelu danych i typów danych, na których opiera się także model bazy danych. Przykładem takiego języka jest PL/SQL firmy Oracle. Standard SQL obejmuje też proponowany język programowania baz danych — *SQL/PSM*. W przypadku obiektowych baz danych obiektowy model danych (patrz rozdział 12.) jest dosyć podobny do modelu danych stosowanego w języku programowania Java, zatem znaczenie problemu niezgodności impedancji jest bardzo ograniczone w rozwiązaniach opartych na języku Java (występującym w roli nadrzędnego języka programowania) i zgodnej z tym językiem obiektowej bazy danych. Kilka języków programowania baz danych zostało zaimplementowanych i osiągnęło status prototypów badawczych (patrz notka bibliograficzna).

### 10.1.3. Typowa sekwencja operacji składających się na interakcję w programowaniu baz danych

Kiedy programista lub inżynier oprogramowania pisze program, który wymaga dostępu do bazy danych, dosyć często okazuje się, że taki program będzie działał w jednym systemie komputerowym, podczas gdy potrzebna baza danych będzie zainstalowana w innym systemie. W podrozdziale 2.5 wspominaliśmy, że popularną architekturą dla dostępu do bazy danych jest trójwarstwowy model klient-serwer, gdzie działający w górnej warstwie **program klienta** odpowiada za wyświetlanie informacji w laptopie lub urządzeniu mobilnym (zwykle w kliencie internetowym albo aplikacji mobilnej), natomiast pracujący w środkowej warstwie **program aplikacji** obsługuje logikę aplikacji biznesowej, a także kieruje wywołania do jednego lub większej liczby **serwerów baz danych** z dolnej warstwy, dzięki czemu może odczytywać lub aktualizować potrzebne dane<sup>2</sup>. Podczas pisania takiego pro-

---

<sup>2</sup> Zgodnie z naszymi ustaleniami z podrozdziału 2.5, istnieją architektury dwuwarstwowe i trójwarstwowe; dla uproszczenia, w tym rozdziale przyjmujemy, że mamy do czynienia z architekturą dwuwarstwową klient-serwer.

gramu jego autor często wprowadza do kodu następującą sekwencję operacji składających się na interakcję z bazą danych:

- (1) Kiedy program klienta wymaga dostępu do konkretnej bazy danych, działające po stronie klienta oprogramowanie w pierwszej kolejności musi *nawiązać* lub *otworzyć połączenie* z odpowiednim serwerem bazy danych. Ta operacja zazwyczaj polega na określeniu adresu internetowego (często w postaci URL) komputera, na którym pracuje serwer bazy danych, oraz przekazania nazwy konta i hasła niezbędnego do uzyskania dostępu do zasobów tego serwera.
- (2) Kiedy połączenie zostanie już nawiązane, program może współpracować z bazą danych przez wysyłanie do niej zapytań, żądań aktualizacji i innych poleceń realizowanych przez działający na serwerze system zarządzania bazą danych. W programie aplikacji można w zasadzie umieszczać większość rodzajów poleceń języka SQL.
- (3) Kiedy zostaną wykonane wszystkie operacje wymagające dostępu do bazy danych i program nie będzie tego dostępu dłużej potrzebował, powinien *przerwać* lub *zamknąć* połączenie z serwerem bazy danych.

Pojedynczy program może — w razie potrzeby — uzyskiwać dostęp do wielu baz danych. Niektóre strategie programowania baz danych przewidują co prawda możliwość jednoczesnego utrzymywania tylko jednego połączenia z bazą danych, ale nie brakuje także rozwiązań pozwalających na nawiązywanie w tym samym czasie wielu takich połączeń.

W trzech kolejnych podrozdziałach omówimy przykłady dla każdej z wymienionych strategii programowania baz danych. W podrozdziale 10.2 opiszemy sposób *osadzania* poleceń języka SQL w uniwersalnym języku programowania, w podrozdziale 10.3 omówimy metodę wykorzystywania *wywołań funkcji* lub *bibliotek klas* w procesie uzyskiwania dostępu do baz danych za pomocą technologii SQL/CLI (podobnej do ODBC) i JDBC, natomiast podrozdział 10.4 poświęcimy rozszerzeniu języka SQL nazywanemu SQL/PSM, które umożliwia definiowanie i składowanie w systemie bazy danych modułów (procedur i funkcji) *uniwersalnych języków programowania*<sup>3</sup>. W podrozdziale 10.5 znajdziesz porównanie opisanych strategii.

## 10.2. Osadzony język SQL, dynamiczny język SQL oraz język SQLJ

W tym podrozdziale przedstawimy przegląd metod osadzania poleceń języka SQL w kodzie uniwersalnych języków programowania. Skupimy się tu na dwóch językach: C i Javie. Zaprezentowane w punktach od 10.2.1 do 10.2.3 przykłady używające języka C, z **osadzonym językiem SQL**, można dostosować do innych podobnych języków programowania. Przykłady oparte na Javie (technologia **SQLJ**) pokażemy w punktach 10.2.4 i 10.2.5. Język programowania, w którym osadzamy polecenia, jest nazywany **językiem nadrzędnym**. Większość

---

<sup>3</sup> SQL/PSM pozwala dobrze zilustrować metody wstawiania do języka SQL typowych konstrukcji uniwersalnych języków programowania, a więc np. pętli i instrukcji warunkowych.

poleceń języka SQL — włącznie z definicjami danych i ograniczeń, zapytaniami, aktualizacjami oraz definicjami perspektyw — może być osadzanych w kodzie programu napisanego w nadrzędnym języku programowania.

### 10.2.1. Wyszukiwanie pojedynczych krotek za pomocą poleceń osadzonego języka SQL

Aby dobrze zilustrować technikę osadzania poleceń języka SQL, w roli nadrzędnego języka programowania wykorzystamy język C<sup>4</sup>. W C osadzony kod języka SQL jest wyróżniany wśród wyrażenń nadrzędnego języka programowania za pomocą prefiksów w postaci słów kluczowych EXEC SQL — takie rozwiązanie umożliwia **preprocesorowi** (lub **prekompilatorowi**) oddzielanie osadzonych poleceń języka SQL od kodu języka nadrzędnego. Koniec poleceń języka SQL można oznaczać za pomocą średników lub odpowiadających sobie słów kluczowych END-EXEC. Podobne reguły dotyczą osadzania języka SQL w innych językach programowania.

Wewnątrz osadzonego polecenia języka SQL możemy się odwoływać do specjalnie zadeklarowanych w języku C zmiennych danego programu. Są to tzw. **zmiennie dzielone**, ponieważ wykorzystuje się je zarówno we właściwym programie napisanym w języku C, jak i w osadzonych poleceniach języka SQL. *Występujące w poleceniach języka SQL* nazwy zmiennych dzielonych poprzedza się znakiem dwukropka. Takie rozwiązanie umożliwia odróżnianie tych zmiennych od nazw przypisanych do takich konstrukcji schematu bazy danych jak atrybuty (nazwy kolumn) czy relacje (nazwy tabel). Dzięki stosowaniu dodatkowych prefiksów zmienne programu mogą mieć takie same nazwy jak atrybuty bazy danych, ponieważ wewnątrz poleceń języka SQL i tak będą poprzedzane symbolem dwukropka. Nazwy konstrukcji schematu bazy danych (takich jak atrybuty i relacje) mogą być wykorzystywane oczywiście tylko wewnątrz poleceń języka SQL, natomiast zmienne dzielone programu mogą występować w kodzie języka C bez prefiksu.

Przypuśćmy, że chcemy napisać w języku C programy przetwarzające bazę danych FIRMA z rysunku 5.5. Musimy zadeklarować zmienne programów, które będą odpowiadały typom atrybutów tej bazy danych, których zawartość będzie przetwarzana w naszych programach. Programista może oczywiście wybrać dowolne nazwy dla **zmiennych programu** — nazwy te mogą, ale nie muszą być identyczne jak nazwy odpowiadających im atrybutów bazy danych. We wszystkich naszych przykładach będziemy wykorzystywali zmienne zadeklarowane w kodzie zawartym na rysunku 10.1 — prezentowane w dalszej części tego rozdziału fragmenty kodu źródłowego napisanego w języku C będą *pozbawione deklaracji zmiennych*. Zmienne dzielone zostały zadeklarowane w **bloku deklaracji** naszego programu (patrz wiersze od 1. do 7. rysunku 10.1)<sup>5</sup>. W analizowanym programie wykorzystujemy następujące odwzorowanie typów danych języka programowania C w typy danych języka SQL: typy danych long, short, float i double języka C odpowiadają odpowiednio typom INTEGER, SMALLINT, REAL i DOUBLE języka SQL. Stosowane

<sup>4</sup> Omówienie to dotyczy także języków programowania C++ i C#, ponieważ nie używamy tu żadnych mechanizmów obiektowych, a koncentrujemy się na mechanizmach programowania baz danych.

<sup>5</sup> Dla uproszczenia występujących w tekście odwołań, umieściliśmy w kodzie numerację wierszy (numery te nie są oczywiście częścią rzeczywistego kodu).



w języku SQL ciągi znaków o stałej i zmiennej długości (reprezentowane odpowiednio przez typy `CHAR[i]` oraz `VARCHAR[i]`) mogą być odwzorowane w tablice znaków (odpowiednio `char[i+1]` oraz `varchar[i+1]`) języka C, które zawsze są o jeden znak dłuższe niż ich odpowiedniki w języku SQL, ponieważ są zakończone znakiem pustym ("`\0`"), nie będącym częścią samego ciągu<sup>6</sup>. Choć `varchar` nie jest standardowym typem danych języka C, można go stosować, gdy C służy do programowania baz danych z użyciem języka SQL.

```
0)  int loop;
1)  EXEC SQL BEGIN DECLARE SECTION;
2)  varchar nazwadz[16], imię[16], nazwisko[16], adres[31];
3)  char pesel[10], dataur[11], płeć[2], inicjałdrimienia[2];
4)  float pensja, podwyżka;
5)  int nrdz, numerdz;
6)  int SQLCODE; char SQLSTATE[6];
7)  EXEC SQL END DECLARE SECTION;
```

RYSUNEK 10.1. Zmienne programu C wykorzystywane w przykładach zagnieżdżonego kodu języka SQL: P1 i P2

Łatwo zauważyć, że na rysunku 10.1 jedynymi osadzonymi poleceniami języka SQL są wyrażenia zawarte w wierszach 1. i 7., które nakazują prekompilatorowi zwrócić uwagi na nazwy zmiennych języka C zadeklarowanych pomiędzy słowami kluczowymi `BEGIN DECLARE` a `END DECLARE`, ponieważ zmienne te mogą być wykorzystywane w osadzonych poleceniach języka SQL (oczywiście pod warunkiem zastosowania wymaganego prefiksu w postaci dwukropka). Wiersze od 2. do 5. zawierają standardowe deklaracje programu napisanego w języku C. Zadeklarowane w tych wierszach zmienne programu odpowiadają atrybutom tabel `PRACOWNIK` i `DZIAŁ` należących do bazy danych `FIRMA` (przedstawionej na rysunku 5.5 i zadeklarowanej za pomocą odpowiednich poleceń języka SQL DDL na rysunku 6.1). Zmienne zadeklarowane w wierszu 6. (`SQLCODE` i `SQLSTATE`) to **zmienne komunikacji języka SQL**, które będą tu wykorzystywane do przekazywania pomiędzy systemem bazy danych a programem informacji o błędach i wyjątkach. Wiersz 0. zawiera deklarację zmiennej programu nazwanej `loop`, która nie będzie wykorzystywana w żadnym z osadzonych poleceń języka SQL, zatem może zostać zadeklarowana poza blokiem deklaracji języka SQL.

**Łączenie z bazą danych.** Polecenie języka SQL wykorzystywane do nawiązywania połączeń z bazą danych ma następującą postać:

```
CONNECT TO <nazwa serwera> AS <nazwa połączenia>
AUTHORIZATION <nazwa konta użytkownika i hasło>;
```

Ponieważ użytkownik lub program może jednocześnie korzystać z dostępu do wielu serwerów baz danych, zasadniczo możliwe jest nawiązywanie wielu tego typu połączeń, ale w tym samym czasie aktywne może być tylko jedno połączenie. Programista lub użytkownik może wykorzystywać *<nazwę połączenia>* do zmiany aktualnie aktywnego połączenia na inne — służy do tego następujące polecenie języka SQL:

```
SET CONNECTION <nazwa połączenia>;
```

---

<sup>6</sup> Ciągi znaków języka SQL można także odwzorować do postaci typu `char*` języka C.



Kiedy nawiązane wcześniej połączenie nie będzie już potrzebne, można je w języku SQL przerwać za pomocą następującego polecenia:

```
DISCONNECT <nazwa połączenia>;
```

W przykładzie, który wykorzystujemy w tym rozdziale, zakładamy, że zostało już nawiązane odpowiednie połączenie z bazą danych FIRMA oraz że jest to aktualnie jedyne aktywne połączenie.

**Komunikacja pomiędzy programem a systemem zarządzania bazą danych z wykorzystaniem zmiennych SQLCODE i SQLSTATE.** Dwie specjalnymi zmiennymi komunikacji, które są wykorzystywane przez system zarządzania bazą danych do sygnalizowania programowi klienta wykrytych wyjątków lub błędów, są SQLCODE i SQLSTATE. SQLCODE (patrz rysunek 10.1) jest zmienną całkowitoliczbową. Po wykonaniu każdego polecenia wywołanego w bazie danych, system zarządzania bazą danych zwraca odpowiedni kod zakończenia polecenia właśnie w zmiennej SQLCODE. Wartość 0 oznacza, że polecenie zostało wykonane przez system zarządzania bazą danych pomyślnie. Jeśli wartość zmiennej SQLCODE jest większa od zera (lub, mówiąc bardziej precyzyjnie, jest równa 100), oznacza to, że wynik zapytania nie zawiera żadnych dodatkowych danych (rekordów). Ujemna wartość zmiennej SQLCODE oznacza wystąpienie błędu. W niektórych systemach zarządzania bazami danych (np. w systemach firmy Oracle) zmienna SQLCODE jest jednym z pól struktury rekordu nazywanej w skrócie SQLCA (ang. *SQL Communication Area*, czyli obszar komunikacji języka SQL), zatem odwołania do wartości tej zmiennej powinny mieć postać: SQLCA.SQLCODE. W takim przypadku konieczne jest umieszczenie w kodzie programu napisanego w języku programowania C specjalnej definicji struktury SQLCA przez dodanie następującego wiersza:

```
EXEC SQL include SQLCA;
```

W późniejszych wersjach standardu SQL dodano nową zmienną komunikacji nazwaną SQLSTATE, która jest ciągiem pięciu znaków. Wartość "00000" w zmiennej SQLSTATE oznacza, że nie wystąpił żaden błąd ani wyjątek; wszystkie pozostałe wartości oznaczają wystąpienie błędu lub wyjątku. Przykładowo, wartość "02000" w zmiennej SQLSTATE oznacza „brak dodatkowych danych”. Zarówno zmienna SQLSTATE, jak i zmienna SQLCODE jest obecnie dostępna w standardzie SQL. Wiele kodów błędów i wyjątków zwracanych w zmiennej SQLSTATE ma w założeniu być zgodnych z jednym standardem<sup>7</sup> (niezależnie od producentów i platform), natomiast kody zwracane w postaci wartości zmiennej SQLCODE mogą być dowolnie definiowane przez producentów systemów zarządzania bazami danych. Oznacza to, że w zasadzie lepszym rozwiązaniem jest wykorzystywanie zmiennej SQLSTATE, ponieważ w ten sposób można zapewnić niezależność stosowanych w programach aplikacji mechanizmów obsługi błędów od konkretnych systemów zarządzania bazami danych. W formie ćwiczenia Czytelnik powinien ponownie opracować wszystkie przykłady prezentowane w dalszej części tego rozdziału w taki sposób, aby wykorzystywały zmienną SQLSTATE zamiast SQLCODE.

---

<sup>7</sup> W szczególności, ustandaryzowane powinny być kody zwracane w zmiennej SQLSTATE rozpoczynające się od cyfr 0 – 4 oraz od liter A – H, znaczenia wszystkich pozostałych wartości mogą być uzależnione od poszczególnych implementacji standardu SQL.

**Przykład programowania z osadzeniem kodu języka SQL.** Naszym pierwszym przykładem ilustrującym programowanie z osadzaniem kodu języka SQL będzie powtarzalny blok programu (pętla), który pobiera od użytkownika numer PESEL pracownika i wyświetla pewne informacje na temat tego pracownika zawarte w bazie danych (w odpowiedniej krotce relacji PRACOWNIK). Kod omawianego programu przedstawiono w postaci fragmentu F1 na rysunku 10.2. Program odczytuje na wejściu wartość numeru PESEL i odnajduje (za pomocą osadzonego polecenia języka SQL) w bazie danych krotkę relacji PRACOWNIK z zawartością atrybutu PESEL równą tej wartości. **Klauzula INTO** (wiersz 5.) określa zmienne programu, w których zostaną umieszczone znalezione w bazie danych wartości atrybutów. Wymienione w klauzuli INTO zmienne programu napisanego w języku C są — zgodnie z naszymi wcześniejszymi uwagami — oznaczane prefiksem (znakiem dwukropka). Klauzulę INTO można stosować w ten sposób tylko wtedy, gdy wynikiem zapytania jest *jeden rekord*. Jeśli pobieranych jest wiele rekordów, taki kod spowoduje błąd. Obsługa wielu rekordów jest przedstawiona w punkcie 10.2.2.

*// Fragment F1 programu*

```
0) loop = 1;
1) while (loop) {
2)     prompt("Wpisz numer PESEL: ", pesel);
3)     EXEC SQL
4)         select IMIĘ, INICJAŁDRIMIENIA, NAZWISKO, ADRES, PENSJA
5)         into :imię, :inicjałdrimienia, :nazwisko, :adres, :pensja
6)         FROM PRACOWNIK WHERE PESEL = :pesel;
7)     if (SQLCODE == 0) printf(imię, inicjałdrimienia, nazwisko, adres, pensja)
8)     else printf("Podany numer PESEL nie istnieje: ", pesel);
9)     prompt("Czy chcesz sprawdzić więcej numerów PESEL (1 - TAK, 0 - NIE): ",
10)         loop);
10) }
```

RYSUNEK 10.2. Fragment F1 programu napisanego w języku C z osadzonymi poleceniami języka SQL

Wiersz 7. fragmentu F1 ilustruje komunikację pomiędzy bazą danych a programem — wykorzystujemy w tym celu wspomnianą już specjalną zmienną SQLCODE. Zwrócenie przez system zarządzania bazą danych zmiennej SQLCODE z wartością równą 0 oznacza, że podczas wykonywania ostatniego polecenia nie wykryto błędów ani wyjątków. Instrukcja warunkowa w wierszu 7. sprawdza ten warunek i zakłada, że ewentualny błąd może wynikać z braku krotki relacji PRACOWNIK zawierającej podaną wartość w atrybucie PESEL — efektem tego sprawdzenia jest więc odpowiedni komunikat (wiersz 8.).

Zdefiniowane we fragmencie F1 polecenie języka SQL zwraca *pojedynczą krotkę*; dzięki temu w klauzuli INTO w wierszu 5. listingu możemy przypisywać wartości jej atrybutów bezpośrednio do zmiennych programu napisanego w języku C. Generalnie, zapytanie języka SQL może wyszukiwać wiele krotek. W takim przypadku program stworzony w języku programowania C będzie zwykle przeszukiwał otrzymany zbiór krotek i przetwarzał te krotki po jednej w pętli. Do przetwarzania krotek w ten sposób wykorzystuje się tzw. *kursor* w nadrzędnym języku programowania. Kursory omówimy w kolejnym punkcie.

## 10.2.2. Przetwarzanie wyników zapytań za pomocą kursorów

**Kursor** możemy traktować jako wskaźnik do *pojedynczej krotki* (*pojedynczego wiersza*) pochodzącej z **wyniku zapytania**, który składa się z wielu krotek. Kursor służy do przetwarzania wyników w pętli rekord po rekordzie. Kursor jest deklarowany w momencie, kiedy w programie głównym **deklarowane** jest polecenie zapytania języka SQL. W dalszej części programu polecenie `OPEN CURSOR` wczytuje z bazy danych wynik zapytania i ustawia kursor na pozycji *przed pierwszym wierszem* w tabeli zwróconej przez zapytanie. Wiersz ten staje się dla nowo otwartego kursora **wierszem bieżącym**. W programie jest następnie wykonywane polecenie `FETCH`, które (za każdym razem) przesuwa kursor do *następnego wiersza* w tabeli będącej wynikiem zapytania — następny wiersz staje się, tym samym, wierszem bieżącym kursora, a wartości jego atrybutów są kopiowane do zmiennych (zadeklarowanych w programie napisanym w nadrzędnym języku programowania C) określonych w klauzuli `INTO` polecenia `FETCH`. Zmienna kursora jest po prostu **iteratorem**, który umożliwia przeglądanie (w pętli) kolejnych krotek z wyniku zapytania — po jednej krotce w każdej iteracji.

Aby określić, w którym momencie wszystkie krotki w wyniku zapytania zostały już przetworzone, można sprawdzić wartość zmiennej komunikacji `SQLCODE` (lub `SQLSTATE`). Jeśli wykonano polecenie `FETCH`, w którego rezultacie kursor został przeniesiony za ostatnią krotkę tabeli wynikowej zapytania, w zmiennej całkowitoliczbowej `SQLCODE` zostanie zwrócona wartość dodatnia oznaczająca, że nie znaleziono żadnych dodatkowych danych (krotek); w takim przypadku w zmiennej `SQLSTATE` otrzymalibyśmy wartość "02000". Programista może wykorzystać fakt otrzymania takiej wartości do zakończenia pętli iteracyjnie przeszukującej wynik zapytania. Zasadniczo, jednocześnie można wykorzystywać wiele otwartych kursorów. Aby określić, że przetwarzanie tabeli wynikowej zapytania jest zakończone i zamknąć związany z tą tabelą kursor, należy wykonać polecenie `CLOSE CURSOR`.

Przykładem zastosowania kursorów jest zaprezentowany na rysunku 10.3 fragment kodu, w którym (w wierszu 4.) zadeklarowano kursor nazwany `PRAC`. Kursor `PRAC` jest powiązany z zapytaniem języka SQL zadeklarowanym w wierszach 5. i 6., przy czym zapytanie jest wykonywane dopiero w ramach przetwarzania polecenia `OPEN PRAC` (wiersz 8.). Polecenie `OPEN <nazwa kursora>` wykonuje zapytanie i pobiera jego wynik jako tabelę do przestrzeni roboczej programu, gdzie program może w pętli przetwarzać poszczególne wiersze (krotki) w dalszych instrukcjach `FETCH <nazwa kursora>` (wiersz 9.). Zakładamy, że odpowiednie zmienne programu napisanego w języku C zostały już zadeklarowane (tak jak na rysunku 10.1). Fragment programu oznaczony symbolem `F2` odczytuje na wejściu nazwę działu (w wierszu 0.), wyszukuje numer odpowiedniego działu (wiersze od 1. do 3.) i wykorzystuje kursor `PRAC` do znalezienia w bazie danych informacji o wszystkich pracownikach, którzy są zatrudnieni w danym dziale. Następnie (w pętli zawartej w wierszach od 10. do 18.) przeglądamy kolejno rekordy pracowników (po jednym w każdej iteracji) oraz wyświetlamy imiona i nazwiska. W wierszu 12. program odczytuje na wejściu wysokość podwyżki, po czym (w wierszach od 14. do 16.) aktualizuje w bazie danych pensję pracownika.

// Fragment F2 programu

```
0)  prompt("Wpisz nazwę działu: ", nazwadz);
1)  EXEC SQL
2)      select NUMERDZ into :numerdz
3)      FROM DZIAŁ WHERE NAZWADZ = :nazwadz;
```

```

4) EXEC SQL DECLARE PRAC CURSOR FOR
5)   SELECT PESEL, IMIĘ, INICJAŁDRIMIENTA, NAZWISKO, PENSJA
6)   FROM PRACOWNIK WHERE NRDZ = :numerdz;
7)   FOR UPDATE OF PENSJA;
8) EXEC SQL OPEN PRAC;
9) EXEC SQL FETCH from PRAC into :pesel, :imię, :inicjałdrimienta, :nazwisko,
   :pensja;
10) while (SQLCODE == 0) {
11)   printf("Nazwisko pracownika: ", imię, inicjałdrimienta, nazwisko);
12)   prompt("Wpisz wysokość podwyżki: ", podwyżka);
13)   EXEC SQL
14)     UPDATE PRACOWNIK
15)     SET PENSJA = PENSJA + :podwyżka
16)     WHERE CURRENT OF PRAC;
17)   EXEC SQL FETCH from PRAC into :pesel, :imię, :inicjałdrimienta,
     :nazwisko, :pensja;
18) }
19) EXEC SQL CLOSE PRAC;

```

RYSunek 10.3. Fragment F2 programu napisanego w języku C wykorzystującego kursory i osadzone polecenia języka SQL aktualizujące informacje zapisane w bazie danych

Ten przykład pokazuje też, w jaki sposób programista może *aktualizować* rekordy bazy danych. Podczas definiowania kursora dla wierszy przeznaczonych do zmodyfikowania (**aktualizacji**), musimy w deklaracji kursora dodać klauzulę `FOR UPDATE OF` i wymienić listę nazw wszystkich atrybutów, które będą aktualizowane przez nasz program. Przykładem zastosowania tej klauzuli jest wiersz 7. przedstawionego powyżej fragmentu kodu. Jeśli stworzymy kursor z zamiarem **usuwania** wierszy, musimy dodać do jego deklaracji słowa kluczowe `FOR UPDATE` bez określania żadnych atrybutów. Wykorzystywany w osadzonych poleceniach `UPDATE` (lub `DELETE`) warunek `WHERE CURRENT OF <nazwa kursora>` określa, że operacja aktualizacji (lub usuwania) ma dotyczyć bieżącej krotki wskazywanej przez dany kursor (patrz wiersz 16. fragmentu F2).

Nie ma potrzeby dołączania wspomnianej już klauzuli `FOR UPDATE OF` (jak w wierszu 7. fragmentu F2), jeśli wynik zapytania będzie wykorzystywany wyłącznie do *wyszukiwania potrzebnych informacji* (nie do aktualizacji ani usuwania).

**Ogólne opcje używane do deklarowania kursora.** Podczas deklarowania kursorów można stosować wiele ciekawych opcji. Ogólna postać deklaracji kursora jest następująca:

```

DECLARE <nazwa kursora> [INSENSITIVE] [SCROLL] CURSOR
[WITH HOLD] FOR <specyfikacja zapytania>
[ORDER BY <specyfikacja uporządkowania>]
[FOR READ ONLY | FOR UPDATE [OF <lista atrybutów>]];

```

Omówiliśmy już w skrócie opcje wymienione w ostatnim wierszu powyższej deklaracji. Domyślnym rozwiązaniem jest stosowanie zapytań mających na celu jedynie wyszukiwanie danych (opcja `FOR READ ONLY`). Jeśli któraś z krotek zwróconych w tabeli wynikowej zapytania ma być przedmiotem aktualizacji, musimy użyć klauzuli `FOR UPDATE OF <lista atrybutów>` i wymienić wszystkie atrybuty, które mogą być aktualizowane. Jeśli natomiast któraś z tych krotek będzie usuwana, musimy użyć samych słów kluczowych `FOR UPDATE` (bez wymieniania jakichkolwiek nazw atrybutów).

Jeśli w deklaracji kursora użyjemy opcjonalnego słowa kluczowego `SCROLL`, będziemy mieli możliwość pozycjonowania kursora na różne sposoby, nie tylko w oparciu o domyślną technikę dostępu sekwencyjnego. Do polecenia `FETCH` można wówczas dodać specyfikację tzw. **orientacji przeglądania**, która może mieć postać jednego z następujących słów kluczowych: `NEXT`, `PRIOR`, `FIRST`, `LAST`, `ABSOLUTE i` oraz `RELATIVE i`. W przypadku dwóch ostatnich poleceń wartość `i` musi być liczbą całkowitą określającą odpowiednio bezwzględną pozycję krotki lub pozycję krotki wyznaczoną względem bieżącego położenia kursora. Domyślną orientacją przeglądania, którą będziemy stosowali w prezentowanych przykładach, jest orientacja `NEXT`. Dzięki dostępnej opcji określania orientacji przeglądania, programista ma możliwość bardziej elastycznego przenoszenia kursora pomiędzy krotkami należącymi do tabeli wynikowej zapytania — może się odwoływać do dowolnej wybranej przez siebie krotki lub przeglądać wynikowy zbiór krotek np. od końca. Zastosowanie słowa kluczowego `SCROLL` w deklaracji kursora powoduje, że zmienia się ogólna postać polecenia `FETCH` (elementy umieszczone w nawiasach kwadratowych są opcjonalne):

```
FETCH [[<orientacja przeglądania>] FROM] <nazwa kursora>  
      INTO <docelowa lista wypełniania>;
```

Klauzula `ORDER BY` umożliwia określanie uporządkowania krotek — w ten sposób można określić kolejność odczytywania wierszy przez polecenie `FETCH`. Klauzula ta ma podobną postać jak odpowiednia klauzula stosowana w zapytaniach języka SQL (patrz punkt 6.3.6). Ostatnie dwie opcje polecenia deklarującego kursor (`INSENSITIVE` i `WITH HOLD`) są związane z właściwościami transakcji realizowanych przez programy bazy danych (patrz rozdział 20.).

### 10.2.3. Określanie zapytań w czasie wykonywania programu — stosowanie dynamicznego języka SQL

W prezentowanych do tej pory przykładach osadzane zapytania języka SQL były zapisywane w postaci odpowiednio oznaczonych fragmentów kodu źródłowego nadrzędnego języka programowania. Oznacza to, że w momencie, w którym uznamy, że konieczne jest wykonanie innego zapytania, będziemy musieli napisać nowy program i wykonać wszystkie związane z tym kroki (kompilacja, diagnostyka, testowanie itp.). W niektórych przypadkach bardziej wygodnym rozwiązaniem jest więc stworzenie programu, który będzie mógł *dynamicznie*, w czasie działania wykonywać różne zapytania i operacje aktualizacji (a także inne polecenia) języka SQL. Przykładowo, może zaistnieć potrzeba napisania programu, który będzie pobierał na wejściu zapytania języka SQL (np. wpisywane przez użytkownika), wykonywał te zapytania i wyświetlał na ekranie otrzymane wyniki — tego typu interaktywne interfejsy są dostępne w większości relacyjnych systemów zarządzania bazami danych. Innym przykładem zastosowania tej techniki jest dynamiczne generowanie zapytań w oparciu o przyjazny użytkownikowi interfejs internetowy lub aplikację mobilną. W tym podrozdziale przedstawimy krótki przegląd **dynamicznego języka SQL**, czyli jednej z technik pisania tego typu oprogramowania baz danych — wykorzystamy prosty przykład, który ilustruje działanie tej techniki w praktyce. W podrozdziale 10.3 omówimy inny sposób obsługi dynamicznych zapytań (za pomocą bibliotek funkcji lub klas).

Przedstawiony na rysunku 10.4 fragment programu F3 odczytuje na wejściu podany przez użytkownika ciąg znaków (ciąg powinien zawierać prawidłowe *polecenie aktualizacji* wyrażone w języku SQL) i zapisuje go w zmiennej znakowej `sqlupdatestring` (patrz wiersz 3.). Następnie (w wierszu 4.) program **przygotowuje** otrzymany ciąg znaków do roli polecenia języka SQL przez związanie go ze zmienną tego języka nazwaną `sqlcommand`. W wierszu 5. polecenie jest **wykonywane**. Łatwo zauważyć, że w takim przypadku nie jest możliwa ani weryfikacja składni, ani przeprowadzenie żadnych innych testów polecenia w *czasie kompilacji*, ponieważ nie jest ono znane aż do momentu wykonywania programu. Brak tej możliwości odróżnia tę technikę od rozwiązań stosowanych we wcześniejszych przykładach osadzonych poleceń języka SQL, w których zapytania mogły być sprawdzane w czasie kompilacji, ponieważ ich tekst znajdował się w kodzie źródłowym programu.

*// Fragment F3 programu*

```
0) EXEC SQL BEGIN DECLARE SECTION;
1)  varchar sqlupdatestring[256];
2) EXEC SQL END DECLARE SECTION;
...
3)  prompt("Wpisz polecenie aktualizacji: ", sqlupdatestring);
4) EXEC SQL PREPARE sqlcommand FROM :sqlupdatestring;
5) EXEC SQL EXECUTE sqlcommand;
...
```

RYСУNEK 10.4. Fragment F3 programu napisanego w języku C wykorzystującego dynamiczny język SQL do aktualizowania tabeli bazy danych

Powodem rozdzielenia instrukcji `PREPARE` i `EXECUTE` we fragmencie F3 jest to, że jeśli polecenie ma być wykonywane w programie wielokrotnie, wystarczy je w tym podejściu przygotować raz. **Przygotowanie polecenia** zwykle obejmuje sprawdzanie jego składni i innych aspektów przez system, a także generowanie kodu potrzebnego do wykonania polecenia. Można też połączyć instrukcje `PREPARE` i `EXECUTE` (wiersze 4. i 5. w F3) w jedno polecenie:

```
EXEC SQL EXECUTE IMMEDIATE :sqlupdatestring ;
```

Jest to użyteczne, jeśli polecenie ma być wykonywane tylko raz. Programista może też rozdzielić obie instrukcje (tak jak w F3) w celu wykrycia błędów po wywołaniu `PREPARE`.

Chociaż w dynamicznym języku SQL stosowanie *dynamicznej operacji aktualizacji* jest stosunkowo proste, używanie *dynamicznego zapytania* jest znacznie bardziej skomplikowane. Wynika to z faktu, że w ogólnym przypadku w czasie pisania programu nie jest znany typ ani liczba atrybutów, które zostaną zwrócone przez tak definiowane zapytanie języka SQL. Jeśli programista nie dysponuje znanymi z góry informacjami na temat dynamicznego zapytania, do obsługi różnych liczb i typów atrybutów składających się na wynik tego zapytania niekiedy niezbędne jest zastosowanie skomplikowanych struktur danych. Do przypisywania zmiennym programu nadrzędnego wyników zapytania (i jego parametrów) można wykorzystywać techniki podobne do tych, które omówimy w podrozdziale 10.3.



### 10.2.4. SQLJ: osadzanie poleceń języka SQL w języku Java

W poprzednich punktach przedstawiliśmy przegląd metod osadzania poleceń języka SQL w kodzie źródłowym tworzonym w tradycyjnym języku programowania (posłużyliśmy się przykładem języka C). W tym punkcie skupimy uwagę na technice osadzania poleceń języka SQL w obiektowych językach programowania<sup>8</sup>, a w szczególności w języku Java. Wielu producentów rozwiązań w tym zakresie stosuje standard SQLJ, który umożliwia osadzanie poleceń języka SQL w Javie. Historycznie, standard SQLJ został opracowany później niż interfejs JDBC, który stanowi mechanizm uzyskiwania w języku Java dostępu do baz danych SQL za pomocą bibliotek klas i wywołań funkcji. Standard JDBC omówimy w punkcie 10.3.2. W naszych rozważaniach skupimy się na standardzie SQLJ w wersji stosowanej w relacyjnych systemach zarządzania bazami danych firmy Oracle. Generalnie, translator języka SQLJ przekształca polecenia języka SQL w odpowiednie wyrażenia języka programowania Java, które mogą być następnie wykonywane za pośrednictwem interfejsu JDBC. Oznacza to, że stosowanie standardu SQLJ wymaga zainstalowania *sterownika interfejsu JDBC*<sup>9</sup>. W tym punkcie skupimy się na metodach wykorzystywania standardu SQLJ do osadzania poleceń języka SQL w programach pisanych w języku Java.

Zanim będziemy mogli przetwarzać w systemie Oracle wyrażenia standardu SQLJ umieszczone w kodzie źródłowym programu napisanego w języku Java, musimy zaimportować do tego programu kilka bibliotek klas (patrz rysunek 10.5). Są to między innymi klasy JDBC i IO (odpowiednio wiersze 1. i 2.), plus dodatkowe klasy wymienione w wierszach od 3. do 5. Program musi w pierwszej kolejności nawiązać połączenie z odpowiednią bazą danych za pomocą wywołania funkcji `getConnection`, która jest jedną z metod zaimportowanej w 5. wierszu rysunku 10.5 klasy `oracle`. Poniżej przedstawiono format takiego wywołania, które zwraca obiekt typu `DefaultContext` (reprezentujący *domyślny kontekst*<sup>10</sup> połączenia):

```
public static DefaultContext
getConnection(String url, String user, String password,
    Boolean autoCommit)
throws SQLException;

1) import java.sql.*;
2) import java.io.*;
3) import sqlj.runtime.*;
4) import sqlj.runtime.ref.*;
5) import oracle.sqlj.runtime.*;
...
6) DefaultContext cntxt =
7)     oracle.getConnection("<nazwa URL>", "<nazwa użytkownika>", "<hasło>", true);
8) DefaultContext.setDefaultContext(cntxt);
```

**RYСУNEK 10.5.** Importowanie klas niezbędnych do umieszczania wyrażeń języka SQLJ w programach napisanych w języku Java i używających baz Oracle, a także nawiązywanie połączenia i tworzenie domyślnego kontekstu połączenia

<sup>8</sup> Zakładamy, że Czytelnik rozumie pojęcia związane z programowaniem obiektowym (patrz rozdział 12.) i zna podstawy pisania programów w języku Java.

<sup>9</sup> Zagadnienia związane ze sterownikami interfejsu JDBC omówimy w punkcie 10.3.2.

<sup>10</sup> Ustawienie *domyślnego kontekstu* ma zastosowanie dla wszystkich kolejnych poleceń programu (aż do ewentualnej zmiany kontekstu).



Przykładowo, w programie możemy użyć instrukcji przedstawionych w wierszach od 6. do 8. rysunku 10.5, które nawiązują połączenie z bazą danych Oracle dostępną pod adresem URL *<nazwa URL>*, logują się do tego systemu z *<nazwą użytkownika>* i *<hasłem>*, włączają automatyczne zatwierdzanie każdego z poleceń<sup>11</sup> oraz wyznaczają nowemu połączeniu rolę **domyślnego kontekstu** dla późniejszych poleceń.

W wykorzystywanych w dalszej części tego punktu przykładach nie będziemy analizowali kompletnych klas czy programów napisanych w języku Java, ponieważ naszym celem nie jest uczenie tego języka. Zamiast tego skupimy się na tych fragmentach programu, które dobrze ilustrują stosowanie standardu SQLJ. Fragment kodu zawarty na rysunku 10.6 jest zbiorem deklaracji zmiennych języka Java, które będą nam potrzebne w kolejnych przykładach. Fragment programu z rysunku 10.7 (oznaczony nazwą J1) odczytuje na wejściu numer PESEL pracownika i wyświetla znalezione w bazie danych informacje na temat tego pracownika.

```
1) String nazwadz, pesel, imię, im, nazwisko, na, dataur, adres;
2) char płeć, inicjałdrimienia, idi;
3) double pensja, pen;
4) integer nrdz, numerdz;
```

RYSUNEK 10.6. Zmienne programu napisanego w języku Java, które wykorzystamy w przykładach stosowania standardu SQLJ: J1 oraz J2

```
// Fragment J1 programu
1) pesel = readEntry("Wpisz numer PESEL pracownika: ");
2) try {
3)     #sql{select IMIĘ, INICJAŁDRIMIENIA, NAZWISKO, ADRES, PENSJA
4)         into :imię, :inicjałdrimienia, :nazwisko, :adres, :pensja
5)         from PRACOWNIK where PESEL = :pesel};
6) } catch (SQLException se) {
7)     System.out.println("Podany numer PESEL nie istnieje: " + pesel);
8)     return;
9) }
10) System.out.println(imię + " " + inicjałdrimienia + " " + nazwisko + " " +
    adres + " " + pensja);
```

RYSUNEK 10.7. Fragment J1 programu napisanego w języku Java z blokiem kodu języka SQLJ

Warto zwrócić uwagę na fakt, że ponieważ w języku Java istnieje pojęcie **wyjątków** (wykorzystywanych do obsługi błędów), możemy w naturalny sposób użyć specjalnego wyjątku nazwanego `SQLException` do zwracania informacji o błędach lub wyjątkach wykrytych po wykonaniu w bazie danych polecenia języka SQL. Tak użyty wyjątek `SQLException` odgrywa podobną rolę jak zmienne `SQLCODE` i `SQLSTATE` w omówionej wcześniej technice osadzania poleceń języka SQL w kodzie nadrzędnych języków programowania. W języku Java istnieje wiele typów predefiniowanych wyjątków. Każda operacja

<sup>11</sup> Mówiąc najprościej, *automatyczne zatwierdzanie* oznacza, że każde polecenie jest faktycznie stosowane dla bazy danych bezpośrednio po jego wykonaniu. Alternatywne rozwiązanie polega na wykonywaniu wielu powiązanych ze sobą operacji na bazie danych i ich łącznym zatwierdzeniu. Pojęcia związane z zatwierdzaniem omówimy w rozdziale 20., w którym opiszemy stosowany w bazach danych mechanizm obsługi transakcji.

(funkcja) tego języka musi określać wyjątki, które może **spowodować** — czyli stany, które mogą wystąpić w czasie wykonywania kodu danej operacji. Jeśli zdefiniowany wyjątek faktycznie wystąpi, system przekaże kontrolę do tej części kodu napisanego w języku Java, która została wyznaczona do obsługi danego wyjątku. We fragmencie J1 za obsługę wyjątku `SQLException` odpowiadają wyrażenia umieszczone w wierszach 7. i 8. W języku Java do obsługi wyjątków, jakie zajądą w czasie wykonywania instrukcji `<operacja>`, służy następująca struktura:

```
try {<operacja>} catch (<wyjątek>) {<kod obsługi wyjątku>}
    <dalszy kod>
```

Jeśli wyjątek nie wystąpi, bezpośrednio uruchamiany jest `<dalszy kod>`. Wyjątki, które mogą być wygenerowane wskutek wykonywania kodu określonej operacji, powinny być wyszczególnione w deklaracji lub *interfejsie* tej operacji — np. w przedstawionej poniżej formie:

```
<typ zwracany przez operację> <nazwa operacji>(<parametry>)
    throws SQLException, IOException;
```

W standardzie SQLJ polecenia języka SQL osadzone w kodzie programu napisanego w języku Java są oznaczane prefiksem `#sql` (patrz wiersz 3. fragmentu J1), dzięki czemu mogą być łatwo identyfikowane przez preprocesor. Prefiks `#sql` jest używany zamiast słów kluczowych `EXEC SQL` stosowanych w osadzonym języku SQL w języku C (patrz punkt 10.2.1). W standardzie SQLJ wykorzystuje się *klauzulę INTO* (podobną do tej stosowanej w technice osadzania kodu języka SQL) do przypisywania znalezionych w bazie danych (w oparciu o wykonane zapytanie) wartości atrybutów zmiennym programu napisanego w Javie. Podobnie jak w osadzanych poleceniach języka SQL, także w standardzie SQLJ zmienne programu poprzedza się w kodzie języka SQL znakiem dwukropka.

We fragmencie J1 osadzone zapytanie języka SQL zwraca *pojedynczą krotkę*; oznacza to, że możemy w wierszu 4. z rysunku 10.7 użyć klauzuli `INTO` do przypisania wartości jej atrybutów bezpośrednio do zmiennych programu napisanego w języku Java. W przypadku zapytań zwracających wiele krotek, standard SQLJ przewiduje stosowanie pojęcia *iteratora*, który pod wieloma względami przypomina kursor wykorzystywany w technice osadzania poleceń języka SQL.

## 10.2.5. Używanie iteratorów do przetwarzania wyników zapytań w standardzie SQLJ

**Iterator** w standardzie SQLJ jest typem obiektu powiązanego ze zbiorem (lub wielozbiorem) krotek zwróconym w **wyniku zapytania**<sup>12</sup>. Iterator zawsze jest związany z krotkami i atrybutami występującymi w relacji wynikowej zapytania. Istnieją dwa rodzaje iteratorów:

- (1) **Nazwany iterator** jest związany z wynikiem zapytania przez zawieranie *nazw i typów* atrybutów występujących w relacji wynikowej zapytania. Nazwy atrybutów muszą odpowiadać właściwie zadeklarowanym zmiennym programu w języku Java (patrz rysunek 10.6).

---

<sup>12</sup> Pojęcie iteratora wyjaśnimy bardziej szczegółowo w rozdziale 12., kiedy będziemy analizowali obiektowe bazy danych.

- (2) **Iterator pozycyjny** określa tylko *typy atrybutów*, które występują w relacji wynikowej zapytania.

W obu przypadkach porządek atrybutów na liście powinien być *taki sam* jak kolejność atrybutów wymienionych w klauzuli SELECT wykonanego zapytania. Okazuje się jednak, że sam sposób przeglądania wyniku zapytania w pętli różni się w zależności od zastosowanego rodzaju iteratora. W pierwszej kolejności przeanalizujemy przykład zastosowania *nazwanego* iteratora we fragmencie programu J2A (patrz rysunek 10.8). Wiersz 9. rysunku 10.8 pokazuje sposób, w jaki deklarujemy *nazwany iterator* Prac. Warto pamiętać, że nazwy atrybutów w tego rodzaju iteratorach muszą odpowiadać nazwom atrybutów w relacji wynikowej zapytania języka SQL. W wierszu 10. mamy do czynienia z przykładem stworzenia w programie *obiekту iteratora* p typu Prac, natomiast w wierszach 11. i 12. wykonano operację związania tego obiektu z zapytaniem.

```
// Fragment J2A programu
0)  nazwadz = readEntry("Wpisz nazwę działu: ");
1)  try {
2)    #sql{select NUMERDZ into :numerdz
3)        from DZIAŁ where NAZWADZ = :nazwadz};
4)  } catch (SQLException se) {
5)    System.out.println("Podany dział nie istnieje: " + nazwadz);
6)    return;
7)  }
8)  System.out.println("Informacje o pracownikach działu: " + nazwadz);
9)  #sql iterator Prac(String pesel, String imię, String inicjałdrimienia,
10)    String nazwisko, double pensja);
10) Prac p = null;
11) #sql p = {select pesel, imię, inicjałdrimienia, nazwisko, pensja
12)    from PRACOWNIK where NRDZ = :numerdz};
13) while (p.next()) {
14)   System.out.println(p.pesel + " " + p.imię + " " + p.inicjałdrimienia +
15)     " " + p.nazwisko + " " + p.pensja);
16) }
16) p.close();
```

RYSUNEK 10.8. Fragment J2A programu napisanego w języku Java wykorzystującego nazwany iterator do wyświetlania zawartych w bazie danych informacji o pracownikach wskazanego przez użytkownika działu

Kiedy obiekt iteratora jest już związany z odpowiednim zapytaniem (patrz wiersze 11. i 12. rysunku 10.8), program pobiera z bazy danych wynik zapytania i ustawia iterator na pozycji *przed pierwszym wierszem* relacji wynikowej tego zapytania. Wiersz ten staje się wówczas **wierszem bieżącym** danego iteratora. Następnie dla tak przygotowanego iteratora stosuje się operację *next*, z których każda przesuwają iterator do *kolejnego wiersza* relacji wynikowej zapytania, nadając mu tym samym status wiersza bieżącego iteratora. Jeśli taki kolejny wiersz istnieje, operacja *next* zapisuje w odpowiednich zmiennych programu wartości atrybutów tego wiersza. Jeśli wiersze w relacji wynikowej zapytania zostały wyczerpane, operacja *next* zwraca wartość pustą, którą w prosty sposób możemy wykorzystać do kontroli zakresu pętli. Warto zauważyć, że iterator nazwany *nie wymaga* klauzuli INTO, ponieważ zmienne programu odpowiadające pobieranym atrybutom są już określone w momencie deklarowania typu iteratora (wiersz 9. rysunku 10.8).

Polecenie `p.next()` w wierszu 13. rysunku 10.8 pełni jednocześnie dwie funkcje: zwraca następną krotkę z relacji wynikowej zapytania oraz kontroluje zakres pętli `while`. Kiedy już przeszukamy w pętli wszystkie krotki otrzymane w wyniku wykonanego zapytania, możemy użyć polecenia `p.close()` do zamknięcia niepotrzebnego iteratora `p` (patrz wiersz 16.).

Zajmiemy się teraz identycznym przykładem, tyle że zbudowanym w oparciu o iteratory *pozycyjne* (patrz fragment programu J2B z rysunku 10.9). Polecenie zawarte w wierszu 9. rysunku 10.9 pokazuje sposób, w jaki można zadeklarować *typ iteratora pozycyjnego* `Pracpoz`. Główna różnica pomiędzy iteratorem pozycyjnym a użytym w poprzednim przykładzie nazwanym iteratorem polega na tym, że w pierwszym przypadku nie mamy dostępu do nazw atrybutów (odpowiadającym nazwom zmiennych z programu), a jedynie do ich typów. Może to dawać więcej swobody, jednak sprawia, że przetwarzanie wyników zapytania staje się bardziej skomplikowane. Oczywiście nadal musi istnieć zgodność typów atrybutów zwróconych w relacji wynikowej zapytania języka SQL. Wiersz 10. pokazuje sposób, w jaki tworzymy w programie *obiekt iteratora pozycyjnego* e typu `Pracpoz`, natomiast wiersze 11. i 12. tego listingu prezentują metodę wiązania tego iteratora z odpowiednim zapytaniem.

// Fragment J2B programu

```
0)  nazwadz = readEntry("Wpisz nazwę działu: ");
1)  try {
2)      #sql{SELECT NUMERDZ INTO :numerdz
3)          FROM DZIAŁ WHERE NAZWADZ = :nazwadz};
4)  } catch (SQLException se) {
5)      System.out.println("Podany dział nie istnieje: " + nazwadz);
6)      return;
7)  }
8)  System.out.println("Informacje o pracownikach działu: " + nazwadz);
9)  #sql iterator Pracpoz(String, String, String, String, double);
10) Pracpoz p = null;
11) #sql p = {SELECT pesel, imię, inicjałdrimienia, nazwisko, pensja
12)          FROM PRACOWNIK WHERE NRDZ = :numerdz};
13) #sql {FETCH :p INTO :pesel, :im, :idi, :na, :pen};
14) while (!p.endFetch()) {
15)     System.out.println(pesel + " " + im + " " + idi + " " + na + " " + pen);
16)     #sql {FETCH :p INTO :pesel, :im, :idi, :na, :pen};
17) }
18) p.close();
```

RYSUNEK 10.9. Fragment J2B programu napisanego w języku Java wykorzystującego iterator pozycyjny do wyświetlania zawartych w bazie danych informacji o pracownikach wskazanego przez użytkownika działu

Iterator pozycyjny funkcjonuje w sposób bardziej przypominający elementy techniki osadzania poleceń języka SQL w kodzie nadrzędnego języka programowania (patrz punkt 10.2.2). Polecenie `FETCH <zmienna iteratora> INTO <zmiennie programu>` jest niezbędne do otrzymania kolejnej krotki z relacji wynikowej zapytania. Kiedy polecenie `fetch` jest wywoływane po raz pierwszy (w wierszu 13. rysunku 10.9), w jego rezultacie otrzymujemy pierwszą krotkę. W wierszu 16. odczytujemy kolejne krotki z relacji wynikowej zapytania, aż dojdziemy do punktu, w którym wyczerpane zostaną wszystkie krotki. Do

kontrolowania zakresu pętli wykorzystujemy funkcję iteratora pozycyjnego `p.endFetch()`. Funkcja ma automatycznie ustawianą wartość `TRUE` w momencie początkowego wiązania iteratora z zapytaniem języka SQL (w wierszu 11.) i ma ustawianą wartość `FALSE` za każdym razem, gdy polecenie `fetch` zwróci poprawną krotkę z relacji wynikowej zapytania. Ta sama funkcja ma ponownie ustawianą wartość `TRUE` w chwili, gdy wszystkie krotki relacji wynikowej zostaną wyczerpane. Wyrażenie w wierszu 14. pokazuje, jak możemy wykorzystać logiczny operator negacji do kontrolowania zakresu działania pętli.

## 10.3. Programowanie baz danych z wywołaniami funkcji i bibliotekami klas: SQL/CLI oraz JDBC

Technikę osadzania poleceń języka SQL w kodzie nadrzędnego języka programowania (patrz podrozdział 10.2) nazywa się niekiedy **statycznym** programowaniem bazy danych, ponieważ tekst zapytania jest zapisany w programie i nie może być zmieniany bez jego ponownej kompilacji lub ponownego przetworzenia kodu źródłowego. Zastosowanie wywołań funkcji jest bardziej **dynamicznym** rozwiązaniem w zakresie programowania baz danych niż osadzanie poleceń języka SQL. W punkcie 10.2.3 mieliśmy okazję zapoznać się z jedną z technik dynamicznego programowania baz danych — dynamicznym językiem SQL. Metody omawiane w tym podrozdziale opierają się na nieco innym podejściu do techniki dynamicznego programowania baz danych. Do uzyskiwania dostępu do baz danych wykorzystuje się **biblioteki funkcji** nazywane także **interfejsami programowymi aplikacji** (ang. *Application Programming Interface*, w skrócie *API*). Takie rozwiązanie zapewnia co prawda większą elastyczność (z uwagi na wyeliminowanie preprocesora z procesu), jego niewątpliwą wadą jest jednak fakt, że weryfikacja składni i pozostałe testy poleceń języka SQL muszą być wykonywane w czasie działania programu. Kolejną wadą jest konieczność stosowania bardziej skomplikowanych konstrukcji programistycznych podczas przetwarzania relacji wynikowych zapytań, ponieważ zawarte w niej typy atrybutów i ich liczba nie zawsze są znane z góry.

W tym podrozdziale przedstawimy krótki przegląd dwóch interfejsów wywołań funkcji. W pierwszej kolejności omówimy interfejs **SQL/CLI** (ang. *Call Level Interface*), który stanowi część standardu SQL. Interfejs ten został zaprojektowany z myślą o stworzeniu następcy dla stosowanej wcześniej techniki znanej jako *ODBC* (ang. *Open DataBase Connectivity*). W naszych przykładach praktycznego zastosowania interfejsu SQL/CLI będziemy wykorzystywali język programowania C. W dalszej części tego podrozdziału przeanalizujemy najważniejsze elementy standardu **JDBC**, który jest interfejsem wywoływania funkcji umożliwiającym dostęp do baz danych z poziomu programów napisanych w języku Java. Chociaż wielu programistów sądzi, że skrót JDBC oznacza Java DataBase Connectivity, w rzeczywistości nazwa tego standardu została zastrzeżona przez firmę Sun Microsystems i *nie* jest akronimem.

Najważniejszą zaletą stosowania interfejsów wywołań funkcji są ułatwienia w zakresie uzyskiwania dostępu do wielu baz danych z poziomu tego samego programu aplikacji (nawet jeśli każda z tych baz danych jest kontrolowana przez zupełnie inny pakiet systemu zarządzania bazą danych). Omówimy to zagadnienie bardziej szczegółowo w punkcie

10.3.2, w którym będziemy analizowali programowanie baz danych w języku Java z wykorzystaniem standardu JDBC, chociaż taka możliwość występuje także w przypadku programowania baz danych z wykorzystaniem interfejsów SQL/CLI oraz ODBC (patrz punkt 10.3.1).

### 10.3.1. Programowanie baz danych z wykorzystaniem interfejsu SQL/CLI oraz języka C w roli nadrzędnego języka programowania

Zanim będziemy mogli wykorzystywać wywołania funkcji w oparciu o standard SQL/CLI, konieczne musimy zainstalować na serwerze bazy danych odpowiednie pakiety bibliotek. Takie pakiety są zwykle udostępniane przez producentów systemów zarządzania bazami danych. Przedstawimy teraz krótką analizę możliwości praktycznego stosowania standardu SQL/CLI w programach napisanych w języku C<sup>13</sup>. Naszą prezentację zilustrujemy przykładowym fragmentem programu oznaczonym symbolem CLI1 (patrz rysunek 10.10).

*// Fragment CLI1 programu:*

```
0) #include sqlcli.h;
1) void printSal() {
2)     SQLHSTMT stmt1;
3)     SQLHDBC con1;
4)     SQLHENV env1;
5)     SQLRETURN ret1, ret2, ret3, ret4;
6)     ret1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env1);
7)     if (!ret1) ret2 = SQLAllocHandle(SQL_HANDLE_DBC, env1, &con1) else exit;
8)     if (!ret2) ret3 = SQLConnect(con1, "dbs", SQL_NTS, "js", SQL_NTS, "xyz",
        SQL_NTS);
9)     if (!ret3) ret4 = SQLAllocHandle(SQL_HANDLE_STMT, con1, &stmt1) else exit;
10)    SQLPrepare(stmt1, "select NAZWISKO, PENSJA from PRACOWNIK where PESEL = ?",
        SQL_NTS);
11)    prompt("Wpisz numer PESEL pracownika: ", &pesel);
12)    SQLBindParameter(stmt1, 1, SQL_CHAR, &pesel, 11, &fetchlen1);
13)    ret1 = SQLExecute(stmt1);
14)    if (!ret1) {
15)        SQLBindCol(stmt1, 1, SQL_CHAR, &nazwisko, 15, &fetchlen1);
16)        SQLBindCol(stmt1, 2, SQL_FLOAT, &pensja, 4, &fetchlen2);
17)        ret2 = SQLFetch(stmt1);
18)        if (!ret2) printf(pesel, nazwisko, pensja);
19)        else printf("Podany numer pesel nie istnieje: ", &pesel);
20)    }
21) }
```

RYСУNEK 10.10. Fragment CLI1 programu napisanego w języku C, który wykorzystuje do obsługi bazy danych interfejs SQL/CLI

<sup>13</sup> To omówienie dotyczy też języków C++ i C#, ponieważ nie używamy żadnych mechanizmów obiektowych i koncentrujemy się tylko na mechanizmach programowania baz danych.

**Uchwyty rekordów środowiska, połączenia, instrukcji i opisu.** Jeśli zbudujemy naszą aplikację w oparciu o standard SQL/CLI, polecenia języka SQL będą dynamicznie tworzone i przekazywane w postaci *parametrów znakowych* do wywołań funkcji. Oznacza to, że informacje na temat interakcji programu nadrzędnego z bazą danych musimy śledzić w czasie wykonywania i rejestrować w odpowiednich strukturach danych, ponieważ właśnie w czasie działania przetwarzane są polecenia bazy danych. Tego rodzaju informacje są przechowywane w czterech typach rekordów reprezentowanych w języku C za pomocą odpowiednich *struktur*. **Rekord środowiska** jest wykorzystywany w roli pojemnika śledzącego jedno lub więcej połączeń z bazą (bazami) danych, a także do ustawiania informacji o środowisku. **Rekord połączenia** zawiera informacje niezbędne do utrzymywania konkretnego połączenia z bazą danych. **Rekord polecenia** zawiera informacje potrzebne do obsługi pojedynczego wyrażenia języka SQL. W **rekordzie opisu** zapisywane są informacje na temat krotek lub parametrów — przykładowo, rekord opisu może zawierać liczbę atrybutów pojedynczej krotki wraz z ich typami lub liczbę i typy parametrów pojedynczego wywołania funkcji. Te dane są potrzebne, gdy programista w trakcie pisania programu nie ma takich informacji na temat zapytania. W przykładach zakładamy, że programista zna zapytanie, dlatego nie pokazujemy rekordów opisu.

Dostęp do każdego rekordu programu odbywa się za pośrednictwem zmiennej wskaźnikowej języka C, którą często określa się mianem **uchwyty** tego rekordu. Każdy taki uchwyt jest zwracany w momencie, w którym wskazywany przez niego rekord jest tworzony po raz pierwszy. Do tworzenia pojedynczych rekordów i zwracania ich uchwytów służy następująca funkcja interfejsu SQL/CLI:

```
SQLAllocHandle(<typ uchwyty>, <uchwyt 1>, <uchwyt 2>);
```

Funkcja `SQLAllocHandle()` pobiera następujące parametry:

- *<typ uchwyty>* oznacza typ tworzonego rekordu. Do możliwych wartości tego parametru należą słowa kluczowe `SQL_HANDLE_ENV`, `SQL_HANDLE_DBC`, `SQL_HANDLE_STMT` oraz `SQL_HANDLE_DESC`, które odpowiadają kolejno rekordom środowiska, połączenia, polecenia i opisu.
- *<uchwyt 1>* oznacza pojemnik, wewnątrz którego zostanie utworzony nowy uchwyt. Przykładowo, dla rekordu połączenia będzie to środowisko, w którym dane połączenie jest tworzone, natomiast w przypadku rekordu polecenia byłoby to połączenie dla danego polecenia.
- *<uchwyt 2>* jest wskaźnikiem (uchwytem) do nowoutworzonego rekordu typu *<typ uchwyty>*.

**Kroki w programie bazy danych.** Pisząc program w języku C, który będzie zawierał odwołania do bazy danych realizowane za pośrednictwem interfejsu SQL/CLI, zazwyczaj wykonuje się kolejno poniższe kroki. Zilustrujemy znaczenie poszczególnych czynności posługując się praktycznymi przykładami z fragmentu CLI1 (patrz rysunek 10.10), który odczytuje na wejściu numer PESEL pracownika i wyświetla na ekranie jego nazwisko i wysokość pensji.

- (1) **Dołączanie biblioteki funkcji.** Do programu napisanego w języku C należy dołączyć *bibliotekę funkcji* obsługującą interfejs SQL/CLI. Plik nagłówkowy tej biblioteki został nazwany *sqlcli.h* — dołączyliśmy go do programu w wierszu 0. (patrz rysunek 10.10).



- (2) **Deklarowanie zmiennych uchwytów.** Musimy zadeklarować niezbędne *zmienne uchwytów* należące do typów `SQLHSTMT`, `SQLHDBC`, `SQLHENV` oraz `SQLHDESC` odpowiadających odpowiednio rekordom poleceń, połączeń, środowisk oraz opisów (patrz wiersze od 2. do 4.)<sup>14</sup>. Powinniśmy także zadeklarować zmienne typu `SQLRETURN` (wiersz 5.), które będą przechowywały kody zwracane przez wywołania funkcji biblioteki SQL/CLI. Kod zerowy oznacza *pomyślne wykonanie wywołania funkcji*.
- (3) **Rekord środowiska.** W programie musimy dodatkowo ustawić *rekord środowiska* za pomocą wspomnianej już funkcji `SQLAllocHandle`. Odpowiednie wywołanie umieściliśmy w wierszu 6. Ponieważ rekord środowiska nie może być przechowywany w ramach żadnego innego rekordu, podczas tworzenia środowiska parametr `<uchwyt 1>` jest uchwyt (wskaźnikiem) pustym typu `SQL_NULL_HANDLE`. W naszym przykładzie uchwyt (wskaźnik) nowoutworzonego rekordu środowiska jest zwracany w postaci zmiennej `env1` (patrz wiersz 6.).
- (4) **Nawiązywanie połączenia z bazą.** Funkcję `SQLAllocHandle` wykorzystujemy także do ustawiania *rekordu połączenia* w programie. Utworzony w wierszu 7. rekord połączenia ma przypisany uchwyt `con1` i jest umieszczany w środowisku reprezentowanym przez zmienną `env1`. Następnie nawiązywane jest **połączenie** z konkretnym serwerem bazy danych (umieszczone w zmiennej `con1`) za pomocą funkcji `SQLConnect` biblioteki SQL/CLI (patrz wiersz 8.). W naszym przykładzie łączymy się z serwerem bazy danych nazwanym "dbs" i wykorzystujemy nazwę konta "js" i hasło "xyz".
- (5) **Rekord polecenia.** *Rekord polecenia* jest ustawiany w programie za pomocą znanej nam już funkcji `SQLAllocHandle`. W wierszu 9. tworzymy nowy taki rekord, wykorzystujący połączenie `con1` i z przypisanym uchwycem `stmt1`.
- (6) **Przygotowanie polecenia języka SQL i jego parametrów.** Polecenie języka SQL jest *przygotowywane* za pomocą funkcji biblioteki SQL/CLI nazwanej `SQLPrepare`. W wierszu 10. przypisujemy **ciąg znaków polecenia** języka SQL (w naszym przypadku jest to proste *zapytanie*) do uchwytu polecenia `stmt1`. Użyty w wierszu 10. znak zapytania (?) reprezentuje **parametr polecenia**, którego wartość ma być określana w czasie wykonywania programu — zwykle odbywa się to przez związanie tego symbolu z odpowiednią zmienną programu napisanego w języku C. Ogólnie rzecz biorąc, pojedyncze polecenie języka SQL może zawierać wiele takich parametrów, które można wówczas odróżnić według kolejności występowania znaków zapytania w poleceniu (pierwszy znak zapytania reprezentuje pierwszy parametr, drugi znak zapytania — drugi parametr itd.). Ostatni parametr funkcji `SQLPrepare` powinien określać wyrażoną w bajtach długość ciągu znaków zawierającego polecenie języka SQL; jeśli jednak użyjemy w tym miejscu słowa kluczowego `SQL_NTS`, będzie to oznaczało, że ciąg znaków zawierający zapytanie jest *zakończony znakiem pustym*, zatem długość tego ciągu będzie można obliczyć automatycznie podczas jego przetwarzania. To samo dotyczy *wszystkich pozostałych parametrów znakowych* stosowanych w przykładach.
- (7) **Wiązanie parametrów polecenia.** Zanim będziemy mogli wykonać zapytanie, powinniśmy powiązać wszystkie parametry z odpowiednimi zmiennymi programu — służy do tego funkcja `SQLBindParameter` biblioteki SQL/CLI. W wierszu 12. rysunku

---

<sup>14</sup> Aby dodatkowo nie komplikować omawianego przykładu, nie zastosowaliśmy w nim rekordów opisu.

10.10 parametr (oznaczony za pomocą znaku zapytania) przygotowywanego zapytania reprezentowanego przez zmienną `stmt1` jest wiązany ze zmienną `pesel` programu w języku C. Jeśli w danym poleceniu języka SQL istnieje  $n$  parametrów, powinniśmy użyć  $n$  wywołań funkcji `SQLBindParamater`, po jednym dla każdego parametru znajdującego się na kolejnej pozycji (1, 2, ...,  $n$ ).

- (8) **Wykonywanie polecenia.** Po wykonaniu omówionych powyżej kroków przygotowawczych możemy wykonać wskazywane przez uchwyt `stmt1` polecenie języka SQL za pomocą funkcji `SQLExecute` (patrz wiersz 13.). Warto pamiętać, że chociaż zapytanie zostanie wykonane wskutek wywołania tej funkcji w wierszu 13., wyniki tego zapytania nie zostaną automatycznie przypisane do zmiennych programu napisanego w języku C.
- (9) **Przetwarzanie wyniku zapytania.** Do określania, gdzie został zwrócony wynik naszego zapytania, często wykorzystuje się technikę opartą na tzw. **powiązanych kolumnach**. Każda kolumna w relacji wynikowej zapytania jest wówczas wiązana ze zmienną programu w języku C za pomocą funkcji `SQLBindCol`. Kolumny są od siebie odróżniane na podstawie kolejności występowania w zapytaniu języka SQL. W wierszach 15. i 16. rysunku 10.10 dwie kolumny z zapytania (`NAZWISKO` i `PENSJA`) są wiązane odpowiednio ze zmiennymi `nazwisko` i `pensja` zadeklarowanymi w programie napisanym w języku C<sup>15</sup>.
- (10) **Pobieranie wartości kolumn.** I wreszcie, do odczytywania z wyniku zapytania języka SQL wartości kolumn i ich zapisywania w odpowiednich zmiennych programu w języku C służy funkcja `SQLFetch` (patrz wiersz 17.). Działanie tej funkcji przypomina działanie polecenia `FETCH` stosowanego w osadzanych poleceniach języka SQL. Jeśli wynik zapytania zawiera zbiór krotek, każde wywołanie funkcji `SQLFetch` zwraca i zapisuje w zmiennych programu wartości kolumn składających się na pojedynczą krotkę. Jeśli zbiór krotek zostanie wyczerpany, funkcja `SQLFetch` zwraca niezerowy kod wyjątku<sup>16</sup>.

Łatwo zauważyć, że stosowanie dynamicznych wywołań funkcji wymaga niemałego wysiłku związanego z właściwym przygotowaniem i ustawieniem poleceń języka SQL oraz z powiązaniem parametrów i wyników zapytania z odpowiednimi zmiennymi programu.

W przedstawionym fragmencie CLI1 zapytanie języka SQL zwróciło *pojedynczą krotkę*. Na rysunku 10.11 przedstawiono przykład podobnego programu, tyle że obsługującego zapytanie zwracające wiele krotek. Zakładamy, że niezbędne zmienne programu napisanego w języku C zostały zadeklarowane tak, jak w kodzie z rysunku 10.1. Fragment programu CLI2 odczytuje na wejściu numer działu i wyszukuje pracowników, którzy są w tym dziale zatrudnieni. Następnie zastosowano pętlę, która w kolejnych iteracjach przeszukuje rekordy reprezentujące pracowników (po jednym w każdej iteracji) i wyświetla zawarte w nich nazwiska i pensje pracowników.

---

<sup>15</sup> Istnieje alternatywna technika oparta na **kolumnach niepowiązanych**, w której do odczytywania kolumn z relacji wynikowej zapytania (bez ich uprzedniego wiązania ze zmiennymi programu) wykorzystuje się inne funkcje biblioteki SQL/CLI — `SQLGetCol` lub `SQLGetData`, które są stosowane po wywołaniu polecenia `SQLFetch` w wierszu 17.

<sup>16</sup> Gdybyśmy zastosowali niezwiązane zmienne programu, funkcja `SQLFetch` zwróciłaby krotkę do tymczasowego obszaru przechowywania danych w programie. Każde kolejne wywołanie funkcji `SQLGetCol` (lub `SQLGetData`) zwracałoby wówczas wartość pojedynczego atrybutu. Dla każdego wiersza z wyniku zapytania program powinien iteracyjnie pobrać wartości atrybutów (kolumn) z tego wiersza. Jest to przydatne, gdy liczba kolumn w wynikach zapytania może się zmieniać.

// Fragment CLI2 programu:

```

0) #include sqlcli.h;
1) void printDepartmentEmps() {
2)     SQLHSTMT stmt1;
3)     SQLHDBC con1;
4)     SQLHENV env1;
5)     SQLRETURN ret1, ret2, ret3, ret4;
6)     ret1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env1);
7)     if (!ret1) ret2 = SQLAllocHandle(SQL_HANDLE_DBC, env1, &con1) else exit;
8)     if (!ret2) ret3 = SQLConnect(con1, "dbs", SQL_NTS, "js", SQL_NTS, "xyz",
        SQL_NTS);
9)     if (!ret3) ret4 = SQLAllocHandle(SQL_HANDLE_STMT, con1, &stmt1) else exit;
10)    SQLPrepare(stmt1, "select NAZWISKO, PENSJA from PRACOWNIK where NRDZ = ?",
        SQL_NTS);
11)    prompt("Wpisz numer działu: ", nrdz);
12)    SQLBindParameter(stmt1, 1, SQL_INTEGER, &nrdz, 4, &fetchlen1);
13)    ret1 = SQLExecute(stmt1);
14)    if (!ret1) {
15)        SQLBindCol(stmt1, 1, SQL_CHAR, &nazwisko, 15, &fetchlen1);
16)        SQLBindCol(stmt1, 2, SQL_FLOAT, &pensja, 4, &fetchlen2);
17)        ret2 = SQLFetch(stmt1);
18)        while (!ret2) {
19)            printf(nazwisko, pensja);
20)            ret2 = SQLFetch(stmt1);
21)        }
22)    }
23) }
```

RYSUNEK 10.11. Fragment CLI2 programu napisanego w języku C, który wykorzystuje interfejs SQL/CLI do obsługi zapytania, którego wynikiem jest cały zbiór krotek

### 10.3.2. JDBC: biblioteka klas języka SQL służąca do programowania w języku Java

W tym punkcie skupimy się na omawianiu techniki wywołania poleceń języka SQL z poziomu obiektowego języka programowania Java<sup>17</sup>. Odpowiednia biblioteka funkcji wykorzystywanych podczas uzyskiwania dostępu do bazy danych jest znana po nazwę **JDBC**<sup>18</sup>. Język programowania Java został zaprojektowany z myślą o zapewnieniu niezależności tworzonego oprogramowania od platformy — oznacza to, że program napisany w Javie będzie można uruchamiać w dowolnym typie systemu komputerowego, w którym zainstalowano odpowiedni interpreter tego języka. Właśnie ze względu na przenośność aplikacji tworzonych w języku Java wielu producentów relacyjnych systemów zarządzania bazami danych dołącza do swoich pakietów sterowniki JDBC, które umożliwiają uzyskiwanie dostępu do zasobów ich systemów za pośrednictwem programów napisanych w Javie.

<sup>17</sup> W tym punkcie zakładamy, że Czytelnik zna podstawowe pojęcia związane z programowaniem obiektowym oraz ma elementarną wiedzę na temat samego języka Java. Czytelnicy, którzy nie dysponują odpowiednią wiedzą, powinni odłożyć lekturę tego punktu do czasu zapoznania się z treścią rozdziału 20.

<sup>18</sup> Jak już wspominaliśmy, **JDBC** jest zastrzeżonym przez firmę Sun Microsystems znakiem towarowym, chociaż wielu programistów sądzi, że jest to akronim słów Java Database Connectivity.

**Sterowniki JDBC.** Sterownik JDBC jest tak naprawdę implementacją klas oraz powiązanych z nimi obiektów i wywołań funkcji określonych w specyfikacji interfejsu JDBC *API* (ang. *Application Programming Interface*) dla opracowanego przez danego producenta relacyjnego systemu zarządzania bazą danych. Oznacza to, że program napisany w języku Java i wykorzystujący obiekty oraz wywołania funkcji biblioteki JDBC może uzyskać dostęp do zasobów dowolnych relacyjnych systemów zarządzania bazami danych, które zawierają sterowniki JDBC.

Ponieważ Java jest językiem obiektowym, wykorzystywane w jej kodzie biblioteki funkcji muszą być implementowane w postaci odpowiednich **klas**. Zanim będziemy mogli w programie napisanym w tym języku przetwarzać wywołania funkcji bibliotek JDBC, musimy zaimportować **biblioteki klas JDBC**, nazwane `java.sql.*`. Można je pobrać z internetu i zainstalować<sup>19</sup>.

Interfejs JDBC został zaprojektowany z myślą o umożliwieniu pojedynczym programom napisanym w języku Java jednoczesnego łączenia się z wieloma bazami danych. Tak wykorzystywane bazy danych są niekiedy nazywane **źródłami danych**. Źródła danych mogą mieć postać informacji obsługiwanych przez systemy zarządzania bazami danych stworzonych przez różnych producentów i działających na różnych komputerach. Oznacza to, że różne źródła danych wykorzystywane przez ten sam program napisany w języku Java mogą wymagać stosowania sterowników JDBC różnych producentów. Aby stosowanie tak elastycznego modelu było możliwe, do biblioteki JDBC dodano specjalną klasę nazwaną **menadżerem sterowników**, która utrzymuje niezbędne informacje na temat zainstalowanych sterowników. Każdy sterownik powinien zostać *zarejestrowany* za pomocą menadżera sterowników. Do podstawowych operacji (metod) klasy menadżera sterowników należą: `getDriver`, `registerDriver` oraz `deregisterDriver`. Wymienione metody można wykorzystywać do dynamicznego dodawania i usuwania sterowników. Inne funkcje tej klasy służą do ustanawiania i zamykania połączeń ze źródłami danych.

Do jawnego wczytywania sterowników JDBC służy uniwersalna funkcja języka Java przeznaczona do wczytywania klas. Przykładowo, aby wczytać sterownik JDBC dla systemu zarządzania bazą danych Oracle, powinniśmy użyć następującego polecenia:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

W ten sposób możemy zarejestrować dany sterownik w menadżerze sterowników i udostępnić go w programie. Istnieje także możliwość wczytywania i rejestrowania niezbędnego sterownika (lub sterowników) w wierszu poleceń, w którym uruchamiamy dany program — przykładowo, do polecenia uruchamiającego nasz program możemy dodać następujące wyrażenie:

```
-Djdbc.drivers = oracle.jdbc.driver
```

**Kroki programowania z użyciem interfejsu JDBC.** Poniżej wymieniliśmy typowe kroki podejmowane przez programistów podczas tworzenia w języku Java aplikacji współpracujących z bazami danych w oparciu o wywołania funkcji interfejsu JDBC. Podczas analizy kolejnych czynności będziemy się odwoływali do przykładowego fragmentu JDBC1 (patrz rysunek 10.12), który odczytuje na wejściu numer PESEL pracownika i wyświetla na ekranie jego nazwisko i wysokość pensji.

---

<sup>19</sup> Biblioteki klas JDBC są dostępne na wielu witrynach internetowych (np. na stronie: <http://industry.java.sun.com/products/jdbc/drivers>).

// Fragment JDBC1 programu:

```

0) import java.io.*;
1) import java.sql.*;
...
2) class getEmpInfo {
3)     public static void main(String args[]) throws SQLException, IOException
4)     {
5)         try { Class.forName("oracle.jdbc.driver.OracleDriver");
6)         } catch (ClassNotFoundException x) {
7)             System.out.println("Wczytanie sterownika jest niemożliwe");
8)         }
9)         String dbacct, password, pesel, nazwisko;
10)        double pensja;
11)        dbacct = readentry("Wpisz nazwę konta w bazie danych: ");
12)        password = readentry("Wpisz hasło do bazy danych: ");
13)        Connection conn = DriverManager.getConnection
14)            ("jdbc:oracle:oci8:" + dbacct + "/" + password);
15)        String stmt1 = "select NAZWISKO, PENSJA from PRACOWNIK where PESEL = ?";
16)        PreparedStatement p = conn.prepareStatement(stmt1);
17)        pesel = readentry("Wpisz numer PESEL pracownika: ");
18)        p.clearParameters();
19)        p.setString(1, pesel);
20)        ResultSet r = p.executeQuery();
21)        while (r.next()) {
22)            nazwisko = r.getString(1);
23)            pensja = r.getDouble(2);
24)            System.out.println(nazwisko + pensja);
25)        } }

```

RYSUNEK 10.12. Fragment JDBC1 programu napisanego w języku Java, który wykorzystuje do komunikacji z bazą danych interfejs JDBC

- (1) **Importowanie biblioteki klas interfejsu JDBC.** Biblioteka klas interfejsu JDBC musi zostać zaimportowana w programie napisanym w języku Java. Odpowiednie klasy zostały nazwane `java.sql.*` — można je importować za pomocą polecenia zastosowanego w wierszu 1. rysunku 10.12. Musimy zaimportować także wszystkie pozostałe klasy Javy, które zawierają funkcje niezbędne dla działania naszego programu.
- (2) **Wczytywanie sterownika JDBC.** Zgodnie ze wspomnianymi już regułami, należy wczytać sterownik interfejsu JDBC (patrz wiersze od 4. do 7.). Przechwytywany w wierszu 5. wyjątek Javy występuje w sytuacji, gdy próba wczytania sterownika JDBC zakończyła się niepowodzeniem.
- (3) **Tworzenie odpowiednich zmiennych.** Konieczne jest stworzenie wszystkich potrzebnych zmiennych języka Java (patrz wiersze 8. i 9.).
- (4) **Obiekt typu Connection.** Musimy stworzyć **obiekt połączenia** (typu `Connection`) za pomocą funkcji `getConnection` należącej do klasy `DriverManager` biblioteki JDBC. W wierszach 12. i 13. tworzymy obiekt połączenia, wywołując funkcję `getConnection(ciągnURL)`, gdzie `ciągnURL` ma następującą postać:  
`jdbc:oracle:<typSterownika>:<kontoBazyDanych>/<hasło>`

Alternatywna postać wywołania funkcji `getConnection` jest następująca:

```
getConnection(url, kontoBazyDanych, hasło)
```

Dla każdego obiektu połączenia można ustawiać rozmaite właściwości — zdecydowana większość tych właściwości dotyczy jednak przetwarzania transakcji, które omówimy w rozdziale 21.

- (5) **Obiekt typu `PreparedStatement`.** W naszym programie tworzymy **obiekt polecenia**. W bibliotece JDBC istnieje podstawowa klasa tego typu poleceń (`Statement`) z dwiema wyspecjalizowanymi podklasami `PreparedStatement` oraz `CallableStatement`. Przykład z rysunku 10.13 ilustruje sposób tworzenia i wykorzystywania obiektów klasy `PreparedStatement`; możliwości praktycznego stosowania drugiego rodzaju obiektów klasy `Statement` przedstawiono w kolejnym przykładzie (patrz rysunek 10.13). W wierszu 14. rysunku 10.12 utworzono w zmiennej `stmt1` ciąg znaków zapytania z pojedynczym parametrem (reprezentowanym przez znak zapytania). W wierszu 15. tego samego listingu stworzyliśmy obiekt `p` typu `PreparedStatement` w oparciu o ciąg znaków zapytania umieszczony w zmiennej `stmt1` i połączenie reprezentowane przez obiekt `conn`. Zasadniczo, programista powinien wykorzystywać obiekty typu `PreparedStatement` w sytuacjach, gdy pojedyncze zapytanie ma być wykonywane *wielokrotnie*, ponieważ polecenie jest wtedy przygotowywane, weryfikowane i kompilowane tylko raz, co eliminuje dodatkowy koszt związany z wykonywaniem tego zapytania w przyszłości.
- (6) **Konfigurowanie parametrów polecenia.** Użyty w wierszu 14. rysunku 10.12 znak zapytania reprezentuje **parametr polecenia**, którego wartość jest określana w czasie wykonywania (zwykle z wykorzystaniem powiązanej zmiennej programu napisanego w języku Java). Zgodnie z naszymi wcześniejszymi rozważaniami, pojedyncze polecenie może zawierać wiele parametrów odróżnianych wyłącznie w oparciu o kolejność występowania odpowiednich znaków zapytania (pierwszy znak zapytania reprezentuje pierwszy parametr, drugi znak zapytania — drugi parametr itd.).
- (7) **Wiązanie parametrów polecenia.** Zanim będzie możliwe wykonanie zapytania reprezentowanego przez obiekt typu `PreparedStatement`, wszystkie użyte w tym zapytaniu parametry musimy powiązać ze zmiennymi programu. W zależności od typu parametrów, powinniśmy dla obiektu typu `PreparedStatement` stosować takie funkcje jak `setString` (dla ciągów znaków), `setInteger` (dla liczb całkowitych), `setDouble` (dla liczb zmiennoprzecinkowych) itp. Należy zastosować funkcję dostosowaną do typu danych ustawianego parametru. Na rysunku 10.12 parametr (oznaczony znakiem zapytania) w zapytaniu reprezentowanym przez obiekt `p` wiążemy w wierszu 18. ze zmienną `pesel` programu napisanego w języku Java. Używana jest tu funkcja `setString`, ponieważ zmienna `pesel` to ciąg znaków. Gdyby w jednym poleceniu języka SQL występowało wiele różnych parametrów, powinniśmy użyć  $n$  wywołań funkcji `set...`, po jednym dla każdej pozycji parametru (1, 2, ...,  $n$ ). Generalnie zaleca się wyczyszczenie wszystkich parametrów przed ustawieniem jakichkolwiek nowych wartości (patrz wiersz 17. rysunku 10.12).

*// Fragment JDBC2 programu:*

```
0) import java.io.*;
1) import java.sql.*;
...
```



```
2) class printDepartmentEmps {
3)     public static void main(String args[]) throws SQLException, IOException {
4)         try { Class.forName("oracle.jdbc.driver.OracleDriver");
5)         } catch (ClassNotFoundException x) {
6)             System.out.println("Wczytanie sterownika jest niemożliwe");
7)         }
8)         String dbacct, password, nazwisko;
9)         double pensja;
10)        Integer nrdez;
11)        dbacct = readentry("Wpisz nazwę konta w bazie danych: ");
12)        password = readentry("Wpisz hasło do bazy danych: ");
13)        Connection conn = DriverManager.getConnection
14)            ("jdbc:oracle:oci8:" + dbacct + "/" + password);
15)        nrdez = new Integer(readentry("Wpisz numer działu: "));
16)        String q = "select NAZWISKO, PENSJA from PRACOWNIK where NRDEZ = ?" +
17)            nrdez.toString();
18)        Statement s = conn.createStatement();
19)        ResultSet r = s.executeQuery(q);
20)        while (r.next()) {
21)            nazwisko = r.getString(1);
22)            pensja = r.getDouble(2);
23)            System.out.println(nazwisko + pensja);
24)        } }
```

RYSUNEK 10.13. Fragment JDBC2 programu napisanego w języku Java, który wykorzystuje interfejs JDBC do wykonania w bazie danych zapytania zwracającego zbiór krotek

- (8) **Wykonywanie poleceń języka SQL.** Po wykonaniu kroków przygotowawczych możemy wreszcie użyć funkcji `executeQuery` (patrz wiersz 19.) do wykonania polecenia języka SQL wskazywanego przez obiekt `p`. Biblioteka JDBC udostępnia uniwersalną funkcję `execute` oraz dwie wyspecjalizowane funkcje `executeUpdate` i `executeQuery`. Funkcja `executeUpdate` służy do wykonywania takich poleceń języka SQL jak wstawianie, usuwanie oraz aktualizacja danych — funkcja zwraca wartość całkowitoliczbową określającą liczbę przetworzonych (dodanych, usuniętych lub zaktualizowanych) krotek. Funkcja `executeQuery` została zaprojektowana z myślą o wykonywaniu poleceń języka SQL wyszukujących dane — funkcja zwraca obiekt typu `ResultSet` (patrz kolejny punkt).
- (9) **Przetwarzanie obiektu typu `ResultSet`.** W wierszu 19. rysunku 10.12 zapisujemy wynik zapytania w obiekcie `r` typu `ResultSet`. Taki sposób przechowywania otrzymanych wyników przypomina dwuwymiarową tablicę lub tabelę, w której krotki mają postać kolejnych wierszy, natomiast atrybuty są reprezentowane przez kolumny. Obiekt typu `ResultSet` pełni podobną rolę jak kursor w osadzonym kodzie języka SQL oraz iterator w standardzie SQLJ. W analizowanym przykładzie obiekt `r` bezpośrednio po wykonaniu zapytania wskazuje na krotkę przed pierwszą krotką relacji wynikowej tego zapytania. Funkcja `r.next()` (patrz wiersz 20.) przechodzi do kolejnej krotki (kolejnego wiersza) w obiekcie `ResultSet` i zwraca wartość pustą (`null`), jeśli zbiór krotek został wyczerpany. Ta ostatnia własność służy do kontrolowania pętli. Programista może się odwoływać do atrybutów bieżącej krotki za pomocą rozmaitych funkcji `get...` (w zależności od typu atrybutów; przykładowo, programista



ma do dyspozycji funkcje `getString`, `getInteger`, `getDouble` itp.). W odwołaniach do otrzymanych wartości za pomocą funkcji `get...` można wykorzystywać albo pozycje atrybutów (1, 2), albo ich rzeczywiste nazwy ("NAZWISKO", "PENSJA"). W prezentowanym przykładzie wykorzystujemy notację opartą na pozycjach atrybutów (patrz wiersze 21. i 22.).

Zasadniczo, programista może sprawdzać ewentualne wystąpienia wyjątków języka SQL po każdym wywołaniu funkcji biblioteki JDBC. Aby uprościć przykłady, tu nie stosujemy tego rozwiązania.

Warto pamiętać, że w przeciwieństwie do innych technik omawianych w tym podrozdziale, interfejs JDBC jednakowo traktuje zapytania zwracające pojedyncze krotki i zapytania, które zwracają relacje wynikowe składające się z wielu krotek. Takie podejście jest uzasadnione, ponieważ jednoelementowy zbiór krotek można traktować jako specjalny przypadek.

W przykładowym fragmencie JDBC1 użyte zapytanie języka SQL zwracało *pojedynczą krotkę*, zatem pętla zdefiniowana w wierszach od 20. do 24. rysunku 10.12 była wykonywana najwyżej raz. Nasz kolejny przykład (patrz rysunek 10.13) ilustruje praktyczną metodę obsługi zapytania zwracającego wiele krotek. Fragment programu JDBC2 odczytuje na wejściu numer działu i wyszukuje informacje o wszystkich zatrudnionych w nim pracownikach. Następnie pętla przeszukuje w kolejnych iteracjach wszystkie rekordy reprezentujące pracowników (po jednym w każdej iteracji) i wyświetla ich nazwiska i wysokości pensji. Przedstawiony przykład ilustruje także sposób, w jaki możemy *wykonywać zapytania bezpośrednio*, a więc z pominięciem fazy jego przygotowywania (jak w poprzednim przykładzie). Zastosowana technika jest zalecana w przypadku zapytań, które będą wykonywane tylko raz, ponieważ można w ten sposób uprościć kod. W wierszu 17. rysunku 10.13 programista stworzył obiekt klasy `Statement` (zamiast użytej w poprzednim przykładzie klasy `PreparedStatement`), którego nie związał z żadnym konkretnym ciągiem znaków zapytania. Ciąg *q* jest *przekazywany do obiektu polecenia s* dopiero w momencie jego wykonywania (w wierszu 18.).

Przykładowy fragment JDBC2 kończy nasze krótkie wprowadzenie biblioteki JDBC. Czytelnicy zainteresowani tą technologią powinni odwiedzić witrynę internetową <http://java.sun.com/docs/books/tutorial/jdbc/>, która zawiera wiele szczegółowych informacji na temat JDBC.

## 10.4. Procedury składowane w bazie danych i technika SQL/PSM

Ten podrozdział to omówienie dwóch dodatkowych zagadnień związanych z programowaniem baz danych. W punkcie 10.4.1 przedstawimy pojęcie procedur składowanych, które są modułami programowymi przechowywanymi przez system zarządzania bazą danych na serwerze bazy danych. Następnie, w punkcie 10.4.2, omówimy rozszerzenia języka SQL, które zostały uwzględnione w specyfikacji tego standardu i które wprowadzają do języka SQL konstrukcje typowe dla uniwersalnych języków programowania. Rozszerzenia te są znane pod nazwą *SQL/PSM* (ang. *SQL/ Persistent Stored Modules*) i mogą być wykorzystywane

między innymi do tworzenia procedur składowanych. Standard SQL/PSM jest także przykładem języka programowania baz danych, który stanowi rozszerzenie tradycyjnego modelu i języka baz danych (dokładnie języka SQL) o pewne konstrukcje programistyczne, w tym instrukcje warunkowe i pętle.

### 10.4.1. Procedury i funkcje składowane w bazie danych

W naszej dotychczasowej prezentacji technik programowania baz danych opieraliśmy się na założeniu, że program aplikacji bazy danych działa na komputerze klienta lub, co bardziej prawdopodobne, *na maszynie z serwerem aplikacji* w warstwie środkowej trójwarstwowej architektury klient-serwer (patrz punkt 2.5.4 i rysunek 2.7). W obu sytuacjach maszyna, gdzie wykonywany jest program, nie jest jednocześnie komputerem zawierającym zainstalowany serwer bazy danych (a więc zasadniczą część pakietu oprogramowania systemu zarządzania bazą danych). Chociaż w wielu systemach takie rozwiązanie jest uzasadnione, w niektórych przypadkach warto rozważyć możliwość stworzenia modułów (procedur lub funkcji) programowych bazy danych, które będą składowane i wykonywane przez działający na serwerze bazy danych system zarządzania bazą danych. Co prawda takie moduły mogą mieć postać zarówno procedur, jak i funkcji, zwyczajowo nazywa się je jednak **procedurami składowanymi**. Pojęciem stosowanym w standardzie SQL do określania procedur składowanych są **moduły trwale składowane** (ang. *persistent stored modules*), ponieważ wyznaczone fragmenty programów są trwale przechowywane przez system zarządzania bazą danych, a więc podobnie jak dane trwale przechowywane w systemach tego typu.

Procedury składowane są przydatne w następujących okolicznościach:

- Jeśli program bazy danych ma być wykorzystywany przez wiele aplikacji, można go umieścić na serwerze, aby mógł być wywoływany przez wszystkie programy aplikacji. W ten sposób możemy wyeliminować koszty związane z wielokrotnym wykonywaniem tych samych działań i zwiększyć modularność oprogramowania.
- Wykonanie programu na serwerze może się przyczynić do ograniczenia ilości przesyłanych danych, a więc w pewnych sytuacjach zmniejszyć koszt komunikacji pomiędzy klientem a serwerem.
- Procedury składowane mogą stanowić przydatne rozszerzenie możliwości w zakresie modelowania w oparciu o perspektywy, ponieważ umożliwiają udostępnianie użytkownikom bazy danych bardziej skomplikowanych typów danych pochodnych. Co więcej, takie procedury mogą być wykorzystywane np. do weryfikowania skomplikowanych ograniczeń, których nie można definiować w oparciu o zwykłe asercje bądź wyzwalacze.

Wiele komercyjnych systemów zarządzania bazami danych oferuje możliwość tworzenia składowanych procedur i funkcji w uniwersalnych językach programowania. Alternatywnym rozwiązaniem jest konstruowanie procedur składowanych z wykorzystaniem takich poleceń języka SQL jak proste wyszukiwanie czy aktualizowanie danych. Ogólna postać polecenia deklarującego procedurę składowaną jest następująca:

```
CREATE PROCEDURE <nazwa procedury> (<parametry>)  
<deklaracje lokalne>  
<ciało procedury>;
```

Parametry i deklaracje lokalne są opcjonalne — można je określać tylko wtedy, gdy są niezbędne. W deklaracji funkcji składowanej musimy dodatkowo określić typ zwracanej wartości; deklaracji funkcji składowanej wygląda tak:

```
CREATE FUNCTION <nazwa funkcji> (<parametry>)  
RETURNS <zwracany typ>  
<deklaracje lokalne>  
<ciało procedury>;
```

Jeśli procedura (lub funkcja) została napisana w uniwersalnym języku programowania, w jej deklaracji zwykle określa się nie tylko ten język, ale także nazwę pliku, w którym znajduje się odpowiedni kod programu. Przykładowo, do zadeklarowania takiej procedury można użyć polecenia w następującej postaci:

```
CREATE PROCEDURE <nazwa procedury> (<parametry>)  
LANGUAGE <nazwa języka programowania>  
EXTERNAL NAME <ścieżka do pliku z programem>;
```

W zasadzie, każdy parametr powinien mieć określony **typ parametru** będący jednym z typów danych obsługiwanych w języku SQL. Każdy parametr powinien także mieć określony **tryb parametru**, który jest reprezentowany przez słowo kluczowe IN, OUT lub INOUT. Wymienione tryby odpowiadają parametrom, których wartości są odpowiednio przekazywane tylko na wejściu, przekazywane tylko na wyjściu (a więc zwracane) oraz przekazywane zarówno na wejściu, jak i na wyjściu.

Z uwagi na fakt, że procedury lub funkcje są trwale składowane przez system zarządzania bazą danych, powinna istnieć możliwość ich wywoływania za pośrednictwem różnych interfejsów języka SQL i z wykorzystaniem różnych technik programowania. Do wywoływania procedur składowanych (niezależnie od tego, czy z poziomu interaktywnego interfejsu, czy osadzonego kodu języka SQL lub SQLJ) służy w języku SQL **polecenie CALL**. Format tego polecenia jest następujący:

```
CALL <nazwa funkcji lub procedury> (<lista argumentów>);
```

Jeśli powyższe polecenie ma zostać wywołane za pośrednictwem interfejsu JDBC, powinniśmy je przypisać do obiektu typu CallableStatement (patrz punkt 10.3.2).

## 10.4.2. SQL/PSM: Rozszerzenie standardu SQL o możliwość określania trwale składowanych modułów

Mechanizm SQL/PSM jest częścią standardu SQL, która określa metodę pisania i wykorzystywania trwale składowanych modułów. Wspomniana specyfikacja obejmuje między innymi omówione w poprzednim punkcie polecenia przeznaczone do tworzenia funkcji i procedur. Ważnym elementem tej specyfikacji jest także wprowadzenie dodatkowych konstrukcji programistycznych, które rozszerzają możliwości języka SQL w obszarze definiowania kodu (ciała) składowanych procedur i funkcji.

W tym punkcie omówimy konstrukcje standardu SQL/PSM wykorzystywane do tworzenia instrukcji warunkowych (rozgałęzień) oraz pętli. Dzięki nim Czytelnik będzie mógł lepiej zrozumieć, jakiego rodzaju konstrukcje ma do dyspozycji podczas programowania

baz danych z wykorzystaniem tego standardu<sup>20</sup>. Potem przedstawimy przykład, który dobrze ilustruje praktyczne zastosowanie tych konstrukcji.

Instrukcja warunkowa standardu SQL/PSM ma następującą postać:

```
IF <warunek> THEN <lista poleceń>
  ELSEIF <warunek> THEN <lista poleceń>
  ...
  ELSEIF <warunek> THEN <lista poleceń>
  ELSE <lista poleceń>
END IF;
```

Przeanalizujemy teraz przykład przedstawiony na rysunku 10.14, który ilustruje sposób wykorzystania struktury warunkowego rozgałęzienia w funkcji zdefiniowanej zgodnie ze standardem SQL/PSM. Zaprezentowana funkcja zwraca wartość znakową (patrz wiersz 1.) opisującą rozmiar danego na wejściu działu określony w oparciu o liczbę zatrudnionych w nim pracowników. Funkcja pobiera jeden parametr wejściowy (oznaczony słowem kluczowym IN), nrdziału, który zawiera numer danego działu firmy. W wierszu 2. zadeklarowano zmienną lokalną LiczbaPrac. Zapytanie zdefiniowane w wierszach 3. i 4. zwraca liczbę pracowników zatrudnionych w danym dziale, natomiast instrukcja warunkowa z wierszy od 5. do 8. zwraca jedną z wartości {"OGROMNY", "DUŻY", "ŚREDNI", "MAŁY"} w zależności od znalezionej liczby pracowników.

```
// Funkcja PSM1:
0) CREATE FUNCTION RozmiarDziału(IN nrdziału INTEGER)
1) RETURNS VARCHAR[7];
2) DECLARE LiczbaPrac INTEGER;
3) SELECT COUNT(*) INTO LiczbaPrac
4) FROM PRACOWNIK WHERE NRDZ = nrdziału;
5) IF LiczbaPrac > 100 THEN RETURN "OGROMNY"
6)   ELSEIF LiczbaPrac > 25 THEN RETURN "DUŻY"
7)   ELSEIF LiczbaPrac > 10 THEN RETURN "ŚREDNI"
8)   ELSE RETURN "MAŁY"
9) END IF;
```

RYSUNEK 10.14. Przykład deklarowania funkcji w standardzie SQL/PSM

Standard SQL/PSM definiuje wiele konstrukcji do tworzenia pętli. Specyfikacja tego standardu obejmuje między innymi popularne struktury WHILE i REPEAT, których ogólna postać jest następująca:

```
WHILE <warunek> DO
  <lista poleceń>
END WHILE;
REPEAT
  <lista poleceń>
UNTIL <warunek>
END REPEAT;
```

<sup>20</sup> W tym punkcie przedstawimy tylko skrócone wprowadzenie do standardu SQL/PSM. Standard ten oferuje także wiele innych ciekawych elementów.

Dostępne są też struktury pętli opartych na kursorach. Lista poleceń zdefiniowanych w ciele takiej pętli jest wykonywana dla każdej krotki w relacji wynikowej zapytania. Taka pętla ma następującą postać:

```
FOR <nazwa pętli> AS <nazwa kursora> CURSOR FOR <zapytanie> DO  
    <lista poleceń>  
END FOR;
```

Pętle mogą mieć przypisywane nazwy — zastosowanie polecenia `LEAVE <nazwa pętli>` powoduje przerwanie wykonywania pętli, mimo że warunek definiujący jej zakres może być spełniony. Standard SQL/PSM ma wiele innych elementów, jednak ich omawianie wykracza poza zakres tematyczny tej książki.

## 10.5. Porównanie trzech opisanych podejść

W tym podrozdziale pokrótce porównamy trzy strategie programowania baz danych i omówimy wady oraz zalety każdego z nich.

- (1) **Osadzanie kodu w języku SQL.** Główną zaletą tego podejścia jest to, że tekst zapytania stanowi część kodu źródłowego programu. Dlatego zapytanie można w czasie kompilacji sprawdzić pod kątem błędów składni i zgodności ze schematem bazy. Dzięki temu program jest też czytelny, ponieważ zapytania są widoczne w kodzie źródłowym. Najważniejszymi wadami są utrata swobody modyfikowania zapytania w czasie wykonywania programu i to, że wszystkie zmiany w zapytaniach wymagają całego procesu ponownej kompilacji. Ponieważ zapytania są znane z góry, wybór zmiennych programu przechowujących wyniki zapytania jest prosty, co ułatwia programowanie aplikacji. Jednak w złożonych aplikacjach, gdzie zapytania muszą być generowane w czasie wykonywania programu, lepsze jest podejście z wywołaniami funkcji.
- (2) **Strategia wykorzystująca biblioteki klas i wywołania funkcji.** To podejście zapewnia większą swobodę, ponieważ zapytania można w razie potrzeby generować w czasie wykonywania programu. To jednak komplikuje programowanie, ponieważ zmienne programu odpowiadające kolumnom wyników zapytania mogą nie być z góry znane. Ponieważ zapytania są przekazywane jako tekst polecenia w wywołaniach funkcji, nie można sprawdzać poprawności w czasie kompilacji. Sprawdzanie składni i poprawności zapytań musi się odbywać w czasie wykonywania programu, w ramach przygotowywania zapytania. Programista musi też sprawdzać i uwzględniać w kodzie dodatkowe błędy możliwe w czasie wykonywania programu.
- (3) **Strategia wykorzystująca język programowania baz danych.** W tym podejściu nie występuje problem niezgodności impedancji, ponieważ typy danych w języku programowania są takie same jak w bazie danych. Jednak programiści muszą opanować nowy język, zamiast posługiwać się tym, który już znają. Ponadto niektóre języki programowania baz danych są specyficzne dla producenta, natomiast uniwersalne języki mogą współdziałać z systemami wielu firm.

## 10.6. Podsumowanie

W tym rozdziale zaprezentowaliśmy dodatkowe właściwości stosowanego w bazach danych języka SQL. W szczególności, w podrozdziale 10.1 przedstawiliśmy przegląd najważniejszych technik programowania baz danych. Następnie (w podrozdziałach od 10.2 do 10.4) przeanalizowaliśmy różne podejścia do programowania aplikacji baz danych.

W podrozdziale 10.2 omówiliśmy ogólną technikę osadzania kodu napisanego w języku SQL. Zapytania są w niej częścią kodu źródłowego programu. Do pobierania poleceń w języku SQL z programu (w celu ich przetwarzania przez SZBD) i zastępowania ich wywołaniami funkcji kierowanymi do skompilowanego kodu z SZBD służy zwykle prekompilator. Przedstawiliśmy tu przegląd osadzania kodu napisanego w języku SQL, używając w przykładach języka C jako języka nadrzędnego. Omówiliśmy też osadzanie kodu napisanego w języku SQL w programach Javy za pomocą technologii SQLJ. Zaprezentowaliśmy kursory (w osadzonym kodzie w języku SQL) i iteratory (w technologii SQLJ) oraz przykłady pokazujące, jak stosować te techniki do przetwarzania w pętli krotek z wyników zapytania, a także do pobierania wartości atrybutów do zmiennych programu w celu ich dalszego przetwarzania.

W podrozdziale 10.3 wyjaśniliśmy, jak uzyskać dostęp do opartych na języku SQL baz za pomocą bibliotek funkcji. Ta technika jest bardziej dynamiczna niż osadzanie kodu w języku SQL, jednak programowanie jest wtedy bardziej skomplikowane, ponieważ typy i liczba atrybutów w wynikach zapytania mogą być określane w czasie wykonywania programu. Przedstawiliśmy też przegląd standardu SQL/CLI. W przykładach językiem nadrzędnym był C. Omówiliśmy niektóre funkcje biblioteki SQL/CLI, przekazywanie zapytań w formie ciągów znaków, przypisywanie wartości parametrów zapytań w czasie wykonywania programu i zwracanie wyników do zmiennych programu. Następnie zademonstrowaliśmy używaną w języku Java bibliotekę klas JDBC oraz jej wybrane klasy i operacje. Klasa `ResultSet` służy do tworzenia obiektów przechowujących wyniki zapytań. Iteracyjne przeglądanie tych wyników jest możliwe za pomocą operacji `next()`. Opisaliśmy też funkcje z rodzin `get...` i `set...`, służące do pobierania wartości atrybutów i ustawiania wartości parametrów.

W podrozdziale 10.4 pokrótce omówiliśmy procedury składowane i przykładowy język programowania baz danych — SQL/PSM. Na zakończenie, w podrozdziale 10.5, krótko porównaliśmy trzy opisane strategie. Warto zauważyć, że wybraliśmy porównawczy przegląd trzech podstawowych strategii programowania baz danych, ponieważ szczegółowa analiza konkretnego podejścia zasługuje na odrębną książkę.

## Pytania powtórkowe

- 10.1. Czym jest interfejs ODBC? W jaki sposób jest on powiązany z interfejsem SQL/CLI?
- 10.2. Czym jest interfejs JDBC? Czy jest on związany z osadzonym językiem SQL, czy z wywołaniami funkcji?
- 10.3. Wymień trzy główne strategie programowania aplikacji baz danych. Jakie są wady i zalety każdej z tych strategii?

- 10.4. Na czym polega problem niezgodności impedancji? Która z trzech omówionych w tym rozdziale strategii programowania baz danych w największym stopniu minimalizuje ten problem?
- 10.5. Omów pojęcie kursora i wyjaśnij sposób jego wykorzystywania w osadzonych poleceniach języka SQL.
- 10.6. Do czego służy standard SQLJ? Opisz dwa typy iteratorów stosowanych w tym standardzie.

## Ćwiczenia

- 10.7. Przeanalizuj bazę danych przedstawioną na rysunku 1.2, której schemat zaprezentowaliśmy na rysunku 2.1. Napisz fragment programu, który pobierze na wejściu nazwisko studenta i wyświetli średnią jego ocen. Użyj techniki osadzania kodu języka SQL i wykorzystaj język C w roli nadrzędnego języka programowania.
- 10.8. Powtórz ćwiczenie 10.7, ale użyj tym razem technologii SQLJ i języka Java w roli nadrzędnego języka programowania.
- 10.9. Przeanalizuj schemat relacyjnej bazy danych BIBLIOTEKA z rysunku 6.6. Napisz fragment programu wyszukującego listę książek, których termin oddania upłynął wczoraj. Przygotowany program powinien dla każdej takiej książki wyświetlać tytuł oraz nazwisko czytelnika. Użyj techniki osadzania kodu języka SQL i wykorzystaj język C w roli nadrzędnego języka programowania.
- 10.10. Powtórz ćwiczenie 10.9, ale użyj tym razem technologii SQLJ i języka Java w roli nadrzędnego języka programowania.
- 10.11. Powtórz ćwiczenia 10.7 i 10.9, ale użyj tym razem standardu SQL/CLI oraz języka C w roli nadrzędnego języka programowania.
- 10.12. Powtórz ćwiczenia 10.7 i 10.9, ale użyj tym razem biblioteki JDBC oraz języka Java w roli nadrzędnego języka programowania.
- 10.13. Powtórz ćwiczenie 10.7, ale tym razem napisz składowaną funkcję w oparciu o standard SQL/PSM.
- 10.14. Napisz zgodną ze standardem PSM funkcję, która oblicza medianę pensji dla tabeli PRACOWNIK z rysunku 5.5.

## Wybrane publikacje

Jest wiele książek opisujących różne aspekty programowania baz danych z użyciem języka SQL. Przykładowo, Sunderraman (2007) omówił programowanie w SZBD Oracle 10g, a Reese (1997) skoncentrował się na interfejsie JDBC i programowaniu w języku Java. Dostępne są też liczne źródła internetowe.





# Programowanie internetowych baz danych z użyciem języka PHP

W poprzednim rozdziale przedstawiliśmy techniki programowania baz danych z użyciem tradycyjnych języków. W przykładach posługiwaliśmy się Javą i C. Teraz skupimy się na tym, jak uzyskać dostęp do baz z poziomu języków skryptowych. Wiele aplikacji internetowych oferujących interfejsy WWW do dostępu do informacji przechowywanych w bazach jest opartych na językach skryptowych. Te języki często służą do generowania dokumentów w języku HTML, które są następnie wyświetlane przez przeglądarki na potrzeby interakcji z użytkownikiem. W omówieniu zakładamy, że Czytelnicy znają podstawy języka HTML.

Podstawowe elementy języka HTML są przydatne do generowania *statycznych* stron WWW ze stałym tekstem i innymi obiektami. Jednak większość aplikacji internetowych wymaga stron WWW udostępniających użytkownikom interaktywne mechanizmy. Rozważmy klienta linii lotniczych, który chce sprawdzić czas przylotu i informacje o bramce dla konkretnego lotu. Użytkownik może wprowadzić np. datę i numer lotu w określonych polach strony WWW. Interfejs internetowy prześle wtedy te informacje do programu aplikacji, który uformuje i prześle zapytanie do serwera bazy danych linii lotniczych, aby pobrać dane potrzebne klientowi. Informacje z bazy są przesyłane z powrotem do strony WWW w celu ich wyświetlenia. Takie strony WWW, w których część informacji jest pobierana z baz lub innych źródeł danych, to strony *dynamiczne*. Pobierane i wyświetlane dane dotyczą wtedy za każdym razem innych lotów i dat.

Istnieją różne techniki programowania dynamicznych mechanizmów na stronach WWW. Tu skoncentrujemy się na jednej technice, opartej na PHP — otwartym języku skryptowym działającym po stronie serwera. Pierwotnie akronim PHP oznaczał *Personal Home Page*, jednak obecnie rozwijany jest do postaci *PHP Hypertext Processor*. PHP jest bardzo popularny. Interpretery tego języka są dostępne bezpłatnie i zostały napisane w języku C, dzięki czemu mogą działać w większości systemów komputerowych. Interpreter języka PHP zapewnia preprocesor hipertekstu, który wykonuje polecenia języka PHP z pliku tekstowego i tworzy pożądany plik w języku HTML. Aby uzyskać dostęp do baz, w interpreterze języka PHP trzeba dołączyć bibliotekę funkcji tego języka, co opiszemy w podrozdziale 11.3. Programy w języku PHP są wykonywane na komputerze z serwerem WWW. Stanowi to różnicę w porównaniu z niektórymi innymi językami skryptowymi, takimi jak JavaScript, wykonywanymi na komputerze klienckim. Istnieje wiele popularnych języków skryptowych, które pozwalają na dostęp do danych i tworzenie dynamicznych stron WWW. Te języki to np.: JavaScript, Ruby, Python i PERL.

Ten rozdział jest uporządkowany w następujący sposób: w podrozdziale 11.1 przedstawiamy prosty przykład ilustrujący, jak można zastosować PHP. Podrozdział 11.2 zawiera ogólny przegląd języka PHP i tego, jak używać go do programowania wybranych podstawowych mechanizmów interaktywnych stron WWW. W podrozdziale 11.3 skupimy się na używaniu PHP do interakcji z opartymi na języku SQL bazami za pomocą biblioteki funkcji PEAR DB. W podrozdziale 11.4 wymienimy dodatkowe technologie powiązane z używaniem Javy do programowania rozwiązań internetowych i bazodanowych (technologie JDBC i SQLJ omówiliśmy już w rozdziale 10.). Podrozdział 11.5 zawiera podsumowanie rozdziału.

## 11.1. Prosty przykład zastosowania PHP

PHP to otwarty język skryptowy o ogólnym przeznaczeniu. Interpreter języka PHP jest napisany w C, dzięki czemu można go używać w prawie wszystkich rodzajach komputerów i systemów operacyjnych. W systemie operacyjnym UNIX PHP jest zwykle instalowany domyślnie. Dla platform z innymi systemami operacyjnymi (takimi jak Windows, Linux i macOS) interpreter języka PHP można pobrać ze strony <http://www.php.net>. PHP, podobnie jak inne języki skryptowe, wyjątkowo dobrze nadaje się do operowania na stronach tekstowych, a przede wszystkim na dynamicznych stronach HTML na komputerach z serwerem WWW. Różni się tym od JavaScriptu, ponieważ kod w JavaScriptcie jest pobierany razem ze stronami WWW w celu wykonywania go na komputerze klienckim.

PHP obejmuje biblioteki funkcji zapewniające dostęp do baz przechowywanych w różnych relacyjnych systemach baz danych, takich jak Oracle, MySQL, SQL Server i inne narzędzia obsługujące standard ODBC (patrz rozdział 10.). W architekturze trójwarstwowej (patrz rozdział 2.) SZBD działa na **serwerze bazy danych w dolnej warstwie**. PHP działa na **serwerze WWW w środkowej warstwie**, gdzie polecenia programu napisanego w PHP operują na plikach w HTML-u i tworzą niestandardowe dynamiczne strony WWW. Strona HTML jest następnie przesyłana do **warstwy klienta** w celu wyświetlenia i interakcji z użytkownikiem.

Przyjrzyj się przykładowemu kodowi PHP z rysunku 11.1(a). Ten kod wyświetla prośbę o wpisanie imienia i nazwiska, a następnie wyświetla użytkownikowi komunikat powitalny. Numery wierszy nie są częścią kodu programu; zostały dodane tylko na potrzeby objaśnień.

- (1) Załóżmy, że plik ze skryptem PHP z fragmentu F1 jest zapisany w następującej lokalizacji w internecie: <http://www.myserver.com/example/greeting.php>. Jeśli użytkownik wprowadzi ten adres w przeglądarce, interpreter PHP zacznie interpretować kod i wyświetli formularz widoczny na rysunku 11.1(b). Działanie to objaśnimy w trakcie omawiania wierszy fragmentu F1.
- (2) Wiersz 0. to znacznik początkowy PHP, `<?php`, informujący silnik interpretera, że powinien przetwarzać wszystkie kolejne wiersze tekstu do momentu napotkania znacznika końcowego PHP, `?>`, znajdującego się w wierszu 16. Tekst poza tymi znacznikami jest wyświetlany w pierwotnej postaci. Dzięki temu fragmenty kodu w PHP można umieszczać w większym pliku HTML. Preprocesor języka PHP przetwarza tylko fragmenty pliku umieszczone między znacznikami `<?php` i `?>`.

(a)

```
// Fragment F1:
0) <?php
1) // Wyświetlanie komunikatu powitalnego po tym, jak użytkownik
   // przesłał imię i nazwisko za pomocą formularza HTML
2) if ($_POST['user_name']) {
3)     print("Witaj, " );
4)     print($_POST['user_name']);
5) }
6) else {
7)     // Wyświetlanie formularza na imię i nazwisko użytkownika,
   // ponieważ dane nie zostały jeszcze wprowadzone
8)     print <<<_HTML_
9)     <FORM method="post" action="$_SERVER['PHP_SELF']">
10)    Wprowadź imię i nazwisko: <input type="text" name="user_name">
11)    <BR/>
12)    <INPUT type="submit" value="WYŚLIJ">
13)    </FORM>
14)    _HTML_;
15) }
16) ?>
```

(b)

Wprowadź imię i nazwisko:



(c)

Wprowadź imię i nazwisko:



(d)

Witaj, Jan Kowal

RYSUNEK 11.1. (a) Fragment programu w PHP służącego do wprowadzania danych. (b) Początkowy formularz wyświetlany przez fragment programu w PHP. (c) Użytkownik wpisuje nazwisko Jan Kowal. (d) Formularz wyświetla komunikat powitalny dla Jana Kowala

- (3) W wierszu 1. pokazano jeden ze sposobów dodawania komentarzy w programach w PHP; tu jest to komentarz jednowierszowy zaczynający się od znaków `//`. Komentarze jednowierszowe mogą też rozpoczynać się znakiem `#` i ciągnąć się tylko do końca wiersza, w którym się znajdują. Komentarze wielowierszowe zaczynają się znakami `/*` i kończą sekwencją `*/`.
- (4) Predefiniowana **automatyczna zmienna globalna** języka PHP `$_POST` (wiersz 2.) to tablica przechowująca wszystkie wartości wprowadzone za pomocą parametrów formularza. Tablice w PHP są *dynamiczne* (nie mają stałej liczby elementów). Można tworzyć tablice z indeksami liczbowymi, gdzie indeksy (pozycje) mają wartości (0, 1, 2, ...); używane są też tablice asocjacyjne, których indeksami mogą być dowolne wartości znakowe. Przykładowo, tablica asocjacyjna może mieć indeksy w postaci nazw kolorów `{"czerwony", "niebieski", "zielony"}`. W tym przykładzie zmienna `$_POST` ma indeks asocjacyjny w postaci nazwy przesłanej wartości `user_name`, określonej w atrybucie `name` znacznika `input` w wierszu 10. Tak więc element `$_POST['user_name']` zawiera wartość wprowadzoną przez użytkownika. Tablice języka PHP omówimy dokładnie w punkcie 11.2.2.

- (5) W momencie pierwszego otwarcia strony WWW *http://www.myserver.com/example/greeting.php* warunek `if` z wiersza 2. ma wartość `false`, ponieważ element `$_POST['user_name']` nie zawiera wartości. Dlatego interpreter języka PHP przetwarza wiersze od 6. do 15., tworząc tekst pliku HTML wyświetlającego formularz z rysunku 11.1(b). Plik ten jest następnie wyświetlany po stronie klienta w przeglądarce.
- (6) W wierszu 8. pokazany jest jeden ze sposobów tworzenia **długich ciągów znaków** w plikach HTML. Inne sposoby tworzenia ciągów znaków omówimy dalej w tym podrozdziale. Cały tekst między otwierającym znacznikiem `<<<_HTML_` a zamykającym znacznikiem `_HTML_>` jest zapisywany w dosłownej postaci w pliku HTML. Zamykający znacznik `_HTML_>` musi występować samodzielnie w odrębnym wierszu. Tak więc w pliku HTML przesyłanym do klienta znajdzie się tekst podany w wierszach od 9. do 13. Obejmuje on znaczniki języka HTML potrzebne do utworzenia formularza z rysunku 11.1(b).
- (7) **Nazwy zmiennych** w PHP zaczynają się od znaku `$` i mogą się składać ze znaków, cyfr i podkreśleń (`_`). Predefiniowana automatyczna zmienna globalna `$_SERVER` (wiersz 9.) to tablica z informacjami o lokalnym serwerze. Element `$_SERVER['PHP_SELF']` tej tablicy zawiera ścieżkę do pliku PHP wykonywanego obecnie na serwerze. Tak więc atrybut `action` znacznika `form` (wiersz 9.) nakazuje interpreterowi PHP ponowne przetworzenie tego samego pliku po wprowadzeniu przez użytkownika parametrów formularza.
- (8) Gdy użytkownik wpisze w polu tekstowym nazwisko *Jan Kowal* i kliknie przycisk **WYŚLIJ** (rysunek 11.1(c)), fragment F1 zostanie ponownie przetworzony. Tym razem element `$_POST['user_name']` będzie zawierał ciąg znaków `"Jan Kowal"`, dlatego wiersze 3. i 4. znajdą się w przesyłanym do klienta pliku HTML i wyświetlona zostanie wiadomość z rysunku 11.1(d).

Ten przykład dowodzi, że program w języku PHP może generować dwa różne polecenia w języku HTML w zależności od tego, czy użytkownik dopiero otworzył stronę, czy już przesłał imię i nazwisko za pomocą formularza. Program w języku PHP może tworzyć na serwerze wiele wersji pliku HTML w zależności od warunkowej ścieżki wybranej w programie. Dlatego kod HTML przesyłany do klienta będzie różny w zależności od interakcji z użytkownikiem. Jest to jeden ze sposobów, w jakie PHP można stosować do tworzenia *dynamicznych* stron WWW.

## 11.2. Przegląd podstawowych mechanizmów języka PHP

W tym podrozdziale przedstawiamy przegląd wybranych mechanizmów języka PHP przydatnych do tworzenia interaktywnych stron HTML. W podrozdziale 11.3 skupimy się na tym, jak programy w języku PHP mogą uzyskiwać dostęp do baz na potrzeby aktualizacji i zgłaszania zapytań. Nie możemy przedstawić tu kompletnego omówienia PHP. Dostępnych jest wiele książek poświęconych wyłącznie temu językowi. Skupimy się na opisie konkretnych funkcji języka PHP, wyjątkowo dobrze dostosowanych do tworzenia dynamicznych stron WWW z poleceniami dostępu do baz danych. W tym podrozdziale scharakteryzujemy pewne związane z językiem PHP zagadnienia i funkcje, które będą potrzebne przy omawianiu dostępu do baz w podrozdziale 11.3.

## 11.2.1. Zmienne, typy danych i konstrukcje programistyczne języka PHP

**Nazwy zmiennych** w PHP zaczynają się symbolem \$ i mogą obejmować cyfry, litery i podkreślenie (\_). Żadne inne znaki specjalne nie są dozwolone. Wielkość liter w nazwach zmiennych ma znaczenie, a pierwszym znakiem nie może być cyfra. Zmienne nie mają określonego typu. Typ jest ustalany na podstawie wartości przypisanej do zmiennej. Ta sama zmienna po przypisaniu do niej nowej wartości może przyjąć nowy typ. Przypisywanie odbywa się za pomocą operatora =.

Ponieważ PHP ma służyć głównie do przetwarzania tekstu, dostępnych jest kilka różnych typów ciągów znaków. Dostępnych jest też wiele funkcji do przetwarzania takich ciągów. Tu omawiamy tylko wybrane podstawowe cechy wartości i zmiennych znakovych. Na rysunku 11.2 przedstawione są przykładowe wartości znakowe. Istnieją trzy główne sposoby zapisywania ciągów znaków i tekstu:

- (1) **Ciągi znaków w apostrofach.** Umieść ciąg znaków w apostrofach, tak jak w wierszach 0., 1. i 2. Jeśli apostrof jest potrzebny wewnątrz ciągu znaków, należy zastosować znak ucieczki (\; patrz wiersz 2.).
- (2) **Ciągi znaków w cudzysłowach.** Umieść ciąg znaków w cudzysłowach, tak jak w wierszu 7. Tu *nazwy zmiennych występujące w ciągu znaków* są zastępowane wartościami przypisanymi obecnie do tych zmiennych. Interpreter identyfikuje nazwy zmiennych w ciągach znaków ujętych w cudzysłów, wykrywając początkowy znak \$, a następnie zastępuje te nazwy wartościami zmiennych. Ta technika to **interpolacja zmiennych** w ciągach znaków. Mechanizm ten nie jest stosowany w ciągach znaków w apostrofach.
- (3) **Dokumenty here.** Umieść fragment dokumentu po znaczniku <<<NAZWADOKUMENTU i zakończ go jednym wierszem obejmującym tę nazwę (NAZWADOKUMENTU). Jako NAZWADOKUMENTU można podać dowolny ciąg znaków, przy czym musi być on identyczny na początku i na końcu danego dokumentu here. Ta technika jest przedstawiona na rysunku 11.2 w wierszach od 8. do 11. Jeśli w dokumencie here występują zmienne, są zastępowane ich wartościami tekstowymi (w ramach interpolacji). Mechanizm ten działa w podobny sposób jak w ciągach znaków w cudzysłowach, jednak wygodniej jest stosować go w tekście wielowierszowym.
- (4) **Apostrofy i cudzysłowy.** Apostrofy i cudzysłowy używane w PHP wokół ciągów znaków powinny być *proste* ("). Edytor tekstu, który generuje te symbole, nie powinien tworzyć *zwykłych* (") cudzysłów wokół ciągów znaków.

```
0) print 'Witaj w mojej witrynie.';
1) print 'Powiedziałem mu: "Witaj w domu"';
2) print 'Byliśmy na festiwalu Opolo \'99';
3) printf('Cena wynosi $%.2f, a podatek $%.2f', $cost, $tax) ;
4) print strtolower('AbCdE');
5) print ucwords(strtolower('JAN kowal'));
6) print 'abc' . 'efg'
7) print "prześlij odpowiedź na adres: $email_address"
```

```
8) print <<<FORM_HTML
9) <FORM method="post" action="$ _SERVER['PHP_SELF']">
10) Wprowadź imię i nazwisko: <input type="text" name="user_name">
11) FORM_HTML
```

RYSUNEK 11.2. Podstawy dotyczące ciągów znaków i wartości tekstowych w PHP

Istnieje też operator złączania ciągów znaków (kropka: .), przedstawiony w wierszu 6. rysunku 11.2. Dostępnych jest wiele funkcji operujących na ciągach znaków. Tu prezentujemy tylko kilka z nich. Funkcja `strtolower` zmienia wszystkie litery ciągu znaków na małe, natomiast funkcja `ucwords` przekształca wszystkie litery na wielkie. Funkcje te zastosowano w wierszach 4. i 5. rysunku 11.2.

Zgodnie z ogólną regułą należy stosować ciągi znaków w apostrofach dla dosłownie podawanego tekstu, który nie obejmuje zmiennych programu napisanego w PHP, a dwie pozostałe techniki (ciągi znaków w cudzysłowach i dokumenty *here*), jeśli potrzebna jest interpolacja wartości zmiennych w ciągu znaków. Na potrzeby dużych bloków wielowierszowego tekstu w programie należy stosować *dokumenty here*.

W PHP dostępne są też numeryczne typy danych dla liczb całkowitych i zmiennoprzecinkowych. Przetwarzanie tych typów odbywa się zwykle zgodnie z regułami języka C. Liczby można formatować do postaci ciągu znaków na potrzeby wyświetlania i określać liczbę cyfr po przecinku. Odmiana funkcji `print`, `printf` (od ang. *print formatted*, czyli wyświetlaj sformatowane), pozwala formatować liczby w ciągu znaków. Ilustruje to wiersz 3. rysunku 11.2.

Dostępne są też standardowe konstrukty z języków programowania: pętle `for` i `while` oraz warunkowe instrukcje `if`. Ogólnie są one podobne do ich odpowiedników z języka C, dlatego nie będziemy ich tu omawiać. Ponadto prawie *dowolna wartość* używana jako wyrażenie logiczne jest traktowana jako `TRUE`; *wyjątkami* są tu zero (0) i pusty łańcuch znaków, traktowane jako `FALSE`. Używane są też literały `TRUE` i `FALSE`, które można przypisywać do zmiennych. Operatory porównania przeważnie działają jak w C. Są to operatory: `==` (równości), `!=` (nierówności), `>` (większe niż), `>=` (większe lub równe), `<` (mniejsze niż) i `<=` (mniejsze lub równe).

## 11.2.2. Tablice w PHP

Tablice są bardzo ważne w PHP, ponieważ pozwalają tworzyć listy elementów. Są często używane w formularzach obejmujących menu rozwijane. Do przechowywania list opcji w takich menu służy tablica jednowymiarowa. Na potrzeby wyników zapytań używane są tablice dwuwymiarowe, w których pierwszy wymiar reprezentuje *wiersze* tabeli, a drugi — *kolumny* (atrybuty) wierszy. Są dwa podstawowe rodzaje tablic: liczbowe i asocjacyjne. Dalej omówimy oba te typy w kontekście tablic jednowymiarowych.

**Tablice liczbowe** łączą z każdym elementem tablicy indeks liczbowy (inaczej: pozycję lub numer porządkowy). Indeksy są tu kolejnymi liczbami całkowitymi (począwszy od zera). Element tablicy wskazuje się za pomocą indeksu. **Tablica asocjacyjna** obejmuje pary (klucz => wartość). Wartość elementu jest podawana za pomocą klucza, a wszystkie klucze w danej tablicy muszą być unikatowe. Wartościami elementów mogą być ciągi znaków lub liczby całkowite, a także inne tablice, co pozwala tworzyć tablice wielowymiarowe.



Na rysunku 11.3 przedstawione są dwie przykładowe zmienne tablicowe: `$teaching` i `$courses`. Pierwsza z nich, `$teaching`, to tablica asocjacyjna (wiersz 0. rysunku 11.3). Każdy jej element łączy nazwę przedmiotu (klucz) z nazwiskiem wykładowcy (wartość). Ta tablica zawiera trzy elementy. W wierszu 1. pokazane jest, jak można ją zaktualizować. Pierwsze polecenie w wierszu 1. przypisuje nowego wykładowcę do przedmiotu 'grafika'; odbywa się to w wyniku aktualizacji wartości elementu. Ponieważ w tablicy znajduje się już klucz 'grafika', kod nie tworzy nowego elementu, a tylko aktualizuje istniejącą wartość. Drugie polecenie tworzy nowy element, ponieważ w tablicy nie występuje jeszcze klucz 'eksploracja danych'. Nowe elementy są dodawane na końcu tablicy.

Jeśli jako elementy tablicy podasz same wartości (bez kluczy), automatycznie powstaną klucze liczbowe 0, 1, 2 itd. Ilustruje to wiersz 5. rysunku 11.3 (tablica `$courses`). Oba typy tablic (asocjacyjne i liczbowe) mają nieograniczoną wielkość. Jeśli do zmiennej języka PHP przechowującej tablicę przypisana zostanie wartość innego typu danych niż we wcześniejszych elementach (np. liczba całkowita), w zmiennej zapisana zostanie ta nowa wartość, a dawna zawartość tablicy zostanie utracona. Do większości zmiennych można w każdym momencie przypisać wartość dowolnego typu.

W PHP dostępnych jest kilka różnych technik przetwarzania tablic w pętlach. Na rysunku 11.3 przedstawione są dwie takie metody. W wierszach 3. i 4. pokazana jest jedna technika: pobieranie wszystkich elementów tablicy za pomocą konstrukcji `foreach` i wyświetlanie w odrębnym wierszu klucza oraz wartości każdego elementu. W wierszach od 7. do 10. pokazano, jak zastosować tradycyjną pętlę `for`. Wbudowana funkcja `count` (wiersz 7.) zwraca aktualną liczbę elementów w tablicy, przypisywaną do zmiennej `$num` i używaną do kontroli zakończenia pracy pętli.

```
0) $teaching = array('bazy danych' => 'Szczepan', 'systemy operacyjne' => 'Carski',
                    'grafika' => 'Kamyk');
1) $teaching['grafika'] = 'Bender'; $teaching['eksploracja danych'] = 'Lipski';
2) sort($teaching);
3) foreach ($teaching as $key => $value) {
4)     print " $key : $value\n";}
5) $courses = array('bazy danych', 'systemy operacyjne',
                    'grafika', 'eksploracja danych');
6) $alt_row_color = array('blue', 'yellow');
7) for ($i = 0, $num = count($courses); $i < $num; $i++) {
8)     print '<TR bgcolor="' . $alt_row_color[$i % 2] . '">';
9)     print "<TD>Kurs nr $i to</TD><TD>$courses[$i]</TD></TR>\n";
10) }
```

RYSunek 11.3. Podstawy przetwarzania tablic w PHP

Kod z wierszy 7. – 10. ilustruje też, jak w języku HTML wyświetlać tabelę z różnymi kolorami naprzemiennych wierszy. W tablicy `$alt_row_color` (wiersz 6.) określono dwa kolory. W każdej iteracji pętli operacja modulo (`$i % 2`) wskazuje jeden kolor (indeks 0) lub drugi (indeks 1; patrz wiersz 8.). Kolor jest przypisywany do używanego w języku HTML atrybutu `bgcolor` znacznika wiersza tabeli `<TR>`.

Funkcja `count` (wiersz 7.) zwraca aktualną liczbę elementów tablicy. Funkcja `sort` (wiersz 2.) sortuje tablicę na podstawie wartości elementów (a nie według kluczy). W tablicach asocjacyjnych po sortowaniu każdy klucz pozostaje powiązany z tą samą wartością.



Po posortowaniu tablic liczbowych jest inaczej. Istnieje też wiele innych funkcji, które można stosować do tablic języka PHP, jednak kompletne omówienie tego tematu wykracza poza zakres bieżącego przeglądu.

### 11.2.3. Funkcje w języku PHP

W PHP, podobnie jak w innych językach programowania, można definiować **funkcje**, aby ulepszyć strukturę złożonych programów i utworzyć powtarzające się fragmenty kodu możliwe do wielokrotnego użytku w wielu aplikacjach. W nowszej wersji PHP (PHP5) dostępne są też mechanizmy obiektowe, jednak nie omawiamy ich w tym miejscu, ponieważ koncentrujemy się na podstawach tego języka. Podstawowe funkcje w PHP mogą przyjmować argumenty *przekazywane przez wartość*. W funkcjach dostępne są zmienne globalne. W funkcjach i w kodzie wywołującym te funkcje obowiązują standardowe reguły określania zasięgu zmiennych.

Przedstawimy teraz dwa proste przykłady ilustrujące podstawowe funkcje języka PHP. Na rysunku 11.4 pokazano, jak z wykorzystaniem funkcji zmodyfikować fragment F1 z rysunku 11.1(a). Fragment F1' na rysunku 11.4 obejmuje dwie funkcje: `display_welcome()` (wiersze od 0. do 3.) i `display_empty_form()` (wiersze od 5. do 13.). Żadna z tych funkcji nie przyjmuje argumentów ani nie zwraca wartości. W wierszach od 14. do 19. pokazano, jak wywołać te funkcje w celu uzyskania tych samych efektów co we fragmencie F1 z rysunku 11.1(a). W tym przykładzie widać, że funkcje pozwalają poprawić strukturę kodu w PHP i ułatwić jego zrozumienie.

```
// Fragment F1':
0) function display_welcome() {
1)     print("Witaj, ") ;
2)     print($_POST['user_name']);
3) }
4)
5) function display_empty_form(): {
6)     print <<< HTML_
7)     <FORM method="post" action="$ _SERVER['PHP_SELF']">
8)     Wprowadź imię i nazwisko: <INPUT type="text" name="user_name">
9)     <BR/>
10)    <INPUT type="submit" value="WYŚLIJ">
11)    </FORM>
12)    _HTML_ ;
13) }
14) if ($_POST['user_name']) {
15)     display_welcome();
16) }
17) else {
18)     display_empty_form();
19) }
```

RYSunek 11.4. Nowa, wykorzystująca funkcje wersja fragmentu F1: F1'

Drugi przykład jest pokazany na rysunku 11.5. Tu używana jest tablica `$teaching` z rysunku 11.3. Funkcja `course_instructor()` z wierszy od 0. do 8. rysunku 11.5 przyjmuje dwa argumenty: `$course` (ciąg znaków z nazwą przedmiotu) i `$teaching_assignments` (tablica asocjacyjna z wykładowcami poszczególnych przedmiotów, podobna do tablicy `$teaching` z rysunku 11.3). Wspomniana funkcja wyszukuje nazwisko wykładowcy określonego przedmiotu. W wierszach od 9. do 14. rysunku 11.5 pokazano, jak korzystać z tej funkcji.

```
0) function course_instructor ($course, $teaching_assignments) {
1)   if (array_key_exists($course, $teaching_assignments)) {
2)     $instructor = $teaching_assignments[$course];
3)     RETURN "$instructor prowadzi kurs $course";
4)   }
5)   else {
6)     RETURN "Kurs $course nie jest prowadzony";
7)   }
8) }
9) $teaching = array('bazy danych' => 'Szymczak', 'systemy operacyjne' => 'Carski',
                    'grafika' => 'Kamyk');
10) $teaching['grafika'] = 'Bender'; $teaching['eksploracja danych'] = 'Lipski';
11) $x = course_instructor('bazy danych', $teaching);
12) print($x);
13) $x = course_instructor('architektura komputerów', $teaching);
14) print($x);
```

RYSUNEK 11.5. Funkcja przyjmująca argumenty i zwracająca wartość

Wywołanie funkcji w wierszu 11. zwraca ciąg znaków: *Szymczak prowadzi kurs bazy danych*, ponieważ w tablicy klucz 'bazy danych' jest powiązany z określającą wykładowcę wartością 'Szymczak'. Z kolei wywołanie funkcji w wierszu 13. zwraca ciąg znaków *Kurs architektura komputerów nie jest prowadzony*, ponieważ w tablicy nie występuje element z kluczem 'architektura komputerów'. Oto kilka komentarzy na temat przedstawionego przykładu i funkcji w języku PHP:

- Wbudowana w PHP funkcja do obsługi tablic `array_key_exists($k, $a)` zwraca wartość `TRUE`, jeśli wartość zmiennej `$k` *istnieje jako klucz tablicy asocjacyjnej* zapisanej w zmiennej `$a`. W przykładzie kod sprawdza, czy podana wartość zmiennej `$course` występuje jako klucz tablicy `$teaching_assignments` (wiersz 1. rysunku 11.5).
- Argumenty funkcji są przekazywane przez wartość. Dlatego w tym przykładzie wywołania w wierszach 11. i 13. nie mogą modyfikować tablicy `$teaching` przekazanej jako argument wywołania. Wartości przekazane w wywołaniu funkcji są wtedy przekazywane (kopiowane) do jej argumentów.
- Wartości zwracane przez funkcję są podawane po słowie kluczowym `RETURN`. Funkcja może zwracać wartość dowolnego typu. W tym przykładzie funkcja zwraca ciąg znaków. Tu w zależności od tego, czy podana wartość klucza, `$course`, występuje w tablicy, czy nie, zwracane są dwa różne ciągi znaków.

- Reguły określania zasięgu nazw zmiennych są takie jak w innych językach programowania. Zmienne globalne spoza funkcji zwykle nie mogą być używane, chyba że zostaną wskazane za pomocą wbudowanej tablicy języka PHP `$GLOBALS`. Wyrażenie `$GLOBALS['abc']` zapewnia dostęp do wartości zmiennej globalnej `$abc` zdefiniowanej poza daną funkcją. W innych sytuacjach zmienne występujące w funkcjach są lokalne — nawet jeśli istnieje zmienna globalna o takiej samej nazwie.

Przedstawiliśmy krótki przegląd funkcji w języku PHP. Wiele detali zostało pominiętych, ponieważ szczegółowe omawianie PHP nie jest naszym celem.

## 11.2.4. Zmienne i formularze serwera PHP

We wbudowanej automatycznej globalnej zmiennej tablicowej języka PHP `$_SERVER` znajduje się wiele elementów, które zapewniają programistom użyteczne informacje na temat serwera, gdzie działa interpreter języka PHP, a także zawierają inne dane. Takie informacje mogą być potrzebne w trakcie generowania tekstu dokumentu HTML (patrz wiersz 7. rysunku 11.4). Oto niektóre z dostępnych elementów:

- (1) `$_SERVER['SERVER_NAME']`. Ten element udostępnia nazwę witryny (adres URL) serwera, na którym działa interpreter języka PHP. Przykładowo, jeśli interpreter działa w witrynie `http://www.uta.edu/`, ten adres będzie wartością elementu `$_SERVER['SERVER_NAME']`.
- (2) `$_SERVER['REMOTE_ADDRESS']`. Jest to adres IP należącego do użytkownika komputera klienckiego, który uzyskał dostęp do serwera (np. adres 129.107.61.8).
- (3) `$_SERVER['REMOTE_HOST']`. Jest to nazwa witryny (adres URL) należącego do użytkownika komputera klienckiego, np. `abc.uta.edu`. Serwer musi przekształcić taką nazwę na adres IP, aby uzyskać dostęp do klienta.
- (4) `$_SERVER['PATH_INFO']`. Zwraca końcową część adresu URL znajdującą się po lewym ukośniku (/).
- (5) `$_SERVER['QUERY_STRING']`. Zwraca ciąg znaków przechowujący parametry w końcowej części adresu URL, po znaku zapytania (?). Może obejmować np. parametry wyszukiwania.
- (6) `$_SERVER['DOCUMENT_ROOT']`. Przechowuje nazwę katalogu głównego z plikami na serwerze WWW, które są dostępne dla klientów.

Te i inne elementy tablicy `$_SERVER` są zwykle potrzebne w trakcie tworzenia pliku HTML wysyłanego w celu wyświetlenia klientowi.

Inną tablicową automatyczną globalną zmienną wbudowaną w języku PHP jest `$_POST`. Udostępnia ona programiście wartości wejściowe przesłane przez użytkownika za pomocą formularzy HTML i określone za pomocą znacznika `<INPUT>` i podobnych znaczników języka HTML. Przykładowo, na rysunku 11.4 w wierszu 14. zmienna `$_POST['user_name']` zapewnia programiście wartość wpisaną przez użytkownika w formularzu HTML i powiązaną ze znacznikiem `<INPUT>` z wiersza 8. tego rysunku. Kluczami omawianej tablicy są nazwy różnych parametrów wejściowych przekazanych za pomocą danego formularza, np. wartość atrybutu `name` znacznika HTML `<INPUT>` w wierszu 8. Gdy użytkownicy wprowadzają dane za pomocą formularzy, wartości danych są zapisywane we wspomnianej tablicy.

## 11.3. Przegląd programowania baz danych za pomocą PHP

Znane są różne techniki dostępu do bazy danych za pomocą języka programowania. Niektóre z nich opisaliśmy w rozdziale 10., gdzie pokazaliśmy, jak za pomocą języków C i Java uzyskać dostęp do baz danych opartych na języku SQL. Omówiliśmy przede wszystkim osadzany SQL oraz technologie JDBC, SQL/CLI (podobna do ODBC) i SQLJ. W tym podrozdziale pokazujemy, jak uzyskać dostęp do bazy danych za pomocą języka skryptowego PHP, który nadaje się do tworzenia interfejsów internetowych służących do wyszukiwania i aktualizowania baz danych; PHP jest odpowiedni także do budowania dynamicznych stron WWW.

Istnieje biblioteka PHP z funkcjami dającymi dostęp do baz danych. Stanowi ona część repozytorium PEAR (ang. *PHP Extension and Application Repository*), które jest kolekcją kilku bibliotek funkcji wzbogacających PHP. Biblioteka PEAR DB obejmuje funkcje zapewniające dostęp do baz danych. Za pomocą tej biblioteki można uzyskać dostęp do wielu systemów baz danych, w tym Oracle, MySQL, SQLite i Microsoft SQL Server.

W przykładach przedstawimy kilka różnych funkcji biblioteki PEAR DB. W punkcie 11.3.1 pokażemy, jak za pomocą PHP łączyć się z bazą. W punkcie 11.3.2 wyjaśnimy, jak używać danych z formularzy HTML do wstawiania nowego rekordu do tabeli bazy danych. Z punktu 11.3.3 dowiesz się, jak wykonywać zapytania pobierające dane i wyświetlać wyniki na dynamicznych stronach WWW.

### 11.3.1. Nawiązywanie połączenia z bazą danych

Aby w programie w języku PHP zastosować funkcje związane z bazami, trzeba wczytać moduł *DB.php* biblioteki PEAR DB. Na rysunku 11.6. operacja ta jest wykonywana w wierszu 0. Dostęp do biblioteki funkcji DB można uzyskać za pomocą wyrażenia `DB::<nazwa_funkcji>`. Funkcja używana do nawiązywania połączenia z bazą danych to `DB::connect('string')`, gdzie argument `string` określa informacje o bazie danych. Format argumentu 'string' to:

```
<oprogramowanie SZBD>://<konto użytkownika>:<hasło>@<serwer bazy danych>
```

```
0) require 'DB.php';
1) $d = DB::connect('oci8://acctl:pass12@www.host.com/db1');
2) if (DB::isError($d)) { die("nie można nawiązać połączenia - " .
    $d->getMessage()); }
...
3) $q = $d->query("CREATE TABLE PRACOWNIK
4)   (IDPRAC INT,
5)   IMIENAZWISKO VARCHAR(15),
6)   STANOWISKO VARCHAR(10),
7)   NRDZ INT);" );
8) if (DB::isError($q)) { die("nieudane tworzenie tabeli - " .
    $q->getMessage()); }
...
9) $d->setErrorHandler(PEAR_ERROR_DIE);
...
```

```

10) $eid = $d->nextID('PRACOWNIK');
11) $q = $d->query("INSERT INTO PRACOWNIK VALUES
12) ($eid, $_POST['imienazwiskoprac'], $_POST['stanowiskoprac'],
    $_POST['nrdzprac'])" );
    ...
13) $eid = $d->nextID('PRACOWNIK');
14) $q = $d->query("INSERT INTO PRACOWNIK VALUES (?, ?, ?, ?)",
15) array($eid, $_POST['imienazwiskoprac'], $_POST['stanowiskoprac'],
    $_POST['nrdzprac']) );

```

RYSUNEK 11.6. Nawiązywanie połączenia z bazą, tworzenie tabeli i wstawianie rekordu

Na rysunku 11.6 w wierszu 1. kod łączy się z bazą przechowywaną za pomocą systemu Oracle (reprezentuje go ciąg `oci8`). Człon <oprogramowanie SZBD> argumentu 'string' określa konkretny pakiet oprogramowania SZBD, z którym kod nawiązuje połączenie. Oto niektóre pakiety oprogramowania SZBD dostępne za pomocą biblioteki PEAR DB:

- **MySQL.** Podawany jako `mysql` (starsze wersje) i `mysqli` (nowsze wersje od 4.1.2).
- **Oracle.** Podawany jako `oci8` (wersje 7., 8. i 9.). Użyty w wierszu 1. rysunku 11.6.
- **SQLite.** Podawany jako `sqlite`.
- **Microsoft SQL Server.** Podawany jako `mssql`.
- **Mini SQL.** Podawany jako `msql`.
- **Informix.** Podawany jako `ifx`.
- **Sybase.** Podawany jako `sybase`.
- **Dowolny system zgodny z ODBC.** Podawany jako `odbc`.

Lista ta nie jest kompletna.

W argumencie `string` funkcji `DB::connect` po członie <oprogramowanie SZBD> znajduje się separator `://`, po którym następuje nazwa konta <nazwa konta>, separator `:` i hasło do konta <hasło>. Dalej znajdują się separator `@` oraz nazwa serwera i katalog (<serwer bazy danych>), gdzie przechowywana jest baza.

W wierszu 1. rysunku 11.6 użytkownik łączy się z serwerem `www.host.com/db1` z użyciem konta `acct1` i hasła `pass12`. Baza działa w SZBD Oracle `oci8`. Cały ciąg jest przekazywany do funkcji `DB::connect`. Informacje o połączeniu są przechowywane w zmiennej `połączenia`, `$d`, używanej w każdej operacji wykonywanej na konkretnej bazie danych.

**Wykrywanie błędów.** W wierszu 2. rysunku 11.6 pokazano, jak sprawdzać, czy z powodzeniem nawiązano połączenie z bazą danych. Biblioteka PEAR DB udostępnia funkcję `DB::isError`, która pozwala określić, czy operacja dostępu do bazy danych została zakończona powodzeniem. Argumentem tej funkcji jest zmienna połączenia (tu jest nią `$d`). Programista języka PHP może po każdym wywołaniu kierowanym do bazy sprawdzać, czy ostatnia operacja zakończyła się sukcesem, i kończyć pracę programu (za pomocą funkcji `die`), jeśli miało miejsce niepowodzenie. Komunikat o błędzie zwracany przez bazę jest dostępny za pomocą operacji `$d->get_message()`. Komunikat ten można wyświetlić w sposób pokazany w wierszu 2. rysunku 11.6.

**Przesyłanie zapytań i innych instrukcji języka SQL.** Po nawiązaniu połączenia większość poleceń języka SQL można przysyłać do bazy za pomocą funkcji *query*. Funkcja `$d->query` przyjmuje jako argument polecenie języka SQL i przesyła je na serwer bazy danych w celu wykonania. Na rysunku 11.6 kod z wierszy od 3. do 7. przesyła polecenie `CREATE TABLE`, aby utworzyć tabelę `PRACOWNIK` z czterema atrybutami. Po wykonaniu zapytania lub polecenia języka SQL wynik zapytania zostaje przypisany do zmiennej zapytania (tu nazwanej `$q`). W wierszu 8. kod sprawdza, czy zapytanie zostało z powodzeniem wykonane.

Biblioteka `PEAR DB` języka PHP udostępnia też inną technikę, dzięki której nie trzeba sprawdzać wystąpienia błędów po każdym poleceniu do bazy danych. Poniższa funkcja:

```
$d->setErrorHandler(PEAR_ERROR_DIE)
```

kończy pracę programu i wyświetla domyślne komunikaty o błędach, jeśli w trakcie dostępu do bazy danych za pomocą połączenia `$d` wystąpią problemy (patrz wiersz 9. rysunku 11.6).

### 11.3.2. Pobieranie danych z formularzy i wstawianie rekordów

Aplikacje bazodanowe często pobierają informacje za pomocą formularzy ze stron HTML lub innego rodzaju formularzy internetowych. Przykładowo, gdy kupujesz bilet lotniczy lub zgłaszasz wniosek o kartę kredytową, użytkownik musi wpisać dane osobowe, takie jak nazwisko, adres i numer telefonu. Te informacje są zwykle zbierane i zapisywane w rekordzie bazy danych na serwerze.

Wiersze od 10. do 12. rysunku 11.6 pokazują, jak może się to odbywać. W tym przykładzie pominęliśmy kod tworzący formularz i zbierający dane (może on być wersją kodu z rysunku 11.1). Zakładamy, że użytkownik wprowadził prawidłowe wartości parametrów wejściowych `imienazwiskoprac`, `stanowiskoprac` i `nrdzprac`. Te wartości są dostępne za pomocą automatycznej globalnej tablicy `$_POST`, opisanej w końcowej części punktu 11.2.4.

W poleceniu języka SQL `INSERT` w wierszach 11. i 12. rysunku 11.6 elementy tablicy `$POST['imienazwiskoprac']`, `$POST['stanowiskoprac']` i `$POST['nrdzprac']` przechowują wartości pobrane od użytkownika za pomocą formularza ze strony HTML. Te wartości są następnie wstawiane jako nowy rekord pracownika do tabeli `PRACOWNIK`.

Ten przykład pokazuje także inną cechę biblioteki `PEAR DB`. W niektórych aplikacjach dla każdego nowego rekordu wstawianego do bazy tworzony jest unikatowy identyfikator<sup>1</sup>.

Język PHP udostępnia funkcję `$d->nextID`, która służy do tworzenia sekwencji unikatowych wartości na potrzeby konkretnej tabeli. W przykładzie w tym celu tworzone jest pole `IDPRAC` tabeli `PRACOWNIK` (patrz wiersz 4. rysunku 11.6). W wierszu 10. pokazane jest, jak pobrać następną unikatową wartość sekwencji na potrzeby tabeli `PRACOWNIK`, a w wierszach 11. i 12. ta wartość jest wstawiana jako część nowego rekordu.

Kod wstawiający dane (wiersze od 10. do 12.) na rysunku 11.6 dopuszcza wprowadzenie szkodliwego ciągu znaków, modyfikującego pierwotne polecenie `INSERT`. Bezpieczniejszy sposób wstawiania danych i wywoływania innych zapytań polega na używaniu

---

<sup>1</sup> Jest on podobny do opisanego w rozdziale 12. generowanego przez system identyfikatora OID używanego w obiektowych i obiektowo-relacyjnych systemach baz danych.

**symbolu zastępczego** (znak ?). Przykład pokazano w wierszach od 13. do 15., gdzie wstawiany jest następny rekord. W tej wersji funkcji `$d->query()` występują dwa argumenty. Pierwszym jest polecenie języka SQL z jednym lub kilkoma znakami ? (symbolami zastępczymi). Drugi argument to tablica, której elementy posłużą do zastąpienia tych symboli zgodnie z kolejnością występowania (wiersze od 13. do 15. rysunku 11.6).

### 11.3.3. Zapytania pobierające dane z tabel bazy

Teraz przedstawimy trzy przykłady zapytań pobierających dane za pomocą języka PHP (rysunek 11.7). Kilka pierwszych wierszy (od 0. do 3.) nawiązuje połączenie `$d` z bazą i konfiguruje domyślną obsługę błędów w sposób opisany we wcześniejszym punkcie. Pierwsze zapytanie (wiersze od 4. do 7.) pobiera nazwisko i numer działu z wszystkich rekordów reprezentujących pracowników. Zmienna zapytania `$q` służy do wskazywania na **wyniki zapytania**. W wierszach od 5. do 7. pokazana jest pętla `while` przetwarzająca każdy wiersz wyników. Funkcja `$q->fetchRow()` z wiersza 5. pobiera następny rekord z wyników zapytania i steruje pętlą. Pętla rozpoczyna przetwarzanie od pierwszego rekordu.

Drugie przykładowe zapytanie jest pokazane w wierszach od 8. do 13. Jest to zapytanie dynamiczne. W tym zapytaniu warunki wybierania wierszy zależą od wartości wprowadzonych przez użytkownika. Chcemy tu pobrać nazwiska pracowników zajmujących określone stanowisko i zatrudnionych w konkretnym dziale. Stanowisko i numer działu są wpisywane za pomocą formularza i umieszczane w zmiennych tablicowych `$_POST['stanowiskoprac']` i `$_POST['nrdzprac']`. Jeśli użytkownik wpisał 'inżynier' jako stanowisko i 5 jako numer działu, zapytanie pobierze nazwiska wszystkich inżynierów zatrudnionych w dziale 5. Jest to więc dynamiczne zapytanie, którego wyniki zależą od danych wpisanych przez użytkownika. W tym przykładzie używane są dwa symbole zastępcze ?, opisane w końcowym fragmencie punktu 11.3.2.

```
0) require 'DB.php';
1) $d = DB::connect('oci8://acct1:pass12@www.host.com/dbname');
2) if (DB::isError($d)) { die("nie można się połączyć - " . $d->getMessage()); }
3) $d->setErrorHandler(PEAR_ERROR_DIE);
...
4) $q = $d->query('SELECT IMIENAZWISKO, NRDZ FROM PRACOWNIK');
5) while ($r = $q->fetchRow()) {
6)     print "Pracownik $r[0] jest zatrudniony w dziale $r[1] \n" ;
7) }
...
8) $q = $d->query('SELECT IMIENAZWISKO FROM PRACOWNIK WHERE STANOWISKO = ? AND
NRDZ = ?',
9)     array($_POST['stanowiskoprac'], $_POST['nrdzprac']) );
10) print "Pracownicy działu $_POST['nrdzprac'] na stanowisku
    $_POST['stanowiskoprac']: \n"
11) while ($r = $q->fetchRow()) {
12)     print "Pracownik $r[0] \n" ;
13) }
...
```



```

14) $allresult = $d->getAll('SELECT IMIENAZWISKO, STANOWISKO, NRDZ FROM
    PRACOWNIK');
15) foreach ($allresult as $r) {
16)   print "Pracownik $r[0] zajmuje stanowisko $r[1] i pracuje w dziale $r[2] \n" ;
17) }
    ...

```

RYSUNEK 11.7. Zapytania pobierające dane z bazy

W ostatnim zapytaniu (wiersze od 14. do 17.) pokazano inny sposób tworzenia zapytania i przetwarzania wynikowych wierszy w pętli. Tu funkcja `$d->getAll` zapisuje wszystkie rekordy z wyniku zapytania w jednej zmiennej — `$allresult`. Aby przetworzyć w pętli poszczególne rekordy, można zastosować pętlę `foreach`. Zmienna wiersza, `$r`, służy tu do iteracyjnego przetwarzania wszystkich wierszy ze zmiennej `$allresult`<sup>2</sup>.

Widać więc, że PHP jest dostosowany zarówno do uzyskiwania dostępu do baz danych, jak i do tworzenia dynamicznych stron WWW.

## 11.4. Krótki przegląd technologii programowania internetowych baz danych w Javie

Omówione mechanizmy języka skryptowego PHP działają na serwerze aplikacji i pełnią funkcję pośrednika, który za pomocą formularzy pobiera dane wyjściowe od użytkownika, tworzy zapytania do bazy danych, przesyła je na serwer bazy danych, a następnie tworzy dynamiczne strony WWW w języku HTML, aby wyświetlić wyniki zapytania. Środowisko Javy obejmuje komponenty działające na serwerze i inne komponenty, które można wykonywać w maszynie klienta. Istnieją też standardy wymiany obiektów z danymi. Tu pokrótce omówimy komponenty powiązane z internetem i dostępem do baz danych. Technologie JDBC i SQLJ opisaliśmy już dość szczegółowo w rozdziale 10.

**Serwlety Javy.** Serwlety to obiekty Javy, które mogą działać na serwerze WWW i zarządzać interakcjami z klientem. Te obiekty potrafią przechowywać informacje przesłane przez klienta w czasie sesji, co pozwala wykorzystać te informacje do generowania zapytań do bazy danych. Obiekty serwletów mogą też przechowywać wyniki zapytań, dlatego fragmenty wyników można sformatować za pomocą języka HTML i przesłać do klienta w celu ich wyświetlenia. Obiekt serwleta może przechowywać wszystkie wygenerowane w trakcie konkretnej interakcji z klientem informacje do czasu zamknięcia sesji z danym klientem.

**JSP (ang. *Java Server Pages*).** Ta technologia umożliwia skryptom po stronie serwera generowanie dynamicznych stron WWW i przysyłanie ich do klienta w sposób podobny jak w PHP. JSP jest jednak powiązana z Javą, a skrypty JSP można łączyć z kodem w Javie.

**JavaScript.** JavaScript to język skryptowy różniący się od Javy i rozwijany niezależnie. Jest powszechnie stosowany w aplikacjach internetowych i może działać na komputerze klienta lub na serwerze.

<sup>2</sup> Zmienna `$r` przypomina kursory i zmienne iteratora omówione w rozdziałach 10. i 12.

**JSON (ang. *JavaScript Object Notation*).** JSON służy do tekstowego reprezentowania obiektów z danymi. Dane można zapisać w formacie JSON, a następnie przesyłać w formacie tekstowym między klientami i serwerami w internecie. Format JSON można traktować jako alternatywę dla języka XML (patrz rozdział 13.). JSON reprezentuje obiekty za pomocą par atrybut-wartość. Ten format jest używany jako model danych w niektórych nowszych systemach baz danych, tzw. systemach NOSQL, np. w systemie MongoDB (patrz rozdział 24.).

## 11.5. Podsumowanie

W tym rozdziale pokazaliśmy, jak przekształcać ustrukturyzowane dane z baz w elementy wprowadzane lub wyświetlane na stronach internetowych. Skoncentrowaliśmy się na języku skryptowym PHP, który zyskuje bardzo dużą popularność w kontekście programowania baz internetowych. W podrozdziale 11.1 na prostym przykładzie przedstawiliśmy podstawy języka PHP w obszarze programowania rozwiązań internetowych. W podrozdziale 11.2 opisaliśmy podstawy języka PHP, w tym często stosowane tablice i tekstowe typy danych. W podrozdziale 11.3 pokazaliśmy, jak stosować język PHP do tworzenia różnego rodzaju poleceń do baz danych, w tym poleceń tworzenia tabel, wstawiania nowych rekordów i pobierania rekordów z baz. PHP działa na serwerze, czym różni się od niektórych innych języków skryptowych, wykonywanych na komputerze klienckim. W podrozdziale 11.4 omówiliśmy powiązane z Javą wybrane technologie, które można stosować w podobnych kontekstach co PHP.

Zaprezentowaliśmy tu tylko bardzo podstawowe wprowadzenie do PHP. Istnieje wiele książek, a także witryn poświęconych programowaniu w PHP na podstawowym i zaawansowanym poziomie. Istnieje też wiele bibliotek funkcji dla języka PHP, ponieważ jest to technologia o otwartym dostępie do kodu źródłowego.

## Pytania powtórkowe

- 11.1. Dlaczego języki skryptowe są popularnym narzędziem do programowania aplikacji internetowych? Gdzie w architekturze trójwarstwowej działa program w języku PHP? Gdzie wykonywany jest program w JavaScriptcie?
- 11.2. Jakiego rodzaju językiem programowania jest PHP?
- 11.3. Omów różne sposoby tworzenia ciągów znaków w PHP.
- 11.4. Opisz różne rodzaje tablic w PHP.
- 11.5. Czym są automatyczne zmienne globalne w PHP? Podaj przykłady automatycznych tablic globalnych języka PHP i wyjaśnij, w jaki sposób są one zwykle używane.
- 11.6. Czym jest PEAR? Czym jest PEAR DB?
- 11.7. Omów główne funkcje biblioteki PEAR DB używane do dostępu do baz danych. Opisz, jak wykorzystywana jest każda z nich.
- 11.8. Omów różne sposoby przetwarzania wyników zapytań w pętli w PHP.
- 11.9. Czym są symbole zastępcze? Jak używa się ich w programowaniu baz danych za pomocą PHP?

## Ćwiczenia

- 11.10. Przyjrzyj się schematowi bazy danych BIBLIOTEKA z rysunku 4.6. Napisz w PHP kod tworzący tabele z tego schematu.
- 11.11. Napisz w PHP program, który tworzy formularz internetowy do wprowadzania informacji na temat encji WYPOŻYCZAJĄCY. Wykonaj to samo ćwiczenie dla encji KSIĄŻKA.
- 11.12. Napisz w PHP interfejsy internetowe na potrzeby zapytań omówionych w ćwiczeniu 6.18.

## Wybrane publikacje

Dostępnych jest wiele materiałów (zarówno drukowanych, jak i internetowych) na temat programowania w języku PHP. Tu proponujemy trzy przykładowe książki. Bardzo dobrym wprowadzeniem do PHP jest pozycja Sklar (2005). Jeśli chodzi o zaawansowane budowanie witryn internetowych, w Schlossnagle (2005) znajdziesz wiele szczegółowych przykładów. Nixon (2014) napisał popularną książkę na temat programowania rozwiązań internetowych z uwzględnieniem technologii PHP, JavaScript, jQuery, CSS i HTML5.





# **Podejścia obiektowe, obiekto-relacyjne i XML: zagadnienia, modele, języki i standardy**



## Bazy obiektowe i obiektowo-relacyjne

W tym rozdziale omówimy mechanizmy obiektowych modeli danych i pokażemy, jak niektóre z tych mechanizmów zostały włączone w relacyjne systemy baz danych i standard SQL. Część funkcji obiektowych modeli danych została też wykorzystana w modelach danych systemów baz danych nowszych typów, tzw. systemach NOSQL (patrz rozdział 24.). Ponadto model XML (patrz rozdział 13.) jest pod pewnymi względami podobny do modelu obiektowego. Dlatego wprowadzenie do modelu obiektowego pomoże lepiej zrozumieć wiele nowinek w technologiach bazodanowych. Systemy baz danych oparte na obiektowym modelu danych były czasem nazywane bazami danych zorientowanymi obiektowo, jednak obecnie nazywa się je **obektowymi bazami danych**. Tradycyjne modele danych i systemy baz danych — takie jak relacyjny, sieciowy czy hierarchiczny — okazały się dość przydatne w rozwijaniu technologii bazodanowych mających zastosowanie w wielu powszechnie używanych aplikacjach biznesowych. Wykazują jednak pewne niedostatki przy projektowaniu i tworzeniu bardziej złożonych aplikacji — na przykład systemów baz danych na potrzeby inżynierii oprogramowania, przemysłowych (CAD/CAM i CIM<sup>1</sup>), nauk biologicznych i innych, telekomunikacji, systemów informacji geograficznej czy multimediiów<sup>2</sup>. Obiektowe bazy danych opracowano na potrzeby aplikacji wymagających przechowywania obiektów o bardziej złożonej strukturze. Kluczową cechą obiektowych baz danych jest możliwość określania zarówno *struktury* złożonych obiektów, jak i *operacji*, które na tych obiektach można wykonywać.

Inną przyczyną stworzenia obiektowych baz danych jest rosnąca popularność obiektowych języków programowania używanych do tworzenia aplikacji. Bazy danych stają się podstawowym elementem wielu programów, zaś tradycyjne systemy baz danych były trudne do zastosowania w obiektowych aplikacjach tworzonych w językach takich jak C++ czy JAVA. Obiektowe bazy danych są zaprojektowane w taki sposób, aby mogły być bezpośrednio (*niezauważalnie*) zintegrowane z oprogramowaniem tworzonym w obiektowych środowiskach programowania.

Potrzeba dodatkowych możliwości modelowania danych została również dostrzeżona przez producentów systemów zarządzania relacyjnymi bazami danych (SZRBD); nowe wersje systemów relacyjnych zawierają zatem wiele cech zaproponowanych w systemach

---

<sup>1</sup> Projektowanie wspomagane komputerowo (CAD — ang. *Computer-Aided Design*), wytwarzanie wspomagane komputerowo (CAM — ang. *Computer-Aided Manufacturing*) i wytwarzanie komputerowe (CIM — ang. *Computer-Integrated Manufacturing*).

<sup>2</sup> Multimedialne bazy danych muszą przechowywać różnego typu obiekty multimedialne, takie jak zapis wideo, dźwięk, obrazy, grafiki i dokumenty (więcej w rozdziale 26.).



obiektowych. W ten sposób powstały *obiektowo-relacyjne*. Najnowsza wersja standardu SQL (z 2008 roku) dla relacyjnych SZBD, SQL/Foundation, zawiera wiele z tych cech. Początkowo znajdowały się one w języku SQL/Object, a obecnie włączono je do podstawowej specyfikacji standardu SQL.

Pomimo powstania wielu prototypów, a także komercyjnych obiektowych systemów baz danych, nie znajdują one szerokiego zastosowania ze względu na popularność systemów relacyjnych i obiektowo-relacyjnych. Eksperymentalne prototypy to między innymi system ORION stworzony w MCC, OPENOODB z Texas Instruments, IRIS opracowany w laboratoriach firmy Hewlett-Packard, ODE z AT&T Bell Labs oraz projekt ENCORE/ObServer z Brown University. Dostępne obecnie komercyjnie systemy to między innymi: GemStone Object Server pochodzący z GemStone Systems, ONTOS DB z Ontos, Objectivity/DB z Objectivity Inc., Versant Object Database i FastObjects z Versant Corporation (i Poet), ObjectStore z ObjectDesign oraz Ardent Database z Ardent.

Wraz z powstaniem komercyjnych obiektowych systemów baz danych pojawiła się potrzeba stworzenia standardowego modelu i języka opisującego takie systemy. Ponieważ oficjalna procedura zatwierdzania standardów zabiera zazwyczaj kilka lat, konsorcjum producentów i użytkowników obiektowych systemów, nazywane ODMG, zaproponowało standard, którego obecna wersja jest znana jako ODMG 3.0.

Obiektowe bazy danych przyjęły wiele rozwiązań opracowanych początkowo dla obiektowych języków programowania<sup>3</sup>. W podrozdziale 12.1 opiszemy najważniejsze elementy stosowane w wielu obiektowych systemach baz danych i wprowadzone później w systemach obiektowo-relacyjnych oraz w standardzie SQL. Te elementy to *tożsamość obiektów*, *struktura obiektów*, *konstruktory typów*, *enkapsulacja operacji*, definicje *metod* będące częścią deklaracji klas, mechanizmy zapisywania obiektów w bazie w celu ich *utrwalenia*, *typy*, *hierarchia klas* i *dziedziczenie*. W podrozdziale 12.2 zobaczysz, jak te zagadnienia zostały włączone do najnowszych standardów języka SQL, co doprowadziło do powstania obiektowo-relacyjnych baz danych. Funkcje obiektowe pierwotnie dodano do standardu SQL:1999, a następnie zaktualizowano w standardzie SQL:2008. W podrozdziale 12.3 zajmiemy się standardami „czysto” obiektowych baz danych i zaprezentujemy cechy standardu ODMG 3.0 oraz język definiowania obiektów ODL. W podrozdziale 12.4 przedstawiony jest przegląd procesu projektowania obiektowych baz danych. W podrozdziale 12.5 omówiono obiektowy język zapytań OQL będący częścią standardu ODMG 3.0. W podrozdziale 12.6 opisane są wiązania języków programowania, określające, jak rozbudowywać obiektowe języki programowania o mechanizmy ze standardu obiektowych baz danych. Podrozdział 12.7 to podsumowanie rozdziału. Osoby niezainteresowane szczegółowym wprowadzeniem do obiektowych baz danych mogą pominąć podrozdziały od 12.3 do 12.6.

---

<sup>3</sup> Podobne pojęcia zostały również opracowane na potrzeby koncepcyjnego modelowania danych i reprezentacji wiedzy.

## 12.1. Przegląd pojęć obiektowych

### 12.1.1. Wprowadzenie do pojęć i cech obiektowych

Termin *obiektyowy* (ewentualnie *zorientowany obiektowo*), skracany do postaci zapisu *OO* lub *O-O*, zaczerpnięto z obiektowych języków programowania (OOPL). Obecnie, koncepcje obiektowości znajdują zastosowanie również w obszarach baz danych, inżynierii oprogramowania, baz wiedzy, sztucznej inteligencji, czy ogólnie — systemów komputerowych. Początki OOPL wiążą się z powstaniem języka SIMULA w końcu lat sześćdziesiątych. Język SMALLTALK opracowany w Xerox PARC<sup>4</sup> w latach siedemdziesiątych był jednym z pierwszych języków programowania uwzględniających dodatkowe obiektywne pojęcia, takie jak przekazywanie komunikatów i dziedziczenie. Nazywa się go *czystym OOPL*, ponieważ od początku został wyraźnie zaprojektowany jako obiektowy, w odróżnieniu od *hybrydowych OOPL*, które wprowadzają idee obiektowości do istniejących już wcześniej języków. Przykładem hybrydowego OOPL jest język C++, adaptujący obiektowość do popularnego języka programowania C.

**Obiekt** składa się zazwyczaj z dwóch komponentów: stanu (wartości) i zachowania (operacji). Zawiera *złożoną strukturę danych* oraz specyficzne operacje zdefiniowane przez programistę<sup>5</sup>. Obiekty w OOPL istnieją jedynie podczas wykonywania programu i dlatego są nazywane *obiettami ulotnymi*. Obiektowa baza danych może wydłużyć czas istnienia obiektów, trwale je zapisując. Stają się one wtedy *trwałymi obiektami*, a ich istnienie jest przedłużone poza czas zakończenia programu, dzięki czemu mogą być później odczytane i wykorzystane przez inne programy. Innymi słowy, obiektowe bazy danych przechowują *trwałe obiekty* na niezależnym nośniku danych i pozwalają na współdzielenie tych obiektów przez różne programy i aplikacje. Takie zastosowania wymagają wprowadzenia dobrze znanych z systemów baz danych funkcji, takich jak mechanizm indeksujący, sterowanie współbieżne i odtwarzanie danych. Obiektowa baza danych integruje się z co najmniej jednym obiektowym językiem programowania, zapewniając dostęp do trwałych współdzielonych obiektów.

Wewnętrzna struktura obiektu w obiektowych językach programowania zawiera specyfikację **zmiennych instancji**, które przechowują informacje opisujące aktualny stan obiektu. Zmienna instancji jest więc podobna do *atrybutu* w modelu relacyjnym, z tym, że zmienne mogą być zawarte wewnątrz obiektu i pozostać ukryte dla użytkownika. Zmienne instancji mogą mieć dowolnie złożony typ danych. Systemy obiektowe pozwalają również na zdefiniowanie zestawu operacji lub funkcji (zachowania), które mogą być zastosowane do określonego typu obiektu. Niektóre obiektowe modele wymagają wręcz, aby wszystkie operacje na obiekcie były zdefiniowane, przez co uzyskiwana jest *całkowita enkapsulacja* obiektów. To surowe podejście jest w przypadku większości obiektowych modeli danych nieco złagodzone z dwóch powodów. Po pierwsze, użytkownicy bazy danych muszą znać nazwy atrybutów, aby mogli się nimi posługiwać wyszukując obiekty o określonych cechach. Po drugie, całkowita enkapsulacja wymaga, aby każde odczytanie cech obiektu odbywało się za pośrednictwem predefiniowanych operacji, co utrudnia szybkie tworzenie zapytań.

---

<sup>4</sup> Centrum badawcze w Palo Alto (Palo Alto Research Center), Palo Alto, Kalifornia, USA.

<sup>5</sup> Obiekty mogą mieć też wiele innych cech, co zostanie omówione w dalszej części rozdziału.

W celu ułatwienia enkapsulacji definicja operacji jest podzielona na dwie części. Pierwsza z nich, nazywana *sygnaturą* lub *interfejsem*, określa nazwę operacji i jej argumenty (parametry). Druga część, zwana *metodą* lub *ciałem*, określa jej *implementację*, zwykle napisaną w uniwersalnym języku programowania. Operacje mogą być wywoływane poprzez przekazanie do obiektu *komunikatu* zawierającego nazwę operacji i jej parametry. Następnie obiekt wykonuje właściwą metodę. Takie podejście pozwala na modyfikowanie wewnętrznej struktury obiektu i wykonywanie na niej różnych działań bez potrzeby uruchamiania zewnętrznych programów. Enkapsulacja zapewnia więc pewną formę niezależności danych i operacji (patrz rozdział 2.).

Kolejnymi ważnymi pojęciami związanymi z systemami obiektowymi są hierarchia klas i *dziedziczenie*. Zapewniają one możliwość tworzenia nowych typów lub klas dziedziczących część struktury lub operacji z wcześniej zdefiniowanych typów i klas. A zatem, specyfikacja typów obiektów może być systematycznie rozwijana, łatwiej jest też tworzyć kolejne typy danych. Ponadto, można *wielokrotnie wykorzystywać* raz już utworzone definicje typów podczas tworzenia nowych.

Jednym z problemów spotykanych we wczesnych systemach obiektowych były *związki* pomiędzy obiektami. Dążenie do uzyskania całkowitej enkapsulacji w tych systemach prowadziło do wniosku, że związki pomiędzy obiektami nie powinny być określane jawnie, tylko za pomocą osobnych metod. Takie podejście nie nadaje się jednak do zastosowania w złożonych systemach bazodanowych z wieloma związkami, ponieważ istotne jest, aby można było łatwo te związki zidentyfikować i zaprezentować użytkownikowi. Standard ODMG wychodzi naprzeciw tym potrzebom i pozwala na przedstawianie związków binarnych jako pary *wzajemnych odwołań*, co zostanie dokładniej omówione w podrozdziale 12.3.

Kolejną koncepcją związaną z systemami obiektowymi jest *przeciążanie operatorów*, czyli możliwość zastosowania operacji do różnych typów obiektów. W takiej sytuacji *nazwa operacji* może się odnosić do wielu różnych *implementacji* w zależności od typu obiektu, do którego jest stosowana. Taka cecha jest również nazywana *polimorfizmem operatorów*. Na przykład, operacja obliczająca powierzchnię figury geometrycznej może mieć różne implementacje (metody) w zależności od tego, czy obiekt jest trójkątem, okręgiem, czy kwadratem. Może to wymagać tzw. *późnego przypisania* nazwy operacji do odpowiedniej metody, które odbywa się dopiero podczas wykonywania programu, kiedy typ obiektu, do którego stosowana jest operacja, staje się znany.

W następnych punktach szczegółowo omówimy podstawowe cechy obiektowych baz danych. W punkcie 12.1.2 opiszemy tożsamość obiektów. W punkcie 12.1.3 pokażemy, jak za pomocą konstruktorów typów określać typy obiektów o złożonej strukturze. Punkt 12.1.4 zawiera opis enkapsulacji i utrwalania. W punkcie 12.1.5 przedstawimy zagadnienia związane z dziedziczeniem. Punkt 12.1.6 obejmuje omówienie dodatkowych kwestii dotyczących obiektowości, a punkt 12.1.7 to podsumowanie wszystkich przedstawionych zagadnień obiektowych. W podrozdziale 12.2 pokażemy, jak niektóre z opisanych elementów zostały wbudowane w standard SQL:2008 dla relacyjnych baz danych. Dalej, w podrozdziale 12.3, zobaczysz, jak przedstawione rozwiązania zostały uwzględnione w standardzie obiektowych baz danych ODMG 3.0.

### 12.1.2. Tożsamość obiektów i porównanie obiektów z literałami

Jednym z zadań obiektowej bazy danych jest zapewnianie bezpośredniego powiązania obiektów ze światem rzeczywistego i z bazy danych, tak by obiekty nie utraciły integralności i tożsamości oraz by dało się je łatwo identyfikować oraz operować na nich. Dlatego obiektowa baza danych zapewnia **unikatową tożsamość** wszystkim zapisanym w niej obiektom. Ta unikatowa tożsamość jest zazwyczaj określana przez generowany systemowo **unikatowy identyfikator obiektu** (OID). Wartość OID nie jest widoczna dla zewnętrznego użytkownika, lecz jest używana przez system wewnętrznie do jednoznacznej identyfikacji obiektów oraz do tworzenia odniesień. OID może być w razie potrzeby przypisany do zmiennej programowej odpowiedniego typu.

Podstawową własnością OID jest jego **niezmiennność**, co oznacza, że wartość identyfikatora przypisana do określonego obiektu nie powinna się dla niego nigdy zmienić. Zachowuje to również tożsamość reprezentowanego obiektu ze światem rzeczywistego. Obiektowa baza danych musi więc zapewniać mechanizmy pozwalające na generowanie identyfikatorów obiektów oraz na utrzymanie ich niezmienności. Pożądane jest, aby każdy OID mógł być użyty tylko raz i by nawet po usunięciu obiektu z bazy identyfikator nie został wykorzystany ponownie. Wymagania stawiane OID wykluczają możliwość jego określania na podstawie wartości atrybutów obiektu, ponieważ wartość atrybutu może zostać zmieniona lub poprawiona. Można porównać to podejście do modelu relacyjnego, w którym każda relacja musi mieć atrybut klucza głównego, a wartość tego atrybutu w unikatowy sposób identyfikuje każdą krotkę. Jeśli wartość klucza głównego się zmieni, krotka zyska nową tożsamość, choć nadal może reprezentować ten sam obiekt z rzeczywistego świata. Inna możliwość to używanie dla obiektów ze świata rzeczywistego w różnych relacjach innych nazw atrybutów kluczy. Trudno wtedy mieć pewność, że klucze reprezentują ten sam obiekt ze świata rzeczywistego (np. w jednej relacji kluczem jest atrybut `Id_prac` encji `PRACOWNIK`, a w innej — atrybut `Pesel`).

OID nie powinien również zależeć od fizycznego adresu obiektu zapisanego na nośniku danych, ponieważ adres ten może się zmienić po fizycznej reorganizacji zapisu bazy. Pomimo to, w niektórych starszych systemach wykorzystywano fizyczny adres zapisanego obiektu ze względu na dużą wydajność takiego rozwiązania. Kiedy fizyczne położenie obiektu na nośniku się zmienia, w dotychczas zajmowanym miejscu można umieścić *wskaznik* zawierający informację o nowym położeniu obiektu. Najbardziej jednak popularną metodą jest wykorzystywanie jako identyfikatorów obiektów długich liczb całkowitych, które w połączeniu z odpowiednimi funkcjami mieszającymi (ang. *hash functions*) określają aktualne fizyczne położenie obiektu.

Pewne wczesne obiektowe modele danych wymagały, aby wszystko — od prostej wartości po złożone obiekty — było reprezentowane jako obiekty. W takim systemie wszystkie proste wartości liczbowe (np. liczby całkowite), znakowe czy logiczne muszą mieć swój własny identyfikator. Pozwala to używać różnych OID dla dwóch identycznych wartości, co w niektórych sytuacjach może być przydatne. Na przykład liczba 50 może w jednym miejscu oznaczać masę 50 kg, a w innym wiek osoby. W takim przypadku zostaną stworzone dwa niezależne obiekty, z których każdy będzie miał wartość 50. Rozwiązanie to, jakkolwiek słuszne z teoretycznego punktu widzenia, w praktyce prowadzi do tworzenia zbyt wielu identyfikatorów obiektów. Stąd też większość obiektowych systemów baz danych pozwala na przechowywanie zarówno obiektów, jak i prostych **wartości (literałów)**.

Każdy obiekt musi posiadać niezmienny identyfikator, podczas gdy proste wartości ich nie mają. Zazwyczaj wartości są przechowywane wewnątrz obiektu i *nie są dostępne* dla innych obiektów. W niektórych systemach również złożone obiekty mogą istnieć bez OID.

### 12.1.3. Złożone struktury typów obiektów i literałów

Inną cechą obiektowych baz danych jest to, że obiekty i literały mogą mieć *strukturę typu o dowolnej złożoności*, aby przechowywać wszystkie niezbędne informacje opisujące dany element. Z kolei w tradycyjnych systemach baz danych informacje o złożonych obiektach są często *rozproszone* między wiele relacji lub rekordów, co skutkuje utratą bezpośrednich powiązań między obiektami rzeczywistego świata a ich reprezentacją w bazie danych. W obiektowych bazach danych złożony typ jest tworzony z innych przez **zagnieżdżanie konstruktorów typów**. Trzy najbardziej podstawowe konstruktory to konstruktory atomów (`atom`), struktur lub krotek (`struct` lub `tuple`) i kolekcji.

- (1) Jeden z konstruktorów typów to konstruktor **atomów**, choć nazwa ta nie jest używana w najnowszym standardzie obiektowym. Ten konstruktor dotyczy podstawowych wbudowanych typów danych modelu obiektowego, które są podobne do podstawowych typów z wielu języków programowania: liczb całkowitych, ciągów znaków, liczb zmiennoprzecinkowych, typów wyliczeniowych, wartości logicznych itd. Te podstawowe typy danych są nazywane typami **jednowartościowymi** lub **atomowymi**, ponieważ każda wartość danego typu jest uznawana za atomową (niepodzielną).
- (2) Konstruktor drugiego rodzaju to konstruktor **struktur (krotek)**. Pozwala on tworzyć standardowe typy strukturalne, takie jak krotki (typy rekordowe) z podstawowego modelu relacyjnego. Typ strukturalny składa się z kilku komponentów i czasem nazywa się go typem *złożonym*. Bardziej precyzyjnie należałoby napisać, że konstruktor struktur nie jest odpowiednikiem typu, a raczej **generatorem typów**, ponieważ pozwala tworzyć wiele różnych typów strukturalnych. Przykładowo, dwa różne typy strukturalne, jakie można utworzyć, to: `struct Nazwisko<Imie: string, InicjałDrugiegoImienia: char, Nazwisko: string>` i `struct DyplomUczelni<Kierunek: string, Stopien: string, Rok: date>`. Do tworzenia złożonych zagnieżdżonych struktur w modelu obiektowym potrzebne są opisane dalej konstruktory typów *kolekcji*. Warto zauważyć, że w pierwotnym (podstawowym) modelu relacyjnym dostępne były tylko konstruktory typów *atom* i *struct*.
- (3) Do konstruktorów typów **kolekcji** (typów wielowartościowych) należą: `set(T)`, `list(T)`, `bag(T)`, `array(T)` i `dictionary(K, T)`. Dzięki nim jako część obiektu lub literału można w razie potrzeby zapisać kolekcję innych obiektów lub wartości. Te konstruktory też są uznawane za **generatory typów**, ponieważ pozwalają tworzyć wiele różnych typów. Przykładowo, `set(string)`, `set(integer)` i `set(Pracownik)` to trzy różne typy, które można utworzyć za pomocą konstruktora typów `set`. Wszystkie elementy w danej kolekcji wartości muszą być tego samego typu. Przykładowo, wszystkie wartości w kolekcji typu `set(string)` muszą być ciągami znaków.



*Konstruktor atom* służy do reprezentowania wszystkich podstawowych wartości atomowych, takich jak liczby całkowite i rzeczywiste, ciągi znaków, wartości logiczne i inne wartości podstawowych typów danych obsługiwanych bezpośrednio przez system. *Konstruktor tuple* pozwala tworzyć strukturalne wartości i obiekty w postaci  $\langle a_1:i_1, a_2:i_2, \dots, a_n:i_n \rangle$ , gdzie każde  $a_j$  jest nazwą atrybutu<sup>6</sup>, a każde  $i_j$  identyfikatorem obiektu.

Inne często stosowane konstruktory są nazywane typami kolekcji, jednak występują między nimi różnice. **Konstruktor set** (zbioru) tworzy obiekty lub literały w postaci zbioru *różnych* elementów  $\{i_1, i_2, \dots, i_n\}$  tego samego typu. **Konstruktor bag** (nazywany też *wielozbiorem*) działa podobnie, jednak elementy w wielozbiorze *mogą się powtarzać*. **Konstruktor list** tworzy *uporządkowaną listę*  $[i_1, i_2, \dots, i_n]$  identyfikatorów lub wartości tego samego typu. Lista przypomina **wielozbiór**, przy czym elementy listy są *uporządkowane*, dlatego można wskazywać pierwszy, drugi i  $j$ -ty element. **Konstruktor array** tworzy jednowymiarową tablicę elementów tego samego typu. Podstawowa różnica między tablicą a listą polega na tym, że lista może mieć dowolną liczbę elementów, natomiast dla tablicy zwykle określona jest maksymalna wielkość. **Konstruktor dictionary** (słownika) tworzy kolekcję par klucz-wartość  $(K, W)$ , gdzie klucz  $K$  pozwala pobrać powiązaną z nim wartość  $W$ .

Charakterystyczną *cechą* kolekcji jest to, że obiekty lub wartości tego rodzaju są *kolekcjami obiektów lub wartości tego samego typu*, które mogą być nieuporządkowane (konstruktory set i bag) lub uporządkowane (list lub array). Konstruktor typu tuple jest często nazywany **konstruktorem strukturalnym**, ponieważ nawiązuje do słowa kluczowego struct znanego z języków C i C++.

**Język definicji obiektów (ODL<sup>7</sup>)** zawierający wcześniej przywołane konstruktory typów może być używany do definiowania typów obiektów dla konkretnych zastosowań. W podrozdziale 12.3 opiszemy dokładniej standardy ODL i ODMG; najpierw wprowadzimy stopniowo poszczególne pojęcia, używając uproszczonej notacji. Konstruktory typów mogą być użyte do definiowania *struktur danych* dla *schematów obiektowych baz danych*.

Na rysunku 12.1 atrybuty dotyczące innych obiektów, np. dz w typie PRACOWNIK lub projekty w typie DZIAŁ, to **referencje** do innych obiektów, reprezentujące *relacje* między obiektami. Przykładowo, atrybut dz w typie PRACOWNIK jest typu DZIAŁ, dlatego pozwala wskazać konkretny obiekt typu DZIAŁ (obiekt, w którym jest zatrudniony dany pracownik). Wartością takiego atrybutu może być identyfikator konkretnego obiektu typu DZIAŁ. Związek binarny może być jednokierunkowy lub mieć postać *odwołania zwrotnego*. Odwołania zwrotne umożliwiają łatwe przechodzenie między elementami związku w obu kierunkach. Przykładowo, na rysunku 12.1 wartością atrybutu pracownicy typu DZIAŁ jest *zbiór odwołań* (identyfikatorów) do obiektów typu PRACOWNIK reprezentujących pracowników zatrudnionych w danym dziale. Odwołaniem zwrotnym jest atrybut dz w typie PRACOWNIK. W podrozdziale 12.3 zobaczysz, że standard ODMG umożliwia bezpośrednie deklarowanie odwołań zwrotnych jako atrybutów związków, co pozwala zagwarantować spójność takich odwołań.

<sup>6</sup> W terminologii obiektowej nazywany również **nazwą zmiennej instancji**.

<sup>7</sup> ODL (ang. *Object Definition Language*) nawiązuje do DDL (ang. *Data Definition Language*) w tradycyjnych systemach baz danych (patrz rozdział 2.).

```

define type PRACOWNIK:
  tuple (
    imię:          string;
    inicjał_dr_imienia: char;
    nazwisko:      string;
    pesel:         string;
    data_urodzenia: DATA;
    adres:         string;
    płeć:         char;
    pensja:        float;
    przełożony:    PRACOWNIK;
    dz:           DZIAŁ; );
define type DATA
  tuple (
    rok:    integer;
    miesiąc: integer;
    dzień:  ingeter; );
define type DZIAŁ
  tuple (
    nazwadz:    string;
    numerdz:    integer;
    kier:       tuple ( kierownik: Pracownik;
                       data_przej_kierownictwa: Data; );
    lokalizacje: set(string);
    pracownicy  set(PRACOWNIK);
    projekty:   set(PROJEKT); );

```

RYSUNEK 12.1. Określanie typów obiektów PRACOWNIK, DATA i DZIAŁ przy użyciu konstruktorów typów

## 12.1.4. Enkapsulacja operacji i trwałość obiektów

**Enkapsulacja operacji.** Pojęcie *enkapsulacji* (*hermetyzacji*) jest jednym z podstawowych pojęć języków programowania i systemów obiektowych. Łączy się ono z pojęciami *abstrakcyjnych typów danych* i *ukrywania informacji* w językach programowania. W tradycyjnych modelach baz danych pojęcie to nie miało zastosowania, ponieważ zalecane jest uwidacznianie struktury bazy danych dla użytkowników i zewnętrznych programów. W tradycyjnych bazach danych określona lista operacji może być zastosowana do obiektów *wszystkich typów*. Na przykład, w modelu relacyjnym operacje pobierania, wstawiania, usuwania i modyfikowania rekordów mogą być użyte w ten sam sposób dla *wszystkich relacji* w bazie danych. Relacja i jej atrybuty są widoczne dla użytkowników i zewnętrznych programów, które uzyskują do niej dostęp za pośrednictwem dostępnych operacji. Enkapsulacja może być zastosowana również do obiektów w obiektowej bazie danych. Podstawowa idea polega na zdefiniowaniu **zachowania** określonego typu obiektów przy użyciu **operacji**, które mogą być na nim wykonywane. Niektóre z nich mogą służyć do tworzenia (wstawiania) nowych obiektów lub do ich kasowania, inne do zmiany ich stanu, odczytu pojedynczych atrybutów lub przeprowadzania obliczeń na całych obiektach. Inne operacje zasadniczo wykonują różne kombinacje pobierania danych, obliczeń na nich i ich aktualizacji. Zasadniczo, wszystkie wspomniane operacje są **implementowane** w dowolnym *języku programowania ogólnego zastosowania*, co zapewnia elastyczność i duże możliwości definiowania operacji.



Zewnętrzni użytkownicy obiektów mają dostęp jedynie do **interfejsu** operacji, który definiuje nazwy i argumenty (parametry) każdej operacji. Implementacja jest ukryta przed użytkownikami i zawiera definicje wewnętrznych struktur danych oraz implementacje operacji umożliwiające dostęp do tych struktur. Interfejs związany z operacją jest nazywany **sygnaturą**, zaś jej implementacja — **metodą**.

W aplikacjach bazodanowych wymaganie, aby wszystkie obiekty były w pełni objęte enkapsulacją, jest zbyt rygorystyczne. Jednym z rozwiązań jest podzielenie struktury obiektów na atrybuty (zmienne instancji) **widoczne** i **ukryte**, przy czym atrybuty widoczne są dostępne bezpośrednio dla zewnętrznych użytkowników i programistów za pomocą języka zapytań. Ukryte atrybuty obiektu są całkowicie objęte enkapsulacją i można uzyskać do nich dostęp jedynie za pośrednictwem zdefiniowanych operacji. Większość obiektowych systemów baz danych używa do dostępu do widocznych atrybutów języków zapytań wysokiego poziomu. W rozdziale 12.5 opiszemy dokładniej język zapytań OQL, który jest proponowanym standardem dla obiektowych baz danych.

Termin **klasa** jest często używany w odniesieniu do definicji typu obiektu wraz z definicją operacji na tym obiekcie<sup>8</sup>. Na rysunku 12.2 pokazano, jak rozszerzając definicje typów z rysunku 12.1 o operacje można utworzyć definicje klas. Każda klasa posiada kilka zadeklarowanych operacji, których sygnatury (interfejsy) są zawarte w definicji klasy. Metoda (implementacja) każdej operacji musi być zdefiniowana w innym miejscu przy użyciu języka programowania. Typowe operacje to **konstruktor** (często o nazwie *new*), który jest używany do tworzenia nowego obiektu, i **destruktor**, używany do usuwania tego obiektu. Definiuje się również **modyfikatory**, czyli operacje pozwalające na zmianę stanu (wartości) atrybutów obiektów, oraz operacje **pobierające** informacje o obiekcie.

```
define class PRACOWNIK:
  type tuple (
    imię: string;
    inicjał_dr_imienia: char;
    nazwisko: string;
    pesel: string;
    data_urodzenia: DATA;
    adres: string;
    płeć: char;
    pensja: float;
    przełożony: PRACOWNIK;
    dz: DZIAŁ; );
  operations
    wiek: integer;
    nowy_prac: PRACOWNIK;
    usun_prac: boolean;
end PRACOWNIK;

define class DZIAŁ:
  type tuple (
```

<sup>8</sup> Taka definicja klasy nawiązuje do tego, jak klasa jest używana w popularnym języku programowania C++. Standard ODMG używa dodatkowo na określenie klasy terminu *interfejs* (patrz podrozdział 12.3). W modelu EER termin *klasa* był używany w odniesieniu do typu obiektu wraz ze zbiorem wszystkich obiektów tego typu (patrz rozdział 8.).

```

nazwadz: string;
numerdz: integer;
kier: tuple ( kierownik: PRACOWNIK;
              data_przej_kierownictwa: DATA; );
lokalizacje: set( string );
pracownicy set( PRACOWNIK );
projekty: set( PROJEKT ); );
operations liczba_prac: integer;
nowy_dz: DZIAŁ;
usun_dz: boolean;
przypisz_prac( e:PRACOWNIK ): boolean;
(*dodaje pracownika do działu *)
zwolnij_prac( e:PRACOWNIK ): boolean;
(*usuwa pracownika z działu *)
end DZIAŁ;

```

RYСУNEK 12.2. Dodanie operacji do definicji typów PRACOWNIK i DZIAŁ

Operację zastosowaną do obiektu zazwyczaj wywołuje się, używając **notacji z kropką**. Na przykład, jeśli *d* jest odnośnikiem do obiektu typu DZIAŁ, to możemy wywołać operację *liczba\_prac*, pisząc *d.liczba\_prac*. Podobnie, pisząc *d.usun\_dz* spowodujemy, że obiekt, do którego odnosi się *d*, zostanie skasowany (usunięty). Jedynym wyjątkiem jest konstruktor, który zwraca odnośnik do nowo utworzonego obiektu typu DZIAŁ. Stąd też w niektórych modelach obiektowych nazwa konstruktora zwyczajowo jest tożsama z nazwą samej klasy, chociaż ta zasada nie została zastosowana na rysunku 12.2<sup>9</sup>. Notacja z kropką jest używana również do odnoszenia się do atrybutów obiektu, na przykład *d.numerdz* lub *d.kier.data\_przej\_kierownictwa*.

**Utrwalanie obiektów poprzez nazywanie oraz kwestia osiągalności.** Obiektowe systemy baz danych są często blisko powiązane z obiektowymi językami programowania. Język programowania jest używany do określania implementacji metod (operacji), a także pozostałego kodu aplikacji. Nie wszystkie obiekty są przechowywane na stałe w bazie danych. **Obiekty ulotne** istnieją podczas wykonywania programu i znikają, kiedy ten kończy pracę. **Trwałe obiekty** są zapisywane w bazie danych i pozostają tam również po zakończeniu wykonywania programu. Typowymi sposobami na utrwalanie obiektów w bazie danych są *nazywanie* oraz *przypisanie* (i uczynienie ich *osiągalnymi*).

**Mechanizm nazywania** obiektu polega na nadaniu mu unikatowej stałej nazwy w ramach określonej bazy. Taka **nazwa obiektu** może być nadana specjalnym wyrażeniem lub operacją w programie, co zostało zilustrowane na rysunku 12.3. Nazwane trwałe obiekty mogą być używane jako **punkty wejścia** do bazy danych, poprzez które użytkownicy i aplikacje uzyskują dostęp do jej zasobów. Oczywiście nadawanie nazw wszystkim obiektom w bazie danych przechowującej tysiące rekordów nie jest rozwiązaniem praktycznym, więc obiekty mogą być również utrwalane w inny sposób — poprzez **osiągalność**. Mechanizm utrwalania przez osiągalność polega na tworzeniu obiektu osiągalnego z innego, trwałego obiektu. Mówimy, że *obiekt B* jest **osiągalny** z *obektu A*, jeśli ciąg odwołań w grafie reprezentującym *obiekt A* prowadzi do *obektu B*.

<sup>9</sup> Język C++ posiada domyślne nazwy konstruktora i destruktora. Na przykład dla klasy PRACOWNIK domyślna nazwa konstruktora to PRACOWNIK, a domyślna nazwa destruktora to ~PRACOWNIK. Często stosowanym rozwiązaniem jest używanie do tworzenia nowych instancji obiektów operatora new.

```

define class GrupaDziałów:
  type set( Dział );
  operations dodaj_dz( d:Dział ): boolean;
    (* dodaje Dział do obiektu GrupaDziałów *)
  usun_dz( d:Dział ): boolean;
    (* usuwa Dział z obiektu GrupaDziałów *)
  nowy_gr_dz: GrupaDziałów;
  usun_gr_dz: boolean;
end GrupaDziałów;
...
persistent name WszystkieDziały: GrupaDziałów;
...
d := nowy_dz;
(* stwórz nowy Dział i przypisz do zmiennej d *)
...
b := WszystkieDziały.dodaj_dz( d );
(* utwórz d poprzez dodanie do trwałego zbioru WszystkieDziały *)

```

RYSUNEK 12.3. Tworzenie trwałych obiektów poprzez zastosowanie mechanizmów nazywania i osiągalności

Jeśli stworzymy trwały nazwany obiekt  $N$ , którego stanem jest zbiór lub lista obiektów jakiejś klasy  $C$ , to możemy łatwo utrwalić obiekty  $C$ , *dodając je* do tego zbioru lub listy, i tym samym czyniąc osiągalnymi z  $N$ . W ten sposób  $N$  tworzy **trwałą kolekcję** obiektów klasy  $C$ . W standardzie modelu obiektowego  $N$  jest nazywany **ekstensją** obiektów klasy  $C$  (patrz podrozdział 12.3).

Na przykład, możemy zdefiniować klasę *GrupaDziałów* (patrz rysunek 12.3), której obiekty są typu *set(Dział)*<sup>10</sup>. Przypuśćmy, że stworzymy obiekt typu *GrupaDziałów* i utrwalimy go, nadając mu nazwę *WszystkieDziały*, jak to zostało pokazane na rysunku 12.3. Każdy obiekt typu *Dział*, który zostanie dodany do *WszystkieDziały* przy pomocy operacji *dodaj\_Dział*, stanie się obiektem trwałym dzięki osiągalności z obiektu *WszystkieDziały*. Jak zobaczymy w podrozdziale 12.3, standard ODMG IDL daje projektantowi możliwość deklarowania ekstensji jako części definicji klasy.

Zauważmy różnicę pomiędzy obiektowymi a tradycyjnymi bazami danych. W modelach tradycyjnych, takich jak relacyjny, przyjmujemy, że *wszystkie* obiekty są trwałe. Dlatego jeśli w bazie relacyjnej tworzymy tabelę *PRACOWNIK*, reprezentuje ona zarówno *deklarację typu* *PRACOWNIK*, jak i *trwały zbiór wszystkich rekordów (krotek)* tego typu. W podejściu obiektowym deklaracja klasy *PRACOWNIK* określa tylko typy danych i operacje dla obiektów tej klasy. Użytkownik musi dodatkowo zdefiniować trwały obiekt typu *set(PRACOWNIK)*, którego wartością będzie *kolekcja odwołań* do wszystkich trwałych obiektów typu *PRACOWNIK*, jeśli istnieje taka potrzeba (jak pokazano na rysunku 12.3<sup>11</sup>). Pozwala to na tworzenie zarówno trwałych, jak i ulotnych obiektów przy użyciu tych samych deklaracji typów i klas w ODL (języku definicji obiektów) i OOPL (obiektowym języku programowania). O ile jest to potrzebne, można stworzyć kilka trwałych kolekcji obiektów tej samej klasy.

<sup>10</sup> Jak zostanie pokazane w rozdziale 21., ODMG ODL używa składni *set<Dział>* zamiast *set(Dział)*.

<sup>11</sup> Niektóre systemy (np. POET) automatycznie tworzą ekstensje dla klas.

## 12.1.5. Hierarchia typów i dziedziczenie

**Uproszczony model dziedziczenia.** Kolejną istotną cechą systemów obiektowych jest możliwość dziedziczenia i tworzenia hierarchii typów. Tu używamy prostego modelu obiektowego, w którym atrybuty i operacje są traktowane jednakowo, ponieważ jedne i drugie mogą być w taki sam sposób dziedziczone. W rozdziale 12.3 omówimy model dziedziczenia w standardzie ODMG, różniący się od modelu omawianego w tym miejscu, ponieważ obejmuje *dwa rodzaje dziedziczenia*. Dziedziczenie umożliwia definiowanie nowych typów na podstawie istniejących, co prowadzi do powstania **hierarchii typów (klas)**.

Typ jest definiowany poprzez podanie jego nazwy i wyliczenie wszystkich jego atrybutów (zmiennych instancji) oraz operacji (metod)<sup>12</sup>. W używanym w tym podrozdziale uproszczonym modelu atrybuty i operacje są wspólnie nazywane *funkcjami*, jako że atrybuty przypominają operacje bez argumentów. Nazwa funkcji może się odnosić do wartości atrybutu lub do wartości zwracanej przez operację (metodę). W tym podrozdziale będziemy używać terminu **funkcja** w odniesieniu zarówno do atrybutów, *jak i* operacji obiektu, ponieważ z punktu widzenia wprowadzenia do dziedziczności są one traktowane identycznie<sup>13</sup>.

Podstawowa forma definiowania typu polega na nadaniu mu **nazwy**, a następnie wymienieniu nazw jego widocznych (*publicznych*) **funkcji**. Definiując w tym podrozdziale typy, będziemy dla uproszczenia stosowali następujący format, który nie definiuje argumentów funkcji:

```
NAZWA_TYPU: funkcja, funkcja, funkcja, ..., funkcja
```

Na przykład, typ charakteryzujący osobę mógłby wyglądać następująco:

```
OSOBA: Nazwisko, Adres, DataUrodzenia, Wiek, PESEL
```

W typie OSOBA funkcje Nazwisko, Adres, PESEL i DataUrodzenia mogą być przechowywane jako atrybuty, podczas gdy funkcja Wiek może być metodą wyliczającą Wiek z atrybutu Data ↪ Urodzenia i obecnej daty.

Pojęcie **podtypu** jest przydatne, gdy projektant lub użytkownik chce stworzyć nowy typ, który jest podobny, ale nie identyczny z typem już istniejącym. Podtyp dziedziczy wówczas wszystkie funkcje wcześniej zdefiniowanego typu, który będziemy nazywać **nadtypem**. Przypuśćmy na przykład, że chcemy zdefiniować dwa nowe typy PRACOWNIK i STUDENT w następujący sposób:

```
PRACOWNIK: Nazwisko, Adres, DataUrodzenia, Wiek, PESEL, Pensja, DataZatrudnienia, Staż
```

```
STUDENT: Nazwisko, Adres, DataUrodzenia, Wiek, PESEL, Kierunek, ŚredniaOcen
```

Ponieważ obydwa te typy, poza kilkoma dodatkowymi, zawierają wszystkie funkcje zdefiniowane w typie OSOBA, są **podtypami** tego właśnie typu. Każdy z nich może dziedziczyć wszystkie wcześniej zdefiniowane funkcje, czyli Nazwisko, Adres, DataUrodzenia, Wiek i PESEL. Dla typu STUDENT konieczne jest jedynie zdefiniowanie nowych (lokalnych) funkcji Kierunek

<sup>12</sup> W tym podrozdziale będziemy używali terminów *typ* i *klasa* jako opisujących tę samą rzecz — atrybuty i operacje pewnego typu obiektów.

<sup>13</sup> W rozdziale 12.3 zobaczymy, że typy i funkcje są podobne do interfejsów używanych w ODMG ODL.

i `ŚredniaOcen`, które nie zostały odziedziczone. Prawdopodobnie w tym przypadku `Kierunek` będzie przechowywany jako atrybut, zaś `ŚredniaOcen` będzie wyliczana na podstawie listy ocen przechowywanej jako *atrybuty prywatne* (ukryte) obiektu `STUDENT`. Dla typu `PRACOWNIK` funkcje `Pensja` oraz `DataZatrudnienia` będą przechowywane jako atrybuty obiektu, natomiast `Staż` może być metodą wyliczającą staż pracy na podstawie aktualnej daty i atrybutu `DataZatrudnienia`.

Tak więc, możemy zadeklarować typy `PRACOWNIK` i `STUDENT` w następujący sposób:

```
PRACOWNIK subtype-of OSOBA: Pensja, DataZatrudnienia, Staż
STUDENT subtype-of OSOBA: Kierunek, ŚredniaOcen
```

Zasadniczo, podtyp zawiera *wszystkie* funkcje zdefiniowane w jego nadtypie oraz dodatkowe funkcje *specyficzne* tylko dla podtypu. Możliwe jest więc określenie **hierarchii typów** pokazującej zależności podtyp-nadtyp pomiędzy wszystkimi typami zadeklarowanymi w systemie.

Jako inny przykład rozważmy typy opisujące figury geometryczne, które mogą być zdefiniowane następująco:

```
FIGURA: Kształt, Powierzchnia, PunktZaczeplenia
```

Dla obiektów typu `FIGURA` `Kształt` jest zaimplementowany jako atrybut, którego wartość jest elementem z listy „trójkąt”, „prostokąt”, „okrąg” itd., zaś `Powierzchnia` jest metodą obliczającą powierzchnię figury. `PunktZaczeplenia` określa współrzędne punktu określającego położenie obiektu. Przypuśćmy teraz, że chcemy zdefiniować kilka podtypów typu `FIGURA`:

```
PROSTOKĄT subtype-of FIGURA: Szerokość, Wysokość
TRÓJKĄT subtype-of FIGURA: Bok1, Bok2, Kąt
OKRĄG subtype-of FIGURA: Promień
```

Zauważmy, że `Powierzchnia` może być wyliczana dla każdego podtypu w inny sposób, ponieważ sposób obliczania powierzchni prostokąta jest inny od wzorów na pole trójkąta czy okręgu. Podobnie, atrybut `PunktZaczeplenia` może mieć różne znaczenie dla każdego z podtypów; może być środkiem dla `PROSTOKĄTA` lub `OKRĘGU`, a wierzchołkiem pomiędzy dwoma podanymi bokami dla typu `TRÓJKĄT`.

Zauważmy, że definicje typów określają obiekty, ale ich *nie tworzą*. Kiedy obiekt jest tworzony, to zazwyczaj przynależy do jednego lub kilku zadeklarowanych typów. Na przykład, obiekt reprezentujący okrąg jest równocześnie typu `OKRĄG` i `FIGURA` (poprzez dziedziczenie). Każdy obiekt staje się również częścią co najmniej jednej trwałej kolekcji obiektów (lub ekstensji), które są używane do grupowania obiektów w sposób odpowiedni dla aplikacji.

**Więzy ekstensji w zależności od hierarchii klas.** W aplikacjach bazodanowych często wszystkie typy i podtypy posiadają skojarzone ze sobą ekstensje, które przechowują zbiory wszystkich trwałych obiektów danego typu lub podtypu. W takim przypadku więzy ekstensji polegają na tym, że każdy obiekt będący elementem ekstensji podtypu musi być również elementem *ekstensji* nadtypu. Niektóre obiektowe systemy baz danych mają wbudowany typ systemowy (nazywany klasą `ROOT` lub `OBJECT`), którego ekstensja obejmuje wszystkie obiekty w systemie<sup>14</sup>.

<sup>14</sup> W modelu ODMG jest to klasa `OBJECT` (patrz podrozdział 12.3)

Klasyfikacja polega wtedy na przypisywaniu obiektów do dodatkowych podtypów, które mają istotne zastosowanie w konkretnej aplikacji; tworzy się tym samym **hierarchię typów** czy **hierarchię klas** dla całego systemu. Wszystkie ekstensje klas zdefiniowanych przez użytkownika i przez system są podzbiorami ekstensji odpowiadającej klasie `OBJECT`, pośrednio lub bezpośrednio. W modelu ODMG (patrz podrozdział 12.3) użytkownik może, ale nie musi, zdefiniować ekstensję dla każdej klasy (typu), w zależności od potrzeby.

Ekstensja jest nazwanym trwałym obiektem, którego wartość to **trwała kolekcja** przechowująca zbiór obiektów tego samego typu zapisanych trwale w bazie danych, które mogą być udostępnione i współdzielone przez zewnętrzne programy. **Ulotna kolekcja** istnieje tymczasowo podczas wykonania programu, ale nie jest zapisywana, kiedy program kończy pracę. Na przykład, ulotna kolekcja może być stworzona w celu przechowywania rezultatu zapytania, które wybiera niektóre obiekty z trwałej kolekcji i kopiuje je do ulotnej. Program może manipulować obiektami w kolekcji ulotnej, ale kiedy kończy pracę, kolekcja przestaje istnieć. Wiele kolekcji trwałych i ulotnych może zawierać obiekty tego samego typu.

Model dziedziczenia omawiany w tym podrozdziale jest bardzo prosty. Jak zobaczymy w podrozdziale 12.3, model ODMG rozróżnia dwa rodzaje dziedziczenia: jeden jest nazywany *dziedziczeniem interfejsu* i oznaczany symbolem dwukropka (:), drugi — będący *dziedziczeniem ograniczeń ekstensji* — jest oznaczany słowem kluczowym `EXTEND`.

### 12.1.6. Inne pojęcia obiektowe

**Polimorfizm operacji (przeciążanie operatorów).** Kolejną cechą charakterystyczną dla systemów obiektowych jest zapewnianie **polimorfizmu** operacji, nazywanego również **przeciążaniem operatorów**. Pozwala to na przypisanie tej samej nazwy operatora lub symbolu do dwóch lub więcej różnych *implementacji* operatora w zależności od typu obiektu, do którego operator jest zastosowany. Prosty przykładem ilustrującym tę koncepcję może być przywołanie konstrukcji znanych z języków programowania. W niektórych z nich operator „+” może oznaczać różne rzeczy, kiedy zastosujemy go do obiektów różnych typów. Jeśli operandami symbolu „+” są *liczby całkowite*, to zastosowana zostanie operacja dodawania liczb całkowitych. Jeśli operandy są *liczbami zmiennoprzecinkowymi*, to przeprowadzone zostanie dodawanie liczb zmiennoprzecinkowych. Jeśli zaś operandami są *zbiory*, to wykonaną operacją będzie łączenie zbiorów. Kompilator potrafi określić (na podstawie typu dostarczonych argumentów), którą operację należy wybrać.

W obiektowych bazach danych może nastąpić podobna sytuacja. Jako przykładu ilustrującego polimorfizm<sup>15</sup> w obiektowych bazach danych możemy użyć obiektu `FIGURA`, opisywanego w podrozdziale 12.1.5. Funkcja `Powierzchnia` jest tu zadeklarowana dla wszystkich obiektów typu `FIGURA`. Implementacja tej metody może być jednak różna dla każdego z podtypów. Jedną możliwością jest stworzenie ogólnej implementacji obliczania pola figury geometrycznej (na przykład, przy pomocy ogólnego algorytmu obliczającego pole wielokąta), a następnie napisanie bardziej wydajnych algorytmów obliczających pola poszczególnych typów obiektów geometrycznych, takich jak okrąg, prostokąt, trójkąt itd. W tym przypadku funkcja `Powierzchnia` będzie *przeciążona* różnymi implementacjami.

---

<sup>15</sup> Języki programowania pozwalają na kilka rodzajów polimorfizmu. Zainteresowanego czytelnika odsyłamy do noty bibliograficznej na końcu rozdziału, wymieniającej pozycje rozwijające tę kwestię.



Obiektowy system bazy danych musi w takiej sytuacji wybrać odpowiednią metodę dla funkcji *Powierzchnia*, bazując na typie obiektu geometrycznego, do którego ta funkcja jest stosowana. W systemach o ścisłych typach może to mieć miejsce na etapie kompilacji, ponieważ już wtedy typ obiektu musi być znany. Nazywamy to **wczesnym** (lub **statycznym**) **wiązaniem**. W systemach o słabych typach lub nie wspierających w ogóle typów danych (takich jak Smalltalk, LISP, PHP i większość języków skryptowych), typ obiektu, do którego stosowana jest funkcja, jest znany dopiero podczas wykonywania programu. W takim przypadku funkcja musi sprawdzać typ obiektu podczas wykonania i dopiero wtedy wywoływać odpowiednią metodę. Taka sytuacja jest nazywana **późnym** (lub **dynamicznym**) **wiązaniem**.

**Dziedziczenie wielokrotne i wybiórcze.** **Dziedziczenie wielokrotne (wielodziedziczenie)** w hierarchii typów zachodzi wtedy, kiedy pewien typ *T* jest podtypem dwóch (lub więcej) typów, i co za tym idzie — dziedziczy funkcje (atrybuty i metody) obu nadtypów. Na przykład, możemy stworzyć podtyp `KIEROWNIK_TECHNICZNY`, który jest równocześnie podtypem klas `KIEROWNIK` i `TECHNIK`. Prowadzi to do stworzenia raczej **siatki typów** niż ich hierarchii. Potencjalny problem w przypadku wielodziedziczenia wiąże się z sytuacją, kiedy kilka nadtypów, po których dziedziczymy, posiada różne funkcje o tej samej nazwie, co prowadzi do niejednoznaczności. W naszym przykładzie zarówno `KIEROWNIK`, jak i `TECHNIK` mogą mieć funkcję o nazwie `Pensja`. Jeśli w obu nadtypach ta funkcja jest realizowana przez różne metody, to nie jest jednoznacznie określone, którą z tych dwóch funkcji ma dziedziczyć typ `KIEROWNIK_TECHNICZNY`. Możliwe jednak, że obydwa typy (`KIEROWNIK` i `TECHNIK`) odziedziczyły funkcję `Pensja` z tego samego typu (na przykład `PRACOWNIK`) znajdującego się wyżej w siatce typów. Ogólna zasada mówi, że jeżeli funkcja jest dziedziczona z jednego *wspólnego nadtypu*, to jest ona dziedziczona tylko raz. W takim przypadku nie ma niejednoznaczności. Problem istnieje jedynie wtedy, kiedy funkcje obu nadtypów są różne.

Istnieje kilka technik rozwiązywania problemu wieloznaczności przy dziedziczeniu wielokrotnym. Jednym rozwiązaniem jest wymuszenie sprawdzania przez system jednoznaczności dziedziczenia podczas tworzenia podtypu i pozwolenie użytkownikowi na wybranie, która funkcja ma być dziedziczona. Innym rozwiązaniem jest użycie domyślnej funkcji systemowej. Trzecie polega na zablokowaniu wielodziedziczenia w przypadku wykrycia wieloznaczności, co zmusza użytkownika do zmodyfikowania nazwy funkcji w jednym z nadtypów. Niektóre obiektowe systemy baz danych w ogóle nie pozwalają na dziedziczenie wielokrotne. W standardzie obiektowych baz danych (patrz podrozdział 12.3) wielodziedziczenie jest dozwolone w kontekście dziedziczenia operacji interfejsów, ale już nie przy dziedziczeniu klas za pomocą słowa kluczowego `EXTENDS`.

**Wybiórcze (selektywne) dziedziczenie** zachodzi wtedy, gdy podtyp dziedziczy niektóre funkcje nadtypu. Pozostałe funkcje nie są dziedziczone. W tym przypadku można użyć operatora `EXCEPT` do wymienienia tych funkcji nadtypu, które *nie* mają być dziedziczone. Mechanizm dziedziczenia wybiórczego nie jest powszechnie stosowany w obiektowych systemach baz danych, ale jest często używany w systemach sztucznej inteligencji<sup>16</sup>.

---

<sup>16</sup> W modelu ODMG dziedziczenie typów odnosi się wyłącznie do dziedziczenia operacji, a nie atrybutów (patrz podrozdział 12.3).



## 12.1.7. Podsumowanie zagadnień dotyczących obiektowych baz danych

Na zakończenie tego podrozdziału przedstawiamy podsumowanie najważniejszych zagadnień używanych w obiektowych i obiektowo-relacyjnych systemach baz danych:

- **Tożsamość obiektu.** Obiekty mają własne unikatowe tożsamości, które są niezależne od wartości ich atrybutów i generowane przez obiektowych system bazy danych.
- **Konstruktory typów.** Struktury obiektów złożonych mogą być tworzone poprzez rekurencyjne stosowanie zestawu podstawowych konstruktorów (generatorów) typów, takich jak `tuple`, `set`, `list`, `array` i `bag`.
- **Enkapsulacja operacji.** Zarówno struktura danych, jak i operacje, które mogą być zastosowane w odniesieniu do obiektu, są zdefiniowane w definicji jego klasy (typu).
- **Zgodność z językami programowania.** Zarówno trwałe, jak i ulotne obiekty są łatwe w obsłudze. Obiekty są utrwalane poprzez przypisanie do trwałej kolekcji (ekstensji) lub przez jawne nazwanie (nadanie unikatowej nazwy, za pomocą której można odwoływać się do obiektu i go pobierać).
- **Hierarchie typów i dziedziczenie.** Typy obiektów mogą być określane poprzez hierarchię typów, która pozwala na dziedziczenie zarówno atrybutów, jak i metod (operacji) zdefiniowanych w innych typach. W niektórych modelach możliwe jest dziedziczenie wielokrotnie.
- **Ekstensje.** Wszystkie trwałe obiekty danego typu (klasy) `C` mogą być przechowywane w ekstensji, która jest nazwanym trwałym obiektem typu `set(C)`. Ekstensje związane z hierarchią typów mają narzucone więzy zbiorów bądź podzbiorów.
- **Polimorfizm i przeciążanie operatorów.** Nazwy operacji i metod mogą być przeciążane, aby stosować różne implementacje do różnych typów obiektów.

W dalszych podrozdziałach pokażemy, jak te mechanizmy są stosowane w praktyce: najpierw w standardzie SQL (podrozdział 12.2), a następnie w standardzie ODMG (podrozdział 12.3).

## 12.2. Rozszerzenia obiektowe w standardzie SQL

W rozdziałach 6. i 7. przedstawiliśmy SQL jako standardowy język relacyjnych SZBD. Pierwszą specyfikację języka SQL przygotowali Chamberlin i Boyce (1974), a w latach 1989 i 1992 SQL został wzbogacony i ustandaryzowany. Język ten był stale rozwijany w ramach nowego standardu, nazywanego początkowo SQL3; zatwierdzone elementy wersji SQL3 znalazły się w standardzie SQL:1999. Począwszy od wersji SQL3, w standardzie SQL pojawiły się mechanizmy obiektowych baz danych. Początkowo nazywano je SQL/Object, jednak później trafiły one do głównej wersji języka SQL, SQL/Foundation ze standardu SQL:2008.

Model relacyjny z rozszerzeniami obiektowymi jest czasem nazywany modelem **obiektoowo-relacyjnym**. W latach 2003 i 2006 w standardzie SQL wprowadzono dodatkowe poprawki i dodano mechanizmy związane z językiem XML (patrz rozdział 13.).

Poniżej opisujemy wybrane mechanizmy obiektowe dołączone do języka SQL:

- W celu określania złożonych obiektów zostały dodane niektóre **konstruktory typów**. Są to *typ wierszowy* (ang. *row type*), odpowiadający konstruktorowi krotek (tuple lub struct), oraz *typ tablicowy* (ang. *array type*) do tworzenia kolekcji. Inne konstruktory typów kolekcji, takie jak *set*, *list* i *bag*, nie były częścią specyfikacji SQL-99, ale później zostały dodane do standardu SQL:2008.
- Mechanizm określania **tożsamości obiektu** poprzez użycie *typu referencyjnego*.
- **Enkapsulacja operacji** jest zapewniona przez mechanizm definiowania operacji jako części deklaracji typów (tworzonych przez użytkownika). Technika ta przypomina *abstrakcyjne typy danych* tworzone w językach programowania. Ponadto procedury definiowane przez użytkownika umożliwiają definiowanie ogólnych metod (operacji).
- Mechanizmy **dziedziczenia**, dostępne za pomocą słowa kluczowego UNDER.

Omówimy teraz każde z powyższych pojęć bardziej szczegółowo. W tym omówieniu nawiązujemy do przykładu z rysunku 12.4.

```
(a) CREATE TYPE TYP_ADRES_MIESZK AS (
    NRBUD      VARCHAR (5),
    ULICA      VARCHAR (25),
    NRMIESZ    VARCHAR (5),
    NRLOKALU   VARCHAR (5)
);
CREATE TYPE TYP_ADRES_POLSKA AS (
    ADRES_MIESZK TYP_ADRES_MIESZK,
    MIASTO       VARCHAR (25),
    KOD          VARCHAR (10)
);
CREATE TYPE TYP_TELEFON_POLSKA AS (
    RODZAJ_TEL   VARCHAR (5),
    NRKIERUNKOWY CHAR (3),
    NRABONENTA   CHAR (7)
);
(b) CREATE TYPE TYP_OSOBA AS (
    IMIENAZWISKO VARCHAR (35),
    PŁEĆ          CHAR,
    DATAUR       DATE,
    TELEFONY      TYP_TELEFON_POLSKA ARRAY [4],
    ADRES         TYP_ADRES_POLSKA
)
INSTANTIABLE
NOT FINAL
REF IS SYSTEM GENERATED
INSTANCE METHOD WIEK() RETURNS INTEGER;
CREATE INSTANCE METHOD WIEK() RETURNS INTEGER
FOR TYP_OSOBA
BEGIN
    RETURN /* Kod wyznaczający wiek danej osoby
           na podstawie aktualnej daty i SELF.DATAUR */
END;
);
```

```

(c) CREATE TYPE TYP_OCENA AS (
    NRPRZEDMIOTU CHAR (8),
    SEMESTR      VARCHAR (8),
    ROK          CHAR (4),
    OCENA        CHAR
);
CREATE TYPE TYP_STUDENT UNDER TYP_OSOBA AS (
    KOD_KIERUNKU CHAR (4),
    ID_STUD      CHAR (12),
    STOPIEŃ      VARCHAR (5),
    INDEKS       TYP_OCENA ARRAY [100]
INSTANTIABLE
NOT FINAL
INSTANCE METHOD ŚREDNIA( ) RETURNS FLOAT;
CREATE INSTANCE METHOD ŚREDNIA( ) RETURNS FLOAT
FOR TYP_STUDENT
BEGIN
    RETURN /* Kod obliczający średnią studenta na podstawie
            SELF.INDEKS */
END;
);
CREATE TYPE TYP_PRACOWNIK UNDER TYP_OSOBA AS (
    KOD_STANOWISKA CHAR (4),
    PENSJA          FLOAT,
    PESEL           CHAR (11)
INSTANTIABLE
NOT FINAL
);
CREATE TYPE TYP_KIEROWNIK UNDER TYP_PRACOWNIK AS (
    ZARZĄDZANY_DZIAŁ CHAR (20)
INSTANTIABLE
);
(d) CREATE TABLE OSOBA OF TYP_OSOBA
    REF IS ID_OSOBY SYSTEM GENERATED;
CREATE TABLE PRACOWNIK OF TYP_PRACOWNIK
    UNDER OSOBA;
CREATE TABLE KIEROWNIK OF TYP_KIEROWNIK
    UNDER PRACOWNIK;
CREATE TABLE STUDENT OF TYP_STUDENT
    UNDER OSOBA;
(e) CREATE TYPE TYP_FIRMA AS (
    NAZWA_FIRMY VARCHAR (20),
    LOKALIZACJA VARCHAR (20));
CREATE TYPE TYP_ZATRUDNIENIE AS (
    Pracownik REF (TYP_PRACOWNIK) SCOPE (PRACOWNIK),
    Firma REF (TYP_FIRMA) SCOPE (FIRMA) );

```

```
CREATE TABLE FIRMA OF TYP_FIRMA (
    REF IS ID_FIRMY SYSTEM GENERATED,
    PRIMARY KEY(NAZWA_FIRMY) );
CREATE TABLE ZATRUDNIENIE OF TYP_ZATRUDNIENIE;
```

RYSUNEK 12.4. Wybrane mechanizmy obiektowe w języku SQL. (a) Używanie UDT jako typów atrybutów takich jak Adres i Telefon. (b) Tworzenie UDT TYP\_OSOBA. (c) Tworzenie UDT TYP\_STUDENT i TYP\_PRACOWNIK będących podtypami typu TYP\_OSOBA. (d) Tworzenie tabel opartych na wybranych UDT i dziedziczenie tabel. (e) Tworzenie relacji za pomocą słów REF i SCOPE

## 12.2.1. Typy definiowane przez użytkownika za pomocą polecenia CREATE TYPE i obiekty złożone

Aby umożliwić tworzenie obiektów o złożonej strukturze i oddzielenie deklaracji klasy (typu) od procesu tworzenia tabeli (która jest kolekcją obiektów lub wierszy, dlatego odpowiada opisanej w podrozdziale 12.1 ekstensji), w języku SQL udostępniono **typy użytkownika** lub **UDT** (ang. *User Defined Type* — *typy definiowane przez użytkownika*). Ponadto dodane zostały cztery typy kolekcji (dla typów i atrybutów wielowartościowych), aby umożliwić tworzenie obiektów o złożonej strukturze obok prostych (płaskich) rekordów. Użytkownik tworzy typy UDT dla konkretnej aplikacji w ramach schematu bazy danych. **Typ UDT** w najprostszej postaci może być określany przy użyciu następującej składni:

```
CREATE TYPE nazwa_typu AS (<deklaracje elementów składowych>);
```

Na rysunku 12.4 przedstawione są wybrane aspekty obiektowe języka SQL. Przykłady z tego rysunku będziemy objaśniać stopniowo, wraz z omawianiem poszczególnych zagadnień. Typ UDT może być używany zarówno jako typ atrybutu, jak i jako typ tabeli. Stosując UDT jako typ atrybutu w innym UDT, można zbudować złożoną strukturę obiektów (krotek) w tabeli — podobnie jak w wyniku zagnieżdżania konstruktorów (generatorów) opisanych w podrozdziale 12.1. Technika ta przypomina używanie konstruktora *struct* przedstawionego w punkcie 12.1.3. Przykładowo, na rysunku 12.4(a) UDT TYP\_ADRES\_MIESZK jest używany jako typ atrybutu ADRES\_MIESZK w UDT TYP\_ADRES\_POLSKA. Podobnie UDT TYP\_ADRES\_POLSKA jest typem atrybutu ADRES w UDT TYP\_OSOBA z rysunku 12.4(b). Jeśli UDT nie obejmuje żadnych operacji (tak jak w przykładach z rysunku 12.4(a)), można zastosować **typ wierszowy**, aby bezpośrednio utworzyć atrybut strukturalny za pomocą słowa kluczowego ROW. Można np. zastosować następujący kod, zamiast deklarować TYP\_ADRES\_MIESZK jako odrębny typ (co zrobiono na rysunku 12.4(a)):

```
CREATE TYPE TYP_ADRES_POLSKA AS (
    ADRES_MIESZK ROW (   NUMER          VARCHAR (5),
                        ULICA           VARCHAR (25),
                        NRMIESZKANIA    VARCHAR (5),
                        NRLOKALU        VARCHAR (5) ),
    MIASTO   VARCHAR (25),
    KOD      VARCHAR (10)
);
```

Aby dodać obsługę kolekcji i umożliwić tworzenie obiektów o złożonej strukturze, do języka SQL dodano cztery konstruktory: ARRAY, MULTISSET, LIST i SET. Przypominają one konstruktory typów opisane w punkcie 12.1.3. W początkowej specyfikacji języka SQL/Object

znajdował się tylko typ `ARRAY`, ponieważ pozwala on zasymulować działanie pozostałych typów. W późniejszych wersjach standardu SQL znalazły się trzy dodatkowe typy kolekcji. Na rysunku 12.4(b) atrybut `TELEFONY` typu `TYP_OSOBA` jest typu tablicowego z elementami wcześniej zdefiniowanego UDT `TYP_TELEFON_POLSKA`. Tablica `TELEFONY` zawiera maksymalnie cztery elementy, co oznacza, że dla osoby można zapisać cztery numery telefonów. Dla tablic nie trzeba deklarować maksymalnej liczby elementów.

Elementy w typie tablicowym można wskazywać za pomocą standardowej notacji z nawiasami kwadratowymi. Przykładowo, wyrażenie `TELEFONY[1]` wskazuje pierwszą wartość z atrybutu `TELEFONY` (patrz rysunek 12.4(b)). Wbudowana funkcja `CARDINALITY` zwraca aktualną liczbę elementów tablicy (lub innej kolekcji). Przykładowo, wyrażenie `TELEFONY → [CARDINALITY (TELEFONY)]` wskazuje ostatni element tablicy.

Często stosowana notacja z kropką służy do wskazywania komponentów **typu wierszowego** (UDT). Przykładowo, `ADRES.MIASTO` oznacza komponent `MIASTO` atrybutu `ADRES` (patrz rysunek 12.4(b)).

## 12.2.2. Identyfikatory obiektów oparte na odwołaniach

Za pomocą **typów odwołań** (słowo kluczowe `REF`) można tworzyć unikatowe identyfikatory obiektów generowane przez system. Przykładowo, na rysunku 12.4(b) wyrażenie:

```
REF IS SYSTEM GENERATED
```

oznacza, że za każdym razem, gdy tworzony jest nowy obiekt typu `TYP_OSOBA`, system przypisze do niego wygenerowany przez siebie unikatowy identyfikator. W razie potrzeby można też zrezygnować z generowanych przez system identyfikatorów obiektów i stosować tradycyjne klucze z podstawowego modelu relacyjnego.

Zwykle użytkownik może określić generowanie przez system bazy danych identyfikatorów obiektów dla poszczególnych wierszy w tabeli. Używając składni:

```
REF IS <atrybut_oid> <metoda_generowania_wartości>;
```

użytkownik deklaruje, że atrybut o nazwie `<atrybut_oid>` będzie używany do identyfikowania poszczególnych krotek w tabeli. Wyrażenie `<metoda_generowania_wartości>` może przyjmować wartość `SYSTEM GENERATE` lub `DERIVED`. W pierwszym przypadku system automatycznie wygeneruje unikatowy identyfikator dla każdej krotki, natomiast w drugim używana jest tradycyjna metoda opierająca identyfikację na wartościach klucza głównego dostarczonych przez użytkownika.

## 12.2.3. Tworzenie tabel z wykorzystaniem UDT

Na podstawie każdego UDT, który umożliwia tworzenie egzemplarzy (słowo `INSTANTIABLE`; patrz rysunek 12.4(b)), można utworzyć jedną lub więcej tabel. Ilustruje to rysunek 12.4(d), gdzie na podstawie UDT `TYP_OSOBA` tworzona jest tabela `OSOBA`. Zauważ, że UDT z rysunku 12.4(a) *nie umożliwiają tworzenia egzemplarzy*, dlatego mogą być używane tylko jako typy atrybutów, a nie do tworzenia tabel. Na rysunku 12.4(b) atrybut `ID_OSOBY` otrzymuje wygenerowany przez system identyfikator obiektu za każdym razem, gdy nowy rekord (obiekt) typu `OSOBA` jest tworzony i wstawiany do tabeli.

### 12.2.4. Enkapsulacja operacji

W SQL istnieje konstrukcja podobna do **UDT**, dzięki której użytkownik może poza definiowaniem atrybutów przypisywać typom określone zachowanie poprzez określanie metod (operacji). Ogólna forma specyfikacji UDT z metodami jest następująca:

```
CREATE TYPE <nazwa-typu> (  
  <lista atrybutów z określeniem ich typów>  
  <deklaracja funkcji (metod)>  
);
```

Przykładowo, na rysunku 12.4(b) zadeklarowana jest metoda `wiek()`, która oblicza wiek poszczególnych obiektów typu `TYP_OSOBA`.

Nadal trzeba napisać kod z implementacją danej metody. Implementację metody można wywoływać, podając plik, w którym znajduje się kod tej metody. Można też umieścić ten kod w deklaracji typu (patrz rysunek 12.4(b)).

SQL posiada kilka wbudowanych funkcji do obsługi typów użytkownika. Dla UDT o nazwie `Typ_T` **konstruktor** `Typ_T()` zwraca nowy obiekt tego typu. W nowym obiekcie UDT każdy atrybut jest inicjalizowany przy użyciu wartości domyślnej. Dla każdego atrybutu `A` niejawnie tworzona jest **funkcja obserwatora** (ang. *observer function*), która odczytuje wartość tego atrybutu. `A(X)` lub `X.A` zwraca więc wartość atrybutu `A` typu `Typ_T` (przyjmując, że `X` jest typu `Typ_T`). **Funkcja modyfikatora** (ang. *mutator function*) służy do przypisywania atrybutom nowych wartości. SQL pozwala na blokowanie tych funkcji tak, aby tylko użytkownicy posiadający uprawnienia `EXECUTE` mogli ich używać.

Ogólnie, UDT może mieć przypisane przez użytkownika dowolne funkcje. Składnia deklarowania takich metod to:

```
INSTANCE METHOD <nazwa> (<lista_argumentów>) RETURNS  
<typ_zwracanej_wartości>;
```

Można definiować dwa typy funkcji: wewnętrzne i zewnętrzne. Wewnętrzne funkcje są pisane w rozszerzonym języku PSM dla SQL (patrz rozdział 10.). Zewnętrzne funkcje mogą być pisane w dowolnym języku, a tylko ich sygnatura (interfejs) jest zawarta w definicji UDT. Zewnętrzna funkcja może być zadeklarowana następująco:

```
DECLARE EXTERNAL <nazwa_funkcji> <sygnatura>  
LANGUAGE <nazwa_języka_programowania>;
```

Atrybuty i funkcje UDT są podzielone na trzy kategorie określane następującymi słowami kluczowymi:

- **PUBLIC** (widoczne jako interfejs UDT),
- **PRIVATE** (niewidoczne jako interfejs UDT),
- **PROTECTED** (widoczne jako interfejs tylko dla podtypów).

Istnieje również możliwość definiowania **wirtualnych atrybutów** UDT. Ich wartość jest wyliczana i aktualizowana przy użyciu funkcji.

## 12.2.5. Dziedziczenie i przeciążanie funkcji

W języku SQL dziedziczenie można stosować do typów i do tabel. W tym punkcie omówimy obie te możliwości. Warto pamiętać, że wiele zasad dziedziczenia zostało już opisanych w punkcie 12.1.5. SQL obejmuje reguły obsługi **dziedziczenia typów** (dziedziczenie odbywa się za pomocą słowa kluczowego **UNDER**). Zwykle dziedziczone są zarówno atrybuty, jak i metody instancji (operacje). Jeśli możliwe ma być tworzenie podtypów, w UDT trzeba umieścić wyrażenie **NOT FINAL** (patrz rysunki 12.4(a) i (b), gdzie **TYP\_OSOBA**, **TYP\_STUDENT** i **TYP\_PRACOWNIK** są zadeklarowane z użyciem tego wyrażenia). Powiązane z zasadami dziedziczenia są zasady dotyczące przeciążania implementacji funkcji i rozróżniania ich nazw. Zasady te można podsumować następująco:

- Dziedziczone są wszystkie atrybuty.
- Kolejność nadtypów w wyrażeniu **UNDER** określa hierarchię dziedziczenia.
- Instancja podtypu może być używana wszędzie tam, gdzie może być używana instancja nadtypu.
- Podtyp może definiować na nowo każdą funkcję zadeklarowaną w nadtypie, pod warunkiem, że zachowana zostaje jej sygnatura.
- Podczas wywołania funkcji wybierane jest najlepsze dopasowanie na podstawie typów wszystkich argumentów.
- Na potrzeby dynamicznego konsolidowania uwzględniane są typy parametrów z momentu wykonywania.

Rozważmy przykład ilustrujący dziedziczenie typów (patrz rysunek 12.4(c)). Przyjmijmy, że chcemy utworzyć dwa podtypy typu **TYP\_OSOBA**: **TYP\_PRACOWNIK** i **TYP\_STUDENT**. Ponadto utworzymy podtyp **TYP\_KIEROWNIK**, dziedziczący wszystkie atrybuty (i metody) typu **TYP\_PRACOWNIK** i posiadający dodatkowy atrybut **ZARZADZANY\_DZIAŁ**. Te podtypy są przedstawione na rysunku 12.4(c).

Zwykle określone są atrybuty lokalne i dodatkowe specyficzne metody podtypu, który dziedziczy atrybuty i operacje (metody) swojego nadtypu.

Inną możliwością oferowaną przez SQL jest możliwość **dziedziczenia tabel** za pomocą określania nadtabel (i podtabel). Służy do tego słowo kluczowe **UNDER** (patrz rysunek 12.4(d)). W tej technice wstawienie nowego rekordu do podtabeli (np. do tabeli **KIEROWNIK**) powoduje też umieszczenie go w nadtabelach **PRACOWNIK** i **OSOBA**. Zauważ, że gdy do tabeli **KIEROWNIK** wstawiany jest rekord, trzeba podać wartości wszystkich odziedziczonych atrybutów. Operacje **INSERT**, **DELETE** i **UPDATE** są odpowiednio wywoływane w nadtabelach. Dziedziczenie tabel jest odpowiednikiem opisanego w punkcie 12.1.5 *dziedziczenia ekstensji*. Obowiązuje tu reguła, że krotka z podtabeli musi występować też w nadtabeli, aby wymusić ograniczenia obiektów związane z występowaniem w zbiorze i podzbiorze.

## 12.2.6. Określanie związków za pomocą odwołań

Atrybut krotki może być **odwołaniem** (tworzonym przy użyciu słowa kluczowego **REF**) do krotki innej (lub tej samej) tabeli. Przykład przedstawiony jest na rysunku 12.4(e).

Słowo kluczowe **SCOPE** określa nazwę tabeli, do krotek której odwołuje się atrybut referencyjny. Taka konstrukcja jest podobna do klucza obcego (ang. *foreign key*), z tą różnicą, że używa się raczej wartości generowanej przez system niż wartości klucza głównego.



**Wyrażenia ścieżkowe** odnoszące się do atrybutów będących komponentami krotek lub typów wierszowych są w SQL tworzone za pomocą **notacji z kropką**. Wyjątkiem są atrybuty typu REF, w przypadku których używa się symbolu dereferencyjnego (->). Na przykład, poniższe zapytanie znajduje pracowników zatrudnionych w firmie "ABCXYZ" za pomocą tabeli Zatrudnienie:

```
SELECT z.pracownik->imięnazwisko
FROM   Zatrudnienie AS z
WHERE  z.firma->nazwa_firmy = "ABCXYZ";
```

W SQL symbol -> jest używany do **dereferencji** i ma to samo znaczenie co w języku C. Zatem, jeśli r jest odwołaniem (referencją) do krotki, a a jest jej atrybutem składowym, to r->a oznacza wartość atrybutu a w tej krotce.

Jeśli dla tego samego typu wierszowego istnieje kilka relacji, w SQL stosuje się słowo kluczowe SCOPE do utworzenia atrybutu referencyjnego odwołującego się do określonej tabeli tego typu.

## 12.3. Model obiektowy ODMG i język definiowania obiektów ODL

We wprowadzeniu do rozdziału 6. wyjaśniliśmy, że jednym z powodów sukcesu komercyjnych relacyjnych SZBD jest standard SQL. Ponieważ przez lata nie istniał standard dla obiektowych baz danych, część użytkowników mogła zrezygnować z przejścia na tę nową technologię. Dlatego grupa obejmująca producentów i użytkowników obiektowych baz danych, ODMG (ang. *Object Data Management Group*), zaproponowała standard ODMG-93 (ODMG 1.0). Jego poprawione wersje to ODMG 2.0 i ODMG 3.0. Ten standard składa się z kilku części, w tym **modelu obiektowego**, **języka definiowania obiektów ODL**, **języka zapytań OQL** i **powiązań** z obiektowymi językami programowania.

W tym podrozdziale opiszemy model obiektowy ODMG i język ODL. W podrozdziale 12.4 powiemy, jak projektować obiektowe bazy danych na podstawie koncepcyjnych schematów EER. W podrozdziale 12.5 znajdziesz przegląd języka OQL, a w podrozdziale 12.6 — powiązań z językiem C++. W przykładach stosowania języków ODL i OQL oraz powiązań z językiem C++ będziemy korzystać z bazy UNIWERSYTET z rozdziału 4. W omówieniu będziemy się posługiwać modelem obiektowym ODMG 3.0 opisanym w Cattell i in. (2000)<sup>17</sup>. Należy zauważyć, że wiele pomysłów ujętych w modelu obiektowym ODMG jest opartych na dwóch dziesięcioleciach prowadzonych przez wiele osób badań nad modelowaniem koncepcyjnym i obiektowymi bazami danych.

Włączenie aspektów obiektowych do standardu SQL dla relacyjnych baz danych (co doprowadziło do powstania technologii obiektowo-relacyjnej) zostało omówione w podrozdziale 12.2.

---

<sup>17</sup> Wcześniejsze wersje tego modelu obiektowego zostały opublikowane w latach 1993 i 1997.

### 12.3.1. Przegląd modelu obiektowego ODMG

**Model obiektowy ODMG** jest modelem danych opartym na języku definicji obiektu (ODL) i obiektowym języku zapytań (OQL). Ma za zadanie określić standardowy model danych dla obiektowych baz danych, tak jak SQL opisuje standardowy model danych dla relacyjnych baz danych. Zapewnia on również standardową terminologię w obszarze, gdzie te same terminy były używane do opisywania różnych pojęć. Będziemy się starali stosować w tym rozdziale terminologię ODMG. Wiele pojęć z modelu ODMG zostało już omówionych w podrozdziale 12.1 i zakładamy, że Czytelnik zapoznał się z nim. Za każdym razem, kiedy stosowana terminologia będzie się różniła od tej zastosowanej w podrozdziale 12.1, będzie to zaznaczone.

**Obiekty i literały.** Podstawowymi składnikami, z których składa się obiektowy model danych, są obiekty i literały. Podstawowa różnica pomiędzy nimi polega na tym, że obiekty mają zarówno identyfikator, jak i **stan** (aktualną wartość), natomiast literały mają jedynie wartość i *nie posiadają identyfikatora obiektu*<sup>18</sup>. W obu przypadkach wartość może mieć złożoną strukturę. Stan obiektu może się zmieniać wraz z upływem czasu. Literał jest w zasadzie wartością stałą, która się nie zmienia, ale może mieć złożoną strukturę wewnętrzną.

**Obiekt** jest określany przez pięć elementów: *identyfikator, nazwę, okres istnienia, strukturę i sposób utworzenia*.

- (1) **Identyfikator obiektu** (lub `OBIEKT_ID`) jest unikatowy w całym systemie<sup>19</sup>. Każdy obiekt musi mieć swój identyfikator.
- (2) Niektóre obiekty mogą mieć opcjonalnie nadawaną **nazwę** unikatową w określonej bazie danych. Ta nazwa może być używana do odnoszenia się do obiektów, a system powinien zwracać obiekt na podstawie jego nazwy<sup>20</sup>. Oczywiście nie każdy obiekt musi posiadać unikatową nazwę. Zazwyczaj tylko kilka obiektów, zwykle przechowujących kolekcje obiektów określonego typu — takich jak *ekstensje* — będzie miało nazwę. Nazwy te są używane jako **punkty wejścia** do bazy danych, czyli poprzez odnalezienie obiektów o określonych nazwach, użytkownik może odnaleźć inne obiekty, do których nazwany obiekt się odwołuje. Inne istotne obiekty w aplikacji również mogą posiadać swoje unikatowe nazwy. Obiektowi można nadać *więcej niż jedną* nazwę. Wszystkie nazwy w określonej bazie danych muszą być unikatowe.
- (3) **Okres istnienia** obiektu określa, czy jest to *obiekt trwały* (obiekt bazy danych), czy *obiekt ulotny* (czyli obiekt w uruchomionym programie, który zniknie po zakończeniu programu). Czas życia jest niezależny od klas i typów — niektóre obiekty danej klasy mogą być ulotne, a inne trwałe.
- (4) **Struktura obiektu** określa, jak obiekt jest zbudowany, przy pomocy konstruktorów typów. Struktura określa, czy obiekt jest *typu atomowego*, czy nie. **Obiekt atomowy** to pojedynczy obiekt zgodny z UDT takim jak *Pracownik* lub *Dział*. Jeśli obiekt nie jest

---

<sup>18</sup> Będziemy używali zamiennie określeń *wartość* i *stan*.

<sup>19</sup> Identyfikator w modelu ODMG jest podobny do OID z punktu 12.1.2.

<sup>20</sup> Odpowiada to mechanizmowi nadawania nazw na potrzeby utrwalania danych opisanemu w punkcie 12.1.4.

atomowy, składa się z innych obiektów. Na przykład *obiekt kolekcji* nie jest atomowy, ponieważ jego stanem jest kolekcja innych obiektów<sup>21</sup>. Pojęcie *obiekt atomowy* oznacza co innego niż *konstruktor atomowy* opisany w punkcie 12.1.3, dotyczący wszystkich wartości wbudowanych typów danych. W modelu ODMG obiektem atomowym jest każdy *pojedynczy obiekt zdefiniowany przez użytkownika*. Wszystkie wartości wbudowanych typów danych są uznawane za *literały*.

- (5) **Sposób tworzenia** obiektu określa, jak obiekt może być tworzony. Zwykle służy do tego operacja *new* specjalnego interfejsu *ObjectFactory*. Zagadnienie to jest opisane szczegółowo w dalszej części podrozdziału.

W modelu obiektowym **literałem** nazywamy wartość, która *nie posiada* identyfikatora obiektu. Taka wartość może mieć jednak zarówno prostą, jak i złożoną strukturę. Istnieją trzy typy literałów: atomowe, kolekcje i struktury.

- (1) **Literały atomowe**<sup>22</sup> odpowiadają wartościom podstawowych typów danych i są predefiniowane. Podstawowe typy danych w modelu obiektowym to między innymi: długie, krótkie i nieujemne liczby całkowite (oznaczane w języku ODL odpowiednio słowami kluczowymi *Long*, *Short*, *Unsigned Long*, *Unsigned Short*), liczby zmiennoprzecinkowe i zmiennoprzecinkowe o podwójnej precyzji (*Float* i *Double*), zmienne logiczne (*Boolean*), pojedyncze znaki (*Char*), ciągi znaków (*String*) oraz typ wyliczeniowy (*Enum*).
- (2) **Literały strukturalne** odpowiadają mniej więcej wartościom tworzonym przy pomocy konstruktora krotek (*tuple*) opisanego w punkcie 12.1.3. Zawierają wbudowane typy *Date*, *Interval*, *Time* oraz *Timestamp* (patrz rysunek 12.5(b)). W zależności od wymagań aplikacji użytkownik może też definiować dodatkowe literały strukturalne<sup>23</sup>. Struktury definiowane przez użytkownika są tworzone w ODL przy użyciu operatora *STRUCT*, tak samo jak w językach C i C++.
- (3) **Literały kolekcji** określają wartość, która jest kolekcją obiektów lub wartości, przy czym sama kolekcja nie posiada swojego identyfikatora obiektu. Kolekcje w modelu obiektowym mogą być definiowane za pomocą *generatorów typów* *set<T>*, *bag<T>*, *list(T)* lub *array<T>*, gdzie *T* jest typem obiektów lub wartości w kolekcji<sup>24</sup>. Innym typem kolekcji jest słownik *dictionary<K, V>*, który jest kolekcją powiązań *<K, V>*, gdzie każde *K* jest kluczem (unikatową wyszukiwaną wartością) powiązanym z wartością *V*; może on być użyty do tworzenia indeksu kolekcji wartości.

```
(a) interface Object {
    ...
    boolean same_as(in Object inny_obiekt);
    Object copy();
}
```

<sup>21</sup> W modelu ODMG *obiekty atomowe* nie odpowiadają obiektom, których wartość jest typu podstawowego. Wszystkie wartości podstawowe (liczby całkowite, zmiennoprzecinkowe itd.) są *literałami*.

<sup>22</sup> Użyte określenie *atomowy* w literałach atomowych odpowiada sposobowi, w jaki zostało ono użyte w konstruktorze atomowym z punktu 20.2.2.

<sup>23</sup> Struktury *Date*, *Interval*, *Time* i *Timestamp* mogą być użyte przy tworzeniu literałów lub obiektów z identyfikatorami.

<sup>24</sup> Wymienione typy są podobne do odpowiadających im konstruktorów typów z punktu 12.1.3.

```

    void delete();
};

(b) class Date : Object {
    enum Weekday
        {Niedziela, Poniedziałek, Wtorek, Środa, Czwartek, Piątek, Sobota};
    enum Month
        {Styczeń, Luty, Marzec, Kwiecień, Maj, Czerwiec,
         Lipiec, Sierpień, Wrzesień, Październik, Listopad, Grudzień};
    unsigned short year();
    unsigned short month();
    unsigned short day();
    ...
    boolean is_equal(in Date inna_data);
    boolean is_greater(in Date inna_data);
};

class Time : Object {
    ...
    unsigned short hour();
    unsigned short minute();
    unsigned short second();
    unsigned short milisecond();
    ...
    boolean is_equal(in Time inny_czas);
    boolean is_greater(in Time inny_czas);
    ...
    Time add_interval(in Interval jakiś_Okres);
    Time subtract_interval(in Interval jakiś_Okres);
    Interval subtract_time(in Time inny_Czas);
};

class Timestamp : Object {
    unsigned short year();
    unsigned short month();
    unsigned short day();
    unsigned short hour();
    unsigned short minute();
    unsigned short second();
    unsigned short milisecond();
    ...
    Timestamp plus(in Interval jakiś_Okres);
    Timestamp minus(in Interval jakiś_Okres);
    boolean is_equal(in Timestamp inny_ZnacznikCzasu);
    boolean is_greater(in Timestamp inny_ZnacznikCzasu);
};

class Interval : Object {
    unsigned short day();
    unsigned short hour();
    unsigned short minute();

```

```

    unsigned short second();
    unsigned short milisecond();
    ...
    Interval plus(in Interval jakiś_Okres);
    Interval minus(in Interval jakiś_Okres);
    Interval product(in long jakaś_wartość);
    Interval quotient(in long jakaś_wartość);
    boolean is_equal(in Interval inny_Okres);
    boolean is_greater(in Interval inny_Okres);
};

```

(c) **interface** Collection : Object {

```

    ...
    exception ElementNotFound{ Object element; };
    unsigned long cardinality();
    boolean is_empty();
    ...
    boolean contains_element(in Object element);
    void insert_element(in Object element);
    void remove_element(in Object element) raises(ElementNotFound);
    iterator create_iterator(in boolean stable);
};

```

**interface** Iterator {

```

    exception NoMoreElements();
    ...
    boolean at_end();
    void reset();
    Object get_element() raises(NoMoreElements);
    void next_position() raises(NoMoreElements);
    ...
};

```

**interface** set : Collection {

```

    set create_union(in set inny_Zbiór);
    ...
    boolean is_subset_of(in set inny_Zbiór);
    ...
};

```

**interface** bag : Collection {

```

    unsigned long occurrences_of(in Object element);
    bag create_union(in bag inny_Wielozbiór);
    ...
};

```

**interface** list : Collection {

```

    exception Invalid_Index{ unsigned_long pozycja; };
    void remove_element_at(in unsigned long pozycja)
        raises(InvalidIndex);
    Object retrieve_element_at(in unsigned long pozycja)
        raises(InvalidIndex);
    void replace_element_at(in Object element, in unsigned long pozycja)
        raises (InvalidIndex);

```

```

void insert_element_after(in Object element, in unsigned long pozycja)
    raises (InvalidIndex);
...
void insert_element_first(in Object element);
...
void remove_first_element() raises(ElementNotFound);
...
Object retrieve_first_element() raises(ElementNotFound);
...
list concat(in list inna_Lista);
void append(in list inna_Lista);
};
interface array : Collection {
    exception Invalid_Index{ unsigned_long indeks; };
    exception Invalid_Size{ unsigned_long rozmiar; };
    void remove_element_at(in unsigned long indeks)
        raises(InvalidIndex);
    Object retrieve_element_at(in unsigned long indeks)
        raises(InvalidIndex);
    void replace_element_at(in unsigned long indeks, in Object element)
        raises(InvalidIndex);
    void resize(in unsigned long nowy_rozmiar)
        raises(InvalidSize);
};
struct association { Object klucz; Object wartość; };
interface dictionary : Collection {
    exception DuplicateName{ string klucz; };
    exception KeyNotFound{ Object klucz; };
    void bind(in Object klucz, in Object wartość)
        raises(DuplicateName);
    void unbind(in Object klucz) raises(KeyNotFound);
    Object lookup(in Object klucz) raises(KeyNotFound);
    boolean contains_key(in Object klucz);
};

```

RYSUNEK 12.5. Przegląd definicji interfejsu dla fragmentu modelu obiektowego ODMG.

(a) Podstawowy interfejs Object dziedziczony przez każdy obiekt, (b) standardowe interfejsy dla literalów strukturalnych, (c) interfejsy kolekcji i iteratory

Rysunek 12.5 przedstawia uproszczony opis podstawowych typów i generatorów typów modelu obiektowego. W notacji ODMG używamy trzech elementów: interface, literal i class. W terminologii używanej w ODMG słowo **zachowanie** oznacza *operacje*, a **stan** dotyczy *właściwości* (atrybutów i relacji). **Interfejs** określa tylko zachowanie typu obiektu i zwykle **nie umożliwia tworzenia instancji** (nie są tworzone obiekty odpowiadające interfejsowi). Choć interfejs może mieć określony w specyfikacji stan (atrybuty i relacje), *nie można* dziedziczyć po interfejsie. Tak więc interfejs służy do definiowania operacji, które mogą być *dziedziczone* przez inne interfejsy, a także przez klasy tworzące definiowane przez użytkownika obiekty w danej aplikacji. **Klasa** określa zarówno stan (atrybuty), jak i zachowanie (operacje) typu obiektu i **umożliwia tworzenie instancji**.

Obiekty bazy danych i aplikacji są więc zwykle tworzone na podstawie podanych przez użytkownika deklaracji klas składających się na schemat bazy danych. W deklaracji **literału** określony jest stan, ale bez operacji. Tak więc instancja literału przechowuje prostą lub złożoną wartość strukturalną, ale nie ma ani identyfikatora obiektu, ani enkapsulowanych operacji.

Rysunek 12.5 przedstawia uproszczoną wersję modelu obiektowego. Pełna specyfikacja jest dostępna w pracy Cattella i in. (2000). Konstrukcje użyte na rysunku 12.5 opiszemy po omówieniu modelu obiektowego. W modelu obiektowym wszystkie obiekty dziedziczą podstawowy interfejs `Object` pokazany na rysunku 12.5(a), obejmujący operacje takie jak `copy` (tworzy nową kopię obiektu), `delete` (usuwa obiekt) oraz `same_as` (porównuje tożsamość obiektu do innego obiektu)<sup>25</sup>. Operacje obiektów są wywoływane przy pomocy **notacji z kropką**. Na przykład, aby porównać obiekt `o` z innym obiektem `p`, piszemy

```
o.same_as(p)
```

Wynik zwracany przez powyższą operację jest wartością logiczną (`Boolean`) i byłby prawdą, jeśli tożsamość `p` jest taka sama jak tożsamość `o`, lub fałszem w przeciwnym wypadku. Podobnie, chcąc stworzyć obiekt `p`, który jest kopią obiektu `o`, piszemy

```
p = o.copy()
```

Alternatywą dla notacji z kropką jest **notacja ze strzałką**: `o->same_as(p)` lub `o->copy()`.

### 12.3.2. Dziedziczenie w modelu obiektowym ODMG

W modelu obiektowym ODMG występują dwa rodzaje dziedziczenia: dziedziczenie samego zachowania oraz dziedziczenie stanu i zachowania. **Dziedziczenie zachowania** jest nazywane **związkiem „jest”** lub **dziedziczeniem interfejsu** i reprezentowane za pomocą dwukropka (`:`)<sup>26</sup>. Dlatego w modelu obiektowym ODMG dziedziczenie zachowania wymaga, by nadtypem był interfejs. Podtypem może być albo klasa, albo inny interfejs.

Inny sposób dziedziczenia, **dziedziczenie EXTENDS**, jest reprezentowany za pomocą słowa kluczowego `extends`. W tej technice stan i zachowanie są dziedziczone między klasami, dlatego i nadtypem, i podtypem muszą być klasy. Wielodziedziczenie z użyciem słowa kluczowego `extends` nie jest dozwolone. Możliwe jest jednak wielodziedziczenie zachowania za pomocą notacji (`:`). Dlatego interfejs może dziedziczyć zachowanie po kilku innych interfejsach. Klasa też może dziedziczyć zachowanie po kilku interfejsach za pomocą notacji (`:`), a dodatkowo dziedziczyć zachowanie i stan po *najwyżej jednej* innej klasie za pomocą słowa kluczowego `extends`. W punkcie 12.3.4 przedstawimy przykłady tego, jak można stosować obie relacje dziedziczenia — `:` i `extends`.

<sup>25</sup> Dla potrzeb *blokowania* obiektów zdefiniowane są jeszcze dodatkowe operacje nie wymienione na rysunku 12.5. Koncepcja blokowania w bazach danych jest omówiona w rozdziale 22.

<sup>26</sup> W materiałach grupy ODMG dziedziczenie interfejsu jest też nazywane związkami typu-podtypu, jest-czymś i generalizacji-specjalizacji, choć w literaturze te pojęcia są stosowane do opisu dziedziczenia zarówno stanu, jak i operacji (patrz rozdział 8. i podrozdział 12.1).



### 12.3.3. Wbudowane interfejsy i klasy w modelu obiektowym

Na rysunku 12.5 pokazane są wbudowane interfejsy modelu obiektowego. Wszystkie interfejsy, np.: `Collection`, `Date` i `Time`, dziedziczą po podstawowym interfejsie `Object`. W modelu obiektowym obowiązuje podział na kolekcje, których stan obejmuje wiele obiektów lub literałów, oraz atomowe (i strukturalne) obiekty ze stanem w postaci pojedynczego obiektu lub literału. **Obiekty kolekcji** dziedziczą po podstawowym interfejsie `Collection` z rysunku 12.5(c), gdzie pokazane są operacje dostępne we wszystkich obiektach tego rodzaju. Mając daną kolekcję `o`, możemy określić liczbę jej elementów przy pomocy operacji `o.cardinality()`. Operacja `o.is_empty()` zwraca prawdę, jeśli obiekt `o` jest pusty, lub fałsz w przeciwnym przypadku. Operacje `o.insert_element(e)` i `o.remove_element(e)` odpowiednio dodają lub usuwają element `e` z kolekcji `o`. W końcu, operacja `o.contains_element(e)` zwraca prawdę, jeśli kolekcja `o` zawiera element `e`, lub fałsz w przeciwnym przypadku. Instrukcja `i = o.create_iterator()` tworzy **iterator** i dla kolekcji `o`, dzięki któremu można przechodzić kolejno po wszystkich elementach kolekcji. Interfejs obiektu typu `Iterator` jest również pokazany na rysunku 12.5(c). Operacja `i.reset()` ustawia iterator na pierwszym elemencie kolekcji (dla kolekcji nieuporządkowanych będzie to jakiś dowolnie wybrany element), a `i.next_position()` — na następnym elemencie kolekcji. Operacja `i.get_element()` zwraca **bieżący element**, na którym iterator jest w danym momencie ustawiony.

Model obiektowy ODMG używa **wyjątków** do zgłaszania błędów lub określonych warunków. Na przykład wyjątek `ElementNotFound` w interfejsie `Collection` byłby zgłoszony przez operację `o.remove_element(e)`, jeżeli element `e` nie należy do kolekcji `o`. Wyjątek `NoMoreElements` w interfejsie iteratora byłby zgłoszony przez operację `i.next_position()` jeśli iterator byłby ustawiony na ostatnim elemencie kolekcji i nie mógłby się ustawić na następnym.

Kolekcje są podzielone na typy `Set`, `List`, `Bag`, `Array` oraz `Dictionary` i wszystkie dziedziczą operacje interfejsu `Collection`. **Generators typów** `Set<T>` używamy do tworzenia obiektów, których wartością jest *zbiór elementów typu T*. Interfejs `Set` zawiera dodatkową operację `p = o.create_union(s)` (patrz rysunek 12.5(c)), która zwraca nowy obiekt `p` typu `Set<T>`; obiekt ten jest sumą zbiorów `o` i `s`. Inne operacje podobne do `create_union` (niepokazane na rysunku 12.5(c)) to `create_intersection(s)` oraz `create_difference(s)`. Do porównywania zbiorów można użyć operacji `o.is_subset_of(s)`, która zwraca prawdę, jeśli zbiór `o` jest podzbiorem zbioru `s`, lub fałsz w przeciwnym przypadku. Podobne operacje (niewymienione na rysunku 12.5(c)) to `is_proper_subset_of(s)`, `is_superset_of(s)` i `is_proper_superset_of(s)`. **Generator typów** `Bag<T>` pozwala na duplikowanie elementów kolekcji i również dziedziczy operacje interfejsu `Collection`. Posiada trzy dodatkowe operacje: `create_union(b)`, `create_intersection(b)` i `create_difference(b)`; wszystkie one zwracają nowe obiekty typu `Bag<T>`.

**Generator typów** `List<T>` dziedziczy operacje interfejsu `Collection` i może być używany do tworzenia kolekcji obiektów typu `T`, w których istotna jest kolejność elementów. Wartością obiektu tego typu jest *uporządkowana lista elementów typu T*. Można więc odwoływać się do pierwszego, ostatniego i *i*-tego elementu listy. Musimy również, dodając nowy element do listy, określić jego pozycję. Niektóre operacje listy zostały pokazane na rysunku 12.5(c). Jeśli `o` jest obiektem typu `List<T>`, to operacja `o.insert_element_first(e)` dodaje element `e` przed dotychczasowym pierwszym elementem listy `o` tak, że `e` staje się nowym pierwszym elementem listy. Analogiczna operacja (nie wymieniona na rysunku

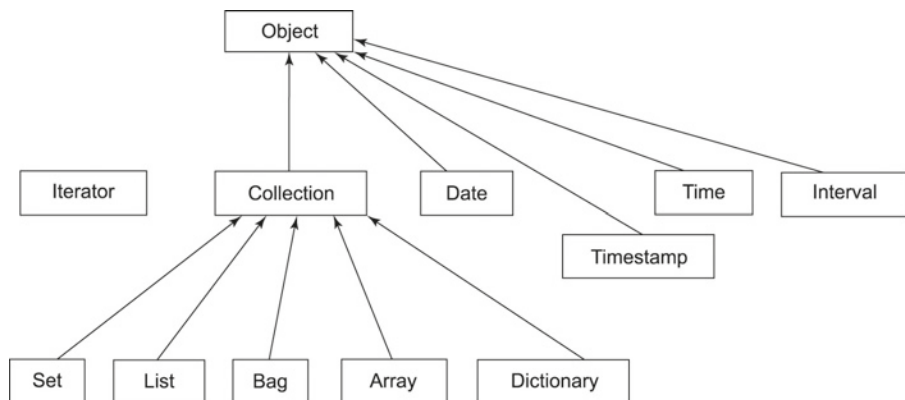
12.5(c)) to `o.insert_element_last(e)`. Operacja `o.insert_element_after(e, i)` z rysunku 12.5(c) wstawi element `e` po `i`-tym elemencie listy `o` albo zgłasza wyjątek `InvalidIndex` jeśli `i`-ty element listy `o` nie istnieje. Podobną operacją (niepokazaną) jest `o.insert_element_before(e, i)`. Operacje usuwające elementy z listy to `e = o.remove_first_element()`, `e = o.remove_last_element()` oraz `e = o.remove_element_at(i)`. Powyższe operacje usuwają wskazany element z listy oraz zwracają go jako wartość operacji. Inne operacje zwracają element listy, nie usuwając go z niej. Są to `e = o.retrieve_first_element()`, `e = o.retrieve_last_element()` oraz `e = o.retrieve_element_at(i)`. Są również dostępne dwie operacje do manipulowania listami. Są to `p = o.concat(l)`, tworząca nową listę `p`, która jest konkatenacją list `o` i `l` (zawiera wszystkie elementy listy `o`, a po nich elementy listy `l`) oraz `o.append(l)`, która dołącza elementy listy `l` na koniec listy `o` (nie tworząc przy tym nowego obiektu typu `List`).

Obiekty tworzone za pomocą **generatora typów** `Array<T>` również dziedziczą operacje interfejsu `Collection`. Są one podobne do list. Specyficzne operacje dostępne dla typu `Array` to `o.replace_element_at(i, e)`, która zastępuje element tablicy `o` na pozycji `i` elementem `e`; `e = o.remove_element_at(i)`, która zwraca `i`-ty element i zastępuje go pustym oraz `e = o.retrieve_element_at(i)`, która po prostu zwraca `i`-ty element tablicy. Każda z tych operacji może zgłosić wyjątek `InvalidIndex`, jeśli `i` jest większe niż rozmiar tablicy. Operacja `o.resize(n)` zmienia liczbę elementów tablicy `o` na `n`.

Ostatni rodzaj kolekcji to `Dictionary<K, V>`. Pozwala on na tworzenie kolekcji powiązanych par `<K, V>`, gdzie wszystkie wartości `K` (klucze) są unikatowe. Dzięki temu możliwe jest odczytywanie powiązań dysponując kluczem (który jest podobny do indeksu). Jeśli `o` jest kolekcją typu `Dictionary<K, V>`, to `o.bind(K, V)` kojarzy wartość `V` z kluczem `K` jako powiązanie `<K, V>` w kolekcji, podczas gdy `o.unbind(K)` usuwa powiązanie do klucza `K` w kolekcji `o`. Instrukcja `v = o.lookup(K)` zwraca wartość `V` skojarzoną w obiekcie `o` z kluczem `K`. Ostatnie dwie operacje mogą zgłosić wyjątek `KeyNotFound`. Ostatnia operacja — `o.contains_key(K)` — zwraca prawdę, jeśli klucz `K` istnieje w `o`, lub fałsz w przeciwnym przypadku.

Rysunek 12.6 przedstawia diagram ilustrujący hierarchię dziedziczenia wbudowanych konstruktorów w modelu obiektowym ODMG. Operacje są dziedziczone z nadtypu do podtypu. Interfejsy kolekcji opisane powyżej *nie pozwalają na bezpośrednie tworzenie instancji* obiektów bazujących na tych interfejsach. Mogą być one raczej użyte do określania przez użytkownika własnych kolekcji — typu `Set`, `Bag`, `List`, `Array` czy `Dictionary` — mających zastosowanie w konkretnej aplikacji bazodanowej. Jeśli atrybut lub klasa są typu jednej z kolekcji, na przykład `set`, to będą dziedziczyły operacje z predefiniowanego interfejsu `Set`. W przykładowej aplikacji używającej bazy danych UNIwersytet użytkownik może określić klasę typu `set<Student>`, której stanem jest zbiór obiektów typu `Student`. Do manipulowania obiektami typu `set<Student>` programista może używać operacji interfejsu `Set<T>`. Tworząc klasy aplikacji, zazwyczaj używa się języka definicji obiektów ODL (patrz punkt 12.3.6).

Należy zwrócić uwagę, że wszystkie obiekty w danej kolekcji *muszą być tego samego typu*. Choć w specyfikacjach kolekcji prezentowanych na rysunku 12.5(c) użyte jest słowo kluczowe `any`, nie oznacza to, że obiekty dowolnego typu można mieszać w jednej kolekcji. Znaczy to raczej, że dowolny typ obiektów może być użyty podczas określania typu elementów kolekcji (elementy mogą być również kolekcjami).



RYSUNEK 12.6. Hierarchia dziedziczenia dla wbudowanych interfejsów modelu obiektowego

### 12.3.4. Obiekty atomowe (definiowane przez użytkownika)

W poprzednim punkcie opisaliśmy wbudowane kolekcje w modelu obiektowym. Opiszemy teraz, jak można stworzyć typy dla *obiektów atomowych*. Są one określane w języku ODL przy pomocy słowa kluczowego `class`. W modelu obiektowym każdy obiekt zdefiniowany przez użytkownika nie będący kolekcją jest nazywany **obiektem atomowym**<sup>27</sup>.

W przykładowej bazie danych UNIWERSYTET użytkownik może określić typ (klasę) dla obiektów `Student`. Większość takich typów będzie **obiektami strukturalnymi**, na przykład typ `Student` będzie miał złożoną strukturę z wieloma atrybutami, związkami i operacjami, będzie jednak nadal nazywany typem atomowym, ponieważ nie jest kolekcją. Taki stworzony przez użytkownika typ obiektu atomowego jest definiowany jako klasa poprzez określenie jego **właściwości i operacji**. Właściwości określają stan obiektu i są podzielone na **atrybuty** i **związki**. W tym punkcie rozważamy trzy typy komponentów: atrybuty, związki i operacje, które mogą być użyte przy definiowaniu typów dla obiektów atomowych (strukturalnych). Zilustrujemy nasze rozważania dwiema klasami `Pracownik` i `Dział` pokazanymi na rysunku 12.7.

```

class Pracownik
( extent wszyscy_pracownicy
  key PESEL )
{
  attribute string nazwisko;
  attribute string pesel;
  attribute date data_urodzenia;
  attribute enum Płeć{M, K} płeć;
  attribute short wiek;
  relationship Dział pracuje_w
    inverse Dział::zatrudnia;
}
  
```

<sup>27</sup> Jak już zostało wspomniane wcześniej, definicja *obiektu atomowego* w modelu ODMG jest inna od definicji konstruktora atomowego podanej w punkcie 12.1.3, która jest często używana w literaturze dotyczącej obiektowych baz danych.

```

        void przenieś_prac(in string nowy_dział)
            raises(błędny_nowy_dział);
};
class Dział
( extent wszystkie_działy
  key nazwadz, numerdz )
{
    attribute string nazwadz;
    attribute short numerdz;
    attribute struct kier {Pracownik kierow, date data_przej_kierownictwa}
        kierownik;
    attribute set<string> lokalizacje;
    attribute struct Proj {string nazwa_proj, time godz_tygodniowo} projekty;
    relationship set<Pracownik> zatrudnia inverse Pracownik::pracuje_w;
    void dodaj_prac (in string nowy_prac_nazw) raises(błędne_nazw);
    void zmien_kierownika(in string nowy_kier_nazw; in date
        data_przej_kierownictwa);
};

```

RYSUNEK 12.7. Atrybuty, związki i operacje w definicji klasy

Atrybut (*attribute*) służy do opisanienia własności obiektu. Atrybuty mają swoje wartości, które zazwyczaj są literałami o prostej lub złożonej strukturze i są przechowywane wewnątrz obiektu. Atrybuty mogą jednak przechowywać również identyfikatory innych obiektów. Wartości atrybutów mogą być nawet określane poprzez metody obliczające wartość atrybutu. Na rysunku 12.7<sup>28</sup> atrybutami klasy *Pracownik* są nazwisko, pesel, data\_urodzenia, płeć i wiek, a dla klasy *Dział* są to nazwadz, numerdz, kierownik, lokalizacje i projekty. Atrybuty *kierownik* i *projekty* klasy *Dział* mają złożoną strukturę i są definiowane instrukcją *struct*, która odpowiada *konstruktorowi tuple* z punktu 12.1.3. Stąd na wartość atrybutu *kierownik* w każdym obiekcie typu *Dział* będą się składały dwie części: *kierow*, którego wartością jest identyfikator obiektu odwołujący się do obiektu typu *Pracownik*, który zarządza Działem, oraz *data\_przej\_kierownictwa* o wartości typu *date*. Atrybut *lokalizacje* obiektów *Dział* jest definiowany konstruktorem *set*, ponieważ każdy *Dział* może mieć zbiór kilku lokalizacji.

Słowo kluczowe *relationship* określa własność, która mówi, że dwa obiekty w bazie danych są ze sobą powiązane. W modelu obiektowym ODMG jawnie reprezentowane są jedynie związki binarne (patrz podrozdział 3.4), a każdy związek jest przedstawiany jako *para wzajemnych odwołań* określanych słowem kluczowym *relationship*. Na rysunku 12.7 istnieje jeden związek łączący każdego *Pracownika* z *Działem*, w którym dany *Pracownik* *pracuje* (związek *pracuje\_w* klasy *Pracownik*). Z drugiej strony, każdy *Dział* jest powiązany ze zbiorem *Pracowników* pracujących w tym *Dziale* poprzez związek *zatrudnia* klasy *Dział*. Słowo kluczowe *inverse* oznacza, że te dwie własności klas określają jedno logiczne powiązanie w dwóch kierunkach<sup>29</sup>.

<sup>28</sup> Na rysunku 12.7 używamy notacji ODL, która będzie bardziej szczegółowo omówiona w punkcie 12.3.6.

<sup>29</sup> W podrozdziale 7.4 omówiono, jak relacja może być przedstawiona przy pomocy dwóch atrybutów w obu kierunkach.

Dzięki związkom dwukierunkowym, system bazy danych może automatycznie utrzymywać integralność odwołań związków. Oznacza to, że jeśli wartość `pracuje_w` określonego obiektu `Pracownik` `p` odnosi się do Działu `d`, to wartość `zatrudnia_dla` dla Działu `d` musi zawierać odniesienie do `p` w swoim zbiorze odwołań do obiektów typu `Pracownik`. Jeżeli projektant bazy danych chce, aby związek był przedstawiany *tylko w jedną stronę*, to musi on być reprezentowany jako atrybut (lub operacja). Przykładem jest komponent kierownika atrybutu `kierownik` w klasie `Dział`.

Poza atrybutami i związkami projektant może do specyfikacji typu (klasy) dodać **operacje**. Każdy typ obiektu może mieć kilka **sygnaturek operacji**, które określają nazwę operacji, jej argumenty i zwracaną wartość. Nazwy operacji są unikatowe w ramach jednego typu, ale mogą być przeciążane poprzez używanie tej samej nazwy operacji w różnych typach obiektów. Sygnatura operacji może również określać nazwy **wyjątków**, które mogą być zgłoszone podczas wykonywania operacji. Na rysunku 12.7 klasa `Pracownik` ma jedną operację `przenieś_prac`, a `Dział` ma dwie operacje: `dodaj_prac` i `zmień_kierownika`.

### 12.3.5. Ekstensje, klucze i obiekty-fabryki

W modelu ODMG projektant bazy danych może deklarować *ekstensje* (słowo kluczowe `extent`) obiektów dowolnego typu określonego definicją **klasy**. Ekstensja ma nadaną nazwę i zawiera wszystkie trwałe obiekty danej klasy. Zachowuje się więc jak *obiekt zbiorowy* przechowujący wszystkie trwałe obiekty danej klasy. Na rysunku 12.7 klasy `Pracownik` i `Dział` mają zdefiniowane ekstensje `wszyscy_pracownicy` i `wszystkie_działy`. Jest to podobne do stworzenia dwóch obiektów typów `Set<Pracownik>` oraz `Set<Dział>` i utrwalenia ich poprzez nadanie nazw, odpowiednio, `wszyscy_pracownicy` i `wszystkie_działy`. Użycie ekstensji wymusza automatycznie narzucenie związku zbioru (podzbioru) pomiędzy ekstensjami typu i jego podtypu. Jeśli klasy `A` i `B` mają zdefiniowane ekstensje `wszystkie_A` i `wszystkie_B` oraz klasa `B` jest podtypem klasy `A` (czyli `class B extends A`), to kolekcja obiektów `wszystkie_B` musi być zawsze podzbiorem kolekcji `wszystkie_A`. Takie ograniczenie jest automatycznie narzucane przez system bazy danych.

Klasa z ekstensją może mieć jeden lub kilka kluczy. **Klucz** składa się z jednej lub z kilku własności klasy (atrybutów lub związków), których wartości muszą być unikatowe wewnątrz danej ekstensji. Przykładowo, na rysunku 12.7 klasa `Pracownik` ma zdefiniowany jako klucz atrybut `pesel` (każdy obiekt `Pracownik` w ekstensji musi mieć unikatowy numer `pesel`), a klasa `Dział` dwa klucze: `nazwadz` i `numerdz` (każdy `Dział` musi mieć unikatową nazwę i numer). W przypadku klucza kompozytowego<sup>30</sup>, który składa się z kilku właściwości klasy, jego składniki są zapisywane w nawiasach. Na przykład, jeśli klasa `Pojazd` z ekstensją `wszystkie_pojazdy` ma klucz składający się z pól `numer_rejestracyjny` i `województwo`, to deklarację klucza zapiszemy jako `(numer_rejestracyjny, województwo)`.

Jako następne przedstawimy pojęcie **obektu-fabryki** — jest to obiekt służący do tworzenia pojedynczych instancji obiektów poprzez swoje operacje. Niektóre interfejsy obiektów-fabryk będące częścią modelu obiektowego ODMG zostały pokazane na rysunku 12.8. Interfejs `ObjectFactory` ma tylko jedną operację — `new()` — która zwraca nowy obiekt z identyfikatorem `Object_Id`. Poprzez dziedziczenie tego interfejsu, użytkownicy mogą

<sup>30</sup> Klucz kompozytowy jest w raporcie ODMG nazywany *kluczem złożonym* (ang. *compound key*).

tworzyć swoje własne interfejsy obiektów-fabryk dla każdego (atomowego) zdefiniowanego przez użytkownika typu obiektu, a programista może dla każdego typu utworzyć inną implementację operacji *new*. Na rysunku 12.8 przedstawiono również interfejs `DateFactory`, posiadający dodatkowe operacje do tworzenia daty kalendarzowej (`calendar_date`), do tworzenia obiektów, których wartość wskazuje aktualną datę (`current_date`), i inne nie-pokazane na rysunku 12.8. Jak widzimy, obiekty-fabryki zapewniają **operacje konstruujące** nowe obiekty.

```
interface ObjectFactory {
    Object new();
};
interface SetFactory : ObjectFactory {
    Set new_of_size(in long rozmiar);
};
interface ListFactory : ObjectFactory {
    List new_of_size(in long rozmiar);
};
interface ArrayFactory : ObjectFactory {
    Array new_of_size(in long rozmiar);
};
interface DictionaryFactory : ObjectFactory {
    Dictionary new_of_size(in long rozmiar);
};

interface DateFactory : ObjectFactory {
    exception InvalidDate{};
    ...
    Date calendar_date(in unsigned short rok,
                      in unsigned short miesiąc,
                      in unsigned short dzień)
        raises(InvalidDate);
    ...
    Date current();
};
interface DatabaseFactory {
    Database new();
};
interface Database {
    ...
    void open(in string nazwa_bazy_danych)
        raises(DatabaseNotFound, DatabaseOpen);
    void close() raises(DatabaseClosed, ...);
    void bind(in Object jakiś_obiekt, in string nazwa_obiektu)
        raises(DatabaseClosed, ObjectNameNotUnique, ...);
    Object unbind(in string nazwa_obiektu)
        raises(DatabaseClosed, ObjectNameNotFound, ...);
    Object lookup(in string nazwa_obiektu)
        raises(DatabaseClosed, ObjectNameNotFound, ...);
};
```

RYSUNEK 12.8. Interfejsy ilustrujące obiekty-fabryki i obiekty reprezentujące bazy danych

I wreszcie, omówimy pojęcie **bazy danych** (ang. **database**). Ponieważ system baz danych może tworzyć i zarządzać wieloma różnymi bazami danych, z których każda ma własny schemat, model obiektowy ODMG posiada interfejsy obiektów `DatabaseFactory` oraz `Database`, które zostały pokazane na rysunku 12.8. Każdy obiekt typu `Database` ma przypisaną swoją własną nazwę i zdefiniowaną operację `bind`, służącą do wiązania trwałych obiektów z ich unikatowymi nazwami w bazie danych. Operacja `lookup` zwraca obiekt z bazy danych, który odpowiada określonej nazwie obiektu, a `unbind` usuwa nazwę trwałego obiektu z bazy danych.

### 12.3.6. Język definicji obiektów ODL

Po omówieniu podstaw modelu obiektowego ODMG w poprzednim podrozdziale, pokażemy teraz, jak przedstawione koncepcje mogą być zastosowane przy tworzeniu schematu obiektowej bazy danych przy użyciu języka definicji obiektów ODL<sup>31</sup>.

ODL został zaprojektowany w celu realizacji konstrukcji semantycznych modelu obiektowego ODMG i jest całkowicie niezależny od języków programowania. Jego podstawowym zastosowaniem jest tworzenie specyfikacji obiektów, czyli klas i interfejsów. ODL nie jest więc pełnym językiem programowania. Użytkownik może określić schemat bazy danych w ODL niezależnie od jakiegokolwiek języka programowania, a następnie używać wiązania do określonego języka takiego jak C++, Smalltalk czy Java. Wiązanie z językiem C++ opisujemy w podrozdziale 12.6.

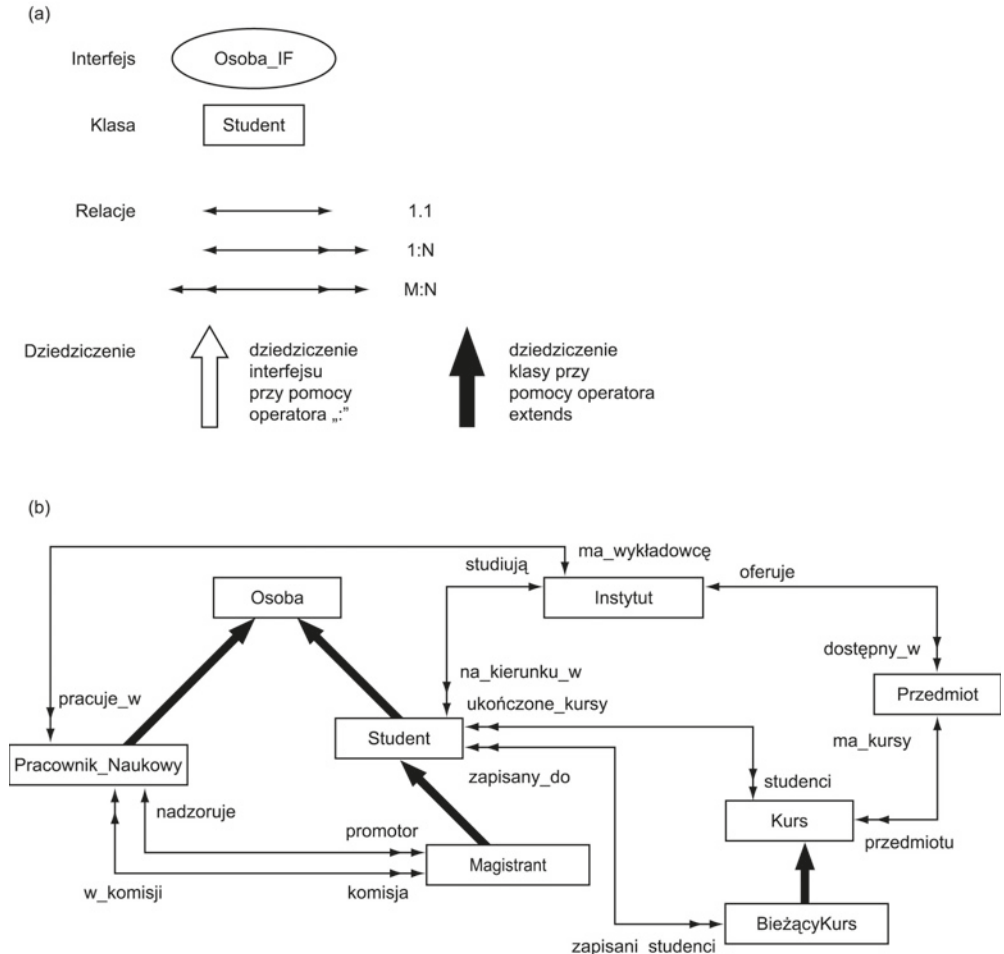
Na rysunku 12.9(b) pokazano przykładowy schemat obiektów dla części bazy danych `UNIWERSYTET` wprowadzonej w rozdziale 4. Opiszemy pojęcia ODL wykorzystując ten przykład oraz przykład z rysunku 12.11. Graficzna notacja rysunku 12.9(b) jest opisana na rysunku 12.9(a) i może być potraktowana jako odmiana diagramów EER (patrz rozdział 4.) z dodanym dziedziczeniem interfejsów, ale bez kilku pojęć znanych z modelu EER, takich jak kategorie (unie typów) oraz atrybuty związku.

Rysunek 12.10 prezentuje jeden z możliwych zestawów definicji klas w języku ODL dla bazy danych `UNIWERSYTET`. Należy pamiętać, że zazwyczaj istnieje kilka możliwych odwzorowań ze schematu obiektowego (lub modelu EER) do zapisu w języku ODL. Alternatywne rozwiązania omówimy później w podrozdziale 12.4.

Rysunek 12.10 przedstawia proste odwzorowanie części bazy danych `UNIWERSYTET` z rozdziału 4. Typy encji są przekształcone w klasy ODL, a dziedziczenie jest realizowane przez związek `EXTENDS`. Jednak nie istnieje prosty sposób na przedstawienie kategorii (typów unii) ani na realizację dziedziczenia wielokrotnego. Na rysunku 12.10 klasy `Osoba`, `Pracownik_Naukowy`, `Student` i `Magistrant` mają odpowiednio ekstensje `osoby`, `wykładowcy`, `studenci` i `magistranci`. Klasy `Student` i `Pracownik_Naukowy` dziedziczą po klasie `Osoba`, a `Magistrant` dziedziczy po klasie `Student`. Związki dziedziczenia narzucają na kolekcje `studenci` i `wykładowcy` ograniczenie do bycia podzbiorem kolekcji `osoby`. Analogicznie, kolekcja `magistranci` musi cały czas być podzbiorem kolekcji `studenci`. Pojedyncze obiekty typów `Student` i `Pracownik_Naukowy` będą dziedziczyły własności (atrybuty i związki) oraz operacje klasy `Osoba`, a z kolei obiekty `Magistrant` dziedziczą własności i operacje klasy `Student`.

<sup>31</sup> Składnia i typy danych ODL są kompatybilne z *IDL* (ang. *Interface Definition Language*) standardu *CORBA* (ang. *Common Object Request Broker Architecture*), z rozszerzeniami dla relacji i innych pojęć związanych z bazami danych.





RYСУNEK 12.9. Przykładowy schemat bazy danych. (a) Graficzne oznaczenia służące do przedstawiania schematów ODL. (b) Graficzny schemat części bazy danych UNIУERSYTET (klasy OCENA i STOPIEŃ zostały pominięte)

```

class Osoba
( extent osoby
  key pesel)
{
  attribute struct ImięNazwisko {string imię,
                                string drugie imię,
                                string nazwisko} imięnazwisko;

  attribute string pesel;
  attribute date data_urodzenia;
  attribute Płeć {M, K} płeć;
  attribute struct Adres {short numer,
                          string ulica,
                          short nr_mieszkania,
  
```

```

        string miasto,
        string województwo,
        string kod} adres;

    short wiek(); };

class Pracownik_Naukowy extends Osoba
(extent wykładowcy)
{
    attribute string stanowisko;
    attribute float pensja;
    attribute string biuro;
    attribute string telefon;
    relationship Instytut pracuje_w inverse Instytut::ma_wykładowcę;
    relationship set<Magistrant> nadzoruje inverse Magistrant::promotor;
    relationship set<Magistrant> w_komisji inverse Magistrant::komisja;
    void daj_podwyżkę(in float podwyżka);
    void awansuj(in string nowe_stanowisko); };

class Ocena
( extent oceny )
{
    attribute enum MożliweOceny {2, 3, 4, 5} ocena;
    attribute string data;
    relationship Kurs kurs inverse Kurs::studenci;
    relationship Student student inverse Student::ukończone_kursy; };

class Student extends Osoba
( extent studenci )
{
    attribute string rokst;
    attribute Instytut specjalność_w;
    relationship Instytut na_kierunku inverse Instytut::studiują;
    relationship set<Ocena> ukończone_kursy inverse Ocena::student;
    relationship set<BieżącyKurs> zapisany_na inverse
        BieżącyKurs::zapisani_studenci;
    void zmień_kierunek(in string nazwai) raises(błędna_nazwai);
    float średnia_ocen();
    void zapisz(in short numer_kursu) raises(błędny_numer_kursu);
    void przypisz_ocenę(in short numer_kursu, in DostępnaOcena ocena)
        raises(błędny_kurs, błędna_ocena); };

class StopieńNauk
{
    attribute string instytut;
    attribute string stopień;
    attribute string rok; };

class Magistrant extends Student
( extent magistranci )
{
    attribute set<Ocena> oceny;
    relationship Pracownik_Naukowy promotor inverse
Pracownik_Naukowy::nadzoruje;
    relationship set<Pracownik_Naukowy> komisja inverse
        Pracownik_Naukowy::w_komisji;
    void przypisz_promotora(in string nazwisko; in string imię)
        raises(błędny_wykładowca);
    void przypisz_członka_komisji(in string nazwisko; in string imię)
        raises(błędny_wykładowca); };

```

```

class Instytut
( extent instytuty
  key nazwai)
{ attribute string nazwai;
  attribute string telefoni;
  attribute string sekretariat;
  attribute string wydział;
  attribute Pracownik_Naukowy dyrektor;
  relationship set<Pracownik_Naukowy> ma_wykladowcę inverse
    Pracownik_Naukowy::pracuje_w;
  relationship set<Student> studiują inverse Student::na_kierunku;
  relationship set<Przedmiot> oferuje inverse Przedmiot::dostępny_w; };

class Przedmiot
( extent przedmioty
  key numerp )
{ attribute string nazwap;
  attribute string numerp;
  attribute string opis;
  relationship set<Kurs> ma_kursy inverse Kurs::przedmiotu;
  relationship <Instytut> dostępny_w inverse Instytut::oferuje; };

class Kurs
( extent kursy )
{ attribute short numer_kursu;
  attribute string rok;
  attribute enum Semestr {zimowy, letni} semestr;
  relationship set<Ocena> studenci inverse Ocena::kurs;
  relationship Przedmiot przedmiotu inverse Przedmiot::ma_kursy; };

class BieżącyKurs extends Kurs
( extent bieżące_kursy )
{ relationship set<Student> zapisani_studenci inverse Student::zapisany_do;
  void zapisz_studenta(in string pesel) raises(błędny_student, kurs_pełny); };

```

RYSUNEK 12.10. Propozycja schematu ODL dla bazy danych UNIWERSYTET z rysunku 12.9(b)

Klasy Wydział, Przedmiot, Kurs i BieżącyKurs z rysunku 12.10 są prostymi odwzorowaniami odpowiednich typów encji z rysunku 12.9(b).

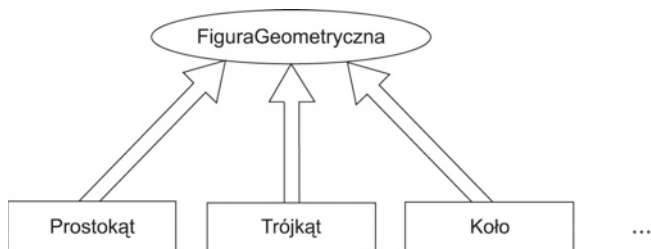
Klasa Ocena wymaga jednak dodatkowych objaśnień. Odpowiada ona związkowi  $M:N$  pomiędzy obiektami Student i Kurs z rysunku 12.9(b). Związek ten jest reprezentowany za pomocą osobnej klasy (a nie pary wzajemnych odwzorowań), ponieważ jest jej przypisany atrybut Ocena<sup>32</sup>.

Stąd, związek  $M:N$  jest odwzorowany klasą Ocena i parą związków typu  $1:N$  pomiędzy obiektami Student i Ocena oraz Kurs i Ocena<sup>33</sup>. Te dwa związki są przedstawiane za pomocą własności związku ukończone\_kursy w klasie Student, student i kurs w klasie Ocena i studenci w klasie Kurs (patrz rysunek 12.10). Klasa Ocena jest wykorzystywana do przedstawienia złożonego, wielowartościowego atrybutu oceny w obiektach typu Magistrant (patrz rysunek 8.10).

<sup>32</sup> Inne sposoby odwzorowywania atrybutów związku omówimy w podrozdziale 12.4.

<sup>33</sup> Takie odwzorowanie jest podobne do odwzorowań związków  $M:N$  w modelu relacyjnym (patrz podrozdział 9.1) i w tradycyjnym modelu sieciowym.

Ponieważ w poprzednim przykładzie użyliśmy tylko definicji klas, pomijając interfejsy, proponujemy teraz inny przykład ilustrujący użycie interfejsów i dziedziczenia zachowań. Rysunek 12.11(a) jest częścią bazy danych służącej do przechowywania obiektów geometrycznych. Zdefiniowano interfejs *FiguraGeometryczna* z operacjami pozwalającymi na obliczanie własności powierzchni i obwód oraz operacjami przesun i obróć pozwalającymi na manipulację figurami. Interfejs *FiguraGeometryczna* jest dziedziczony przez kilka różnych klas (*Prostokąt*, *Trójkąt*, *Koło*, ...). Ponieważ *FiguraGeometryczna* jest interfejsem, nie można tworzyć obiektów bezpośrednio na podstawie jej definicji. Obiekty typu *Prostokąt*, *Trójkąt*, *Koło*, ... mogą jednak być tworzone i każdy z tych obiektów dziedziczy wszystkie operacje interfejsu *FiguraGeometryczna*. Należy zwrócić uwagę, że w przypadku dziedziczenia po interfejsie dziedziczone są jedynie operacje, a nie własności (atrybuty i związki). Wynika z tego, że w sytuacji, kiedy wymagane jest dziedziczenie własności, to musi ona być ponownie zadeklarowana w definicji klasy (tak jak to ma miejsce z własnością *punkt\_zaczeplenia* na rysunku 12.11(b)). Zauważmy, że dziedziczone operacje mogą mieć różne implementacje w różnych klasach. Na przykład implementacje operacji *powierzchnia* i *obwód* mogą być różne dla obiektów typu *Prostokąt*, *Trójkąt* i *Koło*.



```

interface FiguraGeometryczna
{
    attribute enum Kształt {Prostokąt, Trójkąt, Koło, ...} kształt;
    attribute struct Punkt {short x, short y} punkt_zaczeplenia;
    float obwód();
    float powierzchnia();
    void przesun(in short przesun_x; in short przesun_y);
    void obróć(in float ką_obrotu);
};

class Prostokąt : FiguraGeometryczna
( extent prostokąty )
{
    attribute struct Punkt {short x, short y} punkt_zaczeplenia;
    attribute short długość;
    attribute short wysokość;
    attribute float ką_obrotu; };

class Trójkąt : FiguraGeometryczna
( extent trójkąty )
{
    attribute struct Punkt {short x, short y} punkt_zaczeplenia;
    attribute short bok_1;
    attribute short bok_2;
    attribute float ką_bok1_bok2;
    attribute float ką_obrotu_bok1; };
  
```

```
class Koło : FiguraGeometryczna
( extent koła )
{ attribute struct Punkt (short x, short y) punkt_zaczeplenia;
  attribute short promień; };
...
```

RYSUNEK 12.11. Ilustracja dziedziczenia przez operator „:”. (a) Graficzna reprezentacja schematu. (b) Odpowiednie definicje klas i interfejsów w języku ODL

Istnieje możliwość *wielokrotnego dziedziczenia* interfejsów przez klasę, a także interfejsów przez inny interfejs. Jednakże, w przypadku dziedziczenia typu EXTENDS (dziedziczenie klas), wielokrotne dziedziczenie *nie jest dozwolone*. Klasa może więc dziedziczyć przez operator extends tylko po jednej klasie (poza dziedziczeniem po dowolnej liczbie interfejsów).

## 12.4. Projektowanie koncepcyjne obiektowej bazy danych

W punkcie 12.4.1 pokazujemy, czym projektowanie obiektowej bazy danych różni się od projektowania relacyjnej bazy danych. Punkt 12.4.2 przedstawia algorytm odwzorowujący, którego można używać przy tworzeniu opisywanego definicjami klas w języku ODL schematu obiektowej bazy danych na podstawie schematu bazy EER.

### 12.4.1. Różnice pomiędzy koncepcyjnym projektowaniem obiektowych i relacyjnych baz danych

Jedną z podstawowych różnic w projektowaniu obiektowych i relacyjnych baz danych jest odmienne traktowanie związków. W obiektowych bazach danych związki są zazwyczaj reprezentowane przez właściwości związków lub atrybuty wskaźnikowe przechowujące identyfikatory OID powiązanych obiektów. Możemy je nazwać *wskaźnikami OID* do powiązanych obiektów, przy czym dozwolone jest zarówno używanie pojedynczych wskaźników, jak i ich kolekcji. Wskaźniki w związkach binarnych mogą być deklarowane w jednym lub w obu kierunkach, w zależności od przewidywanego sposobu dostępu do obiektów w związku. Jeśli wskaźniki są zadeklarowane w obu kierunkach, to mogą być deklarowane jako wzajemne odwrotności, przez co wymuszany jest obiektowy odpowiednik więzów integralności odwołań znanych z relacyjnych baz danych.

W relacyjnej bazie danych związki pomiędzy rekordami (krotkami) są określone przez atrybuty zawierające pasujące wartości. Można o nich mówić jako o *wskaźnikach wartości*, a są określane poprzez *klucze obce*, które są wartościami kluczy podstawowych powtórzonych w polach rekordów powiązanych związkiem. Pola te są ograniczone do pojedynczych wartości, ponieważ w standardowym modelu relacyjnym pole pojedynczego rekordu nie może przechowywać wielu wartości. Stąd związki typu *M:N* nie mogą być przedstawiane bezpośrednio, ale wymagają osobnej relacji (tabeli; zostało to dokładniej omówione w podrozdziale 9.1).

Odwzorowanie binarnych związków posiadających atrybuty nie jest w obiektowych bazach danych bezpośrednie, ponieważ projektant bazy musi wybrać stronę związku, której należy przypisać atrybuty. Jeśli atrybuty są przypisane obu stronom związku, powstanie nadmiarowość, która może prowadzić do niespójności danych. W niektórych przypadkach zaleca się więc relacyjne podejście opierające się na dodatkowej tabeli, poprzez stworzenie osobnej klasy reprezentującej związek. Takie podejście może być również stosowane do przedstawiania związków  $n$ -arnych stopnia wyższego niż 2.

Innym obszarem istotnie różniącym projektowanie obiektowych i relacyjnych baz danych jest kwestia dziedziczenia. W obiektowej bazie danych dziedziczenie jest wbudowane w model, więc jego reprezentacja jest wprost dostępna poprzez operatory „:” i EXTENDS. Ponieważ w relacyjnym modelu nie istnieją wbudowane struktury dziedziczenia, istnieje kilka opcji do wyboru przez projektanta (omówiliśmy je w podpunkcie 9.2). Należy jednak zauważyć, że systemy obiektowo-relacyjne oraz rozszerzone systemy relacyjne wprowadzają funkcje pozwalające na bezpośrednie oddanie tych struktur i na przypisywanie operacji abstrakcyjnym typom danych (patrz podrozdział 12.2).

Trzecią podstawową różnicą jest konieczność definiowania operacji w obiektowej bazie danych już w fazie projektowej, jako że są one częścią definicji klas. Co prawda we wszystkich typach baz danych istotne jest jak najwcześniejsze określenie operacji, ale w relacyjnych bazach danych nie jest to ściśle wymagane przed fazą implementacji systemu.

Występuje też „filozoficzna” różnica między modelem relacyjnym a modelem obiektowym, związana ze specyfikacją operacji. W modelu relacyjnym projektanci bazy *nie muszą* wstępnie definiować zestawu prawidłowych zachowań lub operacji, natomiast w modelu obiektowym tworzenie takich definicji jest niepisanywym wymogiem. Jedną z przytaczanych zalet modelu relacyjnego jest obsługa zapytań i transakcji *ad hoc*, choć są one sprzeczne z zasadą enkapsulacji.

W praktyce często się zdarza, że zespoły projektujące bazy danych stosują techniki obiektowe we wczesnych fazach projektowania koncepcyjnego, aby uwzględnić zarówno strukturę, jak i korzystanie z danych oraz operacje na nich. W fazie projektowania koncepcyjnego powstaje wtedy kompletna specyfikacja. Takie specyfikacje są potem odwzorowywane na schematy relacyjne, ograniczenia i elementy związane z operacjami, np. wyzwalacze i procedury składowane (patrz podrozdziały 5.2 i 13.4).

## 12.4.2. Odwzorowywanie schematu EER na schemat obiektowy

Odwzorowywanie schematu EER na deklaracje klas w obiektowej bazie danych jest dość proste pod warunkiem, że początkowy schemat nie zawiera *ani* kategorii, *ani* związków stopnia wyższego niż 2. Jednakże diagram EER nie zawiera operacji klas, dlatego muszą być one określone już po odwzorowaniu struktur danych. Zarys przechodzenia z bazy EER do obiektowej wygląda następująco:

**Krok 1.** Utwórz klasę ODL dla każdego typu encji lub podklasy EER. Typ klasy ODL powinien zawierać wszystkie atrybuty klasy EER<sup>34</sup>. *Atrybuty wielowartościowe* są dekla-

---

<sup>34</sup> Powoduje to niejawne zastosowanie konstruktora krotek na najwyższym poziomie deklaracji typów, ale w ogólności konstruktor ten nie jest jawnie wymieniany w deklaracji klasy w języku ODL.

rowane przy użyciu konstruktorów `set`, `bag` lub `list`<sup>35</sup>. Jeśli wartości atrybutu wielowartościowego dla danego obiektu powinny być uporządkowane, to należy użyć konstruktora listy; jeśli dozwolone są duplikaty elementów, to należy użyć konstruktora wielozbioru, natomiast w pozostałych przypadkach używany jest konstruktor zbioru. *Atrybuty złożone* są odwzorowywane przy pomocy konstruktora krotek (przy użyciu w ODL deklaracji `struct`).

Zadeklaruj ekstensję dla każdej klasy i określ atrybuty kluczowe jako jej klucze.

**Krok 2.** Dodaj własności związków oraz atrybuty wskaźnikowe dla wszystkich związków *binarnych* do każdej klasy ODL będącej elementem takiego związku. Można te własności dodać w jedną lub obie strony. Jeśli związek binarny jest przedstawiany przez odwołania w *obu* kierunkach, zadeklaruj odwołania do własności związku jako ich wzajemne związki odwrotne, o ile system baz danych to umożliwia.<sup>36</sup> Jeśli związek binarny jest reprezentowany tylko w *jednym* kierunku, to zdefiniuj typ atrybutu odwołującego się do drugiego obiektu w związku jako nazwę klasy, do której się odwołuje.

W zależności od proporcji licznosci w związkach binarnych, właściwości związków lub atrybuty wskaźnikowe mogą przechowywać pojedyncze wartości lub ich kolekcje. W przypadku związków typu  $N:1$  i  $1:1$  będą przechowywać **pojedyncze wartości**, a w przypadku związków  $1:N$  i  $N:M$  będą kolekcjami (typu zbiór lub lista)<sup>37</sup>. Alternatywna metoda odwzorowywania związków typu  $M:N$  jest omówiona w kroku 7. poniżej.

Jeśli związkowi przypisane są atrybuty, to można użyć konstruktora krotek (`struct`) do stworzenia struktur typu `<odwołanie, atrybut związku>`, które mogą być używane zamiast atrybutów wskaźnikowych. Taka konstrukcja nie pozwala jednak na używanie więzów odwrotności (ang. *inverse constraint*). Jeśli takie atrybuty są przechowywane w *obu kierunkach*, to te same dane będą przechowywane podwójnie, powodując nadmiarowość.

**Krok 3.** Dla każdej klasy dodaj odpowiednie operacje. Nie są one dostępne w schemacie EER i muszą być określone przez projektanta bazy na podstawie początkowych założeń systemu. Konstruktor powinien zawierać fragment sprawdzający wszystkie więzy (ograniczenia) nowo tworzonego obiektu. Destruktor powinien sprawdzać więzy, które mogą być naruszone poprzez usunięcie obiektu. Inne metody powinny zawierać fragmenty kodu sprawdzające odpowiednie więzy.

**Krok 4.** Klasy ODL odpowiadające podklasom w schemacie EER dziedziczą atrybuty, związki i metody swojej nadklasy w schemacie ODL (przez `extends`). Ich *specyficzne* (nie-dziedziczone) atrybuty, wskaźniki związków i operacje są określone w sposób opisany w krokach 1., 2. i 3.

**Krok 5.** Słabe typy encji można odwzorowywać w ten sam sposób co zwyczajne typy encji. Słabe typy encji, które nie są powiązane żadnym związkiem poza ich związkiem identyfikującym, mogą być też przedstawione w inny sposób: można je odwzorować

---

<sup>35</sup> Wymagana jest analiza pierwotnych wymagań konkretnej aplikacji bazy danych, ponieważ ze schematu EER nie wynika, którego konstruktora kolekcji należy użyć.

<sup>36</sup> Standard ODL daje możliwość jawnego zdefiniowania związków odwrotnych. Niektóre obiektowe systemy baz danych nie obsługują jednak tej możliwości. W takiej sytuacji programiści muszą jawnie zaprogramować metody, które aktualizują odwołania w odpowiednich obiektach.

<sup>37</sup> Informacja pozwalająca na wybór listy lub zbioru nie jest dostępna w schemacie EER i decyzję należy podejmować w oparciu o początkowe założenia systemu.



tak, jakby były *złożonymi atrybutami wielowartościowymi* typu encji, której są elementami. Służą do tego konstruktory `set<struct<...>>` lub `list<struct<...>>`. Atrybuty słabej encji są zawarte w konstrukcji `struct<...>`, która odpowiada konstruktorowi krotek. Atrybuty są odwzorowywane tak jak w krokach 1. i 2.

**Krok 6.** Kategorie (unie) w schemacie EER są trudne do odwzorowania w ODL. Można użyć odwzorowania podobnego do odwzorowania schematu EER w schemat relacyjny (patrz podrozdział 9.2), deklarując klasę reprezentującą kategorię i definiując związek 1:1 pomiędzy kategorią i każdą z jej nadklas.

**Krok 7.** Związek stopnia wyższego niż 2. może być odwzorowany na osobną klasę z odpowiednimi odwołaniami do każdej klasy będącej elementem związku. Odwołania te bazują na odwzorowaniu związku typu 1:N z każdej klasy reprezentującej typ encji należącej do związku na klasę reprezentującą związek *n*-tego stopnia. Tego samego sposobu mapowania można użyć dla związków binarnych typu *M:N*, które posiadają dodatkowe atrybuty.

Powyższe odwzorowanie zostało zastosowane do podzbioru schematu bazy danych UNIWERSYTET z rysunku 4.10 w kontekście standardu ODMG dla obiektowych baz danych. Odwzorowany schemat został przedstawiony w notacji ODL na rysunku 12.10.

## 12.5. Obiektowy język zapytań OQL

**Obiektowy język zapytań (OQL)** jest językiem zapytań obiektowych baz danych zaproponowanym przez grupę ODMG. Jest przystosowany do ścisłej współpracy z językami programowania, dla których zostały zdefiniowane wiązania w standardzie ODMG, takimi jak C++, Smalltalk czy Java, co oznacza, że zapytanie OQL umieszczone w kodzie programu w jednym z tych języków może zwracać obiekty odpowiadające systemowi typów tego języka. Implementacje operacji klas w schemacie ODMG mogą być również programowane w wyżej wymienionych językach. Składnia zapytań OQL jest zbliżona do języka SQL, będącego standardem w relacyjnych systemach baz danych, przy czym jest ona wzbogacona o dodatkowe cechy odpowiadające pojęciom modelu ODMG takim jak tożsamość obiektu, obiekty złożone, operacje, dziedziczenie, polimorfizm i związki.

Zacznijmy od omówienia w punkcie 12.5.1 składni prostych zapytań OQL i zastosowania nazwanych obiektów lub ekstensji jako punktów wejścia do bazy danych. Następnie, w punkcie 12.5.2, omówimy strukturę wyników zapytań i wyrażenia ścieżkowe do poruszania się pomiędzy obiektami powiązanych związkiem. Inne cechy OQL obsługujące tożsamość obiektu, dziedziczenie, polimorfizm i inne pojęcia stosowane w obiektowych bazach danych przedstawimy w punkcie 12.5.3. Przykłady ilustrujące zapytania OQL wykorzystują przykładowy schemat bazy danych UNIWERSYTET przedstawiony na rysunku 12.10.

## 12.5.1. Proste zapytania OQL, punkty wejścia bazy danych i zmienne iterujące

Podstawowa składnia zapytań OQL ma strukturę `select ... from ... where ...`, analogiczną jak w SQL. Na przykład, zapytanie wyszukujące nazwy wszystkich instytutów na wydziale *Inżynieria* mogłoby wyglądać następująco:

```
Z0: SELECT i.nazwai
      FROM i in instytuty
      WHERE i.wydział = 'Inżynieria';
```

Dla każdego zapytania wymagany jest **punkt wejścia** do bazy danych, którym może być dowolny *nazwany trwały obiekt*. Dla wielu zapytań punktem wejścia będzie nazwa ekstensji klasy. Przypomnijmy, że nazwa ekstensji jest nazwą trwałego obiektu będącego kolekcją (w większości przypadków jest to zbiór) obiektów wybranej klasy. Przeglądając rysunek 12.10 zauważymy, że nazwany obiekt `instytuty` jest typu `set<Instytut>`; osoby jest typu `set<Osoba>`; wykładowcy — typu `set<Pracownik_Naukowy>` itd.

Użyta jako punkt wejścia do bazy danych w zapytaniu *Z0* nazwa ekstensji `instytuty` odnosi się do trwałej kolekcji obiektów. Za każdym razem, kiedy używamy w zapytaniu odwołania do kolekcji, należy zdefiniować **zmienną iterującą**<sup>38</sup> (w *Z0* jest to `i`), która obejmuje każdy obiekt w kolekcji. W wielu przypadkach, tak jak w *Z0*, zapytanie wybierze z kolekcji określone obiekty bazując na warunku określonym słowem kluczowym `where`. W przypadku *Z0*, zapytanie zwróci jedynie te trwałe obiekty z kolekcji `instytuty`, które spełniają warunek `i.wydział = 'Inżynieria'`. Dla każdego wybranego obiektu `i` w wyniku zapytania zwracana jest wartość `i.nazwai`. *Typem wyniku* zwracanego przez *Z0* jest więc `bag<string>`, ponieważ typem atrybutu `nazwai` jest `string` (choć rzeczywisty wynik jest zbiorem, ponieważ `nazwai` to atrybut klucza). Uogólniając, wynik zapytania `select ... from ...` będzie typu `bag`, a wynik zapytania `select distinct ... from ...` będzie typu `set` (podobnie jak w SQL, słowo kluczowe `distinct` eliminuje duplikaty z wyniku wyszukiwania).

Rozważając przykładowe zapytanie *Z0*, składnia OQL pozwala na trzy możliwości określenia zmiennej iterującej:

```
i in instytuty
instytuty i
instytuty as i
```

W naszych przykładach będziemy używali pierwszego zapisu<sup>39</sup>.

Używanie nazwanych obiektów jako punktów wejścia do bazy danych nie jest ograniczone do nazw ekstensji. Każdy nazwany trwały obiekt odnoszący się zarówno do atomowego (pojedynczego) obiektu, jak i do kolekcji może być użyty jako punkt wejścia.

<sup>38</sup> Jest to konstrukcja podobna do zmiennych krotek (rekordów) obejmujących zakres rekordów w zapytaniach SQL.

<sup>39</sup> Zauważmy, że druga i trzecia opcja są podobne do składni określającej zmienne krotek w zapytaniach SQL.

## 12.5.2. Wyniki zapytań i wyrażenia ścieżkowe

Ogólnie wynik zapytania może być dowolnego typu, który da się określić w modelu obiektowym ODMG. Zapytanie nie zawsze musi mieć strukturę `select ... from ... where ...`, w najprostszym przypadku sama trwała nazwa jest już zapytaniem, którego wynikiem jest odwołanie do tego obiektu. Na przykład, zapytanie

```
Z1: instytut;
```

zwraca odwołanie do kolekcji wszystkich trwałych obiektów typu `Instytut`, która jest typu `set<Instytut>`. Podobnie, po nadaniu nazwy `instytut_inf` (za pomocą operacji `bind`, patrz rysunek 12.8) pojedynczemu obiektowi `Instytut` (`instytutowi informatyki`), zapytanie

```
Z1a: instytut_inf;
```

zwróci odwołanie do pojedynczego obiektu `Instytut`. Po określeniu punktu wejścia możemy użyć pojęcia **wyrażenia ścieżkowego** (ang. *path expression*) do określenia ścieżki do powiązanych atrybutów i obiektów. Wyrażenie ścieżkowe zazwyczaj rozpoczyna się *nazwą trwałego obiektu* lub zmienną iterującą obejmującą określony zakres obiektów z kolekcji. Po pierwszej nazwie następuje ciąg zera lub nazw związków lub atrybutów połączonych *znakiem kropki*. Odnosząc się do przykładowej bazy danych `UNIwersytet` z rysunku 12.10, możemy podać następujące przykłady wyrażeń ścieżkowych będących prawidłowymi zapytaniami OQL:

```
Z2: instytut_inf.dyrektor;  
Z2a: instytut_inf.dyrektor.stanowisko;  
Z2b: instytut_inf.ma_wykladowcę;
```

Pierwsze wyrażenie `Z2` zwraca obiekt typu `Pracownik_Naukowy`, ponieważ taki jest typ atrybutu `dyrektor` w klasie `Instytut`. Będzie to odwołanie do obiektu `Pracownik_Naukowy`, który jest powiązany z obiektem `Instytut` o trwałej nazwie `instytut_inf` poprzez jego atrybut `dyrektor`. Innymi słowy, będzie to odwołanie do obiektu `Pracownik_Naukowy` odpowiadającego dyrektorowi instytutu informatyki. Zapytanie `Z2a` jest podobne, przy czym zwraca atrybut `stanowisko` obiektu `Pracownik_Naukowy` (`dyrektor instytutu informatyki`), a nie odwołanie do samego obiektu. Rezultatem zapytania `Z2a` jest więc `string`, ponieważ taki jest typ atrybutu `stanowisko` w klasie `Pracownik_Naukowy`.

Wyrażenia ścieżkowe `Z2` i `Z2a` zwracają pojedyncze wartości, ponieważ atrybuty `dyrektor` (klasy `Instytut`) i `stanowisko` (klasy `Pracownik_Naukowy`) przechowują pojedyncze wartości i odnoszą się do pojedynczych obiektów. Trzecie wyrażenie — `Z2b` — jest inne: zwraca obiekt typu `set<Instytut>`, nawet jeśli będzie zastosowane do pojedynczego obiektu, ponieważ właśnie taki jest typ związku `ma_wykladowcę` w klasie `Instytut`. Zwrócona kolekcja będzie zawierała odwołania do wszystkich obiektów typu `Pracownik_Naukowy`, które są powiązane poprzez związek `ma_wykladowcę` z trwałym obiektem o nazwie `instytut_inf`, czyli odwołania do wszystkich obiektów typu `Pracownik_Naukowy`, którym odpowiadają pracownicy instytutu informatyki. Aby otrzymać dane o stanowiskach wszystkich pracowników instytutu informatyki, nie możemy napisać:

```
Z3': instytut_inf.ma_wykladowcę.stanowisko;
```

ponieważ nie jest jasne, czy otrzymany obiekt będzie typu `set<string>`, czy `bag<string>` (prawdopodobnie będzie to druga możliwość, ponieważ większość pracowników będzie miała takie samo stanowisko). W związku z problemem wieloznaczności OQL nie zezwala

na wyrażenia podobne do tego zaprezentowanego w przykładzie Z3'. Użytkownik powinien raczej użyć zmiennej iterującej w danej kolekcji, co pokazano w przykładach Z3a i Z3b:

```
Z3a: select p.stanowisko
      from p in instytut_inf.ma_wykładowcę;
Z3b: select distinct p.stanowisko
      from p in instytut_inf.ma_wykładowcę;
```

Wynik zapytania Z3a jest typu `bag<string>` (może przechowywać kilka takich samych stanowisk), podczas gdy Z3b zwróci `set<string>` (duplikaty zostały wyeliminowane przy pomocy słowa kluczowego `distinct`). Zarówno Z3a, jak i Z3b ilustrują, jak można w wyrażeniu `from` zdefiniować zmienną iterującą obejmującą zakresem wszystkie elementy z kolekcji określonej w zapytaniu. Zmienna `p` w zapytaniach Z3a i Z3b obejmuje wszystkie elementy kolekcji `instytut_inf.ma_wykładowcę`, która jest typu `set<Pracownik_Naukowy>` i zawiera tylko pracowników pracujących w instytucie informatyki.

Ogólnie zapytanie OQL może zwracać dowolnie złożoną strukturę opisaną w samym zapytaniu przy pomocy słowa kluczowego `struct`. Rozważmy następujące przykłady:

```
Z4: instytut_inf.dyrektor.nadzoruje;
Z4a: select struct ( imięnazwisko: struct (
                        nazwisko:s.imięnazwisko.nazwisko,
                        imię:s.imięnazwisko.imię),
                    oceny: (select struct (ocena: o.ocena,
                                           data_wpisu: o.data,
                                           wydział: o.wydział)
                           from o in s.oceny)
                    from s in instytut_inf.dyrektor.nadzoruje;
```

Zapytanie Z4 jest dość proste i zwraca obiekt typu `set<Magistrant>`. Jest to kolekcja studentów ostatniego roku, których promotorem jest dyrektor wydziału informatyki. Załóżmy teraz, że chcemy uzyskać imiona i nazwiska tych studentów wraz z listą ocen każdego z nich. Możemy to osiągnąć przy pomocy zapytania Z4a, gdzie zmienna `s` obejmuje kolekcję magistrantów, których promotorem jest dyrektor, a zmienna `o` obejmuje listę ocen dla każdego studenta `s`. Rezultat Z4a jest kolekcją struktur, gdzie każda z nich składa się z dwóch (na pierwszym poziomie) części składowych: `imięnazwisko` oraz `oceny`<sup>40</sup>.

`imięnazwisko` jest również strukturą składającą się z pól `imię` i `nazwisko`, z których każde jest typu `string`. Pole `oceny` jest zdefiniowane jako zapytanie zagnieżdżone i jest kolekcją kolejnych struktur, z których każda składa się z trzech komponentów: `ocena`, `data_wpisu` i `wydział`.

Zauważmy, że OQL jest *ortogonalny* w stosunku do określania wyrażen ścieżkowych. Oznacza to, że wewnątrz wyrażenia ścieżkowego można dowolnie używać atrybutów, związków i nazw operacji (metod), o ile tylko zachowuje się zgodność typów OQL. Przykładowo, zapytanie pobierające średnią ocen wszystkich studentów kończących kierunek informatyka, posortowane według średniej ocen, a następnie według nazwisk i imion, można zapisać następująco:

---

<sup>40</sup> Jak już zostało wspomniane wcześniej, wyrażenie `struct` odpowiada konstruktorowi krotek opisanemu w punkcie 12.1.3.

```

Z5a: select struct (nazwisko: s.imięnazwisko.nazwisko, imię:
s.imięnazwisko.imię,
        średnia_ocen: s.średnia_ocen)
    from s in instytut_inf.studiują
   where s.rokst = 'ostatni';
   order by średnia_ocen desc, nazwisko asc, imię asc;
Z5b: select struct (nazwisko: s.imięnazwisko.nazwisko, imię:
s.imięnazwisko.imię,
        średnia_ocen: s.średnia_ocen)
    from s in studenci
   where s.na_kierunku.nazwai = 'Informatyka' and s.rokst='ostatni'
   order by średnia_ocen desc, nazwisko asc, imię asc;

```

Zapytanie *Z5a* używa jako punktu wejścia obiektu `instytut_inf`, dzięki czemu uzyskuje dostęp bezpośrednio do obiektu reprezentującego instytut informatyki, a następnie odnajduje studentów poprzez związek `studiują`. Zapytanie *Z5b* natomiast przeszukuje ekstensję `studenci` i odnajduje studentów ostatniego roku kończących ten instytut. Nazwy atrybutów, związków i operacji (metod) są używane w wyrażeniach ścieżkowych wymiennie (ortogonalnie): `średnia_ocen` jest operacją, `studiują` i `na_kierunku` są związkami, a `rokt`, `nazwai`, `nazwisko` i `imię` atrybutami. Implementacja operacji `średnia_ocen` oblicza średnią ocen i zwraca wartość typu `float` dla każdego wybranego studenta.

Wyrażenie `order by` jest podobne do odpowiadającej mu konstrukcji w SQL i określa, w jakiej kolejności mają być wyświetlone wyniki zapytania. Kolekcja zwracana przez zapytanie używające tego wyrażenia jest więc zawsze typu `list`.

### 12.5.3. Inne cechy OQL

**Określanie perspektyw jako nazwanych zapytań.** Mechanizm perspektyw korzysta z koncepcji **nazwanych zapytań**. Do zadeklarowania identyfikatora nazwanego zapytania używane jest słowo kluczowe `define`. Nazwa identyfikatora musi być unikalna wśród wszystkich nazwanych obiektów, nazw klas, metod i funkcji ze schematu bazy danych. Jeśli identyfikator jest taki sam jak identyfikator wcześniej określonego zapytania, to nowa definicja zastępuje starą. Definicja nazwanego zapytania istnieje dopóty, dopóki nie zostanie usunięta lub zastąpiona nową. Definicja perspektywy może też obejmować parametry (argumenty).

Przedstawiona poniżej przykładowa perspektywa *W1* definiuje nazwane zapytanie `specjalizują_się_na`, aby odnaleźć zbiór obiektów odpowiadających studentom o specjalności na danym wydziale:

```

W1: define specjalizują_się_na(nazwa_inst) as
    select s
    from s in studenci
   where s.specjalność_w.nazwai = nazwa_inst;

```

Ponieważ schemat ODL z rysunku 12.10 definiuje dla obiektów typu `Student` jedynie jednokierunkowy atrybut `specjalność_w`, możemy użyć powyższej perspektywy do przedstawienia odwrotności tego atrybutu bez potrzeby jawnego określania związku. Ten rodzaj perspektyw może być używany do przedstawiania rzadziej używanych odwrotności zwią-

ków. Użytkownik może teraz wykorzystać powyższą perspektywę do pisania wyrażeń takich jak

```
specjalizują_się_na('Informatyka');
```

które zwróci wielozbiór studentów będących na specjalności w instytucie informatyki. Zwróćmy uwagę, że na rysunku 12.10 został jawnie zadeklarowany związek studiują, co wynika z założenia, że ten związek będzie znacznie częściej wykorzystywany.

**Wydobywanie pojedynczych elementów z kolekcji jednoelementowych.** Każde zapytanie OQL zwróci jako wynik kolekcję będącą wielozbiorem, zbiorem (jeśli zostało użyte słowo kluczowe *distinct*) albo uporządkowaną listę (jeśli zostało użyte wyrażenie *order by*). Jeśli chcemy, żeby zapytanie zwróciło tylko jeden element, to należy użyć operatora *element*, który spowoduje, że wynikiem będzie pojedynczy element *e* z jednoelementowej kolekcji *k*. Jeśli *k* zawiera więcej niż jeden element lub jest pusta, to operator *element* zgłasza wyjątek. Przykładowo, zapytanie Z6 zwraca pojedynczy element odwołujący się do instytutu informatyki:

```
Z6: element (select i
             from i in instytuty
             where i.nazwai = 'Informatyka');
```

Ponieważ nazwa instytutu jest unikatowa, to rezultatem powinien być jeden instytut. Wynik jest typu *i:Instytut*.

**Operatory kolekcji (funkcje agregujące, kwantyfikatory).** Ponieważ wynikiem wielu zapytań są kolekcje, zdefiniowano operatory pozwalające na operacje na kolekcjach. Są to operatory agregujące, członkostwa oraz kwantyfikatory (uniwersalny i egzystencjalny).

Operatory agregujące (*min*, *max*, *count*, *sum* i *avg*) operują na kolekcjach<sup>41</sup>. *Count* zwraca liczbę całkowitą, a pozostałe operatory (*min*, *max*, *sum* i *avg*) zwracają taki sam typ, jakiego są elementy danej kolekcji. Poniżej przytaczamy dwa przykłady: zapytanie Z7 zwraca liczbę studentów specjalizujących się na wydziale 'Informatyka', a Z8 zwraca średnią ocen dla całego ostatniego roku studentów na kierunku informatyka.

```
Z7: count (s in specjalizują_się_na('Informatyka'));
Z8: avg (select s.średnia_ocen
         from s in studenci
         where s.na_kierunku.nazwai = 'Informatyka' and s.rokst = 'ostatni');
```

Zauważmy, że operatory agregujące mogą być użyte w stosunku do dowolnej kolekcji odpowiedniego typu i w dowolnym miejscu zapytania. Na przykład, odnajdywanie nazwy wszystkich instytutów, które mają więcej niż 100 studentów, można zrealizować przy pomocy następującego zapytania Z9:

```
Z9: select i.nazwai
     from i in instytuty
     where count (i.studiują) > 100;
```

Wyrażenia *członkostwa* i *kwantyfikatory* zwracają typ logiczny (czyli prawdę albo fałsz). Niech *z* będzie zmienną, *k* wyrażeniem kolekcji, *l* wyrażeniem typu logicznego (czyli warunkiem logicznym), a *e* — elementem tego samego typu co elementy kolekcji *k*. Wtedy

---

<sup>41</sup> Są one odpowiednikami funkcji agregujących w SQL.

(e in k) zwraca prawdę, jeżeli element e należy do kolekcji k;

(for all z in k: l) zwraca prawdę, jeśli *wszystkie* elementy z kolekcji k spełniają l;

(exists z in k: l) zwraca prawdę, jeżeli istnieje co najmniej jeden element z k spełniający l.

Aby zilustrować warunek członkostwa, przypuśćmy, że chcemy uzyskać listę wszystkich studentów, którzy ukończyli przedmiot "Bazy danych I". Takie informacje zwróci zapytanie *Z10*, w którym zagnieżdżone zapytanie zwraca kolekcję przedmiotów ukończonych przez każdego studenta s, a warunek członkostwa zwraca prawdę, jeśli "Bazy danych I" są elementem takiej kolekcji dla określonego studenta s:

```
Z10: select s.imięnazwisko.imię, s.imięnazwisko.nazwisko
      from s in studenci
      where "Bazy danych I" in
        (select p.nazwap
         from k in s.ukończone_kursy.kursy.przedmiotu);
```

*Z10* demonstruje również prostszy sposób na określanie zapytań zwracających kolekcje krotek; zauważmy, że wynik *Z10* jest typu `bag<struct(string, string)>`.

Można również tworzyć zapytania zwracające prawdę albo fałsz. Dla przykładu przyjmijmy, że istnieje nazwany obiekt *Jerzy* typu `Student`. Zapytanie *Z11* odpowie na pytanie „czy specjalnością Jerzego jest informatyka?”. Zapytanie *Z12* odpowie natomiast na pytanie „czy wszyscy magistranci z instytutu informatyki mają promotorów z tego wydziału?”. Zarówno *Z11*, jak i *Z12* zwracają prawdę albo fałsz, które są rozumiane jako twierdząca lub przecząca odpowiedź na powyższe pytania:

```
Z11: Jerzy in specjalizują_się_na('Informatyka');
Z12: for all m in
      (select s
       from s in magistranci
       where s.na_kierunku.nazwai = 'Informatyka')
      :m.promotor in instytut_inf.ma_wykladowcę;
```

Zapytanie *Z12* obrazuje również, jak odnosi się do zapytań dziedziczenie atrybutów, związków i operacji. Pomimo że s jest iteratorem nad ekstensją magistranci, możemy napisać s.na\_kierunku, ponieważ związek na\_kierunku jest dziedziczony przez obiekty `Magistrant` po klasie `Student` za pomocą operatora `extends` (patrz rysunek 12.10). Zastosowanie kwantyfikatora `exists` ilustruje zapytanie *Z13* odpowiadające na pytanie „czy którykolwiek z magistrantów informatyki ma średnią ocen równą 4,0?”. Operacja `średnia_ocen` jest znowu dziedziczona przez obiekty `Magistrant` po klasie `Student` dzięki operatorowi `extends`.

```
Z13: exists m in
      (select s
       from s in magistranci
       where s.na_kierunku.nazwai = 'Informatyka')
      :m.średnia_ocen = 4;
```

**Uporządkowane (indeksowane) wyrażenia kolekcji.** Jak zostało wspomniane w punkcie 12.3.3, kolekcje takie jak listy i tablice posiadają dodatkowe operacje, np. pozwalające na pobieranie *i*-tego, pierwszego, czy ostatniego elementu. Poza tym istnieją operacje



pozwalające na wyodrębnianie podkolekcji lub łączenie dwóch list w jedną. Takie operacje mogą być wykorzystywane w wyrażeniach dotyczących list lub tablic. Pokażemy niektóre z dostępnych operacji przy pomocy przykładowych zapytań. *Z14* pobiera nazwisko pracownika naukowego, który otrzymuje najwyższe wynagrodzenie:

```
Z14: first (select struct(pracownik_naukowy: w.imięnazwisko.nazwisko, pensja:
    w.pensja)
    from w in wykładowcy
    order by pensja desc);
```

*Z14* ilustruje użycie operatora `first` do listy zawierającej wynagrodzenia pracowników naukowych uporządkowane malejąco. Stąd pierwszy element takiej listy zawiera wykładowcę o najwyższej pensji. To zapytanie przyjmuje, że tylko jeden z pracowników naukowych ma najwyższą pensję. Następne zapytanie — *Z15* — wyszukuje troje studentów informatyki o najwyższej średniej ocen.

```
Z15: (select struct (nazwisko:s.imięnazwisko.nazwisko, imię:
    s.imięnazwisko.imię,
    średnia:s.średnia_ocen)
    from s in instytut_inf.studiuja
    order by średnia desc) [0:2];
```

Typowe zapytanie `select-from-order-by` zwraca listę studentów informatyki uporządkowaną według średniej ich ocen. Pierwszy element kolekcji ma numer indeksu równy 0, więc wyrażenie `[0:2]` zwraca listę zawierającą pierwszy, drugi i trzeci element wyniku `select-from-order-by`.

**Operator grupowania.** Wyrażenie `group by` w OQL, pomimo że podobne do swojego odpowiednika w SQL, zapewnia jawne odwołanie do kolekcji obiektów wewnątrz każdej grupy lub partycji. Najpierw podamy przykład, a następnie opiszemy ogólną formę tego typu zapytań.

*Z16* zwraca liczbę studentów każdego instytutu. Studenci w tym zapytaniu są przypisani do jednej grupy (partycji), jeżeli studiują ten sam kierunek (czyli mają przypisaną taką samą wartość `s.na_kierunku.nazwai`):

```
Z16: select struct(nazwa_instytutu, liczba_studentów: count(partition))
    from s in studenci
    group by nazwa_instytutu: s.na_kierunku.nazwai;
```

Wynikiem specyfikacji grupowania jest typ `set<struct(nazwa_instytutu: string, partition: bag<struct(s:Student)>>)`, który dla każdej grupy (partycji) zawiera krotkę (`struct`) składającą się z dwóch elementów: wartości atrybutu grupującego (`nazwa_instytutu`) oraz wielozbioru obiektów typu `Student` w grupie (partycji). Wyrażenie `select` zwraca atrybut grupujący (nazwę instytutu) oraz liczbę elementów w danej partycji (czyli liczbę studentów danego instytutu), przy czym `partition` jest słowem kluczowym odnoszącym się do każdej kolejnej partycji. Wynik całego zapytania jest typu `set<struct(nazwa_instytutu: string, liczba_studentów: integer)>`. Ogólnie składnia wyrażenia `group by` wygląda następująco:

```
group by f1: e1, f2:e2, ..., fk: ek
```

gdzie  $f_1: e_1, f_2: e_2, \dots, f_k: e_k$  jest listą atrybutów grupujących (tworzących partycje), a każde określenie atrybutu grupującego  $f_i: e_i$  definiuje nazwę atrybutu (pola)  $f_i$  oraz wyrażenie  $e_i$ . Wynikiem zastosowania grupowania (określonym w wyrażeniu `group by`) jest zbiór struktur:

```
set<struct(f1: t1, f2: t2, ..., fk: tk, partition: bag<B>>>
```

gdzie  $t_i$  jest typem zwracanym przez wyrażenie  $e_i$ , `partition` jest wyróżnioną nazwą pola (słowem kluczowym) a  $B$  jest strukturą, której pola są zmiennymi iterującymi ( $s$  w *Z16*) odpowiedniego typu zadeklarowanych w wyrażeniu `from`.

Podobnie jak w SQL wyrażenie `having` może być użyte do filtrowania pogrupowanych zbiorów (czyli wybierania tylko niektórych partycji na podstawie warunków grupowania). W *Z17* poprzednie zapytanie zostało zmodyfikowane w celu zilustrowania zastosowania słowa kluczowego `having` (a równocześnie prezentuje prostszą składnię wyrażenia `select`). *Z17* oblicza średnią wartość ze średnich ocen w wydziałach liczących więcej niż 100 studentów. Wyrażenie `having` zastosowane w *Z17* wybiera tylko te partycje (grupy), które zawierają więcej niż 100 elementów (czyli wydziały z więcej niż 100 studentami).

```
Z17: select nazwa_instytutu, sr_średnia_ocen: avg (select p.s.średnia_ocen from p
      in partition)
      from s in studenci
      group by nazwa_instytutu: s.na_kierunku.nazwai
      having count (partition) > 100;
```

Zauważmy, że wyrażenie `select` z *Z17* zwraca średnią ze średnich ocen dla studentów w partycji. Wyrażenie:

```
select p.s.średnia_ocen from p in partition
```

zwraca wielozbiór średnich ocen studentów dla danej partycji. Wyrażenie `from` deklaruje zmienną iterującą  $p$  nad kolekcją wszystkich partycji, która jest typu `bag<struct(s: Student)>`. Wyrażenie ścieżkowe `p.s.średnia_ocen` jest użyte, aby odczytać średnią ocen każdego studenta w danej partycji.

## 12.6. Przegląd wiązania z językiem C++ w standardzie ODMG

Wiązanie z językiem C++ określa mapowanie konstrukcji języka ODL na konstrukcje języka C++. Jest ono zrealizowane za pomocą biblioteki klas C++ dostarczającej definicji klas i operacji implementujących struktury ODL. Odczytywanie i manipulowanie obiektami bazy danych wewnątrz programu w języku C++ odbywa się przy pomocy *języka manipulacji obiektami* (OML — ang. *Object Manipulation Language*) bazującego na składni i semantyce języka C++. Programista ma również do dyspozycji, poza wiązaniami ODL/OML, zestaw konstrukcji nazywanych *dyrektywami fizycznymi*, pozwalających na kontrolę nad niektórymi fizycznymi aspektami bazy danych takimi jak grupowanie obiektów, używanie indeksów czy zarządzanie pamięcią.

Biblioteka klas obsługująca w C++ standard ODMG używa przedrostka `d_` do deklarowania klas mających związek z systemem baz danych<sup>42</sup>. Chodzi o to, aby programista miał wrażenie, że używa tylko jednego, a nie dwóch niezależnych języków. Dla każdej klasy `T` zdefiniowanej w bazie danych zdefiniowana jest klasa `d_Ref<T>`, za której pomocą programista może się odwoływać do obiektów klasy `T`. Zmienne w programie typu `d_Ref<T>` mogą się odnosić zarówno do trwałych, jak i ulotnych obiektów klasy `T`.

W celu umożliwienia używania typów wbudowanych w model obiektowy ODMG (np. typów kolekcji), biblioteka wprowadza zestaw szablonów klas. Na przykład, abstrakcyjna klasa `d_Object<T>` określa operacje dziedziczone przez wszystkie obiekty. Podobnie, abstrakcyjna klasa `d_Collection<T>` określa operacje kolekcji. Klasy te nie pozwalają na tworzenie instancji obiektów, określają jedynie operacje dziedziczone odpowiednio przez wszystkie obiekty lub kolekcje. Szablony są zdefiniowane dla każdego z rodzajów kolekcji, a są to między innymi: `d_Set<T>`, `d_List<T>`, `d_Bag<T>`, `d_Array<T>` oraz `d_Dictionary<T>`, odpowiadające typom kolekcji z modelu obiektowego ODMG (patrz punkt 12.3.1). Programista może więc tworzyć klasy takie jak `d_Set<d_Ref<Student>>`, których instancje byłyby zbiorem odwołań do obiektów typu `Student`, lub `d_Set<string>` reprezentujące zbiory ciągów znaków. Poza tym klasa `d_Iterator` odpowiada klasie `Iterator` z modelu obiektowego.

ODL w C++ pozwala użytkownikowi na opisywanie klas schematu bazy danych przy pomocy zarówno konstrukcji C++, jak i konstrukcji oferowanych przez bibliotekę klas bazy danych. Do określania typów atrybutów<sup>43</sup> zdefiniowane zostały podstawowe typy takie jak `d_Short` (short integer), `d_UShort` (unsigned short integer), `d_Long` (long integer) i `d_Float` (float). Poza typami podstawowymi biblioteka definiuje kilka typów odpowiadających typom literałów strukturalnych zdefiniowanych w standardzie ODMG. Są wśród nich `d_String`, `d_Interval`, `d_Date`, `d_Time` i `d_Timestamp` (patrz rysunek 12.5(b)).

Do określania związków stosuje się słowo kluczowe `Rel_` użyte jako przedrostek w nazwie typu. Na przykład pisząc:

```
d_Rel_Ref<Instytut, _studiują> na_kierunku;
```

w klasie `Student` oraz

```
d_Rel_Set<Student, _na_kierunku> studiują;
```

w klasie `Instytut`, deklarujemy `studiują` i `na_kierunku` jako właściwości związków, które są dla siebie wzajemnie odwrotne i reprezentują binarny związek 1:N pomiędzy obiektami typu `Instytut` i `Student`.

W języku OML wiązanie przeciąża operację `new`, która może być użyta do tworzenia zarówno trwałych, jak i ulotnych obiektów. Aby stworzyć trwały obiekt, należy podać nazwę bazy danych oraz trwałą nazwę obiektu w tej bazie. Na przykład, pisząc:

```
d_Ref<Student> s = new(DB1, 'Jan_Kowalski') Student;
```

programista tworzy nazwany trwały obiekt typu `Student` w bazie danych `DB1` i nadaje mu trwałą nazwę `Jan_Kowalski`. Inna operacja, `delete_object()`, jest używana do kasowania obiektów. Obiekt jest modyfikowany za pośrednictwem operacji (metod) zdefiniowanych dla każdej klasy przez programistę.

<sup>42</sup> Przedrostek `d_` oznacza klasy bazy danych.

<sup>43</sup> W terminologii obiektowej nazywanych *zmiennymi składowymi*.

Wiązanie C++ pozwala również na tworzenie ekstensji przy pomocy klasy `d_Extent`. Na przykład, pisząc:

```
d_Extent<Osoba> WszystkieOsoby(DB1);
```

programista stworzyłby w bazie DB1 nazwaną kolekcję `WszystkieOsoby`, typu `d_Set<Osoba>`, która przechowywałaby wszystkie trwałe obiekty typu `Osoba`. Wiązanie C++ nie obsługuje jednak ograniczeń kluczy, które muszą być zaimplementowane w metodach klas<sup>44</sup>. Nie jest również obsługiwane utrwalanie obiektów za pomocą osiągalności; obiekty muszą być zadeklarowane jako trwałe w momencie ich tworzenia.

## 12.7. Podsumowanie

W tym rozdziale zaczęliśmy od przeglądu w podrozdziale 12.1 elementów wykorzystywanych w obiektowych bazach danych. Opisaliśmy, jak opracowano te elementy na podstawie ogólnych zasad obiektowych. Główne z opisanych zagadnień to: tożsamość obiektów i identyfikatory, enkapsulacja operacji, dziedziczenie, złożone struktury obiektów tworzone za pomocą zagnieżdżania typów, utrwalanie obiektów.

Dalej, w podrozdziale 12.2, pokazaliśmy, jak liczne z tych elementów wbudowano w model relacyjny i standard SQL. Zobaczyłeś, że doprowadziło to do zwiększenia możliwości relacyjnych baz danych. Nowe systemy zostały nazwane bazami obiektowo-relacyjnymi.

Następnie omówiliśmy standard obiektowych baz danych ODMG 3.0. Zaczęliśmy od opisanie różnych elementów modelu obiektowego (podrozdział 12.3). Przedstawiliśmy różne typy wbudowane, takie jak `Object`, `Collection`, `Iterator`, `set`, `list` itd., opisując ich interfejsy, które określają wbudowane operacje każdego typu. Typy wbudowane są podstawą dla języków ODL i OQL. Opisaliśmy też różnicę między obiektami (posiadającymi identyfikator obiektu) a literałami (wartościami bez takiego identyfikatora). Użytkownicy mogą deklarować klasy aplikacji dziedziczące operacje po odpowiednich wbudowanych interfejsach. W klasach zdefiniowanych przez użytkownika można tworzyć dwa rodzaje cech: atrybuty i związki, a także operacje stosowane do obiektów danej klasy. ODL umożliwia użytkownikom tworzenie zarówno interfejsów, jak i klas oraz obsługuje dwa rodzaje dziedziczenia: dziedziczenie interfejsów (z użyciem symbolu `:`) i dziedziczenie klas (za pomocą słowa `extends`). Klasa może mieć ekstensje i klucze. Dalej znalazł się opis języka ODL, a do zilustrowania elementów tego języka posłużył schemat przykładowej bazy UNIWERSYTET.

Po omówieniu modelu obiektowego ODMG opisaliśmy ogólną technikę projektowania schematów obiektowych baz danych (podrozdział 12.4). Wyjaśniliśmy, że obiektowe bazy różnią się od baz relacyjnych w trzech głównych obszarach: używaniu odwołań do reprezentowania związków, dodaniu operacji i dziedziczeniu. Dalej pokazaliśmy, jak odwzorować koncepcyjny projekt bazy danych z modelu EER na elementy obiektowych baz danych.

---

<sup>44</sup> Wiązanie do C++ zostało tu opisane bardzo pobieżnie. Więcej szczegółów można znaleźć w pracy pod redakcją Cattella (2000), rozdział 5.

W podrozdziale 12.5 zaprezentowaliśmy język OQL. Umożliwia on ortogonalne tworzenie zapytań, co oznacza, że operację można zastosować do wyniku innej operacji (pod warunkiem, że typ wyniku jest odpowiednim typem danych wejściowych następnej operacji). Składnia OQL jest zgodna z wieloma elementami języka SQL, ale obejmuje dodatkowe rozwiązania, takie jak wyrażenia ścieżkowe, dziedziczenie, metody, związki i kolekcje. Przedstawiliśmy przykłady stosowania języka OQL do bazy UNIWERSYTET.

Dalej omówiliśmy wiązanie z językiem C++ (podrozdział 12.6), co polega na rozszerzeniu deklaracji klas C++ o konstruktory typów języka ODL, a przy tym umożliwia płynną integrację kodu w C++ z obiektowymi SZBD.

W 1997 r. firma Sun zatwierdziła interfejs API ODMG. Pierwszą firmą, która udostępniła SZBD zgodny z ODMG, była O2 Technologies. Standard ODMG został przyjęty przez wielu producentów obiektowych SZBD, w tym przez Object Design (eXcelon), Gemstone Systems, POET Software i Versant Corporation<sup>45</sup>.

## Pytania powtórkowe

- 12.1. Jakie jest pochodzenie podejścia obiektowego w bazach danych?
- 12.2. Jakie podstawowe cechy powinien posiadać OID?
- 12.3. Omów różne konstruktory typów. Jak mogą być użyte do tworzenia złożonych obiektów?
- 12.4. Omów pojęcie enkapsulacji i wyjaśnij, jak jest zastosowane przy tworzeniu abstrakcyjnych typów danych.
- 12.5. Wyjaśnij, co w terminologii obiektowych baz danych oznaczają pojęcia: *metoda*, *sygnatura*, *komunikat*, *kolekcja*, *ekstensja*.
- 12.6. Jakie jest powiązanie w hierarchii typów pomiędzy typem i podtypem? Jakie więzy są narzucone na ekstensje w związku z hierarchią typów?
- 12.7. Jaka jest różnica pomiędzy obiektami trwałymi i ulotnymi? Jak jest obsługiwana trwałość obiektów w obiektowych systemach baz danych?
- 12.8. Czym różnią się dziedziczenie, wielodziedziczenie i dziedziczenie wybiórcze?
- 12.9. Omów pojęcie polimorfizmu i przeciążania operatorów.
- 12.10. Wyjaśnij, w jaki sposób w SQL 2008 realizowane są następujące mechanizmy: *identyfikatory obiektów*, *dziedziczenie typów*, *enkapsulacja operacji* i *złożone struktury obiektowe*.
- 12.11. W tradycyjnym modelu relacyjnym utworzenie tabeli powoduje zdefiniowanie zarówno jej typu (schematów lub atrybutów), jak i samej tabeli (ekstensji lub zbioru aktualnie przechowywanych krotek). Jak można rozdzielić te dwa aspekty w SQL 2008?
- 12.12. Opisz reguły dziedziczenia obowiązujące w SQL 2008.
- 12.13. Jakie są różnice, a jakie podobieństwa pomiędzy obiektami i literałami w obiektowym modelu ODMG?

---

<sup>45</sup> Firma Versant Object Technology obecnie jest częścią korporacji Actian.

- 12.14. Wymień podstawowe operacje następujących typów wbudowanych w obiektowy model ODMG: *Object*, *Collection*, *Iterator*, *Set*, *List*, *Bag*, *Array* i *Dictionary*.
- 12.15. Opisz wbudowane literały strukturalne z modelu ODMG oraz ich operacje.
- 12.16. Jakie są podobieństwa, a jakie różnice pomiędzy własnościami atrybutów i związków w (atomowej) klasie definiowanej przez użytkownika?
- 12.17. Jakie są podobieństwa, a jakie różnice pomiędzy dziedziczeniem za pomocą operatora *EXTENDS* i znaku dwukropka (:) w modelu obiektowym ODMG?
- 12.18. Omów, jak określana jest trwałość obiektów w modelu ODMG w wiązaniu C++.
- 12.19. Dlaczego pojęcia ekstensji i kluczy są istotne w aplikacjach bazodanowych?
- 12.20. Opisz następujące pojęcia OQL: *punkt wejścia bazy danych*, *wyrażenia ścieżkowe*, *zmienne iteratora*, *nazwane zapytania (perspektywy)*, *funkcje agregujące*, *grupowanie* i *kwantyfikatory*.
- 12.21. Co rozumiemy przez pojęcie ortogonalności OQL?
- 12.22. Opisz podstawowe własności wiązania C++ w standardzie ODMG.
- 12.23. Jakie są podstawowe różnice pomiędzy projektowaniem relacyjnej i obiektowej bazy danych?
- 12.24. Opisz poszczególne kroki algorytmu odwzorowującego bazę danych EER na obiektową bazę danych.

## Ćwiczenia

- 12.25. Przetwórz przykładowe *FIGURY* podane w punkcie 12.1.5 z notacji funkcji na notację podaną na rysunku 12.2, która rozróżnia atrybuty i operacje. Użyj słowa kluczowego *INHERIT* do zaznaczenia, że jedna klasa dziedziczy funkcje z innej.
- 12.26. Porównaj dziedziczenie w modelu EER (patrz rozdział 4.) i dziedziczenie w modelu obiektowym opisanym w podrozdziale 12.25.
- 12.27. Rozważ schemat *UNIWERSYTET* z rysunku 4.10. Pomyśl, jakie operacje są potrzebne dla typów (klas) w tym schemacie. Pomiń konstruktory i destruktory.
- 12.28. Rozważ schemat *FIRMA* z rysunku 3.2. Pomyśl, jakie operacje są potrzebne dla typów w tym schemacie. Pomiń konstruktory i destruktory.
- 12.29. Przygotuj projekt obiektowej aplikacji bazodanowej, która cię interesuje. Najpierw skonstruuj schemat EER odpowiadający tej aplikacji, a następnie stwórz odpowiadające mu klasy w ODL. Określ metody dla każdej z klas i opracuj zapytania OQL odpowiadające projektowanej aplikacji.
- 12.30. Rozważając bazę danych *LOTNISKO* opisaną w ćwiczeniu 4.21, określ operacje (metody), które uważasz za odpowiednie dla takiej aplikacji. Zadeklaruj odpowiednie klasy i metody w języku ODL.
- 12.31. Odwzoruj schemat *FIRMA* z rysunku 3.2 na klasy ODL. Określ odpowiednie metody dla każdej z klas.
- 12.32. Przepisz zapytania z ćwiczeń do rozdziałów 6. i 7. (związane z bazą danych *FIRMA*) w języku OQL.



## Wybrane publikacje

Pojęcia związane z obiektowymi bazami danych są mieszaniną pojęć pochodzących z obiektowych języków programowania, systemów baz danych i koncepcyjnych modeli danych. Obiektowe języki programowania są opisywane w wielu książkach, np. Stroustrupa (1997) w odniesieniu do C++, czy Goldberga i Robsona (1989) dla języka Smalltalk. Książki Cattella (1994) oraz Lausena i Vossena (1997) opisują pojęcia dotyczące obiektowych baz danych. Inne książki na temat modeli obiektowych obejmują szczegółowy opis eksperymentalnego obiektowego SZBD o nazwie ORION opracowanego w Microelectronic Computer Corporation, a także powiązanych zagadnień obiektowych (Kim i Lochovsky, 1989). W Bancelhon i in. (1992) opisano historię tworzenia obiektowego SZBD O2; praca ta obejmuje szczegółowe omówienie decyzji projektowych i implementacji języka. W Dogac i in. (1994) znajdziesz szczegółowe omówienie zagadnień z obszaru baz obiektowych autorstwa ekspertów z warsztatów NATO.

Istnieje obszerna bibliografia dotycząca obiektowych baz danych; w tym miejscu możemy zamieścić jedynie przykładowe pozycje. Wydanie *CACM* z października 1991 i grudniowe wydanie *IEEE Computer* z 1990 roku opisują pojęcia i systemy obiektowych baz danych. Dittrich (1986) oraz Zaniolo i in. (1986) badają podstawowe pojęcia obiektowych modeli danych. Wcześniejszą pracą o obiektowych bazach danych jest praca Barroodiego i DeWitta (1981). Su i in. (1988) przedstawia zastosowanie obiektowego modelu danych w systemach CAD/CAM. Gupta i Horowitz (1992) opisują obiektowe rozwiązania w systemach CAD, zarządzaniu siecią i innych obszarach. Mitschang (1989) rozszerza algebrę relacyjną o obiekty złożone. Języki zapytań i graficzne interfejsy użytkownika dla systemów obiektowych są opisywane przez Gyssensa i in. (1990), Kima (1989), Alashqura i in. (1989), Bertino i in. (1992), Agrawala i in. (1990) oraz Cruza (1992).

Ciekawym artykułem prezentującym zdanie zespołu ekspertów na temat wymaganych i opcjonalnych aspektów zarządzania bazami obiektowymi jest „Object-Oriented Manifesto” autorstwa Atkinson i in. (1990). Polimorfizm w bazach danych i obiektowych językach programowania jest omawiany przez Osborna (1989), Atkinsona i Bunemana (1987) oraz Danforth i Tomlinsona (1988). Tożsamość obiektu jest opisywana przez Abiteboul i Kanellakisa (1989). Obiektowe języki programowania zastosowane w bazach danych — przez Kenta (1991). Więzy obiektów są opisywane w pracy Decambre’a i in. (1991) oraz Elmasriego, Jamesa i Kouramajiana (1993). Uwierzytelnianie i bezpieczeństwo w obiektowych bazach danych jest badane przez Rabbittiego i in. (1991) oraz Bertina (1992).

Praca Cattella i in. (2000) opisuje standard ODMG 3.0, a prace tych samych autorów z 1993 i 1997 r. przedstawiają wcześniejsze wersje standardu. Bancelhon i Ferrari (1995) przedstawiają samouczek dotyczący ważnych aspektów standardu ODMG. Kilka książek przedstawia architekturę CORBA — na przykład praca autorstwa Bakera (1996).

System O2 został opisany przez Deux i in. (1991) oraz Bancelhona i in. (1992), gdzie znajdziesz listę odwołań do innych publikacji opisujących różne aspekty tego systemu. Model O2 został sformalizowany w pracy Veleza i in. (1989). System ObjectStore został opisany w pracy Lamba i in. (1991). Fishman i in. (1987) oraz Wilkinson i in. (1990) omawiają IRIS — obiektowy system baz danych opracowany w laboratoriach firmy Hewlett-Packard. Maier i in. (1986) oraz Butterworth i in. (1991) opisują projekt GEMSTONE. System ODE opracowany w AT&T Bell Labs został opisany przez Agrawala i Gehaniego (1989). System ORION opracowany w MCC został opisany przez Kima i in. (1990). Morsi i in. (1992) opisują środowisko testowe dla systemów obiektowych.



Catell (1991) analizuje zagadnienia dotyczące baz relacyjnych i obiektowych oraz omawia kilka prototypów obiektowych i rozszerzonych relacyjnych systemów baz danych. Alagic (1997) wskazuje na rozbieżności między modelem danych ODMG a wiązaniami języków i podsuwa kilka rozwiązań tego problemu. Bertino i Guerrini (1998) proponują rozszerzenie modelu ODMG o obsługę obiektów złożonych. Alagic (1999) prezentuje kilka modeli danych z rodziny ODMG.

## XML — rozszerzalny język znaczników

Wiele aplikacji internetowych oferuje dostęp do zasobów swoich baz danych poprzez interfejs WWW. Będziemy się odnosić do tych baz danych jako do **źródeł danych**. W aplikacjach internetowych często wykorzystuje się trójwarstwowe architektury klient-serwer (patrz podrozdział 2.5). Internetowe aplikacje bazodanowe są zaprojektowane z myślą o komunikowaniu się z użytkownikiem za pośrednictwem stron WWW wyświetlanych na komputerach stacjonarnych, laptopach i urządzeniach mobilnych. Popularną metodą określania zawartości i formatowania stron WWW jest stosowanie **dokumentów hipertekstowych**. Istnieją różne języki tworzenia tego typu dokumentów, a najpopularniejszym z nich jest *HTML* (ang. *Hypertext Markup Language*). Co prawda HTML jest często stosowany do formatowania dokumentów WWW, ale nie nadaje się do wyrażania pobieranych z bazy danych posiadających *wewnętrzną strukturę*. Niedawno opracowano nowy język — *XML* (ang. *Extensible Markup Language*) — jako standard strukturyzowania i wymiany danych w sieci WWW za pomocą plików tekstowych. Za pomocą XML można wyrazić raczej strukturę i znaczenie danych, a nie sposób ich prezentacji na ekranie. Dokumenty w formatach XML i JSON to pliki tekstowe, które udostępniają informacje opisowe, np. nazwy atrybutów, a także wartości tych atrybutów. Kwestie formatowania są określane oddzielnie, na przykład za pomocą języka formatującego *XSL* (ang. *Extensible Stylesheet Language*) lub języka przekształceń *XSLT* (ang. *Extensible Stylesheet Language for Transformations* lub *XSL Transformations*). Niedawno XML został zaproponowany jako model składowania i pobierania danych, choć do tej pory opracowano niewiele eksperymentalnych systemów baz danych opartych na tym języku.

Podstawowy HTML jest przydatny do generowania *statycznych* stron WWW ze stałym tekstem i innymi obiektami. Jednak większość sklepów elektronicznych wymaga stron WWW z interaktywnymi funkcjami. Strony te wykorzystują przekazane przez użytkownika informacje do wyboru z bazy określonych danych do wyświetlenia. Takie strony WWW są nazywane *dynamicznymi*, ponieważ pobierane i wyświetlane dane są różne w zależności od danych wejściowych od użytkownika. Przykładowo, aplikacja banku może pobierać numer konta użytkownika, a następnie wczytywać stan konta tej osoby z bazy, aby go wyświetlić. Wyjaśniliśmy już, jak posługiwać się językami skryptowymi (takimi jak PHP) do generowania dynamicznych stron WWW takich jak prezentowane w rozdziale 11. XML może służyć do przekazywania w samoopisowych plikach tekstowych informacji między różnymi programami z różnych komputerów, gdy dane są potrzebne aplikacjom.

W tym rozdziale skoncentrujemy się na opisie modelu danych XML i powiązanych języków, a także na tym, jak dane pobrane z baz relacyjnych formatować do postaci dokumentów XML przekazywanych w internecie. W podrozdziale 13.1 omówione zostały

różnice pomiędzy danymi strukturalnymi, półstrukturalnymi i niestukturalnymi. W podrozdziale 13.2 przedstawiony jest model danych XML, oparty na strukturach drzewiastych (hierarchicznych), a nie na płaskich strukturach z modelu relacyjnego. Podrozdział 13.3 omawia dokumenty XML i języki określania ich struktury, a konkretnie XML DTD (ang. *Document Type Definition*) i *XML Schema*. Podrozdział 13.4 przedstawia związki między standardem XML a bazami relacyjnymi. W podrozdziale 13.5 opiszemy wybrane języki powiązane ze standardem XML, np. XPath i XQuery. W podrozdziale 13.6 wyjaśnimy, jak dane pobrane z baz relacyjnych formatować do postaci dokumentów XML. W podrozdziale 13.7 omówimy nowe funkcje dodane do standardu XML w celu generowania dokumentów XML na podstawie baz relacyjnych. Podrozdział 13.8 to podsumowanie rozdziału.

## 13.1. Dane strukturalne, półstrukturalne i niestukturalne

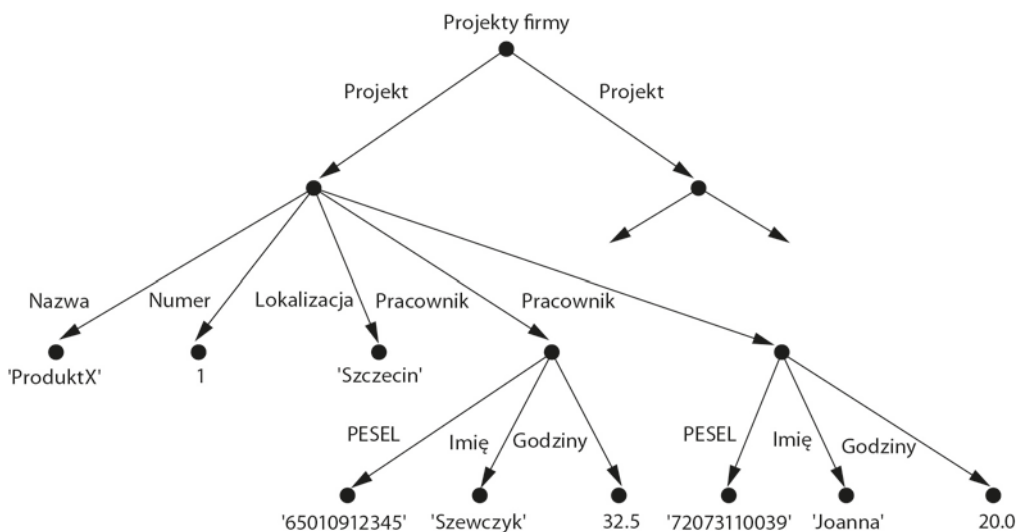
Informacje zapisane w bazach danych są określane mianem **danych strukturalnych**, ponieważ są przedstawione w ściśle określonym formacie. Każdy rekord w tabeli relacyjnej bazy danych (takiej jak tabele z bazy FIRMA z rysunku 5.6) ma ten sam format co pozostałe rekordy tej tabeli. W przypadku danych strukturalnych istotne jest staranne zaprojektowanie schematu bazy danych przy użyciu technik opisanych w rozdziałach 3. i 4., aby zdefiniować strukturę bazy. System bazy danych sprawdza później, czy dane odpowiadają opisanej strukturze i ograniczeniom zawartym w schemacie.

Nie zawsze jednak dane są przechowywane w bazach danych o precyzyjnie określonej strukturze. W niektórych przypadkach dane są gromadzone na bieżąco zanim jeszcze wiadomo, w jaki sposób będą przechowywane i zarządzane. Takie dane mogą mieć pewną określoną strukturę, ale nie zawsze jednakową. Niektóre atrybuty mogą być współdzielone przez kilka obiektów, a inne istnieć tylko dla niektórych. Co więcej, w przyszłości może zaistnieć potrzeba zdefiniowania dodatkowych atrybutów dla nowych obiektów o nieznaney z góry strukturze. Ten typ danych nazywamy **półstrukturalnymi**. Istnieje wiele modeli danych pracujących z danymi półstrukturalnymi. Modele takie często, zamiast wykorzystywać płaskie struktury modelu relacyjnego, opierają się na strukturze drzew lub grafów.

Podstawowa różnica pomiędzy danymi strukturalnymi a półstrukturalnymi leży w sposobie traktowania konstrukcji schematu (nazw atrybutów, relacji i typów encji). W przypadku danych półstrukturalnych struktura schematu jest *wymieszana* z samymi danymi, ponieważ każdy obiekt może mieć inne nieznanne wcześniej atrybuty. Taki rodzaj danych nazywany jest **danymi samoopisowymi**. W wielu nowszych systemach NOSQL stosowane są samoopisowe schematy składowania danych (patrz rozdział 24.). Rozważmy następujący przykład: chcemy zgromadzić listę odniesień bibliograficznych związanych z określonym projektem badawczym. Niektóre elementy tej listy mogą być książkami lub raportami technicznymi, inne artykułami z prasy specjalistycznej lub materiałów konferencyjnych, lub wręcz całymi wydaniem fachowych czasopism. Oczywiście, każdy z nich będzie zawierał inne atrybuty i inne informacje. Nawet elementy tego samego typu (np. artykuły konferencyjne) mogą zawierać różne typy informacji — jeden artykuł może być w pełni opisany wraz z nazwiskami autorów, tytułem, datą, numerem strony

itd., podczas gdy informacje o innym mogą być uboższe. W przyszłości mogą zaistnieć nowe rodzaje odwołań bibliograficznych (np. odwołania do stron WWW lub samouczki z konferencji), które będą wymagać opisanie za pomocą nowych atrybutów.

Dane półstrukturalne mogą być przedstawione w postaci grafu skierowanego takiego, jak pokazany na rysunku 13.1. Informacje przedstawione na tym rysunku nawiązują do niektórych elementów struktury z rysunku 5.6. Jak widać, model ten nieco przypomina model obiektowy (patrz punkt 12.1.3) w zakresie możliwości reprezentowania złożonych obiektów i ich zagnieżdżonych struktur. Na rysunku 13.1 **etykiety** lub **znaczniki** (ang. *tags*) przypisane skierowanym krawędziom grafu przedstawiają nazwy schematu: *nazwy atrybutów*, *typy obiektów* (*typy encji* lub *klasy*) oraz *związki*. Węzły wewnętrzne reprezentują poszczególne obiekty lub atrybuty złożone. Liście drzewa reprezentują wartości prostych (atomowych) atrybutów.



RYSUNEK 13.1. Reprezentowanie danych półstrukturalnych za pomocą grafu

Istnieją dwie główne różnice pomiędzy modelem półstrukturalnym a obiektowym (omawianym w rozdziale 12.):

- (1) Informacje o schemacie — nazwy atrybutów, związków i klas (typów obiektów) w modelu półstrukturalnym są wymieszane z obiektami i zapisanymi w nich danymi w jednej strukturze.
- (2) W modelu półstrukturalnym nie ma wymogu stworzenia predefiniowanego schematu, do którego muszą stosować się wszystkie obiekty, choć w razie potrzeby możliwe jest zdefiniowanie schematu. Opisany w rozdziale 12. model obiektowy wymaga schematu.

Poza danymi strukturalnymi i półstrukturalnymi istnieje jeszcze trzecia kategoria — **danych niestrukturalnych**, posiadających bardzo nikłe ograniczenia dotyczące rodzaju informacji. Typowym przykładem jest dokument tekstowy zawierający wewnątrz informacje. Napisane w języku HTML strony WWW zawierające dane są również traktowane

jako dane niestrukuralne. Rozważmy część pliku HTML pokazaną na rysunku 13.2. Napisy w nawiasach ostrych `<...>` są **znacznikami HTML**. Znacznik z prawym ukośnikiem `</...>` jest **znacznikiem końcowym**, który wskazuje koniec działania odpowiedniego **znacznika początkowego**. Znaczniki **opisują** dokument<sup>1</sup> i służą do wskazywania procesorowi HTML, jak ma być wyświetlany tekst pomiędzy znacznikiem początkowym a końcowym. Znaczniki określają zatem raczej sposób formatowania dokumentu niż znaczenie różnych elementów danych. Mogą zawierać takie informacje jak rozmiar czcionki i jej odmianę (kursywa, wytłuszczona itp.), kolor, nagłówki różnych poziomów. Niektóre znaczniki pozwalają na wprowadzenie struktury tekstu, na przykład tabeli, albo listy numerowanej bądź nienumerowanej. Nawet jednak te strukturalne znaczniki wskazują raczej sposób prezentacji danych niż ich znaczenie.

```
<html>
<head>
...
</head>
<body>
<H1>Lista firmowych projektów i przypisanych im pracowników</H1>
<H2>Projekt ProduktX:</H2>
  <table width="100%" border=0 cellpadding=0 cellspacing=0>
    <TR>
      <TD width="50%"><font size="2" face="Arial">Jan Szewczyk:</font></TD>
      <TD>32,5 godziny tygodniowo</TD>
    </TR>
    <TR>
      <TD width="50%"><font size="2" face="Arial">Joanna Englert:
        ↳</font></TD>
      <TD>20 godzin tygodniowo</TD>
    </TR>
  </table>

  <H2>Projekt ProduktY:</H2>
  <table width="100%" border=0 cellpadding=0 cellspacing=0>
    <TR>
      <TD width="50%"><font size="2" face="Arial">Jan Szewczyk:</font></TD>
      <TD>7,5 godziny tygodniowo</TD>
    </TR>
    <TR>
      <TD width="50%"><font size="2" face="Arial">Joanna Englert:
        ↳</font></TD>
      <TD>20 godzin tygodniowo</TD>
    </TR>
    <TR>
      <TD width="50%"><font size="2" face="Arial">Franciszek Wieszczycki:
        ↳</font></TD>
      <TD>10 godzin tygodniowo</TD>
    </TR>
  </table>
```

---

<sup>1</sup> Od oznaczania tekstu pochodzi angielska nazwa tego typu dokumentów — *Hypertext Markup Language*, czyli hipertekstowy język znaczników.

```
        </TR>
    </table>

    ...
</body>
</html>
```

RYSUNEK 13.2. Fragment dokumentu HTML zawierającego dane niestrukuralne

HTML używa wielu predefiniowanych znaczników opisujących różne polecenia dotyczące formatowania stron WWW przy ich wyświetlaniu. Znaczniki początkowe i końcowe wskazują zakres tekstu objęty danym formatowaniem. Kilka przykładów znaczników z rysunku 13.2 jest przedstawionych poniżej:

- Znaczniki `<html>...</html>` określają początek i koniec dokumentu.
- **Nagłówek dokumentu**, zawarty pomiędzy znacznikami `<head>...</head>`, zawiera różne polecenia, które mogą być użyte w innych miejscach dokumentu. Na przykład może definiować różne **funkcje skryptów** w językach takich jak JavaScript lub PERL oraz różne **formatowania styli** (znaków, akapitów, nagłówków i innych), które mogą być wykorzystane w dokumencie. Można też podać tytuł (aby określić, do czego służy dany plik HTML) i inne podobne informacje, które nie będą wyświetlane jako część dokumentu.
- **Ciało** dokumentu jest określone pomiędzy znacznikami `<body>...</body>` i zawiera tekst dokumentu oraz znaczniki definiujące sposób jego wyświetlania. Zawiera również odwołania do innych zewnętrznych obiektów, takich jak obrazki, animacje, dźwięki i inne dokumenty.
- Znaczniki `<H1>...</H1>` określają tekst, który ma być wyświetlany jako nagłówek pierwszego stopnia. Istnieją również inne znaczniki (`<H2>`, `<H3>` itd.) określające kolejne, mniej istotne nagłówki.
- Znacznik `<table>...</table>` wskazuje, że tekst ma być wyświetlony w tabeli. Każdy *wiersz tabeli* jest zawarty w znacznikach `<TR>...</TR>`, a poszczególne elementy danych z tabeli — w znacznikach `<TD>...</TD>`<sup>2</sup>.
- Niektóre znaczniki mogą mieć **atrybuty**, które są widoczne wewnątrz znacznika początkowego i opisują dodatkowe właściwości znacznika<sup>3</sup>.

Na rysunku 13.2 znacznik początkowy `<table>` ma cztery atrybuty określające szczegółowe cechy tabeli, a dalsze znaczniki `<TD>` i `<font>` mają odpowiednio jeden i dwa atrybuty.

Zastosowaniu wszystkich predefiniowanych znaczników HTML poświęcone są całe książki. Przy odpowiednim wykonaniu, dokumenty HTML mogą być sformatowane w sposób zrozumiały dla użytkownika i pozwalający na łatwe poruszanie się po dokumentach internetowych. Źródłowe dokumenty tekstowe HTML są jednak trudne do automatycznej interpretacji przez *programy komputerowe*, ponieważ nie zawierają informacji o schemacie użytym przy tworzeniu dokumentu. W miarę rozwoju aplikacji do handlu elektronicznego i innych rozwiązań internetowych możliwość wymiany dokumentów i ich automatycznej interpretacji staje się coraz istotniejszym problemem. Był on jedną z przyczyn

<sup>2</sup> `<TR>` oznacza „table row”, czyli wiersz tabeli, a `<TD>` dane tabeli.

<sup>3</sup> Należy zwrócić uwagę, że w przypadku rodziny języków znacznikowych znaczenie terminu *atrybut* jest odmienne od stosowanego w bazach danych.

powstania standardu XML, który jest przedstawiony w następnym podrozdziale. Ponadto opracowano rozszerzalną wersję języka HTML, XHTML, umożliwiającą użytkownikom dodawanie znaczników HTML na potrzeby różnych aplikacji i interpretowanie plików XHTML przez standardowe programy przetwarzające dane w standardzie XML. Tu koncentrujemy się tylko na standardzie XML.

Przykład z rysunku 13.2 ilustruje **statyczną** stronę HTML, ponieważ wszystkie wyświetlane informacje są bezpośrednio zapisane jako stały tekst w pliku HTML. W wielu sytuacjach część wyświetlanych informacji jest pobierana z bazy. Przykładowo, nazwy projektów i uczestniczących w nich pracowników mogą zostać pobrane z bazy z rysunku 5.6 za pomocą odpowiedniego zapytania SQL. Możesz chcieć wykorzystać te same znaczniki formatujące języka HTML do wyświetlania wszystkich projektów i powiązanych z nimi pracowników, ale zmieniać pokazywane projekty (i osoby). Możliwe, że chcesz zobaczyć stronę z informacjami o projekcie *ProjektX*, a następnie stronę z danymi na temat projektu *ProjektY*. Choć obie strony są wyświetlane za pomocą tych samych znaczników formatujących języka HTML, pokazywane dane są inne. Takie strony są nazywane **dynamicznymi**, ponieważ fragmenty danych za każdym razem mogą być inne, choć wygląd strony się nie zmienia. W rozdziale 11. napisano, że do generowania dynamicznych stron WWW można wykorzystać języki skryptowe (np. PHP).

## 13.2. Hierarchiczny (drzewiasty) model danych w dokumentach XML

W niniejszym podrozdziale zostanie wprowadzony model danych stosowany w dokumentach XML. Podstawowym obiektem w standardzie XML jest dokument XML. W jego konstrukcji stosowane są dwa podstawowe pojęcia: **elementu** i **atrybutu**. Należy od razu zaznaczyć, że termin *atrybut* ma tu znaczenie charakterystyczne dla języków opisu dokumentów (takich jak HTML lub SGML<sup>4</sup>), które jest *inne niż w przypadku baz danych*. Jak pokażemy w dalszej części, atrybuty w XML dostarczają dodatkowe informacje opisujące elementy. W XML stosowane są też inne pojęcia, takie jak encje, identyfikatory, odwołania, ale na początek zaprezentowane zostanie zastosowanie elementów i atrybutów będących istotą tego modelu.

Rysunek 13.3 pokazuje przykładowy dokument XML nazwany `<projekty>`. Podobnie jak w HTML elementy są definiowane za pomocą znacznika początkowego i końcowego. Nazwy znaczników są zawarte pomiędzy ostrymi nawiasami `<...>`, a znaczniki końcowe są zaznaczane prawym ukośnikiem — `</...>`<sup>5</sup>.

---

<sup>4</sup> SGML (ang. *Standard Generalized Markup Language*) jest bardziej ogólnym językiem opisującym dokumenty i pozwalającym na definiowanie przez użytkownika własnych znaczników. Jest on jednak znacznie bardziej skomplikowany niż HTML lub XML.

<sup>5</sup> Znaki lewego i prawego ostrego nawiasu (`<` i `>`) są znakami zastrzeżonymi, podobnie jak ampersand (&), apostrof (') i pojedynczy cudzysłów ('). Aby ich użyć w tekście dokumentu, należy stosować ich zakodowane odpowiedniki: `&lt;`, `&gt;`, `&amp;`, `&apos;` i `&quot;`.



```
<?xml version="1.0" standalone="yes"?>
<projekty>
  <projekt>
    <Nazwa>Produkt X</Nazwa>
    <Numer>1</Numer>
    <Lokalizacja>Kielce</Lokalizacja>
    <NrDziału>5</NrDziału>
    <Pracownik>
      <PESEL>65010912345</PESEL>
      <Nazwisko>Nowak</Nazwisko>
      <godziny>32,5</godziny>
    </Pracownik>
    <Pracownik>
      <PESEL>72073110039</PESEL>
      <Imię>Maciej</Imię>
      <godziny>20,0</godziny>
    </Pracownik>
  </projekt>
  <projekt>
    <Nazwa>Produkt Y</Nazwa>
    <Numer>2</Numer>
    <Lokalizacja>Kraków</Lokalizacja>
    <NrDziału>5</NrDziału>
    <Pracownik>
      <PESEL>65010912345</PESEL>
      <godziny>7,5</godziny>
    </Pracownik>
    <Pracownik>
      <PESEL>72073110039</PESEL>
      <godziny>20,0</godziny>
    </Pracownik>
    <Pracownik>
      <PESEL>55120834598</PESEL>
      <godziny>10,0</godziny>
    </Pracownik>
  </projekt>
</projekty>
```

RYSUNEK 13.3. Złożony element XML &lt;projekty&gt;

**Złożone elementy** są tworzone hierarchicznie z innych elementów, a **elementy proste** zawierają określone wartości. Główną różnicą pomiędzy HTML a XML jest to, że w XML nazwy znaczników służą do opisywania znaczenia elementów w dokumencie, a nie do definiowania sposobu ich wyświetlania. Pozwala to na automatyczne przetwarzanie danych w dokumencie XML przez programy komputerowe. Ponadto nazwy znaczników (elementów) standardu XML można zdefiniować w innym dokumencie, *schemacie*, aby określić semantyczne znaczenie nazw znaczników, które można przekazywać między programami i użytkownikami. W języku HTML wszystkie nazwy znaczników są zdefiniowane i niezmiennicze, dlatego HTML nie jest rozszerzalny.

Podobieństwo pomiędzy tekstową reprezentacją XML z rysunku 13.3 a drzewem z rysunku 13.1 jest oczywiste. W reprezentacji za pomocą drzewa wewnętrzne węzły odpowiadają elementom złożonym, a liście odpowiadają elementom prostym. Dlatego właśnie model XML jest nazywany **hierarchicznym** lub **drzewiastym**. Na rysunku 13.3 prostymi elementami są <Nazwa>, <Numer>, <Lokalizacja>, <NrDziału>, <PESEL>, <Nazwisko>, <Imię> i <godziny>. Złożone elementy to <projekty>, <projekt> i <Pracownik>. Ogólnie, nie istnieją ograniczenia dotyczące głębokości zagnieżdżenia elementów.

Dokumenty XML można podzielić na trzy rodzaje:

- **Dokumenty danocentryczne** (ang. *data-centric*) — są to dokumenty mające wiele małych elementów danych odpowiadających pewnej strukturze, które mogą być odczytywane ze strukturalnej bazy danych. Dane te są przedstawione w formie dokumentu XML w celu ich wymiany w internecie. Takie dane mają zwykle *zdefiniowany schemat* określający nazwy znaczników.
- **Dokumenty dokumentocentryczne** (ang. *document-centric*) — są to pliki z dużą ilością tekstu, takie jak artykuły lub książki. Zawierają bardzo mało elementów strukturalnych.
- **Dokumenty hybrydowe** — tego typu dokumenty zawierają zarówno fragmenty danych o ścisłej strukturze, jak i fragmenty tekstu lub danych niestukturalnych. Mogą, ale nie muszą mieć zdefiniowanego schematu.

Dokumenty XML, które nie mają zdefiniowanego schematu nazw elementów i powiązanej z tym struktury drzewiastej, to **dokumenty XML bez schematu**. Należy zauważyć, że dokumenty danocentryczne mogą być postrzegane zarówno jako dane strukturalne, jak i jako dane półstrukturalne (patrz podrozdział 13.1). Jeśli dokument XML spełnia warunki określone wcześniej w schemacie XML lub w DTD (patrz podrozdział 13.3), to dokument taki zawiera *dane strukturalne*. Z drugiej strony, w standardzie XML można tworzyć dokumenty nie spełniające żadnych narzuconych warunków. W takim przypadku dokument XML będzie zawierał *dane półstrukturalne* i będzie nazywany *dokumentem XML bez schematu*. Jeśli atrybut `STANDALONE` dokumentu XML jest równy "yes" (tak jak w pierwszym wierszu rysunku 13.3), to dokument jest samodzielny i nie posiada schematu.

Atrybuty w XML są zazwyczaj używane w sposób podobny jak w przypadku HTML (patrz rysunek 13.2), czyli do określania dodatkowych właściwości i cech elementów (znaczników), w których się pojawiają. Można w nich również zapisywać wartości elementów prostych, jednak takie zastosowanie nie jest zalecane. Wyjątkiem są sytuacje, gdy potrzebna jest **referencja** do innego elementu z innej części dokumentu XML. Wtedy często stosuje się wartości atrybutów w jednym elemencie jako referencje. Przypomina to klucze obce z baz relacyjnych i jest doskonałym sposobem na poradzenie sobie ze ściśle hierarchicznym modelem wynikającym z modelu drzewiastego ze standardu XML. Atrybuty XML są omówione w podrozdziale 13.3 wraz z DTD i schematami XML.

## 13.3. Dokumenty XML, DTD i schematy

### 13.3.1. Dobrze uformowane i prawidłowe dokumenty XML oraz XML DTD

Na rysunku 13.3 pokazano, jak może wyglądać przykładowy prosty dokument XML. Dokument XML jest **dobrze uformowany** (ang. *well-formed*) jeżeli spełnia kilka warunków. Po pierwsze, musi zaczynać się od **deklaracji XML** zawierającej informację o użytej wersji standardu oraz o innych szczegółach dotyczących dokumentu, widocznych w pierwszym wierszu rysunku 13.3. Musi również spełniać założenia składniowe modelu drzewa. Oznacza to, że musi istnieć *dokładnie jeden element główny*, a każdy element musi się składać z pary odpowiadających sobie znaczników zawartych pomiędzy znacznikiem początkowym i końcowym *elementu nadrzędnego*. Te ograniczenia zapewniają prawidłowe uformowanie struktury drzewa.

Dobrze uformowany dokument XML jest poprawny syntaktycznie. Oznacza to, że może on być przetwarzany przez ogólne procesory tworzące na podstawie dokumentu XML jego wewnętrzną reprezentację drzewiastą. Manipulowanie taką reprezentacją jest możliwe za pomocą standardowego zestawu funkcji *API* (ang. *application programming interface* — programistycznego interfejsu aplikacji) nazywanego *DOM* (ang. *document object model* — obiektowy model dokumentu). Aby go jednak używać, należy najpierw przetworzyć cały dokument. Inny interfejs — *SAX* — umożliwia przetwarzanie dokumentu podczas jego odczytywania, powiadamiając swój program za każdym razem, gdy natrafia na znacznik początkowy lub końcowy. Ułatwia to pracę z dużymi dokumentami i pozwala na przetwarzanie **strumieniowych dokumentów XML**, które muszą być przetwarzane podczas odczytywania. Ten proces to **przetwarzanie oparte na zdarzeniach**. Istnieją też inne wyspecjalizowane procesory, współdziałające z różnymi językami programowania i językami skryptowymi w zakresie przetwarzania dokumentów XML.

W dobrze uformowanym dokumencie XML schemat nie jest konieczny; można używać jako nazw elementów dowolnych ciągów znaków. W takiej sytuacji nie istnieje żaden z góry określony zestaw znaczników, których program przetwarzający dokument może oczekiwać. Daje to twórcy dokumentu swobodę tworzenia nowych elementów, ale ogranicza możliwości automatycznego interpretowania znaczenia (semantyki) elementów dokumentu.

Bardziej restrykcyjne są warunki, które muszą spełniać **prawidłowe** dokumenty XML (ang. *valid XML documents*). W tym przypadku dokument musi być dobrze uformowany, a ponadto nazwy elementów używane w znacznikach muszą odpowiadać strukturze zdefiniowanej w osobnym dokumencie XML **DTD** (ang. *document type definition* — definicji typu dokumentu) lub **pliku schematu XML**. W tym punkcie omówione zostanie XML DTD, a schematy XML zostaną przedstawione w punkcie 13.3.2. Rysunek 13.4 przedstawia prosty plik XML DTD określający elementy (nazwy znaczników) i strukturę ich zagnieżdżenia. Każdy prawidłowy dokument XML odpowiadający temu DTD będzie miał tę samą, z góry określoną strukturę. Do określania dokumentów XML DTD stosowana jest specjalna składnia (patrz rysunek 13.4(a)). Na początku podana jest nazwa **elementu głównego** dokumentu, który w przykładzie z rysunku 13.4 nazywa się *projekt*. Następnie zdefiniowane są dalsze elementy i możliwości ich zagnieżdżenia.

```

(a) <!DOCTYPE Projekty [
  <!ELEMENT Projekty (Projekt+)>
  <!ELEMENT Projekt (Nazwa, Numer, Lokalizacja, NrDziału?, Pracownik)>
    <!ATTLIST Projekt
      IdProj ID #REQUIRED>
  <!ELEMENT Nazwa (#PCDATA)>
  <!ELEMENT Numer (#PCDATA)>
  <!ELEMENT Lokalizacja (#PCDATA)>
  <!ELEMENT NrDziału (#PCDATA)>
  <!ELEMENT Pracownicy (Pracownik*)>
  <!ELEMENT Pracownik (PESEL, Nazwisko?, Imię?, godziny)>
  <!ELEMENT PESEL (#PCDATA)>
  <!ELEMENT Nazwisko (#PCDATA)>
  <!ELEMENT Imię (#PCDATA)>
  <!ELEMENT godziny (#PCDATA)>
]>

(b) <!DOCTYPE Firma [
  <!ELEMENT Firma( (Pracownik|Dział|Projekt)*)>
  <!ELEMENT Dział (NazwaDziału, Lokalizacja+)>
    <!ATTLIST Dział
      IdDziału ID #REQUIRED>

  <!ELEMENT Pracownik (NazwiskoPrac, Stanowisko, Pensja)>
    <!ATTLIST Projekt
      IdPrac ID #REQUIRED
      IdDziału IDREF #REQUIRED>
  <!ELEMENT Projekt (NazwaProjektu, Lokalizacja)>
    <!ATTLIST Projekt
      IdProjektu ID #REQUIRED
      Pracownicy IDREFS #IMPLIED>
  <!ELEMENT NazwaDziału (#PCDATA)>
  <!ELEMENT NazwiskoPrac (#PCDATA)>
  <!ELEMENT NazwaProjektu (#PCDATA)>
  <!ELEMENT Stanowisko (#PCDATA)>
  <!ELEMENT Lokalizacja (#PCDATA)>
  <!ELEMENT Pensja (#PCDATA)>
] >

```

RYSunek 13.4. (a) Plik XML DTD Projekty. (b) Plik XML DTD Firma

Przy definiowaniu elementów stosuje się następującą notację:

- Znak \* po nazwie elementu oznacza, że ten element może być pominięty lub powtórzony dowolną liczbę razy. Ten rodzaj elementu jest nazywany *opcjonalnym elementem wielowartościowym*.
- Znak + po nazwie elementu oznacza, że ten element może być w dokumencie powtórzony jeden lub kilka razy. Ten rodzaj elementu jest nazywany *wymaganym elementem wielowartościowym*.

- Znak ? po nazwie elementu oznacza, że ten element może być pominięty albo użyty jeden raz. Ten rodzaj elementu jest nazywany *opcjonalnym elementem jednowartościowym*.
- Nazwa elementu bez żadnego z powyższych trzech znaków oznacza, że element ma być użyty dokładnie jeden raz. Ten typ elementu jest nazywany *wymaganym elementem jednowartościowym*.
- **Typ** elementu jest określony w nawiasach bezpośrednio po nazwie elementu. Jeżeli w nawiasach zawarta jest nazwa innych elementów, to są one *dziećmi* (elementami podrzędnymi) tego elementu w strukturze drzewa. Jeśli w nawiasach występuje słowo kluczowe #PCDATA lub inny typ predefiniowany w DTD, to element ten jest liściem drzewa. Wyrażenie PCDATA oznacza przetworzone dane znakowe (ang. *parsed character data*), co odpowiada mniej więcej ciągowi znaków.
- Za pomocą słowa kluczowego !ATTLIST można określić listę atrybutów, jakie mogą występować w elemencie. Na rysunku 13.3 element Projekt ma atrybut IdProjektu. Jeśli typem atrybutu jest ID, można zastosować referencję do niego w atrybucie typu IDREF w innym elemencie. Zauważ, że atrybuty można też stosować do przechowywania wartości prostych elementów danych typu #PCDATA.
- Nawiasy w definicji elementów mogą być zagnieżdżone.
- Symbol pionowej kreski ( $e_1 \mid e_2$ ) oznacza, że w dokumencie może być użyte  $e_1$  albo  $e_2$ .

Jak widać, struktura drzewa z rysunku 13.1 i dokument XML z rysunku 13.3 odpowiadają definicji typu dokumentu z rysunku 13.4. Aby zaznaczyć potrzebę sprawdzenia zgodności dokumentu z DTD, musimy zamieścić odpowiednią deklarację w samym dokumencie. Początek rysunku 13.3 należałoby zmienić na:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE projekty SYSTEM "proj.dtd">
```

Zmiana atrybutu standalone na "no" oznacza, że dokument powinien być sprawdzony pod kątem zgodności z zewnętrznym plikiem DTD lub dokumentem ze schematem XML (patrz podrozdział 13.2.2). Definicja typu dokumentu z rysunku 13.4 powinna być zapisana w pliku *proj.dtd* w tym samym systemie plików co sam dokument. Można również dołączyć DTD do początku dokumentu XML.

Na rysunku 13.4(b) pokazany jest inny dokument DTD, Firma, ilustrujący zastosowanie typu IDREF. W tym dokumencie można umieścić dowolną liczbę elementów Dział, Pracownik i Projekt z identyfikatorami działu, pracownika i projektu. Element Pracownik ma atrybut NrDziału typu IDREF, będący referencją do elementu Dział, w którym dana osoba pracuje. Taki atrybut działu podobnie jak klucz obcy. Element Projekt ma atrybut Pracownicy typu IDREFS; ten atrybut przechowuje listę identyfikatorów osób pracujących nad danym projektem. Ten typ działu podobnie jak kolekcja lub lista kluczy obcych. Słowo kluczowe #IMPLIED oznacza, że atrybut jest opcjonalny. Można też podać wartość domyślną każdego atrybutu.

Co prawda XML DTD jest odpowiednim narzędziem do definiowania struktury drzewa z opcjonalnymi, wymaganymi i powtarzаныmi elementami, ale nie jest wolne od ograniczeń. Po pierwsze, typy danych w DTD są mało ogólne. Po drugie, XML DTD ma swoją własną składnię, która wymaga specjalnych procesorów. Określanie schematu doku-

mentów XML za pomocą innych dokumentów XML byłoby niewątpliwą zaletą i pozwalałoby na stosowanie tych samych aplikacji do przetwarzania zarówno opisów schematów, jak i samych dokumentów. Po trzecie, wszystkie elementy DTD trzeba podawać w dokumencie zgodnie z określoną kolejnością, dlatego stosowanie elementów nieuporządkowanych jest niedozwolone. Te wady doprowadziły do opracowania schematów XML (*XML schema*) — bardziej ogólnego, ale też bardziej skomplikowanego języka definiowania elementów i struktury dokumentów XML.

### 13.3.2. Schematy XML

**Język schematów XML** jest standardem opisującym strukturę dokumentów XML. Używa on tej samej składni co zwyczajne dokumenty XML, więc może być przetwarzany za pomocą tych samych programów co dokumenty XML. Aby rozróżnić oby typy dokumentów, będziemy nazywali zwykłe dokumenty XML (obejmujące zarówno nazwy znaczników, jak i wartości danych) *dokumentami instancji XML* lub po prostu *dokumentami XML*, a dokumenty opisujące schematy XML — *dokumentami schematu XML*. Dokument schematu XML może obejmować tylko nazwy znaczników, informacje o strukturze drzewa, ograniczenia i inne opisy, ale nie zawiera wartości danych. Rysunek 13.5 przedstawia dokument schematu XML odpowiadający bazie danych FIRMA pokazanej na rysunku 5.5. Co prawda zapisywanie całej bazy danych jako pojedynczego dokumentu XML jest mało prawdopodobne, ale pojawiły się propozycje zapisywania danych w *natywnym* formacie XML zamiast w relacyjnych bazach danych. Schemat z rysunku 13.5 mógłby posłużyć do określenia struktury bazy danych FIRMA, jeśli miałaby ona być zapisana w całości jako dokument XML. Temat ten będzie dalej rozważany w punkcie 13.4.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
    <xsd:documentation xml:lang="pl">Schemat firma (podejście elementowe)
    ↪ - opracowany przez Marka Nowakowskiego</xsd:documentation>
  </xsd:annotation>
  <xsd:element name="firma">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="dział" type="Dział" minOccurs="0"
        ↪ maxOccurs="unbounded"
        ↪ />
        <xsd:element name="pracownik" type="Pracownik" minOccurs="0"
        ↪ maxOccurs="unbounded">
          <xsd:unique name="imięCzłonkaRodzinyUnikatowe">
            <xsd:selector xpath="członekRodzinyPracownika" />
            <xsd:field xpath="imięCzłonkaRodziny" />
          </xsd:unique>
        </xsd:element>
        <xsd:element name="projekt" type="Projekt" minOccurs="0"
        ↪ maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  <xsd:unique name="nazwaDziałuUnikatowa">
```

```

    <xsd:selector xpath="dział" />
    <xsd:field xpath="nazwaDziału" />
  </xsd:unique>
  <xsd:unique name="nazwaProjektuUnikatowa">
    <xsd:selector xpath="projekt" />
    <xsd:field xpath="nazwaProjektu" />
  </xsd:unique>
  <xsd:key name="numerProjektuKlucz">
    <xsd:selector xpath="projekt" />
    <xsd:field xpath="numerProjektu" />
  </xsd:key>
  <xsd:key name="numerDziałuKlucz">
    <xsd:selector xpath="dział" />
    <xsd:field xpath="numerDziału" />
  </xsd:key>
  <xsd:key name="peselPracownikaKlucz">
    <xsd:selector xpath="pracownik" />
    <xsd:field xpath="peselPracownika" />
  </xsd:key>
  <xsd:keyref name="peselKierownikaDziałuKluczRef"
    ↪refer="peselPracownikaKlucz">
    <xsd:selector xpath="dział" />
    <xsd:field xpath="peselKierownikaDziału" />
  </xsd:keyref>
  <xsd:keyref name="numerDziałuPracownikaKluczRef" refer="numerDziałuKlucz">
    <xsd:selector xpath="pracownik" />
    <xsd:field xpath="numerDziałuPracownika" />
  </xsd:keyref>
  <xsd:keyref name="peselPrzełożonegoPracownikaKluczRef"
    ↪refer="peselPracownikaKlucz">
    <xsd:selector xpath="pracownik" />
    <xsd:field xpath="peselPrzełożonegoPracownika" />
  </xsd:keyref>
  <xsd:keyref name="numerDziałuProjektuKluczRef" refer="numerDziałuKlucz">
    <xsd:selector xpath="projekt" />
    <xsd:field xpath="numerDziałuProjektu" />
  </xsd:keyref>
  <xsd:keyref name="peselUczestnikaProjektuKluczRef"
    ↪refer="peselPracownikaKlucz">
    <xsd:selector xpath="projekt/uczestnikProjektu" />
    <xsd:field xpath="pesel" />
  </xsd:keyref>
  <xsd:keyref name="numerProjektuPracownikaKluczRef"
    ↪refer="numerProjektuKlucz">
    <xsd:selector xpath="pracownik/pracownikPracujeNad" />
    <xsd:field xpath="numerProjektu" />
  </xsd:keyref>
</xsd:element>
<xsd:complexType name="Dział">
  <xsd:sequence>
    <xsd:element name="nazwaDziału" type="xsd:string" />

```



```

<xsd:element name="numerDziału" type="xsd:string" />
<xsd:element name="peselKierownikaDziału" type="xsd:string" />
<xsd:element name="dataPoczKierownikaDziału" type="xsd:date" />
<xsd:element name="lokalizacjaDziału" type="xsd:string" minOccurs="0"
  ↳maxOccurs="unbounded" />
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Pracownik">
  <xsd:sequence>
    <xsd:element name="imięNazwiskoPracownika" type="ImięINazwisko" />
    <xsd:element name="peselPracownika" type="xsd:string" />
    <xsd:element name="płećPracownika" type="xsd:string" />
    <xsd:element name="pensjaPracownika" type="xsd:unsignedInt" />
    <xsd:element name="dataUrodzeniaPracownika" type="xsd:date" />
    <xsd:element name="numerDziałuPracownika" type="xsd:string" />
    <xsd:element name="peselPrzełożonegoPracownika" type="xsd:string" />
    <xsd:element name="adresPracownika" type="Adres" />
    <xsd:element name="pracownikPracujeNad" type="PracujeNad" minOccurs="1"
      ↳maxOccurs="unbounded" />
    <xsd:element name="członekRodzinyPracownika" type="Rodzina" minOccurs="0"
      ↳maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Projekt">
  <xsd:sequence>
    <xsd:element name="nazwaProjektu" type="xsd:string" />
    <xsd:element name="numerProjektu" type="xsd:string" />
    <xsd:element name="lokalizacjaProjektu" type="xsd:string" />
    <xsd:element name="numerDziałuProjektu" type="xsd:string" />
    <xsd:element name="uczestnikProjektu" type="Uczestnik" minOccurs="1"
      ↳maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Rodzina">
  <xsd:sequence>
    <xsd:element name="imięCzłonkaRodziny" type="xsd:string" />
    <xsd:element name="płećCzłonkaRodziny" type="xsd:string" />
    <xsd:element name="dataUrodzeniaCzłonkaRodziny" type="xsd:date" />
    <xsd:element name="pokrewieństwoCzłonkaRodziny" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Adres">
  <xsd:sequence>
    <xsd:element name="numer" type="xsd:string" />
    <xsd:element name="ulica" type="xsd:string" />
    <xsd:element name="miasto" type="xsd:string" />
    <xsd:element name="województwo" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ImięINazwisko">
  <xsd:sequence>

```

```

    <xsd:element name="imię" type="xsd:string" />
    <xsd:element name="drugieImię" type="xsd:string" />
    <xsd:element name="nazwisko" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Uczestnik">
  <xsd:sequence>
    <xsd:element name="PESEL" type="xsd:string" />
    <xsd:element name="godziny" type="xsd:float" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="PracujeNad">
  <xsd:sequence>
    <xsd:element name="numerProjektu" type="xsd:string" />
    <xsd:element name="godziny" type="xsd:float" />
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>

```

RYSUNEK 13.5. Schemat XML firma

Podobnie jak XML DTD, schematy XML są oparte na modelu drzewa z elementami i ich atrybutami jako głównymi elementami struktury. Obejmują jednak również inne elementy, takie jak klucze, odwołania i identyfikatory, zapożyczone z modeli bazodanowych i obiektowych. Opiszemy tu krok po kroku poszczególne cechy schematów XML, odwołując się do dokumentu schematu XML z rysunku 13.5. Kolejność wprowadzania poszczególnych pojęć odpowiada z grubsza kolejności na listingu.

- (1) **Opis schematu i przestrzeń nazw XML.** Konieczne jest zidentyfikowanie, poprzez wskazanie pliku w sieci WWW, określonego zestawu używanych elementów (znaczników) standardu XML. Drugi wiersz rysunku 13.5 określa plik zastosowany w przykładzie jako „<http://www.w3.org/2001/XMLSchema>”. Jest to najpopularniejszy standard poleceń schematów XML. Każda w ten sposób wskazana definicja jest nazywana **przestrzenią nazw XML**, ponieważ określa stosowany zestaw poleceń (nazw). Nazwa pliku jest przypisana za pomocą atrybutu `xmlns` (*XML namespace* — przestrzeń nazw XML) do zmiennej `xsd` (*XML schema description* — opis schematu XML), która jest używana jako prefiks wszystkich poleceń schematu XML. Dzięki temu na rysunku 13.5 pisząc `xsd:element` lub `xsd:sequence`, odwołujemy się do definicji `element` i `sequence` zawartej w pliku <http://www.w3.org/2001/XMLSchema>.
- (2) **Adnotacje, dokumentacja i język.** Kilka następnych wierszy z rysunku 13.5 zawiera elementy schematu XML `xsd:annotation` i `xsd:documentation`, które służą do przechowywania komentarzy i opisów dokumentu XML. Atrybut `xml:lang` elementu `xsd:documentation` określa zastosowany język dokumentu, a „pl” oznacza język polski.
- (3) **Elementy i typy.** Dalej określony jest *główny element* schematu XML. W schematach XML atrybut `name` znacznika `xsd:element` określa nazwę elementu, którym w przytoczonym na listingu przykładzie jest *firma* (patrz rysunek 13.5). Następnie można określić strukturę elementu głównego, która w przykładzie jest oznaczona jako `xsd:complexType`, a później jest rozwinięta do listy działów, pracowników

i projektów definiowanych za pomocą struktury `xsd:sequence`. Należy przy tym pamiętać, że nie jest to jedyny możliwy sposób zdefiniowania struktury bazy danych `firma`. Zostanie to dalej omówione w podrozdziale 13.6.

- (4) **Elementy pierwszopoziomowe bazy danych FIRMA.** Następnie zdefiniowane są trzy elementy pierwszopoziomowe głównego elementu `firma` z rysunku 13.5. Elementy te to `pracownik`, `dział` i `projekt`, a każdy z nich jest opisany za pomocą znacznika `xsd:element`. Zauważmy, że jeśli znacznik posiada tylko atrybuty i nie ma żadnych elementów potomnych, to może być zakończony znakiem prawego ukośnika (`/>`), co wyklucza potrzebę dodawania znacznika końcowego elementu. Takie elementy nazywamy **pustymi**, a ich przykładami są przedstawione na rysunku 13.5 elementy `xsd:element` o nazwach `dział` i `projekt`.
- (5) **Określanie typu elementu oraz jego minimalnej i maksymalnej liczby wystąpień.** W schemacie XML atrybuty `type`, `minOccurs` i `maxOccurs` znacznika `xsd:element` określają typ i liczbę każdego elementu zgodnego ze specyfikacją schematu. Jeżeli zdefiniujemy atrybut `type` w `xsd:element`, to jego struktura musi zostać opisana oddzielnie, zazwyczaj za pomocą znacznika `xsd:complexType` ze schematu XML. Koncepcja ta jest zilustrowana na rysunku 13.5 definicjami elementów `pracownik`, `dział` i `projekt`. Z drugiej strony jeśli atrybut `type` nie jest określony, to struktura elementu może być definiowana bezpośrednio po znaczniku początkowym elementu, czego przykładem jest definicja elementu głównego `firma` na rysunku 13.5. Atrybuty `minOccurs` i `maxOccurs` są używane do określania dolnej i górnej granicy liczby wystąpień danego elementu w dokumentach zgodnych z określonym schematem XML. Jeżeli atrybuty te nie są określone, to domyślną wartością jest *dokładnie jedno wystąpienie*. Atrybuty te odpowiadają funkcjonalnie symbolom `*`, `+` i `?` z definicji typu dokumentu (DTD).
- (6) **Określanie kluczy.** W schemacie XML możliwe jest definiowanie ograniczeń odpowiadających unikatowym i głównym kluczom relacyjnych baz danych (patrz punkt 5.2.2) oraz kluczom obcym (lub inaczej integralności odwołań — patrz punkt 5.2.4). Znacznik `xsd:unique` wskazuje elementy, które odpowiadają unikatowym atrybutom relacyjnej bazy danych. Każdemu wymogowi unikatowości można nadać osobną nazwę; trzeba też określić w atrybucie `xpath` elementy `xsd:selector` i `xsd:field` identyfikujące typ i nazwę elementu. Na rysunku 13.5 ilustrują to elementy `nazwaDziałuUnikatowa` i `nazwaProjektuUnikatowa`. Do definiowania **kluczy głównych** służy znacznik `xsd:key`, co jest zilustrowane elementami `numerProjektuKlucz`, `numerDziałuKlucz` i `peselPracownikaKlucz`, a do definiowania **kluczy obcych** służy znacznik `xsd:keyref`, co ilustruje sześć elementów tego typu przedstawionych na rysunku 13.5. Podczas definiowania klucza obcego należy w atrybucie `refer` elementu `xsd:keyref` wskazać klucz główny, a w znacznikach `xsd:selector` i `xsd:field` — typ i klucz obcy elementu referencyjnego (rysunek 13.5).
- (7) **Określanie struktur elementów złożonych za pomocą złożonych typów.** Następną część przykładu definiuje za pomocą znacznika `xsd:complexType` typy złożonych elementów `Dział`, `Pracownik`, `Projekt` i `Rodzina` (patrz rysunek 13.5). Każdy z nich jest zdefiniowany za pomocą znaczników schematu XML `xsd:sequence` i `xsd:element` jako sekwencja podelementów odpowiadających atrybutom bazy danych poszczególnych typów encji (patrz rysunek 7.7). Każdemu elementowi atrybuty `name` i `type` znacznika `xsd:element` przypisują nazwę i typ. Można również zastosować atrybuty `minOccurs` i `maxOccurs`, jeśli istnieje potrzeba

zmiany domyślnej liczności dokładnie jednego elementu. W przypadku atrybutów opcjonalnych należy dla odpowiedniego elementu zdefiniować atrybut XML `minOccurs = 0`, a w przypadku atrybutów wielowartościowych `maxOccurs = „unbounded”`. Zauważmy, że jeśli nie są stosowane żadne więzy kluczy, to elementy potomne mogą być zdefiniowane bezpośrednio w elemencie nadrzędnym bez potrzeby stosowania konstrukcji typu złożonego. W przeciwnym jednak wypadku (kiedy stosowane są klucze główny, obcy lub unikatowy) trzeba zdefiniować typy, by móc je przypisać poszczególnym elementom schematu.

- (8) **Atrybuty kompozytowe (złożone).** Atrybuty kompozytowe z rysunku 9.2 są na rysunku 13.5 również określone jako typy złożone (na przykład `Adres`, `ImięINazwisko`, `Uczestnik` i `PracujeNad`). Mogą być one definiowane bezpośrednio wewnątrz ich elementów nadrzędnych (rodziców).

Przytoczony przykład ilustruje niektóre podstawowe cechy schematów XML. Istnieje ich znacznie więcej, ale wymienianie wszystkich nie jest celem niniejszej książki. W następnym podrozdziale omówione są różne techniki tworzenia dokumentów XML na podstawie relacyjnych baz danych oraz późniejszego przechowywania tych dokumentów.

## 13.4. Zapisywanie dokumentów XML w bazach i ich pobieranie

Istnieje kilka metod porządkowania zawartości dokumentów XML w celu ułatwienia zgłaszania zapytań i pobierania danych. Najczęściej spotykane z nich to:

- (1) **Zastosowanie systemu plików lub SZBD do przechowywania dokumentów w postaci tekstowej.** Dokument XML można zapisać jako plik tekstowy w tradycyjnym systemie plików. Relacyjny system baz danych może być wykorzystany do przechowywania całych dokumentów XML w polach tekstowych wewnątrz rekordów bazy danych. Takie podejście może być wykorzystane w stosunku do dokumentocentrycznych lub pozbawionych schematu dokumentów XML, jeśli system bazy danych jest wyposażony w specjalny moduł do przetwarzania dokumentów tekstowych.
- (2) **Zastosowanie systemu baz danych do przechowywania elementów dokumentu jako elementów danych.** To podejście ma zastosowanie przy przechowywaniu kolekcji dokumentów o wspólnym określonym DTD lub schemacie XML. Ponieważ wszystkie dokumenty współdzielą jedną strukturę, można zaprojektować relacyjną bazę danych przechowującą wartości liści z dokumentu XML. To podejście wymaga algorytmów odwzorowujących, aby zaprojektować schemat bazy danych zgodny ze strukturą dokumentu XML podaną w schemacie XML lub DTD oraz odtwarzać dokumenty XML z zapisanych danych. Algorytmy te mogą być zrealizowane w postaci wewnętrznych modułów systemu baz danych lub oddzielnego oprogramowania warstwy pośredniej. Jeśli wszystkie elementy dokumentu XML mają identyfikator, prostą reprezentacją jest utworzenie tabeli z atrybutami `XDOC(DId, Rid, Eznaczn, War)`, gdzie `DId` i `Rid` to identyfikatory dziecka i rodzica, `Eznaczn` to nazwa elementu o identyfikatorze dziecka, a `War` to wartość tego elementu (jeśli analizowany jest węzeł liścia). Zakładamy tu, że wszystkie wartości są tego samego typu.

- (3) **Opracowanie specjalizowanego systemu do przechowywania macierzystych danych dokumentu XML.** Można zaprojektować i zrealizować specjalny nowy system bazy danych na podstawie modelu hierarchicznego (drzewiastego). Takie systemy to **natywne XML-owe SZBD**. System taki mógłby realizować specjalne techniki zapytań i indeksowania, które działałyby z każdym rodzajem dokumentu XML. Mógłby również zawierać mechanizmy kompresujące dane i zmniejszające tym samym wielkość składowanych dokumentów. Dwa popularne produkty z natywnymi XML-owymi mechanizmami to Tamino firmy Software AG i Dynamic Application Platform firmy eXcelon. Także Oracle umożliwia składowanie natywnych XML-owych danych.
- (4) **Tworzenie lub publikowanie dowolnych dokumentów XML na podstawie istniejących relacyjnych baz danych.** Ponieważ w relacyjnych bazach danych zgromadzono już wyjątkowo dużo danych, część z nich może być sformatowana w celu łatwej wymiany danych lub wyświetlania w internecie. Przy tym podejściu wymagane jest zastosowanie oprogramowania warstwy pośredniej w transformacji danych z relacyjnych baz danych do postaci dokumentów XML. To podejście, polegające na pobieraniu przeznaczonych na dane dokumentów XML z istniejących baz, jest szczegółowo opisane w podrozdziale 13.6. Przede wszystkim pokażemy tam, jak tworzyć drzewiaste dokumenty strukturalne na podstawie płaskich relacyjnych baz zaprojektowanych w oparciu o grafie modelu ER. W punkcie 13.6.2 opiszemy problem cykli i radzenia sobie z nimi.

W ciągu ostatnich lat każde z wymienionych podejść przyciągało uwagę użytkowników. W podrozdziale 13.6 skupimy się na metodzie 4., ponieważ pozwala ona na rozważanie różnic pomiędzy strukturą drzewa dokumentów XML, tradycyjnymi bazami danych opartymi na płaskiej strukturze plików (model relacyjny) oraz reprezentacjami grafowymi modelu ER. Najpierw jednak w podrozdziale 13.5 przedstawimy przegląd języków zapytań powiązanych ze standardem XML.

## 13.5. Języki związane ze standardem XML

Istnieje kilka propozycji dotyczących języków zapytań XML, ale tylko dwie z nich stały się obowiązującymi standardami. Pierwszy to **XPath**, pozwalający na konstruowanie w dokumencie XML wyrażeń ścieżkowych identyfikujących określone węzły (elementy) lub atrybuty, które spełniają zadane warunki. Drugi ze standardów to **XQuery**, który jest bardziej ogólnym językiem zapytań. XQuery używa wyrażeń XPath, ale oferuje kilka dodatkowych konstrukcji. W niniejszym podrozdziale przedstawione są podstawy każdego z tych standardów. Dalej, w punkcie 13.5.3, opiszemy dodatkowe języki związane ze standardem HTML.

### 13.5.1. XPath, czyli określanie ścieżek w dokumentach XML

Wyrażenie XPath zwraca zbiór jednostek, które spełniają określone w wyrażeniu warunki. Tymi jednostkami są albo wartości (z liści), albo elementy, albo atrybuty. Najczęściej stosowany rodzaj wyrażeń XPath zwraca kolekcję węzłów elementów lub atrybutów zgodnych z określonym wzorcem podanym w wyrażeniu. Nazwy używane wewnątrz wyrażenia

XPath są nazwami węzłów w dokumencie XML (elementów bądź atrybutów) z ewentualnymi **warunkami kwalifikującymi** dodatkowo ograniczającymi zbiór węzłów w wyniku. Dwoma głównymi **separatorami** używanymi w XPath są znaki pojedynczego i podwójnego ukośnika (/ i //). Pojedynczy ukośnik przed znacznikiem oznacza, że musi on być bezpośrednim dzieckiem poprzedniego (nadrzędnego) znacznika, natomiast podwójny ukośnik oznacza potomka *dowolnego stopnia*. Aby zastosować nazwę atrybutu zamiast nazwy elementu (znacznika), należy przed nazwą podać przedrostek @. Kilka przykładów wyrażeń XPath zostało przedstawionych na rysunku 13.6.

1. /firma
2. /firma/dział
3. //pracownik [pensjaPracownika gt 7000]/imięNazwiskoPracownika
4. /firma/pracownik [pensjaPracownika gt 7000]/imięNazwiskoPracownika
5. /firma/projekt/uczestnikProjektu [godziny © 20.0]

RYSUNEK 13.6. Przykładowe wyrażenia XPath dla dokumentów XML zgodnych ze schematem XML FIRMA przedstawionym na rysunku 13.5

Pierwsze wyrażenie z rysunku 13.6 zwraca jako rezultat węzeł główny *firma* i wszystkie jego węzły podrzędne, czyli cały dokument XML. Należy nadmienić, że w wyrażeniu XPath można opcjonalnie zawrzeć nazwę pliku z dokumentem XML. Pozwala to na wskazanie lokalnego pliku lub nawet ścieżki prowadzącej do pliku w internecie. Jeżeli na przykład dokument XML FIRMA jest dostępny pod adresem

`www.firma.pl/info.xml`

to pierwsze wyrażenie z rysunku 13.6 można zapisać jako

`doc(www.firma.pl/info.xml)/firma`

Taki sam przedrostek można dodać również do pozostałych przykładowych wyrażeń XPath.

Drugie wyrażenie rysunku 13.6 zwraca wszystkie węzły (elementy) odpowiadające działom wraz z ich poddrzewami potomnymi. Węzły (elementy) w dokumencie XML są uporządkowane, więc wynik wyrażenia XPath zawierający wiele węzłów będzie je zawierał w takiej samej kolejności, w jakiej są umieszczone w oryginalnym dokumencie.

Trzecie wyrażenie XPath z rysunku 26.7 ilustruje zastosowanie symbolu //, użyteczne w sytuacji, gdy nie jest znana pełna ścieżka elementu, a jedynie nazwy niektórych szukanych elementów. Jest to przydatne przede wszystkim w sytuacji, w której mamy do czynienia z dokumentem bez określonego schematu lub w przypadku dokumentu z wieloma zagnieżdżonymi znacznikami<sup>6</sup>.

Wyrażenie zwraca wszystkie elementy `imięNazwiskoPracownika` będące bezpośrednimi potomkami takich węzłów `pracownik`, których inny węzeł potomny o nazwie `pensjaPracownika` ma wartość większą niż 7000. Ilustruje to zastosowanie warunku kwalifikującego, który ogranicza wybrane węzły do spełniających dany warunek. XPath umożliwia używanie standardowych operatorów porównania, w tym operatorów arytmetycznych, zbiorów i ciągów znaków.

---

<sup>6</sup> Terminy *węzeł*, *znacznik* i *element* mogą być tu używane wymiennie.

Czwarte wyrażenie z rysunku 13.6 powinno zwrócić ten sam rezultat co poprzednie, z tym że używa pełnej ścieżki do elementu. Piąty przykład zwraca wszystkie węzły uczestnik ↪Projektu (wraz z ich poddrzewami), które są elementami potomnymi węzła /firma/projekt i których elementy potomne o nazwie godziny mają wartość większą lub równą 20,0.

Gdy w wyrażeniu XPath potrzebne są atrybuty, nazwa atrybutu jest poprzedzana symbolem @, aby odróżnić ją od nazwy elementu (znacznika). Można też zastosować symbol **wieloznacznik** \*, który reprezentuje dowolny element. Poniższe przykładowe wyrażenie pobiera wszystkie elementy (niezależnie od ich typu), które są dziećmi korzenia. Gdy stosowane są symbole wieloznacznik, wynikiem może być sekwencja elementów różnego typu.

```
/firma/*
```

Wcześniejsze przykłady ilustrują proste wyrażenia XPath, w których można przechodzić z danego węzła tylko w dół struktury drzewiastej. Zaproponowano też ogólniejszy model wyrażen ścieżkowych, w których można poruszać się w wielu kierunkach z danego węzła. Te kierunki to **osie** wyrażenia XPath. W pokazanych przykładach używane były tylko *trzy osie*: dziecko bieżącego węzła (/), bieżący węzeł lub jego potomek z dowolnego poziomu (//) i atrybut bieżącego węzła (@). Inne osie to rodzic, przodek (z dowolnego poziomu), poprzedni brat (występujący po lewej węzeł z tego samego poziomu) i następny brat (znajdujący się po prawej węzeł z tego samego poziomu). Te osie umożliwiają tworzenie bardziej złożonych wyrażen ścieżkowych.

Głównym ograniczeniem wyrażen ścieżkowych XPath jest to, że ścieżka ze wzorcem określa też pobierane elementy. Dlatego trudno jest zdefiniować warunki we wzorcu i niezależnie od tego podać, które wynikowe elementy należy pobrać. W języku XQuery te dwa zagadnienia są niepowiązane, a język ten zapewnia bardziej rozbudowane mechanizmy tworzenia zapytań.

### 13.5.2. XQuery: definiowanie zapytań w XML

XPath pozwala na definiowanie wyrażen wybierających węzły z dokumentów XML o strukturze drzewa. XQuery pozwala na określanie bardziej ogólnych zapytań dotyczących jednego lub kilku dokumentów XML. Typowa forma zapytania XQuery jest określana jako **wyrażenie FLWOR**. Nazwa ta pochodzi od pierwszych liter pięciu podstawowych klauzul XQuery:

```
FOR <przypisanie zmiennej poszczególnych węzłów (elementów)>
LET <przypisanie zmiennej kolekcji węzłów (elementów)>
WHERE <warunek kwalifikujący>
ORDER BY <specyfikacje sortowania>
RETURN <określenie wyniku zapytania>
```

Liczba klauzul FOR i LET w jednym zapytaniu XQuery to zero lub więcej. Klauzule WHERE i ORDER BY są opcjonalne, ale mogą występować najwyżej raz. Klauzula RETURN musi występować dokładnie raz. Zilustrujmy te klauzule za pomocą prostego przykładowego zapytania XQuery.



```

LET $d := doc(www.firma.pl/info.xml)
FOR $x IN $d/firma/projekt[numerProjektu = 5]/uczestnikProjektu,
  $y IN $d/firma/pracownik
WHERE $x/godziny gt 20.0 AND $y.pesel = $x.pesel
ORDER BY $x/godziny
RETURN <res> $y/imięNazwiskoPracownika/imię,
  $y/imięNazwiskoPracownika/nazwisko,
  $x/godziny </res>

```

- (1) Zmienne są poprzedzone znakiem \$. W tym przykładzie zmienne to \$d, \$x i \$y. Klauzula LET przypisuje konkretne wyrażenie do zmiennej, po czym można jej używać w dalszej części zapytania. Tu do \$d przypisywana jest nazwa pliku z dokumentem. Można utworzyć zapytanie, które dotyczy wielu dokumentów, przypisując w ten sposób wartości do wielu zmiennych.
- (2) Klauzula FOR ustawia zmienną używaną do przetwarzania poszczególnych elementów z sekwencji. Tu sekwencje są określone za pomocą wyrażień ścieżkowych. Zmienna \$x obejmuje elementy zgodne z wyrażeniem ścieżkowym \$d/firma/projekt[numerProjektu = 5]/uczestnikProjektu. Zmienna \$y obejmuje elementy zgodne z wyrażeniem ścieżkowym \$d/firma/pracownik. Tak więc \$x odpowiada elementom uczestnikProjektu reprezentującym pracowników zajmujących się projektem 5., a zmienna \$y odpowiada elementom pracownik.
- (3) Klauzula WHERE określa dodatkowe warunki pobierania elementów. Tu pierwszy warunek powoduje pobranie tylko tych elementów uczestnikProjektu, które spełniają warunek (godziny gt 20.0). Drugi warunek to warunek złączenia wiążący elementy pracownik i uczestnikProjektu tylko wtedy, jeśli mają one ten sam numer PESEL.
- (4) Klauzula ORDER BY określa tu, że wynikowe elementy będą uporządkowane rosnąco według liczby godzin poświęcanych przez pracowników tygodniowo na dany projekt.
- (5) Klauzula RETURN pokazuje, które elementy lub atrybuty należy pobrać z elementów spełniających warunki zapytania. W tym przykładzie zwracana jest sekwencja elementów zawierających atrybuty <imię, nazwisko, godziny> i reprezentujących pracowników, którzy pracują ponad 20 godzin tygodniowo nad projektem 5.

Rysunek 13.7 przedstawia przykłady zapytań XQuery, które dotyczą dokumentów XML spełniających warunki schematu opisanego na rysunku 13.5. Pierwsze zapytanie odczytuje imiona i nazwiska pracowników, którzy zarabiają więcej niż 7000 zł. Zmienna \$x jest przypisana do każdego elementu imięNazwiskoPracownika, który jest dzieckiem elementu pracownik, który z kolei spełnia warunek mówiący, że wartość pensjaPracownika jest większa od 7000. Wynik zwraca elementy podrzędne imię i nazwisko pobranych elementów imię ↪ NazwiskoPracownika. Drugie zapytanie jest innym sposobem osiągnięcia tego samego wyniku co w zapytaniu pierwszym.

```

a. FOR $x IN
  doc(www.firma.pl/info.xml)//pracownik [pensjaPracownika gt 7000]/
  ⚡ imięNazwiskoPracownika
  RETURN <res> $x/imię, $x/nazwisko </res>

b. FOR $x IN
  doc(www.firma.pl/info.xml)/firma/pracownik
  WHERE $x/pensjaPracownika gt 7000

```

```

    RETURN <res> $x/imięNazwiskoPracownika/imię.
           $x/imięNazwiskoPracownika/nazwisko</res>
  c.FOR $x IN
  doc(www.firma.pl/info.xml)/firma/projekt[numerProjektu = 5]/
  ↳uczestnikProjektu,
    $y IN doc(www.firma.pl/info.xml)/firma/pracownik
    WHERE $x/godziny gt 20.0 AND $y.pesel = $x.pesel
  RETURN <res> $y/imięNazwiskoPracownika/imię.
           $y/imięNazwiskoPracownika/nazwisko, $x/godziny </res>

```

RYSUNEK 13.7. Przykładowe zapytania XQuery dla dokumentów XML odpowiadających schematowi FIRMA z rysunku 13.5

Trzecie zapytanie ilustruje użycie operacji złączenia więcej niż jednej zmiennej. Zmiennej *\$x* jest przypisany każdy element *uczestnikProjektu*, który jest potomkiem projektu numer 5, a zmiennej *\$y* jest przypisany każdy element *pracownik*. Warunek złączenia dopasowuje numer PESEL pracownika do uczestników projektu. Zauważ, że jest to inny sposób na zapisanie tego samego zapytania co we wcześniejszym przykładzie, ale bez klauzuli *LET*.

XQuery udostępnia bardzo rozbudowane mechanizmy do tworzenia złożonych zapytań. Przede wszystkim pozwala stosować kwantyfikatory uniwersalne i egzystencjalne w warunkach zapytania, funkcje agregujące, instrukcje sortowania wyników zapytania, selekcję opartą na pozycjach w sekwencji, a nawet warunkowe wykonywanie instrukcji. Dlatego pod pewnymi względami jest kompletnym językiem programowania.

Ten przykład kończy rozważania na temat XQuery. Czytelnik zainteresowany szczegółami powinien zapoznać się z zawartością witryny [www.w3.org](http://www.w3.org), która zawiera informacje o standardach związanych z XML i XQuery. W następnym punkcie pokrótce omówimy inne języki i protokoły związane ze standardem XML.

### 13.5.3. Inne języki i protokoły związane ze standardem XML

Istnieje też kilka innych języków i protokołów powiązanych z technologią XML. Długoterminowym celem ich tworzenia jest zapewnienie technologii dla sieci semantycznej, w której wszystkie informacje dostępne w internecie można w inteligentny sposób znajdować i przetwarzać.

- XSL (ang. *Extensible Stylesheet Language*) służy do definiowania sposobu wyświetlania dokumentu przez przeglądarki internetowe.
- XSLT (ang. *Extensible Stylesheet Language for Transformations*) służy do przekształcania jednej struktury w inną. Dlatego pozwala przetwarzać dokumenty z jednej postaci w inną.
- WSDL (ang. *Web Services Description Language*) umożliwia opis usług sieciowych w standardzie XML. Zapewnia to dostęp do usługi sieciowej użytkownikom i programom w internecie.
- SOAP (ang. *Simple Object Access Protocol*) to niezależny od systemu i języków programowania protokół przesyłania komunikatów i zdalnych wywołań procedur.

- RDF (ang. *Resource Description Framework*) zapewnia języki i narzędzia do wymiany oraz przetwarzania opisów metadanych (schematów) i specyfikacji w internecie.

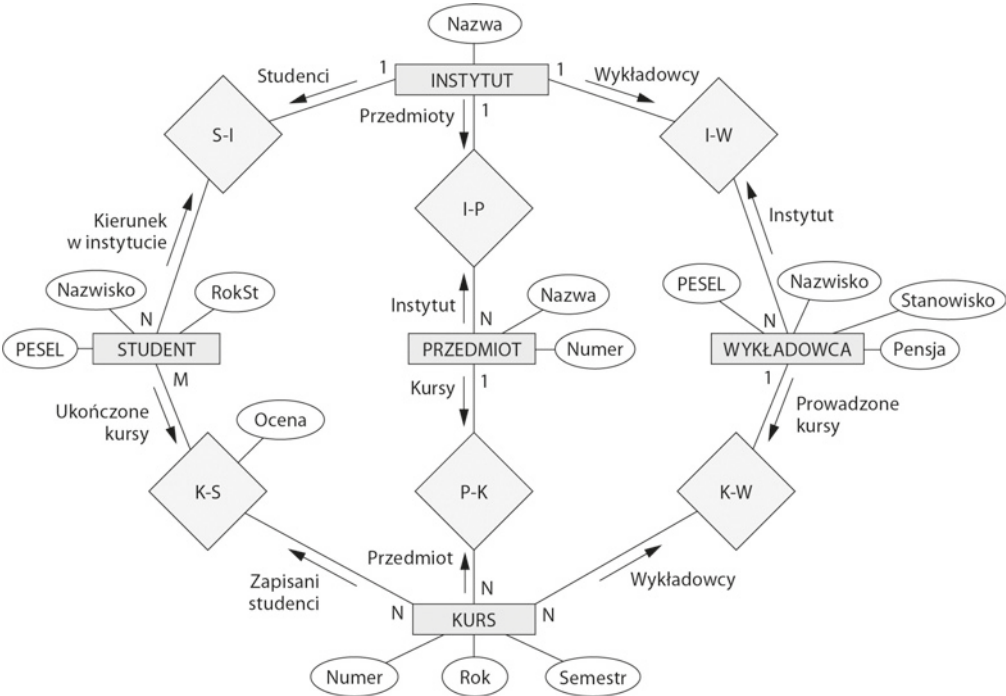
## 13.6. Pobieranie dokumentów XML z relacyjnych baz danych

### 13.6.1. Tworzenie hierarchicznych perspektyw w formacie XML dla danych płaskich lub zapisanych w grafie

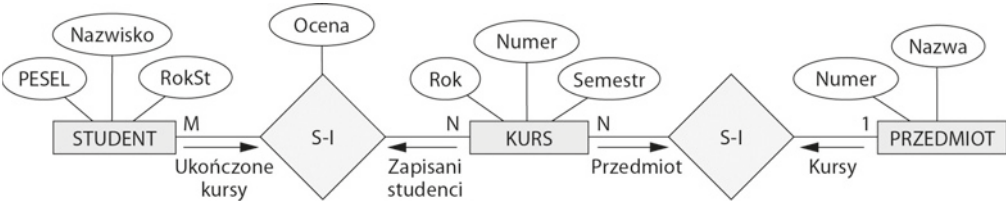
W niniejszym punkcie omówione zostaną problemy reprezentacji związane z transformacją danych z systemu bazy danych w dokument XML. Jak już zostało wspomniane, w dokumentach XML stosowany jest hierarchiczny (drzewiasty) model danych. Najczęściej używane systemy bazodanowe stosują natomiast płaski model relacyjny. Po dodaniu do modelu relacyjnego więzów integralności odwołań staje się on strukturą grafową (patrz rysunek 3.7). Podobnie reprezentowany jest również model ER (na przykład na rysunku 7.2). W rozdziale 9. przedstawiono odwzorowania pozwalające na przekształcanie modelu ER w model relacyjny i odwrotnie, można więc koncepcyjnie przedstawić schemat bazy relacyjnej za pomocą odpowiadającego mu schematu ER. Wystarczy więc, że w naszych rozważaniach ograniczymy się do przekształcania schematu ER w dokument XML, a te same uwagi będą dotyczyły również baz danych w modelu relacyjnym.

Do ilustrowania naszych rozważań użyjemy uproszczonego schematu ER UNIWERSYTET przedstawionego na rysunku 13.8. Przypuśćmy, że aplikacja ma za zadanie stworzyć na podstawie bazy danych dokumenty XML zawierające informacje o studencie, przedmiocie i ocenach. Dane potrzebne do stworzenia tych dokumentów są zapisane w atrybutach bazy danych typów encji PRZEDMIOT, KURS i STUDENT przedstawionych na rysunku 13.8 oraz w istniejących pomiędzy nimi relacjach K-S i P-K. Większość dokumentów tworzonych na podstawie bazy danych będzie używała tylko części atrybutów, typów encji i relacji bazy danych. W tym przypadku potrzebny podzbiór jest przedstawiony na rysunku 13.9.

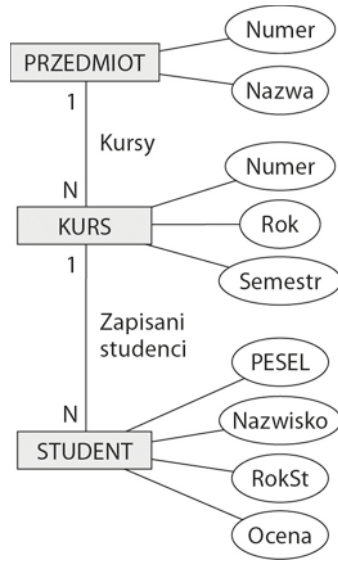
Istnieją co najmniej trzy hierarchie możliwe do stworzenia w oparciu o podzbiór bazy danych z rysunku 13.9. Najpierw wybieramy jako korzeń drzewa element PRZEDMIOT tak, jak pokazano na rysunku 13.10. W tym przypadku każda encja reprezentująca przedmiot ma podelementy odpowiadające kursom tego przedmiotu, a każdy kurs zawiera jako podelementy studentów. Widoczną konsekwencją modelowania danych w strukturze drzewa jest to, że jeśli student jest zapisany na kilka kursów, to informacja o nim pojawi się w dokumencie kilkakrotnie — po jednym razie dla każdego kursu. Możliwe uproszczenie schematu XML jest przedstawione na rysunku 13.11. Atrybut Ocena relacji K-S jest przeniesiony do elementu STUDENT. Przeniesienie takie jest możliwe, ponieważ STUDENT jest w drzewie XML dzieckiem elementu KURS, więc każdy STUDENT może mieć przypisany pod każdym kursem odpowiadającą mu ocenę. W przyjętej hierarchii student zapisany na więcej niż jeden kurs będzie miał kilka replik odpowiadającego mu elementu i każda z replik będzie miała osobną ocenę związaną z nadrzędnym elementem — kursem.



RYSUNEK 13.8. Diagram schematu ER dla uproszczonej bazy danych UNIWERSYTET



RYSUNEK 13.9. Podzbiór schematu bazy danych UNIWERSYTET potrzebny do stworzenia dokumentu XML



RYSUNEK 13.10. Widok hierarchiczny z elementem PRZEDMIOT jako korzeniem drzewa

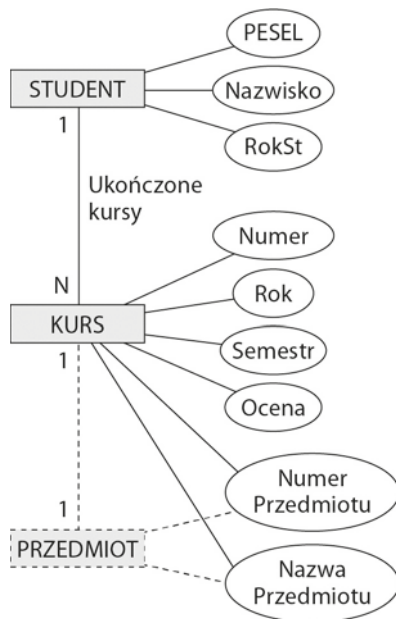
```

<xsd:element name="root">
  <xsd:sequence>
    <xsd:element name="przedmiot" minOccurs="0" maxOccurs="unbounded">
      <xsd:sequence>
        <xsd:element name="nazwap" type="xsd:string" />
        <xsd:element name="numerp" type="xsd:unsignedInt" />
        <xsd:element name="kurs" minOccurs="0" maxOccurs="unbounded">
          <xsd:sequence>
            <xsd:element name="numerkursu" type="xsd:unsignedInt" />
            <xsd:element name="rok" type="xsd:string" />
            <xsd:element name="semestr" type="xsd:string" />
            <xsd:element name="student" minOccurs="0" maxOccurs="unbounded" />
            <xsd:sequence>
              <xsd:element name="pesel" type="xsd:string" />
              <xsd:element name="nazwiskost" type="xsd:string" />
              <xsd:element name="rokst" type="xsd:string" />
              <xsd:element name="ocena" type="xsd:string" />
            </xsd:sequence>
          </xsd:element>
        </xsd:sequence>
      </xsd:element>
    </xsd:sequence>
  </xsd:element>
</xsd:sequence>
</xsd:element>

```

RYSUNEK 13.11. Dokument schematu XML z elementem PRZEDMIOT jako elementem głównym

W drugim modelu hierarchicznym (przedstawionym na rysunku 13.12), jako korzeń drzewa wybrano element `STUDENT`. W tym widoku każdy student ma przypisany zbiór kursów będących jego dziećmi w drzewie, natomiast każdy kurs ma przypisany dokładnie jeden przedmiot (ponieważ relacja pomiędzy kursem a przedmiotem ma licznosc N:1). W tym przypadku można złączyć elementy `KURS` i `PRZEDMIOT` w jedno, co zostało przedstawione na rysunku 13.12. Ponadto atrybut `ocena` może być również przypisany kursowi. W tej hierarchii złączona informacja elementów `KURS` i `PRZEDMIOT` jest replikowana dla każdego elementu nadrzędnego `STUDENT`. Schemat XML odpowiadający takiej hierarchii jest przedstawiony na rysunku 13.13.



RYСУNEK 13.12. Widok hierarchiczny z elementem `STUDENT` jako korzeniem drzewa

```
<xsd:element name="root">
  <xsd:sequence>
    <xsd:element name="student" minOccurs="0" maxOccurs="unbounded">
      <xsd:sequence>
        <xsd:element name="pesel" type="xsd:string" />
        <xsd:element name="nazwiskost" type="xsd:string" />
        <xsd:element name="rokst" type="xsd:string" />
        <xsd:element name="kurs" minOccurs="0" maxOccurs="unbounded">
          <xsd:sequence>
            <xsd:element name="numerkursu" type="xsd:unsignedInt" />
            <xsd:element name="rok" type="xsd:string" />
            <xsd:element name="semestr" type="xsd:string" />
            <xsd:element name="numerp" type="xsd:unsignedInt" />
            <xsd:element name="nazwap" type="xsd:string" />
            <xsd:element name="ocena" type="xsd:string" />
          </xsd:sequence>
        </xsd:element>
      </xsd:sequence>
    </xsd:element>
  </xsd:sequence>
</xsd:element>
```

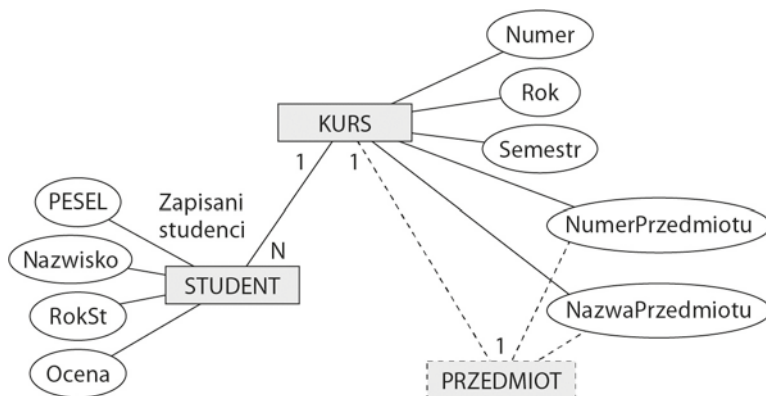
```

    </xsd:element>
  </xsd:sequence>
</xsd:element>
</xsd:sequence>
</xsd:element>

```

RYSUNEK 13.13. Dokument schematu XML z elementem STUDENT jako elementem głównym

Trzeci możliwy model hierarchiczny przyjmuje jako korzeń drzewa element KURS, jak to zostało pokazane na rysunku 13.14. Podobnie jak we wcześniejszym przypadku, informacja z elementu PRZEDMIOT może zostać włączona do elementu KURS, zaś atrybut OCENA — przeniesiony do elementu STUDENT. Jak widać, nawet w tak prostym przykładzie istnieje kilka zupełnie odmiennych możliwości przeniesienia schematu bazy danych na strukturę XML z różnymi elementami głównymi i odmienną konstrukcją danych.



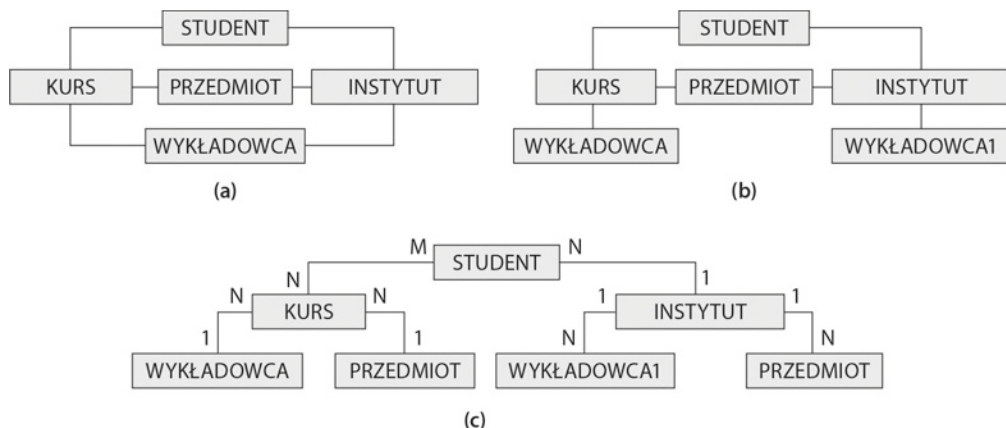
RYSUNEK 13.14. Widok hierarchiczny (drzewiasty) z elementem KURS jako korzeniem drzewa

### 13.6.2. Przerywanie cykli w celu zamiany grafów w drzewa

We wcześniejszych przykładach używany wycinek bazy danych był pozbawiony cykli. W bardziej złożonych podzbiorach danych możliwe jest jednak zaistnienie cykli odpowiadających kilku różnym relacjom pomiędzy encjami. W takim przypadku konstruowanie hierarchii dokumentu XML na podstawie schematu bazy danych staje się znacznie bardziej skomplikowane. Do przedstawienia wielu relacji konieczne może być dodatkowe zwielokrotnienie używanych elementów. Zilustrujemy taki przypadek przykładowym schematem ER z rysunku 13.8.

Przypuśćmy, że określony dokument XML ma zawierać wszystkie informacje o typach encji i relacjach przedstawionych na rysunku 13.8, a jego elementem głównym ma być STUDENT. Możliwa struktura drzewa dla takiego dokumentu jest przedstawiona na rysunku 13.15. Najpierw tworzona jest krata z elementem STUDENT jako głównym, co zostało pokazane na rysunku 13.15(a). W związku z istnieniem cykli nie jest to jeszcze struktura drzewa. Jednym ze sposobów przerywania cykli jest tworzenie replik węzłów tworzących cykle. Najpierw replikowany jest węzeł WYKŁADOWCA, a replika nazwana jest WYKŁADOWCA1 (patrz rysunek 13.15(b)). Replika WYKŁADOWCA po lewej stronie diagramu odpowiada





RYСУNEK 13.15. Przetwarzanie grafu z cyklami w hierarchiczną strukturę drzewa

relacji pomiędzy wykładowcą a prowadzonym przez niego kursem, natomiast replika WYKŁADOWCA1 po prawej stronie odpowiada relacji pomiędzy wykładowcą a zatrudniającym go instytutem. Następnie przerywany jest następny cykl poprzez replikację węzła PRZEDMIOT, co prowadzi do powstania hierarchii przedstawionej na rysunku 13.15(c). Replika PRZEDMIOT1 po lewej stronie odpowiada relacji pomiędzy przedmiotami a ich kursami, natomiast replika PRZEDMIOT po prawej stronie reprezentuje relację pomiędzy instytutem a oferowanymi w nim kursami.

Na rysunku 13.15(c) przetworzono początkowy graf w strukturę hierarchiczną. Przed stworzeniem ostatecznego schematu hierarchii można połączyć niektóre elementy (jak to zostało przedstawione we wcześniejszym przykładzie).

### 13.6.3. Dodatkowe kroki związane z tworzeniem dokumentu XML na podstawie bazy danych

Poza stworzeniem odpowiedniej hierarchii XML i odpowiadającego jej dokumentu schematu XML, należy w celu pełnego odwzorowania bazy danych na dokument XML podjąć następujące kroki:

- (1) Trzeba stworzyć stosowne zapytanie SQL wyłuskujące z bazy danych informacje odpowiednie dla tworzonego dokumentu XML.
- (2) Po wykonaniu zapytania, płaska struktura wyniku musi zostać przekształcona w drzewo XML.
- (3) Zapytanie może być modyfikowane tak, aby pobierało pojedynczy obiekt lub wszystkie obiekty związane z dokumentem. Na przykład w przypadku widoku z rysunku 13.13 zapytanie może zwracać informacje związane z jednym studentem (i tworzyć odpowiadający mu dokument) lub z grupą, a nawet ze wszystkimi studentami.

## 13.7. XML/SQL: funkcje języka SQL generujące dane w formacie XML

W tym podrozdziale opiszemy wybrane funkcje dodane w nowych wersjach standardu SQL na potrzeby generowania danych w formacie XML na podstawie baz relacyjnych. Te funkcje można wykorzystać do formatowania wyników zapytań do postaci elementów i dokumentów XML oraz do określania korzeni hierarchii XML, co pozwala tworzyć zagnieżdżone hierarchie na podstawie płaskich danych relacyjnych. Najpierw wymienimy i pokrótce opiszemy niektóre funkcje dodane do standardu SQL. Następnie przedstawimy kilka przykładów.

Omówimy tu następujące funkcje:

- (1) **XMLEMENT**. Służy ona do podawania nazwy znacznika (elementu) występującego w wynikowym dokumencie XML. Można podać nazwę znacznika odpowiadającego złożonemu elementowi lub pojedynczej kolumnie.
- (2) **XMLFOREST**. Jeśli w wynikowym dokumencie XML potrzebnych jest kilka znaczników (elementów), ta funkcja pozwala utworzyć je w prostszy sposób niż **XMLEMENT**. Można bezpośrednio podać nazwy kolumn rozdzielone przecinkami; opcjonalnie można zmienić te nazwy. Jeśli nazwa kolumny nie zostanie zmieniona, zostanie użyta jako nazwa elementu (znacznika).
- (3) **XMLAGG**. Ta funkcja umożliwia grupowanie (agregację) kilku elementów w celu powiązania ich z elementem nadrzędnym, tworząc kolekcję elementów podrzędnych.
- (4) **XMLROOT**. Pozwala sformatować wybrane elementy jako dokument XML z jednym korzeniem.
- (5) **XMLATTRIBUTES**. Umożliwia tworzenie atrybutów elementów z wynikowego dokumentu XML.

Teraz przedstawimy te funkcje za pomocą kilku przykładów opartych na SQL/XML. Wykorzystamy w nich tabelę **PRACOWNIK** z rysunków 5.5 i 5.6. Pierwszy przykład, X1, pokazuje, jak utworzyć element XML z nazwiskiem pracownika o numerze PESEL 65010912345.

```
X1:  SELECT  XMLEMENT(NAME "nazwisko", E.Nazwisko)
      FROM    PRACOWNIK E
      WHERE   E.PESEL = "65010912345" ;
```

Słowo kluczowe **NAME** ze standardu SQL określa nazwę elementu (znacznika) XML. Wynikiem dla danych z rysunku 5.6 jest:

```
<nazwisko>Szewczyk</nazwisko>
```

Jeśli chcesz pobrać kilka kolumn jednego wiersza, możesz się posłużyć kilkoma wywołaniami **XMLEMENT** w ramach elementu nadrzędnego, ale prościej jest zastosować funkcję **XMLFOREST**, która pozwala podać wiele kolumn bez powtarzania słowa kluczowego **XMLEMENT**. Ilustruje to zapytanie X2:

```
X2:  SELECT  XMLEMENT(NAME "pracownik",
                    XMLFOREST(
                        E.Nazwisko AS "naz",
                        E.Imię     AS "im",
                        E.Pensja   AS "pen" ) )
```

```
FROM   PRACOWNIK AS E
WHERE  E.PESEL = "65010912345" ;
```

Wynikiem zapytania X2 dla danych z rysunku 5.6 jest:

```
<pracownik><naz>Szewczyk</naz><im>Jan</im><pen>3000</pen></pracownik>
```

Założmy, że chcesz pobrać dane obejmujące nazwisko, imię i pensję pracowników z działu 4. oraz sformatować je do postaci dokumentu XML z korzeniem `pracdział4`. Można w tym celu napisać zapytanie SQL/XML X3:

```
X3:  SELECT  XMLROOT (
          XMLELEMENT (NAME "pracdział4",
            XMLAGG (
              XMLELEMENT (NAME "prac"
                XMLFOREST (nazwisko, imię, pensja)
                ORDER BY Nazwisko ) ) )
        FROM   PRACOWNIK
        WHERE  NRDZIAŁU = 4 ;
```

Funkcja `XMLROOT` tworzy jeden korzeń, dlatego dane w formacie XML tworzą poprawnie uformowany dokument (drzewo z jednym korzeniem). Wynikiem zapytania X3 dla danych z rysunku 5.6 jest:

```
<pracdział4>
<prac><nazwisko>Janiszewski</nazwisko><imię>Albert</imię><pensja>2500
</pensja></prac>
<prac><nazwisko>Wojtczak</nazwisko><imię>Janina
</imię><pensja>4300</pensja></prac>
<prac><nazwisko>Zalewska</nazwisko><imię>Alicja</imię><pensja>2500
</pensja></prac>
</pracdział4>
```

Te przykłady dają posmak tego, jak rozbudowano standard SQL, aby umożliwić użytkownikom formatowanie wyników zapytania do postaci danych w formacie XML.

## 13.8. Podsumowanie

Niniejszy rozdział zawiera przegląd standardów służących do reprezentowania i wymiany danych za pośrednictwem Internetu. Na początku omówione zostały różnice pomiędzy danymi strukturalnymi, półstrukturalnymi i niestrukuralnymi. Dane strukturalne są przechowywane w tradycyjnych bazach. Dane półstrukturalne łączą nazwy typów danych z wartościami, przy czym nie muszą być zgodne ze stałą, predefiniowaną strukturą. Dane niestrukuralne to informacje wyświetlane w internecie i zapisywane za pomocą języka HTML; nie obejmują informacji o typach elementów danych. Opisany został też standard XML i jego drzewiasty (hierarchiczny) model danych, a także dokumenty XML i języki opisujące strukturę tych dokumentów, czyli XML DTD (definicja typu dokumentu) i schematy XML. Następnie omówiono różne sposoby przechowywania dokumentów XML od (macierzystego) formatu tekstowego, przez formę skompresowaną, po relacyjne i inne bazy danych. Zaprezentowane zostały także języki zapytań XML XPath i XQuery; przedstawiono również

problemy związane z definiowaniem odwzorowań tradycyjnych baz danych na dokumenty XML. Na koniec opisano standard SQL/XML, wzbogacający XML o dodatkowe mechanizmy formatowania wyników zapytań SQL jako danych w formacie XML.

## Pytania powtórkowe

- 13.1. Jakie są różnice pomiędzy danymi strukturalnymi, półstrukturalnymi i niestructuralnymi?
- 13.2. Do której kategorii z pytania 13.1 zaliczają się dokumenty XML? Co rozumiemy przez pojęcie dokumentu samoopisowego?
- 13.3. Jakie są różnice w użyciu znaczników pomiędzy HTML a XML?
- 13.4. Jaka jest różnica pomiędzy dokumentami danocentrycznymi i dokumentocentrycznymi?
- 13.5. Jaka jest różnica pomiędzy atrybutami a elementami XML? Wymień kilka spośród atrybutów stosowanych przy definiowaniu schematów XML.
- 13.6. Jaka jest różnica pomiędzy XML DTD a schematami XML?

## Ćwiczenia

- 13.7. Stwórz dokument XML odpowiadający strukturze relacyjnej bazy danych z rysunku 5.6 i spełniający warunki schematu XML z rysunku 13.5.
- 13.8. Stwórz schemat XML i dokument DTD odpowiadający hierarchiom pokazanym na rysunkach 13.14 i 13.15©.
- 13.9. Rozważ schemat relacyjnej bazy danych BIBLIOTEKA z rysunku 6.6. Stwórz dokument schematu XML odpowiadający strukturze tej bazy danych.
- 13.10. Określ następujące perspektywy jako zapytania XQuery dla schematu XML FIRMA pokazanego na rysunku 13.5:
  - a) Perspektywa zawierająca nazwę działu, nazwisko kierownika i pensję kierownika dla każdego działu firmy.
  - b) Perspektywa zawierająca nazwisko pracownika, nazwisko przełożonego oraz pensję pracownika dla każdego pracownika działu badawczego.
  - c) Perspektywa zawierająca nazwę projektu, nazwę działu kontrolującego ten projekt, liczbę pracowników i liczbę godzin przeznaczanych przez nich tygodniowo na udział w każdym projekcie.
  - d) Perspektywa zawierająca nazwę projektu, nazwę działu kontrolującego projekt, liczbę pracowników i liczbę godzin przeznaczanych przez nich tygodniowo na udział w każdym projekcie, nad którym pracują co najmniej dwie osoby.

## Wybrane publikacje

Istnieje tak wiele artykułów i książek dotyczących różnych aspektów standardu XML, że niemożliwe jest przedstawienie tu jakiegokolwiek listy. Dlatego wymienimy tylko jedną książkę zawierającą nie tylko omówienia różnych kwestii związanych z XML, ale również obszerne odwołania do innych publikacji. Autorami tej pozycji są Chaudhri, Rashid i Zicari (2003).

# VI

---

## Teoria projektowania baz danych i normalizacja





# Podstawy zależności funkcyjnych i normalizacji w relacyjnych bazach danych

W rozdziałach od 5. do 8. przedstawiono różne aspekty modelu relacyjnego i związanych z nim języków. Każdy *schemat relacji* (ang. *relation schema*) składa się z pewnej liczby atrybutów, zaś *schemat relacyjnej bazy danych* (ang. *relational database schema*) składa się z pewnej liczby schematów relacji. Jak dotąd zakładaliśmy, że atrybuty są pogrupowane w taki sposób, że tworzą schemat relacji, a grupowanie odbywa się na podstawie doświadczeń projektantów baz danych lub odwzorowania koncepcyjnego modelu danych (takiego jak ER, rozszerzony ER (EER) lub jeszcze inny) na projekt schematu bazy danych. Takie modele umożliwiają projektantowi identyfikowanie typów encji oraz typów związków i atrybutów, co umożliwia naturalne i logiczne pogrupowanie atrybutów w formie relacji, jeżeli zastosuje się procedury dokonywania odwzorowań przedstawione w rozdziale 9. Jednak wciąż jest nam potrzebny pewien formalny sposób określenia, dlaczego jedno pogrupowanie atrybutów w postaci schematu relacji jest lepsze od innego. Jak dotąd w przedstawionej dyskusji poświęconej projektom baz danych (rozdziały 3., 4. i 9.) nie opracowaliśmy żadnej metody weryfikacji poprawności i jakości projektu poza zdaniem się na intuicję projektanta. W niniejszym rozdziale zostaną omówione pewne zagadnienia z zakresu teorii dokonywania oceny schematów relacji pod względem ich jakości, tzn. formalnego określania, dlaczego jedno pogrupowanie atrybutów do postaci schematu relacji jest lepsze od innych.

*Poprawność* schematów relacji można rozpatrywać na dwóch płaszczyznach. Pierwszą z nich jest **poziom logiczny** (ang. *logical level*), czyli **koncepcyjny** (ang. *conceptual*) — sposób, w jaki użytkownicy interpretują schematy relacji oraz znaczenie ich atrybutów. Posiadanie dobrych schematów relacji na tym poziomie pozwala użytkownikom na zrozumienie znaczenia danych występujących w relacjach, dzięki czemu mogą poprawnie formułować swoje zapytania. Drugim jest **poziom realizacji** (ang. *implementation level*), czyli **składowania** (ang. *storage*) — sposób przechowywania i aktualizowania krotek relacji głównej. Poziom ten ma zastosowanie jedynie w przypadku schematów relacji głównych, które są fizycznie przechowywane w formie plików, natomiast na poziomie logicznym jesteśmy zainteresowani zarówno schematami relacji głównych, jak i perspektyw (relacji wirtualnych). Teoria projektowania relacyjnych baz danych przedstawiona w niniejszym rozdziale ma zastosowanie przede wszystkim względem *relacji głównych* (ang. *base relations*), aczkolwiek niektóre kryteria poprawności sprawdzają się również w przypadku perspektyw, co omówiono w podrozdziale 14.1.

Podobnie jak w przypadku wielu innych problemów projektowych projektowanie baz danych może się odbywać według dwóch metod postępowania: wstępującej i zstępującej. **Wstępująca metodologia projektowania** (ang. *bottom-up design methodology*), określana również mianem *projektowania przez syntezę* (ang. *design by synthesis*), traktuje jako punkt wyjścia do dalszych działań podstawowe związki występujące między pojedynczymi atrybutami i używa ich w celu konstruowania schematów relacji. Podejście to nie jest w praktyce zbyt często wykorzystywane<sup>1</sup>, ponieważ występuje tu problem związany z koniecznością określenia na samym początku dużej liczby związków binarnych występujących między atrybutami. W zasadzie uwzględnienie związków binarnych między wszystkimi parami atrybutów jest prawie niemożliwe. Z kolei **zstępująca metodologia projektowania** (ang. *top-down design methodology*), określana również mianem *projektowania przez analizę* (ang. *design by analysis*), jako punkt wyjścia przyjmuje pewną liczbę pogrupowanych atrybutów w formie relacji, które w naturalny sposób istnieją obok siebie, na przykład na fakturze, formularzu lub w raporcie. Następnie relacje są badane pojedynczo i zbiorczo, co prowadzi do dalszej dekompozycji projektu, do momentu, aż zostaną spełnione określone wymagania. Teoria opisana w tym rozdziale dotyczy przede wszystkim podejścia zstępującego, dlatego lepiej nadaje się do projektowania baz w wyniku analizy i dekompozycji zbiorów atrybutów występujących w praktyce razem w plikach, raportach i formularzach.

Projektowanie relacyjnej bazy danych ostatecznie prowadzi do powstania zbioru relacji. Pośrednimi celami projektowania są *zachowanie informacji* i *minimalizowanie nadmiarowości*. Informacje bardzo trudno jest tu opisać ilościowo, dlatego ich zachowywanie jest analizowane w kategoriach uwzględnienia wszystkich ich elementów (w tym typów atrybutów, typów encji i typów związków, a także relacji generalizacji i specjalizacji), które zostały opisane w modelu takim jak EER. Dlatego w projekcie relacyjnym trzeba zachować wszystkie elementy, które zostały pierwotnie ujęte w projekcie koncepcyjnym, a później w projekcie logicznym (po odwzorowaniu z projektu koncepcyjnego). Minimalizowanie nadmiarowości wymaga unikania wielokrotnego zapisywania tych samych informacji i ograniczenia potrzeby wielokrotnego ich aktualizowania w celu zachowania spójności licznych kopii tych samych danych w reakcji na zdarzenia z rzeczywistego świata.

Na początku, w podrozdziale 14.1, zostaną nieformalnie omówione pewne kryteria oceny poprawnych i niepoprawnych schematów relacji. Następnie, w podrozdziale 14.2, zostanie zdefiniowane pojęcie *zależności funkcyjnej* (ang. *functional dependency*), formalnego ograniczenia występującego między atrybutami, które stanowi podstawowe narzędzie dokonywania formalnej oceny poprawności grupowania atrybutów w postaci schematów relacji. W podrozdziale 14.3 omówimy postaci normalne i proces normalizacji z wykorzystaniem zależności funkcyjnych. Zdefiniowane zostaną kolejne postaci normalne pozwalające uzyskać zgodność z zestawem ograniczeń zapisanych za pomocą kluczy głównych i zależności funkcyjnych. Proces *normalizacji* (ang. *normalization*) polega na analizowaniu relacji w celu zapewnienia, aby spełniały wymagania coraz bardziej rygorystycznych postaci normalnych, i (w razie potrzeby) dekompozycji tych relacji. W podrozdziale 14.4 zostaną omówione ogólniejsze definicje postaci normalnych, które można bezpośrednio stosować w przypadku dowolnego projektu i które nie wyma-

---

<sup>1</sup> Wyjątkiem jest model noszący nazwę binarnego modelu relacyjnego. Jego reprezentatywnym przykładem jest metodologia NIAM (Verheijen i VanBekum, 1982).

gają przeprowadzania stopniowej analizy oraz normalizacji. W podrozdziałach od 14.5 do 14.7 opiszemy dodatkowe postacie normalne aż do piątej. W podrozdziale 14.6 przedstawimy zależności wielowartościowe, a w podrozdziale 14.7 — zależności łączeniowe. Podrozdział 14.8 zawiera podsumowanie rozdziału.

W rozdziale 15. będzie kontynuowana tematyka teorii związanej z projektowaniem poprawnych schematów relacyjnych. Zostaną określone pożądane właściwości dekompozycji relacyjnej — właściwośćłączenia nieaddytywnego (bezstratnego) i właściwość zachowania zależności funkcyjnych. Zostanie tu również określony ogólny algorytm sprawdzający, czy dekompozycja posiada właściwośćłączenia *nieaddytywnego* (algorytm 15.3). Dalej omówimy właściwości zależności funkcyjnych i kwestię minimalnego pokrycia zależności. Opiszemy projektowanie baz danych metodą wstępującą, obejmujące zestaw algorytmów do projektowania relacji w odpowiedniej postaci normalnej. Te algorytmy przyjmują określony zestaw zależności funkcyjnych i generują projekt relacyjny w docelowej postaci normalnej przy zachowaniu pożądanych właściwości. W rozdziale 15. zostaną także zdefiniowane dodatkowe typy zależności oraz zaawansowane postacie normalne, które w jeszcze większym stopniu gwarantują *poprawność* schematów relacji.

Jeżeli zajęcia nie uwzględniają treści rozdziału 15., zaleca się krótkie wprowadzenie do pożądanych właściwości dekompozycji z podrozdziału 15.2 oraz omówienie znaczenia właściwościłączenia nieaddytywnego w trakcie dekompozycji.

## 14.1. Nieformalne wskazówki dotyczące projektowania schematów relacji

Przed omówieniem formalnej teorii projektowania relacyjnych baz danych przedstawimy cztery *nieformalne wskaźniki*, które mogą posłużyć jako *miary jakości* projektu schematu relacji:

- upewnienie się, że semantyka atrybutów w schemacie jest jasna;
- eliminacja nadmiarowych informacji w krotkach;
- eliminacja wartości null w krotkach;
- uniemożliwienie generowania fałszywych krotek.

Powyższe wskaźniki, jak się niebawem okaże, nie zawsze są od siebie niezależne.

### 14.1.1. Wymuszanie jednoznacznej semantyki atrybutów relacji

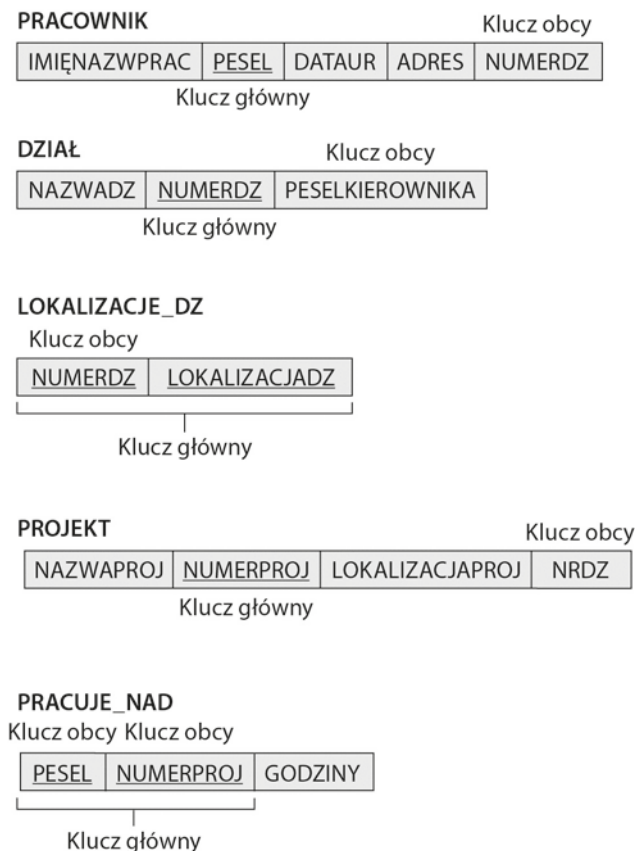
Kiedy grupujemy atrybuty w celu utworzenia schematu relacji, zakładamy, że atrybuty należące do jednej relacji posiadają określone znaczenie związane z rzeczywistością oraz powiązaną z nimi poprawną interpretację. **Semantyka** relacji to znaczenie wynikające z interpretacji wartości atrybutów przechowywanych w krotce relacji. W rozdziale 5. omówiono sposób interpretowania każdej relacji jako zbioru faktów lub zdań. Jeżeli projekt koncepcyjny (opisany w rozdziałach 3. i 4.) zostanie wykonany starannie, a po nim dokona się systematycznego odwzorowania na relacje (patrz rozdział 9.), projekt wynikowy będzie niemal na pewno zrozumiały.

Ogólnie rzecz biorąc, im łatwiej jest objaśnić znaczenie relacji (czyli to, co dokładnie ona oznacza), tym lepszy jest projekt jej schematu. W celu zilustrowania tej tezy weźmy pod uwagę rysunek 14.1, na którym przedstawiono uproszczoną wersję schematu relacyjnej bazy danych FIRMA z rysunku 5.5, oraz rysunek 14.2, na którym przedstawiono przykład stanów relacji tego schematu wypełnionego danymi. Znaczenie relacji PRACOWNIK jest oczywiste: każda krotka reprezentuje pracownika i określono w niej wartości dla imienia i nazwiska (IMIĘNAZWPRAC), numeru PESEL (PESEL), daty urodzenia (DATAUR), adresu (ADRES) oraz numeru działu, w którym dany pracownik pracuje (NUMERDZ). Atrybut NUMERDZ jest kluczem obcym, który reprezentuje *związek niejawny* (ang. *implicit relationship*) między relacjami PRACOWNIK a DZIAŁ. Semantyka schematów DZIAŁ i PROJEKT również jest oczywista: każda krotka relacji DZIAŁ reprezentuje encję działu, zaś każda krotka relacji PROJEKT — encję projektu. Atrybut PESELKIEROWNIKA relacji DZIAŁ wiąże dział z pracownikiem, który jest jego kierownikiem, zaś atrybut NRDZ relacji PROJEKT wiąże projekt z jego działem nadzorczym. W obu przypadkach mamy do czynienia z atrybutami klucza obcego. Łatwość, z jaką można wyjaśnić znaczenie atrybutów relacji, stanowi *nieformalny wskaźnik* poprawności projektu relacji.

Semantyka pozostałych dwóch relacji schematu przedstawionego na rysunku 14.1 jest nieco bardziej skomplikowana. Każda krotka relacji LOKALIZACJE\_DZ określa numer działu (NUMERDZ) oraz *jedną* z lokalizacji działu (LOKALIZACJADZ). Każda krotka relacji PRACUJE\_NAD określa numer PESEL pracownika (PESEL), numer *jednego* z projektów, w którym bierze udział pracownik (NUMERPROJ), oraz liczbę godzin na tydzień, które pracownik poświęca projektowi (GODZINY). Jednak oba schematy posiadają dobrze zdefiniowaną i jednoznaczną interpretację. Schemat LOKALIZACJE\_DZ reprezentuje wielowartościowy atrybut relacji DZIAŁ, zaś schemat PRACUJE\_NAD reprezentuje związek wiele do wielu między relacjami PRACOWNIK a PROJEKT. Stąd wszystkie schematy relacji z rysunku 14.1 można uważać za łatwe do objaśnienia, a przez to za poprawne z punktu widzenia zrozumiałej semantyki. Możemy więc sformułować poniższą nieformalną wskazówkę projektową.

**Wskazówka 1.** Schemat relacji należy zaprojektować tak, aby można było w prosty sposób wyjaśnić jego znaczenie. Nie należy łączyć w ramach pojedynczej relacji atrybutów pochodzących z różnych typów encji oraz typów związków. Intuicyjnie można stwierdzić, że jeżeli schemat relacji odpowiada jednemu typowi encji lub jednemu typowi związkowi, wyjaśnienie jego znaczenia jest bardzo proste. W przeciwnym wypadku, jeżeli relacja odpowiada połączeniu wielu encji i związków, pojawiają się pewne niejednoznaczności semantyczne i znaczenia relacji nie da się wyjaśnić w prosty sposób.

**Przykłady naruszenia wskazówki 1.** Schematy relacji z rysunków 14.3(a) oraz 14.3(b) również posiadają zrozumiałe semantyki. Czytelnik powinien na razie zignorować strzałki występujące pod relacjami — służą one zilustrowaniu notacji zależności funkcyjnych, co omówiono w podrozdziale 14.2. Krotka schematu relacji PRAC\_DZ z rysunku 14.3(a) reprezentuje pojedynczego pracownika, ale zawiera informacje dodatkowe — nazwę działu (NAZWADZ), w którym on pracuje, oraz numer PESEL kierownika działu (PESELKIEROWNIKA). W przypadku relacji PRAC\_PROJ z rysunku 14.3(b) każda krotka wiąże ze sobą pracownika oraz projekt, ale zawiera również imię i nazwisko pracownika (IMIĘNAZWPRAC), nazwę projektu (NAZWAPROJ) oraz lokalizację projektu (LOKALIZACJAPROJ). Choć z logicznego punktu widzenia relacjom tym nie można niczego zarzucić, należy je traktować jako źle zaprojektowane, gdyż naruszają wytyczne wskazówki 1., łącząc atrybuty pochodzące z odrębnych encji. Relacja PRAC\_DZ łączy atrybuty pracowników i działów, zaś relacja PRAC\_PROJ



RYSUNEK 14.1. Uproszczona wersja schematu relacyjnej bazy danych FIRMA

— pracowników, projektów i relacji PRACUJE\_NAD. Dlatego relacje te nie wypadają dobrze w kontekście opisanej wcześniej miary jakości projektu. Mogą one być wykorzystywane jako perspektywy, ale powodują problemy w przypadku użycia jako relacje główne, co zostanie omówione w kolejnym podrozdziale.

### 14.1.2. Nadmiarowe informacje w krotkach oraz anomalie aktualizacji

Jednym z celów projektu schematu jest zminimalizowanie przestrzeni pamięciowej zajmowanej przez relacje główne (a stąd przez odpowiednie pliki). Grupowanie atrybutów w schematy relacji ma ogromny wpływ na zajmowaną przestrzeń. Przykładowo, wystarczy porównać przestrzeń zajmowaną przez dwie relacje główne PRACOWNIK oraz DZIAŁ z rysunku 14.2 z przestrzenią zajmowaną przez relację główną PRAC\_DZ z rysunku 14.4, która jest wynikiem zastosowania operacji złączenia naturalnego (NATURAL JOIN) względem relacji PRACOWNIK i DZIAŁ. W przypadku relacji PRAC\_DZ wartości atrybutów związanych z określonym działem (NUMERDZ, NAZWADZ, PESELKIEROWNIKA) są powtarzane dla *każdego* pracownika, który pracuje

PRACOWNIK

IMIĘNAZWPRAC	PESEL	DATAUR	ADRES	NUMERDZ
Szewczyk, Jan	65010912345	1965-01-09	ul. Kręta 4/3, Szczecin	5
Wieszczyski, Franciszek	55120834598	1955-12-08	os. T. Kościuszki 4a/11, Szczecin	5
Zalewska, Alicja	68011932514	1968-01-19	os. Piastowskie 2/23, Swarzędz	4
Wojtczak, Janina	41062013258	1941-06-20	ul. Kwiatowa 78, Mosina	4
Napierski, Robert	62091502054	1962-09-15	ul. Podgórna 7, Police	5
Englert, Joanna	72073110039	1972-07-31	ul. Wielka 40, Szczecin	5
Janiszewski, Albert	69032923149	1969-03-29	os. Jana Pawła II 13/4, Szczecin	4
Bąk, Józef	37111045873	1937-11-10	os. Centrum 45, Szczecin	1

DZIAŁ

NAZWADZ	NUMERDZ	PESELKIEROWNIKA
Badawczy	5	33344445555
Administracja	4	98765432109
Dyrekcja	1	88886665555

LOKALIZACJE\_DZ

NUMERDZ	LOKALIZACJADZ
1	Warszawa
4	Gliwice
5	Kielce
5	Kraków
5	Warszawa

PRACUJE\_NAD

PESEL	NUMERPROJ	GODZINY
12345678901	1	32,5
12345678901	2	7,5
66688844444	3	40,0
45345345345	1	20,0
45345345345	2	20,0
33344445555	2	10,0
33344445555	3	10,0
33344445555	10	10,0
33344445555	20	10,0
99988877777	30	30,0
99988877777	10	10,0
98798798798	10	35,0
98798798798	30	5,0
98765432109	30	20,0
98765432109	20	15,0
88886665555	20	Null

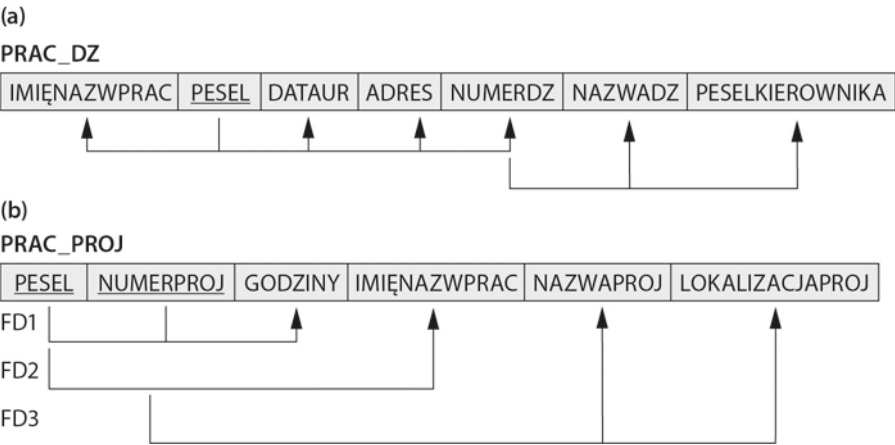
PROJEKT

NAZWAPROJ	NUMERPROJ	LOKALIZACJAPROJ	NRDZ
ProduktX	1	Kielce	5
ProduktY	2	Kraków	5
ProduktZ	3	Warszawa	5
Komputeryzacja	10	Gliwice	4
Reorganizacja	20	Warszawa	1
Nowe zyski	30	Gliwice	4

RYСУNEK 14.2. Przykład stanu bazy danych dla schematu relacyjnej bazy danych z rysunku 14.1

w danym dziale. Z kolei informacje o każdym dziale występują tylko raz w relacji DZIAŁ z rysunku 14.2. W relacji PRACOWNIK dla każdego pracownika pracującego w danym dziale powtarza się wyłącznie numer tego działu (NUMERDZ), będący tu kluczem obcym. Podobne spostrzeżenia można poczynić w przypadku relacji PRAC\_PROJ (rysunek 14.4), która uzupełnia relację PRACUJE\_NAD o dodatkowe atrybuty pochodzące z relacji PRACOWNIK i PROJEKT.





RYSUNEK 14.3. Dwa schematy relacji, których dotyczy problem anomalii aktualizacji

							Nadmiarowość	
PRAC_DZ								
IMIĘNAZWPRAC	PESEL	DATAUR	ADRES	NUMERDZ	NAZWADZ	PESELKIEROWNIKA		
Nowak, Jan	12345678901	1965-01-09	ul. Nowa 11, 00-900 Warszawa	5				
Kowalski, Fryderyk	33344445555	1955-12-08	Al. Niepodległości 23, 44-100 Gliwice	5	Badawczy	33344445555		
Zielińska, Alicja	99988887777	1968-07-19	Pl. Wolności, 25-600 Kielce	4	Administracja	98765432109		
Owsiak, Maria	98765432109	1941-06-20	ul. Stara 22, 09-000 Warszawa	4	Administracja	98765432109		
Głębocki, Adam	66688844444	1962-09-15	ul. Długa 34A, 70-300 Łomża	5	Badawczy	33344445555		
Polakowski, Maciej	45345345345	1972-07-31	ul. Krótka 1, 50-550 Szczecin	5	Badawczy	33344445555		
Kuc, Waldemar	98798798798	1969-03-29	Pl. Powstańców 10B, 00-111 Warszawa	4	Administracja	98765432109		
Szymkowiak, Krystyna	88886665555	1937-11-10	ul. Szeroka 8, 26-100 Starachowice	1	Dyrekcja	88886665555		

						Nadmiarowość		Nadmiarowość	
PRAC_PROJ									
PESEL	NUMERPROJ	GODZINY	IMIĘNAZWPRAC	NAZWAPROJ	LOKALIZACJAPROJ				
12345678901	1	32,5	Nowak, Jan	ProduktX	Kielce				
12345678901	2	7,5	Nowak, Jan	ProduktY	Kraków				
66688844444	3	40,0	Głębocki, Adam	ProduktZ	Warszawa				
45345345345	1	20,0	Polakowski, Maciej	ProduktX	Kielce				
45345345345	2	20,0	Polakowski, Maciej	ProduktY	Kraków				
33344445555	2	10,0	Kowalski, Fryderyk	ProduktY	Kraków				
33344445555	3	10,0	Kowalski, Fryderyk	ProduktZ	Warszawa				
33344445555	10	10,0	Kowalski, Fryderyk	Komputeryzacja	Gliwice				
33344445555	20	10,0	Kowalski, Fryderyk	Reorganizacja	Warszawa				
99988887777	30	30,0	Zielińska, Alicja	Nowe zyski	Gliwice				
99988887777	10	10,0	Zielińska, Alicja	Komputeryzacja	Gliwice				
98798798798	10	35,0	Kuc, Waldemar	Komputeryzacja	Gliwice				
98798798798	30	5,0	Kuc, Waldemar	Nowe zyski	Gliwice				
98765432109	30	20,0	Owsiak, Maria	Nowe zyski	Gliwice				
98765432109	20	15,0	Owsiak, Maria	Reorganizacja	Warszawa				
88886665555	20	Null	Szymkowiak, Krystyna	Reorganizacja	Warszawa				

RYSUNEK 14.4. Przykład stanów relacji PRAC\_DZ oraz PRAC\_PROJ wynikających z zastosowania operacji złączenia naturalnego na relacjach z rysunku 14.2. Mogą one być przechowywane jako relacje główne ze względów wydajnościowych



Kolejnym poważnym problemem związanym z użyciem relacji z rysunku 14.4 jako relacji głównych są **anomalie aktualizacji** (ang. *update anomalies*). Można je podzielić na anomalie wstawiania, anomalie usuwania oraz anomalie modyfikowania<sup>2</sup>.

**Anomalie wstawiania.** Anomalie wstawiania można podzielić na dwa rodzaje, które ilustrują poniższe przykłady oparte na relacji PRAC\_DZ.

- W celu wstawienia krotki nowego pracownika do relacji PRAC\_DZ, musimy uwzględnić albo wartości atrybutów działu, w którym on pracuje, albo wartości null (jeżeli pracownik tymczasowo nie pracuje w żadnym z działów). Przykładowo, w celu wstawienia nowej krotki dla pracownika pracującego w dziale o numerze 5 musimy wstawić poprawne wartości atrybutów działu 5., tak aby były one *spójne* (ang. *consistent*) z wartościami dla działu 5. znajdującymi się w innych krotkach relacji PRAC\_DZ. W projekcie z rysunku 14.2 nie trzeba martwić się o problem ze spójnością, ponieważ w krotce pracownika wprowadzany jest jedynie numer działu. Wartości wszystkich pozostałych atrybutów działu 5. są zapisywane w bazie tylko raz — jako pojedyncza krotka w relacji DZIAŁ.
- Wstawienie do relacji PRAC\_DZ nowego działu, który nie posiada żadnych pracowników, jest trudne. Jedynym sposobem zapewnienia tego jest wstawienie wartości null dla atrybutów pracownika. Narusza to integralność encji w relacji PRAC\_DZ, ponieważ klucz główny PESEL nie może mieć wartości null. Ponadto w momencie przypisania do takiego działu pierwszego pracownika krotka zawierająca wartości null nie jest już dłużej potrzebna. Problem ten nie występuje w projekcie z rysunku 14.2, ponieważ informacje o dziale są wstawiane do relacji DZIAŁ bez względu na to, czy pracuje w nim jakiś pracownik, i w momencie przypisania do takiego działu pracownika odpowiednia krotka zostaje wstawiona do relacji PRACOWNIK.

**Anomalie usuwania.** Problem anomalii usuwania ma związek z drugim rodzajem anomalii wstawiania, który omówiono powyżej. Jeżeli z relacji PRAC\_DZ usuniemy krotkę, która reprezentuje ostatniego pracownika pracującego w danym dziale, informacje dotyczące tego działu zostaną nieodwracalnie usunięte z bazy danych. Problem ten nie występuje w bazie danych z rysunku 14.2, ponieważ krotki relacji DZIAŁ są przechowywane oddzielnie.

**Anomalie modyfikowania.** W relacji PRAC\_DZ jeżeli zmienimy wartość jednego z atrybutów określonego działu, na przykład kierownika działu o numerze 5, to musimy zaktualizować krotki *wszystkich* pracowników pracujących w tym dziale. W przeciwnym razie baza danych znajdzie się w niespójnym stanie. Jeżeli nie zaktualizuje się wartości niektórych krotek, ten sam dział będzie posiadał dwie różne wartości określające kierownika w różnych krotkach pracowników, co bez wątpienia jest błędem<sup>3</sup>.

Łatwo dostrzec, że te trzy anomalie są niepożądane i utrudniają zachowanie spójności danych, a także wymagają zbędnych aktualizacji, których można uniknąć. Można więc sformułować poniższą wskazówkę.

---

<sup>2</sup> Anomalie te zostały zidentyfikowane przez Codda (1972a) w celu uzasadnienia konieczności normalizacji relacji, co zostanie omówione w podrozdziale 15.3.

<sup>3</sup> Nie jest to problem równie poważny jak w przypadku wcześniej opisanych sytuacji, ponieważ wszystkie krotki można zaktualizować za pomocą pojedynczej instrukcji SQL.

**Wskazówka 2.** Schematy relacji głównych należy projektować tak, aby w relacjach nie występowały żadne anomalie wstawiania, usuwania i modyfikowania. W razie występowania pewnych anomalii<sup>4</sup> należy je dokładnie określić i zapewnić, aby programy dokonujące aktualizacji w bazie danych działały poprawnie.

Wskazówka 2. jest zgodna ze wskazówką 1. i w pewnym sensie stanowi jej przeformułowanie. Jednocześnie widać potrzebę stosowania bardziej formalnego podejścia do kwestii dokonywania oceny tego, czy projekt spełnia zalecenia przedstawionych wskazówek. W podrozdziałach od 14.2 do 14.4 zostaną przedstawione takie formalne pojęcia. Istotną sprawą jest zauważenie, że przedstawione wskazówki niekiedy *należy naruszyć* w celu *zwiększenia wydajności* pewnych zapytań. Jeśli relacja PRAC\_DZ jest używana jako relacja składowana (*perspektywa zmaterializowana*) obok relacji głównych PRACOWNIK i DZIAŁ, należy zauważyć i uwzględnić anomalie w relacji PRAC\_DZ (na przykład używając wyzwalaczy lub procedur składowanych, które dokonują automatycznych aktualizacji), tak aby dokonywanie aktualizacji relacji głównej nie spowodowało wystąpienia niespójności danych. Ogólnie rzecz biorąc, zaleca się używanie relacji głównych pozbawionych anomalii i określanie perspektyw, które zawierają złączenia służące do umieszczania razem atrybutów, do których często występują odwołania w ważnych zapytaniach.

### 14.1.3. Wartości null w krotkach

W przypadku niektórych projektów schematów można pogrupować wiele atrybutów razem, tworząc relację „bogatą”. Jeżeli wiele z tych atrybutów nie ma zastosowania w przypadku wszystkich krotek relacji, otrzymujemy wiele wartości null w tych krotkach. Może to powodować marnowanie przestrzeni oraz prowadzić do problemów ze zrozumieniem znaczenia atrybutów oraz z określaniem operacji złączenia na poziomie logicznym<sup>5</sup>. Kolejnym problemem związanym z wartościami null jest sposób ich uwzględnienia w przypadku operacji agregujących, takich jak COUNT lub SUM. Operacje SELECT i JOIN obejmują porównania; jeśli w danych występują wartości null, wyniki mogą być nieprzewidywalne<sup>6</sup>. Ponadto, wartości null mogą posiadać wiele interpretacji, na przykład:

- atrybut *nie ma zastosowania* w przypadku danej krotki (np. stan\_wizy może nie dotyczyć studentów z Polski);
- wartość atrybutu w przypadku danej krotki jest *nieznana* (np. data\_ur pracownika może być nieznana);
- wartość jest *znana, ale nieobecna*, to znaczy nie została jeszcze zarejestrowana (np. numer\_domowy pracownika może istnieć, ale nie jest jeszcze określony lub zarejestrowany).

---

<sup>4</sup> Inne kwestie związane z aplikacją mogą sprawić, że niektórych anomalii nie da się uniknąć. Przykładowo, relacja PRAC\_DZ może odpowiadać często potrzebnemu zapytaniu lub raportowi.

<sup>5</sup> Wynika to z faktu, że złączenia wewnętrzne i zewnętrzne dają inne wyniki w przypadku, gdy w złączeniach uczestniczą wartości null. Stąd użytkownicy muszą pamiętać o różnym znaczeniu różnych typów złączeń. Choć można tego wymagać od doświadczonych użytkowników, w przypadku pozostałych może to stanowić spory problem.

<sup>6</sup> W punkcie 5.5.1 opisano porównania z dozwolonymi wartościami pustymi, gdzie wynikiem (w logice trójwartościowej) może być TRUE, FALSE lub UNKNOWN.

Posiadanie takiej samej reprezentacji dla wszystkich wartości null powoduje utratę informacji o różnicach w ich znaczeniu. Stąd też można sformułować kolejną wskazówkę.

**Wskazówka 3.** W jak największym stopniu należy unikać umieszczania w relacji głównej atrybutów, które często mogą przyjmować wartość null. Jeżeli nie da się uniknąć występowania wartości null, należy zapewnić, aby występowały tylko w przypadkach wyjątkowych, a nie w przypadku większości krotek relacji.

Efektywne używanie dostępnej przestrzeni oraz unikanie złączeń to dwa najważniejsze kryteria, które określają, czy zawrzeć w relacji kolumny mogące posiadać wartości null, czy też określić dla tych kolumn odrębną relację (z odpowiednimi kolumnami klucza). Przykładowo, jeżeli tylko 15 procent pracowników posiada samodzielne biura, trudno uzasadnić zawarcie atrybutu `NUMER_BIURA` w relacji `PRACOWNIK`. Zamiast tego można utworzyć nową relację `PRAC_BIURA(PESELPRAC, NUMER_BIURA)`, która będzie zawierała krotki tylko dla pracowników posiadających samodzielne biura.

### 14.1.4. Generowanie fałszywych krotek

Weźmy pod uwagę schematy relacji `PRAC_LOK` oraz `PRAC_PROJ1` z rysunku 14.5(a), których można użyć zamiast pojedynczej relacji `PRAC_PROJ` z rysunku 14.3(b). Krotka relacji `PRAC_LOK` oznacza, że pracownik, którego imię i nazwisko określa atrybut `IMIENAZWPAC`, pracuje nad *przynajmniej jednym projektem*, którego lokalizacją jest `LOKALIZACJAPROJ`. Krotka relacji `PRAC_PROJ1` oznacza, że pracownik, którego numerem `PESEL` jest `PESEL`, pracuje `GODZINY` godzin tygodniowo nad projektem, którego nazwą, numerem oraz lokalizacją są, odpowiednio, `NAZWAPROJ`, `NUMERPROJ` oraz `LOKALIZACJAPROJ`. Na rysunku 14.5(b) przedstawiono stany relacji `PRAC_LOK` oraz `PRAC_PROJ1` odpowiadające relacji `PRAC_PROJ` z rysunku 14.4, które otrzymano poprzez zastosowanie odpowiednich operacji rzutowania ( $\pi$ ) względem relacji `PRAC_PROJ`.

Załóżmy, że jako relacji bazowych użyliśmy `PRAC_PROJ1` oraz `PRAC_LOK` zamiast `PRAC_PROJ`. W rezultacie otrzymujemy bardzo zły projekt schematu, ponieważ z relacji `PRAC_PROJ1` i `PRAC_LOK` nie da się uzyskać informacji, które początkowo zawarto w relacji `PRAC_PROJ`. Jeżeli spróbujemy wykonać operację złączenia naturalnego (`NATURAL JOIN`) na relacjach `PRAC_PROJ1` i `PRAC_LOK`, w wyniku otrzymamy o wiele więcej krotek niż zawierał ich początkowy zbiór krotek relacji `PRAC_PROJ`. Na rysunku 14.6 przedstawiono wynik zastosowania złączenia względem krotek pracownika o numeru `PESEL` równym 123456789 (w celu zredukowania rozmiaru relacji wynikowej). Dodatkowe krotki, które nie występowały w relacji `PRAC_PROJ`, określa się mianem **fałszywych krotek** (ang. *spurious tuples*), ponieważ reprezentują one fałszywe informacje, które są niepoprawne. Fałszywe krotki oznaczono na rysunku 14.6 znakiem gwiazdki (\*). Zadanie dla Czytelnika polega na uzupełnieniu wyniku złączenia naturalnego całych tabel `PRAC_PROJ1` i `PRAC_LOK` oraz oznaczeniu w tym wyniku fałszywych krotek.

Dekompozycja relacji `PRAC_PROJ` do postaci dwóch relacji `PRAC_LOK` oraz `PRAC_PROJ1` jest niepożądana, ponieważ kiedy z powrotem złączymy je przy użyciu złączenia naturalnego, nie otrzymamy poprawnych danych. Dzieje się tak dlatego, że w opisywanym przypadku `LOKALIZACJAP` jest atrybutem wiążącym relacje `PRAC_LOK` oraz `PRAC_PROJ1`, a atrybut ten nie jest ani kluczem głównym, ani kluczem obcym w relacjach `PRAC_LOK` i `PRAC_PROJ1`. Możemy więc nieformalnie sformułować kolejną wskazówkę projektową.



RYSUNEK 14.5. Szczególnie słaby projekt dla relacji PRAC\_PROJ z rysunku 14.3(b). (a) Dwa schematy relacji PRAC\_LOK oraz PRAC\_PROJ1. (b) Wynik zrzutowania rozszerzenia relacji PRAC\_PROJ z rysunku 14.4 na relacje PRAC\_LOK oraz PRAC\_PROJ1

	PESEL	NUMERPROJ	GODZINY	NAZWAPROJ	LOKALIZACJAP	IMIĘNAZWPRAC
	12345678901	1	32,5	ProduktX	Kielce	Nowak, Jan
*	12345678901	1	32,5	ProduktX	Kielce	Polakowski, Maciej
	12345678901	2	7,5	ProduktY	Kraków	Nowak, Jan
*	12345678901	2	7,5	ProduktY	Kraków	Polakowski, Maciej
*	12345678901	2	7,5	ProduktY	Kraków	Kowalski, Fryderyk
	66688844444	3	40,0	ProduktZ	Warszawa	Głębocki, Adam
*	66688844444	3	40,0	ProduktZ	Warszawa	Kowalski, Fryderyk
*	45345345345	1	20,0	ProduktX	Kielce	Nowak, Jan
	45345345345	1	20,0	ProduktX	Kielce	Polakowski, Maciej
*	45345345345	2	20,0	ProduktY	Kraków	Nowak, Jan
	45345345345	2	20,0	ProduktY	Kraków	Polakowski, Maciej
*	45345345345	2	20,0	ProduktY	Kraków	Kowalski, Fryderyk
*	33344445555	2	10,0	ProduktY	Kraków	Nowak, Jan
*	33344445555	2	10,0	ProduktY	Kraków	Polakowski, Maciej
	33344445555	2	10,0	ProduktY	Kraków	Kowalski, Fryderyk
*	33344445555	3	10,0	ProduktZ	Warszawa	Głębocki, Adam
	33344445555	3	10,0	ProduktZ	Warszawa	Kowalski, Fryderyk
	33344445555	10	10,0	Komputeryzacja	Gliwice	Kowalski, Fryderyk
*	33344445555	20	10,0	Reorganizacja	Warszawa	Głębocki, Adam
	33344445555	20	10,0	Reorganizacja	Warszawa	Kowalski, Fryderyk

RYSUNEK 14.6. Wynik zastosowania złączenia naturalnego względem krotek znajdujących się nad linią przerywaną w relacjach PRAC\_PROJ1 oraz PRAC\_LOK na rysunku 14.5. Wygenerowane fałszywe krotki oznaczono znakiem gwiazdki

**Wskazówka 4.** Schematy relacji należy projektować tak, aby mogły być łączone z użyciem warunków równości atrybutów, które są kluczami głównymi lub kluczami obcymi, w sposób gwarantujący, że nie będą generowane fałszywe krotki. Należy unikać relacji zawierających pasujące atrybuty, które nie stanowią kombinacji (klucz obcy, klucz główny), ponieważ łączenie po takich atrybutach może powodować otrzymywanie fałszywych krotek.

Powyższa nieformalna wskazówka bez wątpienia wymaga formalnego uściślenia. W podrzdziale 15.2 zostanie omówiona nieaddytywna (inaczej — bezstratna) właściwość łączenia, która gwarantuje, że określone łączenia nie powodują powstawania fałszywych krotek.

### 14.1.5. Podsumowanie i omówienie wskazówek projektowych

W punktach od 14.1.1 do 14.1.4 w nieformalny sposób omówiono sytuacje, które mogą prowadzić do problematycznych schematów relacji i zaproponowano nieformalne wskazówki zapewniające tworzenie poprawnych projektów relacyjnych. Wymienione problemy, których występowanie można stwierdzić bez użycia dodatkowych narzędzi analitycznych, to:

- Anomalie powodujące wykonywanie nadmiarowych działań w czasie operacji wstawiania do relacji lub jej modyfikowania, oraz które mogą powodować przypadkową utratę informacji w czasie operacji usuwania z relacji.
- Marnotrawstwo przestrzeni związane z występowaniem wartości null oraz wynikające z tego trudności w wykonywaniu selekcji, operacji agregujących oraz złączeń.
- Generowanie niepoprawnych i fałszywych danych w czasie złączeń relacji głównych na podstawie dopasowywania atrybutów, które nie tworzą prawidłowego związku (klucz obcy, klucz główny).

W pozostałej części niniejszego rozdziału zostaną zaprezentowane formalne pojęcia i zagadnienia teoretyczne, z których można korzystać w celu precyzyjniejszego zdefiniowania warunków *poprawności* i *niepoprawności pojedynczych* schematów relacji. W pierwszej kolejności zostanie omówiona zależność funkcyjna jako narzędzie analityczne. Następnie zostaną opisane trzy postaci normalne oraz postać normalna Boyce'a-Codda (BCNF) dla schematów relacji. Są to uznane i przyjęte standardy jakości w projektowaniu baz relacyjnych. Strategia tworzenia dobrego projektu polega na odpowiedniej dekompozycji źle zaprojektowanej relacji w celu uzyskania wyższych postaci normalnych. Pókrótce przedstawimy tu dodatkowe postacie normalne związane z dodatkowymi zależnościami. W rozdziale 15. szczegółowo omówimy właściwości dekompozycji i zaprezentujemy różnorodne algorytmy powiązane z zależnościami funkcyjnymi, poprawnością dekompozycji i projektowaniem relacji metodą wstępującą z wykorzystaniem zależności funkcyjnych.

## 14.2. Zależności funkcyjne

Do tego miejsca omawialiśmy nieformalne miary jakości projektu bazy danych. Teraz wprowadzimy formalne narzędzie analizy schematów relacyjnych, umożliwiające precyzyjne wykrywanie i opisywanie niektórych z wcześniej wspomnianych problemów. Najważniejszym

pojęciem w teorii projektowania schematów relacji jest zależność funkcyjna. W niniejszym podrozdziale zostanie ono formalnie zdefiniowane, zaś w podrozdziale 14.3 Czytelnik zostanie zapoznany z metodami jego użycia w celu definiowania postaci normalnych dla schematów relacji.

### 14.2.1. Definicja zależności funkcyjnej

Zależność funkcyjna to więzy występujące między dwoma zbiorami atrybutów w bazie danych. Załóżmy, że schemat relacyjnej bazy danych posiada  $n$  atrybutów  $A_1, A_2, \dots, A_n$ . Przyjmijmy, że całą bazę danych opisuje schemat pojedynczej **relacji uniwersalnej**  $R = \{A_1, A_2, \dots, A_n\}$ <sup>7</sup>. Nie określamy, że baza danych będzie przechowywana jako pojedyncza uniwersalna tabela, a jedynie używamy tego pojęcia w celu opracowania formalnej teorii zależności danych<sup>8</sup>.

**Definicja. Zależność funkcyjna** (ang. *functional dependency*), zapisywana jako  $X \rightarrow Y$ , między dwoma zbiorami atrybutów  $X$  i  $Y$ , które są podzbiorami zbioru  $R$ , określa *więzy* (ang. *constraint*) względem możliwych krotek, które mogą tworzyć stan  $r$  relacji  $R$ . Więzy określają, że dla dowolnych dwóch krotek  $t_1$  i  $t_2$  w  $r$ , takich że  $t_1[X] = t_2[X]$ , musi również zachodzić równość  $t_1[Y] = t_2[Y]$ .

Oznacza to, że wartości komponentu  $Y$  krotki w  $r$  zależą od (są *determinowane przez*) wartości komponentu  $X$ . Inaczej mówiąc, wartości komponentu  $X$  krotki jednoznacznie (**funkcyjnie**) *determinują* wartości komponentu  $Y$ . Mówi się również, że istnieje zależność funkcyjna od  $X$  do  $Y$  lub że  $Y$  jest **funkcyjnie zależne** od  $X$ . Skrótem dla zapisu zależności funkcyjnej jest **FD** lub **f.d.** Zbiór atrybutów  $X$  nosi nazwę **lewej strony** zależności funkcyjnej, zaś zbiór  $Y$  to jej **prawa strona**.

Zatem  $X$  funkcyjnie determinuje  $Y$  w schemacie relacji  $R$  wtedy i tylko wtedy, gdy dwie krotki stanu  $r(R)$ , które są zgodne co do swoich wartości  $X$ , zgadzają się co do swoich wartości  $Y$ . Należy zwrócić uwagę na dwa fakty:

- Jeżeli więzy nałożone na relację  $R$  określają, że nie może występować więcej niż jedna krotka o danej wartości  $X$  w dowolnej realizacji relacji  $r(R)$  — to znaczy, kiedy  $X$  jest **kluczem kandydującym** (ang. *candidate key*) relacji  $R$  — implikuje to występowanie zależności funkcyjnej  $X \rightarrow Y$  dla dowolnego podzbioru atrybutów  $Y$  relacji  $R$  (ponieważ więzy klucza implikują, że żadne dwie krotki w przypadku dowolnego dopuszczalnego stanu  $r(R)$  nie będą posiadać takich samych wartości  $X$ ). Jeśli  $X$  jest kluczem kandydującym relacji  $R$ , to  $X \rightarrow R$ .
- Jeżeli  $X \rightarrow Y$  w  $R$ , nie musi to oznaczać, że  $Y \rightarrow X$  w  $R$ .

Zależność funkcyjna jest właściwością **semantyki**, czyli **znaczenia atrybutów**. Projektanci baz danych wykorzystują swoją znajomość semantyki atrybutów relacji  $R$  — to znaczy sposobu ich wzajemnych powiązań — w celu określania zależności funkcyjnych,

<sup>7</sup> Pojęcie relacji uniwersalnej będzie istotne przy omawianiu algorytmów projektowania relacyjnych baz danych w rozdziale 15.

<sup>8</sup> Założenie to implikuje konieczność posiadania przez każdy atrybut w bazie danych odrębnej nazwy. W rozdziale 5. w celu zapewnienia unikatowości atrybutów o takich samych nazwach należących do dwóch różnych relacji ich nazwy były uzupełniane przedrostkiem nazwy relacji.



które powinny być zachowane dla *wszystkich* stanów relacji (rozszerzeń)  $r$  relacji  $R$ . Rozszerzenia relacji  $r(R)$  spełniające więzy zależności funkcyjnych są określane mianem **dopuszczalnych stanów relacji** (ang. *legal relation states*) lub **dopuszczalnych rozszerzeń** (ang. *legal extensions*) relacji  $R$ . Stąd, głównym zastosowaniem zależności funkcyjnych jest dalsze opisanie schematu relacji  $R$  poprzez określenie na jej atrybutach więzów, które muszą być *zawsze* zachowane. Pewne zależności funkcyjne można określać bez odwoływania się do określonych relacji, jako właściwość atrybutów o dobrze znanym znaczeniu. Przykładowo, zależność  $\{\text{WOJEWÓDZTWO}, \text{NUMER\_PRAWA\_JAZDY}\} \rightarrow \text{PESEL}$  powinna być zachowana w przypadku dowolnej osoby dorosłej w Polsce, dlatego powinna obowiązywać *zawsze*, gdy te atrybuty występują w relacji<sup>9</sup>. Istnieje również możliwość, że pewne zależności funkcyjne będą znikać w razie występowania zmian związków. Przykładowo, zależność funkcyjna  $\text{KOD\_POCZTOWY} \rightarrow \text{NUMER\_KIERUNKOWY}$  występowała kiedyś w Stanach Zjednoczonych jako związek między kodami pocztowymi a telefonicznymi numerami kierunkowymi, jednak po znacznym zwiększeniu liczby numerów kierunkowych obecnie już nie obowiązuje.

Weźmy pod uwagę schemat relacji  $\text{PRAC\_PROJ}$  z rysunku 14.3(b). Na podstawie semantyki atrybutów i relacji wiemy, że powinny być zachowane następujące zależności funkcyjne:

- a)  $\text{PESEL} \rightarrow \text{IMIĘNAZWPRAC}$ ;
- b)  $\text{NUMERPROJ} \rightarrow \{\text{NAZWAPROJ}, \text{LOKALIZACJAPROJ}\}$ ;
- c)  $\{\text{PESEL}, \text{NUMERPROJ}\} \rightarrow \text{GODZINY}$ .

Powyższe zależności funkcyjne określają, że (a) wartość numeru PESEL (PESEL) pracownika jednoznacznie identyfikuje jego imię i nazwisko (IMIĘNAZWPRAC), (b) wartość numeru projektu (NUMERPROJ) jednoznacznie identyfikuje nazwę (NAZWAPROJ) oraz lokalizację (LOKALIZACJAPROJ) projektu oraz (c) połączenie wartości PESEL i NUMERPROJ jednoznacznie identyfikuje liczbę godzin (GODZINY), jaką tygodniowo poświęca pracownik na pracę nad projektem. Można również powiedzieć, że wartość IMIĘNAZWPRAC jest funkcyjnie determinowana przez (jest funkcyjnie zależna od) wartości PESEL lub „mając daną wartość PESEL, znamy wartość IMIĘNAZWPRAC” itd.

Zależność funkcyjna jest *właściwością schematu relacji  $R$* , a nie pewnego konkretnego dopuszczalnego stanu  $r$  relacji  $R$ . Stąd zależność funkcyjna *nie może* być automatycznie wywnioskowana na podstawie danego rozszerzenia relacji  $r$ , ale jawnie zdefiniowana przez osobę znającą semantykę atrybutów relacji  $R$ . Przykładowo, na rysunku 14.7 przedstawiono *konkretny stan* schematu relacji UCZY. Choć na pierwszy rzut oka można by uznać, że zachodzi zależność  $\text{PODRĘCZNIK} \rightarrow \text{PRZEDMIOT}$ , nie możemy tego potwierdzić, jeżeli nie będziemy wiedzieć, czy jest tak dla *wszystkich możliwych dopuszczalnych stanów* relacji UCZY. Wystarczy jednak przedstawić *pojedynczy kontrprzykład*, aby wykazać brak zależności funkcyjnej. Przykładowo, ze względu na fakt, że *Nowak* uczy zarówno przedmiotu *Struktury danych*, jak i *Zarządzanie danymi*, możemy wywnioskować, że wartość atrybutu WYKŁADOWCA nie determinuje funkcyjnie wartości atrybutu PRZEDMIOT.

<sup>9</sup> Zauważ, że występują bazy danych (np. u operatorów kart kredytowych lub w wydziałach policji), gdzie ta zależność funkcyjna nie obowiązuje. Powodem są rekordy z fałszywymi danymi, które są wynikiem używania tego samego numeru prawa jazdy przez przynajmniej dwie osoby.



## UCZY

WYKŁADOWCA	PRZEDMIOT	PODRĘCZNIK
Nowak	Struktury danych	Bartram
Nowak	Zarządzanie danymi	Al.-Nour
Jankowski	Kompilatory	Hoffman
Wiśniewski	Struktury danych	Augenthaler

RYSUNEK 14.7. Stan relacji UCZY z *możliwą* zależnością funkcyjną PODRĘCZNIK  $\rightarrow$  PRZEDMIOT. Można jednak wykluczyć zależności WYKŁADOWCA  $\rightarrow$  PRZEDMIOT, PODRĘCZNIK  $\rightarrow$  WYKŁADOWCA i PRZEDMIOT  $\rightarrow$  PODRĘCZNIK

Gdy dana jest wypełniona relacja, to bez znajomości znaczenia atrybutów i relacji między nimi nie można ustalić, które zależności funkcyjne są spełnione, a które nie. Można tylko stwierdzić, że określona zależność funkcyjna *może* występować, jeśli jest spełniona w danym rozszerzeniu. Bez zrozumienia znaczenia istotnych atrybutów nie można zagwarantować jej występowania. Jeśli jednak występują krotki naruszające daną zależność funkcyjną, można z przekonaniem stwierdzić, że *nie jest ona spełniona*. Przyjrzyj się przykładowej relacji z rysunku 14.8. Wymienione tu zależności funkcyjne ( $B \rightarrow C$ ,  $C \rightarrow B$ ,  $\{A, B\} \rightarrow C$ ,  $\{A, B\} \rightarrow D$  i  $\{C, D\} \rightarrow B$ ) *mogą występować*, ponieważ cztery krotki z bieżącego rozszerzenia ich nie naruszają. Jednak inne zależności funkcyjne ( $A \rightarrow B$  naruszona w krotkach 1. i 2.;  $B \rightarrow A$  naruszona w krotkach 2. i 3.;  $D \rightarrow C$  naruszona w krotkach 3. i 4.) *nie obowiązują*, ponieważ w rozszerzeniu występują ich naruszenia.

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

RYSUNEK 14.8. Relacja  $R(A, B, C, D)$  i jej rozszerzenie

Na rysunku 14.3 wprowadzono **notację schematyczną** (ang. *diagrammatic notation*) zapisu zależności funkcyjnych: każda zależność funkcyjna jest zapisywana jako linia pozioma. Atrybuty lewostronne zależności funkcyjnej są łączone za pomocą linii pionowych z linią reprezentującą zależność funkcyjną, natomiast atrybuty prawostronne są łączone za pomocą strzałek wskazujących na atrybuty.

$F$  to zbiór zależności funkcyjnych określonych w schemacie relacji  $R$ . Projektant schematu zwykle określa zależności funkcyjne, które są *semantycznie oczywiste*. Jednak dla *wszystkich* dopuszczalnych realizacji relacji spełniających zależności ze zbioru  $F$  jest zachowanych również wiele innych zależności funkcyjnych. Owe pozostałe zależności można wywnioskować lub wydedukować na podstawie zależności funkcyjnych ze zbioru  $F$ . Szczegóły reguł wnioskowania i właściwości zależności funkcyjnych opiszemy w rozdziale 15.

## 14.3. Postaci normalne oparte na kluczach głównych

Po wprowadzeniu do zależności funkcyjnych możemy przejść do pokazania, jak je wykorzystać do opracowania formalnej metodyki sprawdzania i ulepszania schematów relacji. Zakładamy, że dla każdej relacji jest dany zbiór zależności funkcyjnych oraz że każda relacja posiada określony klucz główny. Informacje te wraz z testami (warunkami) postaci normalnych sterują przebiegiem *procesu normalizacji* (ang. *normalization process*) projektów schematów relacji. Większość praktycznych projektów relacji przyjmuje jedno z dwóch rozwiązań:

- Najpierw jest wykonywany projekt schematu koncepcyjnego przy użyciu modelu koncepcyjnego, takiego jak ER lub EER, a następnie dokonywane jest odwzorowanie projektu koncepcyjnego na zbiór relacji.
- Relacje są projektowane na podstawie posiadanej wiedzy wynikającej z zapoznania się z istniejącymi implementacjami plików lub formularzami albo raportami.

W przypadku postępowania zgodnie z jednym z powyższych podejść przydatne jest określenie poprawności posiadanych relacji i, w razie potrzeby, dalsza ich dekompozycja w celu otrzymania wyższych postaci normalnych przy wykorzystaniu teorii normalizacyjnej prezentowanej w tym i następnym rozdziale. W niniejszym podrozdziale skupimy się na pierwszych trzech postaciach normalnych schematów relacji oraz ich intuicyjnym znaczeniu. Zostanie również przedstawiony rys historyczny ich opracowania. Bardziej ogólne definicje tych postaci normalnych, uwzględniające wszystkie klucze kandydujące relacji, a nie tylko klucz główny, przedstawiono w podrozdziale 14.4.

Rozpoczynamy od nieformalnego omówienia postaci normalnych oraz źródeł ich opracowania, jak również przypomnienia niektórych definicji z rozdziału 3., które będą nam potrzebne. Następnie, w punkcie 14.3.4, zostanie omówiona pierwsza postać normalna (1NF), zaś w punktach 14.3.5 i 14.3.6 zostaną przedstawione definicje drugiej postaci normalnej (2NF) oraz trzeciej postaci normalnej (3NF), opartych na kluczach głównych.

### 14.3.1. Normalizacja relacji

Proces normalizacji, zaproponowany przez Codda (1972a), polega na poddaniu schematu relacji serii testów, *określających*, czy spełnia on wymagania określonej **postaci normalnej** (ang. *normal form*). Proces przebiegający według schematu zstępującego poprzez weryfikowanie zgodności relacji z określonymi warunkami postaci normalnych oraz, w razie potrzeby, dekompozycję relacji, można więc postrzegać jako *projektowanie relacyjne przez analizę*. Początkowo Codd zaproponował trzy postaci normalne, które nazwał pierwszą, drugą i trzecią postacią normalną. Bardziej rygorystyczna definicja trzeciej postaci normalnej, nosząca nazwę postaci normalnej Boyce'a-Codda (BCNF), została zaproponowana później przez Boyce'a i Codda. Wszystkie te postaci normalne są oparte na jednym narzędziu analitycznym: zależnościach funkcyjnych występujących między atrybutami relacji. Później zaproponowano czwartą (4NF) i piątą (5NF) postać normalną, których definicje bazują na pojęciach zależności wielowartościowych oraz zależności złączeniowych. Zostały one pokrótce omówione w podrozdziałach 14.6 i 14.7.

**Normalizację danych** (ang. *normalization of data*) można postrzegać jako proces analizowania określonych schematów relacji w oparciu o ich zależności funkcyjne oraz klucze główne w celu otrzymania pożądanych właściwości. Właściwościami tymi są: (1) minimalizacja nadmiarowości oraz (2) minimalizacja anomalii wstawiania, usuwania i aktualizowania omówionych w podrozdziale 14.1.2. Niepoprawne schematy relacji, które nie spełniają określonych warunków — **testów postaci normalnej** (ang. *normal form tests*), są rozkładane na mniejsze schematy relacji, które z powodzeniem przechodzą testy, a zatem mają odpowiednie właściwości. Zatem procedura normalizacji oferuje projektantom baz danych następujące elementy:

- formalną strukturę analizy schematów relacji w oparciu o ich klucze oraz zależności funkcyjne występujące między ich atrybutami;
- szereg testów postaci normalnych, które można wykonywać w odniesieniu do pojedynczych schematów relacji, dzięki czemu relacyjna baza danych może zostać **znormalizowana** do dowolnego wymaganego poziomu.

**Definicja.** Pojęcie **postaci normalnej** relacji odnosi się do warunków najwyższej postaci normalnej, jakie spełnia, a przez to określa poziom, do jakiego została znormalizowana.

Postaci normalne, rozpatrywane *oddzielnie* od innych czynników, nie gwarantują dobrego projektu bazy danych. Ogólnie rzecz biorąc, nie wystarczy oddzielne sprawdzenie, czy każdy schemat relacji w bazie danych znajduje się, na przykład, w postaci BCNF lub 3NF. Proces normalizacji przez dekompozycję musi również potwierdzać istnienie dodatkowych właściwości, które, w ujęciu całościowym, posiadają schematy relacji. Mowa tu o dwóch właściwościach:

- **właściwość złączenia bezstratnego** (ang. *lossless join property*), inaczej **właściwość złączenia nieaddytywnego** (ang. *nonadditive join property*), która gwarantuje, że problem generowania fałszywych krotek, opisany w podrozdziale 14.1.4, nie występuje w przypadku schematów relacji utworzonych po dekompozycji;
- **właściwość zachowania zależności** (ang. *dependency preservation property*), która zapewnia, że każda zależność funkcyjna jest reprezentowana w pewnej pojedynczej relacji wynikającej z dekompozycji.

Właściwość złączenia nieaddytywnego jest niezmiernie istotna i **musi być zachowana za wszelką cenę**, natomiast właściwość zachowania zależności, choć pożądana, czasem może być pomijana, co omówiono w podrozdziale 15.2.2. Prezentację formalnych pojęć i technik gwarantujących zachowanie dwóch powyższych właściwości odkładamy do rozdziału 15.

### 14.3.2. Praktyczne zastosowania postaci normalnych

Większość praktycznych projektów w środowisku komercyjnym i rządowym posiłkuje się już istniejącymi projektami baz danych, projektami pochodzącymi ze starych modeli lub istniejącymi plikami. Projektanci są oczywiście zainteresowani zapewnieniem wysokiej jakości projektów i możliwości utrzymywania ich przez długi czas. Istniejące projekty są oceniane na podstawie testów postaci normalnych, a normalizacja jest w praktyce przeprowadzana tak, aby projekty wynikowe cechowała wysoka jakość oraz aby posiadały one pożądane właściwości, które omówiono powyżej. Choć zdefiniowano kilka wyższych postaci

normalnych, takich jak 4NF lub 5NF, omawianych w podrozdziałach 14.6 i 14.7, praktyczne wykorzystanie tych postaci wydaje się co najmniej problematyczne, kiedy więzy, na których bazują, są trudne do zrozumienia lub wykrycia przez projektantów bazy danych oraz użytkowników, którzy muszą znać lub odkrywać takie więzy. Stąd projektowanie baz danych, zgodnie z normami obowiązującymi we współczesnym świecie, jest związane z przykładaniem dużej wagi do procesu normalizacji, ale tylko do poziomu 3NF, BCNF lub 4NF.

Kolejną istotną sprawą jest zauważenie, że projektanci *nie muszą* przeprowadzać normalizacji do najwyższej możliwej postaci normalnej. Relacje można pozostawić w niższej postaci normalnej, na przykład 2NF, ze względów wydajnościowych, co omówiono na końcu podrozdziału 14.1.2. Powoduje to jednak problemy w postaci konieczności radzenia sobie z anomaliami.

**Definicja.** Proces zachowywania złączenia wyższych postaci normalnych jako relacji głównej, która znajduje się w niższej postaci normalnej, nosi nazwę **denormalizacji** (ang. *denormalization*).

### 14.3.3. Definicje kluczy i atrybutów należących do kluczy

Zanim przejdziemy do omawiania kolejnych zagadnień, należy raz jeszcze przyjrzeć się definicjom kluczy schematów relacji przedstawionym w rozdziale 3.

**Definicja. Nadklucz** (ang. *superkey*) schematu relacji  $R = \{A_1, A_2, \dots, A_n\}$  to zbiór atrybutów  $S \subseteq R$  o tej właściwości, że żadne dwie krotki  $t_1$  i  $t_2$  w dowolnym dopuszczalnym stanie  $r$  relacji  $R$  nie spełniają równości  $t_1[S] = t_2[S]$ . **Klucz** (ang. *key*)  $K$  jest nadkluczem posiadającym dodatkową właściwość, która określa, że usunięcie dowolnego atrybutu z  $K$  powoduje, że  $K$  przestaje być nadkluczem.

Różnica między kluczem a nadkluczem polega na tym, że klucz musi być *minimalny*, to znaczy, jeżeli posiadamy klucz  $K = \{A_1, A_2, \dots, A_n\}$  relacji  $R$ , to  $K - \{A_i\}$  nie jest kluczem relacji  $R$  dla dowolnego  $A_i$ , gdzie  $1 \leq i \leq n$ . Na rysunku 14.1  $\{\text{PESEL}\}$  jest kluczem relacji PRACOWNIK, zaś  $\{\text{PESEL}\}$ ,  $\{\text{PESEL}, \text{IMIĘNAZWPRAC}\}$  oraz  $\{\text{PESEL}, \text{IMIĘNAZWPRAC}, \text{DATAUR}\}$  oraz dowolny zbiór atrybutów zawierający atrybut PESEL są nadkluczami.

Jeżeli schemat relacji posiada więcej niż jeden klucz, każdy z nich jest określany mianem **klucza kandydującego** (ang. *candidate key*). Jeden z kluczy kandydujących zostaje *arbitralnie* określony jako **klucz główny** (ang. *primary key*), zaś pozostałe określa się mianem kluczy drugorzędnych. Każdy schemat relacji musi posiadać klucz główny. Jeśli nie jest znany klucz kandydujący relacji, całą relację można traktować jako domyślny nadklucz. Na rysunku 14.1  $\{\text{PESEL}\}$  jest jedynym kluczem kandydującym dla relacji PRACOWNIK, więc jest również kluczem głównym.

**Definicja.** Atrybut schematu relacji  $R$  określa się mianem **atrybutu podstawowego** (ang. *prime attribute*) relacji  $R$ , jeżeli jest składową pewnego klucza kandydującego relacji  $R$ . Atrybut określa się jako **niepodstawowy** (ang. *nonprime*), jeżeli nie jest atrybutem podstawowym, to znaczy jeśli nie jest składową żadnego klucza kandydującego.

Na rysunku 14.1 zarówno PESEL, jak i NUMERPROJ są atrybutami podstawowymi relacji PRACUJE\_NAD, natomiast inne atrybuty tej relacji są niepodstawowe.

Poniżej zostaną przedstawione pierwsze trzy postaci normalne: 1NF, 2NF oraz 3NF. Zostały one zaproponowane przez Codd'a (1972a) w formie sekwencji działań zmierzających do otrzymania pożądanego stanu relacji w trzeciej postaci normalnej w wyniku przejścia, w razie potrzeby, przez stany pośrednie pierwszej i drugiej postaci normalnej. Jak zostanie pokazane, druga i trzecia postać normalna pozwalają na rozwiązanie różnych problemów wynikających ze sprawiających kłopoty zależności funkcyjnych między atrybutami. Jednakże, ze względów historycznych, przyjęło się wykonywać działania właśnie w takiej kolejności; z definicji relacja w trzeciej postaci normalnej *spełnia już* warunki drugiej postaci normalnej.

### 14.3.4. Pierwsza postać normalna

**Pierwsza postać normalna** (ang. *first normal form*, 1NF) jest obecnie traktowana jako element formalnej definicji relacji w podstawowym (płaskim) modelu relacyjnym. Historycznie zdefiniowano ją jako warunek zapobiegania występowaniu atrybutów wielowartościowych, atrybutów złożonych oraz ich kombinacji. Określa ona, że dziedzina atrybutu musi zawierać wyłącznie *wartości niepodzielne* (proste) oraz że wartość dowolnego atrybutu krotki musi być *pojedynczą wartością* pochodzącą z dziedziny tego atrybutu. Zatem pierwsza postać normalna zabrania występowania zbioru wartości, krotki wartości lub ich kombinacji jako wartości atrybutu *pojedynczej krotki*. Innymi słowy, pierwsza postać normalna zabrania występowania *relacji w relacjach* lub *relacji jako wartości atrybutu w krotkach*. Jedyne wartości atrybutów dopuszczalne w ramach postaci 1NF to pojedyncze **wartości niepodzielne** (ang. *atomic values*).

Weźmy pod uwagę schemat relacji DZIAŁ z rysunku 14.1, której kluczem głównym jest NUMERDZ, i założmy, że rozszerzamy ją przez dołączenie atrybutu LOKALIZACJEDZ, tak jak na rysunku 14.9(a). Zakładamy, że każdy dział może posiadać *wiele* lokalizacji. Schemat DZIAŁ oraz przykładowy stan relacji przedstawiono na rysunku 14.9. Jak widać, nie znajduje się ona w postaci 1NF, ponieważ LOKALIZACJEDZ nie jest atrybutem niepodzielnym, co ilustruje pierwsza krotka na rysunku 14.9(b). Atrybut ten można postrzegać na dwa sposoby:

- Dziedzina atrybutu LOKALIZACJEDZ zawiera wartości niepodzielne, jednak niektóre krotki mogą zawierać zbiór takich wartości. W tym przypadku atrybut LOKALIZACJEDZ *nie jest* zależny funkcyjnie od klucza głównego NUMERDZ.
- Dziedzina atrybutu LOKALIZACJEDZ zawiera zbiory wartości, a stąd nie jest niepodzielna. W tym przypadku mamy do czynienia z zależnością  $\text{NUMERDZ} \rightarrow \text{LOKALIZACJEDZ}$ , ponieważ każdy zbiór można traktować jako pojedynczy element należący do dziedziny atrybutu<sup>10</sup>.

W obu przypadkach relacja DZIAŁ z rysunku 14.9 nie jest w pierwszej postaci normalnej. W rzeczywistości nie można jej nawet uznać za relację, zgodnie z definicją przedstawioną w podrozdziale 3.1. Istnieją trzy podstawowe techniki otrzymywania pierwszej postaci normalnej dla tego rodzaju relacji.

---

<sup>10</sup> W takim przypadku dziedzinę atrybutu LOKALIZACJEDZ można traktować jako **zbiór potęgowy** (ang. *power set*) zbioru pojedynczych lokalizacji. Oznacza to, że dziedzina składa się ze wszystkich możliwych podzbiorów zbioru pojedynczych lokalizacji.

(a)

DZIAŁ

NAZWADZ	NUMERDZ	PESELKIEROWNIKA	LOKALIZACJEDZ
---------	---------	-----------------	---------------

(b)

DZIAŁ

NAZWADZ	NUMERDZ	PESELKIEROWNIKA	LOKALIZACJEDZ
Badawczy	5	33344445555	{Kielce, Kraków, Warszawa}
Administracja	4	98765432109	{Gliwice}
Dyrekcja	1	88886665555	{Warszawa}

(c)

DZIAŁ\_1NF

NAZWADZ	NUMERDZ	PESELKIEROWNIKA	LOKALIZACJEDZ
Badawczy	5	33344445555	Kielce
Badawczy	5	33344445555	Kraków
Badawczy	5	33344445555	Warszawa
Administracja	4	98765432109	Gliwice
Dyrekcja	1	88886665555	Warszawa

RYСУNEK 14.9. Normalizacja do pierwszej postaci normalnej. (a) Schemat relacji niebędącej w postaci 1NF. (b) Przykładowy stan relacji DZIAŁ. (c) Wersja tej samej relacji w postaci 1NF z nadmiarowością

- (1) Usuwamy atrybut LOKALIZACJEDZ, który narusza właściwości pierwszej postaci normalnej, i umieszczamy go w odrębnej relacji LOKALIZACJE\_DZ wraz z kluczem głównym NUMERDZ relacji DZIAŁ. Kluczem głównym tej relacji jest połączenie {NUMERDZ, LOKALIZACJEDZ}, jak na rysunku 14.2. Dla *każdej lokalizacji* działu istnieje odrębna krotka w relacji LOKALIZACJE\_DZ. W ten sposób otrzymujemy rozkład relacji niebędącej w postaci 1NF na dwie relacje będące w postaci 1NF.
- (2) Rozszerzamy klucz tak, aby w oryginalnej relacji DZIAŁ dla każdej lokalizacji działu występowała oddzielna krotka, co pokazano na rysunku 14.9(c). W tym przypadku klucz główny staje się połączeniem {NUMERDZ, LOKALIZACJEDZ}. Rozwiązanie to ma tę wadę, że wprowadza do relacji *nadmiarowość*, dlatego jest rzadko stosowane.
- (3) Jeżeli znana jest *maksymalna liczba wartości* atrybutu, na przykład jeżeli wiadomo, że dla dowolnego działu mogą istnieć *najwyżej trzy lokalizacje*, zastępujemy atrybut LOKALIZACJEDZ trzema atrybutami niepodzielnymi: LOKALIZACJEDZ1, LOKALIZACJEDZ2 oraz LOKALIZACJEDZ3. Rozwiązanie to ma tę wadę, że wprowadza *wartości null*, jeżeli większość działów posiada mniej niż trzy lokalizacje. Ponadto wprowadza ono fałszywe informacje znaczeniowe o *porządku* lokalizacji, który początkowo nie był istotny. Wykonywanie zapytań względem atrybutu staje się utrudnione. Weźmy na



przykład pod uwagę sposób zapisu w przypadku takiego projektu zapytania: *wybierz działy, dla których jedną z lokalizacji są „Kielce”*. Z tych powodów najlepiej jest unikać tej techniki.

Spośród trzech powyższych rozwiązań pierwsze jest, ogólnie rzecz biorąc, uważane za najlepsze, ponieważ nie niesie ze sobą problemów z nadmiarowością i charakteryzuje się ogólnością, nie nakładając ograniczeń na maksymalną liczbę wartości. W rzeczywistości jeżeli wybierze się drugie rozwiązanie, zostanie ono poddane dalszej dekompozycji w czasie kolejnych etapów normalizacji, w efekcie dając pierwsze rozwiązanie.

Pierwsza postać normalna zabrania również występowania atrybutów wielowartościowych, które same są złożone. Są to tak zwane **relacje zagnieżdżone** (ang. *nested relations*), ponieważ każda krotka może posiadać relację *w sobie*. Na rysunku 14.10 przedstawiono, jak mogłaby wyglądać relacja PRAC\_PROJ, gdyby zagnieżdżanie było dozwolone. Każda krotka reprezentuje encję pracownika i relacja PROJEKTY(NUMER\_PROJ, GODZINY) w *każdej krotce* reprezentuje projekty pracownika oraz liczbę godzin, jakie poświęca tygodniowo na każdy projekt. Schemat takiej relacji PRAC\_PROJ można przedstawić następująco:

PRAC\_PROJ({PESEL, IMIĘNAZWPRAC, {PROJEKTY(NUMERPROJ, GODZINY)}})

(a)

PRAC_PROJ		PROJEKTY	
PESEL	IMIĘNAZWPRAC	NUMERPROJ	GODZINY

(b)

PRAC_PROJ			
PESEL	IMIĘNAZWPRAC	NUMERPROJ	GODZINY
12345678901	Nowak, Jan	1	32,5
		2	7,5
33344445555	Kowalski, Fryderyk	3	40,0
99988887777	Zielińska, Alicja	1	20,0
		2	20,0
98765432109	Owsiak, Maria	2	10,0
		3	10,0
		10	10,0
		20	10,0
66688844444	Głębocki, Adam	30	30,0
		10	10,0
45345345345	Polakowski, Maciej	10	35,0
		30	5,0
98798798798	Kuc, Waldemar	30	20,0
		20	15,0
88886665555	Szymkowiak, Krystyna	20	NULL

(c)

PRAC_PROJ1	
PESEL	IMIĘNAZWPRAC

PRAC_PROJ2		
PESEL	NUMERPROJ	GODZINY

RYСУNEK 14.10. Normalizacja relacji zagnieżdżonych do pierwszej postaci normalnej. (a) Schemat relacji PRAC\_PROJ z atrybutem „relacji zagnieżdżonej” PROJEKTY. (b) Przykładowe rozszerzenie relacji PRAC\_PROJ pokazujące relacje zagnieżdżone w każdej krotce. (c) Dekompozycja relacji PRAC\_PROJ do relacji PRAC\_PROJ1 oraz PRAC\_PROJ2 poprzez propagację klucza głównego



Nawiasy klamrowe {} identyfikują atrybut PROJEKTY jako wielowartościowy i w nawiasach () wymieniono atrybuty składające się na PROJEKTY. Co interesujące, bieżące trendy związane z obsługą obiektów złożonych (patrz rozdział 12.) oraz danych XML (patrz rozdział 13.) przy użyciu modelu relacyjnego próbują dopuszczać występowanie i formalizować relacje zagnieżdżone w relacyjnych systemach baz danych, które wcześniej były zabronione przez warunki pierwszej postaci normalnej.

Należy zauważyć, że atrybut PESEL jest kluczem głównym relacji PRAC\_PROJ na rysunkach 14.10(a) i (b), zaś atrybut NUMERPROJ jest **kluczem częściowym** (ang. *partial key*) relacji zagnieżdżonej. Oznacza to, że w każdej krotce relacja zagnieżdżona musi posiadać unikatowe wartości atrybutu NUMERPROJ. W celu jej znormalizowania do pierwszej postaci normalnej usuwamy atrybuty relacji zagnieżdżonej, przenosząc je do nowej relacji, do której *propagujemy klucz główny*. Klucz główny nowej relacji stanowi połączenie klucza częściowego z kluczem głównym oryginalnej relacji. Dekompozycja i propagacja klucza głównego dają w efekcie schematy PRAC\_PROJ1 i PRAC\_PROJ2, przedstawione na rysunku 14.10(c).

Procedurę tę można zastosować rekurencyjnie względem relacji o wielu poziomach zagnieżdżenia w celu **usunięcia zagnieżdżenia** (ang. *unnest*) relacji i utworzenia zbioru relacji w pierwszej postaci normalnej. Jest to przydatne w przypadku konwertowania nieznormalizowanego schematu relacji o wielu poziomach zagnieżdżenia do relacji w postaci 1NF. Przykładowo, weźmy pod uwagę następującą relację:

```
KANDYDAT(PESEL, IMIĘNAZWISKO, {HIST_ZATR(FIRMA, NAJWYŻ_STAN,
{HIST_SPRZED(ROK, MAKS_SPRZED)}}))
```

Ta relacja opisuje dane na temat kandydatów aplikujących o pracę. Historia zatrudnienia jest zagnieżdżoną relacją, a historia sprzedaży jest zapisana jako jeszcze głębiej zagnieżdżona relacja. Pierwsza normalizacja, z wykorzystaniem kluczy częściowych FIRMA i ROK, skutkuje następującymi relacjami w pierwszej postaci normalnej:

```
KANDYDAT_1(PESEL, IMIĘNAZWISKO)
KANDYDAT_HIST_ZATR(PESEL, FIRMA, NAJWYŻ_STAN)
KANDYDAT_HIST_SPRZED(PESEL, FIRMA, ROK, MAKS_SPRZED)
```

Jeśli w relacji występuje więcej niż jeden atrybut wielowartościowy, należy zachować ostrożność. Rozważ np. następującą relację, która nie jest w pierwszej postaci normalnej:

```
OSOBA(PESEL#, {DOWÓD_REJESTR#}, {TELEFON#})
```

Relacja ta reprezentuje fakt, że osoba może posiadać wiele samochodów i wiele telefonów. Jeżeli wykorzystamy strategię postępowania opisaną wcześniej jako drugą, w rezultacie otrzymamy relację, której kluczem będą wszystkie jej atrybuty:

```
OSOBA_W_1NF(PESEL#, DOWÓD_REJESTR#, TELEFON#)
```

W celu uniknięcia wprowadzania dodatkowych związków między atrybutami DOWÓD\_REJESTR# a TELEFON#, wszystkie możliwe kombinacje wartości są reprezentowane dla każdej wartości atrybutu PESEL#, co zwiększa nadmiarowość. Prowadzi to do problemów (zwykle wykrywanych na dalszych etapach normalizacji), z którymi radzą sobie zależności wielowartościowe i czwarta postać normalna, omawiane w podrozdziale 14.6. Poprawny sposób postępowania w przypadku dwóch atrybutów wielowartościowych w powyższej relacji OSOBA polega na rozłożeniu jej na dwie oddzielne relacje przy użyciu strategii 1., omówionej powyżej: 01(PESEL#, DOWÓD\_REJESTR#) oraz 02(PESEL#, TELEFON#).

Warto wspomnieć o relacjach obejmujących atrybuty inne niż proste dane numeryczne lub znakowe. Obecnie w bazach coraz częściej pojawiają się zdjęcia, dokumenty, nagrania wideo, muzyka itd. Gdy takie dane są zapisywane w relacji, cały obiekt lub plik jest traktowany jak wartość atomowa, zapisywana za pomocą języka SQL jako obiekt typu BLOB (ang. *binary large object*) lub CLOB (ang. *character large object*). W celach praktycznych taki obiekt jest traktowany jak atomowy atrybut jednowartościowy, dlatego nie narusza zgodności relacji z pierwszą postacią normalną.

### 14.3.5. Druga postać normalna

**Druga postać normalna** (ang. *second normal form, 2NF*) jest oparta na pojęciu *zupełnej zależności funkcyjnej* (ang. *full functional dependency*). Zależność funkcyjna  $X \rightarrow Y$  jest **zupełną zależnością funkcyjną**, jeżeli usunięcie dowolnego atrybutu  $A$  ze zbioru  $X$  sprawia, że zależność przestaje obowiązywać, to znaczy dla dowolnego atrybutu  $A \in X$ ,  $(X - \{A\})$  nie determinuje funkcyjnie  $Y$ . Zależność funkcyjna  $X \rightarrow Y$  jest **zależnością częściową** (ang. *partial dependency*), jeżeli ze zbioru  $X$  można usunąć pewien atrybut  $A \in X$ , a zależność wciąż będzie zachowana. Oznacza to, że dla pewnego  $A \in X$ ,  $(X - \{A\}) \rightarrow Y$ . Na rysunku 14.3(b) zależność  $\{\text{PESEL}, \text{NUMERPROJ}\} \rightarrow \text{GODZINY}$  jest zależnością zupełną (nie jest zachowana ani zależność  $\text{PESEL} \rightarrow \text{GODZINY}$ , ani  $\text{NUMERPROJ} \rightarrow \text{GODZINY}$ ), jednak zależność  $\{\text{PESEL}, \text{NUMERPROJ}\} \rightarrow \text{IMIĘ} \rightarrow \text{NAZWPRAC}$  jest częściowa, gdyż zachowana jest zależność  $\text{PESEL} \rightarrow \text{IMIĘ} \rightarrow \text{NAZWPRAC}$ .

**Definicja.** Schemat relacji  $R$  znajduje się w drugiej postaci normalnej, jeżeli każdy niepodstawowy atrybut  $A$  należący do  $R$  jest *zupełnie zależny funkcyjnie* od klucza głównego relacji  $R$ .

Test dotyczący drugiej postaci normalnej wiąże się ze sprawdzeniem zależności funkcyjnych, których atrybuty lewostronne są częścią klucza głównego. Jeżeli klucz główny zawiera jeden atrybut, testu w ogóle nie trzeba przeprowadzać. Relacja  $\text{PRAC\_PROJ}$  z rysunku 14.3(b) jest w postaci 1NF, ale nie 2NF. Niepodstawowy atrybut  $\text{IMIĘ} \rightarrow \text{NAZWPRAC}$  narusza reguły drugiej postaci normalnej ze względu na zależność funkcyjną  $\text{FD2}$ , podobnie jak niepodstawowe atrybuty  $\text{NAZWAPROJ}$  i  $\text{LOKALIZACJAPROJ}$  ze względu na zależność  $\text{FD3}$ . Obie te zależności funkcyjne ( $\text{FD2}$  i  $\text{FD3}$ ) naruszają drugą postać normalną, ponieważ  $\text{IMIĘ} \rightarrow \text{NAZWPRAC}$  można ustalić za pomocą samego numeru  $\text{PESEL}$ , a  $\text{NAZWAPROJ}$  i  $\text{LOKALIZACJA} \rightarrow \text{PROJ}$  mogą zostać określone za pomocą samego numeru projektu. Atrybuty  $\text{PESEL}$  i  $\text{NUMERPROJ}$  są częścią klucza głównego  $\{\text{PESEL}, \text{NUMERPROJ}\}$  relacji  $\text{PRAC\_PROJ}$ , co narusza 2NF.

Jeżeli schemat relacji nie jest w drugiej postaci normalnej, może zostać *znormalizowany do 2NF* do wielu relacji 2NF, w których atrybuty niepodstawowe są powiązane tylko z częścią klucza głównego, od którego są zupełnie zależne funkcyjnie. Zależności funkcyjne  $\text{FD1}$ ,  $\text{FD2}$  oraz  $\text{FD3}$  z rysunku 14.3(b) prowadzą więc do dekompozycji relacji  $\text{PRAC\_PROJ}$  na trzy schematy relacji  $\text{PP1}$ ,  $\text{PP2}$  oraz  $\text{PP3}$ , będące w drugiej postaci normalnej, które przedstawiono na rysunku 14.11(a).

(a)

PRAC\_PROJ

PESEL	NUMERPROJ	GODZINY	IMIĘNAZWPRAC	NAZWAPROJ	LOKALIZACJAPROJ
-------	-----------	---------	--------------	-----------	-----------------



Normalizacja do drugiej postaci normalnej

PP1

PESEL	NUMERPROJ	GODZINY
-------	-----------	---------



PP2

PESEL	IMIĘNAZWPRAC
-------	--------------



PP3

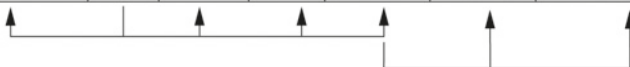
NUMERPROJ	NAZWAPROJ	LOKALIZACJAPROJ
-----------	-----------	-----------------



(b)

PRAC\_DZ

IMIĘNAZWPRAC	PESEL	DATAUR	ADRES	NUMERDZ	NAZWADZ	PESELKIEROWNIKA
--------------	-------	--------	-------	---------	---------	-----------------



Normalizacja do trzeciej postaci normalnej

PW1

IMIĘNAZWPRAC	PESEL	DATAUR	ADRES	NUMERDZ
--------------	-------	--------	-------	---------



PW2

NUMERDZ	NAZWADZ	PESELKIEROWNIKA
---------	---------	-----------------



RYSunEK 14.11. Normalizacja do drugiej i trzeciej postaci normalnej. (a) Normalizacja relacji PRAC\_PROJ do relacji w drugiej postaci normalnej. (b) Normalizacja relacji PRAC\_DZ do relacji w trzeciej postaci normalnej

### 14.3.6. Trzecia postać normalna

**Trzecia postać normalna** (ang. *third normal form, 3NF*) jest oparta na pojęciu *zależności przechodniej* (ang. *transitive dependency*). Zależność funkcyjna  $X \rightarrow Y$  w schemacie relacji  $R$  jest **zależnością przechodnią**, jeżeli istnieje taki zbiór atrybutów  $Z$ , który nie tworzy ani klucza kandydującego, ani podzbioru żadnego klucza w relacji  $R$ <sup>11</sup> oraz zarówno zależność  $X \rightarrow Z$ , jak i  $Z \rightarrow Y$  jest spełniona. Zależność  $PESEL \rightarrow PESELKIEROWNIKA$  jest przechodnia przez atrybut  $NUMERDZ$  z relacji  $PRAC\_DZ$  na rysunku 14.3(a), ponieważ zarówno zależność  $PESEL \rightarrow NUMERDZ$ , jak i  $NUMERDZ \rightarrow PESELKIEROWNIKA$  jest spełniona *oraz* atrybut  $NUMERDZ$  nie jest sam kluczem, ani nie stanowi podzbioru klucza relacji  $PRAC\_DZ$ . Intuicyjnie możemy stwierdzić, że zależność atrybutu  $PESELKIEROWNIKA$  od atrybutu  $NUMERDZ$  jest niepożądana w relacji  $PRAC\_DZ$ , gdyż  $NUMERDZ$  nie jest kluczem relacji  $PRAC\_DZ$ .

<sup>11</sup> Jest to ogólna definicja zależności przechodniej. Ze względu na fakt, że w niniejszym podrozdziale jesteśmy zainteresowani tylko kluczami głównymi, dopuszczamy zależności przechodnie, w przypadku których  $X$  jest kluczem głównym, ale  $Z$  może być kluczem kandydującym (lub jego podzbiorem).

**Definicja.** Według oryginalnej definicji Codd'a schemat relacji  $R$  jest w **trzeciej postaci normalnej**, jeżeli jest w drugiej postaci normalnej *oraz* żaden niepodstawowy atrybut relacji  $R$  nie jest przechodnio zależny od klucza głównego.

Schemat relacji PRAC\_DZ z rysunku 14.3(a) jest w drugiej postaci normalnej, gdyż nie istnieje żadna zależność częściowa względem klucza. Jednakże relacja ta nie jest w trzeciej postaci normalnej ze względu na występowanie zależności przechodniej atrybutu PESEL  $\rightarrow$  KIEROWNIKA (oraz NAZWADZ) od PESEL poprzez NUMERDZ. Można znormalizować relację PRAC\_DZ, rozkładając ją na dwa schematy relacji w postaci 3NF, PD1 oraz PD2, przedstawione na rysunku 14.11(b). Intuicyjnie możemy stwierdzić, że relacje PD1 oraz PD2 reprezentują niezależne encje faktów o pracownikach i działach. Operacja złączenia naturalnego wykonana na relacjach PD1 i PD2 pozwala na odtworzenie oryginalnej relacji PRAC\_DZ bez generowania fałszywych krotek.

Intuicyjnie stwierdzamy, że dowolna zależność funkcyjna, której lewa strona jest częścią (podzbiorem właściwym) klucza głównego, lub dowolna zależność funkcyjna, której lewa strona jest atrybutem niekluczowym, stanowią zależności *problematyczne*. Normalizacja do drugiej i trzeciej postaci normalnej pozwala na pozbycie się takich problematycznych zależności funkcyjnych poprzez rozłożenie oryginalnej relacji na nową relację. W kontekście procesu normalizacji nie jest konieczne usunięcie zależności częściowych przed zależnościami przechodnimi, jednak ze względów historycznych trzecie postaci normalną zdefiniowano przy założeniu, że relację sprawdzono najpierw pod względem drugiej postaci normalnej. Ponadto ogólna definicja 3NF przedstawiona w punkcie 14.4.2 automatycznie uwzględnia warunek zgodności z 2NF. W tabeli 14.1 przedstawiono nieformalne podsumowanie trzech postaci normalnych opartych na kluczach głównych, testów wykonywanych w każdym przypadku oraz odpowiednich *recept* lub działań normalizacyjnych wykonywanych w celu otrzymania danej postaci normalnej.

TABELA 14.1. Podsumowanie postaci normalnych opartych na kluczach głównych oraz odpowiednie metody normalizacji

Postać normalna	Test	Receptura (normalizacja)
Pierwsza (1NF)	Relacja powinna mieć tylko atrybuty niepodzielne oraz nie mieć relacji zagnieżdżonych.	Tworzymy nowe relacje dla każdego atrybutu relacji zagnieżdżonej nie będącego atrybutem niepodzielnym.
Druga (2NF)	W przypadku relacji, których klucz główny zawiera wiele atrybutów, żaden atrybut niekluczowy nie powinien być zależny funkcyjnie od części klucza głównego.	Rozkładamy i określamy nową relację dla każdego klucza częściowego z jego zależnym atrybutem (atrybutami). Zapewniamy, aby została zachowana relacja z oryginalnym kluczem głównym oraz wszelkimi atrybutami, które są od niego zupełnie zależne funkcyjnie.
Trzecia (3NF)	Relacja nie powinna mieć żadnego atrybutu niekluczowego zdeteminowanego funkcyjnie przez inny atrybut niekluczowy (lub zbiór takich atrybutów). Oznacza to, że nie powinna występować żadna zależność przechodnia atrybutu niekluczowego od klucza głównego.	Rozkładamy i określamy relację zawierającą atrybut(y) niekluczowy(e), który(e) determinuje(a) inny(e) atrybut(y) niekluczowy(e).

## 14.4. Definicje ogólne drugiej i trzeciej postaci normalnej

Ogólnie rzecz biorąc, schematy relacji chcemy projektować w taki sposób, aby nie posiadały ani zależności częściowych, ani przechodnich, ponieważ te rodzaje zależności powodują występowanie anomalii aktualizowania omówionych w punkcie 14.1.2. Działania normalizacyjne zmierzające do otrzymania relacji w trzeciej postaci normalnej, które omówiono dotychczas, zabraniają występowania zależności częściowych lub przechodnich na *kluczu głównym*. Opisany wcześniej proces normalizacji jest przydatny do praktycznej analizy bazy danych, w której już zdefiniowano klucze główne. Jednak definicje te nie uwzględniają innych kluczy kandydujących (o ile takie istnieją). W niniejszym podrozdziale zostaną przedstawione bardziej ogólne definicje drugiej i trzeciej postaci normalnej, które uwzględniają *wszystkie* klucze kandydujące relacji. Należy zauważyć, że nie ma to wpływu na definicję pierwszej postaci normalnej, gdyż jest ona niezależna od kluczy i zależności funkcyjnych. W przypadku ogólnej definicji **atrybutu podstawowego** (ang. *prime attribute*), za atrybut taki uważamy atrybut będący częścią *dowolnego klucza kandydującego*. Zależności funkcyjne częściowe i zupełne oraz zależności przechodnie będziemy teraz rozważać w kontekście *wszystkich kluczy kandydujących* danej relacji.

### 14.4.1. Definicja ogólna drugiej postaci normalnej

**Definicja.** Schemat relacji  $R$  znajduje się w **drugiej postaci normalnej** (2NF), jeżeli żaden niepodstawowy atrybut  $A$  relacji  $R$  nie jest częściowo zależny od *dowolnego* klucza relacji  $R$ <sup>12</sup>.

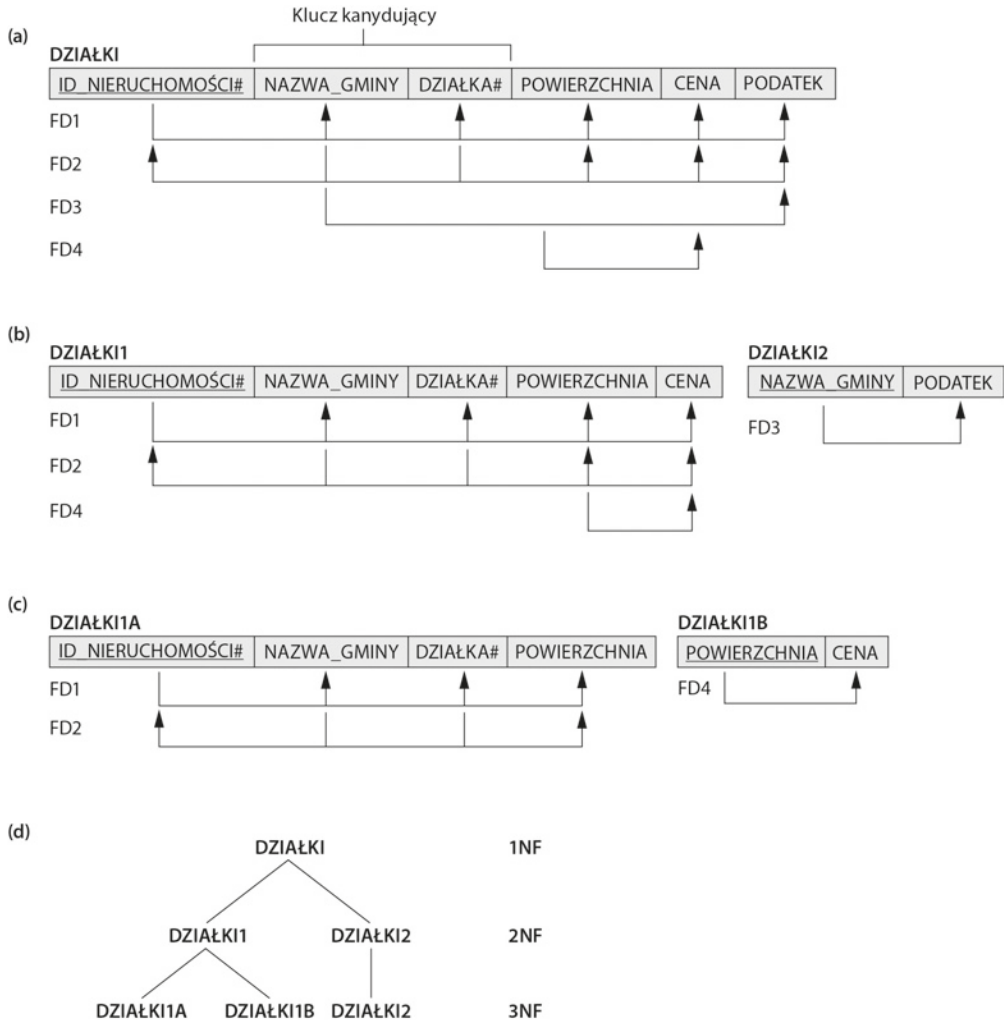
Test dla drugiej postaci normalnej polega na sprawdzeniu zależności funkcyjnych, których atrybuty lewostronne stanowią *część* klucza głównego. Jeżeli klucz główny zawiera pojedynczy atrybut, testu w ogóle nie trzeba przeprowadzać. Weźmy pod uwagę schemat relacji DZIAŁKI z rysunku 14.12(a), który opisuje działki terenu przeznaczone do sprzedaży w różnych gminach województwa. Załóżmy, że istnieją dwa klucze kandydujące: ID\_NIERUCHOMOŚCI# oraz {NAZWA\_GMINY, DZIAŁKA#}. Oznacza to, że numery działek są unikatowe tylko w ramach pojedynczych gmin, zaś wartości ID\_NIERUCHOMOŚCI# są unikatowe w ramach wszystkich gmin całego województwa.

Na podstawie dwóch kluczy kandydujących ID\_NIERUCHOMOŚCI# oraz {NAZWA\_GMINY, DZIAŁKA#} wiemy, że zależności funkcyjne FD1 oraz FD2 z rysunku 14.12(a) są zachowane. Jako klucz główny wybieramy atrybut ID\_NIERUCHOMOŚCI#, więc na rysunku 14.12(a) został on podkreślony, ale żadne szczególne względy nie przemawiają za użyciem właśnie jego jako klucza głównego, zamiast innych kluczy kandydujących. Załóżmy, że w ramach relacji DZIAŁKI zachowane są dwie dodatkowe zależności funkcyjne:

FD3: NAZWA\_GMINY  $\rightarrow$  PODATEK

FD4: POWIERZCHNIA  $\rightarrow$  CENA

<sup>12</sup> Definicję tę można również sformułować następująco: schemat relacji  $R$  znajduje się w drugiej postaci normalnej, jeżeli każdy niepodstawowy atrybut  $A$  relacji  $R$  jest zupełnie zależny od *każdego* klucza relacji  $R$ .



RYSUNEK 14.12. Normalizacja do drugiej i trzeciej postaci normalnej. (a) Relacja DZIAŁKI z jej zależnościami funkcyjnymi od FD1 do FD4. (b) Rozkład na relacje w drugiej postaci normalnej DZIAŁKI1 i DZIAŁKI2. (c) Rozkład relacji DZIAŁKI1 na relacje w trzeciej postaci normalnej DZIAŁKI1A i DZIAŁKI1B. (d) Stopniowa normalizacja relacji DZIAŁKI do 3NF

Zależność FD3 określa, że wysokość podatku jest stała dla danej gminy (nie różni się w przypadku różnych działek jednej gminy), natomiast zależność FD4 określa, że cena działki jest determinowana przez jej powierzchnię bez względu na gminę, w jakiej się znajduje (zakładamy, że jest to cena działki podlegająca opodatkowaniu).

Schemat relacji DZIAŁKI narusza definicję ogólną drugiej postaci normalnej, ponieważ atrybut PODATEK jest częściowo zależny od klucza kandydującego {NAZWA\_GMINY, DZIAŁKA#}, co można stwierdzić na podstawie zależności FD3. W celu znormalizowania relacji DZIAŁKI do drugiej postaci normalnej rozkładamy ją na dwie relacje, DZIAŁKI1 oraz DZIAŁKI2, przedstawione na rysunku 14.12(b). Relację DZIAŁKI1 tworzymy poprzez usunięcie z relacji DZIAŁKI atrybutu PODATEK (który narusza warunki drugiej postaci normalnej) i umiesz-

czenie go wraz z atrybutem NAZWA\_GMINY (lewa strona zależności FD3 odpowiedzialna za zależność częściową) w nowej relacji DZIAŁKI2. Obie relacje, DZIAŁKI1 i DZIAŁKI2, są w drugiej postaci normalnej. Należy zauważyć, że zależność FD4 nie narusza warunków drugiej postaci normalnej i zostaje przeniesiona do relacji DZIAŁKI1.

### 14.4.2. Definicja ogólna trzeciej postaci normalnej

**Definicja.** Schemat relacji  $R$  znajduje się w **trzeciej postaci normalnej** (3NF), jeżeli zawsze, kiedy *nietrywialna* zależność funkcyjna  $X \rightarrow A$  jest zachowana w  $R$ , to albo (a)  $X$  jest nadkluczem w  $R$ , albo (b)  $A$  jest atrybutem podstawowym w relacji  $R$ <sup>13</sup>.

Zgodnie z powyższą definicją relacja DZIAŁKI2 (rysunek 14.12(b)) znajduje się w trzeciej postaci normalnej, jednak zależność FD4 w relacji DZIAŁKI1 narusza warunki trzeciej postaci normalnej, gdyż atrybut POWIERZCHNIA nie jest nadkluczem, zaś atrybut CENA nie jest atrybutem podstawowym relacji DZIAŁKI1. W celu znormalizowania relacji DZIAŁKI1 do trzeciej postaci normalnej rozkładamy ją na schematy relacji DZIAŁKI1A i DZIAŁKI1B przedstawione na rysunku 14.12(c). Relację DZIAŁKI1A tworzymy przez usunięcie atrybutu CENA (który narusza warunki trzeciej postaci normalnej) z relacji DZIAŁKI1 i umieszczenie go wraz z atrybutem POWIERZCHNIA (lewa strona zależności FD4 odpowiedzialna za zależność przechodnią) w relacji DZIAŁKI1B. Zarówno relacja DZIAŁKI1A, jak i DZIAŁKI1B znajduje się w trzeciej postaci normalnej.

W związku z powyższym przykładem i definicją ogólną trzeciej postaci normalnej warto wspomnieć o dwóch sprawach.

- Relacja DZIAŁKI1 narusza warunki trzeciej postaci normalnej, ponieważ atrybut CENA jest przechodnio zależny od każdego z kluczy kandydujących relacji DZIAŁKI1 poprzez niepodstawowy atrybut POWIERZCHNIA.
- Powyższą definicję ogólną można zastosować *bezpośrednio* w celu sprawdzenia, czy schemat relacji znajduje się w trzeciej postaci normalnej. *Nie* jest konieczne wykonywanie najpierw testu dla drugiej postaci normalnej.

W razie zastosowania tej definicji względem relacji DZIAŁKI z zależnościami od FD1 do FD4, możemy określić, że *zarówno* zależność FD3, jak i FD4 narusza warunki trzeciej postaci normalnej. Zatem można bezpośrednio rozłożyć relację DZIAŁKI na relacje DZIAŁKI1A i DZIAŁKI1B oraz DZIAŁKI2. Stąd zależności (przechodnia i częściowa) naruszające warunki trzeciej postaci normalnej, mogą zostać usunięte w *dowolnej kolejności*.

### 14.4.3. Interpretacja definicji ogólnej trzeciej postaci normalnej

Schemat relacji  $R$  narusza definicję ogólną trzeciej postaci normalnej, jeżeli w relacji  $R$  jest zachowana zależność funkcyjna  $X \rightarrow A$  i naruszone są *oba* warunki (a) i (b) powyższej definicji. Pierwszy warunek pozwala wykryć dwa rodzaje problematycznych zależności:

<sup>13</sup> Warto zauważyć, że wyprowadzanie zależności funkcyjnych (omówione w podrozdziale 15.1) sprawia, iż przy spełnionej zależności  $Y \rightarrow A$  spełniona jest także zależność  $Y \rightarrow YA$ . Dlatego bardziej precyzyjne byłoby stwierdzenie, że to  $\{A-X\}$  jest atrybutem podstawowym relacji  $R$ .



- Atrybut niepodstawowy determinuje inny atrybut niepodstawowy. Występuje wtedy zwykle zależność przechodnia naruszająca 3NF.
- Podzbiór właściwy klucza relacji  $R$  funkcyjnie determinuje atrybut niepodstawowy. Występuje wtedy zależność częściowa naruszająca 2NF.

Tak więc już sam warunek (a) uwzględnia sprawiające problemy zależności, które były powodem opisanych normalizacji do 2NF i 3NF.

Zatem możemy sformułować **ogólną alternatywną definicję trzeciej postaci normalnej**:

**Alternatywna definicja.** Schemat relacji  $R$  znajduje się w trzeciej postaci normalnej, jeżeli każdy niepodstawowy atrybut relacji  $R$  spełnia oba poniższe warunki:

- jest zupełnie zależny funkcyjnie od każdego klucza w  $R$ ;
- jest nieprzechodnio zależny od każdego klucza w  $R$ .

Zwróć jednak uwagę na punkt (b) w definicji ogólnej 3NF. Umożliwia on powstanie pewnych zależności funkcyjnych zgodnych z 3NF (i tym samym „niewykrywanych” na podstawie definicji 3NF), które jednak mogą sprawiać trudności. Postać normalna Boyce’a-Codda „wykrywa” te zależności, ponieważ ich nie dopuszcza. Dalej opiszemy tę właśnie postać normalną.

## 14.5. Postać normalna Boyce’a-Codda

**Postać normalna Boyce’a-Codda** (ang. *Boyce-Codd normal form*, BCNF) została zaproponowana jako uproszczona wersja trzeciej postaci normalnej, ale okazało się, że w rzeczywistości jest ona bardziej restrykcyjna od reguł postaci 3NF. Oznacza to, że każda relacja będąca w postaci BCNF jest również w postaci 3NF, jednak relacja będąca w postaci 3NF *niekoniecznie* jest w postaci BCNF. We wcześniejszym punkcie stwierdziliśmy, że choć 3NF dopuszcza zależności funkcyjne zgodne z warunkiem (b) z definicji 3NF, BCNF nie pozwala na takie zależności, dlatego jest bardziej restrykcyjną postacią normalną.

Intuicyjnie możemy stwierdzić, że potrzebna jest postać normalna silniejsza od postaci 3NF, analizując raz jeszcze relację DZIAŁKI z rysunku 14.12(a) z jej czterema zależnościami funkcyjnymi od FD1 do FD4. Załóżmy, że w relacji posiadamy tysiące działek, ale działki te należą tylko do dwóch gmin: Sandomierz i Dwikozy. Załóżmy również, że rozmiary działek w gminie Sandomierz wynoszą wyłącznie 5, 6, 7, 8, 9 i 10 arów, natomiast rozmiary działek w gminie Dwikozy wynoszą 11, 12, ..., 19 i 20 arów. W takiej sytuacji występuje dodatkowa zależność funkcyjna FD5: POWIERZCHNIA  $\rightarrow$  NAZWA\_GMINY. Jeżeli dodamy ją do innych zależności, schemat relacji DZIAŁKI1A wciąż pozostaje w trzeciej postaci normalnej, ponieważ zależność jest zgodna z punktem (b) z ogólnej definicji 3NF i NAZWA\_GMINY jest atrybutem podstawowym.

Powierzchnię działki determinującą gminę, zgodnie z zależnością FD5, można reprezentować za pomocą 16 krotek w oddzielnej relacji  $R(\text{POWIERZCHNIA}, \text{NAZWA\_GMINY})$ , gdyż istnieje tylko 16 różnych wartości atrybutu POWIERZCHNIA (zobacz rysunek 14.13). Taka reprezentacja redukuje nadmiarowość związaną z powtarzaniem tych samych informacji w tysiącach krotek relacji DZIAŁKI1A. Postać BCNF jest *silniejszą postacią normalną*, która nie pozwala na występowanie relacji DZIAŁKI1A i sugeruje potrzebę jej rozłożenia.

**Definicja.** Schemat relacji  $R$  znajduje się w **postaci normalnej Boyce'a-Codda** (BCNF), jeżeli zawsze, kiedy *nietrywialna* zależność funkcyjna  $X \rightarrow A$  jest zachowana w  $R$ , to  $X$  jest nadkluczem w  $R$ .

Formalna definicja postaci BCNF różni się nieco od definicji postaci 3NF. Różnicą między nimi jest niewystępowanie w BCNF warunku (b), który pozwala, aby atrybut  $A$  był podstawowy. To sprawia, że BCNF jest silniejszą postacią normalną od 3NF. W przedstawionym przykładzie zależność FD5 narusza warunki postaci BCNF w relacji DZIAŁKI1A, ponieważ atrybut POWIERZCHNIA nie jest nadkluczem tej relacji. Relację DZIAŁKI1A można rozłożyć na dwie relacje BCNF DZIAŁKI1AX oraz DZIAŁKI1AY, przedstawione na rysunku 14.13(a). Rozkład ten powoduje usunięcie zależności funkcyjnej FD2, ponieważ jej atrybuty, po rozłożeniu, nie występują już w tej samej relacji.

(a) DZIAŁKI1A

ID_NIERUCHOMOŚCI#	NAZWA_GMINY	DZIAŁKA#	POWIERZCHNIA
-------------------	-------------	----------	--------------



Normalizacja do postaci  
normalnej Boyce'a-Codda

DZIAŁKI1AX

ID_NIERUCHOMOŚCI#	POWIERZCHNIA	DZIAŁKA#
-------------------	--------------	----------

DZIAŁKI1AY

POWIERZCHNIA	NAZWA_GMINY
--------------	-------------

(b)  $R$

A	B	C
---	---	---



RYСУNEK 14.13. Postać normalna Boyce'a-Codda. (a) Normalizacja do postaci BCNF relacji DZIAŁKI1A polegająca na usunięciu zależności funkcyjnej FD2 w dekompozycji. (b) Relacja schematyczna z zależnościami funkcyjnymi. Znajduje się ona w trzeciej postaci normalnej, ale nie w postaci BCNF (powodem jest zależność funkcyjna  $C \rightarrow B$ ).

W praktyce większość schematów relacji, które są w postaci 3NF, jest również w postaci BCNF. Tylko wtedy, gdy zależność  $X \rightarrow A$  jest zachowana w schemacie relacji  $R$ , a  $X$  nie jest nadkluczem oraz  $A$  jest atrybutem podstawowym, relacja  $R$  znajduje się w trzeciej postaci normalnej, ale nie w postaci normalnej Boyce'a-Codda. Schemat relacji  $R$  przedstawiony na rysunku 14.13(b) ilustruje przypadek ogólny takiej relacji. Widoczna tam zależność funkcyjna może skutkować nadmiarowością danych, co ilustruje zależność FD5,  $POWIERZCHNIA \rightarrow NAZWA\_GMINY$ , w relacji DZIAŁKI1A. W idealnej sytuacji projekt relacyjnej bazy danych powinien osiągać postać BCNF lub 3NF dla każdego schematu relacji. Osiągnięcie stanu normalizacji tylko na poziomie postaci 1NF lub 2NF nie jest uważane za odpowiednie, gdyż historycznie zostały one zdefiniowane tylko jako etapy przekształceń wiodących do osiągnięcia postaci normalnych 3NF lub BCNF.

### 14.5.1. Dekompozycja relacji niebędących w BCNF

Jako kolejny przykład rozważmy rysunek 14.14, na którym przedstawiono relację UCZY o następujących zależnościach:

FD1: {STUDENT, PRZEDMIOT} → WYKŁADOWCA  
 FD2<sup>14</sup>: WYKŁADOWCA → PRZEDMIOT

UCZY

STUDENT	PRZEDMIOT	WYKŁADOWCA
Głębocki	Bazy danych	Robakiewicz
Nowak	Bazy danych	Nowosielski
Nowak	Systemy operacyjne	Anielewicz
Nowak	Teoria	Szuman
Owsiak	Bazy danych	Robakiewicz
Owsiak	Systemy operacyjne	Ostrowski
Kowalski	Bazy danych	Omieciński
Zielińska	Bazy danych	Nowosielski
Głębocki	Systemy operacyjne	Anielewicz

RYСУNEK 14.14. Relacja UCZY będąca w trzeciej postaci normalnej 3NF, ale nie w postaci normalnej Boyce'a-Codda

Należy zauważyć, że zależność {STUDENT, PRZEDMIOT} stanowi klucz kandydujący dla tej relacji oraz że przedstawione zależności są zgodne z wzorcem przedstawionym na rysunku 14.13(b), gdzie atrybut STUDENT to *A*, PRZEDMIOT to *B*, zaś WYKŁADOWCA — *C*. Zatem relacja ta znajduje się w trzeciej postaci normalnej, ale nie w postaci normalnej BCNF. Rozkład jej schematu na dwa schematy nie jest oczywisty, gdyż można tego dokonać, tworząc następujące trzy możliwe pary:

1. R1 {STUDENT, WYKŁADOWCA} i R2 {STUDENT, PRZEDMIOT}.
2. R1 {PRZEDMIOT, WYKŁADOWCA} i R2 {PRZEDMIOT, STUDENT}.
3. R1 {WYKŁADOWCA, PRZEDMIOT} i R2 {WYKŁADOWCA, STUDENT}.

Wszystkie trzy rozkłady wiążą się z „utrata” zależności funkcyjnej FD1. Należy zadać tu pytanie: który z tych trzech punktów przedstawia *pożądany rozkład*? Wcześniej (w punkcie 14.3.1) stwierdziliśmy, że w procesie normalizacji staramy się zapewnić dwie cechy dekompozycji: właściwość złączenia nieaddytywnego i właściwość zachowania zależności funkcyjnych. Żaden z przedstawionych rozkładów BCNF nie pozwala zachować zależności funkcyjnych, musimy jednak zapewnić właściwość złączenia nieaddytywnego. Pomocny jest tu prosty test sprawdzający rozkład binarny relacji na dwie inne relacje.

**Test złączenia nieaddytywnego dla dekompozycji binarnych.** Dekompozycja  $D = \{R_1, R_2\}$  relacji  $R$  posiada właściwość złączenia bezstratnego (nieaddytywnego) w kontekście zbioru zależności funkcyjnych  $F$  na relacji  $R$  wtedy i tylko wtedy, gdy jest spełniony jeden z poniższych warunków:

<sup>14</sup> Zależność FD2 oznacza, że „każdy wykładowca uczy jednego przedmiotu” to więzy w omawianym przypadku.

- zależność funkcyjna  $((R_1 \cap R_2) \rightarrow (R_1 - R_2))$  należy do domknięcia  $F^{+15}$ ;
- zależność funkcyjna  $((R_1 \cap R_2) \rightarrow (R_2 - R_1))$  należy do domknięcia  $F^+$ .

Jeśli zastosujemy ten test do trzech przedstawionych wcześniej dekompozycji, odkryjemy, że przejdzie go tylko trzecia z nich. W trzeciej dekompozycji  $R_1 \cap R_2$  to WYKŁADOWCA, a  $R_2 - R_1$  to PRZEDMIOT. Ponieważ  $WYKŁADOWCA \rightarrow PRZEDMIOT$ , test jest spełniony, a dekompozycja jest nieaddytywna (jako ćwiczenie dla Czytelników pozostawiamy wykazanie, że dwie pierwsze dekompozycje nie przechodzą tego testu). Dlatego poprawna dekompozycja relacji UCZY na relacje zgodne z BCNF to:

UCZY1(WYKŁADOWCA, PRZEDMIOT) i UCZY2(WYKŁADOWCA, STUDENT)

Należy się upewnić, że ta właściwość jest spełniona, ponieważ w trakcie normalizacji dekompozycja nieaddytywna jest koniecznością. Powinieneś sprawdzić, czy ta właściwość zostaje zachowana w kontekście nieformalnych kolejnych przykładów normalizacji z podrozdziałów 14.3 i 14.4 oraz w trakcie dekompozycji relacji DZIAŁKI1A na dwie relacje zgodne z BCNF: DZIAŁKI1AX i DZIAŁKI1AY.

Relację  $R$ , która nie jest w BCNF, można rozłożyć w sposób zgodny z właściwością złączenia nieaddytywnego za pomocą opisanej dalej procedury<sup>16</sup>. Dzieli ona  $R$  stopniowo na zbiór relacji w postaci BCNF:

Niech  $R$  będzie relacją niebędącą w postaci BCNF,  $X \subseteq R$ , a  $X \rightarrow A$  to zależność funkcyjna powodująca naruszenie BCNF.  $R$  można rozdzielić na dwie relacje:

$R-A$

$XA$

Jeśli  $R-A$  lub  $XA$  nie są w postaci BCNF, proces należy powtórzyć.

Czytelnik powinien się przekonać, że po zastosowaniu tej procedury do relacji DZIAŁKI1A uzyska (tak jak wcześniej) relacje DZIAŁKI1AX i DZIAŁKI1AY. Podobnie posłużenie się tą procedurą do relacji UCZY powinno dać relacje UCZY1 i UCZY2.

Zauważ, że jeśli zastosujemy parę (STUDENT, WYKŁADOWCA) jako klucz główny relacji UCZY, to zależność funkcyjna  $WYKŁADOWCA \rightarrow PRZEDMIOT$  spowoduje częściową (nie w pełni funkcyjną) zależność przedmiotu od tej części klucza. Tę zależność funkcyjną można usunąć w ramach drugiej normalizacji (lub w wyniku bezpośredniego zastosowania opisanej procedury w celu uzyskania BCNF), co w efekcie da te same dwie relacje. Jest to przykład sytuacji, w której za pomocą różnych ścieżek normalizacji można otrzymać ten sam ostateczny projekt zgodny z BCNF.

<sup>15</sup> Zapis  $F^+$  oznacza pokrycie zbioru zależności funkcyjnych i obejmuje wszystkie zależności wynikające z  $F$ . Zagadnienie to jest szczegółowo opisane w podrozdziale 15.1. Tu wystarczy się upewnić, że jedna z dwóch zależności funkcyjnych zostaje zachowana, aby dekompozycja nieaddytywna na relacje  $R_1$  i  $R_2$  była zgodna z testem.

<sup>16</sup> Zauważ, że jest to procedura oparta na algorytmie 15.5 z rozdziału 15., tworzącym schemat BCNF w wyniku dekompozycji dowolnego schematu.

## 14.6. Zależności wielowartościowe i czwarta postać normalna

Weźmy na przykład pod uwagę relację *PRAC* z rysunku 14.5(a). Krotka tej relacji reprezentuje fakt, że pracownik, którego nazwiskiem jest *NAZWPRAC*, pracuje nad projektem, którego nazwą jest *NAZWAPROJ*, i posiada członka rodziny, którego imieniem jest *IMIĘCZŁRODZ*. Pracownik może pracować nad kilkoma projektami i posiadać wielu członków rodziny, zaś jego projekty i członkowie rodziny to wartości niezależne od siebie<sup>17</sup>. W celu zachowania spójności stanu relacji musimy posiadać oddzielną krotkę dla reprezentowania każdej kombinacji członków rodziny pracownika z jego projektami. Na rysunku 14.5(a) w relacji *PRAC* pracownik o nazwisku *Nowak* pracuje nad projektami o nazwach *X* i *Y* oraz posiada dwóch członków rodziny o imionach *Jan* i *Anna*. Dlatego do przedstawienia tych faktów potrzebne są cztery krotki. Relacja *PRAC* jest **relacją o kluczu całościowym** (składającym się z wszystkich atrybutów), nie występują w niej więc zależności funkcyjne i jest to relacja BCNF. Widać tu oczywistą nadmiarowość w relacji *PRAC*: informacje o członkach rodziny są powtarzane dla każdego projektu, a informacje o projekcie — dla każdego członka rodziny.

Relacja *PRAC* pokazuje, że niektóre relacje posiadają więzy, których nie da się wyrazić za pomocą zależności funkcyjnych, dlatego nie naruszają one postaci BCNF. Na potrzeby takiej sytuacji zaproponowano *zależności wielowartościowe* (ang. *multivalued dependency*, MVD) oraz zdefiniowano *czwartą postać normalną* (ang. *fourth normal form*), która bazuje na tego rodzaju zależnościach. Formalne omówienie MVD i ich cech znajdziesz w rozdziale 15. Zależności wielowartościowe stanowią pochodną pierwszej postaci normalnej (1NF) — patrz punkt 14.3.4 — która nie pozwala na występowanie w krotce atrybutu, który posiadałby *zbiór wartości*. Jeśli występuje więcej niż jeden atrybut wielowartościowy, to druga technika normalizacji (patrz punkt 14.3.4) powoduje zależność wielowartościową. Nieformalnie można stwierdzić, że zawsze wtedy, gdy dwie *niezależne* relacje  $A:B$  i  $A:C$  typu 1:N występują w tej samej relacji  $R(A, B, C)$ , mogą pojawić się MVD<sup>18</sup>.

### 14.6.1. Formalna definicja zależności wielowartościowej

**Definicja.** Zależność wielowartościowa  $X \twoheadrightarrow Y$  określona na relacji  $R$ , gdzie  $X$  i  $Y$  są podzbiorami relacji  $R$ , określa następujące więzy na dowolnym stanie  $r$  relacji  $R$ : jeżeli w  $r$  istnieją dwie krotki  $t_1$  i  $t_2$ , takie że  $t_1[X] = t_2[X]$ , to w  $r$  powinny również istnieć dwie krotki  $t_3$  i  $t_4$  o następujących właściwościach<sup>19</sup>, gdzie symbol  $Z$  oznacza zbiór  $(R - (X \cup Y))$ <sup>20</sup>:

<sup>17</sup> W przypadku diagramu ER byłyby one reprezentowane jako atrybuty wielowartościowe lub jako typy encji słabej (patrz rozdział 3.).

<sup>18</sup> Taka MVD jest zapisywana za pomocą notacji  $A \twoheadrightarrow B|C$ .

<sup>19</sup> Krotki  $t_1$ ,  $t_2$ ,  $t_3$  i  $t_4$  nie muszą być różne.

<sup>20</sup> Symbol  $Z$  stanowi skrócony zapis atrybutów pozostałych w relacji  $R$  po usunięciu z niej atrybutów należących do zbioru  $(X \cup Y)$ .



RYСУNEK 14.15. Czwarta i piąta postać normalna.

- (a) Relacja PRAC z dwiema zależnościami wielowartościowymi:  $NAZWPRAC \twoheadrightarrow NAZWAPROJ$  oraz  $NAZWPRAC \twoheadrightarrow IMIĘCZŁRODZ$ .
- (b) Dekompozycja relacji PRAC na dwie relacje w czwartej postaci normalnej PRAC\_PROJEKTY i PRAC\_CZŁRODZINY.
- (c) Relacja DOSTAWA nieposiadająca zależności wielowartościowych znajduje się w czwartej postaci normalnej, ale nie w piątej postaci normalnej, jeżeli posiada zależność złączenia  $JD(R_1, R_2, R_3)$ .
- (d) Dekompozycja relacji DOSTAWA na relacje w piątej postaci normalnej  $R_1, R_2, R_3$
- $t_3[X] = t_4[X] = t_1[X] = t_2[X]$ ;
  - $t_3[Y] = t_1[Y]$  oraz  $t_4[Y] = t_2[Y]$ ;
  - $t_3[Z] = t_2[Z]$  oraz  $t_4[Z] = t_1[Z]$ .

Kiedy jest zachowana zależność  $X \twoheadrightarrow Y$ , mówimy, że  $X$  **determinuje wielokrotnie** (ang. *multidetermines*)  $Y$ . Ze względu na symetryczność, jaka występuje w definicji, kiedy w relacji  $R$  zachodzi  $X \twoheadrightarrow Y$ , zachodzi również  $X \twoheadrightarrow Z$ . Stąd  $X \twoheadrightarrow Y$  implikuje, że  $X \twoheadrightarrow Z$ , co czasem zapisuje się jako  $X \twoheadrightarrow Y \mid Z$ .

Zależność wielowartościową  $X \twoheadrightarrow Y$  w relacji  $R$  określa się mianem **banalnej** zależności wielowartościowej, jeżeli (a)  $Y$  jest podzbiorem  $X$  lub (b)  $X \cup Y = R$ . Przykładowo, relacja PRAC\_PROJEKTY z rysunku 14.15(b) posiada banalną zależność wielowartościową  $NAZWPRAC \twoheadrightarrow NAZWAPROJ$ , a relacja PRAC\_PODWŁADNI — banalną zależność wielowartościową  $NAZWPRAC \twoheadrightarrow IMIĘCZŁRODZ$ . Zależność wielowartościowa nie spełniająca ani warunku (a), ani (b) nosi nazwę **niebanalnej** zależności wielowartościowej. Banalna zależność wielowartościowa zachodzi w *dowolnym* stanie  $r$  relacji  $R$ . Określa się ją mianem banalnej, ponieważ nie określa żadnych istotnych lub znaczących więzów na relacji  $R$ .



Jeżeli w relacji występuje *niebanalna* zależność wielowartościowa, może okazać się, że w krotkach trzeba powtarzać nadmiarowe wartości. W relacji PRAC z rysunku 14.15(a) wartości  $X$  i  $Y$  atrybutu NAZWAPROJ są powtarzane dla każdej wartości atrybutu IMIĘCZŁRODZ (lub, analogicznie, wartości *Jan* i *Anna* atrybutu IMIĘCZŁRODZ są powtarzane dla każdej wartości atrybutu NAZWAPROJ). Taka nadmiarowość jest oczywiście niepożądana. Jednakże schemat PRAC znajduje się w postaci BCNF, ponieważ w relacji PRAC nie jest zachowana żadna zależność funkcyjna. Zatem musimy zdefiniować czwartą postać normalną, która będzie bardziej rygorystyczna od postaci BCNF i będzie zabraniała używania schematów relacji takich jak PRAC. Należy zauważyć, że relacje zawierające niebanalne zależności wielowartościowe zwykle są **relacjami o kluczu całościowym** (ang. *all-key relations*), czyli takimi, w których przypadku klucz składa się ze wszystkich atrybutów relacji. Ponadto w praktyce rzadko się zdarza, by projektowane były tego rodzaju relacje o kluczu całościowym z kombinatoryczną liczbą wystąpień powtarzających się wartości. Jednak wykrywanie zależności wielowartościowych (jako potencjalnego źródła problemów) jest w trakcie projektowania relacyjnego bardzo ważne.

Poniżej zostanie zaprezentowana definicja **czwartej postaci normalnej** (ang. *fourth normal form, 4NF*), która zostaje naruszona w sytuacji, gdy relacja posiada niepożądane zależności wielowartościowe, a przez to może być używana do identyfikowania i dekompozycji takich relacji.

**Definicja.** Schemat relacji  $R$  znajduje się w **czwartej postaci normalnej** w kontekście zbioru zależności  $F$  (zawierającego zarówno zależności funkcyjne, jak i wielowartościowe), jeżeli dla każdej *niebanalnej* zależności wielowartościowej  $X \twoheadrightarrow Y$  w domknięciu  $F^{+21}$  zbiór  $X$  jest nadkluczem relacji  $R$ .

Można zauważyć następujące kwestie:

- Relacja z kluczem całościowym zawsze jest w postaci BCNF, ponieważ nie zawiera zależności funkcyjnych.
- Relacja z kluczem całościowym, taka jak PRAC z rysunku 14.15(a), która nie ma zależności funkcyjnych, ale obejmuje zależność wielowartościową  $\text{NAZWPRAC} \twoheadrightarrow \text{NAZWAPROJ} \mid \text{IMIĘCZŁRODZ}$ , nie jest w postaci 4NF.
- Relację, która z powodu niebanalnej zależności wielowartościowej nie jest w postaci 4NF, trzeba rozłożyć, aby przekształcić ją w zbiór relacji zgodnych z 4NF.
- Dekompozycja umożliwia wyeliminowanie nadmiarowości powodowanej zależnościami wielowartościowymi.

Proces normalizacji relacji obejmującej niebanalne zależności wielowartościowe, która nie jest w 4NF, polega na rozkładzie jej w taki sposób, by każda taka zależność była reprezentowana w odrębnej relacji, gdzie jest banalną zależnością wielowartościową. Relacja PRAC z rysunku 14.15(a) nie jest w czwartej postaci normalnej, ponieważ w przypadku niebanalnych zależności wielowartościowych  $\text{NAZWPRAC} \twoheadrightarrow \text{NAZWAPROJ}$  oraz  $\text{NAZWPRAC} \twoheadrightarrow \text{IMIĘCZŁRODZ}$  atrybut NAZWPRAC nie jest nadkluczem relacji PRAC. Relację PRAC rozkładamy na relacje PRAC\_PROJEKTY oraz PRAC\_CZŁRODZINY przedstawione na rysunku 14.15(b). Obie znajdują się w czwartej postaci normalnej, ponieważ zależności wielowartościowe

<sup>21</sup>  $F^+$  oznacza pokrycie zależności funkcyjnych  $F$ , czyli wszystkie zależności wynikające z  $F$  (patrz definicja w podrozdziale 15.1).



NAZWPRAC  $\rightarrow$  NAZWAPROJ w relacji PRAC\_PROJEKTY oraz NAZWPRAC  $\rightarrow$  IMIĘCZŁRODZ w relacji PRAC\_CZŁRODZINY są zależnościami banalnymi. Żadne inne niebanalne zależności wielowartościowe nie są zachowane w tych relacjach. Podobnie nie występują tu zależności funkcyjne.

## 14.7. Zależności złączeniowe i piąta postać normalna

Wcześniej opisaliśmy sprawiające problemy zależności funkcyjne i pokazaliśmy, jak je wyeliminować za pomocą wielokrotnej dekompozycji binarnej w procesie normalizacji do postaci 1NF, 2NF, 3NF i BCNF. W trakcie dekompozycji binarnej należy zachować właściwość złączenia nieaddytywnego; w podrozdziale 14.5, w ramach omawiania dekompozycji do postaci BCNF, pokazano test dotyczący tej właściwości. Uzyskanie 4NF zwykle powoduje także wyeliminowanie zależności wielowartościowych w wyniku wielokrotnej dekompozycji binarnej. Jednakże w pewnych sytuacjach może nie istnieć dekompozycja ze złączeniem nieaddytywnym relacji  $R$  na dwa schematy relacji, choć może istnieć taka dekompozycja na *więcej niż dwa* schematy. Ponadto, może nie występować zależność funkcyjna w relacji  $R$  naruszająca którąś z postaci normalnych aż do postaci BCNF i w relacji tej mogą nie występować żadne niebanalne zależności wielowartościowe naruszające postać 4NF. W takim przypadku odwołujemy się do kolejnej zależności, określanej mianem *zależności złączeniowej* (ang. *join dependency*) i w przypadku, gdy taka zależność występuje, przeprowadzamy *dekompozycję wielokierunkową* (ang. *multiway decomposition*), otrzymując **piątą postać normalną** (ang. *fifth normal form, 5FN*). Istotne jest spostrzeżenie, że taka zależność stanowi bardzo specyficzne więzy semantyczne, które w praktyce trudno jest wykryć. Stąd normalizację do piątej postaci normalnej przeprowadza się bardzo rzadko.

**Definicja. Zależność złączeniowa** (ang. *join dependency, JD*), oznaczana jako  $JD(R_1, R_2, \dots, R_n)$ , określona na schemacie relacji  $R$ , definiuje więzy na stanach  $r$  relacji  $R$ . Więzy te określają, że każdy dopuszczalny stan  $r$  relacji  $R$  powinien posiadać dekompozycję ze złączeniem nieaddytywnym na relacje  $R_1, R_2, \dots, R_n$ , to znaczy, że dla każdego takiego  $r$  mamy:

$$*(\pi_{R_1}(r), \pi_{R_2}(r), \dots, \pi_{R_n}(r)) = r$$

Należy zauważyć, że zależność wielowartościowa stanowi przypadek szczególny zależności złączeniowej dla  $n = 2$ . Oznacza to, że zależność złączeniowa określona jako  $JD(R_1, R_2)$  implikuje zależność wielowartościową  $(R_1 \cap R_2) \twoheadrightarrow (R_1 - R_2)$  lub, przez analogię,  $(R_1 \cap R_2) \twoheadrightarrow (R_2 - R_1)$ . Zależność złączeniowa  $JD(R_1, R_2, \dots, R_n)$ , określona na schemacie relacji  $R$ , jest **banalną** zależnością złączeniową, jeżeli jeden ze schematów relacji  $R_i$  w zależności  $JD(R_1, R_2, \dots, R_n)$  jest równy  $R$ . Taka zależność jest określana mianem **banalnej**, ponieważ posiada właściwość złączenia nieaddytywnego dla dowolnego stanu  $r$  relacji  $R$ , a stąd nie określa żadnych więzów na relacji  $R$ . Możemy teraz zdefiniować piątą postać normalną, którą określa się również mianem *postaci normalnej złączenia z projekcją* (ang. *project-join normal form, PJNF*).

**Definicja.** Schemat relacji  $R$  znajduje się w **piątej postaci normalnej** (5NF), czyli w **postaci normalnej złączenia z rzutowaniem**, w kontekście zbioru  $F$  zależności funkcyjnych, wielowartościowych oraz złączeniowych, jeżeli dla każdej niebanalnej zależności złączeniowej  $JD(R_1, R_2, \dots, R_n)$  w domknięciu  $F^+$  (to znaczy implikowanej przez elementy zbioru  $F$ )<sup>22</sup> każda relacja  $R_i$  jest nadkluczem relacji  $R$ .

Jako przykład zależności złączeniowej weźmy raz jeszcze pod uwagę relację DOSTAWA z rysunku 14.15(c). Załóżmy, że zawsze są zachowane następujące więzy: kiedy dostawca  $d$  dostarcza część  $c$  oraz projekt  $p$  wykorzystuje część  $c$  oraz dostawca  $d$  dostarcza co najmniej jedną część dla projektu  $p$ , to dostawca  $d$  będzie również dostarczał część  $c$  dla projektu  $p$ . Więzy te można przeformułować i reprezentują one zależność złączeniową  $JD(R_1, R_2, R_3)$  wśród trzech projekcji  $R_1(\text{NAZWADOST}, \text{NAZWACZ})$ ,  $R_2(\text{NAZWADOST}, \text{NAZWAPROJ})$  oraz  $R_3(\text{NAZWACZ}, \text{NAZWAPROJ})$  relacji DOSTAWA. W przypadku, gdy więzy takie występują, krotki znajdujące się pod linią przerywaną na rysunku 14.15(c) muszą występować w każdym dopuszczalnym stanie relacji DOSTAWA, który zawiera również krotki znajdujące się nad linią przerywaną. Na rysunku 14.15(d) przedstawiono sposób dekompozycji relacji DOSTAWA z zależnościami złączeniową na trzy relacje  $R_1$ ,  $R_2$  oraz  $R_3$ , które znajdują się w piątej postaci normalnej. Należy zauważyć, że zastosowanie złączenia naturalnego (NATURAL JOIN) względem dowolnych dwóch z tych relacji powoduje występowanie fałszywych krotek, jednak zastosowanie złączenia naturalnego względem wszystkich trzech razem — nie. Czytelnik powinien sprawdzić to na przykładowej relacji z rysunku 14.15(c) oraz jej projekcji z rysunku 14.15(d). Dzieje się tak dlatego, że istnieje tylko zależność złączeniowa, natomiast nie ma żadnej zależności wielowartościowej. Warto również zauważyć, że zależność złączeniowa  $JD(R_1, R_2, R_3)$  jest określona na wszystkich dopuszczalnych stanach relacji, a nie tylko na przedstawionym na rysunku 14.15(c).

Znajdowanie zależności złączeniowych w rzeczywistych bazach danych liczących setki atrybutów jest niemal niemożliwe. Można tego dokonać tylko dzięki ogromnej intuicji projektanta względem danych. Dlatego też bieżące standardy projektowania baz danych przywiązują do nich niewielką wagę. Zdaniem Date'a i Fagina (1992) efektami tego są m.in. wykrywanie warunków z użyciem samych zależności funkcyjnych i całkowite ignorowanie zależności złączeniowych. Piszą oni: „Jeśli schemat relacji znajduje się w 3NF, a wszystkie klucze obejmują tylko jeden atrybut, schemat jest też w 5NF”.

## 14.8. Podsumowanie

W niniejszym rozdziale zostały omówione pewne pułapki związane z projektowaniem relacyjnych baz danych przy wykorzystaniu argumentacji opierającej się na podejściu intuicyjnym. Zidentyfikowano w sposób nieformalny pewne wskaźniki poprawności schematu relacyjnego oraz przedstawiono nieformalne wskazówki dotyczące reguł poprawnego projektowania. Te wskazówki są oparte na starannym opracowaniu projektu koncepcyjnego za pomocą modelu ER lub EER i wykonaniu opisanego w rozdziale 9. odwzorowania encji i związków na relacje. Przestrzeganie tych wskazówek i eliminowanie nadmiarowości po-

<sup>22</sup> Także tu  $F^+$  oznacza pokrycie zależności funkcyjnych  $F$ , czyli wszystkie zależności wynikające z  $F$  (patrz definicja w podrozdziale 15.1).

zwalają uniknąć anomalii przy wstawianiu, usuwaniu i aktualizowaniu danych, a także generowania fałszywych danych. Zalecamy ograniczenie stosowania wartości null, ponieważ powodują one problemy w operacjach SELECT i JOIN oraz w agregacjach. Następnie przedstawiono pewne formalne pojęcia, które pozwalają na opracowywanie schematów relacyjnych według schematu zstępującego poprzez analizowanie pojedynczych relacji. Taki proces projektowania przez analizę i rozkład zdefiniowano dzięki wprowadzeniu pojęcia procesu normalizacji.

Zdefiniowano również pojęcie zależności funkcyjnej (jest ona podstawowym narzędziem analizowania schematów relacyjnych) oraz omówiono niektóre jej właściwości. Zależności funkcyjne określają więzy semantyczne występujące między atrybutami schematu relacji. Następnie opisano proces normalizacji służący osiągnięciu poprawnych projektów poprzez sprawdzanie relacji pod względem występowania niepożądanych typów *problematicznych* zależności funkcyjnych. Omówiono sposób przeprowadzania stopniowej normalizacji w oparciu o predefiniowany klucz główny w każdej relacji, a następnie nieco liberalizowano ten wymóg i przedstawiono bardziej ogólne definicje drugiej (2NF) i trzeciej (3NF) postaci normalnej, które uwzględniają wszystkie klucze kandydujące relacji. Zaprezentowano również przykłady ilustrujące sposób analizy i rozkładu danej relacji przy użyciu definicji ogólnej trzeciej postaci normalnej w celu otrzymania zbioru relacji znajdujących się w tej postaci.

Dalej omówiono postać normalną Boyce'a-Codda (BCNF) i wyjaśniono, w jaki sposób jest ona bardziej rygorystyczna od postaci 3NF. Zilustrowano również sposób rozkładu relacji niebędącej w postaci BCNF poprzez uwzględnienie wymogu rozkładu nieaddytywanego. Pokazano też test na właściwość złączania nieaddytywnego w dekompozycji binarnej i ogólny algorytm przekształcania dowolnej relacji niebędącej w postaci BCNF na zbiór relacji zgodnych z tą postacią. Wyjaśniliśmy potrzebę stosowania dodatkowych ograniczeń (obok zależności funkcyjnych) wynikającą z umieszczenia niezależnych atrybutów wielowartościowych w jednej relacji. Wprowadziliśmy zależność wielofunkcyjną, która pozwala radzić sobie z takimi sytuacjami, i zdefiniowaliśmy opartą na zależności wielofunkcyjnej czwartą postać normalną. Na zakończenie opisaliśmy piątą postać normalną, która jest oparta na zależności złączeniowej i wprowadza nietypowe ograniczenie związane z dekompozycją relacji na kilka składników w taki sposób, by po złączeniu zawsze otrzymać pierwotną relację. W praktyce większość komercyjnych projektów jest zgodna z postaciami normalnymi do poziomu BCNF. Wówczas rzadko potrzebna jest dekompozycja do 5NF, a w większości sytuacji trudno jest zidentyfikować zależności złączeniowe. Dlatego 5NF ma przede wszystkim wartość teoretyczną.

W rozdziale 15. zostaną przedstawione algorytmy syntezy oraz dekompozycji dla projektów relacyjnych baz danych oparte na zależnościach funkcyjnych. W związku z zagadnieniem dekompozycji zostaną również omówione pojęcia *złączenia bezstratnego* (*nieaddytywnego*) oraz *zachowania zależności*, które są wymuszane przez niektóre z tych algorytmów. Pozostałe zagadnienia poruszane w rozdziale 11. to m.in. szczegółowe omówienie zależności funkcyjnych i wielowartościowych oraz dodatkowych rodzajów zależności.

## Pytania powtórkowe

- 14.1. Omów semantykę atrybutów jako nieformalny wskaźnik poprawności schematu relacji.
- 14.2. Omów anomalie wstawiania, usuwania i modyfikowania. Dlaczego uważa się je za objawy problemów związanych ze schematami? Przedstaw przykłady.
- 14.3. Dlaczego należy za wszelką cenę unikać występowania wartości null w relacjach? Omów problem fałszywych krotek oraz sposoby zapobiegania ich występowaniu.
- 14.4. Wymień omówione w tym rozdziale nieformalne wskazówki dotyczące projektowania schematów relacji. Przedstaw przykłady obrazujące, w jaki sposób naruszenia tych wskazówek mogą być szkodliwe.
- 14.5. Co to jest zależność funkcyjna? Jakie są możliwe źródła informacji definiujące zależności funkcyjne, które są zachowane między atrybutami schematu relacji?
- 14.6. Dlaczego zależności funkcyjnych nie da się określić w sposób automatyczny na podstawie określonego stanu relacji?
- 14.7. Do czego odnosi się pojęcie *relacji nieznormalizowanej*? W jaki sposób, z historycznego punktu widzenia, były opracowywane postaci normalne — od pierwszej postaci normalnej do postaci normalnej Boyce'a-Codda?
- 14.8. Zdefiniuj pierwszą, drugą oraz trzecią postać normalną w przypadku, gdy uwzględniane są tylko klucze główne. W jaki sposób definicje ogólne drugiej i trzeciej postaci normalnej, uwzględniające wszystkie klucze relacji, różnią się od definicji uwzględniających tylko klucze główne?
- 14.9. Jakich niepożądanych zależności unika się w przypadku, gdy relacja znajduje się w drugiej postaci normalnej?
- 14.10. Jakich niepożądanych zależności unika się w przypadku, gdy relacja znajduje się w trzeciej postaci normalnej?
- 14.11. W jaki sposób uogólnione definicje 2NF i 3NF stanowią rozwinięcie podstawowych definicji i wykraczają poza uwzględnianie kluczy głównych?
- 14.12. Zdefiniuj *postać normalną Boyce'a-Codda*. W jaki sposób różni się ona od trzeciej postaci normalnej? Dlaczego jest uważana za bardziej rygorystyczną formę trzeciej postaci normalnej?
- 14.13. Czym jest zależność wielowartościowa? Kiedy powstaje?
- 14.14. Czy relacja o przynajmniej dwóch kolumnach zawsze obejmuje zależność wielowartościową? Przedstaw przykład.
- 14.15. Zdefiniuj *czwartą postać normalną*. Kiedy zostaje naruszona? W jakich typowych sytuacjach można ją zastosować?
- 14.16. Zdefiniuj *zależność złączeniową* i *piątą postać normalną*.
- 14.17. Dlaczego 5NF jest też nazywana *postacią normalną złączenia z projekcją*?
- 14.18. Dlaczego w praktyce w projektach baz danych zwykle dąży się do BCNF, a nie do wyższych postaci normalnych?

## Ćwiczenia

14.19. Załóżmy, że posiadamy następujące wymagania odnośnie do uniwersyteckiej bazy danych, służącej do przechowywania informacji o osiągnięciach edukacyjnych studentów:

- a) Uniwersytet przechowuje informacje o imieniu i nazwisku każdego studenta (IMIĘNAZWS), numerze studenta (NUMERS), numerze PESEL (PESEL), bieżącym adresie (ADRS) i bieżącym numerze telefonu (TELEFONS), adresie stałego zameldowania (STADRS) i numerze telefonu (STTELEFONS), dacie urodzenia (DATAUR), płci (PŁEĆ), roku studiów (ROK) (pierwszy rok, drugi rok, ..., absolwent), wydziale głównego kierunku studiów (KODGŁKIER), wydziale drugiego kierunku studiów (KODDRKIER) (o ile to konieczne) oraz charakterze studiów (CHARAKTER) (inżynierskie, magisterskie, doktoranckie itp.). Zarówno numer PESEL, jak i numer studenta posiadają unikatową wartość dla każdego ze studentów.
- b) Każdy wydział opisuje jego nazwa (NAZWAW), kod (KODW), numer budynku administracji (BUDW), numer telefonu (TELEFONW) oraz numer budynku dydaktycznego (DYDAKTW). Zarówno nazwa, jak i kod posiadają unikatowe wartości dla każdego wydziału.
- c) Każdy przedmiot zajęć posiada nazwę (NAZWAP), opis (OPISP), numer przedmiotu (NUMERP), liczbę godzin w semestrze (GODZINYP), poziom zaawansowania (POZIOM) oraz wydział prowadzący nauczanie (WYDZP). Numer przedmiotu jest unikatowy dla każdego z nich.
- d) Każda grupa zajęciowa posiada określonego wykładowcę (IMIĘNAZW), semestr (SEMESTR), rok (ROK), przedmiot zajęć (PRZEDZAJ) oraz numer grupy zajęciowej (ZAJNUM). Numery zajęć pozwalają na rozróżnienie między różnymi grupami zajęciowymi, w których wykładany jest ten sam przedmiot w tym samym semestrze i roku. Jego wartości to 1, 2, 3, ..., aż do całkowitej liczby grup zajęciowych występujących w danym semestrze.
- e) Rejestr ocen zawiera odniesienia do studenta (PESEL), określonej grupy zajęciowej oraz oceny (OCENA).

Zaprojektuj schemat relacyjnej bazy danych dla takiej aplikacji bazodanowej. Najpierw przedstaw wszystkie zależności funkcyjne, które powinny być zachowane między atrybutami. Następnie opracuj dla bazy danych schematy relacji, które będą się znajdować w trzeciej postaci normalnej lub w postaci normalnej Boyce'a-Codda. Określ atrybuty klucza każdej relacji. Zidentyfikuj wszelkie niewyspecyfikowane wymagania i podejmij względem nich odpowiednie założenia, które pozwolą na uzupełnienie specyfikacji.

14.20. Jakie anomalie aktualizacji występują w relacjach PRAC\_PROJ i PRAC\_WYDZ z rysunków 14.3 i 14.4?

14.21. W której postaci normalnej znajduje się relacja DZIAŁKI z rysunku 14.12(a) w kontekście restrykcyjnej interpretacji postaci normalnej, która uwzględnia wyłącznie klucz główny? Czy byłaby to ta sama postać normalna, gdyby chodziło o ogólne definicje postaci normalnych?

- 14.22. Udowodnij, że każdy schemat relacji o dwóch atrybutach znajduje się w postaci normalnej BCNF.
- 14.23. Dlaczego w wyniku złączenia relacji PRAC\_PROJ1 i PRAC\_LOKALIZACJE z rysunku 14.5 pojawiają się fałszywe krotki (wynik takiego złączenia przedstawia rysunek 14.6)?
- 14.24. Rozważ relację uniwersalną  $R = \{A, B, C, D, E, F, G, H, I, J\}$  oraz zbiór zależności funkcyjnych  $F = \{\{A, B\} \rightarrow \{C\}, \{A\} \rightarrow \{D, E\}, \{B\} \rightarrow \{F\}, \{F\} \rightarrow \{G, H\}, \{D\} \rightarrow \{I, J\}\}$ . Jaki jest klucz relacji  $R$ ? Rozłóż relację  $R$  na relacje w postaci 2NF oraz 3NF.
- 14.25. Powtórz ćwiczenie 14.24 dla następującego innego zbioru zależności funkcyjnych  $G = \{\{A, B\} \rightarrow \{C\}, \{B, D\} \rightarrow \{E, F\}, \{A, D\} \rightarrow \{G, H\}, \{A\} \rightarrow \{I\}, \{H\} \rightarrow \{J\}\}$ .
- 14.26. Przeanalizuj poniższą relację:

A	B	C	KROTKA#
10	b1	c1	#1
10	b2	c2	#2
11	b4	c1	#3
12	b3	c4	#4
13	b1	c1	#5
14	b3	c4	#6

- a) W przypadku podanego rozszerzenia (stanu), które z poniższych zależności *mogą być zachowane* dla powyższej relacji? Jeżeli dana zależność nie może być zachowana, wyjaśnij, dlaczego tak jest, *poprzez określenie krotek naruszających odpowiednie warunki*.
- i.  $A \rightarrow B$ , ii.  $B \rightarrow C$ , iii.  $C \rightarrow B$ , iv.  $B \rightarrow A$ , v.  $C \rightarrow A$ .
- b) Czy powyższa relacja posiada potencjalny klucz kandydujący? Jeśli tak, jaką ma on postać? Jeśli nie, dlaczego?
- 14.27. Rozważ relację  $R(A, B, C, D, E)$  o następujących zależnościach:
- $$AB \rightarrow C, CD \rightarrow E, DE \rightarrow B.$$

Czy  $AB$  jest kluczem kandydującym tej relacji? Jeśli nie, czy jest nim  $ABD$ ? Odpowiedź uzasadnij.

- 14.28. Rozważ relację  $R$ , która posiada atrybuty przechowujące informacje o planach zajęć i grupach zajęciowych na uniwersytecie;  $R = \{\text{NrPrzedm}, \text{NrGrupy}, \text{WydzProwadzący}, \text{LiczbaGodzin}, \text{PoziomZaawansowania}, \text{PESELWykładowcy}, \text{Semestr}, \text{Rok}, \text{Dni\_Godziny}, \text{NrSali}, \text{LiczbaStudentów}\}$ . Załóżmy, że w ramach relacji  $R$  zachodzą następujące zależności funkcyjne:

$\{\text{NrPrzedm}\} \rightarrow \{\text{WydzProwadzący}, \text{LiczbaGodzin}, \text{PoziomZaawansowania}\}$   
 $\{\text{NrPrzedm}, \text{NrGrupy}, \text{Semestr}, \text{Rok}\} \rightarrow \{\text{Dni\_Godziny}, \text{NrSali},$   
 $\rightarrow \text{LiczbaStudentów}, \text{PESELWykładowcy}\}$   
 $\{\text{NrSali}, \text{Dni\_Godziny}, \text{Semestr}, \text{Rok}\} \rightarrow \{\text{PESELWykładowcy}, \text{NrPrzedm},$   
 $\rightarrow \text{NrGrupy}\}$

Spróbuj określić, które zbiory atrybutów tworzą klucze relacji  $R$ . W jaki sposób znormalizowałbyś tę relację?

- 14.29. Rozważ poniższe relacje w kontekście bazy danych aplikacji przetwarzania zamówień w firmie ABC, Sp. z o.o.:

ZAMÓWIENIE (Z#, DataZ, Klient#, Kwota\_całk)

PROD\_ZAMÓWIENIA (Z#, P#, Ilość\_zamówiona, Cena\_całk, Obniżka%)

Założ, że każdy produkt posiada inną obniżkę cenową. Atrybut CENA\_CAŁK odnosi się do jednego produktu, DATAZ to data złożenia zamówienia, zaś KWOTA\_CAŁK to kwota całego zamówienia. Jeżeli względem relacji PROD\_ZAMÓWIENIA i ZAMÓWIENIE zastosujemy operację złączenia naturalnego w tej bazie danych, jak będzie wyglądał wynikowy schemat relacji, WYN? Jaki będzie jej klucz? Wymień zależności funkcyjne występujące w takiej relacji wynikowej. Czy znajduje się ona w postaci 2NF? A może 3NF? Dlaczego lub dlaczego nie? (W razie przyjęcia jakichś założeń, wymień je).

- 14.30. Rozważ poniższą relację:

SPRZEDAŻ\_SAMOCH(Samochód#, Data\_sprzedaży, Sprzedawca#, Prowizja%,  
↳ Kwota\_rabatu)

Założ, że samochód może zostać sprzedany przez wielu sprzedawców, stąd klucz główny ma postać {SAMOCHÓD#, SPRZEDAWCA#}. Dodatkowe zależności to:

Data\_sprzedaży → Kwota\_rabatu

oraz

Sprzedawca# → Prowizja%

Bazując na podanym kluczu głównym, czy ta relacja znajduje się w postaci 1NF, 2NF, czy 3NF? Dlaczego? W jaki sposób można by ją stopniowo znormalizować?

- 14.31. Rozważ poniższą relację związaną z wydanymi książkami:

KSIĄŻKA (Tytuł\_książki, Autor, Typ\_książki, Cena, Autor\_przynał, Wydawca)

Atrybut Autor\_przynał odnosi się do przynależności autora do związku literackiego. Przyjmijmy, że istnieją następujące zależności:

Tytuł\_książki → Wydawca, Typ\_książki

Typ\_książki → Cena

Autor → Autor\_przynał

- W jakiej postaci normalnej jest ta relacja? Odpowiedź uzasadnij.
- Stosuj normalizację do momentu, gdy nie będzie już można dalej rozkładać relacji. Uzasadnij wybór kolejnych dekompozycji.

- 14.32. To ćwiczenie polega na przekształceniu wymagań biznesowych na zależności. Wyobraź sobie relację DYSK\_TWARDY(NR\_SERYJNY, PRODUCENT, MODEL, PARTIA, POJEMNOŚĆ, SPRZEDAWCA). Każda krotka relacji DYSK\_TWARDY zawiera informacje na temat dysku twardego z unikatowym numerem seryjnym, wyprodukowanego przez danego producenta, o określonym numerze modelu, pochodzącego z określonej partii, z podaną pojemnością i oferowanego przez danego sprzedawcę. Przykładowo, krotka DYSK\_TWARDY('1978619', 'WesternDigital', 'A2235X', '765234', 500, 'CompUSA') oznacza, że firma WesternDigital wyprodukowała dysk twardy o numerze seryjnym 1978619 i numerze modelu



A2235X, pochodzący z partii 765234, mający pojemność 500 GB i sprzedawany przez firmę ComUSA.

Zapisz każdą z wymienionych zależności jako zależność funkcyjną:

- Producent i numer seryjny w unikatowy sposób identyfikują dysk.
- Numer modelu jest rejestrowany przez producenta, dlatego nie może zostać użyty przez inną firmę.
- Wszystkie dyski twarde z danej partii to ten sam model.
- Wszystkie dyski twarde jednego modelu danego producenta mają identyczną pojemność.

14.33. Wyobraź sobie następującą relację:

$R(\text{DOKTOR\#, PACJENT\#, DATA, DIAGNOZA, KOD\_LECZENIA, CENA})$

W tej relacji krotka opisuje wizytę pacjenta u lekarza i obejmuje też kod leczenia i opłatę. Załóżmy, że diagnoza jest ustalana (w unikatowy sposób) dla każdego pacjenta przez lekarza. Przyjmijmy też, że dla każdego kodu leczenia ustalona jest stała opłata (niezależna od pacjenta). Czy jest to relacja w 2NF? Odpowiedź uzasadnij i w razie potrzeby przeprowadź dekompozycję. Następnie wyjaśnij, czy potrzebna jest dalsza normalizacja do 3NF. Jeśli tak, przeprowadź ją.

14.34. Zastanów się nad następującą relacją:

$\text{SPRZEDAŻ\_SAMOCHODU}(\text{ID\_SAM, TYP\_OPCJI, CENA\_OPCJI\_KAT, DATA\_SPRZED,} \\ \rightarrow \text{CENA\_OPCJI\_RABAT})$

Ta relacja dotyczy opcjonalnego wyposażenia (np. tempomatu) instalowanego w samochodach sprzedawanych przez salon oraz obejmuje cenę tego wyposażenia (katalogową i po rabacie).

Jeśli  $\text{ID\_SAM} \rightarrow \text{DATA\_SPRZED}$ ,  $\text{TYP\_OPCJI} \rightarrow \text{CENA\_OPCJI\_KAT}$  oraz  $\text{ID\_SAM, TYP\_OPCJI} \rightarrow \text{CENA\_OPCJI\_RABAT}$ , wykaż za pomocą ogólnej definicji 3NF, że ta relacja nie znajduje się w tej postaci. Następnie na podstawie znajomości 2NF uzasadnij, dlaczego relacja nie jest zgodna nawet z 2NF.

14.35. Przyjrzyj się następującej relacji:

$\text{KSIĄŻKA}(\text{TYTUŁ, AUTOR, WYDANIE, ROK})$

Zawiera ona dane:

TYTUŁ	AUTOR	WYDANIE	ROK
Wprowadzenie do systemów baz danych	Navathe	4	2004
Wprowadzenie do systemów baz danych	Elmasri	4	2004
Wprowadzenie do systemów baz danych	Elmasri	5	2007
Wprowadzenie do systemów baz danych	Navathe	5	2007

- Na podstawie zdroworozsądkowego zrozumienia tych danych określ możliwe klucze kandydujące tej relacji.
- Uzasadnij, że w tej relacji występuje zależność wielowartościowa  $\{\text{KSIĄŻKA}\} \rightarrow \{\text{AUTOR}\}\{\text{WYDANIE, ROK}\}$ .
- Jak będzie wyglądać dekompozycja tej relacji na podstawie tej zależności? Podaj najwyższą postać normalną każdej z uzyskanych relacji.

14.36. Przyjrzyj się następującej relacji:

PODRÓŻ(ID\_PODRÓŻY, DATA\_ROZP, ODWIEDZ\_MIASTA, UŻYTE\_KARTY)

Ta relacja dotyczy podróży biznesowych sprzedawców z danej firmy. Załóżmy, że podróż ma jedną datę rozpoczęcia, ale odwiedzanych jest wiele miast i sprzedawcy mogą używać wielu kart kredytowych. Przygotuj fikcyjne dane do tej tabeli:

- a) Określ, jakie zależności funkcyjne i/lub zależności wielowartościowe występują w tej relacji.
- b) Pokaż, jak przeprowadziłbyś normalizację tej relacji.

## Ćwiczenia laboratoryjne

*Uwaga:* w tych ćwiczeniach używany jest system DBD (ang. *Data Base Designer*) opisany w podręczniku do ćwiczeń laboratoryjnych.

Schemat relacji  $R$  i zbiór zależności funkcyjnych  $F$  należy zapisać w postaci list. Oto przykład:  $R$  i  $F$  z tego ćwiczenia są zapisywane w następującej postaci:

$R = [a, b, c, d, e, f, g, h, i, j]$

$F = [[a, b], [c]],$

$[[a], [d, e]],$

$[[b], [f]],$

$[[f], [g, h]],$

$[[d], [i, j]]]$

Ponieważ system DBD jest zaimplementowany w języku Prolog, nazwy pisane wielkimi literami są zarezerwowane dla zmiennych. Dlatego do zapisu atrybutów używane są małe litery. Więcej informacji o systemie DBD znajdziesz w podręczniku do ćwiczeń laboratoryjnych.

14.37. Za pomocą systemu DBD sprawdź odpowiedzi, jakich udzieliłeś w następujących ćwiczeniach:

- a) 14.24 (tylko dla 3NF).
- b) 14.25.
- c) 14.27.
- d) 14.28.

## Wybrane publikacje

Zależności funkcjonalne jako pierwszy opisał Codd (1970). Oryginalne definicje pierwszej, drugiej i trzeciej postaci normalnej również znalazły się w pozycji Codda (1972a), gdzie można także znaleźć omówienie anomalii aktualizacji. Postać normalna Boyce'a-Codda została zdefiniowana w pozycji Codda (1974). Alternatywną definicję trzeciej postaci normalnej podał Ullman (1988), podobnie jak definicję postaci BCNF przedstawioną w niniejszej

książce. Pozycje Ullmana (1988), Maiera (1983) oraz Atzeniego i De Antonellisa (1993) zawierają wiele twierdzeń i dowodów dotyczących zależności funkcyjnych. Date i Fagin (1992) przedstawili proste i praktyczne informacje związane z wyższymi postaciami normalnymi.

Dodatkowe informacje związane z teorią projektowania relacyjnego zostaną podane w rozdziale 15.



## Algorytmy projektowania relacyjnych baz danych i dodatkowe zależności

W rozdziale 14. przedstawiono technikę **relacyjnego projektowania zstępującego** i powiązane metody często wykorzystywane w komercyjnym projektowaniu baz danych. Ta technika polega na opracowaniu schematu koncepcyjnego ER lub EER, a następnie odwzorowaniu takiego schematu na model relacyjny przy użyciu procedur odwzorowujących, takich jak omówione w rozdziale 9. Do każdej relacji na podstawie zależności funkcyjnych przypisywane są klucze główne. W wykonywanym później procesie, który można nazwać **projektowaniem relacyjnym na podstawie analiz**, relacje zaprojektowane wcześniej za pomocą wspomnianej procedury (albo „odziedziczone” ze starszych plików, formularzy lub innych źródeł) są analizowane w celu wykrycia niepożądanych zależności funkcyjnych. Te zależności są usuwane za pomocą opisanej w podrozdziale 14.3 procedury normalizacji; omówiono tam też definicje powiązanych postaci normalnych, będących coraz lepszymi wersjami projektu poszczególnych relacji. W podrozdziale 14.3 przyjęliśmy, że klucze główne zostały przypisane do poszczególnych relacji. W rozdziale 14.4 zaprezentowano ogólniejsze podejście do normalizacji, w którym w każdej relacji uwzględniono wszystkie klucze kandydujące. W podrozdziale 14.5 omówiono kolejną postać normalną, BCNF. Dalej, w podrozdziałach 14.6 i 14.7, opisano dwa dodatkowe rodzaje zależności: wielowartościowe i złączeniowe, które także mogą powodować nadmiarowość. Pokazaliśmy, jak można je wyeliminować za pomocą dalszej normalizacji.

W tym rozdziale wykorzystamy opracowaną w poprzednim rozdziale teorię postaci normalnych i zależności funkcyjnych, wielowartościowych i złączeniowych oraz rozwinie my ją w trzech różnych obszarach. Po pierwsze, omówimy generowanie nowych zależności funkcyjnych na podstawie ich zbioru i przedstawimy zagadnienia takie jak domknięcie, pokrycie, pokrycie minimalne i równoważność. Na poziomie koncepcyjnym trzeba uwzględnić wszystkie atrybuty relacji w zwięzły sposób, a pokrycie minimalne to umożliwia. Po drugie, omówimy pożądane właściwości złączeń nieaddytywnych (bezstratnych) i zachowanie zależności funkcyjnych. Zaprezentujemy też ogólny algorytm sprawdzania nieaddytywności złączeń w zbiorze relacji. Po trzecie, zaprezentujemy **projektowanie relacyjne przez syntezę** zależności funkcyjnych. Jest to **projektowanie metodą wstępującą**, oparte na założeniu, że na wejściu określone są znane zależności funkcyjne ze zbioru atrybutów z dziedziny problemu. Prezentujemy tu algorytmy pozwalające osiągnąć pożądane postaci normalne: 3NF i BCNF, lub zapewnić oczekiwane właściwości nieaddytywności złączeń i zachowania zależności funkcyjnych. Choć metoda oparta na syntezie jest teoretycznie atrakcyjnym podejściem formalnym, w praktyce nie jest stosowana do projektowania dużych baz danych, ponieważ trudno jest z góry (przed rozpoczęciem projektowania) określić wszystkie możliwe zależności funkcyjne. Inna

metoda to podejście opisane w rozdziale 14., w którym kolejne dekompozycje i poprawki projektu są łatwiejsze do wprowadzania oraz możliwe jest modyfikowanie projektu. Ostatnim celem w tym rozdziale jest dokładne omówienie wprowadzonych w rozdziale 14. zależności wielowartościowych i krótki opis innych rodzajów zależności.

W podrozdziale 15.1 omówimy reguły określania zależności funkcyjnych i na ich podstawie zdefiniujemy pokrycie, równoważność i pokrycie minimalne dla zależności funkcyjnych. W podrozdziale 15.2 najpierw opiszemy dwie pożądane **właściwości dekompozycji** (ang. *properties of decomposition*), a dokładnie — właściwość zachowania zależności oraz właściwość złączenia bezstratnego (nieaddytywnego), które są wykorzystywane przez algorytmy projektowania w celu osiągnięcia odpowiedniej dekompozycji. Istotną sprawą jest zauważenie, że *nie wystarczy* sprawdzenie *niezależnie od siebie* schematów relacji pod względem zgodności z wyższymi postaciami normalnymi, takimi jak 2NF, 3NF lub BCNF. Relacje wynikowe muszą *wspólnie* posiadać owe dwie właściwości, aby mogły zostać uznane za składowe poprawnego projektu. Podrozdział 15.3 jest poświęcony algorytmom projektowania relacyjnego, które zaczynają pracę od jednego schematu dużej relacji (**relacji uniwersalnej**), która jest hipotetyczną relacją obejmującą wszystkie atrybuty. Ta relacja jest rozkładana (lub, w innym ujęciu, określone zależności funkcyjne są poddawane syntezie) na relacje zgodne z daną postacią normalną, taką jak 3NF lub BCNF, oraz posiadające przynajmniej jedną z pożądanych właściwości.

W podrozdziale 15.5 omówimy dokładniej zależności wielowartościowe z uwzględnieniem reguł wnioskowania i równoważności. W podrozdziale 15.6 zakończymy opis zależności w danych, wprowadzając zależności zawierania i szablonowe. Zależności zawierania mogą reprezentować więzy integralności odwołań i więzy klasy-podklasy występujące między relacjami. Pokażemy też kilka sytuacji, w których potrzebna jest procedura lub funkcja do określenia i sprawdzenia zależności funkcyjnej między atrybutami. Dalej pokrótce przedstawimy postać normalną klucza dziedziny (ang. *domain-key normal form* — DKNF), uznawaną za najogólniejszą z postaci normalnych. Podrozdział 15.7 to podsumowanie rozdziału.

W przypadku podstawowych zajęć z zakresu teorii baz danych można pominąć podrozdziały 15.3, 15.4 oraz 15.5.

## 15.1. Inne zagadnienia z obszaru zależności funkcyjnych: reguły wnioskowania, równoważności i pokrycie minimalne

W podrozdziale 14.2 wprowadziliśmy zagadnienie zależności funkcyjnych i przedstawiliśmy kilka przykładów. Opracowaliśmy też notację zapisu wielu zależności funkcyjnych w jednej relacji. W podrozdziałach 14.3 i 14.4 zidentyfikowaliśmy i omówiliśmy sprawiające problemy zależności funkcyjne oraz pokazaliśmy, jak je wyeliminować za pomocą odpowiedniej dekompozycji relacji. Ten proces to *normalizacja*. W podrozdziale 14.3 pokazaliśmy, jak uzyskać postacie od pierwszej do trzeciej (1NF do 3NF), posługując się kluczami głównymi. W podrozdziałach 14.4 i 14.5 przedstawiliśmy uogólnione testy postaci 2NF, 3NF i BCNF,

oparte na dowolnej liczbie kluczy kandydujących relacji. Powiedzieliśmy też, jak uzyskać te postacie. Teraz wrócimy do analiz zależności funkcyjnych i pokażemy, jak wywnioskować nowe zależności z ich zbioru. Omówimy też zagadnienia domknięcia, równoważności i pokrycia minimalnego, które będą potrzebne w trakcie późniejszego omawiania projektowania relacji w wyniku syntezy na podstawie zbioru zależności funkcyjnych.

### 15.1.1. Reguły wnioskowania dla zależności funkcyjnych

Niech  $F$  będzie zbiorem zależności funkcyjnych określonych dla schematu relacji  $R$ . Projektant schematu zwykle określa zależności funkcyjne, które są *semantycznie oczywiste*. Jednak przeważnie we *wszystkich* poprawnych instancjach relacji występują też liczne inne zależności funkcyjne, możliwe do uzyskania na podstawie zależności z  $F$  i zgodne z nimi. Te inne zależności można *wywnioskować* lub *wydedukować* na podstawie zależności funkcyjnych z  $F$ . Nazywamy je wywnioskowanymi zależnościami funkcyjnymi.

**Definicja.** Zależność funkcyjna  $X \rightarrow Y$  jest **wywnioskowana** na podstawie zbioru zależności  $F$  relacji  $R$ , jeśli  $X \rightarrow Y$  obowiązuje dla *każdego* poprawnego stanu relacji  $r$  w  $R$ . Oznacza to, że gdy  $r$  jest zgodne ze wszystkimi zależnościami z  $F$ , w  $r$  spełniona jest też zależność  $X \rightarrow Y$ .

W praktyce nie da się określić wszystkich możliwych zależności funkcyjnych w danej sytuacji. Przykładowo, jeśli każdy dział ma jednego kierownika, dzięki czemu NUMERDZ jednoznacznie determinuje PESELKIEROWNIKA ( $\text{NUMERDZ} \rightarrow \text{PESELKIEROWNIKA}$ ), a kierownik ma unikatowy numer telefonu TELKIEROWNIKA ( $\text{PESELKIEROWNIKA} \rightarrow \text{TELKIEROWNIKA}$ ), z tych dwóch zależności można wywnioskować, że  $\text{NUMERDZ} \rightarrow \text{TELKIEROWNIKA}$ . Jest to zależność funkcyjna otrzymana według reguł wnioskowania, której *nie trzeba* bezpośrednio podawać obok dwóch określonych zależności. Dlatego przydatna będzie formalna definicja *domknięcia*, obejmującego wszystkie zależności, jakie można wywnioskować z danego zbioru  $F$ .

**Definicja.** Z formalnego punktu widzenia, zbiór wszystkich zależności, który zawiera zbiór  $F$ , oraz wszelkich zależności możliwych do wywnioskowania na podstawie elementów zbioru  $F$  nosi nazwę **domknięcia** zbioru  $F$  i oznacza się go jako  $F^+$ .

Założmy na przykład, że określiliśmy następujący zbiór  $F$  oczywistych zależności funkcyjnych dla schematu relacji z rysunku 14.3(a):

$$F = \{ \text{PESEL} \rightarrow \{ \text{IMIĘNAZWPRAC}, \text{DATAUR}, \text{ADRES}, \text{NUMERDZ} \}, \\ \text{NUMERDZ} \rightarrow \{ \text{NAZWADZ}, \text{PESELKIEROWNIKA} \} \}$$

Poniżej wymieniono pewne dodatkowe zależności funkcyjne, które można *wywnioskować* na podstawie zbioru  $F$ :

$$\begin{aligned} \text{PESEL} &\rightarrow \{ \text{NAZWADZ}, \text{PESELKIEROWNIKA} \} \\ \text{PESEL} &\rightarrow \text{PESELKIEROWNIKA} \\ \text{NUMERDZ} &\rightarrow \text{NAZWADZ} \end{aligned}$$

Domknięcie  $F^+$  zbioru  $F$  jest zbiorem wszystkich zależności funkcyjnych, które można wywnioskować na podstawie  $F$ . W celu opracowania systematycznej metody wnioskowania zależności, musimy odkryć zbiór **reguł wnioskowania** (ang. *inference rules*), których



można używać w celu wnioskowania nowych zależności na podstawie danego zbioru zależności. Poniżej zostaną omówione niektóre z reguł wnioskowania. Zapis  $F = X \rightarrow Y$  oznacza, że zależność funkcyjna  $X \rightarrow Y$  jest wnioskowana ze zbioru zależności funkcyjnych  $F$ .

Poniżej będziemy wykorzystywać notację skróconą podczas omawiania zależności funkcyjnych. Dla wygody będziemy łączyć zmienne atrybutów i pomijać przecinki. Zatem zapis zależności funkcyjnej  $\{X, Y\} \rightarrow Z$  przyjmie postać  $XY \rightarrow Z$ , zaś  $\{X, Y, Z\} \rightarrow \{U, V\}$  — postać  $XYZ \rightarrow UV$ . Poniżej prezentujemy trzy reguły (od RW1 do RW3), które są dobrze znanymi regułami wnioskowania dla zależności funkcyjnych. Po raz pierwszy zostały one zaproponowane przez Armstronga (1974), dlatego nazywa się je **aksjomatami Armstronga**<sup>1</sup>.

RW1 (reguła zwrotna<sup>2</sup>): jeżeli  $X \supseteq Y$ , to  $X \rightarrow Y$ .

RW2 (reguła zwiększenia<sup>3</sup>):  $\{X \rightarrow Y\} = XZ \rightarrow YZ$ .

RW3 (reguła przechodnia):  $\{X \rightarrow Y, Y \rightarrow Z\} = X \rightarrow Z$ .

Armstrong (1974) wykazał, że reguły wnioskowania od RW1 do RW3 są logiczne i kompletne. Przez **logiczne** (ang. *sound*) rozumiemy, że mając dany zbiór zależności funkcyjnych  $F$  określony dla schematu relacji  $R$ , wszelkie zależności, które możemy wywnioskować z  $F$  na podstawie warunków od RW1 do RW3 są zachowane w przypadku każdego stanu  $r$  relacji  $R$ , który *spełnia zależności* ze zbioru  $F$ . Przez **kompletne** (ang. *complete*) rozumiemy, że wielokrotnie używając zależności od RW1 do RW3 w celu wnioskowania zależności do momentu, aż nie będzie można wywnioskować żadnych dodatkowych zależności, otrzymujemy kompletny zbiór *wszystkich możliwych zależności*, które można wywnioskować ze zbioru  $F$ . Innymi słowy, zbiór zależności  $F^*$ , który nazwaliśmy **domknięciem** zbioru  $F$ , można określić dla zbioru  $F$  poprzez użycie tylko reguł wnioskowania od RW1 do RW3.

Reguła zwrotna (RW1) określa, że zbiór atrybutów zawsze determinuje sam siebie i dowolny swój podzbiór, co jest oczywiste. Ze względu na fakt, że reguła RW1 generuje zależności, które zawsze są prawdziwe, takie zależności określa się jako *trywialne* (ang. *trivial*). Ujmując rzecz formalnie, zależność funkcyjna  $X \rightarrow Y$  jest **trywialna**, jeżeli  $X \supseteq Y$ . W przeciwnym razie jest **nietrywialna**. Reguła zwiększenia (RW2) określa, że dodanie tego samego zbioru atrybutów zarówno do lewej, jak i prawej strony zależności daje w wyniku kolejną poprawną zależność. Zgodnie z regułą RW3, zależności funkcyjne są przechodnie.

Każdą z przedstawionych powyżej reguł wnioskowania można udowodnić na podstawie definicji zależności funkcyjnej przez dowód bezpośredni lub dowód przez **sprowadzenie do sprzeczności**. W przypadku dowodu przez sprowadzenie do sprzeczności przyjmuje

<sup>1</sup> W rzeczywistości są to raczej reguły wnioskowania. W sensie ściśle matematycznym *aksjomatami* (faktami) są zależności funkcyjne w  $F$ , ponieważ zakładamy, że są one poprawne. Z kolei RW1, RW2 i RW3 to *reguły wnioskowania* służące do wnioskowania nowych zależności funkcyjnych (nowych faktów).

<sup>2</sup> Regułę zwrotną można również zapisać jako  $X \rightarrow X$ . Oznacza to, że dowolny zbiór atrybutów determinuje funkcyjnie sam siebie.

<sup>3</sup> Regułę zwiększenia można również zapisać jako  $\{X \rightarrow Y\} = XZ \rightarrow Y$ . Oznacza to, że zwiększenie atrybutów lewostronnych zależności funkcyjnej daje dodatkową poprawną zależność funkcyjną.

się, że reguła nie jest zachowana, i wykazuje, że sytuacja taka nie jest możliwa. Poniżej wykażemy, że pierwsze z trzech reguł od RW1 do RW3 są poprawne. Drugi z dowodów ma charakter sprowadzenia do sprzeczności.

**DOWÓD dla reguły RW1.** Załóżmy, że  $X \supseteq Y$  oraz że dwie krotki  $t_1$  i  $t_2$  występują w pewnej realizacji  $r$  relacji  $R$ , takiej że  $t_1[X] = t_2[X]$ . Wówczas  $t_1[Y] = t_2[Y]$ , ponieważ  $X \supseteq Y$ . Stąd zależność  $X \rightarrow Y$  musi być zachowana w  $r$ .

**DOWÓD dla reguły RW2 (przez sprowadzenie do sprzeczności).** Załóżmy, że zależność  $X \rightarrow Y$  jest zachowana w realizacji  $r$  relacji  $R$ , ale nie jest zachowana zależność  $XZ \rightarrow YZ$ . W takim przypadku w  $r$  muszą występować dwie krotki  $t_1$  i  $t_2$ , takie że (1)  $t_1[X] = t_2[X]$ , (2)  $t_1[Y] = t_2[Y]$ , (3)  $t_1[XZ] = t_2[XZ]$  oraz (4)  $t_1[YZ] \neq t_2[YZ]$ . Nie jest to możliwe, gdyż na podstawie warunków (1) i (3) możemy wywieść warunek (5)  $t_1[Z] = t_2[Z]$ , zaś na podstawie warunków (2) i (5) — warunek (6)  $t_1[YZ] = t_2[YZ]$ , co stanowi sprzeczność względem warunku (4).

**DOWÓD dla reguły RW3.** Załóżmy, że zależności (1)  $X \rightarrow Y$  oraz (2)  $Y \rightarrow Z$  są zachowane w relacji  $r$ . Wówczas, na podstawie warunku (1) dla dowolnych dwóch krotek  $t_1$  i  $t_2$  w  $r$ , takich że  $t_1[X] = t_2[X]$ , musi być spełniony warunek (3)  $t_1[Y] = t_2[Y]$ . Zatem, na podstawie warunku (3) oraz założenia (2) musi być również spełniony warunek (4)  $t_1[Z] = t_2[Z]$ . Ostatecznie stwierdzamy, że zależność  $X \rightarrow Z$  musi być zachowana w  $r$ .

Istnieją też trzy inne reguły wnioskowania obok RW1, RW2 i RW3. Oto one:

RW4 (reguła dekompozycji, inaczej projekcji):  $\{X \rightarrow YZ\} = X \rightarrow Y$ .

RW5 (reguła sumy, czyli addytywna):  $\{X \rightarrow Y, X \rightarrow Z\} = X \rightarrow YZ$ .

RW6 (reguła pseudoprzechodnia):  $\{X \rightarrow Y, WY \rightarrow Z\} = WX \rightarrow Z$ .

Reguła dekompozycji (RW4) określa, że można usunąć atrybuty z prawej strony zależności. Wielokrotne stosowanie tej reguły pozwala na dekompozycję zależności funkcyjnej  $X \rightarrow \{A_1, A_2, \dots, A_n\}$  do postaci zbioru zależności  $\{X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n\}$ . Reguła sumy (RW5) pozwala na dokonanie odwrotnych działań: możemy połączyć zbiór zależności  $\{X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n\}$  w pojedynczą zależność funkcyjną  $X \rightarrow \{A_1, A_2, \dots, A_n\}$ . Reguła pseudoprzechodnia (RW6) umożliwia zastąpienie zbioru atrybutów  $Y$  po lewej stronie zależności innym zbiorem,  $X$ , który funkcyjnie determinuje  $Y$  i może zostać wyprowadzony na podstawie reguł RW2 i RW3, jeśli wzbogacimy pierwszą zależność funkcyjną  $X \rightarrow Y$  o  $W$  (reguła zwiększenia), a następnie zastosujemy regułę przechodnią.

Trzeba wspomnieć o pewnym *ważnym ostrzeżeniu* dotyczącym użycia tych reguł. Choć  $X \rightarrow A$  oraz  $X \rightarrow B$  implikuje na podstawie reguły sumy, że  $X \rightarrow AB$ , to  $X \rightarrow A$  oraz  $Y \rightarrow B$  *nie* implikuje, że  $XY \rightarrow AB$ . Ponadto  $XY \rightarrow A$  *nie* musi oznaczać, że  $X \rightarrow A$  lub  $Y \rightarrow A$ .

Używając podobnej argumentacji można udowodnić reguły wnioskowania od RW4 do RW6 oraz wszelkie inne poprawne reguły. Jednak prostszym sposobem udowodnienia, że dana reguła wnioskowania jest zachowana dla zależności funkcyjnych, jest wykazanie tego poprzez użycie reguł wnioskowania, których poprawność została już dowiedziona. Dlatego reguły RW4, RW5 i RW6 są uważane za uzupełnienie podstawowych reguł wnioskowania Armstronga. Na przykład, możemy udowodnić reguły od RW4 do RW6, używając reguł od RW1 do RW3. Poniżej prezentujemy dowód reguły RW5. Udowodnienie reguł RW4 i RW6 z użyciem reguł od RW1 do RW3 pozostawiamy jako ćwiczenie dla Czytelników.

**DOWÓD dla reguły RW5 (z użyciem reguł od RW1 do RW3).**

1.  $X \rightarrow Y$  (dane).
2.  $X \rightarrow Z$  (dane).
3.  $X \rightarrow XY$  (na podstawie reguły RW2 względem zależności 1. poprzez powiększenie o  $X$ ; należy zauważyć, że  $XX = X$ ).
4.  $XY \rightarrow YZ$  (na podstawie reguły RW2 względem zależności 2. poprzez powiększenie o  $Y$ ).
5.  $X \rightarrow YZ$  (na podstawie reguły RW3 względem zależności 3. i 4.).

Zazwyczaj projektanci baz danych najpierw definiują zbiór zależności funkcyjnych  $F$ , które można z łatwością określić na podstawie semantyki atrybutów relacji  $R$ . Następnie używa się reguł RW1, RW2 oraz RW3 w celu wnioskowania o dodatkowych zależnościach funkcyjnych, które również są zachowane w  $R$ . Systematyczny sposób określania takich dodatkowych zależności funkcyjnych polega na określeniu w pierwszej kolejności każdego zbioru atrybutów  $X$ , który występuje jako lewa strona pewnej zależności funkcyjnej w  $F$ , a następnie na określeniu zbioru *wszystkich atrybutów*, które są zależne od  $X$ .

**Definicja.** Dla każdego takiego zbioru atrybutów  $X$  określamy zbiór  $X^+$  atrybutów, które są funkcyjnie determinowane przez  $X$  na podstawie zbioru  $F$ . Zbiór  $X^+$  nosi nazwę **domknięcia** (ang. *closure*) **zbioru  $X$  względem zbioru  $F$** .

W celu określenia zbioru  $X^+$  można użyć algorytmu 15.1.

**Algorytm 15.1.** Określanie zbioru  $X^+$  — domknięcia zbioru  $X$  względem zbioru  $F$

**Dane wejściowe:** Zbiór  $F$  zależności funkcyjnych dla schematu relacji  $R$  i zbiór atrybutów  $X$  będący podzbiorem  $R$ .

```

 $X^+ := X$ ;
repeat
   $oldX^+ := X^+$ ;
  for each zależność funkcyjna  $Y \rightarrow Z$  w  $F$  do
    if  $X^+ \supseteq Y$  then  $X^+ := X^+ \cup Z$ ;
until ( $X^+ = oldX^+$ );

```

Algorytm 15.1 rozpoczyna działanie od ustawienia zawartości zbioru  $X^+$  na wszystkie atrybuty zawarte w zbiorze  $X$ . Na podstawie reguły RW1 wiemy, że wszystkie te atrybuty są funkcyjnie zależne od  $X$ . Używając reguł wnioskowania RW3 i RW4 dodajemy atrybuty do zbioru  $X^+$ , wykorzystując każdą zależność funkcyjną ze zbioru  $F$ . Kontynuujemy działania dla wszystkich zależności w  $F$  (pętla repeat) do momentu, aż w czasie pełnego cyklu (pętli for) przejścia przez zależności w zbiorze  $F$  okaże się, że do zbioru  $X^+$  nie zostały dodane żadne atrybuty. Domknięcie jest przydatne do zrozumienia znaczenia i skutków występowania atrybutów lub ich zbiorów w relacji. Rozważ np. następujący schemat relacji dotyczący kursów prowadzonych na uczelni w danym roku akademickim.

```

KURS(ID_KURSU, KURS#, NAZW_WYKŁ., PUNKTY, PODRĘCZNIK, WYDAWCA,
      SALA, MIEJSC)

```

Niech  $F$  (zbiór zależności funkcyjnych dla przedstawionej relacji) obejmuje następujące zależności:

FD1:  $ID\_KURSU \rightarrow KURS\#, NAZW\_WYK\#, PUNKTY, PODR\acute{E}CZNIK, WYDAWCA, SALA, MIEJSC$

FD2:  $KURS\# \rightarrow PUNKTY$

FD3:  $\{KURS\#, NAZW\_WYK\# \} \rightarrow PODR\acute{E}CZNIK, SALA$

FD4:  $PODR\acute{E}CZNIK \rightarrow WYDAWCA$

FD5:  $SALA \rightarrow MIEJSC$

Zauważ, że podane zależności funkcyjne określają semantykę danych w relacji KURS. Przykładowo, z zależności FD1 wynika, że każdy kurs ma unikatowy identyfikator. Zgodnie z zależnością FD3 kurs prowadzony przez określonego wykładowcę ma ustalony podręcznik, a wykładowca uczy na danym kursie w stałej sali. Stosując reguły wnioskowania i definicję domknięcia do przedstawionych zależności, można utworzyć następujące domknięcia:

$$\{ID\_KURSU\}^+ = \{ID\_KURSU, KURS\#, NAZW\_WYK\#, PUNKTY, PODR\acute{E}CZNIK, WYDAWCA, SALA, MIEJSC\} = KURS$$

$$\{KURS\#\}^+ = \{KURS\#, PUNKTY\}$$

$$\{KURS\#, NAZW\_WYK\#\}^+ = \{KURS\#, PUNKTY, PODR\acute{E}CZNIK, WYDAWCA, SALA, MIEJSC\}$$

Warto zauważyć, że każde domknięcie ma interpretację dotyczącą atrybutów podanych po lewej stronie. Przykładowo, domknięcie dla atrybutu KURS# obejmuje obok tego atrybutu tylko atrybut GODZINY. Nie zawiera atrybutu NAZW\_WYK#, ponieważ dany kurs mogą prowadzić różne osoby. Nie obejmuje też atrybutu PODRĘCZNIK, ponieważ poszczególni wykładowcy mogą korzystać na danym kursie z różnych podręczników. Zauważ też, że domknięcie pary  $\{KURS\#, NAZW\_WYK\#\}$  nie obejmuje atrybutu ID\_KURSU, z czego wynika, że atrybut ten nie jest kluczem kandydującym. Z tego z kolei można wywnioskować, że kurs o danym numerze KURS# może być prowadzony przez różnych wykładowców, co oznacza, że takie kursy są niezależnymi zajęciami.

### 15.1.2. Równoważność zbiorów zależności funkcyjnych

W niniejszym podrozdziale zostanie omówiona kwestia równoważności dwóch zbiorów zależności funkcyjnych. W pierwszej kolejności przedstawimy pewne definicje wstępne.

**Definicja.** O zbiorze zależności funkcyjnych  $F$  mówi się, że **pokrywa** (ang. *cover*) inny zbiór zależności funkcyjnych  $E$ , jeżeli każda zależność funkcyjna ze zbioru  $E$  występuje również w zbiorze  $F^+$ , to znaczy, jeżeli każdą zależność ze zbioru  $E$  można wywnioskować na podstawie zbioru  $F$ . Można również powiedzieć, że zbiór  $E$  **jest pokrywany** przez zbiór  $F$ .

**Definicja.** Dwa zbiory zależności funkcyjnych  $E$  i  $F$  są **równoważne** (ang. *equivalent*), jeżeli  $E^+ = F^+$ . Zatem równoważność oznacza, że każda zależność funkcyjna ze zbioru  $E$  może zostać wywnioskowana na podstawie zbioru  $F$  oraz każda zależność funkcyjna ze zbioru  $F$  może zostać wywnioskowana na podstawie zbioru  $E$ . Zbiór  $E$  jest równoważny zbiorowi  $F$ , jeżeli są spełnione warunki pokrywania zbioru  $F$  przez  $E$  oraz zbioru  $E$  przez  $F$ .

Możemy stwierdzić, czy zbiór  $F$  pokrywa zbiór  $E$ , określając zbiór  $X^*$  względem zbioru  $F$  dla każdej zależności funkcyjnej  $X \rightarrow Y$  znajdującej się w zbiorze  $E$ , a następnie sprawdzając, czy taki zbiór  $X^*$  zawiera atrybuty ze zbioru  $Y$ . Jeżeli okaże się, że tak jest dla *każdej* zależności funkcyjnej w  $E$ , wówczas zbiór  $F$  pokrywa zbiór  $E$ . Równoważność zbiorów

$E$  i  $F$  określamy sprawdzając, czy zbiór  $E$  pokrywa zbiór  $F$  oraz czy zbiór  $F$  pokrywa zbiór  $E$ . Ćwiczeniem dla Czytelnika jest wykazanie, że poniższe dwa zbiory zależności funkcyjnych są równoważne:

$$F = \{A \rightarrow C, AC \rightarrow D, E \rightarrow AD, E \rightarrow H\}$$

$$\text{ i } G = \{A \rightarrow CD, E \rightarrow AH\}$$

### 15.1.3. Zbiory minimalne zależności funkcyjnych

Podobnie jak zastosowaliśmy reguły wnioskowania, aby rozbudować zbiór zależności funkcyjnych  $F$  w celu uzyskania jego domknięcia  $F^+$ , tak możemy wykonać odwrotną operację i sprawdzić, czy możliwe jest zmniejszenie lub ograniczenie zbioru  $F$  do *minimalnej postaci* równoważnej z pierwotnym zbiorem  $F$ . Mówiąc nieformalnie, **pokrycie minimalne** (ang. *minimal cover*) zbioru zależności funkcyjnych zbioru  $E$  to zbiór zależności funkcyjnych  $F$ , który spełnia właściwość mówiącą o tym, że każda zależność ze zbioru  $E$  występuje w domknięciu  $F^+$  zbioru  $F$ . Ponadto właściwość ta przestaje być spełniona w przypadku usunięcia dowolnej zależności ze zbioru  $F$ . Zbiór  $F$  nie może zawierać żadnych elementów nadmiarowych, zaś zależności w zbiorze  $E$  muszą być w postaci standardowej.

Do zdefiniowania pokrycia minimalnego posłużymy się pojęciem atrybutu nadmiarowego w zależności funkcyjnej.

**Definicja.** Atrybut w zależności funkcyjnej jest uznawany za **nadmiarowy**, jeśli jego usunięcie nie powoduje zmiany domknięcia zbioru zależności. Oto formalne ujęcie: gdy dane są zbiór zależności funkcyjnych  $F$  i zależność  $X \rightarrow A$  w  $F$ , to atrybut  $Y$  jest nadmiarowy w  $X$ , jeśli  $Y \subset X$  i z  $F$  logicznie wynika  $(F - (X \rightarrow A) \cup \{(X - Y) \rightarrow A\})$ .

Można formalnie zdefiniować zbiór zależności funkcyjnych  $F$  jako **minimalny**, jeżeli spełnia on następujące warunki:

- (1) Każda zależność w zbiorze  $F$  posiada po swojej prawej stronie pojedynczy atrybut.
- (2) Nie można zastąpić żadnej zależności  $X \rightarrow A$  w zbiorze  $F$  zależnością  $Y \rightarrow A$ , gdzie  $Y$  jest podzbiorem właściwym zbioru  $X$ , otrzymując zbiór zależności równoważny zbiorowi  $F$ .
- (3) Nie można usunąć żadnej zależności ze zbioru  $F$ , a mimo to wciąż posiadać zbiór zależności równoważny zbiorowi  $F$ .

Minimalny zbiór zależności można postrzegać jako zbiór zależności w *postaci standardowej* (ang. *standard form*), inaczej *kanonicznej* (ang. *canonical*), oraz *niezawierający elementów nadmiarowych*. Warunek 1. reprezentuje każdą z zależności w postaci kanonicznej o pojedynczym atrybucie po prawej stronie i stanowi krok wstępny pozwalający sprawdzić, czy spełnione są warunki 2. i 3.<sup>4</sup> Warunki 2. i 3. zapewniają, że wśród zależności nie występuje nadmiarowość, albo przez umieszczenie atrybutów nadmiarowych po lewej stronie zależności (warunek 2.), albo przez określenie zależności, którą można wywnioskować na podstawie pozostałych zależności ze zbioru  $F$  (warunek 3.).

<sup>4</sup> Jest to standardowa forma upraszczania warunków i algorytmów, zapewniających brak nadmiarowości w zbiorze  $F$ . Wykorzystując regułę wnioskowania RW4, możemy zamienić pojedynczą zależność o wielu atrybutach po prawej stronie na zbiór zależności z jednym atrybutem po prawej stronie.

**Definicja. Pokrycie minimalne** zbioru zależności funkcyjnych  $E$  jest minimalnym zbiorem zależności (w standardowej postaci kanonicznej<sup>5</sup> i bez nadmiarowości) równoważnym zbiorowi  $E$ . Zawsze można znaleźć *co najmniej jedno* pokrycie minimalne  $F$  dla dowolnego zbioru zależności  $E$ , wykorzystując algorytm 15.2.

Jeżeli do zestawu pokryć minimalnych zbioru  $E$ , zgodnie z przedstawioną powyżej definicją, kwalifikuje się kilka zbiorów zależności funkcyjnych, to zazwyczaj używa się dodatkowego kryterium *minimalności*. Możemy, na przykład, jako zbiór minimalny wybrać taki, który zawiera *najmniejszą liczbę zależności* lub *najmniejszą długość całkowitą* (długość całkowita zbioru zależności jest obliczana w wyniku połączenia zależności i traktowania ich jako jednego długiego ciągu znaków).

**Algorytm 15.2.** Znajdowanie pokrycia minimalnego  $F$  dla zbioru zależności funkcyjnych  $E$

**Dane wejściowe:** Zbiór zależności funkcyjnych  $E$ .

*Uwaga:* Po niektórych krokach znajdują się komentarze objaśniające. Są one zapisane w formacie (\*komentarz\*).

- (1) Niech  $F := E$ .
- (2) Zastępujemy każdą zależność funkcyjną  $X \rightarrow \{A_1, A_2, \dots, A_n\}$  w zbiorze  $F$   $n$  zależnościami funkcyjnymi  $X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n$ . (\* To powoduje przekształcenie zależności funkcyjnej na postać kanoniczną na potrzeby dalszych testów \*)
- (3) Dla każdej zależności funkcyjnej  $X \rightarrow A$  w zbiorze  $F$   
 dla każdego atrybutu  $B$  będącego elementem zbioru  $X$   
 jeżeli zbiór  $\{F - \{X \rightarrow A\}\} \cup \{(X - \{B\}) \rightarrow A\}$  jest równoważny  $F$ ,  
 to zastępujemy zależność  $X \rightarrow A$  zależnością  $(X - \{B\}) \rightarrow A$  w zbiorze  $F$ .  
 (\* To pozwala usunąć — jeśli jest to możliwe — nadmiarowy atrybut  $B$  znajdujący się po lewej stronie zależności funkcyjnej  $X \rightarrow A$  \*)
- (4) Dla każdej z pozostałych zależności  $X \rightarrow A$  w zbiorze  $F$   
 jeżeli zbiór  $\{F - \{X \rightarrow A\}\}$  jest równoważny  $F$ ,  
 to usuwamy zależność  $X \rightarrow A$  ze zbioru  $F$ .  
 (\* To pozwala usunąć — jeśli jest to możliwe — nadmiarową zależność funkcyjną  $X \rightarrow A$  z  $F$  \*)

Zilustrujmy działanie tego algorytmu za pomocą przykładów.

**Przykład 1.** Niech zbiór zależności funkcyjnych  $E$  wygląda tak:  $\{B \rightarrow A, D \rightarrow A, AB \rightarrow D\}$ . Zadanie to znalezienie minimalnego pokrycia w zbiorze  $E$ .

- Wszystkie podane zależności są w postaci kanonicznej (posiadają tylko jeden atrybut po prawej stronie), dlatego krok 1. algorytmu 15.2 jest już wykonany i można przejść do kroku 2. W kroku 2. trzeba ustalić, czy w zależności  $AB \rightarrow D$  po lewej stronie występuje nadmiarowy atrybut (tzn. czy można ją zastąpić zależnością  $B \rightarrow D$  lub  $A \rightarrow D$ ).

<sup>5</sup> Można posłużyć się regułą wnioskowania RW5 i połączyć zależności funkcyjne mające te same elementy po lewej stronie w jedną zależność funkcyjną będącą pokryciem minimalnym w niestandardowej postaci. Zbiór wynikowy też jest wtedy pokryciem minimalnym, co ilustruje przykład.



- Ponieważ  $B \rightarrow A$ , to w wyniku zwiększenia zależności o  $B$  po obu stronach (RW2) otrzymujemy  $BB \rightarrow AB$  lub  $B \rightarrow AB$  (i). Jednak  $AB \rightarrow D$  (ii).
- Na podstawie reguły przechodniej (RW3) otrzymujemy z (i) i (ii)  $B \rightarrow D$ . Tak więc  $AB \rightarrow D$  można zastąpić przez  $B \rightarrow D$ .
- Uzyskujemy teraz zbiór równoważny względem  $E$ . Nazwijmy go  $E'$ :  $\{B \rightarrow A, D \rightarrow A, B \rightarrow D\}$ . W kroku 2. nie są możliwe dalsze redukcje, ponieważ wszystkie zależności funkcyjne mają po jednym atrybucie po lewej stronie.
- W kroku 3. szukamy nadmiarowych zależności funkcyjnych w  $E'$ . Stosując regułę przechodnią do  $B \rightarrow D$  i  $D \rightarrow A$ , otrzymujemy  $B \rightarrow A$ . Tak więc zależność  $B \rightarrow A$  jest nadmiarowa w  $E'$  i można ją usunąć.
- Dlatego pokryciem minimalnym dla  $E$  jest zbiór  $F$ :  $\{B \rightarrow D, D \rightarrow A\}$ .

Czytelnik może się przekonać, że na podstawie  $E$  można wywnioskować pierwotny zbiór  $F$ . Oznacza to, że zbiory  $F$  i  $E$  są równoważne.

**Przykład 2.** Niech zbiór zależności funkcyjnych  $G$  wygląda tak:  $\{A \rightarrow BCDE, CD \rightarrow E\}$ .

- Tu zależności funkcyjne NIE są w postaci kanonicznej. Dlatego najpierw należy je przekształcić do postaci:  
 $E$ :  $\{A \rightarrow B, A \rightarrow C, A \rightarrow D, A \rightarrow E, CD \rightarrow E\}$ .
- W kroku 2. algorytmu w zależności  $CD \rightarrow E$  ani atrybut  $C$ , ani  $D$  nie jest nadmiarowy po lewej stronie, ponieważ na podstawie danych zależności funkcyjnych nie można wykazać, że  $C \rightarrow E$  lub  $D \rightarrow E$ . Nie można więc zastąpić żadnego z tych atrybutów.
- W kroku 3. sprawdzamy, czy któraś z zależności jest nadmiarowa. Ponieważ  $A \rightarrow CD$  i  $CD \rightarrow E$ , z reguły przechodniej (RW3) wynika, że  $A \rightarrow E$ . Tak więc zależność  $A \rightarrow E$  jest nadmiarowa w  $G$ .
- Pozostaje więc zbiór  $F$ , który jest równoważny względem pierwotnego zbioru  $G$  i wygląda tak:  $\{A \rightarrow B, A \rightarrow C, A \rightarrow D, CD \rightarrow E\}$ .  $F$  jest pokryciem minimalnym. Zgodnie z przypisem 5. można połączyć pierwsze trzy zależności za pomocą reguły sumy (RW5) i wyrazić pokrycie minimalne w następujący sposób:

Minimalne pokrycie zbioru  $G$  to  $F$ :  $\{A \rightarrow BCD, CD \rightarrow E\}$ .

W podrozdziale 15.3 przedstawimy algorytmy do syntezy relacji w postaci 3NF lub BCNF na podstawie zbioru zależności  $E$ ; te algorytmy najpierw znajdują pokrycie minimalne  $F$  dla zbioru  $E$ .

Teraz pokażemy prosty algorytm określania kluczy relacji.

**Algorytm 15.2(a).** Znajdowanie klucza  $K$  dla  $R$  na podstawie zbioru zależności funkcyjnych  $F$ .

**Dane wejściowe:** Relacja  $R$  i zbiór zależności funkcyjnych  $F$  dla atrybutów relacji  $R$ .

(1) Ustaw  $K := R$ .

(2) Dla każdego atrybutu  $A$  z  $K$

{oblicz  $(K - A)^+$  względem  $F$ ;

jeśli  $(K - A)^+$  obejmuje wszystkie atrybuty z  $R$ , ustaw  $K := K - \{A\}$ };

W algorytmie 15.2(a) zaczynamy od przypisania do  $K$  wszystkich atrybutów z  $R$ . Można powiedzieć, że sama relacja  $R$  zawsze jest **domyślnym nadkluczem**. Następnie atrybuty są usuwane jeden po drugim, po czym należy sprawdzić, czy pozostałe atrybuty nadal



tworzą nadklucz. Warto też zauważyć, że algorytm 15.2(a) określa tylko *jeden* klucz z możliwych kluczy kandydujących relacji  $R$ . To, który klucz zostanie zwrócony, zależy od kolejności usuwania atrybutów z  $R$  w kroku 2.

## 15.2. Właściwości dekompozycji relacyjnych

Teraz skupimy się na procesie dekompozycji stosowanym w rozdziale 14. do eliminowania niepożądanych zależności i osiągania wyższych postaci normalnych. W podrozdziale 15.2.1 zostaną przedstawione przykłady dowodzące, że sprawdzanie *pojedynczych* relacji pod względem ich zgodności z wyższymi postaciami normalnymi samo w sobie nie gwarantuje poprawności projektu. Zamiast tego *zbiór relacji*, które wspólnie tworzą schemat relacyjnej bazy danych, musi posiadać określone dodatkowe właściwości, zapewniające poprawność projektu. W podrozdziałach 15.2.2 i 15.2.3 zostaną omówione dwie takie właściwości: właściwość zachowania zależności oraz właściwość złączenia bezstratnego (nieaddytywnego). W podrozdziale 15.2.4 zostaną omówione dekompozycje binarne, zaś w podrozdziale 15.2.5 — kolejne dekompozycje ze złączeniem nieaddytywnym.

### 15.2.1. Dekompozycja relacji i niewystarczalność postaci normalnych

Algorytmy projektowania relacyjnych baz danych prezentowane w podrozdziale 15.3 rozpoczynają działanie od pojedynczego **schematu relacji uniwersalnej** (ang. *universal relation schema*)  $R = \{A_1, A_2, \dots, A_n\}$ , który zawiera *wszystkie* atrybuty bazy danych. Niejawnie przyjmujemy **założenie relacji uniwersalnej** (ang. *universal relation assumption*), zgodnie z którym nazwa każdego atrybutu jest unikatowa. Projektant bazy danych określa zbiór  $F$  zależności funkcyjnych, które powinny być zachowane dla atrybutów relacji  $R$ , i jest on udostępniany algorytmom projektowym. Wykorzystując zależności funkcyjne, algorytmy rozkładają relację uniwersalną  $R$  na zbiór schematów relacji  $D = \{R_1, R_2, \dots, R_m\}$ , który staje się schematem relacyjnej bazy danych. Zbiór  $D$  określa się mianem **dekompozycji** (ang. *decomposition*) relacji  $R$ .

Musimy się upewnić, że każdy atrybut relacji  $R$  występuje przynajmniej w jednej relacji  $R_i$  w dekompozycji, tak aby żaden z nich nie został *utracony*. Ujmując rzecz formalnie, możemy zapisać to w sposób następujący:

$$\bigcup_{i=1}^m R_i = R$$

Jest to tak zwany warunek **zachowania atrybutów** (ang. *attribute preservation*) dekompozycji.

Kolejnym celem jest zapewnienie, aby każda relacja  $R_i$  występująca w dekompozycji  $D$  znajdowała się w postaci BCNF lub 3NF. Warunek ten sam w sobie nie wystarczy jednak do zagwarantowania poprawności projektu bazy danych. Oprócz uwzględniania pojedynczych relacji należy wziąć pod uwagę dekompozycję relacji uniwersalnej jako całości.

W celu zilustrowania tego spostrzeżenia weźmy pod uwagę relację  $\text{PRAC\_LOKALIZACJE} \hookrightarrow (\text{IMIĘNAZWPRAC}, \text{LOKALIZACJAPROJ})$  z rysunku 14.5, która znajduje się w postaci 3NF i jednocześnie w postaci BCNF. W rzeczywistości każdy schemat relacji zawierający tylko dwa atrybuty automatycznie znajduje się w postaci BCNF<sup>6</sup>. Chociaż relacja  $\text{PRAC\_LOKALIZACJE}$  znajduje się w postaci BCNF, wciąż powoduje występowanie fałszywych krotek w razie złączenia z relacją  $\text{PRAC\_PROJ}(\text{PESEL}, \text{NUMERPROJ}, \text{GODZINY}, \text{NAZWAPROJ}, \text{LOKALIZACJAPROJ})$ , która nie znajduje się w postaci BCNF (patrz wynik złączenia naturalnego na rysunku 14.6). Zatem relacja  $\text{PRAC\_LOKALIZACJE}$  reprezentuje szczególnie zły schemat relacji ze względu na swoją mało zrozumiałą semantykę, która definiuje, że atrybut  $\text{LOKALIZACJAPROJ}$  określa lokalizację *jednego z projektów*, nad którymi pracuje pracownik. Złączenie relacji  $\text{PRAC\_LOKALIZACJE}$  z relacją  $\text{PROJEKT}(\text{NAZWAPROJ}, \text{NUMERPROJ}, \text{LOKALIZACJAPROJ}, \text{NRDZ})$  z rysunku 14.2 — *znajdującą się* w postaci BCNF — z wykorzystaniem atrybutu  $\text{LOKALIZACJA} \hookrightarrow \text{PROJ}$  również powoduje występowanie fałszywych krotek. Unaocznia to potrzebę zdefiniowania dodatkowych kryteriów, które, wraz z warunkami postaci 3NF i BCNF, będą zapobiegać powstawaniu takich niepoprawnych projektów. W trzech kolejnych punktach zostaną omówione takie warunki, które powinny być zachowane w ramach dekompozycji  $D$  postrzeganej całościowo.

### 15.2.2. Właściwość zachowania zależności dekompozycji

Przydatną rzeczą byłoby, gdyby każda zależność funkcyjna  $X \rightarrow Y$  określona w zbiorze  $F$  albo występowała bezpośrednio w jednym ze schematów relacji  $R_i$  w dekompozycji  $D$ , albo mogła być wywnioskowana na podstawie zależności występujących w pewnej relacji  $R_i$ . W ujęciu nieformalnym jest to *warunek zachowania zależności* (ang. *dependency preservation condition*). Chcemy zachowywać zależności, ponieważ każda zależność ze zbioru  $F$  reprezentuje więzy na bazie danych. Jeżeli któraś z zależności nie jest reprezentowana w pewnej pojedynczej relacji  $R_i$  dekompozycji, nie możemy wymusić tych więzów poprzez odwołanie się do pojedynczej relacji. Możliwe, że konieczne będzie złączenie wielu relacji, aby uwzględnić wszystkie atrybuty występujące w danej zależności.

Nie jest wymagane, aby zależności określone w zbiorze  $F$  były dokładnie powielane w poszczególnych relacjach dekompozycji  $D$ . Wystarczy, aby suma zależności zachowanych w poszczególnych relacjach w  $D$  była równoważna zbiorowi  $F$ . Poniżej pojęcia te zostaną zdefiniowane w sposób bardziej formalny.

**Definicja.** Dla danego zbioru zależności  $F$  na relacji  $R$  **projekcja** (ang. *projection*) zbioru  $F$  na  $R_i$ , oznaczany jako  $\pi_{R_i}(F)$ , gdzie  $R_i$  jest podzbiorem relacji  $R$ , to zbiór

zależności  $X \rightarrow Y$  w  $F^+$ , taki że wszystkie atrybuty zbioru  $X \cup Y$  występują w zbiorze  $R_i$ . Stąd projekcja zbioru  $F$  na każdy schemat relacji  $R_i$  w dekompozycji  $D$  jest zbiorem zależności funkcyjnych z  $F^+$ , czyli domknięcia zbioru  $F$ , takim że wszystkie atrybuty lewo- i prawostronne występują w zbiorze  $R_i$ . Mówimy, że dekompozycja  $D = \{R_1, R_2, \dots, R_m\}$  relacji  $R$  **zachowuje zależności** w kontekście zbioru  $F$ , jeżeli suma projekcji  $F$  na każdy ze zbiorów  $R_i$  w  $D$  jest równoważna zbiorowi  $F$ , czyli:

$$\left( \left( \pi_{R_1}(F) \right) \cup K \cup \left( \pi_{R_m}(F) \right) \right)^+ = F^+.$$

<sup>6</sup> W ramach ćwiczenia Czytelnik powinien udowodnić prawdziwość tego stwierdzenia.

Jeżeli dekompozycja nie zachowuje zależności, niektóre zależności są w niej **tracone**. Jak wspomniano powyżej, w celu sprawdzenia, czy utracona zależność jest zachowana, musimy wykonać złączenie dwóch lub większej liczby relacji z dekompozycji w celu otrzymania relacji zawierającej wszystkie lewo- i prawostronne atrybuty utraconej zależności, a następnie sprawdzić, czy zależność jest zachowana w wyniku operacji złączenia — nie jest to jednak praktyczne rozwiązanie.

Przykład dekompozycji niezachowującej zależności przedstawiono na rysunku 14.13(a), gdzie zależność funkcyjna FD2 zostaje utracona, kiedy relację DZIAŁKIIA rozłożymy na zbiór relacji {DZIAŁKIIAX, DZIAŁKIIAY}. Z kolei dekompozycje z rysunku 14.12 zachowują zależności. Natomiast w przypadku rysunku 14.14 bez względu na wybór rozkładu relacji UCZY(STUDENT, PRZEDMIOT, WYKŁADOWCA) spośród trzech opisanych w tekście, jedna lub obie oryginalne zależności zostają utracone. Poniżej przedstawiamy założenie związane z tą właściwością bez przytoczenia żadnego dowodu.

**ZAŁOŻENIE 1.** Zawsze istnieje możliwość znalezienia zachowującej zależności dekompozycji  $D$  w odniesieniu do zbioru  $F$ , takiej że każda relacja  $R_i$  w  $D$  będzie w trzeciej postaci normalnej.

### 15.2.3. Właściwość złączenia bezstratnego (nieaddytywnego) dekompozycji

Kolejną właściwością, którą powinna posiadać dekompozycja  $D$ , jest właściwość złączenia nieaddytywnego, która zapewnia, że w razie zastosowania operacji złączenia naturalnego (NATURAL JOIN) względem relacji powstałych w wyniku dekompozycji nie będą występować żadne fałszywe krotki. Problem ten zilustrowano już w punkcie 14.1.4 na rysunkach 14.5 i 14.6. Ze względu na fakt, że jest to właściwość dekompozycji *schematów* relacji, warunek niewystępowania fałszywych krotek powinien być zachowany dla każdego *dopuszczalnego stanu relacji*, to znaczy każdego stanu relacji spełniającego zależności funkcyjne ze zbioru  $F$ . Zatem właściwość złączenia bezstratnego zawsze definiuje się w kontekście określonego zbioru  $F$  zależności.

**Definicja.** W ujęciu formalnym dekompozycja  $D = \{R_1, R_2, \dots, R_m\}$  relacji  $R$  posiada **właściwość złączenia bezstratnego (nieaddytywnego)** (ang. *lossless (nonadditive) join property*) w kontekście zbioru zależności  $F$  na relacji  $R$ , jeżeli dla *każdego* stanu  $r$  relacji  $R$ , który spełnia zależność  $F$ , jest spełniony poniższy warunek, gdzie symbol  $*$  oznacza operację złączenia naturalnego (NATURAL JOIN) na wszystkich relacjach w dekompozycji  $D$ :  $*(\pi_{R_1}(r), \dots, \pi_{R_m}(r)) = r$ .

Rdzeń „stratny” w wyrazie *bezstratny* odnosi się do *utruty informacji*, a nie utraty krotek. Jeżeli dekompozycja nie posiada właściwości złączenia bezstratnego, istnieje możliwość otrzymania dodatkowych fałszywych krotek po wykonaniu operacji projekcji ( $\pi$ ) oraz złączenia naturalnego ( $*$ ). Owe dodatkowe krotki reprezentują wówczas błędne informacje. Preferuje się używanie pojęcia *złączenia nieaddytywnego*, ponieważ lepiej opisuje ono całą sytuację. Choć w literaturze często stosowana jest nazwa *złączenie bezstratne*, my posługiwaliśmy się zwrotem *złączenie nieaddytywne* w opisie właściwości złączenia nieaddytywnego w punkcie 14.5.1. *Od tego miejsca będziemy używać nazwy złączenie nieaddytywne*, ponieważ jest samoopisowa i jednoznaczna. Jeżeli właściwość ta jest za-

chowana w przypadku dekompozycji, mamy pewność, że do wyniku operacji projekcji iłączenia naturalnego nie będą dodawane żadne fałszywe krotki, niosące ze sobą błędne informacje. Czasem możemy jednak zastosować pojęcie **stratny projekt** do opisu projektu powodującego utratę informacji. Dekompozycja relacji  $PRAC\_PROJ(PESEL, NUMER \hookrightarrow PROJ, GODZINY, IMIĘNAZWPRAC, NAZWAPROJ, LOKALIZACJAPROJ)$  z rysunku 14.3 na relacje  $PRAC\_LOKALIZACJE(IMIĘNAZWPRAC, LOKALIZACJAPROJ)$  i  $PRAC\_PROJ1(PESEL, NUMERPROJ, GODZINY, NAZWAPROJ, LOKALIZACJAPROJ)$ , przedstawione na rysunku 15.5, bez wątpienia nie posiada właściwościłączenia nieaddytywnego, czego dowodzi rysunek 14.6. W punkcie 14.5.1 przedstawiliśmy prostszy test dla dekompozycji binarnych, pozwalający sprawdzić, czy dekompozycja jest nieaddytywna. We wspomnianym punkcie ta właściwość została nazwana właściwościąłączenia nieaddytywnego. W celu sprawdzania, czy dowolna dekompozycja  $D$  pewnej relacji na  $n$  innych relacji jest nieaddytywna w kontekście danego zbioru zależności funkcyjnych  $F$  tej relacji, będziemy korzystać z ogólnej procedury. Przedstawiono ją w formie algorytmu 15.3.

**Algorytm 15.3.** Sprawdzanie występowania właściwościłączenia nieaddytywnego

**Dane wejściowe:** Relacja uniwersalna  $R$ , dekompozycja  $D = \{R_1, R_2, \dots, R_m\}$  relacji  $R$  oraz zbiór  $F$  zależności funkcyjnych.

*Uwaga:* Po niektórych krokach znajdują się komentarze objaśniające. Są one zapisane w formacie (\*komentarz\*).

- (1) Tworzymy początkową macierz  $S$  o jednym wierszu  $i$  dla każdej relacji  $R_i$  z dekompozycji  $D$  oraz jednej kolumnie  $j$  dla każdego atrybutu  $A_j$  z relacji  $R$ .
- (2) Ustawiamy wartość  $S(i, j) := b_{ij}$  dla wszystkich elementów macierzy.
- (3) (\* Każdy element  $b_{ij}$  jest odrębnym symbolem związanym z indeksami  $(i, j)$  \*)
- (4) Dla każdego wiersza  $i$  reprezentującego schemat relacji  $R_i$   
 {dla każdej kolumny  $j$  reprezentującej atrybut  $A_j$   
 {jeżeli (relacja  $R_i$  zawiera atrybut  $A_j$ ), to ustawiamy wartość  $S(i, j) := a_j$ };};  
 (\* Każdy element  $a_j$  jest odrębnym symbolem związanym z indeksem  $(j)$  \*)
- (5) Powtarzamy poniższą pętlę do momentu, aż *pełny przebieg pętli* nie spowoduje żadnych zmian w macierzy  $S$   
 {dla każdej zależności funkcyjnej  $X \rightarrow Y$  w  $F$   
 {dla wszystkich wierszy w macierzy  $S$ , które *posiadają takie same symbole*  
 w kolumnach odpowiadających atrybutom ze zbioru  $X$   
 {zamieniamy symbole w każdej kolumnie odpowiadającej atrybutowi  
 ze zbioru  $Y$  tak, aby były identyczne we wszystkich wierszach  
 w następujący sposób: jeżeli któryś z wierszy posiada symbol „a”  
 w kolumnie, ustawiamy wartości pozostałych wierszy w tej kolumnie  
 na tę samą wartość „a”. Jeżeli w żadnym z wierszy nie występuje symbol  
 „a” dla atrybutu, wybieramy jeden z symboli „b”, który występuje  
 w jednym z wierszy dla atrybutu i ustawiamy wartości pozostałych  
 wierszy w tej kolumnie na tę samą wartość „b”};};};
- (6) Jeżeli wiersz zawiera same symbole „a”, wówczas dekompozycja posiada właściwośćłączenia bezstratnego. W przeciwnym razie — nie posiada tej właściwości.

Dla danej relacji  $R$ , którą rozłożono na wiele relacji  $R_1, R_2, \dots, R_m$ , algorytm 15.3 rozpoczyna działanie od utworzenia macierzy  $S$ , którą traktujemy jako pewien stan  $r$  relacji  $R$ . Wiersz  $i$  w macierzy  $S$  reprezentuje krotkę  $t_i$  (odpowiadającą relacji  $R_i$ ), która posiada symbole „a” w kolumnach odpowiadających atrybutom relacji  $R_i$  oraz symbole „b” w pozostałych kolumnach. Następnie algorytm przekształca wiersze tej macierzy (w pętli kroku 4.), tak aby reprezentowały krotki spełniające wszystkie zależności funkcyjne ze zbioru  $F$ . Na końcu kroku 4. dowolne dwa wiersze w macierzy  $S$  — reprezentujące dwie krotki w stanie relacji  $r$  — o równych wartościach co do lewostronnych atrybutów  $X$  zależności funkcyjnej  $X \rightarrow Y$  w zbiorze  $F$  są również zgodne co do wartości atrybutów prawostronnych  $Y$ . Można udowodnić, że po zakończeniu pętli z kroku 4., jeżeli dowolny wiersz macierzy  $S$  kończy się samymi symbolami „a”, to dekompozycja  $D$  posiada właściwość złączenia bezstratnego w kontekście zbioru  $F$ .

Z drugiej strony, jeżeli żaden wiersz nie kończy się samymi symbolami „a”, to dekompozycja  $D$  nie spełnia wymagań właściwości złączenia bezstratnego. W takim przypadku stan  $r$  relacji reprezentowany przez macierz  $S$  po zakończeniu działania algorytmu będzie przykładem stanu  $r$  relacji  $R$ , która spełnia zależności ze zbioru  $F$ , ale nie spełnia warunków złączenia bezstratnego. Stąd, taka relacja służy jako **kontraprzykład**, dowodzący, że dekompozycja  $D$  nie posiada właściwości złączenia bezstratnego w kontekście zbioru  $F$ . Należy zauważyć, że symbole „a” i „b” nie mają żadnego szczególnego znaczenia po zakończeniu działania algorytmu.

Na rysunku 15.1(a) przedstawiono sposób zastosowania algorytmu 15.3 względem dekompozycji schematu relacji PRAC\_PROJ z rysunku 14.3(b) na dwa schematy relacji PRAC\_PROJ1 i PRAC\_LOKALIZACJE, przedstawione na rysunku 14.5(a). Pętla z kroku 4. algorytmu nie może zmienić wartości żadnego symbolu „b” na „a”, stąd wynikowa macierz  $S$  nie posiada ani jednego wiersza z samymi symbolami „a”, co oznacza, że dekompozycja nie posiada właściwości złączenia bezstratnego.

Na rysunku 15.1(b) przedstawiono kolejną dekompozycję relacji PRAC\_PROJ (na relacje PRAC, PROJEKT oraz PRACUJE\_NAD), która posiada właściwości złączenia bezstratnego, natomiast na rysunku 15.1(c) przedstawiono sposób zastosowania algorytmu względem tej dekompozycji. Kiedy okazuje się, że wiersz zawiera same symbole „a”, wiemy, że dekompozycja posiada właściwość złączenia bezstratnego i możemy zaprzestać stosowania zależności funkcyjnych (krok 4. algorytmu) względem macierzy  $S$ .

### 15.2.4. Testowanie dekompozycji binarnych pod względem występowania właściwości złączenia nieaddytywnego

Algorytm 15.3 pozwala nam na sprawdzenie, czy określony rozkład  $D$  na  $n$  relacji zachowuje właściwość złączenia bezstratnego w kontekście zbioru zależności funkcyjnych  $F$ . Istnieje przypadek szczególny dekompozycji, noszący nazwę **dekompozycji binarnej** (ang. *binary decomposition*). Jest to rozkład relacji  $R$  na dwie relacje. W punkcie 14.5.1 opisany został test złączenia nieaddytywnego, łatwiejszy do zastosowania od algorytmu 15.3, ale *ograniczony* do dekompozycji binarnych. Zastosowaliśmy go do dekompozycji binarnej relacji UCZY (zgodnej z 3NF, ale niezgodnej z BCNF) na dwie relacje posiadające omawianą właściwość.

- (a)  $R = \{PESEL, IMIĘNAZWPRAC, NUMERPROJ, NAZWAPROJ, LOKALIZACJAPROJ, GODZINY\}$      $D = \{R_1, R_2\}$   
 $R_1 = PRAC\_LOKALIZACJE = \{IMIĘNAZWPRAC, LOKALIZACJAPROJ\}$   
 $R_2 = PRAC\_PROJ1 = \{PESEL, NUMERPROJ, GODZINY, NAZWAPROJ, LOKALIZACJAPROJ\}$

$F = \{PESEL \rightarrow IMIĘNAZWPRAC; NUMERPROJ \rightarrow \{NAZWAPROJ, LOKALIZACJAPROJ\};$   
 $\{PESEL, NUMERPROJ\} \rightarrow GODZINY\}$

	PESEL	IMIĘNAZWPRAC	NUMERPROJ	NAZWAPROJ	LOKALIZACJAPROJ	GODZINY
$R_1$	$b_{11}$	$a_2$	$b_{13}$	$b_{14}$	$a_5$	$b_{16}$
$R_2$	$a_1$	$b_{22}$	$a_3$	$a_4$	$a_5$	$a_6$

(brak zmian w macierzy po zastosowaniu zależności funkcyjnych)

- (b) 

PRAC		PROJEKT			PRACUJE_NAD		
PESEL	IMIĘNAZWPRAC	NUMERPROJ	NAZWAPROJ	LOKALIZACJAPROJ	PESEL	NUMERPROJ	GODZINY

- (c)  $R = \{PESEL, IMIĘNAZWPRAC, NUMERPROJ, NAZWAPROJ, LOKALIZACJAPROJ, GODZINY\}$      $D = \{R_1, R_2, R_3\}$   
 $R_1 = PRAC = \{PESEL, IMIĘNAZWPRAC\}$   
 $R_2 = PROJ = \{NUMERPROJ, NAZWAPROJ, LOKALIZACJAPROJ\}$   
 $R_3 = PRACUJE\_NAD = \{PESEL, NUMERPROJ, GODZINY\}$

$F = \{PESEL \rightarrow IMIĘNAZWPRAC; NUMERPROJ \rightarrow \{NAZWAPROJ, LOKALIZACJAPROJ\};$   
 $\{PESEL, NUMERPROJ\} \rightarrow GODZINY\}$

	PESEL	IMIĘNAZWPRAC	NUMERPROJ	NAZWAPROJ	LOKALIZACJAPROJ	GODZINY
$R_1$	$a_1$	$a_2$	$b_{13}$	$b_{14}$	$b_{15}$	$b_{16}$
$R_2$	$b_{21}$	$b_{22}$	$a_3$	$a_4$	$a_5$	$b_{26}$
$R_2$	$a_1$	$b_{32}$	$a_3$	$b_{34}$	$b_{35}$	$a_6$

(oryginalna macierz S na początku działania algorytmu)

	PESEL	IMIĘNAZWPRAC	NUMERPROJ	NAZWAPROJ	LOKALIZACJAPROJ	GODZINY
$R_1$	$a_1$	$a_2$	$b_{13}$	$b_{14}$	$b_{15}$	$b_{16}$
$R_2$	$b_{21}$	$b_{22}$	$a_3$	$a_4$	$a_5$	$b_{26}$
$R_2$	$a_1$	<del><math>b_{32}</math></del> $a_2$	$a_3$	<del><math>b_{34}</math></del> $a_4$	<del><math>b_{35}</math></del> $a_5$	$a_6$

(macierz S po zastosowaniu dwóch pierwszych zależności funkcyjnych – ostatni wiersz zawiera same symbole „a”, więc zaprzestajemy dalszych działań)

RYSUNEK 15.1. Sprawdzanie występowania właściwości złączenia nieaddytywnego w przypadku dekompozycji n-krotnych. (a) Przypadek 1.: Dekompozycja relacji PRAC\_PROJ na relacje PRAC\_PROJ1 i PRAC\_LOKALIZACJE nie spełnia warunków testu. (b) Dekompozycja relacji PRAC\_PROJ posiadająca właściwość złączenia nieaddytywnego. (c) Przypadek 2: Dekompozycja relacji PRAC\_PROJ na relacje PRAC, PROJEKT oraz PRACUJE\_NAD spełnia warunki testu

### 15.2.5. Kolejne dekompozycje o złączeniach nieaddytywnych

W toku procesu normalizacji do drugiej i trzeciej postaci normalnej, opisanego w podrozdziałach 14.3 i 14.4, Czytelnik poznał kolejne dekompozycje tych samych relacji. W celu sprawdzenia, czy dekompozycje te są nieaddytywne, musimy zapewnić występowanie innej właściwości, którą określa założenie 2.



**ZAŁOŻENIE 2. (zachowanie nieaddytywności w kolejnych dekompozycjach).**

Jeżeli dekompozycja  $D = \{R_1, R_2, \dots, R_m\}$  relacji  $R$  posiada właściwość złączenia bezstratnego (nieaddytywnego) w kontekście zbioru zależności funkcyjnych  $F$  na relacji  $R$  oraz jeżeli dekompozycja  $D_i = \{Q_1, Q_2, \dots, Q_k\}$  relacji  $R_i$  posiada właściwość złączenia nieaddytywnego w kontekście projekcji zbioru  $F$  na relację  $R_i$ , to dekompozycja  $D_2 = \{R_1, R_2, \dots, R_{i-1}, Q_1, Q_2, \dots, Q_k, R_{i+1}, \dots, R_m\}$  relacji  $R$  posiada właściwość złączenia nieaddytywnego w kontekście zbioru  $F$ .

## 15.3. Algorytmy projektowania schematów relacyjnych baz danych

Poniżej zostaną przedstawione dwa algorytmy służące do tworzenia dekompozycji relacyjnych na podstawie relacji uniwersalnych. Pierwszy algorytm rozkłada relację uniwersalną na relacje w 3NF z zachowaniem zależności i zapewnia właściwość złączenia nieaddytywnego. Drugi algorytm rozkłada schemat relacji uniwersalnej na schematy w postaci BCNF mające właściwość złączenia nieaddytywnego. Nie da się zaprojektować algorytmu tworzącego relacje w postaci BCNF i zapewniającego jednocześnie zachowanie zależności oraz właściwość złączenia nieaddytywnego.

### 15.3.1. Dekompozycja na schematy w trzeciej postaci normalnej z zachowaniem zależności i właściwością złączenia nieaddytywnego (bezstratnego)

Wiesz już, że *nie da się spełnić wszystkich trzech warunków*: (1) zagwarantować projekt niestratny (nieaddytywny), (2) zagwarantować zachowanie zależności i (3) utworzyć wszystkie relacje w postaci BCNF. Jak już podkreślaliśmy, pierwszy warunek jest konieczny i nie można go naruszyć. Drugi warunek jest pożądanym, ale niewymagany. Czasem trzeba z niego zrezygnować, jeśli celem jest uzyskanie postaci BCNF. Utracone pierwotne zależności funkcyjne można odtworzyć poprzez złączenie wyników dekompozycji. Tu przedstawimy algorytm, który spełnia warunki 1. i 2., ale gwarantuje tylko 3NF. Algorytm 15.4 tworzy dekompozycję  $D$  relacji  $R$  i ma następujące cechy:

- zachowuje zależności,
- zapewnia właściwość złączenia nieaddytywnego,
- sprawia, że każdy wynikowy schemat relacji po dekompozycji jest w 3NF.

**Algorytm 15.4.** Synteza relacyjna do trzeciej postaci normalnej z zachowaniem zależności i zapewnieniem właściwości złączenia nieaddytywnego

**Dane wejściowe:** Relacja uniwersalna  $R$  oraz zbiór zależności funkcyjnych  $F$  na atrybutach relacji  $R$ .

- (1) Znajdujemy pokrycie minimalne  $G$  zbioru  $F$  (wykorzystując algorytm 15.2).
- (2) Dla każdego lewostronnego zbioru  $X$  zależności funkcyjnej, która występuje w pokryciu  $G$ , tworzymy schemat relacji  $D$  o atrybutach  $\{X \cup \{A_1\} \cup \{A_2\} \dots \cup \{A_k\}\}$ ,



gdzie  $X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_k$  są jedynymi zależnościami w  $G$  posiadającymi  $X$  jako zbiór lewostronny ( $X$  jest kluczem tej relacji).

- (3) Jeżeli żaden ze schematów relacji w dekompozycji  $D$  nie zawiera klucza relacji  $R$ , w dekompozycji  $D$  tworzymy jeszcze jeden schemat relacji zawierający atrybuty tworzące klucz relacji  $R$  (do znalezienia klucza można wykorzystać algorytm 15.2(a)).
- (4) Z wynikowego zbioru relacji ze schematu eliminujemy nadmiarowe relacje. Relacja  $R$  jest nadmiarowa, jeśli  $R$  jest projekcją innej relacji  $S$  ze schematu (czyli  $R$  jest podzbiorem  $S$ )<sup>7</sup>.

Krok 3. algorytmu 15.4 polega na zidentyfikowaniu klucza  $K$  relacji  $R$ . Algorytm 15.2(a) może zostać wykorzystany do zidentyfikowania klucza  $K$  relacji  $R$  w oparciu o dany zbiór  $F$  zależności funkcyjnych. Należy zauważyć, że zbiorem zależności funkcyjnych wykorzystywanym do określenia klucza w algorytmie 15.2(a) mógłby być zbiór  $F$  lub  $G$ , gdyż są one równoważne.

**Przykład 1. do algorytmu 15.4.** Zastanów się nad następującą relacją uniwersalną:

$U(\text{PESELPRAC}, \text{NRPROJ}, \text{PENSJAPRAC}, \text{TELPRAC}, \text{NRDZIAŁU}, \text{NAZWAPROJ}, \text{LOKALIZACJAPROJ})$

PESELPRAC, PENSJAPRAC i TELPRAC oznaczają numer PESEL, wynagrodzenie i numer telefonu pracownika. NRPROJ, NAZWAPROJ i LOKALIZACJAPROJ to numer, nazwa i lokalizacja projektu. NRDZIAŁU to numer działu.

Występują tu następujące zależności:

FD1:  $\text{PESELPRAC} \rightarrow \{\text{PENSJAPRAC}, \text{TELPRAC}, \text{NRDZIAŁU}\}$

FD2:  $\text{NRPROJ} \rightarrow \{\text{NAZWAPROJ}, \text{LOKALIZACJAPROJ}\}$

FD3:  $\text{PESELPRAC}, \text{NRPROJ} \rightarrow \{\text{PENSJAPRAC}, \text{TELPRAC}, \text{NRDZIAŁU}, \text{NAZWAPROJ}, \text{LOKALIZACJAPROJ}\}$

Zgodnie z FD3 zbiór atrybutów  $\{\text{PESELPRAC}, \text{NRPROJ}\}$  reprezentuje klucz przedstawionej relacji uniwersalnej. Tak więc  $F$ , czyli zbiór zależności funkcyjnych, obejmuje zależności:  $\{\text{PESELPRAC} \rightarrow \text{PENSJAPRAC}, \text{TELPRAC}, \text{NRDZIAŁU}; \text{NRPROJ} \rightarrow \text{NAZWAPROJ}, \text{LOKALIZACJAPROJ}; \text{PESELPRAC}, \text{NRPROJ} \rightarrow \text{PENSJAPRAC}, \text{TELPRAC}, \text{NRDZIAŁU}, \text{NAZWAPROJ}, \text{LOKALIZACJAPROJ}\}$ .

W trakcie stosowania algorytmu 15.2 (do wykrywania pokrycia minimalnego) w kroku 3. okazuje się, że NRPROJ jest atrybutem nadmiarowym w zależności  $\text{PESELPRAC}, \text{NRPROJ} \rightarrow \text{PENSJAPRAC}, \text{TELPRAC}, \text{NRDZIAŁU}$ . Ponadto PESELPRAC jest nadmiarowy w zależności  $\text{PESELPRAC}, \text{NRPROJ} \rightarrow \text{NAZWAPROJ}, \text{LOKALIZACJAPROJ}$ . Tak więc pokrycie minimalne obejmuje tylko zależności FD1 i FD2 (FD3 jest nadmiarowa) i ma następującą postać (po połączeniu atrybutów o tej samej lewej stronie w jedną zależność funkcyjną):

Minimalne pokrycie  $G$ :  $\{\text{PESELPRAC} \rightarrow \text{PENSJAPRAC}, \text{TELPRAC}, \text{NRDZIAŁU}; \text{NRPROJ} \rightarrow \text{NAZWAPROJ}, \text{LOKALIZACJAPROJ}\}$

<sup>7</sup> Zauważ, że istnieje dodatkowy rodzaj zależności:  $R$  jest projekcją złączenia dwóch lub więcej relacji schematu. Tego typu nadmiar jest uznawany za zależność złączeniową, co opisano w podrozdziale 15.7. Dlatego technicznie może on występować w schemacie zgodnym z 3NF.

Drugi krok algorytmu 15.4 generuje relacje  $R_1$  i  $R_2$ :

$R_1(\underline{\text{PESELPRAC}}, \text{PENSJAPRAC}, \text{TELPRAC}, \text{NRDZIAŁU})$

$R_2(\underline{\text{NRPROJ}}, \text{NAZWAPROJ}, \text{LOKALIZACJAPROJ})$

W kroku 3. generowana jest relacja odpowiadająca kluczowi  $\{\text{PESELPRAC}, \text{NRPROJ}\}$  z U. Wynikowy projekt zawiera więc:

$R_1(\underline{\text{PESELPRAC}}, \text{PENSJAPRAC}, \text{TELPRAC}, \text{NRDZIAŁU})$

$R_2(\underline{\text{NRPROJ}}, \text{NAZWAPROJ}, \text{LOKALIZACJAPROJ})$

$R_3(\underline{\text{PESELPRAC}}, \underline{\text{NRPROJ}})$

W tym projekcie uzyskaliśmy obie pożądane cechy: zachowanie zależności i właściwość złączenia nieaddytywnego.

**Przykład 2. do algorytmu 14.3 (przypadek X).** Przyjrzyj się schematowi relacji DZIAŁKI1A z rysunku 14.13(a).

Założmy, że ta relacja została utworzona jako relacja uniwersalna U ( $\text{ID\_NIERUCHOMOŚCI}$ ,  $\text{NAZWA\_GMINY}$ ,  $\text{DZIAŁKA\#}$ ,  $\text{POWIERZCHNIA}$ ) o następujących zależnościach funkcyjnych:

FD1:  $\text{ID\_NIERUCHOMOŚCI} \rightarrow \text{DZIAŁKA\#}, \text{NAZWA\_GMINY}, \text{POWIERZCHNIA}$

FD2:  $\text{DZIAŁKA\#}, \text{NAZWA\_GMINY} \rightarrow \text{POWIERZCHNIA}, \text{ID\_NIERUCHOMOŚCI}$

FD3:  $\text{POWIERZCHNIA} \rightarrow \text{NAZWA\_GMINY}$

Na rysunku 14.13(a) te zależności to FD1, FD2 i FD3. Znaczenie atrybutów i skutki stosowania podanych zależności funkcyjnych opisano w podrozdziale 14.4. Dla ułatwienia utworzymy skróty nazw atrybutów (jako pierwsze litery) i zapiszmy zależności funkcyjne w postaci zbioru:

$F: \{I \rightarrow \text{DNP}, \text{DN} \rightarrow \text{PI}, \text{P} \rightarrow \text{N}\}$

Relacja uniwersalna ze skrótowymi nazwami atrybutów to U (I, N, D, P). Jeśli zastosujemy do F algorytm tworzenia pokrycia minimalnego 15.2, w kroku 2. najpierw otrzymamy zbiór F w następującej formie:

$F: \{I \rightarrow \text{D}, I \rightarrow \text{N}, I \rightarrow \text{P}, \text{DN} \rightarrow \text{P}, \text{DN} \rightarrow \text{I}, \text{P} \rightarrow \text{N}\}$

W tym zbiorze  $F$   $I \rightarrow \text{P}$  można wywnioskować z  $I \rightarrow \text{DN}$  i  $\text{DN} \rightarrow \text{P}$ .  $I \rightarrow \text{P}$  wynika więc z przechodniości i dlatego jest nadmiarowa. Możliwe pokrycie minimalne wygląda zatem tak:

Pokrycie minimalne  $G_X: \{I \rightarrow \text{DN}, \text{DN} \rightarrow \text{PI}, \text{P} \rightarrow \text{N}\}$

W kroku 2. algorytmu 15.4 tworzymy na podstawie tego pokrycia minimalnego projekt X (przed usunięciem nadmiarowych relacji):

Projekt X:  $R_1(\underline{\text{I}}, \text{D}, \text{N}), R_2(\underline{\text{D}}, \underline{\text{N}}, \text{P}, \text{I})$  i  $R_3(\underline{\text{P}}, \text{N})$

W kroku 4. algorytmu stwierdzamy, że  $R_3$  jest podzbiorem  $R_2$  ( $R_3$  zawsze jest projekcją  $R_2$ ; podobnie  $R_1$  jest projekcją  $R_2$ ). Dlatego dwie relacje są nadmiarowe. Tak więc schemat w postaci 3NF spełniający obie pożądane cechy wygląda po usunięciu nadmiarowych relacji tak:

Projekt X:  $R_2(\text{D}, \text{N}, \text{P}, \text{I})$

Jest on identyczny z relacją DZIAŁKI1A (ID\_NIERUCHOMOŚCI, NAZWA\_GMINY, DZIAŁKA#, POWIERZCHNIA), co do której w punkcie 14.4.2 stwierdziliśmy, że znajduje się w 3NF.

**Przykład 2. do algorytmu 15.4 (przypadek Y).** Zaczynamy od relacji uniwersalnej DZIAŁKI1A i tego samego co wcześniej zestawu zależności funkcyjnych. Drugi krok algorytmu znajdowania pokrycia minimalnego (algorytm 15.2) daje jak wcześniej:

$$F: \{I \rightarrow D, I \rightarrow N, I \rightarrow P, DN \rightarrow P, DN \rightarrow I, P \rightarrow N\}$$

Zależność  $DN \rightarrow P$  można uznać za nadmiarową, ponieważ z  $DN \rightarrow I, I \rightarrow P$  i reguły przechodniej wynika  $DN \rightarrow P$ . Ponadto można przyjąć, że nadmiarowa jest zależność  $I \rightarrow N$ , ponieważ z  $I \rightarrow P$  i  $P \rightarrow N$  wynika  $I \rightarrow N$  (zgodnie z regułą przechodnią). Uzyskujemy więc następujące pokrycie minimalne:

$$\text{Pokrycie minimalne } GY: \{I \rightarrow DP, DN \rightarrow I, P \rightarrow N\}$$

Inny projekt  $Y$ , który został tym razem wygenerowany przez algorytm, to:

$$\text{Projekt } Y: S_1 (\underline{I}, P, D), S_2 (\underline{D}, \underline{N}, I) \text{ i } S_3 (\underline{P}, N)$$

Warto zauważyć, że w tym projekcie występują trzy relacje w postaci 3NF. Żadnej z nich nie można uznać za nadmiarową według warunku z kroku 4. Wszystkie zależności funkcyjne z pierwotnego zbioru  $F$  zostają zachowane. Czytelnicy zauważą, że z trzech podanych relacji  $S_1$  i  $S_3$  zostały utworzone w postaci BCNF zgodnie z procedurą z podrozdziału 14.5 (z czego wynika, że  $S_2$  jest nadmiarowa, gdy występują  $S_1$  i  $S_3$ ). Nie można jednak usunąć relacji  $S_2$  ze zbioru trzech relacji w postaci 3NF, ponieważ  $S_2$  nie jest projekcją ani  $S_1$ , ani  $S_3$ . Łatwo dostrzec, że  $S_2$  jest prawidłową i sensowną relacją o dwóch kluczach kandydujących ( $D, N$ ) i atrybucie  $I$ . Warto też zauważyć, że w  $S_2$  zachowana jest zależność funkcyjna  $DN \rightarrow I$ , która zostanie utracona, jeśli ostateczny projekt będzie obejmował tylko relacje  $S_1$  i  $S_3$ . Dlatego projekt  $Y$  jest jednym z możliwych końcowych wyników zastosowania do podanej relacji uniwersalnej algorytmu 15.4 zwracającego relacje w postaci 3NF.

Istnienie dwóch sposobów zastosowania algorytmu 15.4 do tej samej relacji uniwersalnej o określonym zestawie zależności funkcyjnych ilustruje dwie rzeczy:

- Możliwe jest wygenerowanie różnych projektów w postaci 3NF, wychodząc od tego samego zestawu zależności funkcyjnych.
- Wyobrażalne jest, że w pewnych sytuacjach algorytm wygeneruje relacje zgodne z postacią BCNF i jednocześnie zachowujące zależności.

### 15.3.2. Dekompozycja ze złączeniem nieaddytywnym na schematy w postaci normalnej Boyce'a-Codda

Kolejny algorytm rozkłada relację uniwersalną  $R = \{A_1, A_2, \dots, A_n\}$  do postaci dekompozycji  $D = \{R_1, R_2, \dots, R_m\}$  takiej, że każda relacja  $R_i$  znajduje się w postaci BCNF oraz dekompozycja  $D$  posiada właściwość złączenia bezstratnego w kontekście zbioru  $F$ . Algorytm 15.5 wykorzystuje właściwość złączenia nieaddytywnego oraz założenie 2. (zachowania nieaddytywności w kolejnych dekompozycjach) w celu utworzenia dekompozycji ze złączeniem nieaddytywnym  $D = \{R_1, R_2, \dots, R_m\}$  relacji uniwersalnej  $R$  w oparciu o zbiór zależności funkcyjnych  $F$ , taki że każda relacja  $R_i$  w dekompozycji  $D$  znajduje się w postaci BCNF.

**Algorytm 15.5.** Dekompozycja relacyjna do postaci BCNF z właściwością złączenia nieaddytywnego

**Dane wejściowe:** Relacja uniwersalna  $R$  i zbiór zależności funkcyjnych  $F$  na atrybutach relacji  $R$ .

- (1) Ustawiamy  $D := \{R\}$ ;
- (2) Dopóki istnieje schemat relacji  $Q$  w dekompozycji  $D$ , który nie jest w postaci BCNF, powtarzamy:
 

$\{$   
 wybieramy schemat relacji  $Q$  z dekompozycji  $D$ , który nie jest w postaci BCNF;  
 znajdujemy zależność funkcyjną  $X \rightarrow Y$  w  $Q$ , która narusza warunki postaci BCNF;  
 zastępujemy schemat  $Q$  w dekompozycji  $D$  dwoma schematami relacji  $(Q-Y)$  oraz  $(X \cup Y)$ ;  
 $\}$

W każdym przebiegu pętli algorytmu 15.5 dokonujemy dekompozycji jednego schematu relacji  $Q$ , który nie jest w postaci BCNF, na dwa schematy relacji. Zgodnie z właściwością złączenia nieaddytywnego dla dekompozycji binarnych oraz założenia 2., dekompozycja  $D$  posiada właściwość złączenia nieaddytywnego. Po zakończeniu działania algorytmu wszystkie schematy relacji w  $D$  znajdują się w postaci BCNF. Podobnie, jeżeli zastosujemy powyższy algorytm względem schematu relacji UCZY z rysunku 14.14, zostanie on rozłożony na relacje UCZY1(WYKŁADOWCA, STUDENT) oraz UCZY2(WYKŁADOWCA, PRZEDMIOT) ze względu na fakt, że zależność FD2: WYKŁADOWCA  $\rightarrow$  PRZEDMIOT narusza warunki postaci BCNF.

W kroku 2. algorytmu 15.5 konieczne jest określenie, czy schemat relacji  $Q$  jest, czy nie jest w postaci BCNF. Jedną z metod pozwalających na to polega na sprawdzeniu dla każdej zależności funkcyjnej  $X \rightarrow Y$  ze schematu  $Q$ , czy domknięcie  $X^+$  zawiera wszystkie atrybuty schematu  $Q$ , co pozwala określić, czy  $X$  jest (nad)kluczem schematu  $Q$ . Inna technika bazuje na spostrzeżeniu, że zawsze, kiedy schemat relacji  $Q$  narusza warunki postaci BCNF, istnieje para takich atrybutów  $A$  i  $B$  w  $Q$ , że  $\{Q - \{A, B\}\} \rightarrow A$ . Obliczając domknięcie  $\{Q - \{A, B\}\}^+$  dla każdej pary atrybutów  $\{A, B\}$  schematu  $Q$  i sprawdzając, czy domknięcie zawiera atrybut  $A$  (lub  $B$ ), możemy określić, czy schemat  $Q$  znajduje się w postaci BCNF.

Należy zwrócić uwagę na to, że teoria dekompozycji ze złączeniem nieaddytywnym jest oparta na założeniu, iż w *atrybutach złączeniowych nie są dozwolone wartości puste*. W następnym punkcie opisujemy wybrane problemy, jakie wartości puste mogą powodować w dekompozycjach relacyjnych, i przedstawiamy ogólne omówienie zaprezentowanych algorytmów dekompozycji relacyjnej przez syntezę.

## 15.4. Problemy związane z wartościami pustymi i krotkami zawieszonymi oraz inne projekty relacyjne

W tym podrozdziale omawiamy kilka ogólnych kwestii związanych z problemami, jakie występują z powodu niewłaściwego podejścia do projektu relacyjnego.

### 15.4.1. Problemy związane z wartościami pustymi i krotkami zawieszonymi

W czasie projektowania schematu relacyjnej bazy danych należy z rozważą podejść do problemów związanych z występowaniem wartości pustych. Nie istnieje żadna w pełni satysfakcjonująca teoria z zakresu projektowania relacyjnego, która uwzględniałaby wartości puste. Jeden z problemów pojawia się wtedy, gdy pewne krotki posiadają wartości puste w atrybutach, które są używane do złączania pojedynczych relacji w ramach dekompozycji. W celu jego zilustrowania weźmy pod uwagę bazę danych przedstawioną na rysunku 15.2(a), gdzie występują dwie relacje: PRACOWNIK i DZIAŁ. Krotki dwóch ostatnich pracowników — *Begera* i *Bielawskiego* — reprezentują nowozatrudnionych pracowników, których jeszcze nie przydzielono do żadnego działu (zakładamy, że nie narusza to żadnych więzów integralności). Załóżmy, że chcemy otrzymać listę wartości (IMIĘNAZWPRAC, NAZWADZ) dla wszystkich pracowników. Jeżeli względem relacji PRACOWNIK i DZIAŁ zastosujemy operację złączenia naturalnego (NATURAL JOIN) (rysunek 15.2(b)), dwie wcześniej wspomniane krotki *nie* pojawiają się w wyniku. Operacja złączenia zewnętrznego (OUTER JOIN), omówiona w rozdziale 8., pozwala rozwiązać ten problem. Należy przypomnieć, że jeżeli wykonamy operację lewostronnego złączenia zewnętrznego na relacjach PRACOWNIK i DZIAŁ, krotki z relacji PRACOWNIK posiadające wartości pustą dla atrybutu złączenia pojawiają się jednak w wyniku, złączone z *wyimaginowanymi* krotkami relacji DZIAŁ, które posiadają same wartości puste dla swoich atrybutów. Na rysunku 15.2(c) przedstawiono wynik takiego złączenia.

Ogólnie rzecz biorąc, kiedy schemat relacyjnej bazy danych zostanie zaprojektowany tak, że dwie lub większa liczba relacji jest powiązana za pomocą kluczy obcych, należy zwrócić szczególną uwagę na możliwość występowania wartości pustych w tych kluczach obcych. Może to powodować nieoczekiwaną utratę informacji w przypadku zapytań, które uwzględniają złączenia na kluczach obcych. Ponadto jeżeli wartości puste występują w innych atrybutach, takich jak PENSJA, ich wpływ na działanie funkcji wewnętrznych powoduje, że należy z ostrożnością podchodzić do wyników funkcji takich jak SUM lub AVERAGE.

Podobny problem wiąże się z *krotkami zawieszonymi* (ang. *dangling tuples*), które mogą występować w sytuacji, gdy dekompozycję przeprowadzi się zbyt daleko. Załóżmy, że dokonaliśmy dalszego rozkładu relacji PRACOWNIK z rysunku 15.2(a) na relacje PRACOWNIK\_1 oraz PRACOWNIK\_2, przedstawione na rysunkach 15.3(a) i 15.3(b). Jeżeli względem relacji PRACOWNIK\_1 i PRACOWNIK\_2 zastosujemy operację złączenia naturalnego, otrzymamy oryginalną relację PRACOWNIK. Jednak możemy użyć alternatywnej reprezentacji, przedstawionej na rysunku 15.3(c), gdzie w relacji PRACOWNIK\_3 *nie uwzględniamy krotki*, jeżeli pracownika nie przypisano do żadnego działu (zamiast uwzględnić krotkę z wartością pustą dla atrybutu NRDZ, jak ma to miejsce w relacji PRACOWNIK\_2). Jeżeli użyjemy relacji PRACOWNIK\_3

(a)

## PRACOWNIK

IMIĘNAZWPRAC	PESEL	DATAUR	ADRES	NRDZ
Nowak, Jan	65010912345	1965-01-09	ul. Nowa 11, 00-900 Warszawa	5
Kowalski, Fryderyk	55120834598	1955-12-08	Al. Niepodległości 23, 44-100 Gliwice	5
Zielińska, Alicja	68011932512	1968-07-19	Pl. Wolności, 25-600 Kielce	4
Owsiak, Maria	41062013258	1941-06-20	ul. Stara 22, 09-000 Warszawa	4
Głębocki, Adam	62091502054	1962-09-15	ul. Długa 34A, 70-300 Łomża	5
Polakowski, Maciej	72073110039	1972-07-31	ul. Krótka 1, 50-550 Szczecin	5
Kuc, Waldemar	69032923149	1969-03-29	Pl. Powstańców 10B, 00-111 Warszawa	4
Szymkowiak, Krystyna	37111045873	1937-11-10	ul. Szeroka 8, 26-100 Starachowice	1
Beger, Krzysztof	65042699775	1965-04-26	ul. Wąska 222, 00-500 Warszawa	NULL
Bielawski, Karol	63010988664	1963-01-09	Al. Wyzwolenia 1A, 44-200 Gliwice	NULL

## Dział

NAZWADZ	NRDZ	PESELKIEROWNIKADZ
Badawczy	5	55120834598
Administracja	4	41062013258
Centrala	1	37111045873

(b)

IMIĘNAZWPRAC	PESEL	DATAUR	ADRES	NRDZ	NAZWADZ	PESELKIEROWNIKADZ
Nowak, Jan	65010912345	1965-01-09	ul. Nowa 11, 00-900 Warszawa	5	Badawczy	55120834598
Kowalski, Fryderyk	55120834598	1955-12-08	Al. Niepodległości 23, 44-100 Gliwice	5	Badawczy	55120834598
Zielińska, Alicja	68011932512	1968-07-19	Pl. Wolności, 25-600 Kielce	4	Administracja	41062013258
Owsiak, Maria	41062013258	1941-06-20	ul. Stara 22, 09-000 Warszawa	4	Administracja	41062013258
Głębocki, Adam	62091502054	1962-09-15	ul. Krótka 1, 50-550 Szczecin	5	Badawczy	55120834598
Polakowski, Maciej	72073110039	1972-07-31	ul. Krótka 1, 50-550 Szczecin	5	Badawczy	55120834598
Kuc, Waldemar	69032923149	1969-03-29	Pl. Powstańców 10B, 00-111 Warszawa	4	Administracja	41062013258
Szymkowiak, Krystyna	37111045873	1937-11-10	ul. Szeroka 8, 26-100 Starachowice	1	Centrala	37111045873

(c)

IMIĘNAZWPRAC	PESEL	DATAUR	ADRES	NRDZ	NAZWADZ	PESELKIEROWNIKADZ
Nowak, Jan	65010912345	1965-01-09	ul. Nowa 11, 00-900 Warszawa	5	Badawczy	55120834598
Kowalski, Fryderyk	55120834598	1955-12-08	Al. Niepodległości 23, 44-100 Gliwice	5	Badawczy	55120834598
Zielińska, Alicja	68011932512	1968-07-19	Pl. Wolności, 25-600 Kielce	4	Administracja	41062013258
Owsiak, Maria	41062013258	1941-06-20	ul. Stara 22, 09-000 Warszawa	4	Administracja	41062013258
Głębocki, Adam	62091502054	1962-09-15	ul. Długa 34A, 70-300 Łomża	5	Badawczy	55120834598
Polakowski, Maciej	72073110039	1972-07-31	ul. Krótka 1, 50-550 Szczecin	5	Badawczy	55120834598
Kuc, Waldemar	69032923149	1969-03-29	Pl. Powstańców 10B, 00-111 Warszawa	4	Administracja	41062013258
Szymkowiak, Krystyna	37111045873	1937-11-10	ul. Szeroka 8, 26-100 Starachowice	1	Centrala	37111045873
Beger, Krzysztof	65042699775	1965-04-26	ul. Wąska 222, 00-500 Warszawa	NULL	NULL	NULL
Bielawski, Karol	63010988664	1963-01-09	Al. Wyzwolenia 1A, 44-200 Gliwice	NULL	NULL	NULL

RYSUNEK 15.2. Problemy dotyczące złączeń z wartościami pustymi. (a) Niektóre krotki relacji PRACOWNIK zawierają wartość pustą dla atrybutu złączenia NRDZ. (b) Wynik zastosowania operacji złączenia naturalnego (NATURAL JOIN) względem relacji PRACOWNIK i DZIAŁ. (c) Wynik zastosowania operacji lewostronnego złączenia zewnętrznego (LEFT OUTER JOIN) względem relacji PRACOWNIK i DZIAŁ.

zamiast PRACOWNIK\_2 i zastosujemy operację złączenia naturalnego (NATURAL JOIN) względem relacji PRACOWNIK\_1 i PRACOWNIK\_3, krotki dla pracowników *Begera* i *Bielawskiego* nie pojawią się w wyniku. Określa się je mianem **krotek zawieszonych**, ponieważ są reprezentowane tylko w jednej z dwóch relacji reprezentujących pracowników, a stąd są tracone w razie zastosowania operacji złączenia wewnętrznego (INNER JOIN).



(a) PRACOWNIK\_1

IMIĘNAZWPRAC	PESEL	DATAUR	ADRES
Nowak, Jan	65010912345	1965-01-09	ul. Nowa 11, 00-900 Warszawa
Kowalski, Fryderyk	55120834598	1955-12-08	Al. Niepodległości 23, 44-100 Gliwice
Zielińska, Alicja	68011932512	1968-07-19	Pl. Wolności, 25-600 Kielce
Owsiak, Maria	41062013258	1941-06-20	ul. Stara 22, 09-000 Warszawa
Głębocki, Adam	62091502054	1962-09-15	ul. Długa 34A, 70-300 Łomża
Polakowski, Maciej	72073110039	1972-07-31	ul. Krótka 1, 50-550 Szczecin
Kuc, Waldemar	69032923149	1969-03-29	Pl. Powstańców 10B, 00-111 Warszawa
Szymkowiak, Krystyna	37111045873	1937-11-10	ul. Szeroka 8, 26-100 Starachowice
Beger, Krzysztof	65042699775	1965-04-26	ul. Wąska 222, 00-500 Warszawa
Bielawski, Karol	63010988664	1963-01-09	Al. Wyzwolenia 1A, 44-200 Gliwice

(b) PRACOWNIK\_2

PESEL	NRDZ
65010912345	5
55120834598	5
68011932512	4
41062013258	4
62091502054	5
72073110039	5
69032923149	4
37111045873	1
65042699775	NULL
63010988664	NULL

(c) PRACOWNIK\_3

PESEL	NRDZ
65010912345	5
55120834598	5
68011932512	4
41062013258	4
62091502054	5
72073110039	5
69032923149	4
37111045873	1

RYСУNEK 15.3. Problem „krotek zawieszonych”. (a) Relacja PRACOWNIK\_1 (zawierająca wszystkie atrybuty relacji PRACOWNIK z rysunku 11.2(a) oprócz NRDZ). (b) Relacja PRACOWNIK\_2 (zawierająca atrybut NRDZ z wartościami pustymi). (c) Relacja PRACOWNIK\_3 (zawierająca atrybut NRDZ, ale nie uwzględniająca krotek, dla których jego wartość jest pusta)

## 15.4.2. Omówienie algorytmów normalizacyjnych i innych projektów relacyjnych

Jeden z problemów związanych z opisanymi algorytmami normalizacyjnymi polega na tym, że projektant bazy danych musi najpierw określić *wszystkie* istotne zależności funkcyjne występujące wśród atrybutów bazy danych. Nie jest to zadanie łatwe w przypadku rozbudowanej bazy danych liczącej setki atrybutów. Nieokreślenie jednej lub dwóch ważnych zależności może spowodować wygenerowanie niepoprawnego projektu. Kolejny problem wiąże się z tym, że, ogólnie rzecz biorąc, algorytmy te *nie są deterministyczne*. Przykładowo, *algorytmy syntezy* (algorytm 15.4 oraz 15.5) wymagają określenia pokrycia minimalnego  $G$  dla zbioru zależności funkcyjnych  $F$ . Ze względu na fakt, że w ogólnym przypadku może istnieć wiele pokryć minimalnych danego zbioru  $F$  (co przedstawiono w przykładzie 2. dotyczą-



cym algorytmu 15.4), algorytm może dawać różne projekty wynikowe w zależności od użytego pokrycia minimalnego. Niektóre z nich mogą być niepożądane. *Algorytm dekompozycji* (algorytm 15.5) zależy od porządku, w jakim zostaną do niego przekazane zależności funkcyjne w celu sprawdzenia naruszeń postaci BCNF. I tu istnieje możliwość, że dla tego samego zbioru zależności funkcyjnych będzie można otrzymać wiele różnych projektów, w zależności od kolejności, w jakiej takie zależności będą rozpatrywane w odniesieniu do naruszeń postaci BCNF. Niektóre projekty mogą okazać się o wiele lepsze od innych.

Nie zawsze można znaleźć dekompozycję na zachowującą zależności schematy relacji w postaci BCNF (a nie w uzyskiwanej w algorytmie 15.4 postaci 3NF). Można sprawdzić poszczególne schematy relacji w postaci 3NF, aby ustalić, czy są zgodne z BCNF. Jeśli któryś schemat relacji  $R_i$  nie jest w postaci BCNF, można poddać go dalszej dekompozycji lub pozostawić w postaci 3NF (z możliwymi anomaliami aktualizacji). Na podstawie projektowania metodą wstępującą pokazaliśmy, że różne pokrycia minimalne w przypadkach  $X$  i  $Y$  z przykładu 2. do algorytmu 15.4 dają różne zbiory relacji. Wersja  $X$  dała relację DZIAŁKIIA(ID\_NIERUCHOMOŚCI#, NAZWA\_GMINY, DZIAŁKA#, POWIERZCHNIA), która jest w postaci 3NF, ale nie jest zgodna z BCNF. W wersji  $Y$  uzyskano trzy relacje:  $S_1$ (ID\_NIERUCHOMOŚCI#, POWIERZCHNIA, DZIAŁKA#),  $S_2$ (DZIAŁKA#, NAZWA\_GMINY, ID\_NIERUCHOMOŚCI#) i  $S_3$ (POWIERZCHNIA, NAZWA\_GMINY). Po przetestowaniu każdej z tych trzech relacji okazuje się, że są one w postaci BCNF. Wcześniej zobaczyłeś też, że po zastosowaniu algorytmu 15.5 do relacji DZIAŁKIIY w celu jej dekompozycji na relacje w postaci BCNF wynikowy projekt obejmował tylko relacje  $S_1$  i  $S_3$ . Przykładowe przypadki ( $X$  i  $Y$ ) oparte na różnych pokryciach minimalnych tego samego schematu uniwersalnego są wystarczającym dowodem na to, że zaprezentowane w podrozdziale 15.3 algorytmy projektowania metodą wstępującą mogą prowadzić do uzyskania różnych projektów.

W tabeli 15.1 zawarto podsumowanie właściwości algorytmów omówionych dotąd w niniejszym rozdziale.

TABELA 15.1. Podsumowanie algorytmów omówionych w tym rozdziale

Algorytm	Dane wejściowe	Dane wyjściowe	Właściwości/ cel działania	Uwagi
15.1	Atrybut lub zbiór atrybutów $X$ i zbiór zależności funkcyjnych $F$	Zbiór atrybutów w domknięciu $X$ dla zbioru $F$	Ustalenie wszystkich atrybutów, które można funkcyjnie uzyskać na podstawie $X$	Domknięcie klucza to cała relacja
15.2	Zbiór $F$ zależności funkcyjnych	Pokrycie minimalne zależności funkcyjnych	Wyznaczenie pokrycia minimalnego dla zbioru zależności $F$	Może istnieć wiele pokryć minimalnych (zależą one od kolejności wyboru zależności funkcyjnych)
15.2a	Schemat relacji $R$ ze zbiorem zależności funkcyjnych $F$	Klucz $K$ relacji $R$	Znalezienie klucza $K$ (będącego podzbiorem $R$ )	Cała relacja $R$ zawsze jest domyślnym nadkluczem

TABELA 15.1. Podsumowanie algorytmów omówionych w tym rozdziale — ciąg dalszy

Algorytm	Dane wejściowe	Dane wyjściowe	Właściwości/ cel działania	Uwagi
15.3	Dekompozycja $D$ relacji $R$ oraz zbiór $F$ zależności funkcyjnych	Wartość logiczna: potwierdzenie lub zaprzeczenie występowania właściwości złączenia nieaddytywnego	Sprawdzenie występowania dekompozycji z właściwością złączenia nieaddytywnego	Opis prostszego testu złączenia nieaddytywnego dla dekompozycji binarnych zawarto w podrozdziale 14.5
15.4	Relacja $R$ i zbiór $F$ zależności funkcyjnych	Zbiór relacji w trzeciej postaci normalnej	Właściwość złączania nieaddytywnego i zachowanie zależności	Może nie zapewniać BCNF, ale uzyskuje wszystkie pożądane właściwości i 3NF
15.5	Relacja $R$ i zbiór $F$ zależności funkcyjnych	Zbiór relacji w postaci normalnej Boyce’a-Codda	Dekompozycja ze złączeniem nieaddytywnym	Brak gwarancji zachowania zależności
15.6	Relacja $R$ i zbiór $F$ zależności funkcyjnych oraz wielowartościowych	Zbiór relacji w 4NF	Dekompozycja ze złączeniem nieaddytywnym	Brak gwarancji zachowania zależności

## 15.5. Dalsze omówienie zależności wielowartościowych i 4NF

W podrozdziale 14.6 wprowadziliśmy i zdefiniowaliśmy zależności wielowartościowe i posłużyliśmy się nimi do zdefiniowania czwartej postaci normalnej. Tu przedstawiamy reguły wnioskowania dla zależności wielowartościowych, aby omówienie tych zależności było kompletne.

### 15.5.1. Reguły wnioskowania dla zależności funkcyjnych i wielowartościowych

Podobnie jak w przypadku zależności funkcyjnych, także dla zależności wielowartościowych zdefiniowano reguły wnioskowania. Lepszym rozwiązaniem jest jednak opracowanie jednolitej struktury zawierającej zarówno zależności funkcyjne, jak i wielowartościowe, tak aby oba rodzaje więzów móc rozpatrywać wspólnie. Poniższe reguły wnioskowania o zależnościach funkcyjnych i wielowartościowych na podstawie danego zbioru zależności. Zakładamy, że wszystkie atrybuty zawierają się w schemacie relacji *uniwersalnej*  $R = \{A_1, A_2, ..., A_n\}$  oraz że  $X, Y, Z$  i  $W$  są podzbiórami relacji  $R$ :

RW1 (reguła zwrotna dla zależności funkcyjnych): jeżeli  $X \supseteq Y$ , to  $X \rightarrow Y$ .

RW2 (reguła zwiększenia dla zależności funkcyjnych):  $\{X \rightarrow Y\} \models XZ \rightarrow YZ$ .

RW3 (reguła przechodnia dla zależności funkcyjnych):  $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$ .

RW4 (reguła uzupełnienia dla zależności wielowartościowych):

$\{X \twoheadrightarrow Y\} \models \{X \twoheadrightarrow (R - XY)\}$ .

RW5 (reguła zwiększenia dla zależności wielowartościowych):

jeżeli  $X \twoheadrightarrow Y$  oraz  $W \supseteq Z$ , to  $WX \twoheadrightarrow YZ$ .

RW6 (reguła przechodnia dla zależności wielowartościowych):

$\{X \twoheadrightarrow Y, Y \twoheadrightarrow Z\} \models X \twoheadrightarrow (Z - Y)$ .

RW7 (reguła replikacji zależności funkcyjnych do zależności wielowartościowych):

$\{X \rightarrow Y\} \models X \twoheadrightarrow Y$ .

RW8 (reguła scalenia dla zależności funkcyjnych i wielowartościowych): Jeżeli  $X \twoheadrightarrow Y$  oraz istnieje zbiór  $W$  o takich właściwościach, że (a) zbiór  $W \cap Y$  jest pusty, (b)  $W \rightarrow Z$  oraz (c)  $Y \supseteq Z$ , to  $X \rightarrow Z$ .

Reguły od RW1 do RW3 to reguły wnioskowania Armstronga obowiązujące tylko w przypadku zależności funkcyjnych. Reguły od RW4 do RW6 to reguły wnioskowania związane tylko z zależnościami wielowartościowymi. Reguły RW7 i RW8 są związane i z zależnościami funkcyjnymi, i wielowartościowymi. W szczególności, reguła RW7 określa, że zależność funkcyjna stanowi *przypadek szczególny* zależności wielowartościowej. Oznacza to, że każda zależność funkcyjna jest również zależnością wielowartościową, ponieważ spełnia warunki definicji formalnej zależności wielofunkcyjnej. Jednak z ową równoważnością wiąże się pewna pułapka. Zależność funkcyjna  $X \rightarrow Y$  jest zależnością wielowartościową  $X \twoheadrightarrow Y$  przy *dodatkowym niejawnym ograniczeniu*, określającym, że najwyżej jedna wartość  $Y$  jest powiązana z każdą wartością  $X^8$ . Dla danego zbioru  $F$  zależności funkcyjnych i wielowartościowych określonego na relacji  $R = \{A_1, A_2, \dots, A_n\}$  można używać reguł wnioskowania od RW1 do RW8 w celu wywnioskowania (pełnego) zbioru wszystkich zależności (funkcyjnych lub wielowartościowych)  $F^+$ , które będą spełnione w każdym stanie  $r$  relacji  $R$  spełniającej  $F$ . Ponownie zbiór  $F^+$  określamy mianem **domknięcia** zbioru  $F$ .

## 15.5.2. Jeszcze o czwartej postaci normalnej

Oto nowa wersja definicji **czwartej postaci normalnej (4NF)** z podrozdziału 14.6:

**Definicja.** Schemat relacji  $R$  znajduje się w **czwartej postaci normalnej** w kontekście zbioru zależności  $F$  (zawierającego zarówno zależności funkcyjne, jak i wielowartościowe), jeżeli dla każdej *niebanalnej* zależności wielowartościowej  $X \twoheadrightarrow Y$  w domknięciu  $F^+$  zbiór  $X$  jest nadkluczem relacji  $R$ .

W celu zilustrowania ważności czwartej postaci normalnej na rysunku 15.4(a) przedstawiono relację PRAC z dodatkowym pracownikiem o nazwisku *Wiśniewski*, który ma troje członków rodziny (o imionach *Rafał*, *Joanna* oraz *Robert*) i pracuje nad czterema różnymi projektami ( $W, X, Y$  i  $Z$ ). W relacji PRAC na rysunku 15.4(a) występuje 16 krotek. Jeżeli

<sup>8</sup> Oznacza to, że zbiór wartości  $Y$  zdeterminowany wartością  $X$  musi być *zbiorem jednowartościowym*. Stąd w praktyce zależności funkcyjnej niemal nigdy nie traktuje się jako zależności wielowartościowej.

(a) PRAC

<u>NAZWAPRAC</u>	<u>NAZWAPROJ</u>	<u>IMIĘCZŁRODZ</u>
Nowak	X	Jan
Nowak	Y	Anna
Nowak	X	Anna
Nowak	Y	Jan
Wiśniewski	W	Rafał
Wiśniewski	X	Rafał
Wiśniewski	Y	Rafał
Wiśniewski	Z	Rafał
Wiśniewski	W	Joanna
Wiśniewski	X	Joanna
Wiśniewski	Y	Joanna
Wiśniewski	Z	Joanna
Wiśniewski	W	Robert
Wiśniewski	X	Robert
Wiśniewski	Y	Robert
Wiśniewski	Z	Robert

(b) PRAC\_PROJEKTY

<u>NAZWAPRAC</u>	<u>NAZWAPROJ</u>
Nowak	X
Nowak	Y
Wiśniewski	W
Wiśniewski	X
Wiśniewski	Y
Wiśniewski	Z

PRAC\_CZŁRODZINY

<u>NAZWAPRAC</u>	<u>IMIĘCZŁRODZ</u>
Nowak	Anna
Nowak	Jan
Wiśniewski	Rafał
Wiśniewski	Joanna
Wiśniewski	Robert

RYSUNEK 15.4. Dekompozycja stanu relacji PRAC, która nie jest w czwartej postaci normalnej. (a) Relacja PRAC z dodatkowymi krotkami. (b) Dwie odpowiednie relacje znajdujące się w czwartej postaci normalnej: PRAC\_PROJEKTY i PRAC\_CZŁRODZINY

relację PRAC rozłożymy na relacje PRAC\_PROJEKTY oraz PRAC\_CZŁRODZINY, tak jak na rysunku 15.4(b), musimy przechować w sumie 11 krotek w obu relacjach. Dekompozycja pozwala nie tylko zaoszczędzić na zajętości pamięci, ale również uniknąć występowania anomalii aktualizacji związanych z zależnościami wielowartościowymi. Przykładowo, jeżeli *Wiśniewski* rozpocznie pracę nad nowym projektem *P*, trzeba uwzględnić *trzy* nowe krotki w relacji PRAC — jedną dla każdego członka rodziny. Jeżeli zapomnimy o wstawieniu którejs z nich, relacja będzie naruszać zależność wielowartościową i stanie się niespójna o tyle, że będzie błędnie implikować istnienie związku między projektem a członkiem rodziny.

Jeżeli relacja zawiera niebanalne zależności wielowartościowe, wówczas operacje wstawiania, usuwania i aktualizowania wykonywane na pojedynczych krotkach mogą powodować modyfikację dodatkowych krotek oprócz tych, które bezpośrednio nas interesują. W razie niepoprawnej obsługi operacji aktualizowania znaczenie relacji może ulec zmianie. Jednakże po znormalizowaniu do postaci 4NF anomalie takie znikają. Na przykład w celu dodania informacji o tym, że *Wiśniewski* zostaje przypisany do projektu *P*, należy wstawić tylko jedną krotkę do znajdującej się w czwartej postaci normalnej relacji PRAC\_PROJEKTY.

Relacja PRAC z rysunku 14.15(a) nie jest w postaci 4NF, ponieważ reprezentuje dwa *niezależne* związki typu jeden do wielu — jeden między pracownikami a projektami, nad którymi pracują, i drugi między pracownikami a członkami ich rodzin. Niekiedy wśród trzech encji posiadamy związek, który jest zależny od wszystkich trzech encji, na przykład tak jak w przypadku relacji DOSTAWA przedstawionej na rysunku 14.15(c) (na razie

bierzemy pod uwagę tylko krotki znajdujące się *nad* linią przerywaną). W tym przypadku krotka reprezentuje dostawcę dostarczającego odpowiednią część dla *określonego projektu*, więc nie występują tu *żadne* niebanalne zależności wielowartościowe. Relacja DOSTAWA znajduje się już w postaci 4NF i nie należy poddawać jej dekompozycji.

### 15.5.3. Dekompozycja ze złączeniem nieaddytywnym na relacje w czwartej postaci normalnej

Zawsze kiedy dokonujemy dekompozycji schematu relacji  $R$  na relacje  $R_1 = (X \cup Y)$  oraz  $R_2 = (X - Y)$  w oparciu o zależność wielowartościową  $X \twoheadrightarrow Y$ , która jest zachowana w relacji  $R$ , dekompozycja posiada właściwość złączenia nieaddytywnego. Można udowodnić, że jest to warunek konieczny i wystarczający dekompozycji schematu na dwa schematy posiadające właściwość złączenia nieaddytywnego, co określa właściwość  $ZN'$ , stanowiąca dalsze uogólnienie podanej wcześniej właściwości złączenia nieaddytywnego z punktu 14.5.1. Właściwość złączenia nieaddytywnego dotyczyła jedynie zależności funkcyjnych, natomiast zależność  $ZN'$  dotyczy zarówno zależności funkcyjnych, jak i wielowartościowych (warto przypomnieć w tym miejscu, że zależności funkcyjne są zależnościami wielowartościowymi).

**WŁAŚCIWOŚĆ  $ZN'$ .** Schematy relacji  $R_1$  i  $R_2$  tworzą dekompozycję ze złączeniem nieaddytywnym relacji  $R$  w kontekście zbioru  $F$  zależności funkcyjnych *oraz* wielowartościowych wtedy i tylko wtedy, gdy:

$$(R_1 \cap R_2) \twoheadrightarrow (R_1 - R_2)$$

lub, analogicznie, wtedy i tylko wtedy, gdy:

$$(R_1 \cap R_2) \twoheadrightarrow (R_2 - R_1)$$

Możemy użyć nieco zmodyfikowanej wersji algorytmu 15.5 w celu opracowania algorytmu 15.7, który będzie służył do tworzenia dekompozycji ze złączeniem nieaddytywnym na schematy relacji będące w czwartej postaci normalnej (a nie postaci BCNF). Podobnie jak w przypadku algorytmu 15.5, tak i algorytm 15.6 *nie zawsze* daje w wyniku dekompozycję zachowującą zależności funkcyjne.

**Algorytm 15.6.** Dekompozycja relacyjna do postaci 4NF z właściwością złączenia nieaddytywnego

**Dane wejściowe:** Relacja uniwersalna  $R$  i zbiór zależności funkcyjnych i wielowartościowych  $F$ .

- (1) Ustawiamy  $D := \{R\}$ ;
- (2) Dopóki istnieje schemat relacji  $Q$  w dekompozycji  $D$ , który nie jest w postaci 4NF, powtarzamy:
  - { wybieramy schemat relacji  $Q$  z dekompozycji  $D$ , który nie jest w postaci 4NF;
  - znajdujemy niebanalną zależność wielowartościową  $X \twoheadrightarrow Y$  w  $Q$ , która narusza warunki postaci 4NF;
  - zastępujemy schemat  $Q$  w dekompozycji  $D$  dwoma schematami relacji  $(Q - Y)$  oraz  $(X \cup Y)$ ;
  - };

## 15.6. Inne zależności i postaci normalne

### 15.6.1. Zależności złączeniowe i piąta postać normalna

W podrozdziale 14.7 przedstawiliśmy już inny rodzaj zależności — zależność złączeniową. Powstaje ona, gdy relację można rozdzielić na zbiór projekcji, które można złączyć ponownie w pierwotną relację. Po zdefiniowaniu zależności złączeniowej zdefiniowaliśmy w podrozdziale 14.7 opartą na niej piątą postać normalną. Piąta postać normalna jest też nazywana PJNF (ang. *project join normal form*; Fagin, 1979). W praktyce problem z tą i innymi dodatkowymi zależnościami (i powiązаныmi postaciami normalnymi takimi jak DKNF; patrz punkt 15.6.3) polega na tym, że trudno jest je znaleźć.

Ponadto nie istnieją zestawy skutecznych i kompletnych reguł wnioskowania związanych z tymi zależnościami. W dalszej części podrozdziału przedstawiamy inne wybrane rodzaje zidentyfikowanych zależności. Wśród nich często stosowane są zależność zawierania oraz zależności oparte na funkcjach arytmetycznych i podobnych.

### 15.6.2. Zależności zawierania

Zależności zawierania zdefiniowano w celu sformalizowania dwóch rodzajów więzów międzyrelacyjnych:

- więzy klucza obcego (integralności odwołań) nie mogą być określane jako zależności funkcyjne lub wielowartościowe, ponieważ wiążą one atrybuty należące do różnych relacji;
- więzy między dwiema relacjami, które reprezentują związek klasy-podklasy (patrz rozdziały 4. i 9.), również nie posiadają formalnej definicji w kontekście zależności funkcyjnych, wielowartościowych lub złączeniowych.

**Definicja. Zależność zawierania** (ang. *inclusion dependency*)  $R.X < S.Y$  między dwoma zbiorami atrybutów —  $X$  dla schematu relacji  $R$  oraz  $Y$  dla schematu relacji  $S$  — określa więzy, które w dowolnym momencie, kiedy  $r$  jest stanem relacji  $R$ , zaś  $s$  — stanem relacji  $S$ , stanowią, że:

$$\pi_x(r(R)) \subseteq \pi_y(s(S))$$

Związek  $\subseteq$  (podzbioru) nie musi oznaczać podzbioru właściwego. Oczywiście, zbiory atrybutów, na których określi się zależność zawierania —  $X$  dla  $R$  oraz  $Y$  dla  $S$  — muszą posiadać tę samą liczbę atrybutów. Poza tym, dziedziny każdej z par odpowiednich atrybutów powinny być zgodne. Przykładowo, jeżeli  $X = \{A_1, A_2, \dots, A_n\}$  oraz  $Y = \{B_1, B_2, \dots, B_n\}$ , jedną z możliwości określenia takiej zgodności jest zapewnienie, aby  $\text{dom}(A_i)$  było *kompatybilne* z  $\text{dom}(B_i)$  dla  $1 \leq i \leq n$ . W takim przypadku mówimy, że  $A_i$  **odpowiada** (ang. *corresponds*)  $B_i$ .

Możemy na przykład określić następujące zależności zawierania dla schematu relacyjnego z rysunku 14.1:

```

DZIAŁ.PESELKIEROWNIKA < PRACOWNIK.PESEL
PRACUJE_NAD.PESEL < PRACOWNIK.PESEL
PRACOWNIK.NUMERDZ < WYDZIAŁ.NUMERDZ
PROJEKT.NRDZ < DZIAŁ.NUMERDZ
PRACUJE_NAD.NUMERPROJ < PROJEKT.NUMERPROJ
LOKALIZACJE_DZ.NUMERDZ < DZIAŁ.NUMERDZ

```

Wszystkie powyższe zależności zawierania reprezentują **więzy integralności odwołań** (ang. *referential integrity constraints*). Zależności zawierania można również używać w celu reprezentowania **związków klas-podklas** (ang. *class/subclass relationships*). Na przykład na schemacie relacji z rysunku 9.6 można określić następujące zależności zawierania:

```

PRACOWNIK.PESEL < OSOBA.PESEL
ABSOLWENT.PESEL < OSOBA.PESEL
STUDENT.PESEL < OSOBA.PESEL

```

Podobnie jak w przypadku innych rodzajów zależności, można określić *reguły wnioskowania zależności zawierania* (ang. *inclusion dependencies inference rules, IDIR*). Poniżej podano trzy przykłady takich reguł:

IDIR1 (zwrotności):  $R.X < R.X$

IDIR2 (odpowiedniości atrybutów): Jeżeli  $R.X < S.Y$ , gdzie  $X = \{A_1, A_2, \dots, A_n\}$  i  $Y = \{B_1, B_2, \dots, B_n\}$  oraz  $A_i$  odpowiada  $B_i$ , to  $R.A_i < S.B_i$  dla  $1 \leq i \leq n$ .

IDIR3 (przechodności): Jeżeli  $R.X < S.Y$  oraz  $S.Y < T.Z$ , to  $R.X < T.Z$ .

W przypadku powyższych reguł wnioskowania udowodniono, że są one logiczne i kompletne dla zależności zawierania. Jak dotąd nie zdefiniowano żadnych postaci normalnych w oparciu o zależności zawierania.

### 15.6.3. Zależności funkcyjne oparte na funkcjach i procedurach arytmetycznych

Czasem niektóre atrybuty relacji mogą być powiązane ze sobą funkcjami arytmetycznymi lub bardziej skomplikowanymi relacjami funkcyjnymi. Jeśli z każdym  $X$  powiązana jest unikatowa wartość  $Y$ , można przyjąć, że istnieje zależność funkcyjna  $X \rightarrow Y$ . Przykładowo, w relacji:

```

POZYCJA_ZAMÓWIENIA (ZAMÓWIENIE#, PRODUKT#, SZTUK, CENA_JEDNOST, CENA_SUMA,
CENA_RABAT)

```

każda krotka reprezentuje pozycję z zamówienia (z określoną liczbą sztuk i ceną jednostkową). W tej relacji zależność  $(SZTUK, CENA_JEDNOST) \rightarrow CENA_SUMA$  jest wyrażona wzorem:

$$CENA\_SUMA = CENA\_JEDNOST * SZTUK$$

Występuje więc unikatowa wartość  $CENA\_SUMA$  dla każdej pary  $(SZTUK, CENA\_JEDNOST)$ , co jest zgodne z definicją zależności funkcyjnej.

Ponadto może istnieć procedura, która uwzględni rabat zależny od liczby sztuk, rodzaj produktu i inne aspekty, aby obliczyć cenę po rabacie na podstawie liczby zamówionych sztuk. Można więc stwierdzić, że:



(PRODUKT#, SZTUK, CENA\_JEDNOST) → CENA\_RABAT

lub

(PRODUKT#, SZTUK, CENA\_SUMA) → CENA\_RABAT

Do sprawdzania tych zależności funkcyjnych potrzebna może być bardziej skomplikowana procedura OBLICZ\_CENĘ\_KOŃCOWĄ. Choć technicznie tego rodzaju zależności funkcyjne występują w większości relacji, w trakcie normalizacji nie przywiązuje się do nich większej wagi. Jednak mogą być one istotne w trakcie wczytywania relacji i przetwarzania zapytań, ponieważ zapełnianie i pobieranie atrybutu występującego po prawej stronie zależności wymaga wykonania procedur podobnych do opisanych.

### 15.6.4. Postać normalna klucza dziedziny

Nie istnieje żadna ścisła i prosta reguła dotycząca definiowania postaci normalnych do poziomu piątej postaci normalnej. Ze względów historycznych proces normalizacji oraz znajdowania niepożądanych zależności jest kontynuowany do poziomu postaci 5NF, jednak istnieje możliwość definiowania bardziej rygorystycznych postaci normalnych, które uwzględniają dodatkowe rodzaje zależności i więzów. Istota **postaci normalnej klucza dziedziny** (ang. *domain-key normal form*, *DKNF*) polega na określeniu (przynajmniej teoretycznie) *ostatecznej postaci normalnej*, która uwzględni wszystkie możliwe rodzaje zależności i więzów. O schemacie relacji mówi się, że jest w postaci **DKNF**, jeżeli wszystkie więzy i zależności, które powinny być zachowane w przypadku poprawnych stanów relacji, mogą być wymuszane przez proste wymuszanie więzów domenowych oraz więzów kluczy na relacji. W przypadku relacji znajdującej się w postaci DKNF wymuszanie wszystkich więzów bazy danych staje się bardzo proste i polega na sprawdzaniu, czy wartość każdego atrybutu w krotce należy do odpowiedniej dziedziny oraz czy są wymuszane wszystkie więzy kluczy.

Jednakże, ze względu na trudności związane z uwzględnianiem złożonych więzów w relacjach DKNF, w praktyce użyteczność takiego rozwiązania jest bardzo ograniczona, gdyż określenie ogólnych więzów integralności może okazać się dość trudne. Weźmy na przykład pod uwagę relację SAMOCHÓD(MARKA, SID#), gdzie SID# oznacza numer identyfikacyjny samochodu, oraz relację PRODUCENT(SID#, KRAJ), gdzie KRAJ oznacza kraj pochodzenia producenta. Ogólne ograniczenie może brzmieć następująco: „Jeżeli MARKA to Toyota lub Lexus, pierwszym znakiem SID# jest J, jeżeli krajem producenta jest Japonia; jeżeli MARKA to Honda lub Acura, drugim znakiem SID# jest J, jeżeli krajem producenta jest Japonia”. Nie istnieje żaden uproszczony sposób reprezentowania takich więzów bez konieczności zapisania procedury (lub ogólnych asercji) w celu ich przetestowania. Wspomniana wcześniej procedura OBLICZ\_CENĘ\_KOŃCOWĄ jest przykładem procedury potrzebnej do wymuszania odpowiednich więzów odwołań.

Dlatego choć postać DKNF jest atrakcyjna i wydaje się prosta, nie można jej bezpośrednio sprawdzić ani zaimplementować, gwarantując jednocześnie spójność lub brak nadmiaru w projekcie. Z tego powodu w praktyce postać tę stosuje się rzadko.

## 15.7. Podsumowanie

W tym rozdziale opisaliśmy kolejny zestaw tematów związanych z zależnościami i dekompozycją oraz algorytmy dotyczące tych zagadnień i procesu projektowania relacji w postaciach 3NF, BCNF i 4NF na podstawie zbioru zależności funkcyjnych i wielowartościowych. W podrozdziale 15.1 przedstawiliśmy reguły wnioskowania dla zależności funkcyjnych, domknięcie atrybutów, równoważność między zbiorami zależności funkcyjnych oraz algorytmy wyszukiwania domknięć atrybutów (algorytm 15.1) i pokrycia minimalnego dla zbioru zależności funkcyjnych (algorytm 15.2). Dalej omówiono dwie istotne właściwości dekompozycji: właściwość złączenia bezstratnego (nieaddytywnego) oraz właściwość zachowania zależności. Opisaliśmy też algorytm sprawdzania właściwości złączenia nieaddytywnego dla dekompozycji na  $n$  relacji (algorytm 15.3). Prostszy test, dla binarnych dekompozycji nieaddytywnych (właściwość złączenia nieaddytywnego), został przedstawiony wcześniej, w punkcie 14.5.1. Następnie omówiliśmy projektowanie relacyjne za pomocą syntezy na podstawie zbioru danych zależności funkcyjnych. *Algorytm syntezy relacyjnej* (algorytm 15.4) tworzy relacje w postaci 3NF na podstawie schematu relacji uniwersalnej i określonego przez projektanta bazy zestawu zależności funkcyjnych. *Algorytmy dekompozycji relacji* (takie jak algorytmy 15.5 i 15.6) tworzą relacje w postaci BCNF lub 4NF w wyniku kolejnych nieaddytywnych dekompozycji nieznormalizowanych relacji na dwie relacje składowe. Pokazaliśmy, że możliwa jest synteza schematów relacji w postaci 3NF, które spełniają obie pożądane właściwości. Jednak dla postaci BCNF możliwe jest zapewnienie tylko nieaddytywności złączeń; zachowania zależności *nie da się* zagwarantować. Jeśli projektant musi wybrać jedną z dwóch właściwości, zapewnienie nieaddytywności złączeń jest koniecznością. W podrozdziale 15.4 pokazaliśmy, że z powodu wartości pustych mogą wystąpić pewne problemy w kolekcji relacji, nawet jeśli poszczególne relacje są w postaci 3NF lub BCNF. Czasem w wyniku nadmiernej dekompozycji mogą pojawić się zawieszane krotki, które nie występują w wynikach złączenia, dlatego mogą stać się niewidoczne. Pokazaliśmy również, że algorytmy takie jak 15.4 (algorytm syntezy do relacji w postaci 3NF) mogą dawać różne projekty w zależności od wybranego pokrycia minimalnego. W podrozdziale 15.5 ponownie omówiliśmy zależności wielowartościowe. Są one skutkiem nieprawidłowego połączenia dwóch lub więcej niezależnych atrybutów wielowartościowych w tej samej relacji. Zależności wielowartościowe skutkują eksplozją kombinatoryczną krotek używanych w 4NF. Na zakończenie, w podrozdziale 15.6, opisaliśmy zależności zawierania, służące do określania więzów odwołań i więzów klasy-podklasy. Zwróciliśmy też uwagę na potrzebę stosowania funkcji arytmetycznych lub bardziej złożonych procedur do wymuszania określonych zależności funkcyjnych. Rozdział zakończyliśmy krótkim opisem postaci normalnej klucza dziedziny.

### Pytania powtórkowe

- 15.1. Jaką rolę w rozwoju teorii projektów relacyjnych odgrywają reguły Armstronga — trzy reguły wnioskowania od RW1 do RW3?
- 15.2. Co należy rozumieć pod pojęciami kompletności i logiczności reguł wnioskowania Armstronga?
- 15.3. Co należy rozumieć pod pojęciem domknięcia zbioru zależności funkcyjnych? Podaj adekwatny przykład.

- 15.4. Kiedy dwa zbiory zależności funkcyjnych są równoważne? W jaki sposób można określić ich równoważność?
- 15.5. Czym jest minimalny zbiór zależności funkcyjnych? Czy każdy zbiór zależności posiada równoważny zbiór minimalny? Czy zawsze jest on unikatowy?
- 15.6. Co należy rozumieć pod pojęciem warunku zachowania atrybutów dla dekompozycji?
- 15.7. Dlaczego same postaci normalne są niewystarczającym warunkiem opracowania poprawnego projektu schematu?
- 15.8. Czym dla dekompozycji jest właściwość zachowania zależności? Dlaczego jest istotna?
- 15.9. Dlaczego *nie* można zagwarantować, że schematy relacji w postaci BCNF będą dawać zachowujące zależności dekompozycje schematów relacji? Podaj kontrprzykład dowodzący tej tezy.
- 15.10. Czym jest właściwośćłączenia bezstratnego (nieaddytywnego) dekompozycji? Dlaczego jest istotna?
- 15.11. Spośród właściwości zachowania zależności oraz bezstratności, którą z nich należy bezwzględnie spełnić? Dlaczego?
- 15.12. Omów problemy związane z wartościami pustymi i krotkami zawieszonymi.
- 15.13. Pokaż, w jaki sposób proces tworzenia relacji w pierwszej postaci normalnej może prowadzić do powstawania zależności wielowartościowych. Jak należy przeprowadzić pierwszą normalizację, aby tego uniknąć?
- 15.14. Jakie rodzaje więzów mają reprezentować zależności zawierania?
- 15.15. Czy zależności szablonowe różnią się od innych rodzajów omówionych zależności?
- 15.16. Dlaczego postać normalna klucza dziedziny jest znana jako ostateczna postać normalna?

## Ćwiczenia

- 15.17. Wykaż, że schematy relacji tworzone przez algorytm 15.4 znajdują się w trzeciej postaci normalnej.
- 15.18. Wykaż, że jeżeli macierz  $S$  będąca wynikiem działania algorytmu 15.3 nie posiada wiersza zawierającego same symbole  $a$ , to projekcja  $S$  na dekompozycję iłączenie jej z powrotem zawsze da w wyniku co najmniej jedną fałszywą krotkę.
- 15.19. Wykaż, że schematy relacji tworzone przez algorytm 15.5 znajdują się w postaci normalnej Boyce'a-Codda.
- 15.20. Napisz programy z implementacją algorytmów 15.4 i 15.5.
- 15.21. Rozważ relację  $LODÓWKA(MODEL\#, ROK, CENA, FABRYKA, KOLOR)$ , którą w skrócie można zapisać jako  $LODÓWKA(M, R, C, F, K)$ , oraz następujący zbiór  $F$  zależności funkcyjnych:  $F = \{M \rightarrow MF, \{M, R\} \rightarrow C, MF \rightarrow K\}$ .
  - a) Sprawdź poniższe elementy jako ewentualne klucze kandydujące relacji  $LODÓWKA$ , podając uzasadnienie tego, dlaczego mogą lub nie mogą być kluczami:  $\{M\}$ ,  $\{M, R\}$ ,  $\{M, K\}$ .

- b) W oparciu o powyższe stwierdzenia, określ, czy relacja *LODÓWKA* znajduje się w postaci 3NF, czy BCNF. Odpowiedź uzasadnij.
- c) Rozważ dekompozycję relacji *LODÓWKA* do postaci  $D = \{R_1(M, R, C), R_2(M, MF, K)\}$ . Czy dekompozycja ta jest bezstratna? Wykaż dlaczego (może okazać się konieczne odwołanie do testu związanego z właściwością LJ1 z podrozdziału 14.5.1).
- 15.22. Określ wszystkie zależności zawierania dla schematu relacji z rysunku 5.5.
- 15.23. Udowodnij, że zależność funkcyjna spełnia formalną definicję zależności wielowartościowej.
- 15.24. Rozważ przykład normalizacji relacji *DZIAŁKI* z podrozdziałów 14.4 i 14.5. Określ, czy dekompozycja na relacje  $\{DZIAŁKI1AX, DZIAŁKI1AY, DZIAŁKI1B, DZIAŁKI2\}$  posiada właściwość złączenia bezstratnego, wykorzystując algorytm 15.3 oraz używając testu związanego z właściwością LJ1 z punktu 14.5.1.
- 15.25. Pokaż, w jaki sposób zależności wielowartościowe  $NAZWPRAC \rightarrow NAZWAPROJ$  oraz  $NAZWPRAC \rightarrow IMIĘCZŁRODZ$  z rysunku 14.15(a) mogą pojawić się w czasie normalizacji do postaci 1NF, gdzie atrybuty *NAZWAPROJ* i *IMIĘCZŁRODZ* są wielowartościowe.
- 15.26. Zastosuj algorytm 15.2(a) względem relacji z ćwiczenia 14.24 w celu określenia klucza relacji *R*. Utwórz minimalny zbiór zależności *G*, który będzie równoważny *F*, i zastosuj algorytm syntezy (algorytm 15.4) w celu dokonania dekompozycji relacji *R* na relacje w postaci 3NF.
- 15.27. Powtórz ćwiczenie 15.26 dla zależności funkcyjnych z ćwiczenia 14.25.
- 15.28. Zastosuj algorytm dekompozycji (algorytm 15.5) względem relacji *R* oraz zbioru zależności *F* z ćwiczenia 15.24. Powtórz działania dla zależności *G* z ćwiczenia 15.25.
- 15.29. Zastosuj algorytm 15.2(a) względem relacji z ćwiczeń 14.27 i 14.28 w celu określenia klucza relacji *R*. Zastosuj algorytm syntezy (algorytm 15.4) w celu dokonania dekompozycji relacji *R* na relacje w postaci 3NF oraz algorytm dekompozycji (algorytm 15.5) w celu dekompozycji relacji *R* na relacje w postaci BCNF.
- 15.30. Rozważ poniższą dekompozycję dla schematu relacji *R* z ćwiczenia 14.24. Określ, czy każda dekompozycja posiada (1) właściwość zachowania zależności i (2) właściwość złączenia bezstratnego w kontekście zbioru *F*. Określ również, w której postaci normalnej znajduje się każda relacja dekompozycji.
- a)  $D_1 = \{R_1, R_2, R_3, R_4, R_5\}; R_1 = \{A, B, C\}, R_2 = \{A, D, E\}, R_3 = \{B, F\}, R_4 = \{F, G, H\}, R_5 = \{D, I, J\}.$
- b)  $D_2 = \{R_1, R_2, R_3\}; R_1 = \{A, B, C, D, E\}, R_2 = \{B, F, G, H\}, R_3 = \{D, I, J\}.$
- c)  $D_3 = \{R_1, R_2, R_3, R_4, R_5\}; R_1 = \{A, B, C, D\}, R_2 = \{D, E\}, R_3 = \{B, F\}, R_4 = \{F, G, H\}, R_5 = \{D, I, J\}.$

## Ćwiczenia laboratoryjne

*Uwaga:* w tych ćwiczeniach używany jest system DBD (ang. *Data Base Designer*) opisany w podręczniku do ćwiczeń laboratoryjnych. Schemat relacji  $R$  i zbiór zależności funkcyjnych  $F$  należy zapisać w postaci list. Oto przykład:  $R$  i  $F$  z ćwiczenia 14.24 są zapisywane w następującej postaci:

$$R = [a, b, c, d, e, f, g, h, i, j]$$
$$F = [[a, b], [c]],$$
$$[[a], [d, e]],$$
$$[[b], [f]],$$
$$[[f], [g, h]],$$
$$[[d], [i, j]]]$$

Ponieważ system DBD jest zaimplementowany w języku Prolog, nazwy pisane wielkimi literami są zarezerwowane dla zmiennych. Dlatego do zapisu atrybutów używane są małe litery. Więcej informacji o systemie DBD znajdziesz w podręczniku do ćwiczeń laboratoryjnych.

15.31. Za pomocą systemu DBD sprawdź odpowiedzi, jakich udzieliłeś w następujących ćwiczeniach:

- a) 15.24.
- b) 15.26.
- c) 15.27.
- d) 15.28.
- e) 15.29.
- f) 15.31 (a) i (b).
- g) 15.32 (a) i (c).

## Wybrane publikacje

Pozycje Maiera (1983) oraz Atzeniego i De Antonellisa (1993) zawierają kompletne omówienie teorii zależności relacyjnych. Algorytm 15.4 jest oparty na algorytmie normalizacji przedstawionym w pracy Biskupa i in. (1979). Algorytm dekompozycji (algorytm 15.5) został opisany w pracy Bernsteina (1976). Tsou i Fischer (1982) podali algorytm o złożoności wielomianowej dla dekompozycji do postaci BCNF.

Teoria zachowania zależności oraz złączeń bezstratnych została opisana w pracy Ullmana (1988), gdzie zawarto między innymi dowody poprawności niektórych z prezentowanych tu algorytmów. Właściwość złączenia bezstratnego została poddana analizie w książce Aho i in. (1979). Algorytmy określające klucze relacji na podstawie zależności funkcyjnych podano w pracy Osborna (1977). Sprawdzanie występowania postaci BCNF omówiono w pracy Osborna (1979). Sprawdzanie występowania postaci 3NF opisano w pozycji Tsou i Fischera (1982). Algorytmy projektowania relacji w postaci BCNF podano w pracach Wanga (1990) oraz Hernandeza i Chana (1991).

Zależności wielowartościowe oraz czwartą postać normalną zdefiniowali Zaniolo (1976) i Nicolas (1978). Wiele zaawansowanych postaci normalnych zdefiniował Fagin: czwartą postać normalną w pozycji Fagin (1977), postać PJNF w pozycji Fagin (1979) oraz postać DKNF w pozycji Fagin (1981). Zbiór logicznych i kompletnych reguł dla zależności funkcyjnych i wielowartościowych określono w pozycji Beeriego i in. (1977). Zależności złączeniowe omówiono w pracy Rissanena (1977) oraz Aho i in. (1979). Reguły wnioskowania dla zależności złączeniowych zdefiniowano w pracy Sciorego (1982). Zależności zawierania wprowadzono w książce Casanovy i in. (1981) i poddano dalszej analizie w pozycji Cosmadakisa i in. (1990). Ich użycie w zakresie optymalizacji schematów relacyjnych omówiono w pozycji Casanovy i in. (1989). Zależności szablonowe (jest to ogólna postać zależności oparta na krotkach hipotetycznych i wnioskach szablonu) omówili Sadri i Ullman (1982). Inne zależności stanowią temat pozycji Nicolasa (1978), Furtado (1978) oraz Mendelzona i Maiera (1979). Publikacja Abiteboula i in. (1995) zawiera teoretyczne omówienie wielu pojęć przedstawionych w rozdziale 14. oraz niniejszym.





# VII

---

**Struktury plikowe, funkcje mieszające,  
indeksowanie i projekty fizyczne  
baz danych**



# Składowanie danych na dysku, podstawowe struktury plikowe, funkcje mieszające i nowoczesne struktury składowania

Bazy danych są fizycznie przechowywane jako pliki rekordów, które zazwyczaj składa się na magnetycznych twardych dyskach. W rozdziale niniejszym i następnym zajmiemy się organizacją baz danych na nośnikach oraz technikami wydajnego uzyskiwania do nich dostępu przy użyciu różnych algorytmów, z których część wymaga pomocniczych struktur danych noszących nazwę *indeksów*. Takie struktury są często nazywane **fizycznymi strukturami plikowymi baz danych** i znajdują się na poziomie fizycznym trójwarstwowej architektury opisanej w rozdziale 2. Rozpoczynamy od wprowadzenia w podrozdziale 16.1 pojęć hierarchii komputerowych mechanizmów składowania danych oraz sposobów ich użycia w systemach bazodanowych. Podrozdział 16.2 zostanie poświęcony opisowi magnetycznych urządzeń pamięci masowej oraz ich cech, jak również pamięci flash, dyskom SSD, dyskom optycznym i urządzeniom taśmowym służącym do archiwizowania danych. Przedstawimy też techniki zwiększania wydajności dostępu do danych na dyskach. Po omówieniu różnych technik składowania danych zajmiemy się metodami organizacji danych na dyskach. Podrozdział 16.3 zostanie poświęcony technice podwójnego buforowania, która jest używana w celu przyspieszenia pobierania wielu bloków danych z dysku. Opiszemy też zarządzanie buforami i strategię zastępowania danych w buforach. W podrozdziale 16.4 zostaną omówione różne sposoby formatowania i przechowywania rekordów pliku na dysku. Z kolei podrozdział 16.5 będzie zawierał omówienie różnych rodzajów operacji stosowanych zwykle względem rekordów pliku. Następnie zostaną przedstawione trzy podstawowe metody organizacji rekordów pliku na dysku: rekordy nieuporządkowane, omawiane w podrozdziale 16.6, rekordy uporządkowane — omawiane w podrozdziale 16.7 — oraz rekordy poddane mieszaniu — w podrozdziale 16.8.

W podrozdziale 16.9 zostanie zawarte krótkie omówienie plików rekordów mieszanych oraz innych podstawowych metod organizacji rekordów, takich jak B-drzewa. Jest to szczególnie istotne w przypadku przechowywania baz obiektowych, które są omówione w rozdziale 11. W podrozdziale 16.10 zostanie opisany mechanizm RAID (ang. *redundant arrays of inexpensive/independent disks*) — architektura systemów składowania danych używana często w dużych przedsiębiorstwach w celu zwiększenia niezawodności i wydajności. Wreszcie w podrozdziale 16.11 zostaną opisane nowsze rozwiązania z obszaru architektur składowania danych, ważne w kontekście przechowywania danych firm. Te

rozwiązania to: sieci SAN (ang. *storage area networks*), pamięć NAS (ang. *network-attached storage*), iSCSI (ang. *internet small computer system interface*) i inne sieciowe protokoły składowania, dzięki którym sieci SAN stają się tańsze, nie wymagają infrastruktury zgodnej ze standardem Fibre Channel i zyskują popularność w branży. Omówimy też warstwy składowania danych i pamięć obiektową. Podrozdział 16.12 to podsumowanie rozdziału. W rozdziale 17. zostaną omówione techniki tworzenia pomocniczych struktur danych, noszących nazwę indeksów, które przyspieszają wyszukiwanie i pobieranie rekordów. Do technik tych należy składowanie danych pomocniczych, noszących nazwę plików indeksów, oprócz samych plików rekordów.

Czytelnicy dobrze znający problematykę organizacji plików i indeksowania mogą jedynie przejrzeć lub w ogóle pominąć rozdziały 16. i 17. Prezentowany tu materiał (a zwłaszcza podrozdziały 16.1 – 16.8) jest konieczny dla zrozumienia treści rozdziałów 18. i 19., które poświęcono problematyce przetwarzania zapytań i ich optymalizacji, a także dostrajaniu baz danych w celu poprawy wydajności obsługi zapytań.

## 16.1. Wprowadzenie

Zbiór danych tworzących skomputeryzowaną bazę danych musi być przechowywany fizycznie na pewnym komputerowym **nośniku danych** (ang. *storage medium*). Oprogramowanie systemu SZBD może wówczas pobierać, aktualizować i przetwarzać takie dane według potrzeb. Komputerowe nośniki danych tworzą *hierarchię mechanizmów składowania* (ang. *storage hierarchy*), która zawiera trzy główne kategorie elementów:

- **Podstawowy mechanizm składowania** (ang. *primary storage*). Ta kategoria uwzględnia nośniki danych, na których może operować bezpośrednio *procesor główny* (ang. *central processing unit, CPU*), na przykład pamięć główną komputera i mniejsze, ale szybsze pamięci podręczne. Mechanizm podstawowy zwykle zapewnia szybki dostęp do danych, ale wiąże się z ograniczoną pojemnością. Choć w ostatnich latach pojemność pamięci głównej szybko rosła, pamięć ta nadal jest droższa i mniej pojemna niż to konieczne w typowych bazach używanych przez korporacje. Zawartość pamięci głównej zostaje utracona w wyniku przerwania zasilania lub awarii systemu.
- **Drugorzędny mechanizm składowania** (ang. *secondary storage*). Najczęściej wybieranym nośnikiem pamięci aktywnej baz korporacyjnych były dyski magnetyczne. Jednak do przechowywania umiarkowanych ilości trwałych danych coraz częściej stosuje się pamięci flash. Gdy taka pamięć jest używana jako zastępnik dysku, jest nazywana **SSD** (ang. *solid-state drive*).
- **Trzeciorzędny mechanizm składowania** (ang. *tertiary storage*). Dyski optyczne (CD-ROM, DVD i podobne) oraz taśmowe to wymienne nośniki stosowane obecnie w systemach jako pamięć offline na potrzeby archiwizowania baz danych. Ta kategoria jest nazywana trzeciorzędnym mechanizmem składowania. Urządzenia te zwykle charakteryzują się większą pojemnością, mniejszymi kosztami i oferują wolniejszy dostęp do danych niż ma to miejsce w przypadku mechanizmów podstawowych. Dane składowane w urządzeniach drugo- i trzeciorzędnych nie mogą być przetwarzane bezpośrednio przez procesor główny — najpierw muszą zostać skopiowane do pamięci głównej.

W pierwszej kolejności, w punkcie 16.1.1, przedstawimy ogólne omówienie różnych urządzeń pamięciowych używanych jako podstawowe, drugorzędne i trzeciorzędne mechanizmy składowania, a później, w punkcie 16.1.2, omówimy sposób obsługi baz danych w hierarchiach składowania.

### 16.1.1. Hierarchie pamięciowe i urządzenia składowania danych<sup>1</sup>

W nowoczesnym systemie komputerowym dane znajdują się i są transportowane przez hierarchię nośników danych. Pamięć o największej szybkości działania jest najdroższa, a przez to jest dostępna w najmniejszej ilości. Pamięcią o najmniejszej szybkości działania są używane w trybie offline napędy taśmowe, które zasadniczo oferują nieograniczoną przestrzeń składowania.

Na *poziomie podstawowych struktur składowania danych* (ang. *primary storage level*) hierarchia pamięci zawiera po swojej droższej stronie **pamięć podręczną** (ang. *cache memory*), którą jest statyczna pamięć RAM (ang. *Random Access Memory*, pamięć o dostępie swobodnym). Pamięć podręczna jest zwykle używana przez procesor główny (CPU) w celu zwiększenia szybkości wykonywania programów za pomocą technik takich jak wstępne pobieranie i potoki. Na kolejnym poziomie pamięci podstawowej znajduje się pamięć DRAM (ang. *Dynamic RAM*), która zapewnia główny obszar działania dla procesora głównego w celu przechowywania programów oraz danych i potocznie określana jest mianem **pamięci głównej** (ang. *main memory*). Zaletą pamięci DRAM jest jej niska cena, która wciąż spada. Wadą jest jej ulotność<sup>2</sup> oraz niższa szybkość działania w porównaniu ze statyczną pamięcią RAM.

Na *drugo- i trzeciorzędnym poziomie struktur składowania danych* (ang. *secondary storage level*) znajduje się hierarchia uwzględniająca dyski magnetyczne oraz **pamięci masowe** (ang. *mass storage*) w postaci napędów CD-ROM (ang. *Compact Disk-Read-Only Memory*) i DVD (ang. *Digital Video Disk* lub *Digital Versatile Disk*), a także napędów taśmowych, które znajdują się na najmniej kosztownym końcu hierarchii. **Pojemność składowania** (ang. *storage capacity*) jest mierzona w kilobajtach (kB, czyli 1000 bajtów), megabajtach (MB, czyli milion bajtów), gigabajtach (GB, czyli miliard bajtów), a nawet terabajtach (TB, 1000 gigabajtów). Obecnie stosowana zaczyna być nazwa *petabajt* (1000 terabajtów, czyli  $10^{15}$  bajtów) w kontekście bardzo dużych repozytoriów z obszaru fizyki, astronomii, nauk o ziemi i innych zastosowań naukowych.

Programy są umieszczane i uruchamiane w pamięci DRAM. Ogólnie rzecz biorąc, duże, nieulotne bazy danych znajdują się na drugorzędnych nośnikach danych (dyskach magnetycznych) i fragmenty bazy są wczytywane do buforów znajdujących się w pamięci głównej i odczytywane z nich w razie potrzeby. Obecnie, kiedy komputery osobiste i stacje robocze mają po kilkaset megabajtów pamięci RAM i DRAM, możliwe stało się ładowanie dużych fragmentów bazy danych do pamięci głównej. Coraz częściej spotyka się laptopy posiadające od 8 do 16 GB pamięci głównej, a serwery nierzadko mają 256 GB takiej

---

<sup>1</sup> Doceniamy wartościową pomoc Dana Forsytha w zakresie aktualnego stanu systemów składowania w firmach. Dziękujemy również Sathishowi Damle za jego sugestie.

<sup>2</sup> Pamięci ulotne zazwyczaj tracą swoją zawartość w momencie odcięcia zasilania, co nie ma miejsca w przypadku pamięci nieulotnych.

pamięci. W niektórych przypadkach całe bazy danych można przechowywać w pamięci głównej (a kopie bezpieczeństwa na dysku magnetycznym), co prowadzi do powstania **baz danych pamięci głównej** (ang. *main memory databases*). Są one szczególnie przydatne w przypadku aplikacji działających w czasie rzeczywistym, które wymagają bardzo krótkich czasów odpowiedzi. Przykładem może tu być aplikacja obsługująca przełączanie rozmów telefonicznych, która przechowuje bazy danych zawierające informacje o trasach i liniach w pamięci głównej.

**Pamięć flash.** Między pamięcią DRAM a dyskami magnetycznymi można wyróżnić jeszcze jeden rodzaj pamięci — **pamięć flash** (ang. *flash memory*). Staje się ona coraz bardziej popularna, w szczególności ze względu na swój nieulotny charakter. Pamięci flash to charakteryzujące się wysoką gęstością, wysokowydajne pamięci wykonane w technologii *EEPROM* (ang. *Electrically Erasable Programmable Read-Only Memory*, programowalna pamięć stała wymazywalna elektrycznie). Zaletą pamięci flash jest ich krótki czas dostępu, zaś wadą fakt, że za każdym razem musi być kasowany i zapisywany cały blok. Pamięć flash występuje w dwóch rodzajach: NAND i NOR. Zależą one od używanych układów logicznych. Urządzenia NAND są tańsze w przeliczeniu na gigabajt i stosuje się je jako nośniki w urządzeniach o pojemności od 8 do 64 GB; popularne karty kosztują kilka złotych za gigabajt. Pamięć flash jest używana w aparatach, odtwarzaczach MP3/MP4, telefonach komórkowych, urządzeniach PDA itd. Dyski flash podłączone przez USB stały się najbardziej mobilnym nośnikiem do transferu danych między komputerami osobistymi. Urządzenie z pamięcią masową flash jest w nich zintegrowane z interfejsem USB.

**Dyski optyczne.** Najpopularniejszą postacią optycznej pamięci wymiennej są dyski CD i DVD. Dyski CD mają pojemność 700 MB, natomiast pojemność dysków DVD wynosi od 4,5 do 15 GB. Dane na dyskach CD-ROM są przechowywane optycznie i odczytywane przy użyciu lasera. Dyski CD-ROM zawierają wcześniej nagrane dane, których nie można nadpisywać. Dyski *WORM* (ang. *Write-Once-Read-Many*, zapisz raz, odczytaj wielokrotnie; inne nazwy to CD-R, DVD-R lub DVD+R) stanowią rodzaj nośnika używanego do archiwizowania danych. Pozwalają one na jednokrotne zapisanie danych i dowolnie częste ich odczytywanie bez możliwości skasowania. Ich pojemność wynosi około pół gigabajta i są one znacznie trwalsze od dysków magnetycznych<sup>3</sup>. Bardziej pojemny format dysków DVD, Blu-ray DVD, umożliwia zapis 27 GB danych na jednowarstwowym i 54 GB na dwuwarstwowym dysku. **Optyczne pamięci masowe z urządzeniami automatycznej zmiany dysków** (ang. *optical juke box memories*) wykorzystują macierze płyt CD-ROM, które są ładowane do napędów na żądanie. Choć pojemność takich urządzeń sięga setek gigabajtów, czas dostępu do danych sięga setek milisekund, co jest wartością bardzo dużą w porównaniu z dyskami magnetycznymi. Tego rodzaju rozwiązania powoli znikają z rynku ze względu na gwałtowny spadek kosztów i wzrost pojemności oferowanych przez dyski magnetyczne. Większość współczesnych komputerów osobistych posiada napędy CD-ROM lub DVD. Zwykle są to napędy CD-R, które pozwalają nagrywać płyty CD-ROM i płyty CD z dźwiękiem, a także płyty DVD.

---

<sup>3</sup> Ich szybkość obrotowa jest niższa (ok. 400 obr/m), co oznacza wyższe opóźnienie i niską szybkość transferu (100 – 200 KB/sekundę) dla dysku o szybkości 1X. Dyski o szybkości  $nX$  (np. 16X, gdzie  $n = 16$ ) mają zapewniać szybkość transferu na poziomie  $n$  razy wyższym (obrotów na minutę razy  $n$ ). Szybkość transferu dla dysków DVD 1X wynosi ok. 1,385 MB/sekundę.

**Taśmy magnetyczne.** Wreszcie należy wymienić **taśmy magnetyczne**, które są używane do archiwizacji i tworzenia kopii zapasowych danych. **Pamięci masowe z urządzeniami automatycznej zmiany taśm** (ang. *tape jukeboxes*), zawierające zestaw taśm, które podlegają katalogowaniu i mogą być automatycznie ładowane do napędów, stają się popularne jako **trzeciorzędny mechanizm składowania danych** (ang. *tertiary storage*), umożliwiające przechowywanie terabajtów danych. Przykładowo, system *EOS* (ang. *Earth Observing System*, system obserwacji Ziemi) prowadzony przez NASA przechowuje w ten sposób archiwa swoich baz danych.

Coraz więcej dużych przedsiębiorstw zaczyna posiadać bazy danych o pojemności terabajtów. Pojęcia **bardzo dużej bazy danych** (ang. *very large database*) nie można już zdefiniować precyzyjnie, ponieważ pojemności dysków ciągle rosną, a ceny spadają. Być może wkrótce termin **bardzo duże bazy danych** będzie zarezerwowany dla baz zawierających setki terabajtów lub petabajty danych.

Oto podsumowanie — obecnie do składowania danych stosuje się hierarchię urządzeń i systemów. W zależności od planowanych zastosowań danych i wymogów aplikacji dane są przechowywane na jednym lub kilku poziomach hierarchii. W tabeli 16.1 przedstawiono aktualny stan urządzeń i systemów oraz zakres pojemności, średni czas dostępu, przepustowość (szybkość transferu) i ceny detaliczne. Koszty składowania spadają na wszystkich poziomach hierarchii.

TABELA 16.1. Rodzaje pamięci masowej i ich pojemność, czas dostępu, maksymalna przepustowość (szybkość transferu) i ceny detaliczne

Typ	Pojemność*	Czas dostępu	Maksymalna przepustowość	Ceny detaliczne (2014)**
Pamięć główna — RAM	4 GB – 1 TB	30 ns	35 GB/s	370 – 74 000 zł
Pamięć flash — SSD	64 GB – 1 TB	50 $\mu$ s	750 MB/s	185 – 2220 zł
Pamięć flash — pendrive	4 – 512 GB	100 $\mu$ s	50 MB/s	7,4 – 740 zł
Dysk magnetyczny	400 GB – 8 TB	10 ms	200 MB/s	259 – 1850 zł
Pamięć optyczna	50 – 100 GB	180 ms	72 MB/s	370 zł
Taśma magnetyczna	2,5 – 8,5 TB	10 – 80 s	40 – 250 MB/s	92 500 – 111 000 zł
Urządzenie automatycznej zmiany taśm	25 – 2 100 000 TB	10 – 80 s	250 MB/s – 1,2 PB/s	11 000 – ponad 3 000 000 zł

\* Pojemność określana na podstawie popularnych urządzeń dostępnych w sprzedaży w 2014 r.

\*\* Ceny z detalicznych sklepów internetowych.

## 16.1.2. Przechowywanie baz danych

Bazy danych zwykle przechowują duże ilości danych, które muszą istnieć przez długi okres czasu; takie dane często nazywa się **trwałymi**. Dostęp do danych i ich przetwarzanie odbywa się w sposób powtarzalny. Stoi to w sprzeczności z pojęciem danych **tymczasowych** (ang. *transient*), które istnieją tylko przez okres wykonywania programu. Większość baz danych jest przechowywana *trwale* na dyskach magnetycznych drugorzędного mechanizmu przechowywania z następujących powodów:



- Ogólnie rzecz biorąc, bazy danych są zbyt obszerne, aby w całości mieściły się w pamięci<sup>4</sup>.
- Warunki, w których może nastąpić utrata przechowywanych danych, występują znacznie rzadziej w przypadku drugorzędnych mechanizmów składowania niż mechanizmów podstawowych. Stąd dyski i inne urządzenia drugorzędne określamy mianem **pamięci nieulotnej** (ang. *nonvolatile storage*), natomiast pamięć główna to **pamięć ulotna** (ang. *volatile storage*).
- Koszt pamięci w przeliczeniu na jednostkę danych jest o rząd wielkości mniejszy w przypadku dysków niż mechanizmów podstawowych.

Niektóre nowe techniki, takie jak dyski SSD, będą być może stanowić rozsądną alternatywę dla dysków magnetycznych. Bazy danych w przyszłości będą być może przechowywane na innych poziomach hierarchii pamięci niż opisano to w podrozdziale 16.1.1. Mogą to być poziomy od najszybszej pamięci głównej po wolne systemy automatycznej zmiany taśm działające w trybie offline. Jednakże przewiduje się, że dyski magnetyczne pozostaną najczęściej wybieranym rozwiązaniem w przypadku dużych baz danych jeszcze w ciągu najbliższych lat. Stąd istotną rzeczą jest poznanie i zrozumienie właściwości dysków magnetycznych oraz sposobów, dzięki którym można na nich przechowywać pliki danych w celu projektowania wydajnych baz o akceptowalnej szybkości działania.

Taśmy magnetyczne są często używane jako pamięć służąca do archiwizacji baz danych, ponieważ związane z nimi koszty są jeszcze mniejsze niż w przypadku dysków. Taśmy lub wymienne dyski optyczne wymagają interwencji operatora (lub automatycznego urządzenia wczytującego), aby zostały załadowane i odczytane; dopiero potem możliwe jest przetwarzanie danych. Z kolei dyski są urządzeniami **podłączonymi** (ang. *online*), do których dostęp może się odbywać bezpośrednio w dowolnym momencie.

Techniki używane do przechowywania dużych ilości ustrukturyzowanych danych na dysku mają duże znaczenie dla projektantów baz, administratorów oraz implementatorów systemów SZBD. Projektanci baz danych oraz administratorzy muszą znać korzyści i wady określonych rozwiązań w czasie projektowania, implementowania i zarządzania bazą danych w ramach określonego systemu SZBD. Zazwyczaj system SZBD oferuje kilka opcji organizacji danych, a proces **fizycznego projektowania bazy danych** jest związany z wybraniem spośród dostępnych opcji określonej techniki organizacji danych, która będzie najlepiej odpowiadała wymaganiom danej aplikacji. Implementatorzy systemu SZBD muszą dokładnie poznać techniki organizacji danych, tak aby mogli implementować je w sposób wydajny, oferując administratorowi i użytkownikom wystarczające możliwości.

Typowe aplikacje bazodanowe wymagają w celach przetwarzania dostępu tylko do niewielkiego fragmentu bazy danych jednocześnie. Kiedy określona porcja danych jest potrzebna, musi zostać zlokalizowana na dysku, skopiowana do pamięci głównej, a następnie, po przetworzeniu, z powrotem zapisana na dysku. Dane przechowywane na dysku są zorganizowane w **pliki** (ang. *files*) **rekordów** (ang. *records*). Każdy rekord to zbiór wartości, które mogą być interpretowane jako fakty związane z encjami, ich atrybutami oraz

---

<sup>4</sup> Sprzeczne z tym twierdzeniem są ostatnie nowinki w obszarze systemów baz danych pamięci głównej. Przykładowe znane systemy komercyjne tego typu to HANA firmy SAP i TIMESTEN firmy Oracle.

związkami. Rekordy powinny być składowane na dysku w sposób umożliwiający ich wydajne lokalizowanie w razie potrzeby. Wybrane techniki poprawy wydajności dostępu do dysku omawiamy w punkcie 17.2.2.

Można wyróżnić kilka **podstawowych metod organizacji plików** (ang. *primary file organizations*), które określają sposób *fizycznego rozmieszczenia* rekordów na dysku, a stąd *możliwości uzyskiwania do nich dostępu*. *Plik stertowy* (ang. *heap file*), inaczej *plik nieuporządkowany* (ang. *unordered file*), umieszcza rekordy na dysku w dowolnej kolejności, dopisując nowe rekordy na końcu pliku. Z kolei *plik posortowany* (ang. *sorted file*) zachowuje uporządkowanie rekordów według wartości określonego pola (nazywanego *kluczem sortowania*). *Plik mieszający* (ang. *hash file*) wykorzystuje funkcję mieszającą stosowaną względem określonego pola (nazywanego *kluczem mieszania*) w celu określenia miejsca umieszczenia rekordu na dysku. Inne podstawowe organizacje plików, takie jak *B-drzewa*, wykorzystują struktury drzewiaste. Zagadnienia te zostaną omówione w podrozdziałach od 16.6 do 16.9. **Drugorzędne mechanizmy organizacji, czyli pomocnicze struktury dostępu**, pozwalają na wydajne uzyskiwanie dostępu do rekordów pliku w oparciu o *inne pola* niż te, które są używane przez mechanizmy podstawowe. Większość z nich występuje w postaci indeksów, które zostaną omówione w rozdziale 17.

## 16.2. Drugorzędne urządzenia pamięciowe

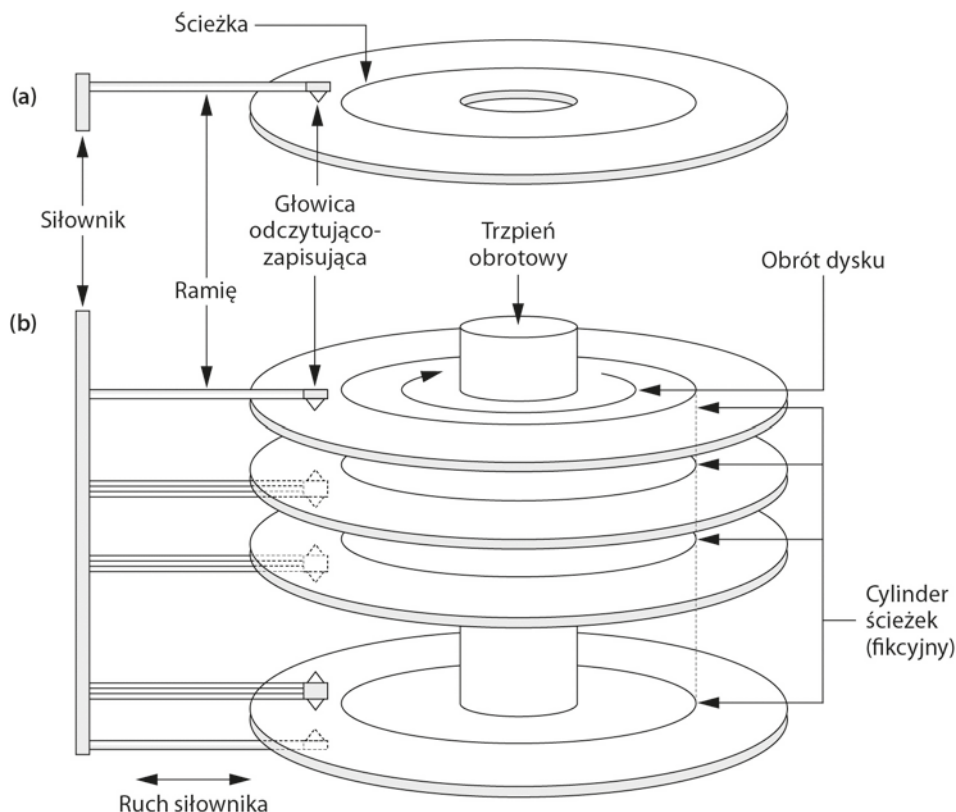
W niniejszym podrozdziale zostaną opisane pewne cechy dysków magnetycznych oraz urządzeń taśmowych. Czytelnicy znający te zagadnienia mogą jedynie przejrzeć poniższe podrozdziały.

### 16.2.1. Opis sprzętowy napędów dyskowych

Dyski magnetyczne są używane w celu przechowywania dużych ilości danych. Urządzenia przechowujące dyski są nazywane **dyskami twardymi** (ang. *hard disk drive* — HDD). Podstawową jednostką informacji na dysku jest pojedynczy **bit** danych. Magnetyzując obszar dysku w określony sposób można sprawić, że będzie on reprezentował wartość bitową zera lub jedynki. W celu zakodowania danych bity są grupowane w **bajty** (inaczej **znaki**). Rozmiary bajtów wynoszą zwykle od 4 do 8 bitów, w zależności od używanego komputera i urządzenia. Zwykle jest to 8 bitów. My zakładamy, że jeden znak jest przechowywany w jednym bajcie i będziemy używać pojęć *bajt* i *znak* zamiennie. **Pojemność** (ang. *capacity*) dysku to liczba bajtów, jakie może on pomieścić, i zwykle jest to bardzo wysoka wartość. Niewielkie dyski elastyczne były używane w komputerach osobistych i zwykle przechowywały od 400 kB do 1,5 MB danych. Obecnie prawie całkowicie wyszły już z użycia. Twarde dyski zwykle przechowują od kilkuset gigabajtów do kilku terabajtów danych, zaś duże pakiety dyskowe używane w serwerach i maszynach klasy mainframe mają pojemność od kilkuset gigabajtów. Pojemności dysków stale rosną, w miarę jak rozwijane są technologie ich produkcji.

Bez względu na pojemność wszystkie dyski są wykonane z materiału magnetycznego o kształcie cienkiego dysku (rysunek 16.1(a)) i są chronione plastikową lub akrylową obudową. O dysku mówimy, że jest **jednostronny** (ang. *single-sided*), jeżeli przechowuje

dane tylko na jednej ze swoich powierzchni, oraz **dwustronny** (ang. *double-sided*), jeżeli wykorzystywane są obie powierzchnie. W celu zwiększenia oferowanej pojemności dyski łączy się w **pakiety dysków** (ang. *disk pack*, rysunek 16.1(b)), które mogą zawierać wiele dysków, a stąd również wiele powierzchni. Dane są przechowywane na powierzchni dysku w formie koncentrycznych okręgów o *niewielkiej szerokości*<sup>5</sup>, z których każdy posiada określony promień. Każdy taki okrąg określa się mianem **ścieżki** (ang. *track*). W przypadku pakietów dysków ścieżki o jednakowym promieniu znajdujące się na różnych powierzchniach określa się mianem **cylindra** (ang. *cylinder*) ze względu na kształt, jaki utworzyłyby one po połączeniu w przestrzeni. Pojęcie cylindra jest istotne, ponieważ dane przechowywane na jednym cylindrze mogą być pobierane o wiele szybciej niż w przypadku ich rozrzucenia po różnych cylindrach.

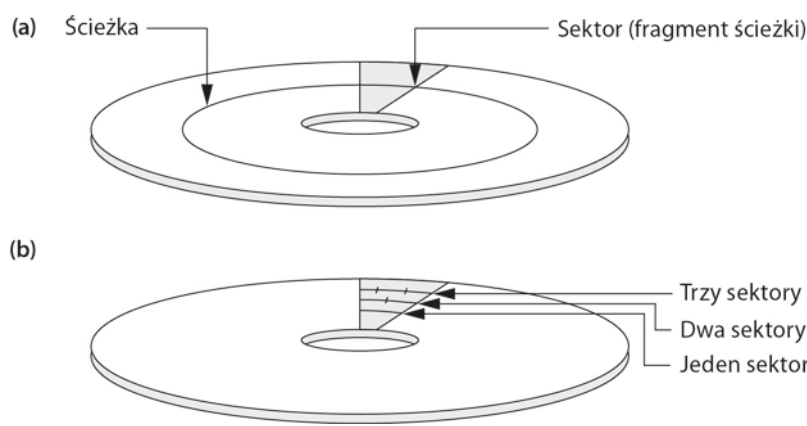


RYСУNEK 16.1. (a) Dysk jednostronny z mechanizmem odczytująco-zapisującym. (b) Pakiet dysków z mechanizmem odczytująco-zapisującym

Liczba ścieżek na dysku sięga od kilku tysięcy do 152 tysięcy (w dyskach opisanych w tabeli 16.2), a pojemność każdej ścieżki zwykle waha się w granicach od kilkudziesięciu kB do 150 kB. Ze względu na fakt, że ścieżka zwykle zawiera dużą ilość informacji, jest dzielona na mniejsze bloki, czyli sektory. Podział ścieżki na **sektory** (ang. *sectors*) jest naszytywno zapisany na powierzchni dysku i nie może być zmieniany. W przypadku jednego

<sup>5</sup> W niektórych nowszych dyskach okręgi te są łączone i tworzą rodzaj ciągłej spirali.

z typów organizacji sektorów mianem sektora określa się fragment ścieżki leżący na-przeciw stałego kąta względem środka dysku (rysunek 16.2(a)). Istnieje możliwość wy-korzystania kilku innych organizacji, z których jedna polega na zapewnieniu, aby w miarę oddalania się od środka dysku sektory zajmowały mniejszą powierzchnię kątową, co po-zwala na zachowanie stałej gęstości zapisu (rysunek 16.2(b)). Technika nosząca nazwę *ZBR* (ang. *Zone Bit Recording*, zapisywanie stref bitowych) pozwala zapewnić, aby prze-dział cylindrów posiadał jednakową liczbę sektorów na łuk. Przykładowo, cylindry od 0 do 99 mogą zawierać po jednym sektorze na ścieżkę, sektory od 100 do 199 — po dwa sektory na ścieżkę itd. Typowa wielkość sektora to 512 bajtów. Nie wszystkie dyski posia-dają ścieżki podzielone na sektory.



RYSUNEK 16.2. Różna organizacja sektorów na dysku. (a) Sektory leżące naprzeciwko stałego kąta. (b) Sektory zachowujące stałą gęstość zapisu

Podział ścieżek na posiadające taki sam rozmiar **bloki dyskowe** (ang. *disk blocks*), inaczej **strony** (ang. *pages*), jest określany przez system operacyjny w czasie operacji **forma-towania**, czyli **inicjalizacji** dysku. Rozmiar bloku zostaje określony w czasie inicjalizacji i nie może być dynamicznie zmieniany. Typowe rozmiary bloków dyskowych wynoszą od 512 do 8192 bajtów. Dysk o zapisanych na stałe sektorach często podlega ich podzia-łowi na bloki w trakcie inicjalizacji. Bloki są rozdzielone **szczelinami międzyblokowymi** (ang. *interblock gaps*) o stałym rozmiarze, które zawierają specjalnie zakodowane in-formacje sterujące zapisywane w czasie inicjalizowania dysku. Informacje te są używane w celu określenia, który blok na ścieżce występuje po każdej szczelinie. W tabeli 16.2 przedstawiono specyfikacje typowych dysków. Przedrostek 10K przed nazwami dysków oznacza szybkość obrotową w obr/m.

TABELA 16.2. (b) Cechy dysków wewnętrznych firmy Seagate (od 300 do 900 GB pojemności)

Specyfikacja	1200 GB
Numer modelu SED	ST1200MM0017
Numer modelu SED FIPS 140-2	ST1200MM0027
Nazwa modelu	Enterprise Performance 10K HDD v7
Interfejs	6Gb/s SAS

TABELA 16.2. (b) Cechy dysków wewnętrznych firmy Seagate (od 300 do 900 GB pojemności)  
— ciąg dalszy

Specyfikacja	1200 GB
<b>Możliwości</b>	
Przy formatowaniu 512 bajtów na sektor (w GB)	1200
Szybkość transferu zewnętrznego (w MB/s)	600
<b>Wydajność</b>	
Szybkość obrotowa (w obr/m)	10 000
Średnie opóźnienie (w ms)	2,9
Stały transfer od zewnętrznych do wewnętrznych sektorów (w MB/s)	Od 204 do 125
Pamięć podręczna wielosegmentowa (w MB)	64
<b>Konfiguracja i niezawodność</b>	
Dyski	4
Głowice	8
Nienaprawialne błędy odczytu na liczbę wczytanych bitów	1 na 10E16
Współczynnik AFR (ang. <i>Annualized Failure Rate</i> )	0,44%
<b>Cechy fizyczne</b>	
Maksymalna wysokość (cale/milimetry)	0,591/15,00
Maksymalna szerokość (cale/milimetry)	2,760/70,10
Maksymalna długość (cale/milimetry)	3,955/100,45
Waga (funty/kilogramy)	0,450/0,204

TABELA 16.2. (b) Cechy dysków wewnętrznych firmy Seagate (od 300 do 900 GB pojemności)

	ST900MM0006	ST600MM0006	ST450MM0006	ST300MM0006	
	ST900MM0026	ST600MM0026	ST450MM0026	ST300MM0026	
	ST900MM0046	ST600MM0046	ST450MM0046	ST300MM0046	
	ST900MM0036				
Pojemność dysku	900	600	450	300	GB (sformatowane, po zaokrągleniu)
Głowice odczytująco-zapisujące	6	4	3	2	
Bajtów na ścieżkę	997,9	997,9	997,9	997,9	KB (średnio, po zaokrągleniu)
Bajtów na powierzchnię	151 674	151 674	151 674	151 674	MB (niesformatowane, po zaokrągleniu)
Ścieżek na powierzchnię (w sumie)	152	152	152	152	Tysiące ścieżek (dostępnych dla użytkownika)
Ścieżek na cal	279	279	279	279	Tysiące ścieżek na cal (średnio)

TABELA 16.2. (b) Cechy dysków wewnętrznych firmy Seagate (od 300 do 900 GB pojemności)  
— ciąg dalszy

	ST900MM0006	ST600MM0006	ST450MM0006	ST300MM0006	
	ST900MM0026	ST600MM0026	ST450MM0026	ST300MM0026	
	ST900MM0046	ST600MM0046	ST450MM0046	ST300MM0046	
	ST900MM0036				
Maksymalna liczba bitów na cal	1925	1925	1925	1925	KB/cal
Gęstość powierzchniowa	538	538	538	538	GB/cal <sup>2</sup>
Szybkość obrotowa dysku	10 000	10 000	10 000	10 000	obr/m
Średnie opóźnienie obrotowe	2,9	2,9	2,9	2,9	ms

Za pozwoleniem firmy Seagate Technology.

Pojemności dysków stale się zwiększają, podobnie jak szybkości transmisji danych. Jednocześnie spadają ich ceny i obecnie cena megabajta pojemności dyskowej wynosi kilka groszy. Ceny spadają tak gwałtownie, że już wynoszą ok. 300 zł za 1 TB.

Dysk jest urządzeniem adresowalnym o *dostępie swobodnym* (ang. *random access*). Transfer danych między pamięcią główną a dyskiem odbywa się w postaci jednostek bloków dyskowych. **Adres sprzętowy** (ang. *hardware address*) bloku — kombinacja numeru cylindra, ścieżki (numer powierzchni w cylindrze, na której znajduje się ścieżka) oraz bloku (w ramach ścieżki) zostaje przekazany do sterownika wejścia-wyjścia. W wielu nowoczesnych napędach dyskowych pojedyncza liczba określana mianem *LBA* (ang. *Logical Block Address*, logiczny adres bloku), stanowiąca wartość z przedziału od 0 do  $n$  (zakładając, że całkowita pojemność dysku wynosi  $n+1$  bloków), jest automatycznie odwzorowywana na odpowiedni blok przez kontroler napędu dyskowego. Podawany jest również adres **bufora** — ciągłego zarezerwowanego obszaru w pamięci głównej, przechowującego jeden blok. W przypadku polecenia **odczytu** blok z dysku jest kopiowany do bufora, natomiast w przypadku polecenia **zapisu** zawartość bufora jest kopiowana do bloku na dysku. Niekiedy jako pojedyncza jednostka mogą być przesyłane ciągłe bloki, określane mianem **klastrów** (ang. *cluster*). W takim przypadku rozmiar bufora jest dostosowywany do liczby bajtów w klastrze.

Rzeczywistym mechanizmem odczytu i zapisu bloków jest **głowica odczytująco-zapisująca** (ang. *read/write head*), stanowiąca część systemu noszącego nazwę **napędu dysku** (ang. *disk drive*). Dysk lub pakiet dysków jest montowany w napędzie dysku, który zawiera również silnik wprawiający dyski w ruch obrotowy. Głowica odczytująco-zapisująca zawiera komponent elektroniczny dołączony do **ramienia mechanicznego** (ang. *mechanical arm*). Pakiety dysków o wielu powierzchniach są sterowane kilkoma głowicami — jedną na każdą powierzchnię (patrz rysunek 16.1(b)). Wszystkie ramiona są podłączone do **siłownika** (ang. *actuator*), który z kolei jest dołączony do silnika elektrycznego poruszającego jednocześnie wszystkie głowice i pozycjonującego je dokładnie nad cylindrem ścieżek określonego adresu bloku.



Napędy dyskowe obracają pakiety dysków ze stałą prędkością (zwykle wynosi ona od 5400 do 15 000 obr/m). Po umieszczeniu głowicy nad odpowiednią ścieżką, kiedy odpowiedni blok określony w adresie bloku znajdzie się pod głowicą, komponent elektroniczny zostaje aktywowany w celu przesłania danych. Niektóre jednostki dyskowe posiadają stałe głowice, których liczba jest równa liczbie ścieżek. Są to dyski z **głowicami stałymi**, natomiast jednostki dyskowe zawierające siłownik określa się mianem dysków z **głowicą ruchomą**. W przypadku dysków z głowicami stałymi ścieżka lub cylinder jest wybierana przez elektroniczne przełączenie do odpowiedniej głowicy, a nie przez mechaniczny ruch. W konsekwencji wiąże się to ze znacznym zwiększeniem szybkości działania. Jednak koszt dodatkowych głowic jest dość wysoki, co sprawia, że takie dyski nie są wykorzystywane zbyt często.

**Komunikacja dysków z systemami komputerowymi. Kontroler dysku** (ang. *disk controller*), zazwyczaj wbudowany w napęd dyskowy, steruje napędem i sprzęga go z systemem komputerowym. Jeden ze standardowych interfejsów używanych w przypadku napędów dyskowych w komputerach osobistych i stacjach roboczych nosił nazwę **SCSI** (ang. *Small Computer Storage Interface*). Obecnie do podłączania dysków HDD oraz napędów CD i DVD do komputera używany jest zwykle interfejs SATA (ang. *serial AT attachment*). Jego poprzednikiem jest standard „PC/AT attachment”, z bezpośrednim podłączaniem urządzeń do 16-bitowej magistrali. Został on wprowadzony przez firmę IBM. AT oznacza tu „advanced technology” (czyli zaawansowana technologia), jednak zwrot ten nie występuje w rozwinięciu nazwy SATA, ponieważ jest zastrzeżoną nazwą handlową. Inny popularny obecnie interfejs to **SAS** (ang. *serial attached SCSI*). Interfejs SATA został wprowadzony w 2002 r. i umożliwia umieszczenie kontrolera w samym dysku. Na płycie głównej potrzebny jest tylko prosty układ. W latach 2002 – 2008 szybkość transferu w interfejsie SATA wzrosła z 1,5 GB/s do 6 GB/s. Obecnie interfejs SATA jest nazywany NL-SAS (ang. *nearline SAS*). Największe 3,5-calowe dyski SATA i SAS mają pojemność 8 TB, natomiast 2,5-calowe dyski SAS są mniejsze i mają pojemność do 1,2 TB. Szybkość dysków 3,5-calowych wynosi 7200 lub 10 000 obr/m, a dyski 2,5-calowe mają szybkość do 15 000 obr/m. Jeśli jako kryterium oceny stosunku ceny do wydajności przyjąć liczbę operacji wejścia-wyjścia na sekundę, dyski SAS wypadają lepiej od dysków SATA.

Kontroler przyjmuje wysokopoziomowe polecenia wejścia-wyjścia i podejmuje odpowiednie działania w celu poprawnego ustawienia ramienia oraz powoduje wykonanie faktycznej operacji odczytu lub zapisu. W celu przesłania bloku dysku, którego adres został podany, kontroler musi najpierw mechanicznie ustawić głowicę odczytująco-zapisującą nad odpowiednią ścieżką. Wymagany do tego czas określa się mianem **czasu wyszukiwania** (ang. *seek time*). Typowe wartości tego parametru wynoszą od 5 do 10 ms w przypadku komputerów osobistych i 3 do 8 ms w przypadku serwerów. Później występuje kolejne opóźnienie, określane mianem **opóźnienia rotacyjnego** (ang. *rotational delay*) lub po prostu **opóźnienia** (ang. *latency*), związane z oczekiwaniem na przesunięcie się początku odpowiedniego bloku pod głowicę. Zależy ono od prędkości obrotowej dysku. Przykładowo, dla wartości 15 000 obr/m czas jednego obrotu wynosi 4 ms, więc średnie opóźnienie rotacyjne wynosi 2 ms. Dla wartości 10 000 obr/m średnie opóźnienie rotacyjne wynosi 3 ms. Wreszcie potrzebny jest pewien dodatkowy czas w celu przesłania danych — jest to **czas transmisji bloku** (ang. *block transfer time*). Ostatecznie całkowity czas potrzebny na zlokalizowanie i przesłanie dowolnego bloku o podanym adresie jest sumą czasu wyszukiwania, opóźnienia rotacyjnego oraz czasu transmisji bloku.



Czas wyszukiwania i opóźnienie rotacyjne mają zwykle o wiele większe wartości niż czas transmisji bloku. W celu zwiększenia wydajności działania często przesyła się kilka kolejnych bloków znajdujących się na tej samej ścieżce cylindra. Niweluje to czas wyszukiwania i opóźnienie rotacyjne dla wszystkich bloków oprócz pierwszego i może dawać w efekcie znaczne oszczędności czasowe w przypadku przesyłania wielu kolejnych bloków. Zazwyczaj producent dysku określa **zbiorczą prędkość transmisji danych** (ang. *bulk transfer rate*) w celu obliczania czasu wymaganego do przesłania kolejnych bloków. Ten oraz inne parametry dysków omówiono w dodatku B.

Czas wymagany do zlokalizowania i przesłania bloku dysku jest rzędu milisekund i zazwyczaj wynosi od 9 do 60 ms. W przypadku bloków ciągłych zlokalizowanie pierwszego bloku zajmuje od 9 do 60 ms, jednak przesyłanie kolejnych bloków może zajmować tylko od 0,4 do 2 ms na każdy blok. Wiele technik wyszukiwania wykorzystuje pobieranie kolejnych bloków w czasie szukania danych na dysku. W każdym razie czas przesyłania danych liczony w milisekundach uważa się za dość duży w porównaniu z czasem wymaganym do przetworzenia danych w pamięci głównej przez procesor. Stąd też lokalizowanie danych na dysku stanowi *poważne wąskie gardło* w działaniu aplikacji bazodanowych. Struktury plików omawiane w rozdziale 17. stanowią próby *zminimalizowania liczby operacji na blokach* w celu lokalizowania i przesyłania wymaganych danych z dysku do pamięci głównej. Głównym celem organizacji danych na dysku zawsze jest umieszczanie powiązanych informacji w przyległych blokach.

## 16.2.2. Zwiększanie wydajności dostępu do danych na dysku

W tym punkcie wymienimy wybrane często stosowane techniki zwiększania wydajności dostępu do danych na dyskach twardych.

- (1) **Buforowanie danych.** Aby poradzić sobie z niezgodnością szybkości procesora i urządzeń elektromechanicznych (takich jak dyski HDD), które są z natury wolniejsze, przeprowadza się buforowanie danych w pamięci, tak by można było umieszczać w buforze nowe dane w trakcie przetwarzania wcześniejszych przez aplikację. W podrozdziale 16.3 omówimy strategię podwójnego buforowania oraz ogólne zagadnienia związane z zarządzaniem buforem i strategię zastępowania danych w buforze.
- (2) **Odpowiednia organizacja danych na dysku.** Jeśli chodzi o strukturę i uporządkowanie danych na dysku, korzystne jest umieszczanie powiązanych danych w przyległych blokach. Gdy relacja wymaga wielu cylindrów, należy wykorzystać przyległe cylindry. Pozwala to uniknąć zbędnego ruchu głowicy odczytująco-zapisującej i związanego z tym czasu wyszukiwania.
- (3) **Wczytywanie danych przed zgłoszeniem żądania.** Aby zminimalizować czas wyszukiwania, należy w momencie przenoszenia bloku do bufora wczytywać także pozostałe bloki ze ścieżki, które mogą być potrzebne (choć jeszcze nie były żądane). Sprawdza się to dobrze w aplikacjach, które mogą wymagać przyległych bloków. Przy odczycie swobodnym ta strategia jest nieprzydatna.
- (4) **Odpowiednie szeregowanie żądań wejścia-wyjścia.** Jeśli trzeba wczytać z dysku kilka bloków, można zminimalizować łączny czas dostępu, szeregując operacje w taki sposób, by ramię poruszało się tylko w jednym kierunku, i pobierać bloki

w trakcie ruchu ramienia. Popularnym rozwiązaniem jest „algorytm windy”, który odzwierciedla działanie windy i w odpowiedniej kolejności szereguje żądania z różnych pięter. W ten sposób ramię może bez istotnych zakłóceń obsługiwać żądania, poruszając się na zewnątrz i do wewnątrz.

- (5) **Wykorzystanie dysków dziennika do tymczasowego przechowywania operacji zapisu.** Jednemu z dysków można przypisać tylko jedno zadanie — zapis dzienników. Wszystkie bloki przeznaczone do zapisu można sekwencyjnie umieszczać na tym dysku, co eliminuje czas wyszukiwania. Ta operacja jest znacznie szybsza niż zapis pliku w losowych lokalizacjach, co wymaga wyszukiwania przy każdym zapisie. Dysk dziennika pozwala porządkować zapisy według cylindrów i ścieżek oraz minimalizować dzięki temu ruch ramienia w trakcie zapisu. Dyskiem dziennika może być obszar (ekstent) dysku. Przechowywanie pliku z danymi i pliku dziennika na tym samym dysku jest tańszym rozwiązaniem, które jednak obniża wydajność. Choć dysk dziennika może poprawić wydajność zapisu, dla większości aplikacji używających danych ta technika się nie sprawdza.
- (6) **Używanie dysków SSD lub pamięci flash do odzyskiwania danych.** W aplikacjach, gdzie aktualizacje są częste, awaria systemu może prowadzić do utraty aktualizacji w pamięci głównej. Środkiem zapobiegawczym może być tu przyspieszenie aktualizacji i zapisów na dysku. Jedną z metod polega na zapisie aktualizacji w trwałym buforze SSD, którym może być pamięć flash lub zasilana bateryjnie pamięć DRAM (oba te rodzaje pamięci są bardzo szybkie; patrz tabela 16.1). Kontroler dysku może wtedy aktualizować plik z danymi w okresach bezczynności, a także po zapewnieniu bufora. W trakcie odzyskiwania danych po awarii niezapisane bufory SSD trzeba zapisać w pliku z danymi na dysku HDD. Dalsze omówienie dzienników i odzyskiwania danych znajdziesz w rozdziale 22.

### 16.2.3. Pamięć masowa SSD

Tego rodzaju pamięć jest czasem nazywana pamięcią masową flash, ponieważ jest oparta na technologii flash (patrz punkt 16.1.1).

Nowym trendem jest używanie pamięci flash jako warstwy pośredniej między pamięcią główną a drugorzędną pamięcią obrotową w postaci dysków magnetycznych (HDD). Ponieważ pamięć SSD przypomina dyski (jeśli chodzi o możliwość składowania danych w pamięci drugorzędnej bez konieczności ciągłego zasilania), urządzenia tego typu są nazywane **dyskami SSD** (ang. *solid-state drives*). Najpierw omówimy dyski SSD na ogólnym poziomie, a następnie opiszemy ich zastosowania w środowisku korporacyjnym, gdzie czasem są nazywane **dyskami EFD** (ang. *enterprise flash drives*); nazwa ta została po raz pierwszy wprowadzona przez firmę EMC Corporation.

Głównym komponentem dysków SSD jest kontroler i zestaw powiązanych kart z pamięcią flash. Najczęściej stosowana jest pamięć flash NAND. Dyski SSD w postaciach zgodnych z 3,5- i 2,5-calowymi dyskami HDD można umieszczać w laptopach i serwerach w slotach przeznaczonych na dyski HDD. Na potrzeby ultrabooków, tabletów i podobnych urządzeń powstają standardy mSATA i M.2 dotyczące dysków w kształcie kart. Aby naśledzić za rozwojem dysków SSD, opracowano interfejsy takie jak **SATA express**. Ponieważ w takich dyskach nie ma ruchomych elementów, dyski te są odporniejsze, dzia-

łają cicho i są szybsze (jeśli chodzi o czas dostępu i szybkość transferu) od dysków HDD. W odróżnieniu od dysków HDD, gdzie powiązane dane z tej samej relacji należy umieszczać w przyległych blokach i najlepiej w kolejnych cylindrach, na dyskach SSD nie ma ograniczeń związanych z rozmieszczeniem danych, ponieważ można bezpośrednio przejść pod każdy adres. Zmniejsza to fragmentację danych i potrzebę zmiany ich organizacji. W trakcie zapisu na dysku HDD zwykle ten sam blok jest zastępowany nowymi danymi. Na dyskach SSD dane są zapisywane w różnych komórkach NAND, aby zapewnić **jednolity poziom zużycia**, co przedłuża czas życia takich dysków. Głównym problemem utrudniającym upowszechnienie się dysków SSD są odstraszające ceny (patrz tabela 16.1), które wynoszą ok. 2,6 – 3 zł za GB (w porównaniu do 0,6 – 0,7 zł za GB w dyskach HDD).

Obok dysków SSD opartych na pamięci flash dostępne są też takie dyski wykorzystujące pamięć DRAM. Są one droższe niż dyski z pamięcią flash, ale zapewniają krótszy czas dostępu na poziomie 10 μs (mikrosekund); czas dostępu do pamięci flash wynosi ok. 100 μs. Główną wadą jest konieczność stosowania wewnętrznej baterii lub adaptera do zasilacza.

Przykładem dysków SSD o zastosowaniach korporacyjnych jest seria UCS (Unified Computing System®) Invicta firmy CISCO. Firma umożliwiła instalowanie dysków SSD w centrach danych, aby ujednolicić obciążenie robocze dowolnego rodzaju (w tym związane z bazami danych i infrastrukturą VDI — ang. *virtual desktop infrastructure*) oraz udostępnić rozwiązania ekonomiczne, wydajne energetycznie i oszczędne ze względu na miejsce. CISCO utrzymuje, że w wielodostępnej architekturze obejmującej wiele sieci dyski SSD Invicta, dzięki wymienionym zaletom, oferują lepszy stosunek ceny do wydajności. Zdaniem CISCO zwykle potrzeba cztery razy więcej dysków HDD, aby uzyskać wydajność porównywalną z macierzami RAID wykorzystującymi dyski SSD<sup>6</sup>. Konfiguracja z dyskami SSD może mieć pojemność od 6 do 144 TB, obsługiwać do 1,2 mln operacji wejścia-wyjścia na sekundę i zapewniać przepustowość do 7,2 GB/s ze średnim opóźnieniem 200 μs<sup>7</sup>. Nowoczesne centra danych przechodzą gwałtowną transformację i muszą zapewniać odpowiedzi w czasie rzeczywistym w architekturze opartej na chmurze. W takim środowisku dyski SSD zapewne będą odgrywać kluczową rolę.

#### 16.2.4. Taśmowe urządzenia pamięciowe

Dyski są urządzeniami pamięci drugorzędnej o **dostępie swobodnym**, ponieważ można *swobodnie* uzyskiwać dostęp do dowolnych bloków po podaniu ich adresu. Taśmy magnetyczne są urządzeniami o **dostępie sekwencyjnym** (ang. *sequential access*). W celu uzyskania dostępu do  $n$ -tego bloku na taśmie, najpierw musimy przejrzeć wcześniejsze  $n-1$  bloków. Dane są przechowywane na szpulach taśmy magnetycznej o dużej pojemności, nieco podobnie, jak ma to miejsce w przypadku taśm z muzyką lub filmem. **Napęd taśmowy** (ang. *tape drive*) służy do odczytywania i zapisywania danych na **szpuli taśmy** (ang. *tape reel*). Zazwyczaj każda grupa bitów tworzących bajt jest przechowywana wzdłuż taśmy, a same bajty są przechowywane po kolei na taśmie.

---

<sup>6</sup> Materiały firmy CISCO (CISCO, 2014).

<sup>7</sup> Specyfikacja produktu CISCO UCS Invicta Scaling System.

Głowica odczytująco-zapisująca służy do odczytywania i zapisywania danych na taśmie. Rekorde danych na taśmie są również przechowywane w blokach, aczkolwiek bloki te mogą mieć znacznie większy rozmiar niż ma to miejsce w przypadku dysków i szczeliny międzyblokowe są równie duże. W przypadku typowych gęstości zapisu danych na taśmach jest to od 1600 do 6250 bajtów na cal, zaś typowy rozmiar szczeliny międzyblokowej<sup>8</sup> wynoszący 0,6 cala, co przekłada się na 960 do 3750 bajtów traconej przestrzeni. W celu zapewnienia lepszego wykorzystania przestrzeni często grupuje się wiele rekordów w ramach jednego bloku.

Główną cechą taśmy jest wymaganie, aby dostęp do bloków danych odbywał się w **porządku sekwencyjnym**. W celu uzyskania dostępu do bloku znajdującego się w środku szpuli taśmy jest ona montowana, a następnie przeglądana do momentu, aż potrzebny blok znajdzie się pod głowicą. Z tego względu dostęp do danych na taśmach może być powolny i nie są one używane do przechowywania danych bieżących z wyjątkiem specjalnych aplikacji. Jednakże taśmy spełniają bardzo ważną funkcję — **archiwizacji** baz danych. Jednym z powodów przeprowadzania archiwizacji jest chęć zachowania kopii plików dyskowych na wypadek utraty danych z powodu awarii dysku, co może się zdarzyć w momencie, gdy głowica styka się z powierzchnią dysku i nastąpi awaria mechaniczna. Dlatego też pliki dyskowe są okresowo kopiowane na taśmy. W przypadku wielu aplikacji działających w trybie online, takich jak systemy rezerwacji biletów lotniczych, w celu uniknięcia jakichkolwiek przestoju wykorzystuje się systemy lustrzane, przechowujące trzy zbiory identycznych dysków — dwa w trybie online i jeden jako kopię bezpieczeństwa. W tym przypadku urządzeniem archiwizującym stają się dyski pracujące w trybie offline. Wszystkie trzy dyski pracują i w każdej chwili mogą zająć miejsce jednego z dysków aktywnych, który ulegnie awarii. Taśmy mogą być również wykorzystywane do przechowywania bardzo dużych plików danych. Wreszcie pliki bazy danych, które są rzadko używane lub zawierają stare dane, ale ich zachowanie jest konieczne w celu prowadzenia rejestru zmian historycznych, mogą zostać **zarchiwizowane** na taśmie. Pierwotnie do składowania danych używane były szpule półcalowe z taśmami dziewięćciścowymi. Później popularne wśród użytkowników stacji roboczych i komputerów osobistych zdobywają popularność mniejsze, 8-milimetrowe taśmy magnetyczne (podobne do używanych w kamerach), które mogą przechowywać do 50 GB danych, jak również 4-milimetrowe kartridże danych o zapisie ukośnym (ang. *helical scan*) oraz zapisywalne płyty CD i DVD. Są one również wykorzystywane do przechowywania danych graficznych i bibliotek systemowych.

Tworzenie kopii bezpieczeństwa baz danych przedsiębiorstw w celu zapewnienia, że nie zostaną utracone informacje o żadnych transakcjach, stanowi jedno z ważniejszych zadań. Kiedyś popularne były biblioteki taśmowe obsługujące do kilkuset kartridży w urządzeniach *DLT* (ang. *Digital Linear Tapes*) oraz *SDLT* (ang. *Super Digital Linear Tapes*), których pojemność sięgała setek gigabajtów danych zapisywanych na ścieżkach liniowych. Takie biblioteki nie są już rozwijane. Założone przez firmy IBM, Hadoop i Seagate konsorcjum *LTO* (ang. *Linear Tape Open*) wprowadziło w 2012 r. najnowszy standard *LTO-6* dla taśm. Opisano w nim półcalowe taśmy magnetyczne, podobne do tych używanych wcześniej w napędach taśmowych, ale umieszczone w mniejszym, jednoszpułowym kartridżu. Nowa generacja bibliotek używa napędów *LTO-6* z kartridżami o pojemności 2,5 TB i z szybkością transferu 160 MB/s. Średni czas wyszukiwania wynosi ok. 80 s. Napęd

---

<sup>8</sup> W terminologii napędów taśmowych określanej mianem *szczeliny międzyrekordowej*.

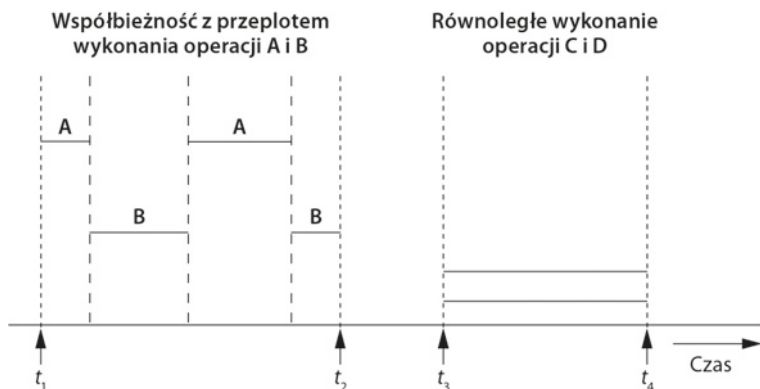
T10000D opracowany przez Oracle i StorageTek obsługuje kartridże o pojemności 8,5 TB i z szybkością transferu do 252 MB/s.

Używa się prostych ramion robotów w celu dokonywania zapisu na wielu kartridżach równolegle przy użyciu wielu napędów taśmowych z oprogramowaniem do automatycznego opatrywania etykietami kartridży z kopiami bezpieczeństwa. Przykładem gigantycznej biblioteki jest model SL8500 firmy Sun Storage Technology. Obsługuje on od 1450 do nieco ponad 10 000 slotów i od 1 do 64 napędów taśmowych w każdej bibliotece. Współdziała zarówno z taśmami DLT i SDLT, jak również z taśmami LTO. Można połączyć do 10 urządzeń SL8500 w jeden zespół biblioteczny obejmujący ponad 100 000 slotów i do 640 napędów. Przy 100 000 slotów SL8500 potrafi przechowywać 2,1 eksabajta (eksabajt = 1000 petabajtów = 1 000 000 terabajtów =  $10^{18}$  bajtów). Omówienie architektury dyskowej o nazwie RAID oraz sieci SAN, sieci NAS i systemów iSCSI Czytelnik znajdzie pod koniec niniejszego rozdziału.

## 16.3. Buforowanie bloków

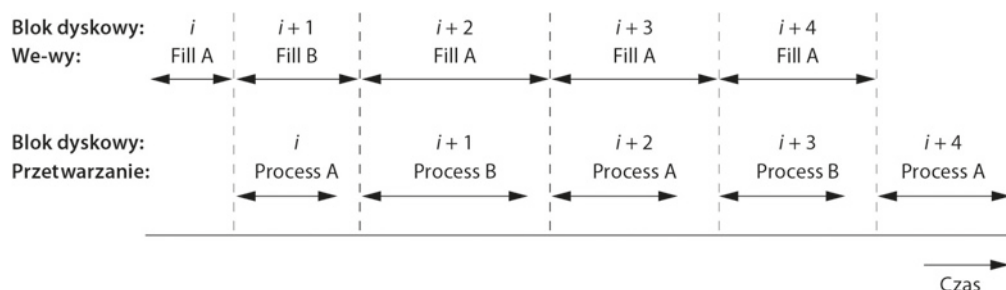
Kiedy z dysku do pamięci głównej trzeba przesłać kilka bloków i znane są adresy każdego z nich, w pamięci głównej można zarezerwować kilka buforów w celu przyspieszenia tej operacji. Kiedy jeden bufor jest odczytywany lub zapisywany, procesor może przetwarzać dane z innych buforów. Jest to możliwe dzięki istnieniu odrębnego procesora (kontrolera) dyskowych operacji wejścia-wyjścia, który po uruchomieniu może przysyłać bloki danych między pamięcią a dyskiem niezależnie od procesora głównego i równoległe z nim.

Na rysunku 16.3 przedstawiono sposób, w jaki dwa procesy mogą działać równoległe. Procesy A i B działają **współbieżnie** (ang. *concurrently*) w trybie **z przeplotem** (ang. *interleaved*), natomiast procesy C i D działają **współbieżnie** w trybie **równoległym** (ang. *parallel*). Kiedy jeden procesor steruje wieloma procesami, działanie równoległe nie jest możliwe. Jednak procesy mogą wówczas działać współbieżnie w trybie z przeplotem. Buforowanie jest najbardziej przydatne w przypadku, gdy procesy mogą działać współbieżnie w trybie równoległym, czy to ze względu na występowanie odrębnego kontrolera dyskowych operacji wejścia-wyjścia, czy też wielu procesorów głównych.



RYСУNEK 16.3. Współbieżność z przeplotem a wykonywanie równoległe

Na rysunku 16.4 przedstawiono, w jaki sposób może równoległe następować odczytywanie i przetwarzanie danych w przypadku, gdy czas wymagany do przetworzenia bloku dyskowego w pamięci jest mniejszy od czasu wymaganego do odczytania kolejnego bloku i napełnienia bufora. Procesor główny może rozpocząć przetwarzanie bloku od razu po zakończeniu jego przesyłania do pamięci głównej. W tym samym czasie kontroler dyskowych operacji wejścia-wyjścia może odczytać i przesłać kolejny blok do innego bufora. Taka technika nosi nazwę **podwójnego buforowania** (ang. *double buffering*) i może być również wykorzystywana w celu zapisywania ciągłego strumienia bloków z pamięci na dysk. Podwójne buforowanie pozwala na ciągłe odczytywanie i zapisywanie danych w kolejnych blokach na dysku, co eliminuje spowolnienia związane z czasem wyszukiwania i opóźnieniem rotacyjnym w przypadku wszystkich bloków oprócz pierwszego. Ponadto dane są od razu gotowe do przetworzenia, co redukuje czas oczekiwania programów.



RYСУNEK 16.4. Użycie dwóch buforów, A i B, w celu dokonywania odczytów z dysku

### 16.3.1. Zarządzanie buforem

**Zarządzanie buforem i strategię zastępowania danych.** Większość plików z dużymi bazami danych, zawierających miliony stron, nie umożliwia jednoczesnego umieszczenia wszystkich danych w pamięci głównej. Wspomnieliśmy o podwójnym buforowaniu, które pozwala zwiększyć wydajność dzięki wykonywaniu operacji wejścia-wyjścia dla dysku i pamięci głównej w jednym buforze równoległe z przetwarzaniem danych z drugiego bufora. Zarządzanie buforami i decyzje o tym, w którym z nich umieścić nowo wczytaną stronę, to złożony proces. Określenie **bufor** oznacza tu część pamięci głównej dostępną na bloki lub strony danych z dysku<sup>9</sup>. **Menedżer bufora** to programowy komponent SZBD reagujący na żądania danych i decydujący o tym, którego bufora użyć i które strony w buforze zastąpić, aby zmieścić ostatnio żądane bloki. Menedżer bufora traktuje dostępną pamięć główną jako **pulę bufora** obejmującą kolekcję stron. Wielkość współużytkowanej puli jest zwykle parametrem SZBD kontrolowanym przez administratora bazy. W tym punkcie pokrótce opiszemy działanie menedżera bufora i kilka strategii zastępowania danych.

Są dwa rodzaje menedżerów bufora. Menedżery pierwszego typu kontrolują pamięć główną bezpośrednio, tak jak w większości relacyjnych SZBD. Menedżery drugiego rodzaju tworzą bufory w pamięci wirtualnej, co pozwala przekazać kontrolę systemowi operacyjnemu. System operacyjny kontroluje, które bufory znajdują się w danym mo-

<sup>9</sup> W tym kontekście stosujemy zamiennie nazwy „strona” i „blok”.



mencie w pamięci głównej, a które są zapisane na dysku zarządzanym przez ten system. Menedżery bufora drugiego rodzaju są popularne w systemach baz danych pamięci głównej i niektórych obiektowych SZBD. Menedżer bufora ma dwa nadrzędne cele: (1) zmaksymalizować prawdopodobieństwo znalezienia żądanej strony w pamięci głównej i (2) w sytuacji odczytu nowego bloku z dysku ustalić zastępowaną stronę, której usunięcie spowoduje najmniej szkód (która nie będzie wkrótce ponownie potrzebna).

Aby umożliwić sobie pracę, menedżer bufora przechowuje dwa rodzaje informacji na temat każdej strony z puli bufora:

- (1) **Licznik** (ang. *pin-count*). Liczba żądań strony lub liczba aktualnych jej użytkowników. Jeśli ta wartość spadnie do zera, strona jest uznawana za **odpiętą** (ang. *unpinned*). Początkowo licznik każdej strony jest ustawiany na zero. Zwiększenie wartości licznika jest nazywane **przypięciem** (ang. *pinning*). Zwykle nie powinno się przenosić przypiętego bloku na dysk.
- (2) **Bit aktualizacji** (ang. *dirty bit*), początkowo ustawiany dla wszystkich stron na zero, ale przyjmujący wartość 1 za każdym razem, gdy strona zostanie zaktualizowana przez aplikację.

Jeśli chodzi o zarządzanie pamięcią, menedżer bufora musi zapewnić, że buforów zmieszczą się w pamięci głównej. Gdy ilość żądanych danych przekracza dostępną pojemność buforów, menedżer musi wybrać, które buforów opróżnić (zgodnie ze stosowaną strategią zastępowania danych). Jeżeli menedżer bufora przydzieli miejsce w pamięci wirtualnej i wszystkie używane buforów przekraczają pojemność pamięci głównej, występuje częsty w systemie operacyjnym problem migotania, a strony są przenoszone do pliku wymiany na dysku i z powrotem, przy czym system nie wykonuje użytecznej pracy.

Gdy żądana jest konkretna strona, menedżer bufora wykonuje następujące operacje: sprawdza, czy dana strona znajduje się już w buforze z puli. Jeśli tak jest, zwiększa licznik i udostępnia stronę. Jeżeli strony nie ma w puli buforów, menedżer wykonuje następujące zadania:

- a. Wybiera stronę do zastąpienia (na podstawie strategii zastępowania) i zwiększa licznik.
- b. Jeśli bit aktualizacji zastępowanej strony jest ustawiony, menedżer zapisuje tę stronę na dysku, zastępując jej wcześniejszą kopię. Jeżeli bit aktualizacji nie jest ustawiony, strona nie została zmodyfikowana i menedżer nie musi ponownie zapisywać jej na dysku.
- c. Wczytuje żadaną stronę do zwolnionej pamięci.
- d. Przekazuje adres nowej strony w pamięci głównej do aplikacji żądającej tej strony.

Jeśli w puli buforów nie znajduje się żadna odczepiona strona, a żądana strona nie jest dostępna w puli, menedżer czasem musi poczekać na zwolnienie jakiejś strony. Transakcja żądająca strony przechodzi wtedy w stan oczekiwania lub może nawet zostać anulowana.



### 16.3.2. Strategie zastępowania danych w buforze

Poniżej opisane są popularne strategie zastępowania danych w buforze. Przypominają one techniki używane w innych miejscach, np. w systemie operacyjnym.

- (1) **Najdawniej używane** (ang. *least recently used* — LRU). Ta strategia polega na usuwaniu strony, która nie była używana (do odczytu lub zapisu) przez najdłuższy czas. Wymaga to, by menedżer bufora przechowywał tabelę z zarejestrowanym czasem dostępu do każdej strony z bufora. Choć oznacza to dodatkowe koszty, ta strategia działa dobrze, ponieważ jeśli dane z bufora nie były używane przez długi czas, prawdopodobieństwo ponownego dostępu do nich jest niskie.
- (2) **Strategia zegarowa**. Jest to cykliczna wersja strategii LRU. Wyobraź sobie, że bufony są uporządkowane w okrąg przypominający zegar. Każdy bufor ma flagę o wartości 0 lub 1. Bufory z wartością 0 są zagrożone wymianą i można zastąpić ich zawartość, zapisując ją z powrotem na dysk. Po wczytaniu bloku do bufora flaga jest ustawiana na 1. Także dostęp do bufora powoduje ustawienie flagi na tę wartość. Wskazówka zegara jest ustawiona na „aktualny bufor”. Gdy menedżer potrzebuje bufora dla nowego bloku, przesuwa wskazówkę do momentu znalezienia bufora z flagą 0, po czym używa tego bufora do zapisu nowego bloku. Jeśli dla wcześniejszej strony ustawiony jest bit aktualizacji, ta strona zostanie zapisana, co spowoduje zastąpienie dawnej wersji strony pod jej adresem na dysku. Gdy wskazówka zegara dojdzie do bufora z flagą 1, następuje zmiana tej wartości na 0. Tak więc blok jest zastępowany w buforze tylko wtedy, jeśli nie był używany przez czas obrotu wskazówki, a ta wróciła do danego bloku z ustawioną ostatnim razem wartością 0.
- (3) **Pierwszy na wejściu, pierwszy na wyjściu** (ang. *first in, first out* — FIFO). W tej strategii gdy potrzebny jest bufor, zastępowane są dane z tego, w którym najdłużej znajduje się strona. Menedżer rejestruje tu czas wczytania każdej strony do bufora, nie musi jednak śledzić czasu dostępu do stron. Choć strategia FIFO wymaga mniej operacji administracyjnych niż strategia LRU, może działać w niepożądany sposób. W tym podejściu blok pozostający w buforze przez długi czas (ponieważ jest stale potrzebny), np. blok główny indeksu, może zostać usunięty, po czym natychmiast znów trzeba będzie go wczytać.

Strategie LRU i zegarowa nie sprawdzają się dobrze w aplikacjach bazodanowych, które wymagają sekwencyjnego przeglądania danych i używają plików niemieszczących się w całości w buforze. Ponadto w niektórych sytuacjach pewne strony z bufora nie mogą zostać usunięte i zapisane na dysku, ponieważ prowadzą do nich inne, przypięte strony. Strategie takie jak FIFO można zmodyfikować, aby się upewnić, że przypięte bloki, np. blok główny indeksu, pozostaną w buforze. Istnieją też wersje strategii zegarowej, w których ważnym buforom można przypisać wartości większe niż 1, dzięki czemu zawartość tych buforów nie zostanie zastąpiona przez kilka obrotów wskazówki. Zdarzają się też sytuacje, w których SZBD może zapisywać niektóre bloki na dysku nawet wtedy, gdy pamięć zajmowana przez te bloki nie jest potrzebna. Ta technika to **zapis wymuszony**, a stosuje się ją zwykle wtedy, gdy rekordy dziennika trzeba zapisać na dysku przed stronami zmodyfikowanymi przez transakcję (ma to na celu odzyskiwanie danych; patrz rozdział 22.). Istnieją też inne strategie zastępowania danych, np. **ostatnio używane dane** (ang. *most recently used* — MRU), które działają dobrze dla niektórych typów transakcji w bazach danych — przykładowo wtedy, gdy ostatnio używany blok nie jest potrzebny do czasu przetworzenia wszystkich dalszych bloków relacji.

## 16.4. Rozmieszczanie rekordów plików na dysku

Dane w bazie są traktowane jak zbiór rekordów uporządkowanych w zbiorze plików. W niniejszym podrozdziale zdefiniujemy pojęcia rekordów, typów rekordów oraz plików. Następnie omówimy techniki rozmieszczania rekordów plików na dysku. Od tego miejsca rozdziału urządzenia trwałej pamięci drugorzędnej o dostępie swobodnym będziemy nazywali „dyskami”. Dyski mogą mieć różną postać; mogą to być np. obrotowe dyski magnetyczne lub dyski SSD z dostępem elektronicznym i bez mechanicznych opóźnień.

### 16.4.1. Rekordy i typy rekordów

Dane są zwykle przechowywane w postaci **rekordów** (ang. *records*). Każdy rekord składa się z kolekcji powiązanych **wartości** danych, czyli **elementów**, gdzie każda wartość jest tworzona z jednego lub większej liczby bajtów i odpowiada konkretnemu **polu** (ang. *field*) rekordu. Rekordy zwykle opisują encje oraz ich atrybuty. Przykładowo, rekord `PRACOWNIK` reprezentuje encję pracownika i każda wartość pola w rekordzie określa pewien atrybut danego pracownika, na przykład `IMIE`, `DATAURODZENIA`, `PENSJA` lub `PRZEŁOŻONY`. Zestaw nazw pól i odpowiadających im typów danych tworzy definicję **typu rekordu** (ang. *record type*) lub **formatu rekordu** (ang. *record format*). **Typ danych** związany z każdym polem określa rodzaj wartości, jakie może przyjmować dane pole.

Typ danych pola jest zwykle jednym ze standardowych typów danych używanych w programowaniu. Chodzi tu o typy numeryczne (liczby całkowite, duże liczby całkowite, liczby zmiennoprzecinkowe), ciągi znaków (o stałej i zmiennej długości), logiczne (posiadające wyłącznie wartości 0 i 1 lub `TRUE` i `FALSE`) oraz niekiedy specjalnie kodowane typy danych **daty** i **czasu**. Liczba bajtów wymaganych przez każdy typ danych jest stała w przypadku określonego systemu komputerowego. Liczba całkowita może zajmować 4 bajty, duża liczba całkowita — 8 bajtów, liczba rzeczywista — 4 bajty, wartość logiczna — 1 bajt, data — 10 bajtów (zakładając format `YYYY-MM-DD`), zaś ciąg znaków o stałej długości  $k$  —  $k$  bajtów. Ciągi znaków o zmiennej długości mogą wymagać tylu bajtów, ile jest znaków w każdej wartości pola. Przykładowo, wykorzystując notację języka C, typ rekordu `PRACOWNIK` można zdefiniować następująco:

```
struct pracownik{
    char imięnazw[30];
    char pesel[11];
    int pensja;
    int kodstanowiska;
    char dział[20];
};
```

W niektórych aplikacjach bazodanowych może zachodzić potrzeba przechowywania danych składających się z dużych obiektów nie posiadających spójnej struktury, które reprezentują obrazy, strumienie audio i wizualne lub swobodny tekst. Określa się je mianem obiektów **BLOB** (ang. *Binary Large Object*, duży obiekt binarny). Element danych typu BLOB jest zwykle przechowywany oddzielnie od swojego rekordu w puli bloków dyskowych i w rekordzie umieszcza się tylko wskaźnik do niego. Do przechowywania swobodnego tekstu niektóre SZBD (np. Oracle, DB2) udostępniają typ danych CLOB (ang. *character large object*). W części SZBD ten typ danych nosi nazwę `text`.

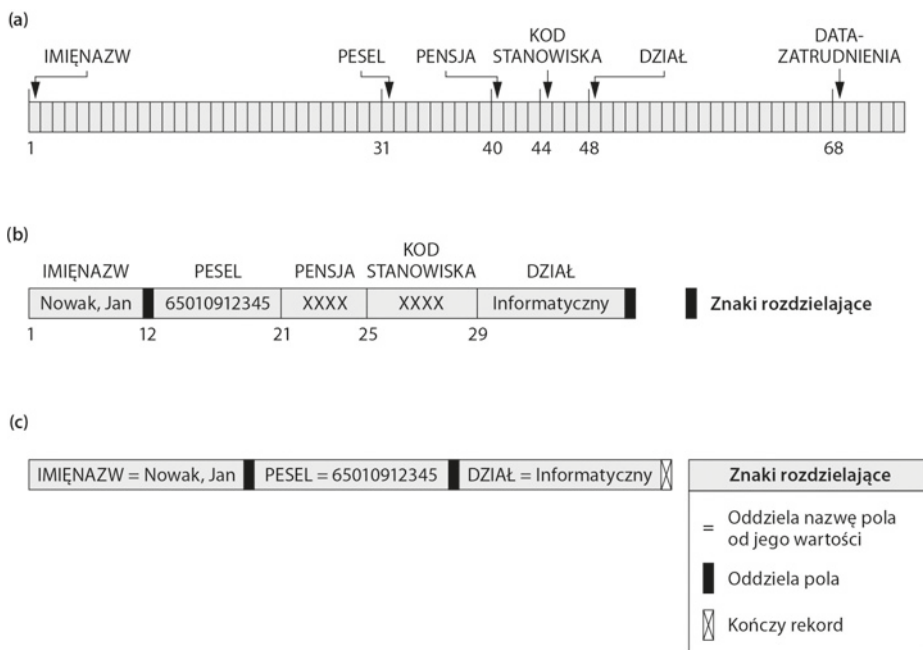
## 16.4.2. Pliki oraz rekordy o stałej i zmiennej długości

**Plik** jest *ciągim* rekordów. W wielu przypadkach wszystkie rekordy w pliku mają ten sam typ rekordu. Jeżeli każdy rekord w pliku ma dokładnie ten sam rozmiar (w bajtach), o pliku takim mówi się, że składa się z **rekordów o stałej długości** (ang. *fixed-length records*). Jeżeli różne rekordy w pliku mają różny rozmiar, o pliku mówi się, że składa się z **rekordów o zmiennej długości** (ang. *variable-length records*). Plik może zawierać rekordy o zmiennej długości z kilku powodów:

- Rekordy pliku mają ten sam typ rekordu, ale jedno lub więcej pól ma zmienną długość (**pola o zmiennej długości**). Przykładowo, pole IMIĘNAZW rekordu PRACOWNIK może być polem o zmiennej długości.
- Rekordy pliku mają ten sam typ rekordu, ale jedno lub więcej pól może zawierać wiele wartości dla pojedynczego rekordu. Takie pole określa się mianem **pola powtarzalnego** (ang. *repeating field*) a grupa wartości takiego pola to **grupa powtarzalna** (ang. *repeating group*).
- Rekordy pliku mają ten sam typ rekordu, ale jedno lub więcej pól jest **opcjonalne**, to znaczy może zawierać wartości dla pewnych, ale nie wszystkich rekordów pliku (**pola opcjonalne**).
- Plik zawiera rekordy o różnych typach rekordów, a przez to o różnych rozmiarach (**plik mieszany**). Sytuacja taka występuje, kiedy powiązane rekordy różnych typów zostaną *sklastrowane* (umieszczone wspólnie) w blokach na dysku. Przykładowo, rekord RAPORT\_OCEN określonego studenta może zostać umieszczony za rekordem STUDENT.

Rekordy PRACOWNIK o stałej długości z rysunku 16.5(a) posiadają rozmiar 71 bajtów. Każdy rekord posiada te same pola i ich długości są stałe, więc system może identyfikować pozycję bajta początkowego każdego pola względem początku rekordu. Ułatwia to lokalizowanie wartości pól przez programy uzyskujące dostęp do takich plików. Należy zauważyć, że istnieje możliwość reprezentowania pliku, który logicznie powinien posiadać rekordy o zmiennej długości, jako pliku rekordów o stałej długości. Przykładowo, w przypadku pól opcjonalnych można wymagać, aby *każde pole* było uwzględnione w *każdym rekordzie pliku*, ale przechowywać specjalną wartość pustą w przypadku, gdy dla danego pola nie istnieje żadna wartość. W przypadku pola powtarzalnego można by przydzielać w każdym rekordzie tyle przestrzeni, ile potrzeba dla *maksymalnej liczby wartości*, jakie może przyjąć pole. W obu przypadkach marnuje się pewna ilość przestrzeni, kiedy określone rekordy nie posiadają wartości dla wszystkich fizycznych lokalizacji zarezerwowanych w rekordzie. Poniżej rozpatrzemy inne opcje formatowania rekordów pliku o zmiennej długości.

W przypadku *pól o zmiennej długości* każdy rekord posiada wartość dla każdego pola, jednak nie znamy dokładnej długości niektórych wartości pól. W celu określenia bajtów w pewnym rekordzie, który reprezentuje każde pole, możemy użyć specjalnych znaków **separatorów** (takich jak ?, % lub \$) — które nie występują w wartości żadnego pola — w celu określania końca pól o zmiennej długości (rysunek 16.5(b)) lub możemy przechowywać długość pola w bajtach w rekordzie, poprzedzając nią wartość danego pola.



RYSUNEK 16.5. Formaty przechowywania rekordów. (a) Rekord o stałej długości z sześcioma polami o rozmiarze 71 bajtów. (b) Rekord z dwoma polami o zmiennej długości i trzema polami o stałej długości. (c) Rekord o zmiennej długości z trzema typami znaków rozdzielających

Plik rekordów z *polami opcjonalnymi* można formatować w inny sposób. Jeżeli całkowita liczba pól dla typu rekordu jest duża, ale liczba pól faktycznie występujących w typowym rekordzie jest mała, możemy w każdym rekordzie uwzględnić sekwencję par *<nazwa-pola, wartość-pola>* zamiast samych wartości pól. Na rysunku 16.5(c) użyto trzech typów znaków rozdzielających, choć moglibyśmy użyć tego samego znaku separatora w pierwszych dwóch celach — oddzielania nazwy pola od jego wartości oraz oddzielania jednego pola od drugiego. Bardziej praktycznym rozwiązaniem jest przypisanie krótkiego kodu **typu pola** — na przykład liczby całkowitej — do każdego pola i zawarcie w każdym rekordzie sekwencji par *<typ-pola, wartość-pola>* zamiast par *<nazwa-pola, wartość-pola>*.

*Pole powtarzalne* wymaga jednego znaku separatora dla rozdzielania powtarzających się wartości pola i drugiego znaku separatora dla określenia końca pola. Wreszcie w przypadku pliku zawierającego *rekordy różnych typów*, każdy rekord zostaje poprzedzony wskaźnikiem **typu rekordu**. Co zrozumiałe, programy przetwarzające pliki o rekordach zmiennej długości — stanowiące zwykle część systemu plików, a stąd ukryte przed zwykłymi programistami — muszą być bardziej złożone niż w przypadku plików z rekordami o stałej długości, kiedy to pozycja początkowa i rozmiar każdego pola są znane i stałe<sup>10</sup>.

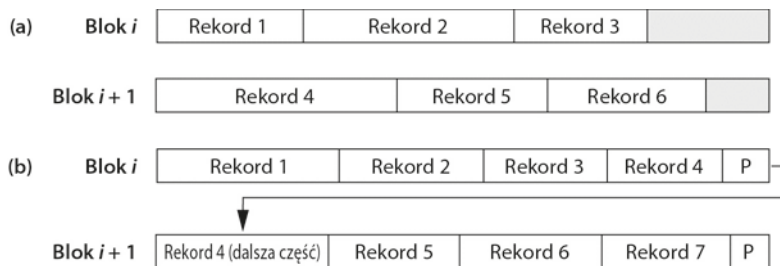
<sup>10</sup> Możliwe są również inne schematy reprezentowania rekordów o zmiennej długości.

### 16.4.3. Rozmieszczenie rekordów w blokach i rekordy segmentowane oraz niesegmentowane

Rekordy pliku muszą być przydzielane w blokach dyskowych, ponieważ blok jest *jednostką przesyłania danych* między dyskiem a pamięcią. Kiedy rozmiar bloku jest większy od rozmiaru rekordu, każdy blok będzie zawierał wiele rekordów, choć pewne pliki mogą zawierać niezwykle duże rekordy, niemieszczące się w jednym bloku. Załóżmy, że rozmiar bloku wynosi  $B$  bajtów. W przypadku pliku rekordów o stałej długości o rozmiarze  $R$ , gdzie  $B \geq R$ , możemy zmieścić  $bfr = \lfloor B/R \rfloor$  rekordów na blok, gdzie funkcja *podłoga* ( $\lfloor x \rfloor$ ) zaokrągla wartość  $x$  w dół do najbliższej wartości całkowitej. Wartość  $bfr$  jest określana mianem **współczynnika bloku** (ang. *blocking factor*) dla pliku. Generalnie,  $R$  może nie być podzielne przez  $B$ , w każdym bloku wystąpi wówczas pewien obszar nieużywanej przestrzeni o rozmiarze:

$$B - (bfr \cdot R) \text{ bajtów}$$

W celu wykorzystania takiej nieużywanej przestrzeni możemy przechowywać część rekordu w jednym bloku, a resztę w drugim. **Wskaźnik** znajdujący się na końcu pierwszego bloku wskazuje na blok zawierający resztę rekordu w przypadku, gdy nie będzie to następny w kolejności blok na dysku. Taką organizację określa się mianem **segmentowanej** (ang. *spanned*), ponieważ rekordy mogą być rozłożone na segmenty znajdujące się w więcej niż jednym bloku. Kiedy rozmiar rekordu przekracza rozmiar bloku, *musimy* użyć organizacji segmentowej. Jeżeli rekordy nie mogą przekraczać granic bloków, organizacja taka nosi nazwę **niesegmentowanej** (ang. *unspanned*). Jest ona wykorzystywana w przypadku rekordów o stałej długości, gdy  $B > R$ , ponieważ powoduje to, że każdy rekord zaczyna się od znanej lokalizacji w bloku, co upraszcza ich przetwarzanie. W przypadku rekordów o zmiennej długości można użyć organizacji segmentowej lub niesegmentowej. Jeżeli przeciętny rozmiar rekordu jest duży, korzystniejsze jest użycie segmentacji w celu zredukowania ilości traconej przestrzeni w każdym bloku. Na rysunku 16.6 przedstawiono oba rodzaje organizacji.



RYСУNEK 16.6. Rodzaje organizacji rekordu. (a) Niesegmentowana. (b) Segmentowana

W przypadku rekordów o zmiennej długości i organizacji segmentowanej każdy blok pliku może przechowywać różną liczbę rekordów. W takim przypadku współczynnik bloku  $bfr$  reprezentuje *średnią* liczbę rekordów na blok dla danego pliku. Możemy go wykorzystać w celu obliczenia liczby bloków  $b$  wymaganych dla pliku zawierającego  $r$  rekordów.

$$b = \lceil (r/bfr) \rceil \text{ bloków}$$

gdzie funkcja *sufit* ( $\lceil x \rceil$ ) zaokrągla wartość  $x$  w górę do najbliższej wartości całkowitej.

### 16.4.4. Alokowanie bloków pliku na dysku

Istnieje kilka standardowych technik alokowania bloków pliku na dysku. W przypadku **alokacji ciągłej** (ang. *contiguous allocation*) bloki pliku są alokowane na kolejnych blokach dysku. Sprawia to, że odczyt całego pliku jest bardzo szybki przy użyciu podwójnego buforowania, ale utrudnia rozszerzanie pliku. W przypadku **alokacji łączonej** (ang. *linked allocation*) każdy blok pliku zawiera wskaźnik na następny blok pliku. Ułatwia to rozszerzanie pliku, ale spowalnia odczyt całego pliku. Kombinacją obu tych rozwiązań jest alokacja **klastrów** kolejnych bloków dysku i połączenie tych klastrów. Klastry określa się niekiedy mianem **segmentów pliku** (ang. *file segments*) lub **ekstentów** (ang. *extents*). Kolejną możliwością jest użycie **alokacji indeksowej** (ang. *index allocation*), gdzie jeden lub więcej **bloków indeksu** zawiera wskaźniki na faktyczne bloki pliku. Często stosuje się również kombinacje wymienionych technik.

### 16.4.5. Nagłówki plików

**Nagłówek pliku** (ang. *file header*), inaczej **deskryptor pliku** (ang. *file descriptor*), zawiera informacje o pliku wymagane przez programy systemowe uzyskujące dostęp do rekordów pliku. Nagłówek zawiera informacje służące do określania adresów na dysku bloków pliku, jak również opisy formatów, które mogą uwzględniać długości pól oraz kolejność pól w rekordzie dla niesegmentowanych rekordów o stałej długości, a także kody typów pól, znaki separatorów oraz kody typów rekordów dla rekordów o zmiennej długości.

W celu wyszukania rekordu na dysku jeden lub większa liczba bloków zostaje skopiowana do buforów pamięci głównej. Następnie programy wyszukują potrzebny rekord lub rekordy w buforach, wykorzystując informacje z nagłówka pliku. Jeżeli adres bloku zawierającego potrzebny rekord nie jest znany, programy wyszukujące muszą przeprowadzić **wyszukiwanie liniowe** (ang. *linear search*) na blokach pliku. Każdy blok pliku zostaje skopiowany do bufora i jest przeszukiwany do momentu zlokalizowania rekordu lub zakończonego niepowodzeniem przeszukania wszystkich bloków pliku. W przypadku dużego pliku może to być bardzo czasochłonną operacją. Celem poprawnej organizacji plików jest możliwość lokalizowania bloku zawierającego potrzebny rekord z wykorzystaniem jak najmniejszej operacji przesyłania bloków.

## 16.5. Operacje wykonywane na plikach

Operacje wykonywane na plikach są zwykle grupowane w **operacje pobierania** (ang. *retrieval operations*) lub **operacje aktualizacji** (ang. *update operations*). Te pierwsze nie zmieniają żadnych danych w pliku, a tylko lokalizują określone rekordy, tak aby można było odczytać i przetworzyć odpowiednie wartości pól. Te drugie zmieniają plik poprzez wstawienie lub usunięcie rekordów albo zmodyfikowanie wartości pól. W obu przypadkach konieczne może być **wybranie** jednego lub większej liczby rekordów do pobrania, usunięcia lub zmodyfikowania na podstawie **warunku wyboru** (inaczej **warunku filtrowania**), określającego kryteria, które musi spełniać rekord lub rekordy.



Weźmy pod uwagę plik PRACOWNIK z polami IMIĘNAZW, PESEL, PENSJA, KODSTANOWISKA oraz DZIAŁ. **Prosty warunek wyboru** może zawierać porównanie równościowe na wartości pewnego pola — przykładowo, (PESEL = '65010912345') lub (DZIAŁ = 'Badawczy'). Bardziej skomplikowane warunki mogą uwzględniać inne operatory porównania, takie jak > lub ≥, na przykład, (PENSJA ≥ 3000). W ogólnym przypadku jako warunek wyboru występuje dowolne wyrażenie logiczne na polach pliku.

Operacje wyszukiwania przeprowadzane na pliku są, ogólnie rzecz biorąc, oparte na prostych warunkach wyboru. Złożony warunek musi zostać rozłożony przez system SZBD (lub programistę), tak aby można było otrzymać prosty warunek, możliwy do użycia w celu zlokalizowania rekordów na dysku. Każdy zlokalizowany rekord jest następnie sprawdzany w celu określenia, czy spełnia wszystkie warunki wyboru. Przykładowo, ze złożonego warunku ((PENSJA ≥ 3000) AND (DZIAŁ = 'Badawczy')) można wyodrębnić warunek (DZIAŁ = 'Badawczy'). Każdy rekord spełniający warunek (DZIAŁ = 'Badawczy') jest lokalizowany, a następnie wykonywane jest sprawdzenie w celu stwierdzenia, czy spełnia również warunek (PENSJA ≥ 3000).

Kiedy warunek wyszukiwania spełnia kilka rekordów pliku, najpierw jest lokalizowany *pierwszy* rekord (pod względem fizycznego rozłożenia rekordów pliku) i określany jako **rekord bieżący** (ang. *current record*). Kolejne operacje wyszukiwania zaczynają od tego rekordu i lokalizują *następny* rekord w pliku, który spełnia warunek.

Faktyczne operacje lokalizowania i uzyskiwania dostępu do rekordów pliku są różne w przypadku różnych systemów. Poniżej omówiono zbiór reprezentatywnych operacji. Zazwyczaj programy wysokopoziomowe, takie jak programy systemu SZBD, uzyskują dostęp do rekordów przy użyciu takich poleceń, tak więc w poniższych opisach niekiedy używamy pojęcia **zmiennych programu**.

- **Otwórz.** Przygotowuje plik do odczytu i zapisu. Zostają przydzielone odpowiednie bufor (zazwyczaj co najmniej dwa) w celu przechowywania bloków danych pobranych z dysku oraz zostaje pobrany nagłówek pliku. Wskaźnik pliku zostaje ustawiony na jego początek.
- **Zeruj.** Ustawia wskaźnik otwartego pliku na jego początek.
- **Znajdź (Zlokalizuj).** Znajduje pierwszy rekord, który spełnia warunki wyszukiwania. Blok zawierający taki rekord zostaje przesłany do bufora pamięci głównej (o ile jeszcze się tam nie znajduje). Wskaźnik pliku wskazuje na rekord w buforze i staje się on *rekordem bieżącym*. Czasami w celu określenia, czy zlokalizowany rekord ma zostać pobrany lub zaktualizowany, używa się różnych czasowników.
- **Odczytaj (Pobierz).** Kopiuje bieżący rekord z bufora do zmiennej programu w aplikacji użytkownika. Polecenie to może również powodować przesunięcie wskaźnika bieżącego rekordu na następny rekord w pliku, co może za sobą pociągać konieczność odczytania następnego bloku pliku z dysku.
- **ZnajdźNastępny.** Wyszukuje w pliku kolejny rekord spełniający warunki wyszukiwania. Blok zawierający taki rekord zostaje przesłany do bufora pamięci głównej (o ile jeszcze się tam nie znajduje). Rekord zostaje zlokalizowany w buforze i staje się rekordem bieżącym. W starszych SZBD, opartych na modelach hierarchicznych i sieciowych, dostępne są różne rodzaje polecenia ZnajdźNastępny



(np.: znajdź następny rekord w aktualnym rekordzie nadrzędnym, znajdź następny rekord danego typu lub znajdź następny rekord, dla którego spełniony jest złożony warunek wyszukiwania).

- **Usuń.** Usuwa rekord bieżący i (w konsekwencji) aktualizuje plik na dysku w celu uwzględnienia usunięcia.
- **Zmodyfikuj.** Modyfikuje niektóre wartości pól rekordu bieżącego i (w konsekwencji) aktualizuje plik na dysku w celu uwzględnienia modyfikacji.
- **Wstaw.** Wstawia nowy rekord do pliku poprzez zlokalizowanie bloku, do którego ma on zostać wstawiony, przesłanie bloku do bufora pamięci głównej (o ile już się tam nie znajduje), zapisanie rekordu do bufora i (w konsekwencji) zapisanie bufora na dysku w celu uwzględnienia wstawienia.
- **Zamknij.** Kończy korzystanie z pliku, zwalniając bufony i wykonując wszelkie inne wymagane operacje czyszczące.

Wszystkie powyższe operacje (oprócz *Otwórz* i *Zamknij*) określa się mianem operacji **jednorekordowych** (ang. *record-at-a-time*), ponieważ każda z nich ma zastosowanie względem pojedynczego rekordu. Istnieje możliwość połączenia operacji *Znajdź*, *Znajdź-Następny* i *Odczytaj* w jedną, *Przeglądaj*, której opis przedstawiono poniżej.

- *Przeglądaj.* Jeżeli plik został właśnie otwarty lub sprowadzony do stanu wyjściowego, operacja *Przeglądaj* zwraca pierwszy rekord. W przeciwnym razie zwraca następny rekord. Jeżeli wraz z operacją zostanie określony warunek, zwracany rekord jest pierwszym lub następnym rekordem spełniającym ten warunek.

W przypadku systemów bazodanowych względem plików można stosować dodatkowe wysokopoziomowe operacje **zbiorowe** (ang. *set-at-a-time*). Poniżej przedstawiono przykłady takich operacji:

- *ZnajdźWszystkie.* Lokalizuje **wszystkie** rekordy w pliku, które spełniają warunek wyszukiwania.
- *Znajdź (Zlokalizuj) n.* Wyszukuje pierwszy rekord spełniający warunek wyszukiwania i kontynuuje działania w celu zlokalizowania kolejnych  $n-1$  rekordów spełniających ten sam warunek. Bloki zawierające  $n$  rekordów zostają przesłane do bufora pamięci głównej (o ile jeszcze się tam nie znajdują).
- *ZnajdźPosortowane.* Pobiera wszystkie rekordy w pliku w pewnym określonym porządku.
- *Zreorganizuj.* Rozpoczyna proces reorganizacji. Jak się okaże, pewne metody organizacji plików wymagają okresowych reorganizacji. Przykładem może być operacja zmiany kolejności rekordów pliku poprzez posortowanie ich według wartości określonego pola.

W tym momencie warto podkreślić różnicę między pojęciem *metody organizacji pliku* a *metody dostępu*. **Metoda organizacji pliku** (ang. *file organization*) odnosi się do rozmieszczenia danych pliku w ramach rekordów, bloków i struktur dostępu. Chodzi tu także o sposób umieszczania i tworzenia powiązań między rekordami i blokami. Z kolei **metoda dostępu** (ang. *access method*) to grupa operacji — takich jak wymienione powyżej — które można stosować względem pliku. Ogólnie rzecz biorąc, istnieje możliwość zastosowania kilku metod dostępu względem jednej metody organizacji pliku. Jednakże niektóre

metody dostępu mogą być stosowane tylko w przypadku plików o określonej organizacji. Przykładowo, nie można stosować metody dostępu indeksowego względem pliku nieposiadającego indeksu (patrz rozdział 17.).

Zazwyczaj oczekujemy, że pewne warunki wyszukiwania będą wykorzystywane częściej od innych. Niektóre pliki mogą być **statyczne**, co oznacza, że operacje aktualizowania są na nich wykonywane bardzo rzadko. Inne pliki, o bardziej **dynamicznym** charakterze, mogą podlegać częstym zmianom, więc operacje aktualizowania będą względem nich stale stosowane. Jeśli plik nie może być aktualizowany przez użytkownika końcowego, jest uważany za plik tylko do odczytu. Większość hurtowni danych (patrz rozdział 29.) zawiera pliki tylko do odczytu. Poprawna organizacja pliku powinna zapewniać maksymalną wydajność operacji, co do których można oczekiwać, że będą *często* wykonywane na pliku. Rozważmy na przykład plik `PRACOWNIK` z rysunku 16.5(a), który przechowuje rekordy pracowników przedsiębiorstwa. Oczekujemy występowania operacji wstawiania (w momencie zatrudniania pracowników), usuwania (w momencie zwalniania pracowników) oraz modyfikowania rekordów (na przykład w razie zmiany pensji lub stanowiska pracy). Usuwanie i modyfikowanie rekordu wymaga określenia warunku wyboru w celu zidentyfikowania określonego rekordu lub zbioru rekordów. Pobranie jednego lub większej liczby rekordów również wymaga określenia warunku wyboru.

Jeżeli użytkownicy oczekują, że zwykle będą stosować warunek wyszukiwania bazujący na numerze PESEL, projektant musi wybrać metodę organizacji pliku, która ułatwi lokalizowanie rekordów o danej wartości pola PESEL. Może się to wiązać z fizycznym uporządkowaniem rekordów według tej wartości lub zdefiniowaniem indeksu na polu PESEL (patrz rozdział 17.). Załóżmy, że inna aplikacja wykorzystuje plik w celu generowania zestawień płac pracowników i wymaga, aby wartości były grupowane według działów. W takim przypadku najlepszym rozwiązaniem jest przechowywanie rekordów wszystkich pracowników danego działu obok siebie i porządkowanie według nazwiska w ramach każdego działu. Sposób połączenia rekordów w bloki i organizacja bloków w cylindrach wyglądają wtedy inaczej niż wcześniej. Jednak takie ustawienie pozostaje w sprzeczności z koniecznością porządkowania rekordów według wartości pola PESEL. Jeżeli obie aplikacje mają duże znaczenie, projektant powinien wybrać metodę organizacji umożliwiającą wydajne wykonywanie obu operacji. Niestety, w wielu przypadkach może nie istnieć organizacja pozwalająca na znalezienie takiego rozwiązania. Ponieważ plik może być przechowywany tylko za pomocą jednej konkretnej metody organizacji, administratorzy baz danych często muszą dokonać trudnego wyboru projektowego. Polegają przy tym na oczekiwanym znaczeniu operacji pobierania i aktualizowania danych oraz ich liczbie.

W kolejnych podrozdziałach oraz w rozdziale 17. Czytelnik znajdzie omówienie metod organizacji rekordów pliku na dysku. W celu tworzenia metod dostępu używa się kilku ogólnych technik, takich jak szeregowanie, mieszanie i indeksowanie. Ponadto w wielu metodach organizacji plików można stosować różne ogólne techniki obsługi operacji wstawiania i usuwania.

## 16.6. Pliki nieuporządkowanych rekordów (pliki stertowe)

W przypadku tej najprostszej i najbardziej podstawowej metody organizacji pliku rekordy są umieszczane w pliku w kolejności, w jakiej są wstawiane, tak więc nowe rekordy umieszcza się na jego końcu. Taką organizację określa się mianem **pliku stertowego** (ang. *heap file*) lub **pliku stosowego** (ang. *pile file*)<sup>11</sup>. Taka organizacja jest często używana wraz z dodatkowymi ścieżkami dostępu, takimi jak indeksy drugorzędne omówione w rozdziale 17. Używa się jej również w celu zbierania i przechowywania rekordów danych w celu przyszłego użycia.

Wstawianie nowego rekordu jest *bardzo wydajne*: ostatni blok dyskowy pliku zostaje skopiowany do bufora, zostaje do niego dodany nowy rekord i blok zostaje **ponownie zapisany** na dysku. Adres ostatniego bloku jest przechowywany w nagłówku pliku. Jednakże wyszukiwanie rekordu przy użyciu dowolnych warunków wiąże się z koniecznością **liniowego przeszukania** całego pliku blok po bloku, co jest kosztowne. Jeżeli warunki wyszukiwania spełnia tylko jeden rekord, wówczas, średnio rzecz biorąc, program musi wczytać do pamięci i przeszukać połowę bloków pliku w celu jego znalezienia. W przypadku pliku liczącego  $b$  bloków wymaga to przeszukania średnio  $(b/2)$  bloków. Jeżeli warunki wyszukiwania spełnia kilka lub żaden rekord, program musi odczytać i przeszukać wszystkie  $b$  bloków pliku.

W celu usunięcia rekordu program musi najpierw znaleźć jego blok, skopiować go do bufora, a następnie usunąć z bufora i w końcu **zapisać z powrotem** na dysku. Powoduje to pojawienie się nieużywanej przestrzeni w bloku dysku. Usunięcie w ten sposób dużej liczby rekordów powoduje marnowanie przestrzeni składowania danych. Inna technika usuwania rekordów polega na uwzględnieniu dodatkowego bajta lub bitu, nazywanego **znacznikiem usunięcia** (ang. *deletion marker*), przechowywanego wraz z rekordem. Rekord jest usuwany przez ustawienie znacznika usunięcia na określonej wartości. Inna wartość wskaźnika określa rekord poprawny (nieskasowany). Programy wyszukiujące uwzględniają jedynie rekordy poprawne w bloku w czasie swoich działań. Obie powyższe techniki usuwania wymagają przeprowadzania okresowych **reorganizacji** pliku w celu odzyskania nieużywanej przestrzeni po usuniętych rekordach. W czasie reorganizacji dostęp do bloków pliku odbywa się po kolei i rekordy zostają „upakowane” poprzez usunięcie skasowanych. Po przeprowadzeniu takiej reorganizacji bloki są ponownie zapełniane. Inną możliwością jest wykorzystanie przestrzeni po usuniętych rekordach w momencie wstawiania nowych rekordów, aczkolwiek wymaga to wprowadzenia dodatkowych mechanizmów śledzenia pustych lokalizacji.

W przypadku pliku nieuporządkowanego można wykorzystywać organizację segmentowaną lub niesegmentowaną i dotyczy to zarówno rekordów o stałej, jak i zmiennej długości. Modyfikacja rekordu o zmiennej długości może wymagać usunięcia starego rekordu i wstawienia zmodyfikowanego, ponieważ taki zmodyfikowany rekord może nie mieścić się w starym miejscu na dysku.

---

<sup>11</sup> Niekiedy używa się również określenia **plik sekwencyjny** (ang. *sequential file*).

W celu odczytania wszystkich rekordów według wartości pewnego pola tworzymy posortowaną kopię pliku. Sortowanie jest kosztowną operacją w przypadku dużego pliku i wykorzystuje się tu specjalne techniki **sortowania zewnętrznego** (patrz rozdział 18.).

W przypadku pliku nieuporządkowanych rekordów o stałej długości, wykorzystującego niesegmentowane bloki i alokację ciągłą, można w prosty sposób zaimplementować operację uzyskiwania dostępu do dowolnego rekordu na podstawie jego **pozycji** w pliku. Jeżeli rekordy pliku są ponumerowane od 0 do  $r-1$  i rekordy w każdym bloku są ponumerowane od 0 do  $bfr-1$ , gdzie  $bfr$  jest współczynnikiem bloku, wówczas  $i$ -ty rekord pliku znajduje się w bloku  $\lfloor i/bfr \rfloor$  i jest  $(i \bmod bfr)$  rekordem w tym bloku. Taki plik często określa się mianem **pliku względnego** (ang. *relative file*) lub **pliku bezpośredniego** (ang. *direct file*), ponieważ do rekordów można łatwo uzyskiwać dostęp bezpośrednio przy użyciu ich względnych pozycji. Uzyskiwanie dostępu do rekordu na podstawie jego pozycji nie pomaga w lokalizowaniu rekordu na podstawie warunku wyszukiwania, jednak ułatwia konstruowanie ścieżek dostępu do pliku, na przykład indeksów omówionych w rozdziale 17.

## 16.7. Pliki uporządkowanych rekordów (pliki posortowane)

Rekordy w pliku można fizycznie posortować na dysku w oparciu o wartości jednego z pól, określanego mianem **pola uporządkowania** (ang. *ordering field*). Daje to plik **uporządkowany** (ang. *ordered*), czyli **sekwencyjny** (ang. *sequential*)<sup>12</sup>. Jeżeli pole uporządkowania jest również **polem klucza**, czyli takim, które posiada unikatową wartość w każdym rekordzie, wówczas pole to określa się mianem **klucza uporządkowania** (ang. *ordering key*) pliku. Na rysunku 16.7 przedstawiono plik uporządkowany z polem IMIĘNAZW jako kluczem uporządkowania (przy założeniu, że pracownicy mają niepowtarzalne nazwiska i imiona).

Rekordy uporządkowane dają pewne korzyści w porównaniu z rekordami nieuporządkowanymi. Po pierwsze, odczyt rekordów w kolejności zgodnej z wartościami klucza uporządkowania staje się bardzo wydajny, ponieważ nie jest wymagane żadne sortowanie. Warunek wyszukiwania może mieć postać  $< \text{klucz} = \text{wartość} >$ ; można też zastosować warunek oparty na przedziale —  $< \text{wartość 1} < \text{klucz} < \text{wartość 2} >$ . Po drugie, znalezienie kolejnego rekordu dla bieżącego pod względem klucza uporządkowania nie wymaga uzyskiwania dostępu do innych bloków, ponieważ kolejny rekord znajduje się w tym samym bloku co bieżący (chyba że rekord bieżący jest ostatnim rekordem w bloku). Po trzecie, użycie warunku wyszukiwania w oparciu o wartość pola klucza uporządkowania daje w wyniku szybszy dostęp w razie zastosowania techniki wyszukiwania binarnego, która stanowi znaczny postęp w porównaniu z wyszukiwaniem liniowym, choć niezbyt często jest wykorzystywana w przypadku dysków. Pliki uporządkowane są porządkowane w blokach i przechowywane w przyległych cylindrach, aby zminimalizować czas wyszukiwania.

<sup>12</sup> Pojęcie *pliku sekwencyjnego* było również używane w odniesieniu do plików nieuporządkowanych.

**Wyszukiwanie binarne** (ang. *binary search*) w przypadku plików na dysku może być wykonywane na blokach, a nie rekordach. Załóżmy, że plik posiada  $b$  bloków o numerach od 1 do  $b$ . Rekordy są uporządkowane rosnąco według wartości klucza uporządkowania. Szukamy rekordu, którego wartość pola klucza uporządkowania wynosi  $K$ . Zakładając, że adresy dyskowe bloków pliku są dostępne w nagłówku pliku, wyszukiwanie binarne można wykonać za pomocą algorytmu 16.1. Wyszukiwanie binarne zazwyczaj wiąże się z uzyskaniem dostępu do  $\log_2(b)$  bloków, bez względu na to, czy rekord zostanie znaleziony, czy nie — stanowi to spore usprawnienie w porównaniu z wyszukiwaniem liniowym, gdzie średnio należy uzyskać dostęp do  $(b/2)$  bloków w przypadku, gdy rekord zostanie znaleziony, oraz do  $b$  bloków w przypadku, gdy nie zostanie znaleziony.

**Algorytm 16.1.** Wyszukiwanie binarne względem klucza uporządkowania pliku dyskowego

```

 $l \leftarrow 1; u \leftarrow b; (* b \text{ oznacza liczbę bloków pliku } *)$ 
while ( $u \geq l$ ) do
  begin  $i \leftarrow (l + u) \text{ div } 2;$ 
    wczytaj blok  $i$  pliku do bufora;
    if  $K < (\text{wartość pola klucza uporządkowania } \textit{pierwszego} \text{ rekordu w bloku } i)$ 
      then  $u \leftarrow i - 1$ 
    else if  $K > (\text{wartość pola klucza uporządkowania } \textit{ostatniego} \text{ rekordu w bloku } i)$ 
      then  $l \leftarrow i + 1;$ 
    else if rekord z wartością pola klucza uporządkowania =  $K$  znajduje się
      w buforze
      then goto znaleziono
    else goto niezaleziono;
  end;
goto niezaleziono;
```

Kryterium wyszukiwania uwzględniające warunki  $>$ ,  $<$ ,  $\geq$  oraz  $\leq$  względem pola uporządkowania jest dość wydajne, gdyż fizyczne uporządkowanie rekordów oznacza, że wszystkie rekordy spełniające warunek są umieszczone po kolei obok siebie w pliku. Przykładowo, odwołując się do rysunku 16.7, jeżeli kryterium wyszukiwania to  $(\text{IMIĘNAZW} < 'G')$  — gdzie  $<$  oznacza *leksykograficznie przed* — rekordy spełniające to kryterium to te znajdujące się od początku pliku aż do pierwszego rekordu, którego wartość pola IMIĘNAZW rozpoczyna się od litery  $G$ .

Uporządkowanie nie daje żadnych korzyści w przypadku losowego lub uporządkowanego dostępu do rekordów w oparciu o wartości innych pól, *niebędących polami uporządkowania*. W takich przypadkach dostęp swobodny wymaga zastosowania wyszukiwania liniowego. W celu uzyskania uporządkowanego dostępu do rekordów w oparciu o pole inne niż pole uporządkowania konieczne jest utworzenie dodatkowej posortowanej (w innym porządku) kopii pliku.

Wstawianie i usuwanie rekordów to kosztowne operacje w przypadku pliku uporządkowanego, gdyż rekordy muszą pozostać fizycznie uporządkowane. W celu wstawienia rekordu musimy znaleźć jego odpowiednią pozycję w pliku w oparciu o wartości pola uporządkowania, a następnie zapewnić wolne miejsce w pliku w celu wstawienia rekordu na tę pozycję. W przypadku dużego pliku może to być bardzo czasochłonne, ponieważ należy przesunąć średnio połowę rekordów pliku w celu zapewnienia wolnego miejsca dla nowego rekordu. Oznacza to, że połowa bloków pliku będzie musiała zostać odczytana i ponownie zapisana po przesunięciu rekordów. W przypadku usuwania problem jest mniej istotny, jeżeli używa się znaczników usunięcia oraz okresowych reorganizacji.

	IMIĘNAZW	PESEL	DATAUR	STANOWISKO	PENSJA	PŁEĆ
Blok 1	Abilski, Edward					
	Abrolska, Diana					
	⋮					
	Achański, Marek					
Blok 2	Adamski, Jan					
	Agatowska, Regina					
	⋮					
	Akulski, Jan					
Blok 3	Albański, Edward					
	Aliancki, Bronisław					
	⋮					
	Alwarski, Szymon					
Blok 4	Alzacki, Tomasz					
	Amur, Krzysztof					
	⋮					
	Andrucki, Robert					
Blok 5	Antylski, Zygmunt					
	Apolski, Jerzy					
	⋮					
	Arabska, Sylwia					
Blok 6	Armański, Mariusz					
	Armański, Stefan					
	⋮					
	Ateński, Tomasz					
⋮						
Blok n-1	Wiraszka, Jan					
	Wodnik, Damian					
	⋮					
	Wolska, Marianna					
Blok n	Wróblewska, Partycja					
	Wyspiański, Karol					
	⋮					
	Zwolański, Bronisław					

RYSUNEK 16.7. Niektóre bloki uporządkowanego (sekwencyjnego) pliku rekordów PRACOWNIK z polem IMIĘNAZW jako polem klucza uporządkowania



Jedną z możliwości zwiększenia wydajności operacji wstawiania jest zachowywanie pewnej niewykorzystanej przestrzeni w każdym bloku dla nowych rekordów. Jednak kiedy przestrzeń ta zostanie zużyta, znów pojawia się początkowy problem. Inną często stosowaną metodą jest utworzenie tymczasowego pliku *nieuporządkowanego* noszącego nazwę **pliku przepełnienia** (ang. *overflow file*) lub **pliku transakcji**. W przypadku tej techniki rzeczywisty plik uporządkowany jest określany mianem pliku **głównego** (ang. *master*). Nowe rekordy są wstawiane na koniec pliku przepełnienia, a nie na swojej odpowiedniej pozycji w pliku głównym. Okresowo plik przepełnienia jest sortowany i scalany z plikiem głównym w czasie reorganizacji pliku. Wstawianie staje się bardzo wydajne, jednak dzieje się to kosztem zwiększonej złożoności algorytmu wyszukiwania. Jedną z technik polega na przechowywaniu w każdym bloku w odrębnym polu najwyższej wartości klucza z uwzględnieniem kluczy z pliku przepełnienia. W przeciwnym razie plik przepełnienia musi być przeszukiwany przy użyciu techniki wyszukiwania liniowego, jeżeli rekord nie zostanie znaleziony w pliku głównym przy użyciu techniki wyszukiwania binarnego. W przypadku aplikacji niewymagających dostępu do zawsze aktualnych informacji rekordy przepełnienia mogą być w trakcie wyszukiwania ignorowane.

Modyfikacja wartości pola rekordu zależy od dwóch czynników: warunku wyszukiwania służącego do zlokalizowania rekordu oraz pola, które ma być zmodyfikowane. Jeżeli warunek wyszukiwania uwzględni pole klucza uporządkowania, możemy zlokalizować rekord wykorzystując wyszukiwanie binarne. W przeciwnym razie musimy przeprowadzić wyszukiwanie liniowe. Pole niebędące polem uporządkowania może zostać zmodyfikowane poprzez zmianę rekordu i ponowne zapisanie go w tym samym miejscu na dysku — przy założeniu wykorzystania rekordów o stałej długości. Modyfikacja pola uporządkowania oznacza, że rekord może zmienić swoją pozycję w pliku, co wymaga usunięcia starego rekordu oraz wstawienia zmodyfikowanego.

Odczytywanie rekordów pliku w kolejności zgodnej z wartościami pola uporządkowania jest dość wydajne, jeżeli zignorujemy rekordy przepełnienia, gdyż bloki mogą być odczytywane po kolei przy użyciu podwójnego buforowania. W celu uwzględnienia rekordów przepełnienia musimy wstawić je na odpowiednich dla nich pozycjach. W takim przypadku możemy najpierw dokonać reorganizacji pliku, a następnie sekwencyjnie odczytać jego bloki. W celu dokonania reorganizacji pliku najpierw sortujemy rekordy w pliku przepełnienia, a następnie scalamy je z plikiem głównym. Rekordy oznaczone do skasowania są w czasie tych działań usuwane.

W tabeli 16.3 zawarto podsumowanie średnich czasów dostępu do bloków w celu wyszukania określonego rekordu w pliku o  $b$  blokach.

TABELA 16.3. Średni czas dostępu dla pliku o  $b$  blokach w przypadku podstawowych metod organizacji pliku

Metoda organizacji	Metoda dostępu (wyszukiwania)	Średni czas dostępu do określonego rekordu
Sterna (nieuporządkowana)	Przeglądanie sekwencyjne (wyszukiwanie liniowe)	$b/2$
Uporządkowana	Przeglądanie sekwencyjne	$b/2$
Uporządkowana	Wyszukiwanie binarne	$\log_2 b$



Pliki uporządkowane są rzadko używane w aplikacjach bazodanowych, chyba że używa się dodatkowej ścieżki dostępu, noszącej nazwę **indeksu głównego** (ang. *primary index*). Daje to w wyniku **plik indeksowo-sekwencyjny** (ang. *indexed-sequential file*). Pozwala to w jeszcze większym stopniu skrócić czas dostępu w oparciu o pole klucza uporządkowania. Indeksy zostaną omówione w rozdziale 17. Jeśli atrybutem porządkującym nie jest klucz, powstały plik jest **plikiem klastrowanym**.

## 16.8. Techniki mieszania

Kolejny typ podstawowej organizacji pliku bazuje na technice mieszania, która zapewnia bardzo szybki dostęp do rekordów w oparciu o określone warunki wyszukiwania. Taką organizację zwykle określa się mianem **pliku mieszającego** (ang. *hash file*)<sup>13</sup>. Warunkiem wyszukiwania musi być warunek równości na pojedynczym polu, noszącym nazwę **pola mieszającego** (ang. *hash field*) pliku. W większości przypadków pole mieszające jest również polem klucza pliku i wówczas określa się je mianem **klucza mieszającego** (ang. *hash key*). Idea działania techniki mieszania polega na określeniu funkcji  $h$ , nazywanej **funkcją mieszającą** (ang. *hash function*) lub **funkcją randomizacji** (ang. *randomizing function*), która jest stosowana względem wartości pola mieszającego rekordu i daje w wyniku *adres* bloku dyskowego, w którym jest przechowywany rekord. Wyszukiwanie rekordu w bloku można wykonać w buforze pamięci głównej. W przypadku większości rekordów w celu pobrania rekordu wymagany jest dostęp tylko do jednego bloku.

Mieszanie jest również używane jako wewnętrzna struktura przeszukiwania w programie, kiedy dostęp do grupy rekordów odbywa się wyłącznie poprzez użycie wartości jednego pola. Użycie mieszania w przypadku plików wewnętrznych zostanie opisane w podrozdziale 16.8.1. Następnie, w podrozdziale 16.8.2, zostanie przedstawione, w jaki sposób można wprowadzić modyfikacje umożliwiające przechowywanie plików zewnętrznych na dysku. W podrozdziale 16.8.3 zostaną omówione techniki mieszania rozszerzalnego w odniesieniu do plików dynamicznie zwiększających swój rozmiar.

### 16.8.1. Mieszanie wewnętrzne

W przypadku plików wewnętrznych mieszanie jest zwykle implementowane w postaci **tabeli mieszającej** (ang. *hash table*) poprzez użycie tablicy rekordów. Załóżmy, że indeksy tablicy mają wartości od 0 do  $M-1$  (rysunek 16.8(a)). W takim przypadku posiadamy  $M$  **pozycji** (slotów), których adresy odpowiadają indeksom tablicy. Wybieramy funkcję mieszającą, która przekształca pole mieszające na wartość całkowitą z przedziału od 0 do  $M-1$ . Jedną ze znanych funkcji mieszających jest funkcja  $h(K) = K \bmod M$ , która zwraca resztę z dzielenia całkowitoliczbowej wartości pola mieszającego  $K$  przez  $M$ . Następnie wartość ta jest używana w celu określenia adresu rekordu.

---

<sup>13</sup> Plik mieszający określa się również jako *plik bezpośredni* (ang. *direct file*).

(a)

	IMIĘNAZW	PESEL	STANOWISKO	PENSJA
0				
1				
2				
3				
			⋮	
$M-2$				
$M-1$				

(b)

	Pola danych	Wskaźnik przepełnienia	
0		-1	Przeźnięć adresowa
1		$M$	
2		-1	
3		-1	
4		$M+2$	
	⋮		
$M-2$		$M+1$	Przeźnięć przepełnienia
$M-1$		-1	
$M$		$M+5$	
$M+1$		-1	
$M+2$		$M+4$	
	⋮		
$M+0-2$			
$M+0-1$			

- Wskaźnik null = -1
- Wskaźnik przepełnienia wskazuje na pozycję kolejnego rekordu na liście jednokierunkowej

RYSUNEK 16.8. Wewnętrzne mieszanie struktur danych. (a) Tablica  $M$  pozycji używana w ramach mieszania wewnętrznego. (b) Rozstrzygnięcie kolizji poprzez powiązanie rekordów

Wartości niecałkowitoliczbowych pól mieszających można zamienić na wartości całkowite przed zastosowaniem funkcji mod. W przypadku ciągów znaków numeryczne kody ASCII powiązane ze znakami mogą zostać użyte w przekształceniu — na przykład przez pomnożenie tych wartości kodowych. W przypadku pola mieszającego, którego typem danych jest ciąg 20 znaków, algorytm 16.2(a) może zostać użyty w celu obliczenia adresu skrótu. Zakładamy, że funkcja kodu zwraca kod numeryczny znaku oraz że posiadamy wartość  $K$  pola mieszającego typu  $K$ : `array[1..20] of char` (w języku Pascal) lub `char K[20]` (w języku C).

**Algorytm 16.2.** Dwa proste algorytmy mieszające. (a) Zastosowanie funkcji mieszającej mod względem ciągu znaków K. (b) Rozstrzyganie kolizji poprzez jawne adresowanie

```
(a) temp ← 1;
    for i ← 1 to 20 do temp ← temp * kod(K[i]) mod M;
    adres_mieszania ← temp mod M;
(b) i ← adres_mieszania(K); a ← i;
    if pozycja i jest zajęta
    then begin i ← (i + 1) mod M;
        while (i ≠ a) oraz pozycja i jest zajęta
        do i ← (i + 1) mod M;
        if (i = a) wówczas wszystkie pozycje są zajęte
        else nowy_adres_mieszania ← i;
    end;
```

Można również używać innych funkcji mieszających. Jedna z technik, określana mianem **przenoszenia** (ang. *folding*), polega na stosowaniu funkcji arytmetycznej, takiej jak *dodawanie*, lub funkcji logicznej, takiej jak *różnica symetryczna*, względem różnych fragmentów pola mieszającego w celu obliczenia adresu. Przykładowo, w pamięci adresowej z tysiącem komórek (od 0 do 999) sześciocyfrowy klucz 235469 może zostać przeniesiony pod adres  $(235+964) \bmod 1000 = 199$ . Kolejna technika polega na wybraniu pewnych cyfr z wartości pola mieszającego — na przykład trzeciej, piątej i ósmej — w celu utworzenia adresu mieszającego<sup>14</sup>. Gdy zapisywane są dane tysiąca pracowników z 11-cyfrowym numerem PESEL w pliku mieszającym o tysiącu pozycji, przedstawiona funkcja przypisze numerowi PESEL 71121105357 wartość 115. Problem związany z większością funkcji mieszających polega na tym, że nie gwarantują one, iż odrębne wartości będą odwzorowywane na odrębne adresy, ponieważ **przestrzeń pola mieszającego** (ang. *hash field space*), czyli liczba możliwych wartości, jakie pole to może przyjmować, jest zwykle znacznie większa od **przestrzeni adresowej** (ang. *address space*), czyli liczby adresów dostępnych dla rekordów. Funkcja mieszająca odwzorowuje przestrzeń pola mieszającego na przestrzeń adresową.

**Kolizja** (ang. *collision*) ma miejsce wówczas, gdy wartość pola mieszającego wstawianego rekordu po wykonaniu mieszania daje adres zawierający już inny rekord. W takim przypadku musimy wstawić nowy rekord na pewnej innej pozycji, gdyż adres mieszający jest zajęty. Proces znajdowania kolejnej pozycji określa się mianem **rozstrzygania kolizji** (ang. *collision resolution*). Istnieje wiele metod rozstrzygania kolizji, w tym opisane poniżej:

- **Adresowanie jawne** (ang. *open addressing*). Rozpoczynając od pozycji określonej przez adres mieszający, program sprawdza kolejne pozycje po kolei do momentu, aż znajdzie nieużywaną (pustą) pozycję. W tym celu można użyć algorytmu 16.2(b).
- **Łączenie** (ang. *chaining*). W przypadku tej metody przechowywane są informacje o różnych pozycjach przepełnienia, zwykle poprzez rozszerzenie tablicy o liczbę pozycji przepełnienia. Ponadto, do każdej pozycji rekordu zostaje dodane pole wskaźnika. Kolizja jest rozstrzygana przez umieszczenie nowego rekordu na nieużywanej pozycji przepełnienia i ustawienie wskaźnika adresu mieszającego zajmowanej pozycji na adres takiej pozycji przepełnienia. Zatem przechowujemy listę jednokierunkową rekordów przepełnienia dla każdego adresu mieszającego, co przedstawiono na rysunku 16.8(b).

<sup>14</sup> Szczegółowe omówienie funkcji mieszających wykracza poza zakres tematyczny niniejszej książki.

- **Mieszanie wielokrotne** (ang. *multiple hashing*). Program stosuje drugą funkcję mieszającą, jeżeli pierwszy wynik powoduje kolizję. W razie wystąpienia kolejnej kolizji, program wykorzystuje adresowanie jawne lub stosuje trzecią funkcję mieszającą, a następnie, w razie potrzeby, używa adresowania jawnego. Warto zauważyć, że sekwencja funkcji mieszających jest używana w tej samej kolejności na potrzeby pobierania danych.

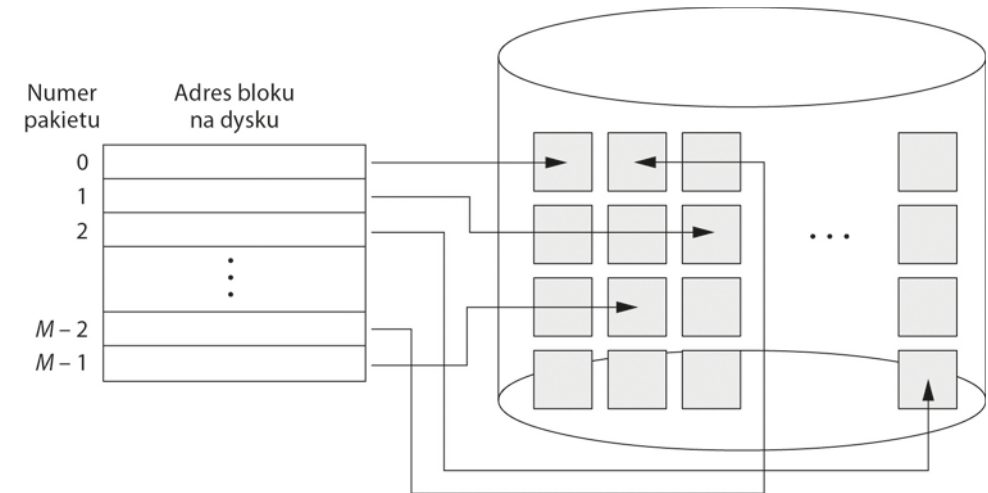
Każda metoda rozstrzygania kolizji wymaga własnego algorytmu wstawiania, pobierania i usuwania rekordów. Algorytmy łączenia są najprostsze. Algorytmy usuwania w przypadku adresowania jawnego są dość skomplikowane. Podręczniki poświęcone strukturom danych omawiają wewnętrzne algorytmy mieszające bardziej szczegółowo.

Dobra funkcja mieszająca ma dwa zadania: po pierwsze, równomiernie rozdzielać rekordy w przestrzeni adresowej, aby zminimalizować kolizje i umożliwić w ten sposób zlokalizowanie rekordu o danym kluczu za pomocą jednej operacji dostępu; po drugie (co stoi w pewnej sprzeczności z poprzednim punktem), osiągnąć pierwszy cel, w pełni wykorzystując pakiety, tak aby nie pozostawiać wielu nieużywanych pozycji. Symulacje i analizy pokazują, że zazwyczaj najlepszym rozwiązaniem jest wypełnienie tabeli mieszającej do poziomu 70 – 90 procent, tak aby liczba kolizji pozostawała na niewysokim poziomie i nie była marnowana zbyt duża ilość przestrzeni. Stąd, jeżeli można oczekiwać, że w tabeli będziemy przechowywać  $r$  rekordów, powinniśmy wybrać  $M$  pozycji dla przestrzeni adresowej, tak aby wartość  $(r/M)$  wynosiła od 0,7 do 0,9. Przydatne może się również okazać wybranie jako  $M$  liczby pierwszej, gdyż wykazano, że powoduje to lepsze rozłożenie adresów mieszających w przestrzeni adresowej, kiedy używana jest funkcja mieszająca mod. Inne funkcje mieszające mogą wymagać, aby  $M$  było potęgą liczby 2.

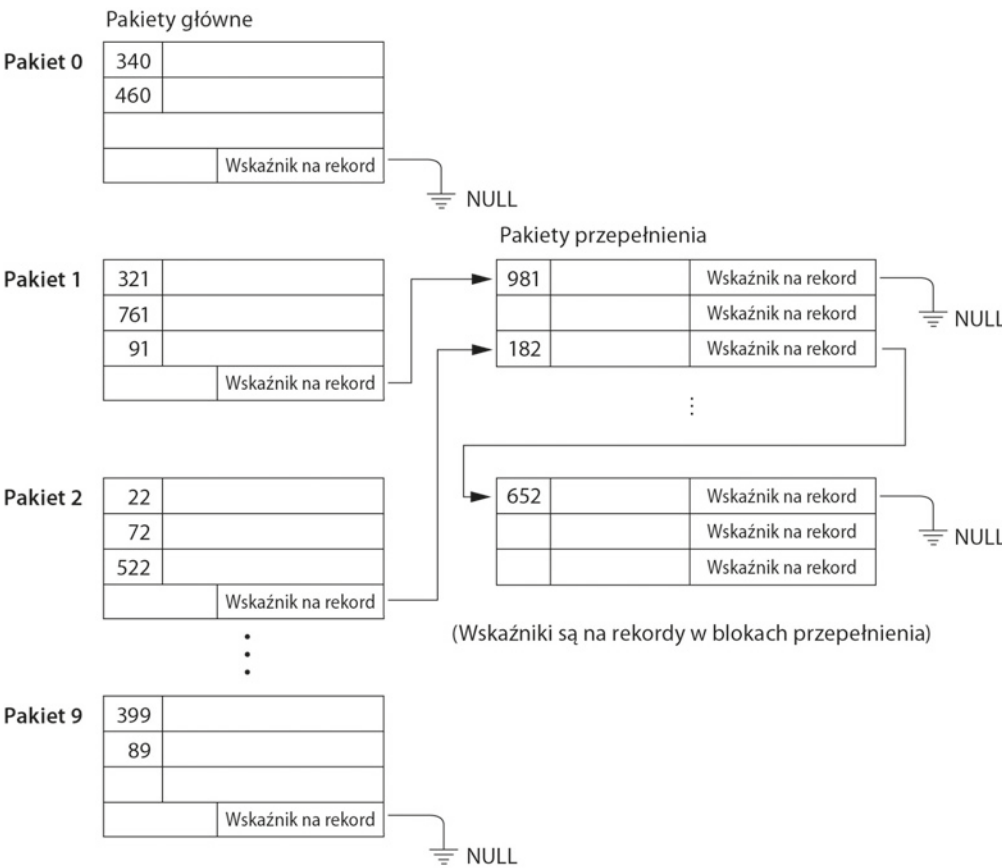
## 16.8.2. Mieszanie zewnętrzne dla plików na dysku

Mieszanie dla plików na dysku określa się mianem **mieszania zewnętrznego** (ang. *external hashing*). W celu dostosowania cech pamięci dyskowej docelowa przestrzeń adresowa jest dzielona na **pakiety** (ang. *buckets*), z których każdy zawiera wiele rekordów. Pakiet jest albo pojedynczym blokiem dyskowym, albo klastrem kolejnych bloków. Funkcja mieszająca odwzorowuje klucz na względny numer pakietu, zamiast przypisywać bezwzględny adres bloku do pakietu. Tabela przechowywana w nagłówku pliku konwertuje numer pakietu na adres odpowiedniego bloku dyskowego, co przedstawiono na rysunku 16.9.

Problem kolizji jest mniej istotny w przypadku pakietów, ponieważ tyle rekordów, ile mieści się w pakiecie, może po zastosowaniu funkcji mieszającej dawać ten sam pakiet bez powodowania problemów. Jednakże należy uwzględnić przypadek, gdy pakiet zostanie całkowicie wypełniony i nowy rekord okaże się pasować do takiego pakietu. Można wówczas wykorzystać odmianę techniki łączenia, gdzie w każdym pakiecie jest przechowywany wskaźnik na listę jednokierunkową rekordów przepełnienia dla danego pakietu, jak na rysunku 16.10. Wskaźniki na liście jednokierunkowej powinny być **wskaźnikami na rekordy**, czyli powinny zawierać zarówno adres bloku, jak i względną pozycję rekordu w bloku.



RYSUNEK 16.9. Odzworowanie numeru pakietu na adres bloku dyskowego



RYSUNEK 16.10. Obsługa przepełnienia pakietów poprzez zastosowanie techniki łączenia

Mieszanie zapewnia najszybszy możliwy sposób dostępu w celu pobrania dowolnego rekordu o danej wartości pola mieszającego. Choć większość dobrych funkcji mieszających nie przechowuje rekordów w porządku wartości pola mieszającego, to jest tak w przypadku niektórych funkcji określanych mianem **zachowujących porządek** (ang. *order preserving*). Prostym przykładem funkcji zachowującej porządek jest wzięcie trzech pierwszych cyfr z pola numeru faktury jako adresu mieszającego i zachowanie tych rekordów w każdym pakiecie posortowanych według numeru faktury. Kolejnym przykładem jest użycie całkowitoliczbowego klucza mieszającego bezpośrednio jako indeksu do pliku względnego, jeżeli wartości klucza mieszającego zapełnia określony przedział wartości. Na przykład, jeżeli numery pracowników w firmie zostaną przypisane jako 1, 2, 3, ... aż do całkowitej liczby pracowników, możemy użyć tożsamościowej funkcji mieszającej (adres względny = klucz), która zachowuje porządek. Niestety, sprawdza się to tylko w przypadku, gdy klucze są generowane przez aplikację po kolei.

Opisany schemat mieszania nosi nazwę **mieszania statycznego** (ang. *static hashing*), ponieważ przydzielana jest stała liczba  $M$  pakietów. Funkcja przeprowadza wtedy odwzorowanie kluczy na adresy przy stałej przestrzeni adresowej. Może to stanowić poważną wadę w przypadku plików dynamicznych. Załóżmy, że przydzielono  $M$  pakietów dla przestrzeni adresowej i  $m$  oznacza maksymalną liczbę rekordów, jakie mogą się zmieścić w jednym pakiecie. Wówczas najwyżej  $(m \cdot M)$  rekordów zmieści się w przydzielonej przestrzeni. Jeżeli liczba rekordów okaże się znacznie mniejsza od  $(m \cdot M)$ , pozostanie nam wiele nieużywanej przestrzeni. Z drugiej strony, jeżeli liczba rekordów znacznie przekroczy wartość  $(m \cdot M)$ , pojawi się wiele kolizji i pobieranie zostanie spowolnione ze względu na długie listy rekordów przepełnienia. W obu przypadkach może zająć potrzeba zmiany liczby przydzielonych bloków  $M$  i użycia nowej funkcji mieszającej (w oparciu o nową wartość  $M$ ) w celu rozłożenia rekordów. Takie reorganizacje mogą być czasochłonne w przypadku dużych plików. Nowsze dynamiczne organizacje plików bazujące na mieszaniu pozwalają na dynamiczne zmienianie liczby pakietów i przeprowadzanie tylko lokalnych reorganizacji (patrz podrozdział 16.8.3).

W przypadku używania mieszania zewnętrznego wyszukiwanie rekordu o podanej wartości pewnego pola, innego niż pole mieszające, jest równie kosztowne jak w przypadku pliku nieuporządkowanego. Kasowanie rekordów może zostać zaimplementowane jako usuwanie ich z pakietu. Jeżeli pakiet posiada łańcuch przepełnień, możemy przenieść jeden z rekordów przepełnienia do pakietu, zastępując usunięty rekord. Jeżeli usuwany rekord jest rekordem przepełnienia, usuwamy go po prostu z listy jednokierunkowej. Należy zauważyć, że usuwanie rekordu przepełnienia implikuje, że należy zapamiętywać puste pozycje przepełnienia. Można to łatwo zapewnić, przechowując listę jednokierunkową nieużywanych pozycji przepełnienia.

Modyfikacja wartości pola rekordu zależy od dwóch czynników: warunku wyszukiwania w celu zlokalizowania rekordu oraz pola, które ma zostać zmodyfikowane. Jeżeli warunek wyszukiwania jest porównaniem równościowym z polem mieszającym, możemy wydajnie zlokalizować rekord, używając funkcji mieszającej. W przeciwnym razie musimy wykonać wyszukiwanie liniowe. Pole niebędące polem mieszającym można zmodyfikować, zmieniając rekord i zapisując go ponownie w tym samym pakiecie. Modyfikacja pola mieszającego oznacza, że rekord może zostać przeniesiony do innego pakietu, co wymaga usunięcia starego rekordu i wstawienia zmodyfikowanego.

### 16.8.3. Techniki mieszania umożliwiające dynamiczne rozszerzanie plików

Główną wadą omówionego powyżej schematu mieszania *statycznego* jest to, że przestrzeń adresów mieszających jest stała. Stąd rozszerzanie lub skracanie pliku w sposób dynamiczny jest trudne. Schematy opisane w niniejszym podrozdziale stanowią próbę poprawienia tej sytuacji. Pierwszy schemat — mieszanie rozszerzalne — przechowuje oprócz pliku strukturę dostępową, stąd przypomina nieco indeksowanie (rozdział 17.). Główna różnica polega na tym, że tutaj struktura dostępową jest oparta na wartościach, które są wynikiem zastosowania funkcji mieszającej względem pola wyszukiwania. W przypadku indeksowania struktura dostępową jest oparta na wartościach samego pola wyszukiwania. Druga technika, określana mianem mieszania liniowego, nie wymaga dodatkowych struktur dostępowych. Jeszcze inna metoda, **haszowanie dynamiczne**, wykorzystuje strukturę dostępową opartą na drzewach binarnych.

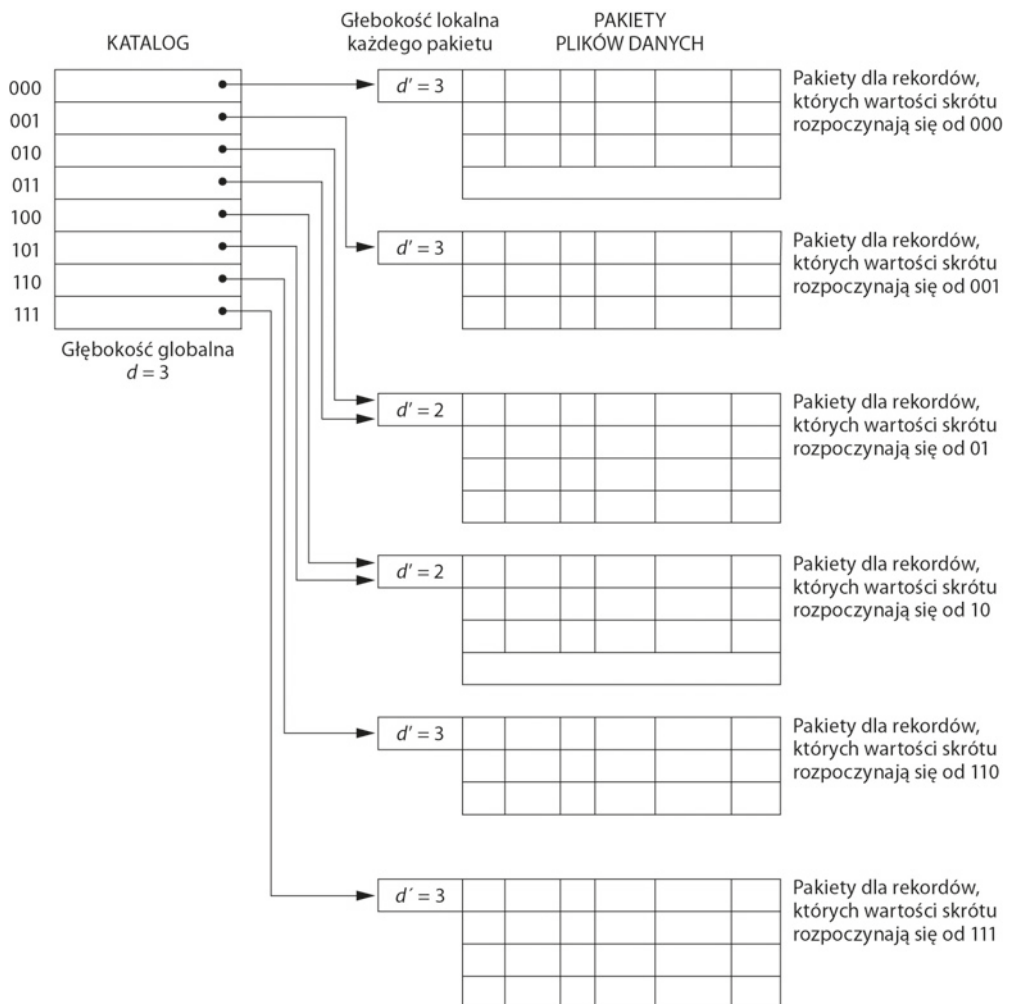
Takie schematy mieszania wykorzystują fakt, że wynik zastosowania funkcji mieszającej jest nieujemną wartością całkowitą, może zatem być reprezentowany jako liczba binarna. Struktura dostępową jest tworzona na podstawie **reprezentacji binarnej** wyniku funkcji mieszającej, czyli ciągu **bitów**. Nazywamy ją **wartością skrótu** (ang. *hash value*) rekordu. Rekordy są rozkładane między pakiety w oparciu o wartości *bitów wiodących* (ang. *leading bits*) w ich wartościach skrótu.

**Mieszanie rozszerzalne.** W przypadku mieszania rozszerzalnego (zaproponowanego przez Fagina, 1979) przechowywany jest typ **katalogu** (ang. *directory*) — tablicy o  $2^d$  adresach pakietów, gdzie  $d$  określa się mianem **głębokości globalnej** (ang. *global depth*) katalogu. Wartość całkowita odpowiadająca pierwszemu (wyższemu)  $d$  bitom wartości skrótu jest używana jako indeks do tablicy w celu określenia pozycji katalogu. Adres znajdujący się na tej pozycji określa pakiet, w którym są przechowywane odpowiednie rekordy. Jednakże nie musi występować oddzielny pakiet dla każdej z  $2^d$  różnych lokalizacji katalogu. Kilka lokalizacji katalogu o tych samych pierwszych  $d'$  bitach dla swoich wartości skrótu może zawierać ten sam adres pakietu, jeżeli wszystkie rekordy odwzorowywane na te lokalizacje mieszczą się w jednym pakiecie. **Głębokość lokalna**  $d'$  (ang. *local depth*  $d'$ ) — przechowywana w każdym pakiecie — określa liczbę bitów, na których jest oparta zawartość pakietu. Na rysunku 16.11 przedstawiono katalog o głębokości globalnej  $d = 3$ .

Wartość  $d$  można zwiększać lub zmniejszać o jeden, co podwaja lub zmniejsza dwukrotnie liczbę wpisów w tablicy katalogu. Podwajanie jest wymagane, kiedy pakiet, którego głębokość lokalna  $d'$  jest równa głębokości globalnej  $d$ , ulegnie przepełnieniu. Dwukrotne zmniejszenie występuje wówczas, gdy  $d > d'$  dla wszystkich pakietów po wystąpieniu pewnej operacji usuwania. Większość operacji pobierania rekordów wymaga uzyskania dostępu do dwóch bloków — jednego do katalogu i jednego do pakietu.

W celu zilustrowania mechanizmu dzielenia pakietów założmy, że nowo wstawiony rekord spowodował przepełnienie w pakiecie, którego wartości skrótu rozpoczynają się od 01 — jest to trzeci pakiet na rysunku 16.11. Rekordy będą rozłożone w dwóch pakietach: pierwszym, zawierającym wszystkie rekordy, których wartości skrótu rozpoczynają się od 010 i drugim, zawierającym rekordy, których wartości skrótu rozpoczynają się od 011. W tym momencie dwie lokalizacje katalogu dla 010 i 011 wskazują na dwa nowe oddzielne pakiety. Przed dokonaniem rozdzielenia wskazywały one na ten sam pakiet. Głębokość lokalna  $d'$  dwóch nowych pakietów wynosi 3, co jest wartością o jeden większą od głębokości lokalnej starego pakietu.





RYSUNEK 16.11. Struktura rozszerzalnego schematu mieszania

Jeżeli pakiet ulegający przepełnieniu i podziałowi wcześniej posiadał głębokość lokalną  $d'$  równą głębokości globalnej  $d$  katalogu, to rozmiar katalogu musi zostać podwojony, tak aby można było wykorzystać dodatkowy bit w celu rozróżnienia dwóch nowych pakietów. Przykładowo, jeżeli pakiet rekordów z rysunku 16.11, których wartości skrótu rozpoczynają się od 111, ulegnie przepełnieniu, dwa nowe pakiety wymagają katalogu o głębokości globalnej równej 4, ponieważ dwa nowe pakiety posiadają etykiety 1110 oraz 1111, a zatem ich głębokości lokalne wynoszą 4. Dlatego też rozmiar katalogu zostaje podwojony i każda z innych oryginalnych lokalizacji w katalogu również jest dzielona na dwie lokalizacje, z których obie posiadają taką samą wartość wskaźnika jak lokalizacja oryginalna.

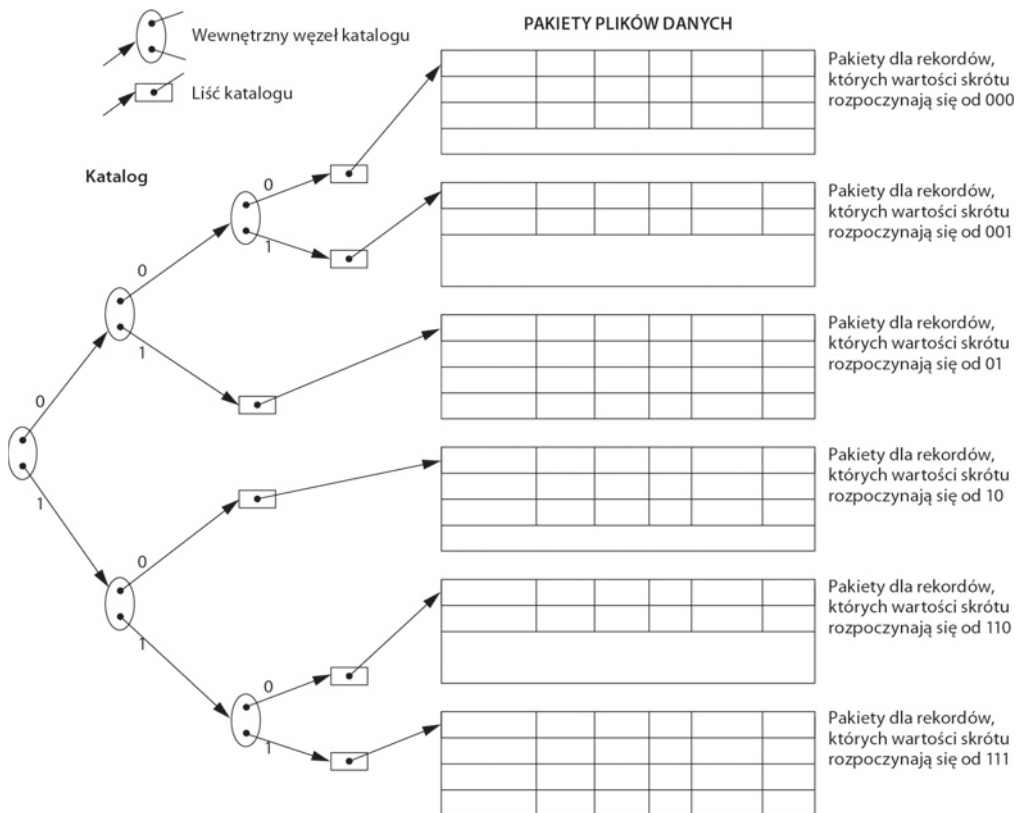
Główną korzyścią związaną z mieszaniem rozszerzalnym jest to, że *wydajność działań na pliku nie spada wraz ze wzrostem jego rozmiaru*, co ma miejsce w przypadku statycznego mieszania zewnętrznego, gdzie liczba kolizji wzrasta i związane z tym łączenia powodują konieczność wykonywania dodatkowych operacji dostępu. Ponadto w przypadku mieszania rozszerzalnego nie ma potrzeby przydzielania żadnej przestrzeni dla przyszłego wzrostu pliku, gdyż dodatkowe pakiety mogą być dowolnie przydzielane w razie potrzeby. Narzut związany z koniecznością przechowywania tabeli katalogu jest pomijalnie mały. Maksymalny rozmiar katalogu wynosi  $2^k$ , gdzie  $k$  jest liczbą bitów wartości skrótu. Kolejną korzyścią jest to, że dzielenie pakietów powoduje w większości przypadków niewielkie reorganizacje, gdyż tylko rekordy z jednego pakietu są rozkładane na dwa pakiety. Jedyna sytuacja, w której reorganizacja jest bardziej kosztowna, to przypadek, gdy katalog musi zostać podwojony (lub zmniejszony dwukrotnie). Wadą takiego rozwiązania jest to, że katalog musi zostać przeszukany przed uzyskaniem dostępu do samych pakietów, co sprawia, że konieczne są dwie operacje dostępu do bloków zamiast jednej, jak ma to miejsce w przypadku mieszania statycznego. Takie zmniejszenie wydajności traktuje się jako mało znaczące, zatem schemat ten jest uważany za bardzo pożądanym w przypadku plików dynamicznych.

**Mieszanie dynamiczne.** Poprzednikiem mieszania rozszerzalnego było mieszanie dynamiczne, zaproponowane przez Larsona (1978). W tej metodzie adresami pakietów było albo  $n$ , albo  $n - 1$  wyższych bitów (w zależności od łącznej liczby kluczy należących do danego pakietu). W mieszaniu dynamicznym zapis rekordów w pakietach przebiega podobnie jak w mieszaniu rozszerzalnym. Główna różnica między tymi metodami dotyczy organizacji katalogów. W mieszaniu rozszerzalnym uwzględniana jest globalna głębokość ( $d$  wyższych bitów) dla płaskiego katalogu, a przyległe powiązane pakiety są łączone w pakiet o lokalnej głębokości  $d - 1$ . W haszowaniu dynamicznym przechowywany jest katalog drzewiasty z dwoma rodzajami węzłów:

- Węzły wewnętrzne z dwoma wskaźnikami — lewy wskaźnik odpowiada bitowi 0 (z adresu uzyskanego za pomocą mieszania), a prawy bitowi 1.
- Liście — przechowują wskaźnik do konkretnego pakietu z rekordami.

Przykład mieszania dynamicznego jest pokazany na rysunku 16.12. Widoczne są tu cztery pakiety („000”, „001”, „110” i „111”) z adresami opartymi na trzech wyższych bitach (głębokość globalna równa 3) i dwa pakiety („01” i „10”) z adresami w postaci dwóch wyższych bitów (głębokość lokalna równa 2). Te dwa ostatnie adresy to wynik złączenia pakietów „010” i „011” w „01” oraz pakietów „100” i „101” w „10”. Zauważ, że w mieszaniu dynamicznym węzły katalogu są pośrednio używane do ustalania globalnej i lokalnej głębokości pakietów. Wyszukiwanie rekordu na podstawie uzyskanego za pomocą mieszania adresu polega na poruszaniu się po drzewie, co prowadzi do pakietu przechowującego dany rekord. Jako zadanie dla Czytelnika pozostawiamy opracowanie algorytmów wstawiania, usuwania i wyszukiwania rekordów w modelu wykorzystującym mieszanie dynamiczne.

**Mieszanie liniowe.** Idea mieszania liniowego (zaproponowanego w Litwin, 1980) polega na dopuszczeniu, aby plik mieszający zwiększał i zmniejszał liczbę swoich pakietów dynamicznie *bez* użycia katalogu. Załóżmy, że plik na początku posiada  $M$  pakietów ponumerowanych od 0 do  $M-1$  i wykorzystywana jest funkcja mieszająca mod postaci  $h(K) = K \bmod M$ . Taką funkcję określamy mianem **początkowej funkcji mieszającej**  $h_i$ . Wciąż



Rysunek 16.12. Struktura schematu mieszania dynamicznego

konieczne jest uwzględnianie przepełnień związanych z kolizjami i można to zapewnić, przechowując oddzielne łańcuchy przepełnienia dla każdego pakietu. Jednakże, kiedy kolizja prowadzi do rekordu przepełnienia w *dowolnym* pakiecie pliku, *pierwszy* pakiet w tym pliku — pakiet 0 — zostaje podzielony na dwa pakiety: oryginalny pakiet 0 oraz nowy pakiet  $M$ , znajdujący się na końcu pliku. Rekordy znajdujące się oryginalnie w pakiecie 0 zostają rozłożone między te dwa nowe pakiety w oparciu o inną funkcję mieszającą  $h_{i+1}(K) = K \bmod 2M$ . Kluczową właściwością dwóch funkcji mieszających  $h_i$  oraz  $h_{i+1}$  jest to, że dowolne rekordy, które wcześniej zostały dopasowane do pakietu 0 w oparciu o funkcję  $h_i$ , teraz będą pasować albo do pakietu 0, albo pakietu  $M$  w oparciu o funkcję  $h_{i+1}$ . Jest to konieczne w celu zapewnienia poprawnego działania mieszania liniowego.

W miarę jak przyszłe kolizje prowadzą do rekordów przepełnienia, kolejne pakiety są dzielone w kolejności *liniowej* 1, 2, 3, .... Jeżeli wystąpi wystarczająca liczba przepełnień, wszystkie oryginalne pakiety pliku 0, 1, ...,  $M-1$  zostaną podzielone, więc plik będzie posiadał  $2M$  zamiast  $M$  pakietów i wszystkie pakiety będą wykorzystywać funkcję mieszającą  $h_{i+1}$ . Stąd rekordy przepełnienia zostaną w konsekwencji rozłożone na pakiety normalne przy użyciu funkcji  $h_{i+1}$  poprzez *opóźniony podział* ich pakietów. Nie ma potrzeby przechowywania katalogu, a tylko wartości  $n$ , która początkowo ma wartość 0 i jest zwiększana o 1, kiedy występuje podział, i która jest potrzebna w celu określenia, który

pakiet został ostatnio podzielony. W celu pobrania rekordu o wartości  $K$  klucza mieszającego najpierw stosujemy funkcję  $h_i$  względem  $K$ . Jeżeli  $h_i(K) < n$ , wówczas stosujemy funkcję  $h_{i+1}$  względem  $K$ , ponieważ pakiet został już podzielony. Początkowo  $n = 0$ , co określa, że funkcja  $h_i$  ma zastosowanie względem wszystkich pakietów. W miarę podziału pakietów wartość  $n$  rośnie liniowo.

Kiedy  $n = M$ , oznacza to, że wszystkie oryginalne pakiety zostały podzielone i funkcja mieszająca  $h_{i+1}$  ma zastosowanie względem wszystkich rekordów w pliku. W tym momencie  $n$  jest zerowane i wszelkie nowe kolizje powodujące przepełnienie prowadzą do wykorzystania nowej funkcji mieszającej  $h_{i+2}(K) = K \bmod 4M$ . Ogólnie rzecz biorąc, używana jest sekwencja funkcji mieszających  $h_{i+j}(K) = K \bmod (2^j M)$ , gdzie  $j = 0, 1, 2, \dots$ . Nowa funkcja mieszająca  $h_{i+j+1}$  jest potrzebna wówczas, gdy wszystkie pakiety  $0, 1, \dots, (2^j M) - 1$  zostaną podzielone i  $n$  zostanie wyzerowane. Wyszukiwanie rekordu o wartości  $K$  klucza mieszającego zostało zdefiniowane w algorytmie 16.3.

ALGORYTM 16.3. Procedura wyszukiwania dla mieszania liniowego

```

if  $n = 0$ 
  then  $m \leftarrow h_j(K)$  (*  $m$  jest wartością skrótu rekordu o kluczu mieszającym  $K$  *)
  else begin
     $m \leftarrow h_j(K)$ ;
    if  $m < n$  then  $m \leftarrow h_{j+1}(K)$ 
  end;
wyszukaj pakiet, którego wartością skrótu jest  $m$  (i pakiet przepełnienia, jeśli
  ↳ taki istnieje);

```

Dzielenie można kontrolować dzięki monitorowaniu współczynnika zapełnienia pliku, zamiast dokonywać go za każdym razem, gdy wystąpi przepełnienie. Ogólnie rzecz biorąc, **współczynnik zapełnienia pliku** (ang. *file load factor*)  $l$  można zdefiniować jako  $l = r/(bfr \cdot N)$ , gdzie  $r$  jest bieżącą ilością rekordów w pliku,  $bfr$  jest maksymalną liczbą rekordów, które mogą zmieścić się w pakiecie, zaś  $N$  jest bieżącą liczbą pakietów pliku. Pakiety, które podzielono, można również ponownie łączyć, jeżeli poziom zapełnienia pliku spadnie poniżej określonej wartości progowej. Bloki są łączone w sposób liniowy, a wartość  $N$  jest odpowiednio zmniejszana. Zapełnienie pliku można wykorzystać jako wyzwalacz zarówno dla podziałów, jak i scaleń. W ten sposób współczynnik ten można utrzymać w pożądanym zakresie wartości. Podziały mogą być wywoływane w momencie, gdy zapełnienie przekroczy określoną wartość progową, na przykład 0,9, zaś scalenia mogą być wywoływane w momencie, gdy zapełnienie spadnie poniżej innej wartości progowej, na przykład 0,7. Główną zaletą mieszania liniowego jest to, że zachowuje ono stosunkowo równomierny współczynnik zapełnienia w trakcie powiększania się i zmniejszania pliku. Ponadto technika ta nie wymaga katalogu<sup>15</sup>.

<sup>15</sup> Szczegółowe omówienie wstawiania i usuwania w plikach z mieszaniem liniowym znajdziesz w Litwin (1980) i Salzberg (1988).

## 16.9. Inne podstawowe metody organizacji plików

### 16.9.1. Pliki rekordów mieszanych

W przypadku metod organizacji plików omawianych powyżej zakładaliśmy, że wszystkie rekordy określonego pliku mają ten sam typ. Rekordami mogły być elementy typu PRACOWNIK, PROJEKT, STUDENT lub WYDZIAŁ, ale każdy plik musiał zawierać rekordy jednego typu. W większości aplikacji bazodanowych spotykamy sytuacje, w których wiele typów encji jest ze sobą powiązanych na różne sposoby, co omówiono w rozdziale 7. Związki między rekordami w różnych plikach można reprezentować za pomocą **pól łączących** (ang. *connecting fields*)<sup>16</sup>. Przykładowo, rekord STUDENT może posiadać pole łączące WYDZKIERUNKU, którego wartość określa nazwę WYDZIAŁU, na którym student prowadzi główny kierunek studiów. To pole WYDZKIERUNKU *odwołuje się* do encji WYDZIAŁ, która powinna być reprezentowana przez własny rekord w pliku WYDZIAŁ. Jeżeli chcemy pobrać wartości pól z dwóch powiązanych rekordów, musimy jeden z nich pobrać jako pierwszy. Następnie możemy wykorzystać wartość jego pola łączącego w celu pobrania powiązanego rekordu z drugiego pliku. Stąd związki są realizowane za pomocą **logicznych odwołań do pól** (ang. *logical field references*) między rekordami z oddzielnych plików.

Metody organizacji plików w przypadku obiektowych systemów SZBD, jak również w starszych systemach, takich jak systemy SZBD hierarchiczne i sieciowe, często zawierają implementacje związków wśród rekordów w postaci **związków fizycznych** (ang. *physical relationships*) realizowanych poprzez zapewnienie fizycznego sąsiedztwa (klastrowanie) powiązanych rekordów lub za pomocą fizycznych wskaźników. Takie metody organizacji zwykle polegają na przydzieleniu **obszaru** dysku dla rekordów więcej niż jednego typu, tak aby rekordy różnych typów mogły być **fizycznie klastrowane** na dysku. Jeżeli można oczekiwać, że określone powiązanie będzie często używane, zastosowanie fizycznego powiązania może zwiększyć wydajność działania systemu w zakresie pobierania powiązanych rekordów. Przykładowo, jeżeli zapytanie pobierające rekord encji WYDZIAŁ i wszystkie rekordy encji STUDENT dla studentów, dla których jest on głównym wydziałem, jest wykonywane bardzo często, pożądanym rozwiązaniem jest umieszczenie każdego rekordu WYDZIAŁ w klastrze z rekordami STUDENT obok siebie na dysku w pliku mieszanym. Pojęcie **klastrowania fizycznego** (ang. *physical clustering*) typów obiektowych jest wykorzystywane w obiektowych systemach SZBD w celu przechowywania powiązanych obiektów wspólnie w pliku mieszanym. W hurtowniach danych (patrz rozdział 29.) dane wejściowe pochodzą z różnych źródeł i są wstępnie integrowane w celu pobrania potrzebnych danych do **operacyjnego magazynu danych** (ang. *operational data store* — ODS). Magazyn ODS zawiera zwykle pliki, w których razem przechowywane są rekordy wielu typów. Magazyn ODS jest przekazywany do hurtowni danych po tym, jak wykonane zostaną na nim operacje ETL (ang. *extract, transform i load*, czyli pobieranie, przekształcanie i wczytywanie).

W celu rozróżnienia rekordów w pliku mieszanym, każdy z nich posiada (oprócz pól wartości) pole **typu rekordu**, które określa typ rekordu. Jest to zwykle pierwsze pole w każdym rekordzie i jest ono używane przez oprogramowanie systemowe w celu określenia

---

<sup>16</sup> Pojęcie kluczy obcych w modelu relacyjnym (rozdział 3.) oraz odwołania między obiektami w modelach obiektowo-relacyjnych (rozdział 11.) stanowią przykłady pól łączących.

typu rekordu, który ma zostać poddany przetworzeniu. Używając informacji katalogowych, system SZBD może określić pola danego typu rekordu i ich rozmiary w celu zinterpretowania wartości danych zawartych w rekordzie.

## 16.9.2. B-drzewa i inne struktury danych służące jako podstawowe metody organizacji

Jako podstawowych metod organizacji plików można używać innych struktur danych. Przykładowo, niektóre systemy SZBD oferują opcję użycia struktury B-drzew jako podstawowej metody organizacji pliku, jeżeli zarówno rozmiar rekordu, jak i liczba rekordów w pliku są małe. B-drzewa zostaną opisane w punkcie 17.3.1 przy omawianiu ich użycia w celu indeksowania. Ogólnie rzecz biorąc, wszelkie struktury danych, które można dostosować do charakterystyk urządzeń dyskowych, mogą być używane jako podstawowe metody organizacji plików dla rozmieszczenia rekordów na dysku. Niedawno zaproponowano przechowywanie danych na podstawie kolumn jako podstawową metodę składowania relacji w bazach relacyjnych. W rozdziale 17. przedstawimy krótkie wprowadzenie do tej techniki jako alternatywnego schematu składowania danych w bazach relacyjnych.

## 16.10. Zapewnianie równoległego dostępu do dysku przy użyciu architektury RAID

Przy wykładniczym wzroście wydajności i pojemności urządzeń półprzewodnikowych dostępne stają się coraz szybsze mikroprocesory i pojemniejsze pamięci główne. Naturalną rzeczą jest oczekiwanie, że techniki drugorzędnych metod składowania danych będą rozwijane w równie szybkim tempie w zakresie wydajności i niezawodności, w celu dostosowania się do tego wzrostu.

Jednym z najważniejszych osiągnięć w zakresie technik składowania danych było opracowanie architektury **RAID** (początkowo skrót ten oznaczał *Redundant Arrays of Inexpensive Disks* — **nadmiarowe macierze niedrogich dysków**, ale obecnie literę *I* z akronimu odczytuje się jako *Independent* — **niezależnych**). Idea działania urządzeń RAID została szybko zaakceptowana przez przemysł i opracowano cały zestaw alternatywnych architektur RAID (o poziomach od 0. do 6.). Poniżej zostaną omówione najważniejsze funkcje oferowane przez to rozwiązanie.

Głównym celem architektury RAID jest zniwelowanie różnicy poziomów wzrostu wydajności działania dysków w porównaniu z pamięciami i mikroprocesorami<sup>17</sup>. Gdy pojemności pamięci wzrastają czterokrotnie co dwa, trzy lata, zmniejszenie *czasu dostępu* w przypadku dysków wynosi około 10 procent rocznie, zaś *szybkość transferu danych* wzrasta około 20 procent rocznie. Co prawda *pojemności* dysków wzrastają o ponad 50 procent każdego roku, jednak wskaźniki wzrostu prędkości i zmniejszania czasu dostępu są o wiele niższe.

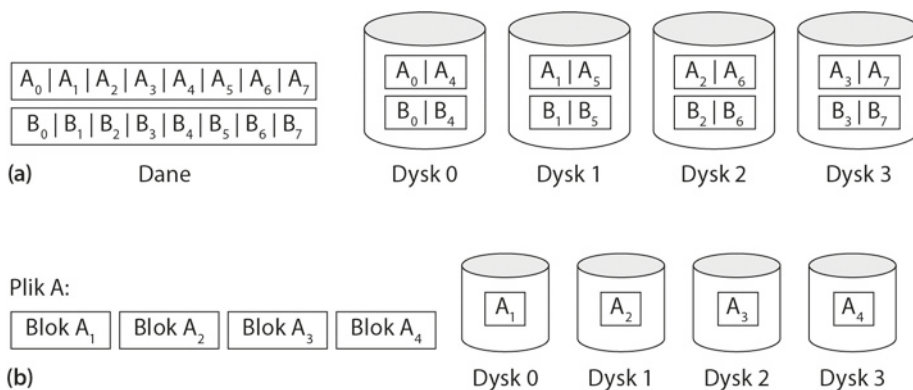
---

<sup>17</sup> Wzrost ten został oszacowany przez Gordona Bella na poziomie 40 procent rocznie w latach 1974 – 1984, a obecnie ocenia się, że wynosi 50 procent rocznie.



Druga spora różnica występuje w przypadku specjalizowanych mikroprocesorów, które oferują nowym aplikacjom możliwość przetwarzania danych wizualnych, audio, grafiki i danych przestrzennych (szczegółowe omówienie takich aplikacji zawiera rozdział 26.), co kontrastuje z brakiem szybkiego dostępu do dużych, współużytkowanych zbiorów danych.

Naturalnym rozwiązaniem tego problemu jest wykorzystanie dużej macierzy niewielkich, niezależnych dysków funkcjonujących jako jeden dysk logiczny o lepszych parametrach. Używa się tu pojęcia **przeplotu danych** (ang. *data striping*), które oznacza wykorzystanie technik działań *równoległych* w celu zwiększenia wydajności działania dysków. **Przeplot danych** powoduje transparentne rozłożenie danych między wieloma dyskami w celu sprawienia, aby wyglądały jak jeden duży i szybki dysk. Na rysunku 16.13(a) przedstawiono plik rozłożony, czyli poddany przeplotowi, między czterema dyskami; bity (0, 4) są przypisane do dysku 0, bity (1, 5) do dysku 1 itd. W tym modelu każdy dysk uczestniczy w każdym odczycie i zapisie. Liczbaostępów na sekundę na poziomie jednego dysku pozostaje taka sama, jednak ilość danych wczytywanych w jednostce czasu rośnie czterokrotnie. Przeplatanie zwiększa więc ogólną wydajność operacji wejścia-wyjścia, co zapewnia wysoką szybkość przesyłania danych. **Przeplot na poziomie bloków** polega na podziale bloków między dyski. Macierz dysków jest wtedy traktowana jak jeden dysk. Bloki są logicznie numerowane, począwszy od 0. Dyski w macierzy  $m$ -dyskowej mają numery od 0 do  $m - 1$ . Przeplot powoduje, że blok  $j$  trafia na dysk  $(j \bmod m)$ . Na rysunku 16.13(b) pokazano przeplot na poziomie bloków z użyciem czterech dysków ( $m = 4$ ). Przeplot danych umożliwia również równoważenie obciążenia między dyskami. Ponadto przechowywanie nadmiarowych informacji na dyskach przy użyciu mechanizmów parzystości lub innych kodów korekcji błędów pozwala na zwiększenie poziomu niezawodności. W punktach 16.10.1 i 16.10.2 omówiono, w jaki sposób architektura RAID umożliwia osiągnięcie dwóch istotnych celów związanych ze zwiększoną niezawodnością i wydajnością. Z kolei w podrozdziale 16.10.3 omówiono metody organizacji i poziomy systemów RAID.



Rysunek 16.13. Przeplot danych z użyciem wielu dysków. (a) Przeplot na poziomie bitów z wykorzystaniem czterech dysków. (b) Przeplot na poziomie bloków z użyciem czterech dysków



## 16.10.1. Zwiększanie niezawodności przy użyciu architektury RAID

W przypadku macierzy  $n$  dysków prawdopodobieństwo wystąpienia awarii jest  $n$  razy większe niż w przypadku jednego dysku. Stąd, jeżeli parametr *MTBF* (ang. *Mean Time Between Failures*, średni czas między awariami) w przypadku pojedynczego dysku wynosi 200 000 godzin, czyli 22,8 lat (dla dysku Seagate Enterprise Performance 10K HDD ten czas wynosi 1,4 mln godzin), to w przypadku 100 dysków jego wartością staje się 2 000 godzin, czyli 83,3 dnia (dla zestawu 100 dysków Seagate Enterprise Performance 10K HDD ten czas to 1400 godzin, czyli 58,33 dnia). Przechowywanie pojedynczej kopii danych w ramach takiej macierzy dysków wiąże się z ogromnym spadkiem niezawodności. Oczywistym rozwiązaniem tego problemu jest wykorzystanie nadmiarowości danych, tak aby awarie dysków mogły być tolerowane. Wiąże się to jednak z wieloma wadami: dodatkowe operacje zapisu, dodatkowy narzut obliczeniowy związany z nadmiarowością i odtwarzaniem po błędach, a także dodatkowa przestrzeń dyskowa, którą należy zarezerwować dla nadmiarowych danych.

Jedna z technik obsługi nadmiarowości nosi nazwę **dublowania** (ang. *mirroring, shadowing*). Dane są zapisywane nadmiarowo na dwóch identycznych dyskach fizycznych, które traktuje się jak pojedynczy dysk logiczny. Kiedy dane są odczytywane, można je pobrać z dysku o krótszym czasie oczekiwania, wyszukiwania i opóźnienia rotacyjnego. W razie awarii jednego z dysków można korzystać z drugiego do czasu naprawienia pierwszego. Załóżmy, że średni czas naprawy wynosi 24 godziny. W takim przypadku średni czas do utraty danych w przypadku dublowanego systemu dyskowego wykorzystującego 100 dysków o średnim czasie do wystąpienia awarii wynoszącym 200 000 godzin wynosi  $(200\,000)^2 / (2 \cdot 24) = 8,33 \cdot 10^8$  godzin, czyli 95,028 lat<sup>18</sup>. Dublowanie dysków podwaja również szybkość obsługi operacji odczytu, gdyż mogą one dotyczyć dowolnego z dwóch dysków. Jednakże szybkość przesyłania danych każdego z takich dysków pozostaje na tym samym poziomie co w przypadku pojedynczego dysku.

Kolejnym rozwiązaniem problemu niezawodności jest przechowywanie dodatkowych informacji, które nie są w normalnej sytuacji potrzebne, ale które mogą być używane w celu rekonstruowania utraconych danych w razie awarii dysku. Uwzględnienie nadmiarowości musi się wiązać z rozwiązaniem dwóch problemów: wyboru techniki obliczania nadmiarowych informacji i wyboru metody dystrybucji nadmiarowych informacji w macierzy dyskowej. Pierwszy problem można rozwiązać, wykorzystując kody korekcji błędów związane z bitami parzystości lub specjalizowane kody, takie jak kody Hamminga. W przypadku schematu parzystości dysk nadmiarowy może być traktowany jako przechowujący sumy wszystkich danych umieszczonych na innych dyskach. W razie awarii dysku utracone informacje mogą zostać zrekonstruowane w ramach procesu podobnego do odejmowania.

W przypadku drugiego problemu dwa główne podejścia polegają albo na przechowywaniu nadmiarowych informacji na niewielkiej liczbie dysków, albo ich jednorodnym rozłożeniu między wszystkimi dyskami. Drugie z rozwiązań zapewnia lepsze równoważenie obciążenia. Różne poziomy architektury RAID wykorzystują kombinacje tych opcji w celu zrealizowania obsługi nadmiarowości, a stąd poprawienia niezawodności.

---

<sup>18</sup> Wzory dla obliczeń średniego czasu do wystąpienia awarii pochodzą z pracy Chena i in. (1994).

## 16.10.2. Poprawianie wydajności przy użyciu architektury RAID

Macierze dyskowe wykorzystują technikę przeplotu danych w celu osiągnięcia wyższych wskaźników przesyłania danych. Należy zauważyć, że dane mogą być odczytywane lub zapisywane tylko po jednym bloku, więc typowa operacja przesyłania danych uwzględnia od 512 do 8192 bajtów. Przeplot dysków można zastosować na wyższym poziomie szczegółowości, rozbijając bajty danych na bity i rozdzielając je między różne dyski. Stąd **przeplot danych na poziomie bitowym** (ang. *bit-level data striping*) polega na podzieleniu bajta danych i zapisaniu bitu  $j$  na  $j$ -tym dysku. W przypadku bajtów 8-bitowych osiem oddzielnych dysków można traktować jako jeden dysk logiczny o ośmiokrotnie większym poziomie transferu danych. Każdy dysk uczestniczy w każdej operacji wejścia-wyjścia i całkowita liczba danych przypadających na operacje wzrasta ośmiokrotnie. Przeplot na poziomie bitowym można uogólnić na liczbę dysków, która jest albo wielokrotnością, albo podzielnikiem ośmiu. Stąd w przypadku macierzy czterodyskowej bit  $n$  jest przekazywany do dysku o numerze  $(n \bmod 4)$ . Przeplot danych na poziomie bitowym jest przedstawiony na rysunku 16.13(a).

Ziarnistość przeplotu danych może być większa niż pojedynczy bit. Przykładowo, bloki pliku mogą być rozdzielane między dyskami, co określa się mianem **przeplotu na poziomie blokowym** (ang. *block-level striping*). Na rysunku 16.13(b) przedstawiono przeplot danych na poziomie blokowym przy założeniu, że plik danych zawiera cztery bloki. W przypadku przeplotu na poziomie blokowym wiele niezależnych żądań uzyskujących dostęp do pojedynczych bloków (żądania o małym rozmiarze) może być obsługiwanych równolegle przez oddzielne dyski, co zmniejsza czas oczekiwania żądań wejścia-wyjścia. Żądania uzyskujące dostęp do wielu bloków (żądania o dużym rozmiarze) również można obsługiwać równolegle, redukując ich czas odpowiedzi. Ogólnie rzecz biorąc, im więcej dysków znajduje się w macierzy, tym większe potencjalne korzyści można uzyskać. Jednakże, zakładając niezależność występowania awarii, macierz dysków składająca się ze 100 jednostek charakteryzuje się 1/100 niezawodności w porównaniu z pojedynczym dyskiem. Stąd w celu zapewnienia niezawodności z jednoczesnym zwiększeniem wydajności trzeba zastosować nadmiarowość poprzez użycie kodów korekcyjnych i dublowanie dysków.

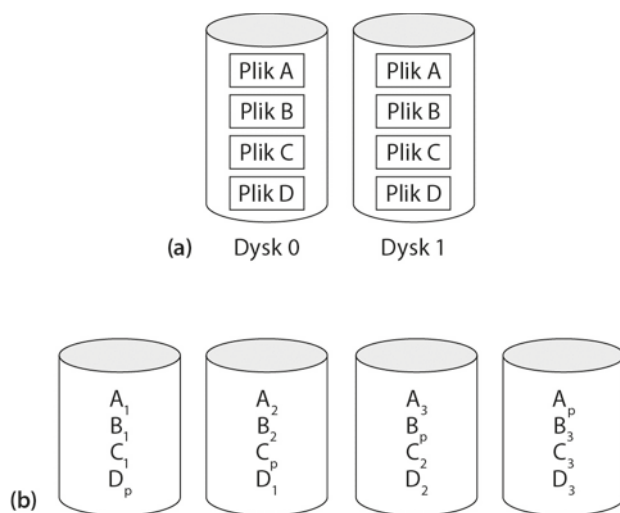
## 16.10.3. Metody organizacji i poziomy architektury RAID

W oparciu o różne kombinacje czynnika ziarnistości przeplotu danych oraz wzorców używanych do obliczania informacji nadmiarowych zdefiniowano różne metody organizacji architektury RAID. Początkowo zaproponowano poziomy od 1. do 5., ale później dodano poziomy 0. i 6.

Architektura RAID poziomu 0. wykorzystuje przeplot danych i nie uwzględnia danych nadmiarowych, stąd charakteryzuje się najlepszymi właściwościami pod względem wydajności zapisu, gdyż aktualizacje nie muszą być duplikowane. Dane są równomiernie rozdzielane między dwa dyski lub większą ich liczbę. Jednak wydajność operacji odczytu nie jest tak dobra jak w przypadku architektury RAID poziomu 1., która wykorzystuje dublowanie dysków. W tym przypadku zwiększenie wydajności jest możliwe poprzez przekazywanie obsługi żądania odczytu do dysku o najkrótszym spodziewanym czasie wyszukiwania i opóźnieniu rotacyjnym. Architektura RAID poziomu 2. wykorzystuje nadmiarowość w stylu pamięci, używając kodów Hamminga, które uwzględniają bity parzy-

stości dla oddzielnych nakładających się podzbiorów komponentów. Stąd w jednej z konkretnych wersji tego poziomu trzy dyski nadmiarowe odpowiadają czterem dyskom normalnym, a w przypadku zastosowania dublowania, tak jak w przypadku poziomu 0., czterem dyskom normalnym. Poziom 2. uwzględnia zarówno wykrywanie, jak i korekcję błędów, choć wykrywanie w zasadzie nie jest wymagane, gdyż uszkodzone dyski same raportują o tym fakcie.

Architektura RAID poziomu 3. stosuje jeden dysk parzystości wykorzystujący kontroler dysku w celu określenia, który dysk uległ awarii. Poziomy 4. i 5. wykorzystują przeplot danych na poziomie blokowym; w przypadku poziomu 5. dane i informacje o parzystości są rozdzielane między wszystkie dyski. Architektury RAID poziomu 5. pokazano na rysunku 16.14(b); bity parzystości są tam oznaczone indeksem dolnym p. Jeśli jeden dysk zawiedzie, brakujące dane są obliczane na podstawie bitów parzystości dostępnych na pozostałych dyskach. Wreszcie poziom 6. wykorzystuje tak zwany schemat nadmiarowości  $P+Q$ , używając kodów Reeda-Solomana w celu zapewnienia ochrony przed maksymalnie dwiema awariami dysków przy użyciu tylko dwóch dysków nadmiarowych.



Rysunek 16.14. Popularne poziomy architektury RAID. (a) Architektura RAID poziomu 1. Dublowanie danych na dwóch dyskach. (b) Architektura RAID poziomu 5. Przeplot danych z bitami parzystości rozproszonymi między cztery dyski

Odtwarzanie w przypadku wystąpienia awarii jest najłatwiejsze dla architektury RAID poziomu 1. Inne poziomy wymagają rekonstrukcji uszkodzonego dysku poprzez odczyt wielu innych dysków. Poziom 1. jest używany w przypadku aplikacji o znaczeniu krytycznym, takich jak przechowywanie dzienników transakcji. Poziomy 3. i 5. są preferowane w przypadku dużych ilości danych, przy czym poziom 3. zapewnia wyższe wskaźniki transferu danych. W większości przypadków zastosowania architektury RAID używa się obecnie poziomu 0. (z przeplotem), poziomu 1. (z dublowaniem) oraz poziomu 5. z dodatkowym dyskiem dla danych o parzystości. Stosowane są też kombinacje różnych poziomów, np.  $0 + 1$  oznacza połączenie przeplotu z dublowaniem z użyciem minimum czterech dysków. Inne niestandardowe poziomy to: RAID 1.5, RAID 7, RAID-DP, RAID S (inaczej Parity RAID), Matrix RAID, RAID-K, RAID-Z, RAIDn, Linux MD RAID 10, IBM ServeRAID 1E

i unRAID. Omawianie tych niestandardowych poziomów wykracza poza zakres tego tekstu. Projektanci konfiguracji mechanizmu RAID dla określonego zbioru aplikacji muszą podjąć wiele decyzji projektowych, takich jak wybór poziomu RAID, liczba dysków, rodzaj stosowanego schematu parzystości oraz grupowanie dysków dla celów przepływu na poziomie blokowym. Prowadzi się szczegółowe badania wydajności odnośnie niewielkich operacji odczytu i zapisu (w kontekście liczby żądań wejścia-wyjścia względem jednej jednostki powielania) oraz dużych operacji odczytu i zapisu (w kontekście liczby żądań wejścia--wyjścia względem jednej jednostki powielania na każdym dysku w grupie korekcji błędów).

## 16.11. Nowoczesne architektury składowania danych

W tym podrozdziale opiszemy nowe osiągnięcia w obszarze systemów składowania danych, stające się integralnym elementem większości architektur systemów informatycznych przedsiębiorstw. Wspomnieliśmy już o interfejsach SATA i SAS, które w laptopach i małych serwerach prawie zastąpiły niegdyś popularne interfejsy SCSI. Interfejs Fibre Channel (czyli światłowody) jest najczęściej wybieranym rozwiązaniem w sieciach składowania w centrach danych. Dalej omówimy niektóre nowoczesne architektury składowania danych.

### 16.11.1. Sieci obszarów składowania danych

Wraz z gwałtownym rozwojem handlu elektronicznego, systemów klasy *Enterprise Resource Planning* (ERP, planowanie wykorzystania zasobów na poziomie korporacyjnym), które integrują dane aplikacji między przedsiębiorstwami, oraz hurtowni danych, które przechowują historyczne zagregowane dane (patrz rozdział 29.), zapotrzebowanie na pamięci masowe znacznie wzrosło. W przypadku dzisiejszych, opartych na internecie przedsiębiorstw koniecznością okazuje się odejście od działań bazujących na wykorzystaniu statycznego centrum danych na rzecz bardziej elastycznej i dynamicznej infrastruktury służącej do przetwarzania posiadanych informacji. Całkowity koszt zarządzania wszystkimi danymi rośnie tak szybko, że w wielu przypadkach koszt zarządzania urządzeniami masowymi dołączanymi do serwera przekracza koszt samego serwera. Ponadto koszty nabycia urządzeń składowania danych stanowią tylko część ogólnej ceny — zazwyczaj jest to 10 do 15 procent całkowitych kosztów zarządzania takimi urządzeniami. Wielu użytkowników systemów RAID nie może efektywnie wykorzystywać dostępnej pojemności, ponieważ urządzenia muszą być na stałe podłączone do jednego lub kilku serwerów. Dlatego też duże przedsiębiorstwa zaczynają wdrażać w życie koncepcję **sieci obszarów składowania danych** (ang. *Storage Area Networks, SAN*). W przypadku takiego rozwiązania aktywne urządzenia peryferyjne są konfigurowane jako węzły szybkich sieci i mogą być podłączane i odłączane od serwerów w bardzo elastyczny sposób.

Pojawiło się kilka firm proponujących rozwiązania klasy SAN, które oferują własne rozwiązania topologiczne. Systemy składowania danych mogą być umieszczane w dużych odległościach od serwerów i zapewniają różne opcje wydajności i łączności. Istniejące aplikacje zarządzania pamięciami masowymi mogą być przenoszone do konfiguracji SAN przy wykorzystaniu sieci światłowodowych, które uwzględniają starszy protokół SCSI. W rezultacie urządzenia podłączane do sieci SAN funkcjonują jak urządzenia SCSI.

Obecne alternatywy architektoniczne dla techniki SAN to między innymi: połączenia bezpośrednie między serwerami a systemami składowania danych przy użyciu linii światłowodowych, użycie przełączników światłowodowych w celu łączenia wielu systemów klasy RAID, biblioteki taśmowe, użycie koncentratorów światłowodowych i przełączników w celu łączenia serwerów i systemów składowania danych w różnych konfiguracjach. Przedsiębiorstwa mogą stopniowo przechodzić od prostszych topologii do bardziej zaawansowanych, dodając serwery i urządzenia pamięci masowej w miarę potrzeb. Nie będziemy zagłębiać się w szczegóły, ponieważ różnią się one w przypadku różnych producentów. Najważniejsze korzyści, jakie się wymienia, to:

- Elastyczny charakter połączeń między serwerami i urządzeniami składowania danych przy użyciu przełączników i koncentratorów światłowodowych.
- Odległość między serwerem a systemem składowania danych w przypadku użycia odpowiednich przewodów światłowodowych, wynosząca do 10 km.
- Większe możliwości izolacyjne, pozwalające na dodawanie nowych urządzeń peryferyjnych i serwerów bez przerywania pracy.
- Szybka replikacja danych między wieloma systemami składowania danych. Typowe technologie wykorzystują replikację synchroniczną dla danych lokalnych i replikację asynchroniczną na potrzeby odzyskiwania danych po katastrofach.

Sieci SAN rozwijają się gwałtownie, ale wciąż jest z nimi związanych wiele problemów, takich jak łączenie mechanizmów składowania danych pochodzących od różnych producentów i obsługa rozwijających się programowych i sprzętowych standardów zarządzania danymi. Większość dużych przedsiębiorstw ocenia sieci SAN jako przydatną technikę przechowywania baz danych.

## 16.11.2. Technologia NAS

Z powodu niesamowitego wzrostu ilości danych cyfrowych, generowanych przede wszystkim przez multimedia i aplikacje korporacyjne, bardzo istotna stała się potrzeba wydajnych rozwiązań do składowania danych. Do tego celu służą m.in. urządzenia **NAS** (ang. *network-attached storage*). Są to serwery, które nie świadczą standardowych usług serwerowych, a jedynie umożliwiają stosowanie pamięci masowej na potrzeby **współużytkownia plików**. Urządzenia NAS umożliwiają dodawanie do sieci dużej ilości pamięci dyskowej i mogą udostępniać ją wielu serwerom bez konieczności zatrzymywania ich na czas konserwacji i aktualizacji. Takie urządzenia można umieścić w dowolnym miejscu sieci LAN i łączyć w różne konfiguracje. Jedno urządzenie, **stacja czołowa NAS** (ang. *NAS box* lub *NAS head*), pełni funkcję interfejsu między systemem NAS a klientami sieciowymi. Urządzenia NAS nie wymagają monitora, klawiatury ani myszy. Jeden dysk lub napęd taśmowy można podłączyć do wielu systemów NAS, aby zwiększyć ich łączną pojemność. Klienci łączą się ze stacją czołową NAS, a nie z poszczególnymi urządzeniami z pamięcią masową. System NAS może przechowywać dowolne dane mające postać plików — zawartość skrzynek e-mailowych, materiały ze stron WWW, kopie zapasowe zdalnych systemów itd. Pod tym względem urządzenia NAS zastępują tradycyjne serwery plików.

Systemy NAS mają zapewniać niezawodną pracę i łatwe administrowanie. Obejmują wbudowane mechanizmy, takie jak bezpieczne uwierzytelnianie lub automatyczne wysyłanie przez e-mail alertów w reakcji na błędy. Oferowane urządzenia NAS zapewniają wysoką

skalowalność, niezawodność, elastyczność i wydajność. Zwykle obsługują architekturę RAID na poziomach 0., 1. i 5. Tradycyjne sieci SAN różnią się od rozwiązań NAS pod kątem względami. Sieci SAN często używają światłowodów zamiast Ethernetu i obejmują wiele urządzeń sieciowych (*punktów końcowych*) w niezależnych, *prywatnych* sieciach LAN. Technologia NAS wykorzystuje pojedyncze urządzenia podłączone bezpośrednio do istniejącej publicznej sieci LAN. Serwery plików z systemami Windows, UNIX i NetWare wymagają obsługi konkretnych protokołów po stronie klienta, natomiast systemy NAS pozwalają klientom na większą niezależność, jeśli chodzi o stosowany system operacyjny. Podsumowując: systemy NAS zapewniają interfejs do systemu plików z obsługą plików sieciowych przy użyciu protokołów takich jak CIFS (ang. *common internet file system*) i NFS (ang. *network file system*).

### 16.11.3. iSCSI i inne sieciowe protokoły składowania danych

Niedawno zaproponowany został nowy protokół, **iSCSI** (ang. *Internet SCSI*). Jest to (podobnie jak SAN) blokowy protokół składowania danych. Umożliwia on klientom (*iniciatorom*) przesyłanie poleceń SCSI do urządzeń SCSI ze zdalnych lokalizacji. Główną zaletą iSCSI jest to, że nie wymaga specjalnego okablowania niezbędnego w technologii Fibre Channel i może działać na dłuższe dystanse, używając istniejącej infrastruktury sieciowej. Dzięki przesyłaniu poleceń SCSI w sieciach IP interfejs iSCSI ułatwia transfer danych między intranetami i zarządzanie danymi na długich dystansach. Możliwy jest transfer danych między sieciami LAN i WAN oraz w internecie.

Interfejs iSCSI działa w następujący sposób: gdy SZBD potrzebuje dostępu do danych, system operacyjny generuje odpowiednie polecenia SCSI i żądania danych przekazywane do procedur enkapsulacji i, w razie potrzeby, szyfrowania. Przed przesłaniem wynikowych pakietów IP połączeniem ethernetowym dodawany jest nagłówek pakietu. Otrzymany pakiet jest odszyfrowywany (jeśli przed transmisją został zaszyfrowany) i poddawany dezasemblacji, co powoduje oddzielenie poleceń SCSI od żądań. Polecenia SCSI trafiają przez kontroler SCSI do urządzenia pamięci masowej SCSI. Ponieważ interfejs iSCSI jest dwukierunkowy, protokół można wykorzystać także do zwracania danych w odpowiedzi na pierwotne żądanie. CISCO i IBM reklamują przełączniki i routery oparte na tej technologii.

**Pamięć masowa iSCSI** zyskała popularność przede wszystkim w małych i średnich firmach z powodu połączenia prostoty, niskich kosztów i funkcji urządzeń iSCSI. Te cechy pozwalają uniknąć dogłębnego poznawania technologii Fibre Channel i zamiast tego wykorzystać znajomość protokołu IP oraz sprzętu ethernetowego. Rozwiązania iSCSI w centrach danych bardzo dużych firm są wprowadzane powoli z powodu wcześniejszych inwestycji w sieci SAN wykorzystujące standard Fibre Channel.

Technologia iSCSI to jedna z dwóch podstawowych metod transmisji składowanych danych z użyciem sieci IP. Druga z tych metod, **FCIP** (ang. *Fibre Channel over IP*), przetwarza kody kontrolne i dane z technologii Fibre Channel na pakiety IP na potrzeby transmisji między geograficznie odległymi sieciami SAN opartymi na tej technologii. Ten protokół, nazywany *tunelowaniem Fibre Channel*, można stosować tylko w połączeniu z technologią Fibre Channel. Interfejs iSCSI może działać w istniejących sieciach ethernetowych.



Najnowszym pomysłem w obszarze składowania danych firmowych z użyciem IP jest **FCoE** (ang. *Fibre Channel over Ethernet*). Technologię tę można traktować jak iSCSI bez IP. Wykorzystywanych jest tu wiele elementów technologii SCSI i FC (podobnie jak w iSCSI), ale nie są uwzględniane komponenty TCP/IP. Technologia FCoE została z powodzeniem zastosowana w produktach firm CISCO (gdzie jest nazywana „Data Center Ethernet”) i Brocade. Wykorzystuje niezawodne technologie ethernetowe z buforowaniem i kontrolą przepływu między węzłami końcowymi w celu unikania utraty pakietów. Oferuje znakomitą wydajność (zwłaszcza przy korzystaniu z Ethernetu 10-gigabitowego — 10GbE), a producenci mogą ją stosunkowo łatwo dodawać do swoich urządzeń.

#### 16.11.4. Technologia Automated Storage Tiering

Innym trendem w składowaniu danych jest technologia Automated Storage Tiering (AST), która w zależności od potrzeb automatycznie przenosi dane między różnymi technologiami składowania takimi jak SATA, SAS i dyski SSD. Administrator może skonfigurować strategię warstwową, w której rzadziej używane dane są przenoszone na wolniejsze i tańsze dyski SATA, a częściej potrzebne informacje znajdują się na dyskach SSD (patrz tabela 16.1, gdzie opisano różne warstwy pamięci masowej uporządkowane rosnąco według szybkości dostępu). Ten zautomatyzowany podział na warstwy pozwala znacznie poprawić wydajność baz danych.

Firma EMC udostępnia implementację tej technologii nazywaną FAST (ang. *fully automated storage tiering*). Rozwiązanie to stale monitoruje aktywność danych i przenosi dane do odpowiednich warstw na podstawie stosowanej strategii.

#### 16.11.5. Obiektowa pamięć masowa

W ostatnich kilku latach pojawiły się ważne nowości związane z szybkim rozwojem chmury, architekturami rozproszonymi dla baz danych i analityki, a także rozwojem internetowych aplikacji intensywnie przetwarzających dane (patrz rozdziały 23., 24. i 25.). Te osiągnięcia doprowadziły do istotnych zmian w infrastrukturze składowania danych firm. Sprzętowe systemy oparte na plikach są przekształcane w nowe, otwarte architektury składowania danych. Najnowszym rozwiązaniem jest tu obiektowa pamięć masowa. W tym modelu dane są zarządzane w formie obiektów, a nie plików składających się z bloków. Obiekty obejmują metadane z właściwościami, które można wykorzystać do zarządzania tymi obiektami. Każdy obiekt ma unikatowy globalny identyfikator służący do lokalizowania obiektu. Początki obiektowej pamięci masowej sięgają projektów badawczych w CMU dotyczących skalowania pamięci NAS (Gibson i in., 1996) oraz systemu OceanStore z uczelni Berkeley, gdzie próbowano zbudować globalną infrastrukturę dla dowolnych zaufanych i niezaufanych serwerów w celu zapewnienia ciągłego dostępu do trwałych danych (Kubiawicz i in., 2000). W tym podejściu nie trzeba wykonywać niskopoziomowych operacji na pamięci związanych z zarządzaniem zasobami lub podejmowaniem decyzji o tym, którą architekturę RAID zastosować do ochrony przed błędami.

Pamięć obiektowa oferuje też dodatkową elastyczność związaną z interfejsami. Zapewnia kontrolę aplikacjom, które mogą zarządzać obiektami bezpośrednio, a także umożliwia adresowanie obiektów w ramach dużej przestrzeni nazw obejmującej wiele urządzeń. Obsługiwane są też replikacja i udostępnianie obiektów. Na ogólnym poziomie



pamięć obiektowa doskonale nadaje się do skalowalnego przechowywania dużych ilości nieustrukturyzowanych danych, np.: stron WWW, zdjęć czy nagrań audio i wideo. Polecenia dla urządzeń pamięci obiektowej (ang. *object-based storage device* — OSD) zostały zaproponowane jako część protokołu SCSI już dawno temu, ale do produktów komercyjnych trafiły dopiero wraz z wprowadzeniem przez firmę Seagate urządzeń OSD na platformie Kinetic Open Storage. Obecnie Facebook używa systemu obiektowej pamięci masowej do przechowywania zdjęć zajmujących 350 petabajtów. Spotify korzysta z systemu pamięci obiektowej do przechowywania utworów muzycznych. Omawiana technologia jest też używana do składowania danych w serwisie Dropbox. Pamięć obiektowa działa ponadto w wielu chmurach, np. w usługach AWS (ang. *Amazon Web Service*) S3 Amazona i Azure Microsoftu, gdzie pliki, relacje, wiadomości itd. są przechowywane w postaci obiektów. Inne przykładowe produkty z pamięcią obiektową to HCP firmy Hitachi, Atmos firmy EMC i RING firmy Scality. OpenStack Swift to otwarty projekt umożliwiający używanie żądań HTTP GET i PUT do pobierania i składowania obiektów (te żądania to w zasadzie cały interfejs API). Projekt OpenStack Swift został oparty na bardzo tanim sprzęcie, jest w pełni odporny na błędy, automatycznie wykorzystuje nadmiarowość w danej lokalizacji geograficznej i dzięki skalowaniu może obsługiwać bardzo dużą liczbę obiektów. Ponieważ pamięć obiektowa wymaga używania blokad na poziomie obiektów, nie jest jasne, jak dobrze nadaje się do przetwarzania współbieżnych transakcji w systemach transakcyjnych o wysokiej przepustowości. Dlatego wciąż nie jest uważana za odpowiednie rozwiązanie dla standardowych aplikacji bazodanowych używanych w firmach.

## 16.12. Podsumowanie

Na początku niniejszego rozdziału omówiono charakterystyki hierarchii pamięciowych, a później skoncentrowano się na drugorzędnych urządzeniach składowania danych. Skupiono się szczególnie na dyskach magnetycznych, ponieważ są one najczęściej używane w celu przechowywania aktywnych plików bazy danych. W tabeli 16.1 przedstawiono przegląd hierarchii rodzajów pamięci wraz z ich aktualną pojemnością, szybkością dostępu, szybkością transferu i cenami.

Dane na dysku są przechowywane w blokach. Uzyskiwanie dostępu do bloków na dysku jest kosztowną operacją ze względu na występowanie czasu wyszukiwania, opóźnienia rotacyjnego oraz czasu transferu bloku. Podwójne buforowanie można wykorzystywać wówczas, gdy uzyskuje się dostęp do kolejnych bloków na dysku; redukuje to średni czas dostępu. Inne parametry dysków omówiono w dodatku A. Przedstawiliśmy też listę strategii usprawniających dostęp do danych z dysków. Omówiliśmy szybko zyskujące popularność dyski SSD, a także napędy optyczne, używane głównie jako pamięć trzeciorzędna. Opisałyśmy działanie menedżera buforów, odpowiedzialnego za obsługę żądań danych, a także różne strategie zastępowania danych w buforach. Omówiono także różne sposoby przechowywania rekordów w pliku na dysku. Rekordy pliku są grupowane w bloki dyskowe i mogą mieć długość stałą lub zmienną, mogą być segmentowane lub niesegmentowane oraz mogą być tego samego typu lub różnych typów. Omówiono także nagłówki pliku, który opisuje formaty rekordów oraz przechowuje informacje o adresach na dysku bloków pliku. Informacje zawarte w nagłówku pliku są używane przez oprogramowanie systemowe uzyskujące dostęp do rekordów pliku.

Następnie przedstawiono zestaw typowych poleceń służących do uzyskiwania dostępu do rekordów pojedynczego pliku oraz omówiono pojęcie rekordu bieżącego. Opisano również, w jaki sposób złożone warunki wyszukiwania rekordów są zamieniane na warunki proste, których używa się w celu lokalizowania rekordów w pliku.

W dalszej kolejności omówiono trzy podstawowe metody organizacji plików: nieuporządkowaną, uporządkowaną i mieszącą. Pliki nieuporządkowane wymagają użycia wyszukiwania liniowego w celu lokalizowania rekordów, ale operacje wstawiania rekordów są bardzo proste. Omówiono również problem usuwania rekordów oraz użycie znaczników usunięcia.

Pliki uporządkowane zmniejszają czas wymagany do odczytania rekordów w porządku zgodnym z wartościami pola uporządkowania. Czas potrzebny do wyszukania dowolnego rekordu dla danej wartości jego pola klucza uporządkowania również zostaje zredukowany w razie użycia wyszukiwania binarnego. Jednakże konieczność zachowania uporządkowania rekordów sprawia, że operacje wstawiania są bardzo kosztowne. Dlatego też omówiono technikę wykorzystania nieuporządkowanego pliku przepełnienia, który redukuje koszty związane ze wstawianiem rekordów. Rekordy przepełnienia są okresowo scalane z plikiem głównym w czasie dokonywania reorganizacji pliku.

Mieszanie oferuje bardzo szybki dostęp do dowolnego rekordu w pliku dla danej wartości jego klucza mieszącego. Najlepszą metodą mieszania zewnętrznego jest technika pakietów, gdzie jeden lub więcej kolejnych bloków odpowiada jednemu pakietowi. Do radzenia sobie z kolizjami, które powodują przepełnienia pakietów, stosuje się adresowanie otwarte, łączenie i mieszanie wielokrotne. Dostęp do pól innych niż mieszące jest powolny, podobnie jak dostęp do rekordów z uporządkowaniem według dowolnego pola. Następnie omówiono trzy techniki mieszące dla plików, które w dynamiczny sposób zwiększają i zmniejszą swój rozmiar — było to mieszanie rozszerzalne, dynamiczne oraz liniowe. W pierwszych dwóch do organizowania katalogu stosuje się wyższe bity z adresu mieszącego. Mieszanie liniowe ma utrzymywać współczynnik zapełnienia pliku w określonym zakresie, a nowe pakiety są wtedy dodawane liniowo.

Omówiono również pokrótce inne podstawowe sposoby organizacji plików, takie jak B-drzewa i pliki rekordów mieszących, które implementują fizycznie jako element struktur składowania związki między rekordami różnych typów. Wreszcie omówiono najnowsze dokonania w technikach produkcji dysków takie jak mechanizm *RAID* (ang. *Redundant Arrays of Inexpensive [Independent] Disks*), który w dużych firmach stał się standardową techniką zwiększania niezawodności i odporności na błędy w pamięci masowej. Na koniec opisaliśmy wybrane nowe trendy w systemach składowania danych przedsiębiorstw: sieci SAN, technologię NAS, iSCSI i inne protokoły sieciowe, technologię AST i obiektową pamięć masową, odgrywającą istotną rolę w architekturach składowania danych w centrach danych oferujących usługi oparte na chmurze.

## Pytania powtórkowe

- 16.1. Na czym polega różnica między podstawowymi a drugorzędnymi metodami składowania danych?
- 16.2. Dlaczego dyski, a nie taśmy, są używane w celu przechowywania aktywnych plików baz danych?

- 16.3. Zdefiniuj następujące pojęcia: *dysk, pakiet dysków, ścieżka, blok, cylinder, sektor, szczelina międzyblokowa, głowica czytająco-zapisująca*.
- 16.4. Omów proces inicjalizacji dysku.
- 16.5. Omów mechanizm używany w celu odczytywania lub zapisywania danych na dysku.
- 16.6. Jakie są komponenty adresu bloku dyskowego?
- 16.7. Dlaczego uzyskiwanie dostępu do bloku dyskowego jest kosztowną operacją? Omów składowe czasu związanego z uzyskaniem dostępu do bloku dyskowego.
- 16.8. W jaki sposób podwójne buforowanie polepsza czas dostępu do bloków dyskowych?
- 16.9. Jakie względy mogą przemawiać za wykorzystywaniem rekordów o zmiennej długości? Jakie rodzaje znaków separatora są potrzebne?
- 16.10. Omów techniki przydzielania bloków plików na dysku.
- 16.11. Na czym polega różnica między metodą organizacji pliku a metodą dostępu?
- 16.12. Na czym polega różnica między plikami statycznymi a dynamicznymi?
- 16.13. Jakie są typowe operacje jednorekordowe związane z uzyskiwaniem dostępu do pliku? Które z nich zależą od bieżącego rekordu w pliku?
- 16.14. Omów techniki usuwania rekordów.
- 16.15. Omów korzyści i wady związane z użyciem (a) pliku nieuporządkowanego, (b) pliku uporządkowanego oraz (c) statycznego pliku mieszającego z pakietami i łączeniem. Które operacje mogą być wykonywane wydajnie w przypadku każdej z tych metod organizacji, a które są kosztowne?
- 16.16. Omów techniki pozwalające, aby plik mieszający rozszerzał się i skracał dynamicznie. Jakie są zalety i wady każdej z nich?
- 16.17. Opisz różnice między katalogami w mieszaniu rozszerzalnym i dynamicznym.
- 16.18. W jakim celu są używane pliki mieszane? Jakie są inne rodzaje podstawowych metod organizacji plików?
- 16.19. Opisz niedopasowanie technologii procesorów i dysków.
- 16.20. Jakie są główne cele technologii RAID? W jaki sposób są one osiągnane?
- 16.21. W jaki sposób metoda dublowania dysków zwiększa niezawodność? Podaj adekwatny przykład.
- 16.22. Co cechuje poszczególne poziomy organizacji RAID?
- 16.23. Opisz najważniejsze cechy popularnych poziomów architektury RAID: 0., 1. i 5.
- 16.24. Czym są sieci SAN? W jaki sposób zwiększają elastyczność i jakie korzyści oferują?
- 16.25. Opisz główne cechy technologii NAS jako narzędzia składowania danych w przedsiębiorstwach.
- 16.26. W jaki sposób nowe systemy iSCSI ułatwiły stosowanie sieci SAN?
- 16.27. Czym są protokoły SATA, SAS i FC?
- 16.28. Czym są dyski SSD i jakie oferują zalety w porównaniu z dyskami HDD?
- 16.29. Do czego służy menedżer buforów? W jaki sposób obsługuje żądania danych?

- 16.30. Wymień często stosowane strategie zastępowania danych w buforach.
- 16.31. Czym są optyczne i taśmowe urządzenia automatycznej zmiany dysków? Wymień różne typy nośników optycznych obsługiwanych przez napędy optyczne.
- 16.32. Czym jest technologia AST? Dlaczego jest ona przydatna?
- 16.33. Czym jest obiektowa pamięć masowa? Pod jakimi względami jest ona lepsza od tradycyjnych systemów składowania danych?

## Ćwiczenia

- 16.34. Rozważ dysk o następujących charakterystykach (nie są to parametry żadnego istniejącego urządzenia): rozmiar bloku  $B = 512$  bajtów, rozmiar szczeliny międzyblokowej  $G = 128$  bajtów, liczba bloków na ścieżkę = 20, liczba ścieżek na powierzchnię = 400. Pakiet dysków składa się z 15 dwustronnych dysków.
- Ile wynosi pojemność całkowita ścieżki i ile wynosi pojemność użytkowa (bez szczelin międzyblokowych)?
  - Ile występuje cylindrów?
  - Ile wynosi pojemność całkowita i użyteczna cylindra?
  - Ile wynosi pojemność całkowita i użyteczna pakietu dysków?
  - Założmy, że napęd dysku obraca pakiet dysków z prędkością 2400 obr/m (obrotów na minutę). Ile wynosi szybkość transmisji ( $tr$ ) w bajtach/ms oraz czas transmisji bloku ( $btt$ ) w ms? Ile wynosi średnie opóźnienie rotacyjne ( $rd$ ) w ms? Jaka jest szybkość zbiorczej transmisji danych (patrz dodatek B)?
  - Założmy, że średni czas wyszukiwania wynosi 30 ms. Ile czasu w ms (średnio) zajmuje zlokalizowanie i przesłanie pojedynczego bloku o podanym adresie?
  - Oblicz średni czas, jaki zajmuje przesłanie 20 losowych bloków i porównaj go z czasem wymaganym do przesłania 20 kolejnych bloków przy użyciu podwójnego buforowania w celu zaoszczędzenia na czasie wyszukiwania i opóźnieniach rotacyjnych.
- 16.35. Plik posiada  $r = 20\,000$  rekordów STUDENT o stałej długości. Każdy rekord posiada następujące pola: IMIĘNAZW (30 bajtów), PESEL (11 bajtów), ADRES (40 bajtów), TELEFON (9 bajtów), DATAUR (8 bajtów), PŁEĆ (1 bajt), KODWYDZKIERUNKU (4 bajty), KODWYDZDRKIERUNKU (4 bajty), KODGRUPY (4 bajty, liczba całkowita) oraz CHARAKTERST (3 bajty). Dodatkowy bajt jest używany jako znacznik usunięcia. Plik jest przechowywany na dysku, którego parametry określono w ćwiczeniu 16.34.
- Oblicz rozmiar rekordu  $R$  w bajtach.
  - Oblicz współczynnik blokowy  $bfr$  oraz liczbę bloków pliku  $b$  zakładając istnienie organizacji niesegmentowej.
  - Oblicz średni czas potrzebny do znalezienia rekordu przy użyciu wyszukiwania liniowego na pliku, jeżeli (i) bloki pliku są przechowywane w sposób ciągły i używane jest podwójne buforowanie, (ii) bloki pliku nie są przechowywane w sposób ciągły.
  - Założmy, że plik jest uporządkowany według wartości PESEL. Oblicz czas wymagany do wyszukania rekordu o danej wartości pola PESEL przy użyciu wyszukiwania binarnego.

- 16.36. Załóżmy, że tylko 80 procent rekordów STUDENT z ćwiczenia 16.35 posiada wartość dla pola TELEFON, 85 procent dla pola KODWYDZKIERUNKU, 15 procent dla pola KODWYDZDRKIERUNKU oraz 90 procent dla pola CHARKTERST. Załóżmy również, że wykorzystujemy rekordy o zmiennej długości. Każdy rekord posiada 1-bajtowe *pole typu* dla każdego pola w rekordzie oraz 1-bajtowy znacznik usunięcia i 1-bajtowy znacznik końca rekordu. Załóżmy, że używamy *segmentowanej* organizacji rekordów, gdzie każdy blok posiada 5-bajtowy wskaźnik na kolejny blok (przestrzeń ta nie jest wykorzystywana do przechowywania danych).
- Oblicz średni rozmiar rekordu  $R$  w bajtach.
  - Oblicz liczbę bloków potrzebnych do przechowania pliku.
- 16.37. Załóżmy, że jednostka dyskowa posiada następujące parametry: czas wyszukiwania  $s = 20$  ms, opóźnienie rotacyjne  $rd = 10$  ms, czas transmisji bloku  $btt = 1$  ms, rozmiar bloku  $B = 2\,400$  bajtów, rozmiar szczeliny międzyblokowej  $G = 600$  bajtów. Plik PRACOWNIK zawiera następujące pola: PESEL (11 bajtów), NAZWISKO (20 bajtów), IMIĘ (20 bajtów), INICJAŁDRIM (1 bajt), DATAUR (10 bajtów), ADRES (35 bajtów), TELEFON (12 bajtów), PESELPRZEŁOŻ (11 bajtów), DZIAŁ (4 bajty), KODSTANOWISKA (4 bajty), *znacznik usunięcia* — 1 bajt. Plik PRACOWNIK posiada  $r = 30\,000$  rekordów o stałej długości i blokach niesegmentowanych. Zapisz odpowiednie wzory oraz oblicz poniższe wartości dla opisanego pliku PRACOWNIK.
- Rozmiar rekordu  $R$  (z uwzględnieniem znacznika usunięcia), współczynnik blokowy  $bfr$  oraz liczbę bloków dyskowych  $b$ .
  - Oblicz przestrzeń w każdym bloku dyskowym marnowaną ze względu na organizację niesegmentową.
  - Oblicz prędkość transmisji  $tr$  oraz zbiorczą prędkość transmisji  $btr$  dla tej jednostki dyskowej (definicje powyższych pojęć zawarto w dodatku B).
  - Oblicz średnią *liczbę operacji dostępu do bloków* potrzebną w celu wyszukania dowolnego rekordu w pliku za pomocą wyszukiwania liniowego.
  - Oblicz w ms średni *czas* wymagany do wyszukania dowolnego rekordu w pliku za pomocą wyszukiwania liniowego, jeżeli bloki pliku są przechowywane w kolejnych blokach dyskowych i używane jest podwójne buforowanie.
  - Oblicz w ms średni *czas* wymagany do wyszukania dowolnego rekordu w pliku za pomocą wyszukiwania liniowego, jeżeli bloki pliku *nie* są przechowywane w kolejnych blokach dysku.
  - Załącz, że rekordy są uporządkowane według pewnego pola klucza. Oblicz średnią *liczbę operacji dostępu do bloków* oraz *średni czas* wymagany do wyszukania dowolnego rekordu w pliku przy użyciu wyszukiwania binarnego.
- 16.38. Plik CZĘŚCI, w którym pole CZĘŚĆ# jest kluczem mieszającym, zawiera rekordy o następujących wartościach tego pola: 2369, 3760, 4692, 4871, 5659, 1821, 1074, 7115, 1620, 2428, 3943, 4750, 6975, 4981, 9208. Plik używa ośmiu pakietów ponumerowanych od 0 do 7. Każdy pakiet zajmuje jeden blok dyskowy i zawiera dwa rekordy. Załaduj te rekordy do pliku w zadanym porządku, używając funkcji mieszającej  $h(K) = K \bmod 8$ . Oblicz średnią liczbę operacji uzyskiwania dostępu do bloków dla losowego pobrania dla wartości CZĘŚĆ#.

- 16.39. Załaduj rekordy z ćwiczenia 16.38 do rozszerzalnego pliku mieszającego bazującego na metodzie mieszania rozszerzalnego. Przedstaw strukturę katalogu w każdym etapie oraz głębokości globalne i lokalne. Wykorzystaj funkcję mieszającą  $h(K) = K \bmod 128$ .
- 16.40. Załaduj rekordy z ćwiczenia 16.38 do rozszerzalnego pliku mieszającego, używając mieszania liniowego. Rozpocznij od pojedynczego bloku dyskowego, używając funkcji mieszającej  $h_0 = K \bmod 2^0$ , i pokaż, jak rośnie plik oraz w jaki sposób zmieniają się funkcje mieszające w miarę wstawiania rekordów. Załóż, że bloki są rozdzielane, kiedy wystąpi przepełnienie, i przedstaw wartość  $n$  w każdym etapie.
- 16.41. Porównaj polecenia plikowe wymienione w podrozdziale 16.5 z dostępnymi w przypadku metody dostępu do plików, którą znasz.
- 16.42. Załóżmy, że posiadamy nieuporządkowany plik rekordów o stałej długości, który używa niesegmentowanej organizacji rekordów. Opracuj algorytmy wstawiania, usuwania i modyfikowania rekordów pliku. Określ wszelkie założenia, jakie przyjmujesz.
- 16.43. Załóżmy, że posiadamy uporządkowany plik rekordów o stałej długości oraz nieuporządkowany plik przepełnienia służący do obsługi operacji wstawiania. Oba pliki wykorzystują rekordy niesegmentowane. Opracuj algorytmy wstawiania, usuwania i modyfikowania rekordów pliku, a także mechanizm reorganizacji pliku. Określ wszelkie założenia, jakie przyjmujesz.
- 16.44. Czy potrafisz określić jakieś techniki inne niż wykorzystanie nieuporządkowanego pliku przepełnienia, które mogłyby być stosowane w celu zwiększenia wydajności operacji wstawiania w plikach uporządkowanych?
- 16.45. Załóżmy, że posiadamy plik mieszający z rekordami o stałym rozmiarze oraz że przepełnienia są obsługiwane przez mechanizm łączenia. Opracuj algorytmy wstawiania, usuwania i modyfikowania rekordów pliku. Określ wszelkie założenia, jakie przyjmujesz.
- 16.46. Czy potrafisz określić jakieś techniki inne niż mechanizm łączenia, które mogłyby obsłużyć przypadek przepełnienia pakietu w mieszanii zewnętrznym?
- 16.47. Zapisz pseudokod algorytmów wstawiania w przypadku mieszania liniowego oraz mieszania rozszerzalnego.
- 16.48. Napisz program służący do uzyskiwania dostępu do poszczególnych pól rekordów w poniższych okolicznościach. W każdym przypadku określ założenia, jakie przyjmujesz, dotyczące wskaźników, znaków separatora itd. Określ rodzaj informacji wymaganych w nagłówku pliku w celu zapewnienia ogólności kodu w każdym przypadku.
- a) Rekordy o stałej długości z blokami niesegmentowanymi.
  - b) Rekordy o stałej długości z blokami segmentowanymi.
  - c) Rekordy o zmiennej długości z polami o zmiennej długości i blokami segmentowanymi.
  - d) Rekordy o zmiennej długości z grupami powtarzalnymi i blokami segmentowanymi.



- e) Rekordy o zmiennej długości z polami opcjonalnymi i blokami segmentowanymi.
  - f) Rekordy o zmiennej długości dopuszczające przypadki z punktów c, d i e.
- 16.49. Załóżmy, że początkowo plik zawiera  $r = 120\,000$  rekordów o rozmiarze  $R = 200$  bajtów w pliku nieuporządkowanym (stertowym). Rozmiar bloku  $B = 2400$  bajtów, średni czas wyszukiwania  $s = 16$  ms, średnie opóźnienie rotacyjne  $rd = 8,3$  ms oraz czas transmisji bloku  $btt = 0,8$  ms. Załóżmy, że na każde 2 wstawiane rekordy 1 rekord jest usuwany, dopóki całkowita liczba aktywnych rekordów nie osiągnie wartości 240 000.
- a) Ile operacji transmisji bloków potrzeba w celu zreorganizowania pliku?
  - b) Ile czasu zajmuje znalezienie rekordu tuż przed dokonaniem reorganizacji?
  - c) Ile czasu zajmuje znalezienie rekordu tuż po dokonaniu reorganizacji?
- 16.50. Załóżmy, że mamy plik sekwencyjny (uporządkowany) 100 000 rekordów, gdzie każdy rekord zajmuje 240 bajtów. Załóżmy, że  $B = 2400$  bajtów,  $s = 16$  ms,  $rd = 8,3$  ms oraz  $btt = 0,8$  ms. Załóżmy wreszcie, że chcemy wykonać  $X$  niezależnych odczytów losowych rekordów z tego pliku. Możemy wykonać  $X$  odrębnych odczytów bloków lub jeden pełny odczyt całego pliku w poszukiwaniu tych  $X$  rekordów. Problem polega na zdecydowaniu, kiedy bardziej wydajnym rozwiązaniem będzie wykonanie pełnego odczytu całego pliku, a nie wykonanie  $X$  losowych odczytów bloków. To znaczy, jaka jest wartość  $X$ , kiedy pełny odczyt pliku jest wydajniejszy od wykonania  $X$  losowych odczytów? Wynik należy podać w postaci funkcji zmiennej  $X$ .
- 16.51. Załóżmy, że statyczny plik mieszający zawiera początkowo 600 pakietów w obszarze podstawowym oraz że wstawiane są rekordy tworzące obszar przepełnienia 600 pakietów. Jeżeli zreorganizujemy plik mieszający, możemy przyjąć, że przepełnienie zostanie wyeliminowane. Jeżeli koszt reorganizacji pliku jest kosztem przesyłania pakietów (odczytu i zapisu wszystkich pakietów) i jedyną okresowo wykonywaną operacją jest operacja pobierania, to ile razy musielibyśmy wykonać pobieranie (z powodzeniem) w celu zapewnienia, że reorganizacja będzie uzasadniona? To znaczy, kiedy koszt reorganizacji oraz późniejsze koszty wyszukiwania są niższe od kosztów wyszukiwania przed reorganizacją? Odpowiedź uzasadnij. Przyjmij wartości  $s = 16$  ms,  $rd = 8,3$  ms,  $btt = 1$  ms.
- 16.52. Załóżmy, że chcemy utworzyć liniową funkcję mieszającą ze współczynnikiem wypełnienia pliku 0,7 oraz współczynnikiem blokowym 20 rekordów na pakiet, co sprowadza się do przechowania początkowo 112 000 rekordów.
- a) Ile pakietów należy przydzielić w obszarze głównym?
  - b) Ile powinna wynosić liczba bitów używanych jako adresy pakietów?

## Wybrane publikacje

Pozycja Wiederholda (1987) zawiera szczegółowe omówienie i analizę drugorzędnych urządzeń składowania danych oraz organizacji plików. Dyski optyczne opisano w pozycji Berga i Rotha (1989) oraz poddano analizie w pracy Forda i Christodoulakisa (1991).



Pamięci flash omówiono w pozycji Dipperta i Levy'ego (1993). Ruemmler i Wilkes (1994) prezentują porównanie technologii dysków magnetycznych. Większość podręczników poświęconych bazom danych zawiera omówienie prezentowanego tutaj materiału. Większość podręczników poświęconych strukturom danych, w tym pozycja Knutha (1998), bardziej szczegółowo omawia mieszanie statyczne. W pozycji Knutha można znaleźć pełne omówienie funkcji mieszających oraz technik rozstrzygania kolizji, jak również porównania wydajności ich działania. Knuth oferuje również szczegółowe omówienie technik sortowania plików zewnętrznych. Do podręczników poświęconych strukturom plików należą pozycje: Claybrooka (1992), Smitha i Barnes (1987) oraz Salzberga (1988). Omówiono w nich dodatkowe organizacje plików, w tym pliki o strukturze drzewiastej, oraz przedstawiono szczegółowe algorytmy działań na plikach. Salzberg i in. (1990) opisuje algorytm rozproszonego sortowania zewnętrznego. Organizacje plików o dużym poziomie odporności na błędy opisano w pozycjach Bittona i Graya (1988) oraz Graya i in. (1990). Przeplatanie dysków zaproponowano w pracy Salema i Garcia Moliny (1986). Pierwsza praca na temat nadmiarowych macierzy niedrogich dysków (RAID) to Patterson i in. (1988). Chen i Patterson (1990) oraz doskonałe omówienie technologii RAID w pracy Chena i in. (1994) to dodatkowe źródła informacji. Grochowski i Hoyt (1996) omawiają przyszłe trendy w zakresie rozwoju napędów dyskowych. Różnorodne wzory związane z architekturą RAID można znaleźć w pracy Chena i in. (1994).

Praca Morrisa (1968) to jedno z pierwszych dzieł poświęconych mieszaniu. Mieszanie rozszerzalne opisano w pracy Fagina i in. (1979). Mieszanie liniowe przedstawiono w pozycji Litwina (1980). Algorytmy wstawiania i usuwania przy zastosowaniu mieszania liniowego omówiono z przykładami w pracy Salzberga (1988). Mieszanie dynamiczne, które omówiliśmy tu pokrótce, zaproponował Larson (1978). Istnieje wiele odmian mieszania rozszerzalnego i liniowego — patrz na przykład prace Cesariniego i Sody (1991), Du i Tonga (1991) oraz Hachema i Berry (1992).

Gibson i in. (1997) opisali skalowanie systemów NAS z użyciem serwerów plików, a Kubiawicz i in. (2000) przedstawili system OceanStore, służący do tworzenia globalnej infrastruktury narzędziowej do składowania trwałych danych. Były to pionierskie rozwiązania, stanowiące źródło pomysłów związanych z obiektową pamięcią masową. Taką pamięć opisali Mesnier i in. (2003). System Lustre (Braam i Schwan, 2002) był jednym z pierwszych produktów z obiektową pamięcią masową. Wykorzystano go w większości superkomputerów, w tym w dwóch najszybszych — w chińskim Tianhe-2 i w Titanie z Oakridge National Lab.

Szczegóły dotyczące urządzeń pamięci masowej można znaleźć na witrynach internetowych producentów, np.: <http://www.seagate.com>, <http://www.ibm.com>, <http://www.emc.com>, <http://www.hp.com>, <http://www.storagetek.com>. Firma IBM prowadzi centrum badawcze technik składowania danych w IBM Almaden (<http://www.almaden.ibm.com>). Inne przydatne witryny to strony dotyczące składowania danych w serwisach: CISCO (<http://www.cisco.com>), Network Appliance (NetApp) — <http://www.netapp.com>, Hitachi Data Storage (HDS) — <http://www.hds.com> i SNIA (Storage Networking Industry Association) — <http://www.snia.org>. We wspomnianych witrynach znajdziesz wiele prac z omawianego obszaru.

# Struktury indeksowe dla plików i fizyczne projekty baz danych

W niniejszym rozdziale zakładamy, że istnieją już pliki o pewnej podstawowej metodzie organizacji, takie jak pliki nieuporządkowane, uporządkowane lub mieszające opisane w rozdziale 16. Zostaną tu opisane dodatkowe pomocnicze **struktury dostępowe** (ang. *access structures*), określane mianem **indeksów** (ang. *indexes*), które są używane w celu przyspieszenia pobierania rekordów na podstawie określonych warunków. Struktury indeksowe zwykle oferują **drugorzędne ścieżki dostępu** (ang. *secondary access paths*), które zapewniają alternatywne sposoby uzyskiwania dostępu do rekordów bez wpływania na fizyczne rozmieszczenie głównego pliku z danymi na dysku. Umożliwiają one wydajny dostęp do rekordów w oparciu o **polą indeksujące** (ang. *indexing fields*), których używa się w celu konstruowania indeksów. Zasadniczo w celu utworzenia indeksu można użyć *dowolnego pola* w pliku i dla jednego pliku można skonstruować *wiele indeksów* na różnych polach (a także indeksów opartych na *wielu polach*). Istnieje wiele rodzajów indeksów, z których każdy wykorzystuje określoną strukturę danych w celu przyspieszenia wyszukiwania. W celu znalezienia rekordu lub rekordów w pliku w oparciu o określony warunek selekcji na polu indeksującym, należy najpierw uzyskać dostęp do indeksu, który wskazuje na jeden lub więcej bloków w pliku, gdzie znajdują się poszukiwane rekordy. Najczęściej używane rodzaje indeksów bazują na plikach uporządkowanych (indeksy jednopoziomowe) oraz strukturach drzewiastych (indeksy wielopoziomowe, B<sup>+</sup>-drzewa). Indeksy mogą być również konstruowane w oparciu o techniki mieszania lub inne struktury wyszukiwania danych. Omówimy też indeksy w postaci wektorów bitów — tzw. *indeksy bitmapowe*.

W podrozdziale 17.1 zostaną omówione różne typy jednopoziomowych indeksów uporządkowanych — podstawowe, drugorzędne i klastrowania. Traktując indeks jednopoziomowy jako plik uporządkowany, można utworzyć dla niego dodatkowe indeksy, co pozwala mówić o indeksach wielopoziomowych. Popularny schemat indeksowania o nazwie **ISAM** (ang. *Indexed Sequential Access Method*, indeksowa metoda dostępu sekwencyjnego) jest oparty właśnie na takim podejściu. Indeksy wielopoziomowe stanowią temat podrozdziału 17.2. W podrozdziale 17.3 zostaną opisane B-drzewa oraz B<sup>+</sup>-drzewa, które są strukturami danych powszechnie używanymi w systemach SZBD w celu implementowania dynamicznie zmienianych indeksów wielopoziomowych. B<sup>+</sup>-drzewa stały się powszechnie akceptowaną domyślną strukturą generowania indeksów na żądanie w większości relacyjnych systemów SZBD. Podrozdział 17.4 zostanie poświęcony alternatywnym metodom uzyskiwania dostępu do danych w oparciu o połączenie wielu kluczy. Z kolei w podrozdziale 17.5 zostaną omówione indeksy mieszające; zostanie również ogólnie przedstawione pojęcie indeksów logicznych, które tworzą dodatkowy poziom pośredni w stosunku do indeksów fizycznych, co pozwala zapewnić elastyczność i rozszerzalność

indeksu fizycznego pod względem jego organizacji. W podrozdziale 17.6 opiszemy indeksowanie z użyciem wielu kluczy i indeksy bitmapowe służące do wyszukiwania z użyciem jednego lub więcej kluczy. W podrozdziale 17.7 przedstawimy projektowanie fizyczne. Podrozdział 17.8 stanowi podsumowanie całego rozdziału.

## 17.1. Rodzaje jednopoziomowych indeksów uporządkowanych

Idea użycia indeksu uporządkowanego przypomina używanie indeksów książkowych, które zawierają listy ważnych pojęć zestawionych w porządku alfabetycznym wraz z listami numerów stron, na których odpowiednie pojęcia występują w książce. Możemy przeszukać indeks w celu znalezienia listy *adresów* — w tym przypadku numerów stron — i użyć tych adresów do zlokalizowania pojęcia w książce *przeszukując* określone strony. Alternatywnym rozwiązaniem, w przypadku, gdy nie zostaną podane żadne wskazówki, jest powolne przeglądanie całej książki słowo po słowie w celu znalezienia interesującego nas pojęcia. Odpowiada to wykonaniu *przeszukiwania liniowego* na pliku. Oczywiście, większość książek posiada także dodatkowe informacje, na przykład tytuły rozdziałów lub podrozdziałów, które mogą pomóc w znalezieniu danego pojęcia bez konieczności przeszukiwania całej książki. Jednakże indeks stanowi jedyne dokładne określenie, w którym miejscu książki znajduje się każde pojęcie.

W przypadku pliku o danej strukturze rekordów składających się z kilku pól (lub atrybutów), strukturę dostępową indeksu definiuje się zwykle dla pojedynczego pola pliku, noszącego nazwę **pola indeksującego (atrybutu indeksującego)**<sup>1</sup>. Indeks zazwyczaj przechowuje każdą wartość pola indeksu wraz z listą wskaźników na wszystkie bloki dysku zawierające rekordy z daną wartością pola. Wartości w indeksie są *uporządkowane*, dzięki czemu możemy wykonywać na nim *wyszukiwanie binarne*. Jeśli uporządkowane są zarówno plik danych, jak i plik indeksu, to plik indeksu jest zwykle znacznie mniejszy od pliku danych, więc jego przeszukiwanie binarne stanowi wydajną operację. Wielopoziomowe indeksy drzewiaste (patrz podrozdział 17.2) stanowią rozszerzenie wyszukiwania binarnego i zmniejszają przestrzeń wyszukiwania dzięki podziałowi na dwie części na każdym etapie; istnieją też techniki podziału na  $n$  części, co powoduje podział przestrzeni wyszukiwania na  $n$  porcji w każdym kroku.

Istnieje kilka rodzajów indeksów uporządkowanych. **Indeks główny** (ang. *primary index*) jest określany na *polu klucza uporządkowania uporządkowanego pliku* rekordów. W podrozdziale 16.7 stwierdzono, że pole klucza uporządkowania jest używane w celu *fizycznego uporządkowania* rekordów pliku na dysku i każdy rekord posiada *unikatową wartość* dla tego pola. Jeżeli pole uporządkowania nie jest polem klucza (to znaczy, jeżeli wiele rekordów w pliku może posiadać tę samą wartość dla pola uporządkowania), można wykorzystać kolejny rodzaj indeksu, nazywany **indeksem klastrowania** (ang. *clustering index*). Plik danych jest wtedy nazywany **plikiem klastrowanym**. Należy zauważyć, że plik może posiadać najwyżej jedno pole uporządkowania fizycznego, a stąd może

---

<sup>1</sup> W niniejszym rozdziale pojęć *pole* i *atrybut* będziemy używać zamiennie.

posiadać najwyżej jeden indeks główny lub jeden indeks klastrowania, ale *nie oba*. Trzeci rodzaj indeksu, noszący nazwę **indeksu drugorzędnego** (ang. *secondary index*), można określić na dowolnym polu pliku *nie będącym polem uporządkowania*. Plik danych może posiadać kilka indeksów drugorzędnych oprócz zdefiniowanej głównej metody dostępu. W kolejnych trzech punktach zostaną omówione wymienione rodzaje indeksów jednopozimowych.

### 17.1.1. Indeksy główne

**Indeks główny** to plik uporządkowany z rekordami o stałej długości posiadającymi dwa pola. Pełni on funkcję struktury dostępowej umożliwiającej wydajne wyszukiwanie rekordów danych w pliku i dostęp do nich. Pierwsze z tych pól ma taki sam typ danych jak pole klucza uporządkowania — nazywanego **kluczem głównym** — w pliku danych, zaś drugie pole jest wskaźnikiem na blok dyskowy (adres bloku). Na każdy *blok* w pliku danych przypada jeden **wpis indeksu** (ang. *index entry*), inaczej **rekord indeksu** (ang. *index record*) w pliku indeksu. Każdy wpis indeksu posiada wartość pola klucza głównego dla *pierwszego* rekordu w bloku oraz wskaźnik na ten blok. Oba pola wpisu indeksu będziemy oznaczać jako  $\langle K(i), P(i) \rangle$ . W dalszej części rozdziału różne rodzaje **wpisów** indeksu  $\langle K(i), X \rangle$  będą miały następujące znaczenie:

- $X$  może być adresem fizycznym bloku (lub strony) w pliku (tak jak w  $P(i)$ ).
- $X$  może być adresem rekordu składającym się z adresu bloku i identyfikatora (lub pozycji) rekordu w bloku.
- $X$  może być adresem logicznym bloku lub rekordu w pliku. Jest to względna wartość odwzorowywana na adres fizyczny (patrz dalsze wyjaśnienia w punkcie 17.6.1).

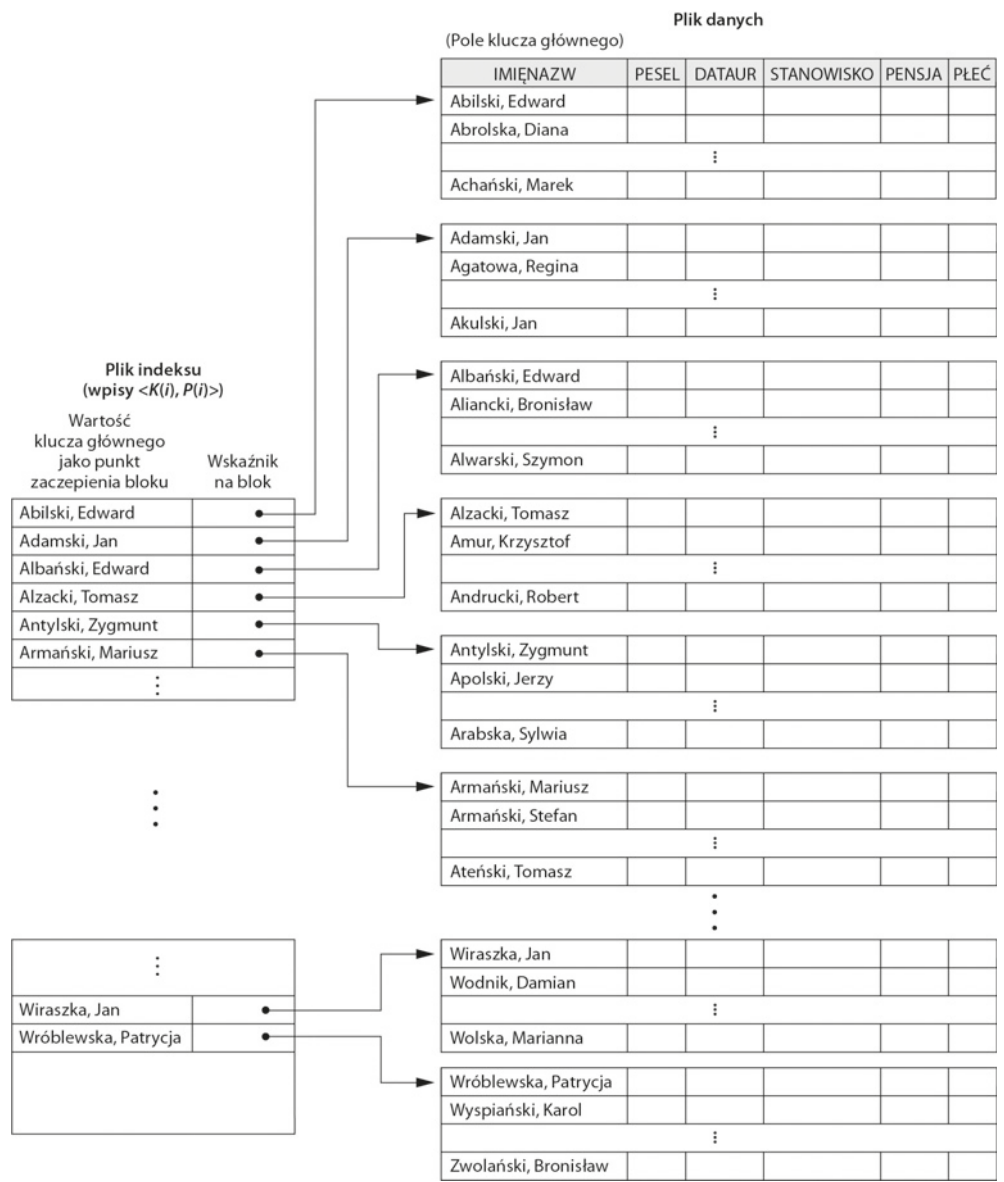
W celu utworzenia indeksu głównego na pliku uporządkowanym przedstawionym na rysunku 16.7 używamy jako klucza głównego pola IMIĘNAZW, ponieważ jest to pole klucza uporządkowania pliku (zakładając, że każda wartość pola IMIĘNAZW jest unikatowa). Każdy wpis w indeksie zawiera wartość pola IMIĘNAZW oraz wskaźnik. Pierwsze trzy wpisy indeksu mają następującą postać:

```
<K(1) = (Abilski, Edward), P(1) = adres bloku 1.>
<K(2) = (Adamski, Jan), P(2) = adres bloku 2.>
<K(3) = (Albański, Edward), P(3) = adres bloku 3.>
```

Na rysunku 17.1 przedstawiono taki indeks główny. Całkowita liczba wpisów w indeksie jest taka sama jak *liczba bloków dyskowych* w uporządkowanym pliku danych. Pierwszy rekord w każdym bloku pliku danych nosi nazwę **rekordu zaczepienia** (ang. *anchor record*) takiego bloku lub po prostu **zaczepienia bloku** (ang. *block anchor*)<sup>2</sup>.

Indeksy można również klasyfikować jako zagęszczone lub rzadkie. **Indeks zagęszczony** (ang. *dense index*) zawiera wpis dla *każdej wartości klucza wyszukiwania* (a stąd dla każdego rekordu) znajdującej się w pliku danych. Z kolei **indeks rzadki** (ang. *sparse index*), inaczej **niezagęszczony** (ang. *nondense*), posiada wpis tylko dla niektórych wartości

<sup>2</sup> Można użyć schematu podobnego do opisanego powyżej, gdzie ostatni rekord w każdym bloku (a nie pierwszy) służyłby jako zaczepienie bloku. Pozwala to nieco zwiększyć wydajność działania algorytmu wyszukiwania.



RYSUNEK 17.1. Indeks główny na polu klucza uporządkowania pliku przedstawionego na rysunku 16.7

wyszukiwania. Indeks rzadki obejmuje więc mniej elementów niż jest rekordów w pliku. Dlatego też indeks główny jest indeksem niezagęszczonym (rzadkim), gdyż zawiera wpis dla każdego bloku dyskowego pliku danych oraz klucza jego rekordu zaczepienia, a nie dla każdej wartości wyszukiwania (czyli dla każdego rekordu)<sup>3</sup>.

<sup>3</sup> Rzadki indeks główny w niektórych książkach i artykułach jest nazywany (głównym) indeksem klastrowanym.

Plik indeksu w przypadku indeksu głównego wymaga znacznie mniejszej liczby bloków niż plik danych. Dzieje się tak z dwóch powodów. Po pierwsze, występuje w nim *mniej wpisów* niż istnieje rekordów w pliku danych. Po drugie, każdy wpis indeksu ma zazwyczaj *mniejszy rozmiar* niż rekord danych, ponieważ posiada tylko dwa pola, a każde z nich jest krótkie. W konsekwencji w jednym bloku może się pomieścić więcej wpisów indeksu niż rekordów danych. Dlatego też wyszukiwanie binarne wykonywane na pliku indeksu wymaga uzyskania dostępu do mniejszej liczby bloków niż ma to miejsce w przypadku przeszukiwania binarnego pliku danych. Odwołując się do tabeli 16.3 można zauważyć, że wyszukiwanie binarne w przypadku uporządkowanego pliku danych wymaga  $\log_2 b$  operacji dostępu do bloków. Jednak jeśli plik indeksu głównego zawiera  $b_i$  bloków, wówczas zlokalizowanie rekordu poprzez wartość klucza wyszukiwania wymaga przeszukania binarnego takiego indeksu oraz uzyskania dostępu do bloku zawierającego dany rekord: w sumie daje to  $\log_2 b_i + 1$  operacji dostępu.

Rekord, którego wartość klucza głównego wynosi  $K$ , znajduje się w bloku, którego adresem jest  $P(i)$ , gdzie  $K(i) \leq K < K(i+1)$ .  $i$ -ty blok w pliku danych zawiera same takie rekordy ze względu na fizyczne uporządkowanie rekordów pliku względem pola klucza głównego. W celu pobrania rekordu dla danej wartości  $K$  jego klucza głównego, wykonujemy przeszukiwanie binarne na pliku indeksu w celu znalezienia odpowiedniego wpisu  $i$ , a następnie pobieramy blok pliku danych, którego adresem jest  $P(i)$ <sup>4</sup>. Przykład 1. ilustruje oszczędności pod względem uzyskiwania dostępu do bloków osiągnięte w razie użycia indeksu głównego w celu wyszukania rekordu.

**PRZYKŁAD 1:** Załóżmy, że mamy plik uporządkowany liczący  $r = 300\,000$  rekordów, przechowywany na dysku o rozmiarze bloku  $B = 4096$  bajtów<sup>5</sup>. Rekordy pliku mają stały rozmiar i są niesegmentowane, zaś długość rekordu wynosi  $R = 100$  bajtów. Współczynnik blokowy dla pliku wynosi  $bfr = \lfloor (B/R) \rfloor = \lfloor (4096/100) \rfloor = 40$  rekordów na blok. Liczba bloków wymaganych dla pliku wynosi  $b = \lceil (r/bfr) \rceil = \lceil (300000/40) \rceil = 7500$  bloków. Przeszukiwanie binarne pliku danych wymaga w przybliżeniu  $\lceil \log_2 b \rceil = \lceil \log_2 7500 \rceil = 13$  operacji dostępu do bloków.

Założmy dodatkowo, że pole klucza uporządkowania w pliku ma długość  $V = 9$  bajtów, wskaźnik na blok ma 6 bajtów długości i dla pliku utworzono indeks główny. Rozmiar każdego wpisu indeksu wynosi  $R_i = (9+6) = 15$  bajtów, więc współczynnik blokowy dla indeksu ma wartość  $bfr_i = \lfloor (B/R_i) \rfloor = \lfloor (4096/15) \rfloor = 273$  wpisy na blok. Całkowita liczba wpisów indeksu  $r_i$  jest równa liczbie bloków zawartych w pliku danych, czyli 7500. Liczba bloków indeksu wynosi więc  $b_i = \lceil (r_i/bfr_i) \rceil = \lceil (7500/273) \rceil = 28$  bloków. W celu wykonania przeszukiwania binarnego na pliku indeksu wymagane jest uzyskanie  $\lceil \log_2 b_i \rceil = \lceil \log_2 28 \rceil = 5$  dostępu do bloków. W celu znalezienia rekordu przy użyciu indeksu musimy uwzględnić dodatkową operację dostępu do bloku pliku danych, co daje w sumie  $5+1 = 6$  operacji dostępu do bloków. Stanowi to usprawnienie w porównaniu do przeszukiwania binarnego pliku danych, gdzie wymagane było 13 operacji. Warto zauważyć,

<sup>4</sup> Należy zauważyć, że powyższy wzór nie byłby poprawny, gdyby plik danych był uporządkowany względem pola nie będącego kluczem. W takim przypadku ta sama wartość indeksu w zaczepieniu bloku mogłaby zostać powtórzona w ostatnich rekordach poprzedniego bloku.

<sup>5</sup> Większość producentów SZBD, w tym Oracle, używa bloków (stron) o standardowej wielkości 4096 bajtów.



że indeks obejmujący 7500 wpisów po 15 bajtów każdy jest dość mały (112 500 bajtów, czyli 112,5 kilobajta) i zwykle będzie przechowywany w pamięci głównej, dlatego przeglądanie go za pomocą wyszukiwania binarnego odbywa się błyskawicznie. W takiej sytuacji pobranie rekordu wymaga tylko jednego dostępu do bloku.

Główny problem związany z indeksami głównymi, który dotyczy wszystkich plików uporządkowanych, to wstawianie i usuwanie rekordów. W przypadku indeksu głównego problem ten jest złożony, ponieważ jeżeli spróbujemy wstawić rekord na jego odpowiedniej pozycji w pliku danych, nie tylko musimy przenieść rekordy w celu zapewnienia wolnej przestrzeni dla nowego rekordu, ale również zmienić niektóre wpisy indeksu, gdyż przesunięcie rekordów powoduje zmiany *rekordów zaczepienia* niektórych bloków. Użycie nieuporządkowanego pliku przepełnienia, zgodnie z opisem z podrozdziału 16.7, pozwala na zniwelowanie do pewnego stopnia tego problemu. Inna możliwość polega na użyciu listy jednokierunkowej rekordów przepełnienia dla każdego bloku w pliku danych. Taka metoda przypomina obsługę rekordów przepełnienia opisaną w podrozdziale 16.8.2 przy okazji omawiania technik mieszania. Rekordy w każdym bloku i jego liście jednokierunkowej przepełnienia można posortować w celu skrócenia czasu pobierania. Kasowanie rekordów jest obsługiwane przy użyciu znaczników usunięcia.

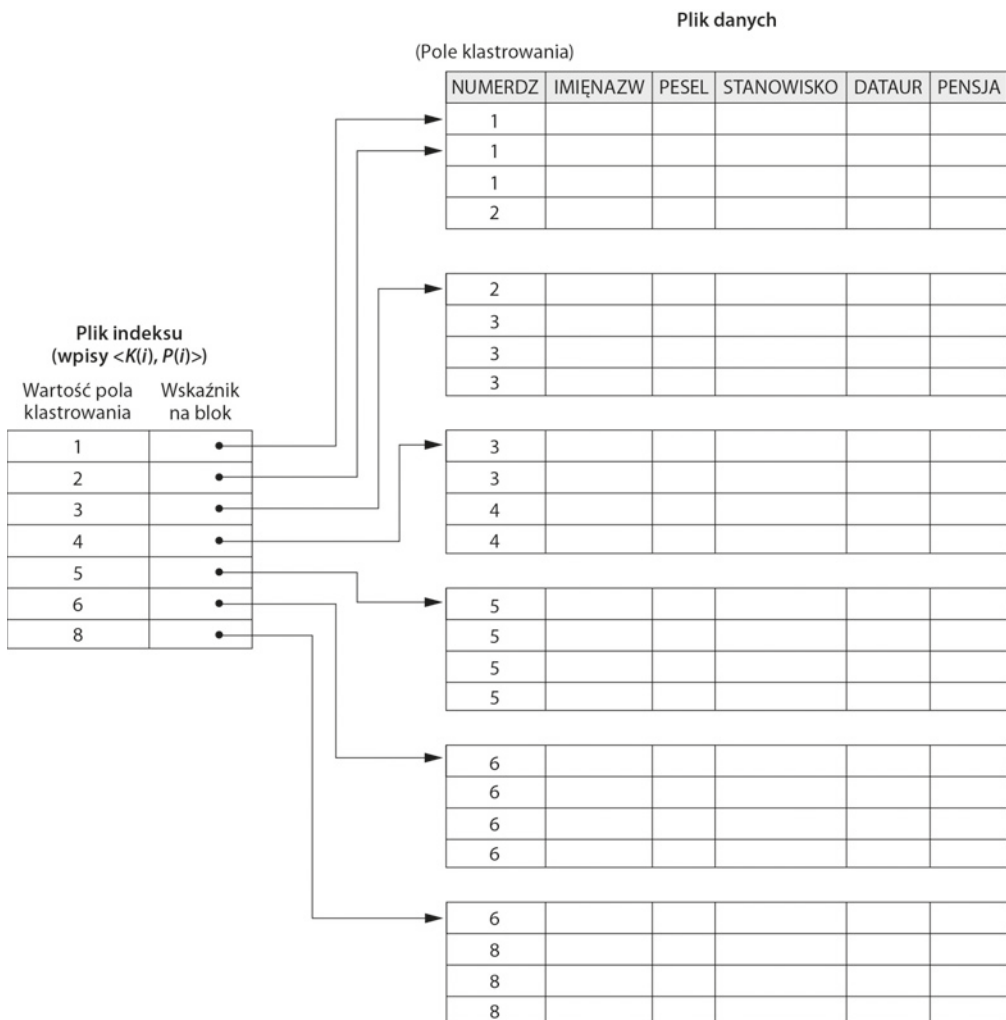
## 17.1.2. Indeksy klastrowania

Jeżeli rekordy pliku są fizycznie posortowane według pola niebędącego polem klucza (takiego, które *nie* posiada niepowtarzalnej wartości dla każdego rekordu), takie pole określa się mianem **pola klastrowania** (ang. *clustering field*), a plik danych — mianem **pliku klastrowanego**. Można utworzyć inny rodzaj indeksu, nazywany **indeksem klastrowania** (ang. *clustering index*), w celu przyspieszenia pobierania rekordów, które posiadają tę samą wartość dla pola klastrowania. Indeks klastrowania różni się od indeksu głównego, który wymaga, aby pola uporządkowania pliku danych posiadały *niepowtarzalne wartości* dla każdego rekordu.

Indeks klastrowania również stanowi plik uporządkowany o dwóch polach. Pierwsze pole ma ten sam typ, co pole klastrowania pliku danych, zaś drugie pole jest wskaźnikiem na blok. Indeks klastrowania zawiera jeden wpis dla każdej *odrębnej wartości* pola klastrowania i składa się z wartości oraz wskaźnika na *pierwszy blok* w pliku danych, który zawiera rekord o danej wartości pola klastrowania. Na rysunku 17.2 przedstawiono adekwatny przykład. Należy zauważyć, że wstawianie i usuwanie rekordów wciąż sprawia kłopoty, ponieważ rekordy danych są fizycznie uporządkowane. W celu obejścia problemu wstawiania zazwyczaj rezerwuje się cały blok (lub klastre kolejnych bloków) dla *każdej wartości* pola klastrowania. Wszystkie rekordy o danej wartości są umieszczane w takim bloku (lub klastrze bloków). Sprawia to, że wstawianie i usuwanie rekordów staje się dość proste. Odpowiedni schemat postępowania przedstawiono na rysunku 17.3.

Indeks klastrowania stanowi kolejny przykład indeksu *niezagęszczanego*, ponieważ posiada on wpis dla każdej *odrębnej wartości* pola indeksowania, które z definicji nie jest kluczem, a stąd zwykle posiada zduplikowane wartości dla rekordów w pliku.

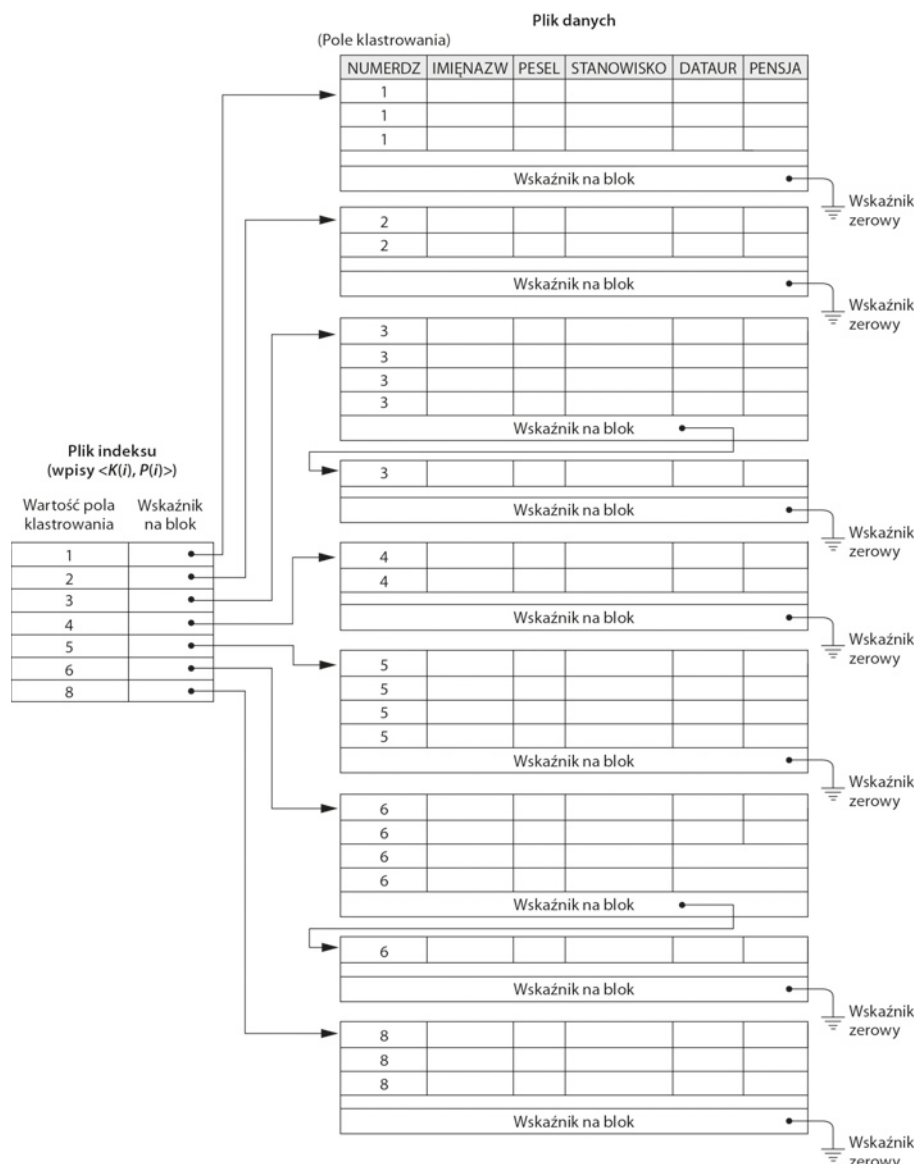




RYСУNEK 17.2. Indeks klastrowania na polu uporządkowania NUMERDZ niebędącym kluczem pliku PRACOWNIK

**PRZYKŁAD 2:** Załóżmy, że używany jest ten sam plik uporządkowany zawierający  $r = 300\,000$  rekordów i przechowywany na dysku z blokami o wielkości  $B = 4096$  bajtów. Przyjmijmy, że plik jest uporządkowany na podstawie pola KOD, a w pliku występuje 1000 kodów pocztowych (co daje średnią 300 rekordów na jeden kod, jeśli założymy równomierny rozkład rekordów względem kodów). Indeks obejmuje tu 1000 wpisów po 11 bajtów każdy (5-bajtowy kod i 6-bajtowy wskaźnik do bloku). Współczynnik blokowy ma wartość  $bfr_i = \lfloor (B/R_i) \rfloor = \lfloor (4096/11) \rfloor = 372$  wpisy na blok. Liczba bloków indeksu wynosi więc  $b_i = \lceil (r_i/bfr_i) \rceil = \lceil (1000/373) \rceil = 3$  bloki. Przeszukiwanie binarne na pliku indeksu wymaga zatem  $\lceil \log_2 b_i \rceil = \lceil \log_2 3 \rceil = 2$  dostępow do bloków. Także ten indeks zwykle zostanie wczytany do pamięci głównej (ponieważ zajmuje 11 000 bajtów, czyli 11 kilobajtów), a jego przeszukiwanie w tej pamięci będzie się odbywać błyskawicznie. Jeden dostęp do bloku pliku danych doprowadzi do pierwszego rekordu z określonym kodem pocztowym.

Istnieje pewne podobieństwo między rysunkami 17.1, 17.2 i 17.3 a rysunkami 16.11 i 16.12. Indeks przypomina nieco strukturę katalogu używanego w przypadku mieszania dynamicznego, opisanego w podrozdziale 16.8.3, i strukturę katalogu używanego w mieszaniu rozszerzalnym. Obie te struktury są przeszukiwane w celu znalezienia wskaźnika na blok danych zawierający odpowiedni rekord. Główna różnica polega na tym, że przeszukiwanie indeksu polega na wykorzystaniu wartości samego pola wyszukiwania, zaś w przypadku przeszukiwania katalogu mieszania wykorzystywana jest wartość skrótu obliczana poprzez zastosowanie funkcji mieszającej względem pola wyszukiwania.



RYSUNEK 17.3. Indeks klastrowania z oddzielnym klastrem bloków dla każdej grupy rekordów, które posiadają tę samą wartość pola klastrowania

### 17.1.3. Indeksy drugorzędne

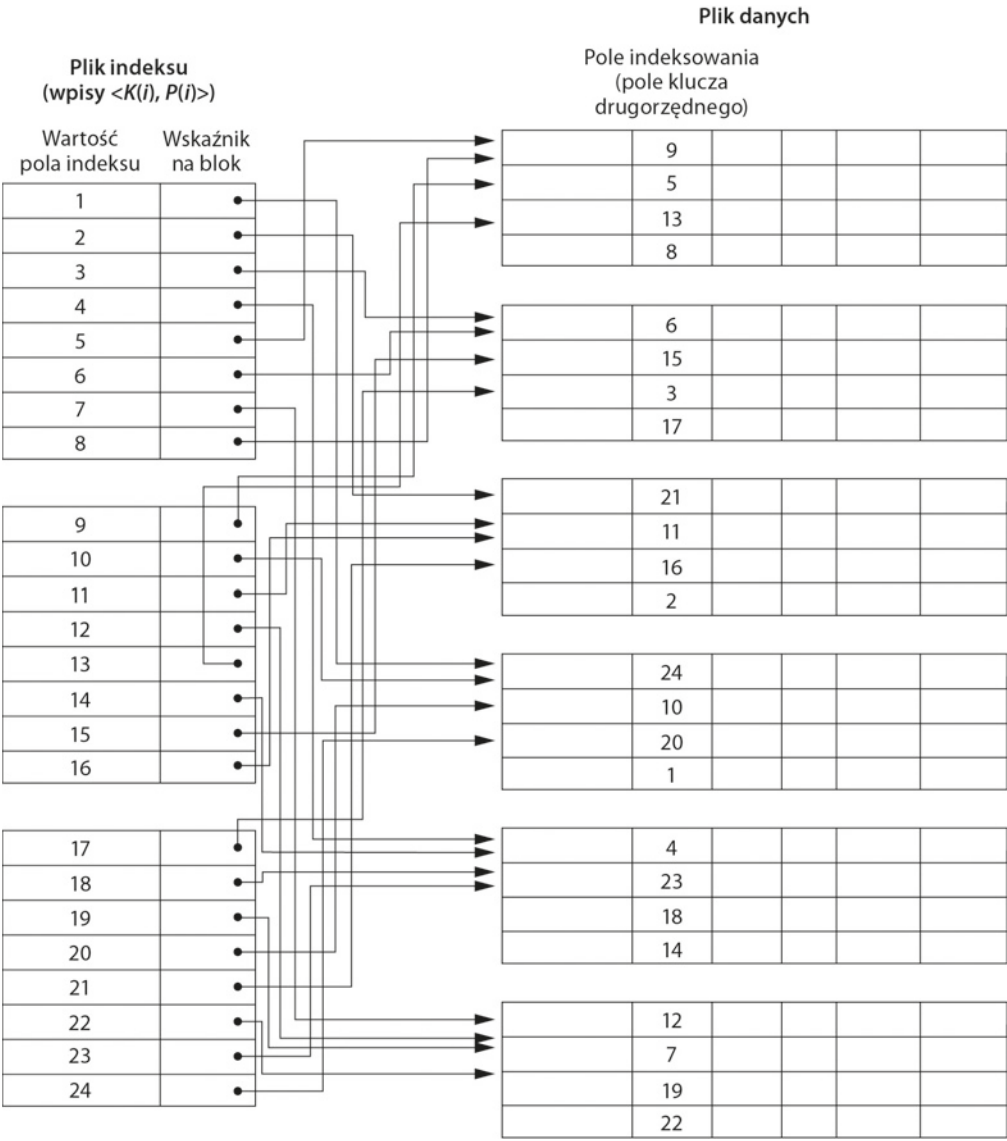
**Indeks drugorzędny** oferuje dodatkowy mechanizm uzyskiwania dostępu do pliku, dla którego określono już mechanizm podstawowy. Rekordy w pliku danych mogą być uporządkowane, nieuporządkowane lub poddane mieszanii. Indeks drugorzędny można utworzyć na polu, które jest kluczem kandydującym i posiada unikatową wartość w każdym rekordzie, lub które nie jest polem klucza i posiada duplikaty wartości. Indeks jest plikiem uporządkowanym o dwóch polach. Pierwsze pole ma ten sam typ danych co pewne pole *niebędące polem uporządkowania* pliku danych, które jest **połem indeksującym** (ang. *indexing field*). Drugie pole jest albo wskaźnikiem na *blok*, albo wskaźnikiem na *rekord*. Może istnieć *wiele* indeksów drugorzędnych (a stąd również pól indeksowania) dla jednego pliku. Każdy z nich reprezentuje dodatkowy sposób dostępu do danego pliku na podstawie określonego pola.

Najpierw rozważymy strukturę dostępową indeksu drugorzędnego na polu klucza (unikatowego), które posiada *odrębną wartość* dla każdego rekordu. Takie pole określa się czasem mianem **klucza drugorzędnego** (ang. *secondary key*). W modelu relacyjnym jego odpowiednikiem jest atrybut klucza unikatowego lub atrybut klucza głównego tabeli. W takim przypadku dla *każdego rekordu* w pliku danych występuje jeden wpis w indeksie, który zawiera wartość pola dla danego rekordu oraz wskaźnik albo na blok, w którym występuje ten rekord, albo na sam rekord. Stąd taki indeks jest **zagęszczony**.

Ponownie do dwóch pól wpisu indeksu będziemy się odwoływać jako  $\langle K(i), P(i) \rangle$ . Wpisy są **uporządkowane** według wartości  $K(i)$ , więc możemy stosować wyszukiwanie binarne. Ze względu na fakt, że rekordy pliku danych *nie* są fizycznie uporządkowane na podstawie wartości pola klucza drugorzędnego, *nie możemy* użyć mechanizmu zaczepień bloków. Z tego względu wpis indeksu jest tworzony dla każdego rekordu w pliku danych, a nie tylko dla każdego bloku, jak ma to miejsce w przypadku indeksów głównych. Na rysunku 17.4 przedstawiono indeks drugorzędny, w którym wskaźniki  $P(i)$  we wpisach indeksu są *wskaźnikami na bloki*, a nie wskaźnikami na rekordy. Kiedy odpowiedni blok zostanie przeniesiony do pamięci głównej, można przeprowadzić wyszukiwanie potrzebnego rekordu w danym bloku.

Indeks drugorzędny zwykle wymaga większej powierzchni składowania i wiąże się z dłuższymi czasami przeszukiwania niż indeks główny ze względu na zawartą w nim dużą liczbę wpisów. Jednakże *usprawnienie* wyszukiwania dowolnego rekordu jest w przypadku indeksu drugorzędnego znacznie większe, gdyż w razie braku indeksu drugorzędnego zachodzi potrzeba wykonywania na pliku danych *wyszukiwania liniowego*. W przypadku indeksu głównego wciąż można stosować wyszukiwanie binarne na pliku głównym, nawet jeżeli indeks nie istnieje. Przykład 3. ilustruje osiągnięte korzyści pod względem liczby bloków, do których uzyskuje się dostęp.

**PRZYKŁAD 3:** Weźmy pod uwagę plik z przykładu 1. dla  $r = 300\,000$  rekordów o stałym rozmiarze wynoszącym  $R = 100$  bajtów, przechowywanych na dysku o rozmiarze bloku  $B = 4096$  bajtów. Plik posiada  $b = 7500$  bloków, zgodnie z obliczeniami z przykładu 1. Załóżmy, że chcemy szukać rekordów o konkretnej wartości klucza drugorzędnego na polu niebędącym kluczem uporządkowania i mającym długość  $V = 9$  bajtów. Bez indeksu drugorzędnego liniowe przeszukiwanie pliku wymaga średnio  $b/2 = 7500/2 = 3750$ ostępów do bloku. Załóżmy, że tworzymy indeks drugorzędny na polu *niebędącym kluczem uporządkowania*.



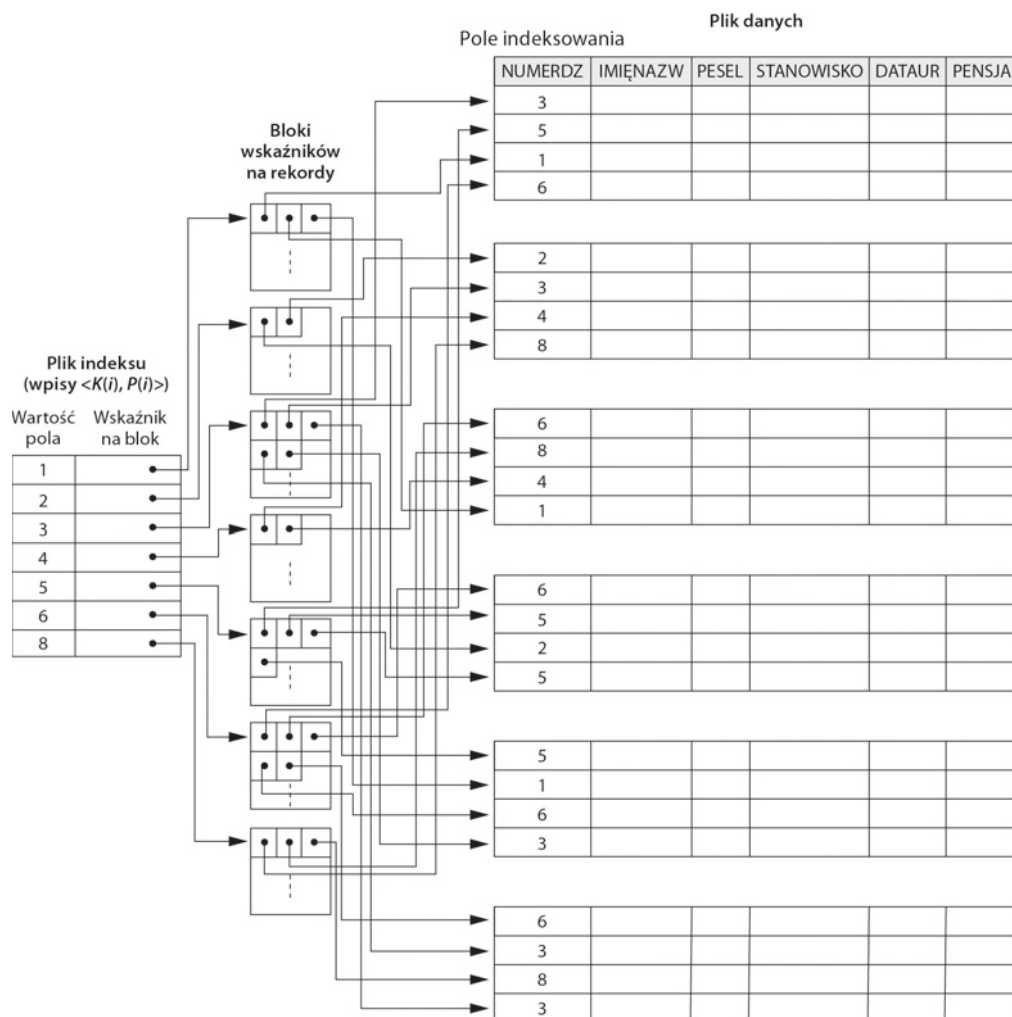
RYSUNEK 17.4. Zagęszczony indeks drugorzędny (ze wskaźnikami na bloki) na polu niebędącym polem klucza pliku

Podobnie jak w przykładzie 1. wskaźnik na blok  $P$  ma 6 bajtów długości, więc każdy wpis indeksu liczy  $R_i = (9+6) = 15$  bajtów długości, a współczynnik blokowy dla indeksu wynosi  $bfr_i = \lfloor (B/R_i) \rfloor = \lfloor (4096/15) \rfloor = 273$  wpisy na blok. W przypadku takiego zagęszczonego indeksu drugorzędnego całkowita liczba wpisów w indeksie  $r_i$  jest równa *liczbie rekordów* w pliku danych, która wynosi 300000. Liczba bloków wymaganych dla indeksu wynosi więc  $b_i = \lceil (r_i/bfr_i) \rceil = \lceil (300000/273) \rceil = 1099$  bloków.

Przeszukiwanie binarne takiego indeksu drugorzędnego wymaga  $\lceil \log_2 b_i \rceil = \lceil \log_2 1099 \rceil = 11$  operacji dostępu do bloków. W celu znalezienia rekordu przy użyciu indeksu musimy wykonać dodatkową operację dostępu do pliku danych, co daje w sumie  $11+1 = 12$  operacji dostępu do bloków. Stanowi to znaczne usprawnienie w porównaniu z 3750 operacjami dostępu wymaganymi średnio w przypadku wyszukiwania liniowego, aczkolwiek jest to wynik nieco gorszy od wymagającego wykonania sześciu operacji dostępu indeksu głównego. Różnica wynika z tego, że indeks główny był rzadki i dlatego krótszy — miał tylko 28 bloków długości w porównaniu z 1099 blokami indeksu zagęszczonego.

Indeks drugorzędny można również utworzyć na polu *niebędącym polem klucza* pliku. W takim przypadku wiele rekordów w pliku danych może posiadać tę samą wartość dla pola indeksującego. Istnieje kilka opcji implementacji takiego indeksu.

- Pierwsze rozwiązanie polega na uwzględnieniu wielu wpisów indeksu o tej samej wartości  $K(i)$  — jednym dla każdego rekordu. Taki indeks jest zagęszczony.
- drugie rozwiązanie polega na określeniu rekordów o zmiennej długości dla wpisów indeksu oraz z polem powtarzalnym dla wskaźników. We wpisie indeksu dla  $K(i)$  przechowujemy listę wskaźników  $\langle P(i, 1), \dots, P(i, k) \rangle$  — jeden wskaźnik przypada na każdy blok zawierający rekord, którego wartość pola indeksującego jest równa  $K(i)$ . W przypadku rozwiązania pierwszego i drugiego należy odpowiednio zmodyfikować algorytm przeszukiwania binarnego, aby uwzględnić zmienną liczbę wpisów w indeksie na wartość klucza.
- Trzecie rozwiązanie, używane najczęściej, polega na zachowaniu stałej długości wpisów indeksu i określeniu jednego wpisu dla każdej wartości pola indeksującego oraz utworzeniu *dodatkowego poziomu pośredniego* w celu obsłużenia wielu wskaźników. W przypadku takiego schematu niezagęszczonego wskaźnik  $P(i)$  we wpisie indeksu  $\langle K(i), P(i) \rangle$  wskazuje na *blok wskaźników na rekordy*. Każdy wskaźnik rekordu w takim bloku wskazuje na jeden z rekordów pliku danych o wartości  $K(i)$  pola indeksującego. Jeżeli pewna wartość  $K(i)$  występuje w zbyt wielu rekordach i jej wskaźniki na rekordy nie mieszczą się w jednym bloku dyskowym, można użyć klastra lub listy bloków. Zastosowanie takiej techniki przedstawiono na rysunku 17.5. Pobieranie danych poprzez indeks wymaga jednej lub dwóch dodatkowych operacji uzyskania dostępu do bloków, ponieważ występuje dodatkowy poziom, jednak algorytmy przeszukiwania indeksu oraz (co ważniejsze) wstawiania nowych rekordów do pliku danych są bardzo proste. Algorytm wyszukiwania binarnego można zastosować bezpośrednio do pliku indeksu, ponieważ ten ostatni jest uporządkowany. Na potrzeby pobierania przedziałów, np. rekordów spełniających warunek  $V_1 \leq K \leq V_2$ , można wykorzystać wskaźniki do bloków zamiast wskaźników do rekordów. Należy wyznaczyć sumę pul wskaźników do bloków odpowiadającą wpisom z indeksu od  $V_1$  do  $V_2$ , aby wyeliminować duplikaty, po czym możliwy będzie dostęp do wynikowych bloków. Ponadto operacje pobierania danych oparte na skomplikowanych warunkach selekcji mogą być obsługiwane poprzez odwoływanie się do wskaźników na rekordy bez potrzeby pobierania wielu niepotrzebnych rekordów pliku (patrz ćwiczenie 17.24).



RYSUNEK 17.5. Indeks drugorzędny (ze wskaźnikami na rekordy) na polu nie będącym polem klucza pliku, który zaimplementowano przy użyciu jednego poziomu pośredniego, tak aby wpisy indeksu posiadały stałą długość oraz unikatowe wartości pola

Należy zauważyć, że indeks drugorzędny zapewnia **uporządkowanie logiczne** (ang. *logical ordering*) rekordów względem pola indeksującego. Jeżeli będziemy chcieli uzyskać dostęp do tych rekordów w kolejności występowania wpisów w indeksie drugorzędnym, pobierzemy je według wartości pola indeksującego. Przy stosowaniu indeksów głównych i klastrowania zakładamy, że to samo pole używane jest do **fizycznego porządkowania** rekordów w pliku i do indeksowania.

### 17.1.4. Podsumowanie

W celu podsumowania niniejszego podrozdziału poniżej przedstawiono dwie tabele. Tabela 17.1 przedstawia charakterystyki pola indeksującego każdego rodzaju omawianych indeksów jednopoziomowych — głównych, klastrowania i drugorzędnych. W tabeli 17.2 podsumowano właściwości każdego rodzaju indeksu poprzez porównanie liczby wpisów indeksu i określenie, które indeksy są zagęszczone oraz które wykorzystują zaczepienia bloków z pliku danych.

TABELA 17.1. Rodzaje indeksów w oparciu o właściwości pola indeksującego

	Pole indeksu używane w celu uporządkowania pliku	Pole indeksu nieużywane w celu uporządkowania pliku
Pole indeksujące jest kluczem	Indeks główny	Indeks drugorzędny (pole klucza)
Pole indeksujące nie jest kluczem	Indeks klastrowania	Indeks drugorzędny (pole nie klucza)

TABELA 17.2. Właściwości rodzajów indeksów

Rodzaj indeksu	Liczba wpisów w indeksie (na pierwszym poziomie)	Zagęszczony lub niezagęszczony (rzadki)	Zaczepienia bloków z pliku danych
Główny	Liczba bloków w pliku danych	Niezagęszczony	Tak
Klastrowania	Liczba odrębnych wartości pola indeksu	Niezagęszczony	Tak/nie <sup>a</sup>
Drugorzędny (pole klucza)	Liczba rekordów w pliku danych	Zagęszczony	Nie
Drugorzędny (pole nie klucza)	Liczba rekordów <sup>b</sup> lub liczba odrębnych wartości pola indeksu <sup>c</sup>	Zagęszczony lub niezagęszczony	Nie

<sup>a</sup> Tak, jeżeli każda odrębna wartość pola uporządkowania rozpoczyna nowy blok; w przeciwnym razie — nie.

<sup>b</sup> Dla rozwiązania pierwszego.

<sup>c</sup> Dla rozwiązania drugiego i trzeciego.

## 17.2. Indeksy wielopoziomowe

Schematy indeksowania omówione powyżej wykorzystują uporządkowany plik indeksu. W celu zlokalizowania wskaźników na blok dyskowy lub rekord (rekordy) w pliku zawierający określoną wartość pola indeksującego stosuje się przeszukiwanie binarne. Przeszukiwanie binarne wymaga około  $(\log_2 b_i)$  operacji dostępu do bloków dla indeksu liczącego  $b_i$  bloków, ponieważ każdy etap algorytmu dwukrotnie redukuje ilość przestrzeni przeszukiwania. Właśnie z tego względu we wzorze stosujemy funkcję logarytmiczną o podstawie 2. Idea działania **indeksu wielopoziomowego** (ang. *multilevel index*) polega na zredukowaniu części indeksu, który przeszukujemy, o współczynnik  $bfr_i$ , czyli współczynnik blokowy indeksu, który jest większy od 2. Stąd przestrzeń przeszukiwania jest zmniejszana znacznie szybciej. Wartość współczynnika  $bfr_i$  określa się mianem **zwielokrotnienia wyjściowego** (ang. *fan-out*) indeksu wielopoziomowego i będziemy się do niego odwoływać za pomocą symbolu **fo**. W wyszukiwaniu binarnym *przestrzeń wyszukiwania rekordów* jest na każdym etapie zmniejszana o połowę; indeksy wielopoziomowe powodują, że w każdym



kroku przestrzeń jest dzielona na  $n$  części (gdzie  $n$  = zwielokrotnienie wyjściowe). Przeszukiwanie indeksu wielopoziomowego wymaga około  $(\log_{fo} b_i)$  operacji uzyskiwania dostępu do bloków, co — kiedy zwielokrotnienie wyjściowe ma wartość większą od 2 — stanowi wartość mniejszą niż w przypadku przeszukiwania binarnego. W większości sytuacji zwielokrotnienie wyjściowe jest znacznie wyższe niż 2. Gdy wielkość bloku wynosi 4096 bajtów (co jest najczęściej spotykaną wartością we współczesnych SZBD), zwielokrotnienie wyjściowe zależy od tego, ile wpisów (klucz + wskaźnik do bloku) zmieści się w bloku. Przy 4-bajtowych wskaźnikach do bloków (co pozwala uwzględnić  $2^{32} - 1 = 4,2 \cdot 10^9$  bloków) i 9-bajtowym kluczu zwielokrotnienie wyjściowe wynosi 315.

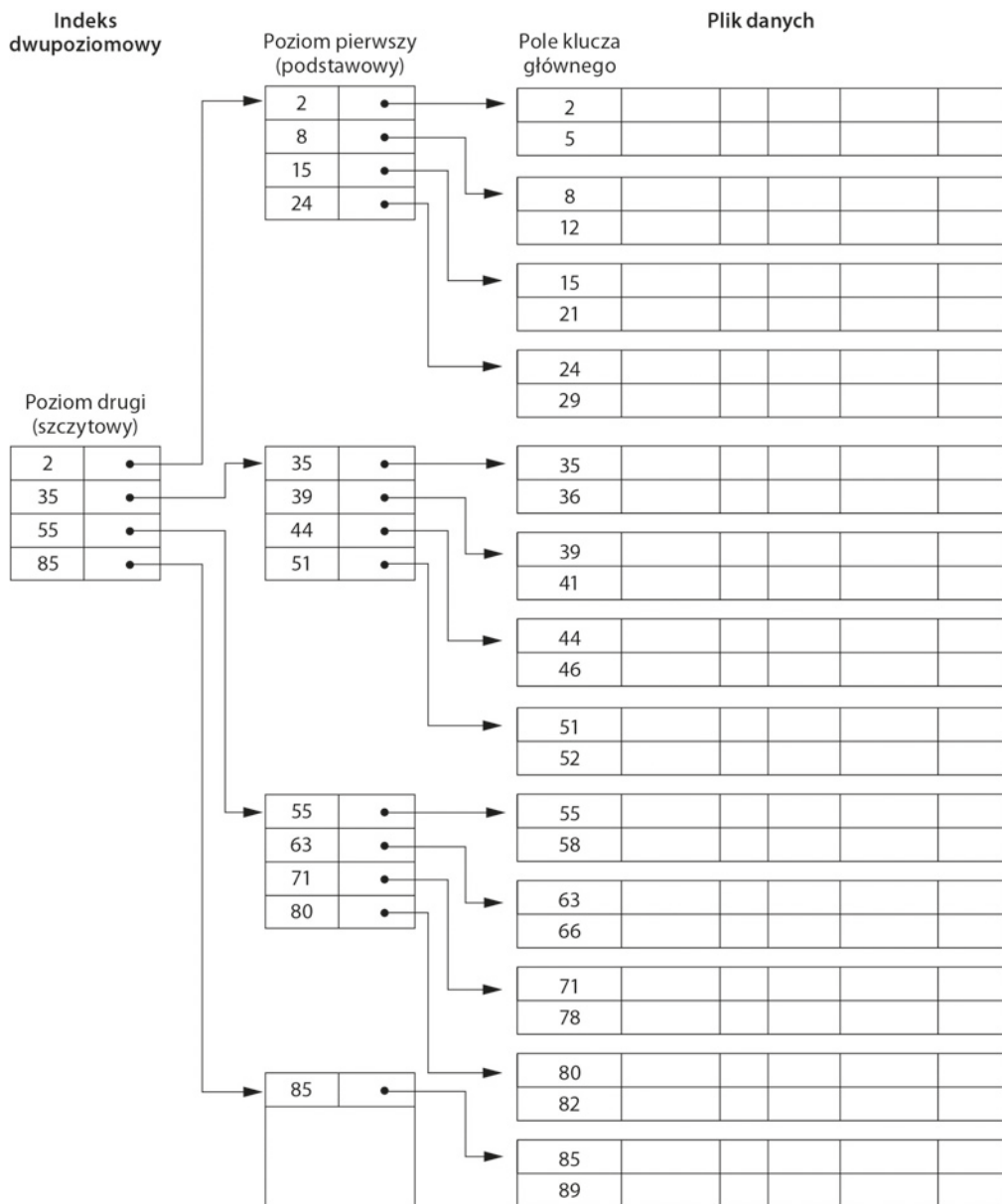
Indeks wielopoziomowy traktuje plik indeksu, który teraz będziemy określać jako **pierwszy** (lub **podstawowy**) **poziom** indeksu wielopoziomowego, jako *plik uporządkowany o odrębnych wartościach* dla każdego  $K(i)$ . Stąd, traktując plik indeksu pierwszego poziomu jako posortowany plik danych, dla poziomu pierwszego możemy utworzyć indeks główny. Taki indeks dla poziomu pierwszego określa się mianem **drugiego poziomu** indeksu wielopoziomowego. Ze względu na fakt, że na drugim poziomie znajduje się indeks główny, możemy użyć zaczepień bloków, tak aby drugi poziom zawierał po jednym wpisie dla *każdego bloku* poziomu pierwszego. Współczynnik blokowy  $bfr_i$  dla poziomu drugiego (oraz wszystkich kolejnych poziomów) jest taki sam jak w przypadku indeksu poziomu pierwszego, ponieważ wszystkie wpisy indeksu mają ten sam rozmiar — każdy z nich zawiera wartość jednego pola oraz adres jednego bloku. Jeżeli pierwszy poziom liczy  $r_1$  wpisów i współczynnik blokowy — stanowiący równocześnie zwielokrotnienie wyjściowe — dla indeksu wynosi  $bfr_i = fo$ , to pierwszy poziom wymaga  $\lceil r_1/fo \rceil$  bloków, co stanowi równocześnie liczbę wpisów  $r_2$  wymaganych na drugim poziomie indeksu.

Cały proces możemy powtórzyć dla poziomu drugiego. **Poziom trzeci**, który jest indeksem głównym dla poziomu drugiego, posiada jeden wpis dla każdego bloku poziomu drugiego, więc liczba wpisów na poziomie trzecim wynosi  $r_3 = \lceil r_2/fo \rceil$ . Należy zauważyć, że drugi poziom jest potrzebny tylko wówczas, gdy poziom pierwszy wymaga więcej niż jednego bloku na dysku i, podobnie, poziom trzeci jest potrzebny tylko wówczas, gdy poziom drugi wymaga więcej niż jednego bloku. Opisany powyżej proces można powtarzać do momentu, aż wszystkie wpisy indeksu pewnego poziomu  $t$  zmieszczą się w pojedynczym bloku. Taki blok poziomu  $t$  określa się mianem **szczytowego** poziomu indeksu<sup>6</sup>. Każdy kolejny poziom redukuje liczbę wpisów z poprzedniego poziomu o współczynnik  $fo$ , czyli zwielokrotnienie wyjściowe indeksu, więc w celu obliczenia  $t$  możemy skorzystać ze wzoru  $1 \leq (r_1/((fo)^t))$ . Stąd indeks wielopoziomowy liczący  $r_1$  wpisów na poziomie pierwszym będzie posiadał około  $t$  poziomów, gdzie  $t = \lceil (\log_{fo} (r_1)) \rceil$ . W trakcie przeszukiwania indeksu na każdym poziomie pobierany jest jeden blok. Tak więc przy przeszukiwaniu indeksu dostęp jest uzyskiwany do  $t$  bloków dysku, gdzie  $t$  jest *liczbą poziomów indeksu*.

Opisywany tu schemat wielopoziomowy może być wykorzystywany w przypadku dowolnego rodzaju indeksu, bez względu na to, czy jest on indeksem głównym, klastrowania czy drugorzędny — o ile indeks poziomu pierwszego posiada *odrębne wartości dla*

<sup>6</sup> Używany przez nas schemat numerowania poziomów indeksu jest odwrotny w stosunku do zwykle stosowanego sposobu definiowania poziomów struktur danych drzewiastych. W przypadku takich struktur o poziomie  $t$  mówi się, że jest poziomem 0. (zerowym),  $t-1$  jest poziomem 1. itd.

wszystkich  $K(i)$  oraz wpisy o stałej długości. Na rysunku 17.6 przedstawiono indeks wielopoziomowy utworzony na indeksie głównym. Przykład 3. ilustruje korzyści osiągane pod względem liczby operacji dostępu do bloków w przypadku użycia indeksu wielopoziomowego w celu wyszukania rekordu.



RYSUNEK 17.6. Dwupoziomowy indeks główny przypominający organizację ISAM (ang. *Indexed Sequential Access Method*, metoda indeksowego dostępu sekwencyjnego)

**PRZYKŁAD 4:** Załóżmy, że zagęszczony indeks drugorzędny z ćwiczenia 3. został zamieniony na indeks wielopoziomowy. Obliczyliśmy współczynnik blokowy pliku, który wynosi  $bfr_1 = 273$  wpisy na blok i stanowi on jednocześnie zwielokrotnienie wyjściowe dla indeksu wielopoziomowego. Obliczyliśmy również liczbę bloków poziomu pierwszego  $b_1 = 1099$ . Liczba bloków poziomu drugiego wyniesie  $b_2 = \lceil (b_1/fo) \rceil = \lceil (1099/273) \rceil = 5$  bloków, a liczba bloków poziomu trzeciego wyniesie  $b_3 = \lceil (b_2/fo) \rceil = \lceil (5/273) \rceil = 1$  blok. Stąd poziom trzeci jest poziomem szczytowym indeksu i  $t = 3$ . W celu uzyskania dostępu do rekordu poprzez przeszukanie indeksu wielopoziomowego musimy uzyskać dostęp do jednego bloku na każdym poziomie oraz do jednego bloku w pliku danych, co oznacza, że potrzebujemy  $t+1 = 3+1 = 4$  operacji dostępu do bloków. W ćwiczeniu 3., używając indeksu jednopoziomowego i przeszukiwania binarnego, potrzebowaliśmy 12 operacji dostępu do bloków.

Należy zauważyć, że moglibyśmy również zdefiniować niezagęszczony indeks główny wielopoziomowy. Ćwiczenie 17.18(c) ilustruje ten przypadek, gdzie *musimy* uzyskać dostęp do bloku danych z pliku przed określeniem, czy szukany rekord znajduje się w pliku. W przypadku indeksu zagęszczonego można tego dokonać, uzyskując dostęp do pierwszego poziomu indeksu (bez uzyskiwania dostępu do bloku danych), gdyż wówczas istnieje wpis indeksu dla *każdego* rekordu w pliku.

Metodą organizacji plików często stosowaną w przypadku biznesowego przetwarzania danych jest plik uporządkowany z wielopoziomowym indeksem głównym utworzonym na jego polu klucza uporządkowania. Taką metodę organizacji określa się mianem **indeksowanego pliku sekwencyjnego** (ang. *indexed sequential file*) i była ona używana w wielu wczesnych systemach firmy IBM. Metoda organizacji **ISAM** firmy IBM wykorzystuje indeks dwupoziomowy, który jest blisko związany z metodą organizacji danych na dysku w cylindrach i ścieżkach (patrz punkt 16.2.1). Na pierwszym poziomie znajduje się indeks cylindryczny, który posiada wartość klucza rekordu zamocowania dla każdego cylindra pakietu dysków oraz wskaźnik na indeks ścieżki cylindra. Indeks ścieżki zawiera wartość klucza rekordu zamocowania dla każdej ścieżki w cylindrze oraz wskaźnik na ścieżkę. W takim przypadku można sekwencyjnie przeszukiwać ścieżkę pod względem określonego rekordu lub bloku. Wstawianie odbywa się z użyciem pliku przepełnienia, który jest okresowo scalany z plikiem danych. Indeks jest odtwarzany w trakcie zmiany organizacji pliku.

Algorytm 17.1 definiuje w zarysie taką procedurę wyszukiwania dla rekordu w pliku danych, który wykorzystuje niezagęszczony wielopoziomowy indeks główny o  $t$  poziomach. Wpis  $i$  na poziomie  $j$  określamy jako  $\langle K_j(i), P_j(i) \rangle$  i szukamy rekordu, którego wartością klucza głównego jest  $K$ . Zakładamy, że wszelkie rekordy przepełnienia są ignorowane. Jeżeli rekord znajduje się w pliku, musi istnieć pewien wpis na poziomie 1., gdzie  $K_1(i) \leq K < K_1(i+1)$  oraz rekord znajduje się w bloku pliku danych, którego adresem jest  $P_1(i)$ . Ćwiczenie 17.23 stanowi omówienie modyfikacji algorytmu wyszukiwania dla innych rodzajów indeksów.

**Algorytm 17.1.** Przeszukiwanie niezagęszczonego indeksu wielopoziomowego o  $t$  poziomach

(\* Zakładamy, że wpis w indeksie prowadzi do rekordu zaczepienia z pierwszym kluczem bloku \*)

$p \leftarrow$  adres bloku indeksu poziomu szczytowego

for  $j \leftarrow t$  step -1 to 1 do

begin

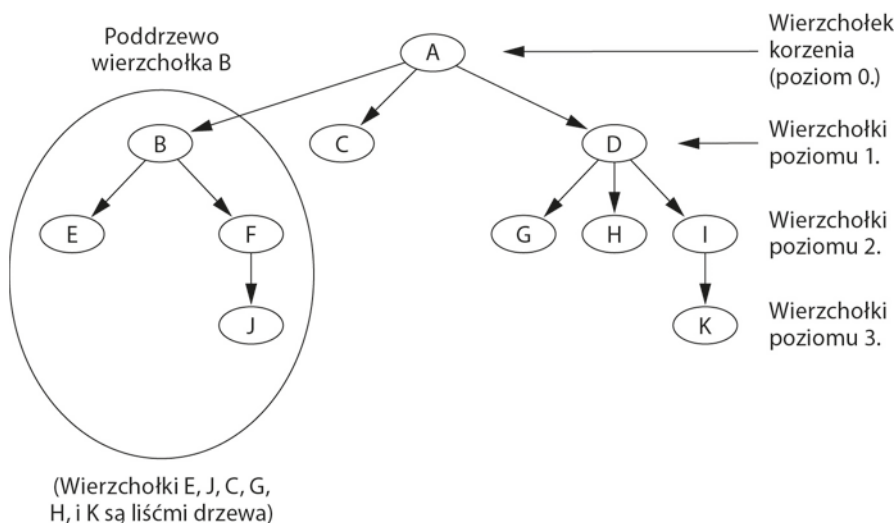
    wczytaj blok indeksu (na  $j$ -tym poziomie indeksu), którego adresem jest  $p$ ;  
 przeszukaj blok  $p$  pod względem wpisu  $i$ , takiego że  $K_j(i) \leq K < K_j(i + 1)$   
 (\* jeżeli  $K_j(i)$   
     jest ostatnim wpisem w bloku, wystarczy aby  $K_j(i) \leq K$  \*);  
 $p \leftarrow P_j(i)$  (\* wybranie odpowiedniego wskaźnika na  $j$ -tym poziomie indeksu \*)  
end;  
wczytaj blok pliku danych, którego adresem jest  $p$ ;  
przeszukaj blok  $p$  pod względem rekordu o kluczu równym  $K$ ;

Jak pokazano, indeks wielopoziomowy redukuje liczbę operacji uzyskiwania dostępu do bloków podczas wyszukiwania rekordu dla danej wartości pola indeksującego. Wciąż występują problemy z obsługą operacji wstawiania i usuwania z indeksu, ponieważ wszystkie jego poziomy stanowią *fizycznie uporządkowane pliki*. W celu zachowania korzyści wynikających z użycia indeksowania wielopoziomowego i jednocześnie niwelowania problemów związanych ze wstawianiem i usuwaniem projektanci opracowali indeks wielopoziomowy, który pozostawia pewną ilość wolnego miejsca w każdym ze swoich bloków dla nowowstawianych wpisów. Jest to tak zwany **dynamiczny indeks wielopoziomowy** (ang. *dynamic multilevel index*) i często implementuje się go przy użyciu struktur określanych mianem B-drzew i B<sup>+</sup>-drzew, które zostaną opisane w następnym punkcie.

## 17.3. Dynamiczne indeksy wielopoziomowe z użyciem B-drzew i B<sup>+</sup>-drzew

B-drzewa i B<sup>+</sup>-drzewa stanowią przypadki szczególne dobrze znanych **drzewiastych** struktur danych. Poniżej bardzo pobieżnie przedstawimy terminologię używaną przy omawianiu tych struktur. **Drzewo** (ang. *tree*) składa się z **wierzchołków** (ang. *nodes*). Każdy wierzchołek w drzewie, oprócz specjalnego wierzchołka, noszącego nazwę **korzenia** (ang. *root*), posiada jeden **wierzchołek nadrzędny** (ang. *parent*) oraz kilka (zero lub więcej) **wierzchołków podrzędnych** (ang. *child*). Korzeń nie posiada wierzchołka nadrzędnego. Wierzchołek nie posiadający wierzchołków podrzędnych określa się mianem **liścia** (ang. *leaf*). Wierzchołki nie będące liśćmi to **wierzchołki wewnętrzne** (ang. *internal nodes*). **Poziom** (ang. *level*) wierzchołka jest zawsze o jeden większy od poziomu jego wierzchołka nadrzędnego, a poziomem korzenia jest zero<sup>7</sup>. **Poddrzewo** (ang. *subtree*) wierzchołka składa się z danego wierzchołka oraz jego wszystkich **wierzchołków potomnych** (ang. *descendant nodes*), czyli jego wierzchołków podrzędnych, ich wierzchołków podrzędnych itd. Ścisła rekurencyjna definicja poddrzewa określa, że składa się ono z wierzchołka  $n$  oraz poddrzew wszystkich jego wierzchołków podrzędnych. Na rysunku 17.7 przedstawiono drzewiastą strukturę danych. Korzeniem jest wierzchołek A, zaś jego wierzchołkami podrzędnymi są wierzchołki B, C i D. Wierzchołki E, J, C, G, H i K są liśćmi. Ponieważ liście znajdują się na różnych poziomach, to drzewo jest **niezrównoważone**.

<sup>7</sup> Jest to standardowa definicja poziomu wierzchołków drzew, używana w podrozdziale 17.3. Różni się ona od definicji podanej w podrozdziale 17.2 dla indeksów wielopoziomowych.



RYSUNEK 17.7. Drzewiasta struktura danych przedstawiająca drzewo nierównoważone

W podrozdziale 17.3.1 zostaną omówione drzewa wyszukiwania, a następnie B-drzewa, których można używać jako dynamicznych indeksów wielopoziomowych w celu sterowania wyszukiwaniem rekordów w pliku danych. Wierzchołki B-drzewa są wypełnione do poziomu od 50 do 100 procent i wskaźniki na bloki danych są przechowywane zarówno w wierzchołkach wewnętrznych, jak i liściach struktur tych drzew. W podrozdziale 17.3.2 zostaną omówione B<sup>+</sup>-drzewa, stanowiące odmianę B-drzew, w których wskaźniki na bloki danych pliku są przechowywane tylko w liściach. Może to dawać mniejszą liczbę poziomów i większą pojemność indeksów. W najpopularniejszych SZBD na rynku do indeksowania używa się głównie B<sup>+</sup>-drzew.

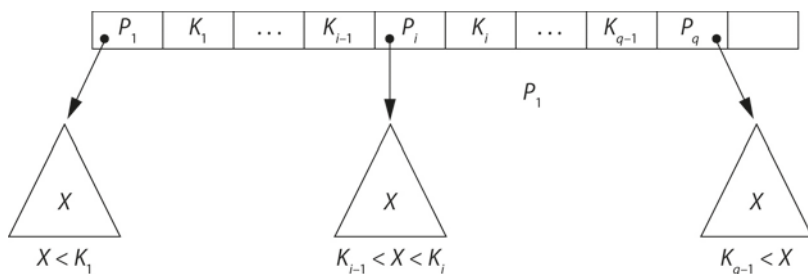
### 17.3.1. Drzewa wyszukiwania i B-drzewa

**Drzewo wyszukiwania** stanowi specjalny rodzaj drzewa, który jest używany w celu sterowania procesem wyszukiwania rekordu dla danej wartości jednego z jego pól. Indeksy wielopoziomowe omówione w podrozdziale 17.2 można postrzegać jako odmianę drzew wyszukiwania: każdy wierzchołek indeksu wielopoziomowego może zawierać najwyżej  $fo$  wskaźników oraz  $fo$  wartości klucza, gdzie  $fo$  oznacza zwielokrotnienie wyjściowe indeksu. Wartości pola indeksu w każdym wierzchołku przekierowują nas do kolejnego wierzchołka do momentu, aż osiągniemy blok pliku danych, który zawiera poszukiwany rekord. Śledząc odwołanie wskaźnika na każdym poziomie ograniczamy wyszukiwanie do poddrzewa wyszukiwania i ignorujemy wszystkie wierzchołki, które nie należą do niego.

**Drzewa wyszukiwania.** Drzewo wyszukiwania różni się nieco od indeksu wielopoziomowego. **Drzewo wyszukiwania** (ang. *search tree*) rzędu  $p$  jest takim drzewem, że każdy jego wierzchołek zawiera *najwyżej*  $p-1$  wartości wyszukiwania oraz  $p$  wskaźników w kolejności  $\langle P_1, K_1, P_2, K_2, \dots, P_{q-1}, K_{q-1}, P_q \rangle$ , gdzie  $q \leq p$ . Każde  $P_i$  oznacza wskaźnik na wierzchołek podrzędny (lub wskaźnik zerowy), zaś każde  $K_i$  jest wartością wyszukiwania należącą do pewnego uporządkowanego zbioru wartości. Zakłada się, że wszystkie wartości

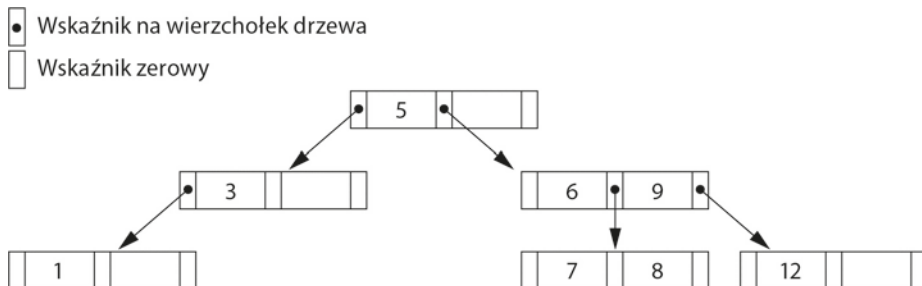
wyszukiwania mają unikatowe wartości<sup>8</sup>. Na rysunku 17.8 przedstawiono wierzchołek w drzewie wyszukiwania. W przypadku takiego drzewa zawsze muszą być spełnione dwa ograniczenia:

- (1) W każdym wierzchołku  $K_1 < K_2 < \dots < K_{q-1}$ .
- (2) Dla wszystkich wartości  $X$  w poddrzewie, na które wskazuje wskaźnik  $P_i$ , zachodzi związek  $K_{i-1} < X < K_i$  dla  $1 < i < q$ ;  $X < K_i$  dla  $i = 1$ ; oraz  $K_{i-1} < X$  dla  $i = q$  (patrz rysunek 17.8).



RYSUNEK 17.8. Wierzchołek w drzewie wyszukiwania ze wskaźnikami na znajdujące się pod nim poddrzewa

Kiedy szukamy wartości  $X$ , wybieramy odpowiedni wskaźnik  $P_i$  zgodnie z wzorami z przedstawionego powyżej warunku 2. Na rysunku 17.9 przedstawiono drzewo wyszukiwania rzędu  $p = 3$  i wartości wyszukiwania typu całkowitoliczbowego. Warto zauważyć, że niektóre wskaźniki  $P_i$  w wierzchołkach mogą być wskaźnikami zerowymi.



RYSUNEK 17.9. Drzewo wyszukiwania rzędu  $p = 3$

Drzewa wyszukiwania możemy użyć jako mechanizmu służącego do znajdowania rekordów przechowywanych w pliku na dysku. Wartości w drzewie mogą być wartościami jednego z pól pliku, określanego mianem **poła wyszukiwania** (ang. *search field*) — jeżeli wyszukiwanie jest sterowane indeksem wielopoziomowym, to jest ono tym samym polem co pole indeksujące. Każda wartość klucza występująca w drzewie jest powiązana ze wskaźnikiem na rekord w pliku danych posiadającym tę wartość. Inne rozwiązanie polega na zapewnieniu, aby wskaźnik mógł wskazywać na blok dyskowy zawierający

<sup>8</sup> Ograniczenie to można złagodzić. Jeżeli indeks utworzono na polu nie będącym polem klucza, mogą występować duplikaty wartości wyszukiwania i można zmodyfikować strukturę wierzchołków oraz reguły przechodzenia drzewa.

dany rekord. Samo drzewo wyszukiwania może być przechowywane na dysku poprzez przypisanie każdego wierzchołka drzewa do bloku dyskowego. W momencie wstawienia nowego rekordu musimy zaktualizować drzewo wyszukiwania, wstawiając wpis do drzewa zawierającego poszukiwaną wartość pola nowego rekordu oraz wskaźnik na nowy rekord.

Zachodzi potrzeba opracowania algorytmów wstawiania i usuwania wartości wyszukiwania z drzewa wyszukiwania przy jednoczesnym zachowaniu dwóch wspomnianych wcześniej ograniczeń. Ogólnie rzecz biorąc, algorytmy takie nie gwarantują, że drzewo wyszukiwania będzie **zrównoważone** (ang. *balanced*), co oznacza, że wszystkie jego liście będą się znajdowały na tym samym poziomie<sup>9</sup>. Drzewo z rysunku 17.7 nie jest zrównoważone, ponieważ posiada liście na poziomach 1., 2. i 3. Oto cele, jakie ma zapewniać zrównoważenie drzewa wyszukiwania:

- Gwarantowanie, że węzły są równomiernie rozłożone, dzięki czemu głębokość drzewa dla danego zbioru kluczy jest minimalizowana, a drzewo nie staje się skośne (i nie obejmuje bardzo głęboko zagnieżdżonych węzłów).
- Zapewnianie stałej szybkości wyszukiwania, tak aby średni czas wyszukiwania dowolnych kluczy był w przybliżeniu taki sam.

Minimalizowanie liczby poziomów drzewa to jeden cel. Innym, pośrednim celem jest upewnianie się, że drzewo indeksu nie wymaga nadmiernej restrukturyzacji, gdy rekordy są wstawiane i usuwane w pliku głównym. Chcemy, by wierzchołki były możliwie wypełnione, i nie chcemy, aby stawały się puste w wyniku zbyt wielu operacji usuwania. Usunięcie rekordu może powodować, że niektóre wierzchołki drzewa staną się prawie puste, co skutkuje marnowaniem pamięci i zwiększaniem liczby poziomów. Oba te problemy rozwiązuje wykorzystanie B-drzew, gdzie nakłada się dodatkowe ograniczenia.

**B-drzewa.** B-drzewo posiada dodatkowe ograniczenia, które zapewniają, że jest ono zawsze zrównoważone oraz że przestrzeń marnowana przez operacje usuwania, o ile w ogóle występuje, nigdy nie staje się duża. Jednak algorytmy wstawiania i usuwania są nieco bardziej skomplikowane, gdyż muszą zapewnić zachowanie tych ograniczeń. Mimo wszystko większość operacji wstawiania i usuwania to proste procesy i stają się one bardziej złożone tylko w szczególnych okolicznościach, a dokładnie wówczas, gdy próbujemy wstawić dane do wierzchołka, który jest już pełny, lub usuwać z wierzchołka, który w ten sposób staje się wypełniony mniej niż w połowie. Ujmując rzecz formalnie, **B-drzewo rzędu  $p$**  używane jako struktura dostępowa względem *pola klucza* w celu wyszukiwania rekordów w pliku danych można zdefiniować w sposób następujący:

(1) Każdy wierzchołek wewnętrzny w B-drzewie (rysunek 17.10(a)) ma postać

$$\langle P_1, \langle K_1, Pr_1 \rangle, P_2, \langle K_2, Pr_2 \rangle, \dots, \langle K_{q-1}, Pr_{q-1} \rangle, Pr_q \rangle$$

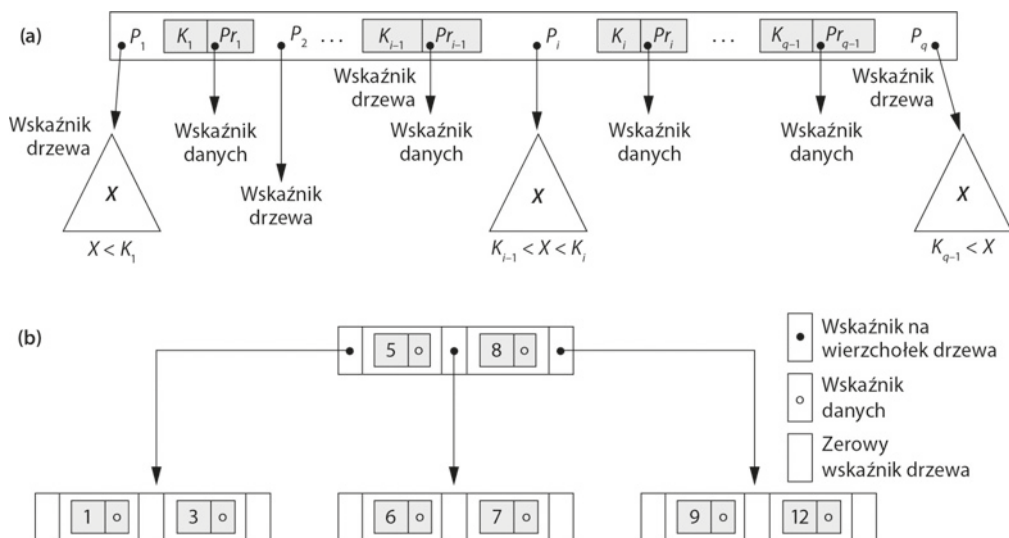
gdzie  $q \leq p$ . Każde  $P_i$  oznacza **wskaźnik drzewa** (ang. *tree pointer*) — wskaźnik na inny wierzchołek w B-drzewie. Każde  $Pr_i$  oznacza **wskaźnik danych** (ang. *data pointer*)<sup>10</sup> — wskaźnik na rekord, którego wartość pola klucza wyszukiwania jest równa  $K_i$  (lub blok pliku danych zawierający ten rekord).

<sup>9</sup> Definicja *zrównoważenia* w przypadku drzew binarnych ma inną postać. Zrównoważone drzewa binarne określa się mianem *drzew AVL*.

<sup>10</sup> Wskaźnik danych jest albo adresem bloku, albo adresem rekordu. W tym drugim przypadku zasadniczo jest to adres bloku oraz przesunięcie rekordu w tym bloku.



(2) W każdym wierzchołku  $K_1 < K_2 < \dots < K_{q-1}$ .



RYSUNEK 17.10. Struktury B-drzewa. (a) Wierzchołek w B-drzewie z  $q-1$  wartościami wyszukiwania. (b) B-drzewo rzędu  $p = 3$ . Wartości zostały do niego wstawione w kolejności 8, 5, 1, 7, 3, 12, 9, 6

(3) Dla wszystkich wartości pola wyszukiwania  $X$  w poddrzewie, na które wskazuje wskaźnik  $P_i$  ( $i$ -te poddrzewo — patrz rysunek 17.10(a)), mamy:

$$K_{i-1} < X < K_i \text{ dla } 1 < i < q; X < K_i \text{ dla } i = 1; \text{ oraz } K_{i-1} < X \text{ dla } i = q.$$

(4) Każdy wierzchołek posiada najwyżej  $p$  wskaźników drzewa.

(5) Każdy wierzchołek oprócz korzenia i liści posiada co najmniej  $\lceil (p/2) \rceil$  wskaźników drzewa. Korzeń posiada co najmniej dwa wskaźniki drzewa, chyba że jest jedynym wierzchołkiem w drzewie.

(6) Wierzchołek o  $q$  wskaźnikach drzewa, gdzie  $q \leq p$ , posiada  $q-1$  wartości pola klucza wyszukiwania (a stąd posiada  $q-1$  wskaźników danych).

(7) Wszystkie liście znajdują się na tym samym poziomie. Liście posiadają taką samą strukturę co wierzchołki wewnętrzne poza tym, że wszystkie ich wskaźniki drzewa  $P_i$  są zerowe.

Na rysunku 17.10(b) przedstawiono B-drzewo rzędu  $p = 3$ . Należy zauważyć, że wszystkie wartości wyszukiwania  $K$  w B-drzewie są unikatowe, ponieważ założyliśmy, że drzewo jest używane jako struktura dostępową względem pola klucza. Jeżeli B-drzewa użyjemy na polu *niebędącym polem klucza*, musimy zmienić definicję wskaźników pliku  $Pr_i$  tak, aby wskazywały na bloki (lub klastry bloków), które zawierają wskaźniki na rekordy pliku. Taki dodatkowy poziom pośredni przypomina rozwiązanie trzecie z podrozdziału 17.1.3 dla indeksów drugorzędnych.

B-drzewo rozpoczyna się pojedynczym korzeniem (który początkowo jest równocześnie liściem) na poziomie 0. Kiedy wierzchołek korzenia zostanie zapełniony  $p-1$  wartościami klucza wyszukiwania i spróbujemy wstawić kolejny wpis w drzewie, wierzchołek korzenia zostanie rozdzielony na dwa wierzchołki na poziomie 1. W korzeniu zostaje za-

chowana tylko wartość środkowa, a pozostałe wartości są równo rozdzielane między dwoma nowymi wierzchołkami. Kiedy wierzchołek niebędący korzeniem jest zapełniony i wstawiamy do niego nowy wpis, wierzchołek ten zostaje rozdzielony na dwa wierzchołki na tym samym poziomie i środkowa wartość zostaje przeniesiona do wierzchołka nadrzędnego wraz z dwoma wskaźnikami na nowe wierzchołki. Jeżeli wierzchołek nadrzędny jest zapełniony, również zostaje rozdzielony. Rozdzielanie może się propagować aż do wierzchołka korzenia, tworząc nowy poziom w razie podziału korzenia. Nie będziemy w tym miejscu szczegółowo omawiać algorytmów dla B-drzew<sup>11</sup>, a zamiast tego ogólnie przedstawimy procedury wyszukiwania i wstawiania dla B<sup>+</sup>-drzew w kolejnym podrozdziale.

Jeżeli usunięcie wartości powoduje, że wierzchołek jest zapełniony mniej niż w połowie, zostaje on połączony ze swoimi wierzchołkami sąsiednimi i proces ten również może propagować się aż do wierzchołka korzenia. Stąd usuwanie może zredukować liczbę poziomów drzewa. Na podstawie analizy i symulacji wykazano, że po wielu losowych wstawieniach i usunięciach wykonanych na B-drzewie wierzchołki są zapełnione mniej więcej w 69 procentach, jeżeli liczba wartości w drzewie ustabilizuje się. Jest tak również w przypadku B<sup>+</sup>-drzew. Jeżeli tak się stanie, dzielenie i łączenie wierzchołków zdarza się rzadko, więc operacje wstawiania i usuwania są wówczas wydajne. Jeżeli liczba wartości rośnie, drzewo rozszerza się bez problemów, choć w razie występowania podziałów niektóre operacje wstawiania mogą trwać nieco dłużej. Każdy wierzchołek B-drzewa może zawierać *najwyżej*  $p$  wskaźników drzewa,  $p - 1$  wskaźników danych i  $p - 1$  wartości pola klucza wyszukiwania (patrz rysunek 17.10(a)).

Wierzchołek B-drzewa może zawierać dodatkowe informacje potrzebne algorytmom, które operują na drzewie (np. liczbę wpisów  $q$  w danym wierzchołku i wskaźnik do węzła nadrzędnego). Dalej pokazujemy, jak wyznaczyć liczbę bloków i poziomów w B-drzewie.

**PRZYKŁAD 5:** Załóżmy, że pole wyszukiwania jest polem niebędącym kluczem uporządkowania i konstruujemy na tym polu B-drzewo (dla  $p = 23$ ). Zakładamy, że każdy wierzchołek tego B-drzewa jest zapełniony w 69 procentach. Każdy wierzchołek będzie posiadał średnio  $p \cdot 0,69 = 23 \cdot 0,69$ , czyli około 16 wskaźników, a zatem 15 wartości pola klucza wyszukiwania. **Średnie zwielokrotnienie wyjściowe**  $f_0 = 16$ . Możemy zacząć od korzenia i sprawdzić, ile wartości i wskaźników może średnio występować na każdym kolejnym poziomie:

Korzeń:	1 wierzchołek	15 wpisów	16 wskaźników
Poziom 1.:	16 wierzchołków	240 wpisów	256 wskaźników
Poziom 2.:	256 wierzchołków	3840 wpisów	4096 wskaźników
Poziom 3.:	4096 wierzchołków	61 440 wpisów	

Na każdym poziomie obliczyliśmy liczbę wpisów mnożąc całkowitą liczbę wskaźników na poprzednim poziomie przez 15, czyli średnią liczbę wpisów w każdym wierzchołku. Stąd dla danego rozmiaru bloku (512 bajtów), rozmiaru wskaźnika rekordu/danych (7 bajtów), rozmiaru wskaźnika drzewa/bloku (6 bajtów) oraz rozmiaru pola klucza wyszukiwania dwupoziomowe B-drzewo rzędu 23 przechowuje średnio  $3840 + 240 + 15 = 4095$  wpisów. Trzypoziomowe B-drzewo przechowuje średnio 65535 wpisów.

<sup>11</sup> Szczegółowe omówienie algorytmów wstawiania i usuwania w B-drzewach znajdziesz w Ramakrishnan i Gehrke (2003).

B-drzewa są czasem używane jako **podstawowa metoda organizacji plików**. W takim przypadku w wierzchołkach B-drzewa są przechowywane *całe rekordy*, a nie tylko wpisy  $\langle \text{klucz wyszukiwania}, \text{wskaźnik na rekord} \rangle$ . Rozwiązanie takie sprawdza się w przypadku plików o względnie *niewielkiej liczbie rekordów* oraz *niewielkim rozmiarze rekordu*. W przeciwnym razie zwielokrotnienie wyjściowe oraz liczba poziomów stają się zbyt duże, aby możliwe było wydajne uzyskiwanie dostępu do danych.

Podsumowując, B-drzewa stanowią wielopoziomową strukturę dostępową, tworzącą drzewo zrównoważone, w którym każdy wierzchołek jest wypełniony co najmniej w połowie. Każdy wierzchołek w B-drzewie rzędu  $p$  może posiadać najwyżej  $p-1$  wartości wyszukiwania.

### 17.3.2. B<sup>+</sup>-drzewa

Większość implementacji dynamicznych indeksów wielopoziomowych wykorzystuje odmianę B-drzewa noszącą nazwę **B<sup>+</sup>-drzewa**. W przypadku B-drzewa każda wartość pola wyszukiwania występuje raz na pewnym poziomie w drzewie wraz ze wskaźnikiem danych. W przypadku B<sup>+</sup>-drzewa wskaźniki danych są przechowywane *tylko w liściach*, a przez to struktura wierzchołków liści różni się od struktury wierzchołków wewnętrznych. Wierzchołki liści posiadają wpis dla *każdej* wartości pola wyszukiwania wraz ze wskaźnikiem danych na rekord (lub blok zawierający dany rekord), jeżeli pole wyszukiwania jest polem klucza. W przypadku pola wyszukiwania niebędącego polem klucza, wskaźnik określa blok zawierający wskaźnik do rekordów pliku danych, co tworzy dodatkowy poziom pośredni.

Wierzchołki liści B<sup>+</sup>-drzewa są zwykle połączone razem, co zapewnia możliwość uzyskiwania uporządkowanego dostępu do rekordów względem pola wyszukiwania. Takie wierzchołki liści przypominają pierwszy (podstawowy) poziom indeksu. Wierzchołki wewnętrzne B<sup>+</sup>-drzewa odpowiadają innym poziomom indeksu wielopoziomowego. Niektóre wartości pola wyszukiwania z liści są *powtarzane* w wierzchołkach wewnętrznych B<sup>+</sup>-drzewa, co pozwala sterować wyszukiwaniem. Struktura *wierzchołków wewnętrznych* B<sup>+</sup>-drzewa rzędu  $p$  (rysunek 17.11(a)) jest następująca:

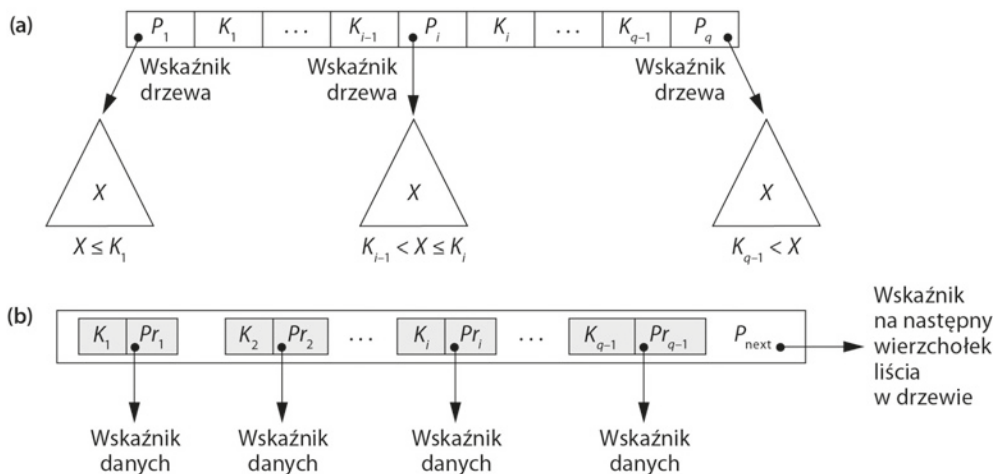
- (1) Każdy wierzchołek wewnętrzny ma postać

$$\langle P_1, K_1, P_2, K_2, \dots, P_{q-1}, K_{q-1}, P_q \rangle$$

gdzie  $q \leq p$  i każde  $P_i$  jest **wskaźnikiem drzewa**.

- (2) W każdym wierzchołku wewnętrznym  $K_1 < K_2 < \dots < K_{q-1}$ .
- (3) Dla wszystkich wartości pola wyszukiwania  $X$  w poddrzewie, na które wskazuje wskaźnik  $P_i$ , mamy:  $K_{i-1} < X < K_i$  dla  $1 < i < q$ ;  $X \leq K_i$  dla  $i = 1$ ; oraz  $K_{i-1} < X$  dla  $i = q$  (patrz rysunek 17.11(a))<sup>12</sup>.
- (4) Każdy wierzchołek wewnętrzny posiada najwyżej  $p$  wskaźników drzewa.
- (5) Każdy wierzchołek wewnętrzny oprócz korzenia i liści posiada co najmniej  $\lceil (p/2) \rceil$  wskaźników drzewa. Korzeń posiada co najmniej dwa wskaźniki drzewa, jeżeli jest wierzchołkiem wewnętrznym.

<sup>12</sup> Prezentowana definicja jest zgodna z definicją Knutha (1998). B<sup>+</sup>-drzewo można zdefiniować inaczej, zamieniając miejscami symbole  $< i \leq (K_{i-1} \leq X < K_i; K_{q-1} \leq X)$ , jednak zasada pozostaje ta sama.



RYSUNEK 17.11. Wierzchołki B<sup>+</sup>-drzewa. (a) Wierzchołek wewnętrzny w B<sup>+</sup>-drzewie z  $q-1$  wartościami wyszukiwania. (b) Wierzchołek liścia w B<sup>+</sup>-drzewie z  $q-1$  wskaźnikami danych

- (6) Wierzchołek wewnętrzny o  $q$  wskaźnikach drzewa, gdzie  $q \leq p$ , posiada  $q-1$  wartości pola klucza wyszukiwania.

Struktura *wierzchołków liści* B<sup>+</sup>-drzewa rzędu  $p$  (rysunek 17.11(b)) jest następująca:

- (1) Każdy wierzchołek liścia ma postać

$$\langle \langle K_1, Pr_1 \rangle, \langle K_2, Pr_2 \rangle, \dots, \langle K_{q-1}, Pr_{q-1} \rangle, P_{nast} \rangle$$

gdzie  $q \leq p$  i każde  $Pr_i$  jest **wskaźnikiem danych**, zaś  $P_{nast}$  wskazuje na *następny wierzchołek liścia* B<sup>+</sup>-drzewa.

- (2) W każdym wierzchołku liścia  $K_1 \leq K_2 \dots, K_{q-1}$ , dla  $q \leq p$ .
- (3) Każde  $Pr_i$  jest **wskaźnikiem danych** wskazującym na rekord, którego wartość pola wyszukiwania jest równa  $K_i$ , lub na blok pliku zawierający dany rekord (lub na blok rekordu wskaźników wskazujących na rekordy, których wartością pola wyszukiwania jest  $K_i$ , jeżeli pole wyszukiwania nie jest kluczem).
- (4) Każdy wierzchołek liścia posiada co najmniej  $\lceil (p/2) \rceil$  wskaźników drzewa.
- (5) Wszystkie wierzchołki liści znajdują się na tym samym poziomie.

Wskaźniki w wierzchołkach wewnętrznych są *wskaźnikami drzewa* na bloki, które są wierzchołkami drzewa, natomiast wskaźniki w wierzchołkach liści są *wskaźnikami danych* na rekordy pliku danych lub bloki — oprócz wskaźnika  $P_{nast}$ , który jest wskaźnikiem drzewa na następny wierzchołek liścia. Rozpoczynając od pierwszego od lewej strony wierzchołka liścia, można przejść wierzchołki liści na liście jednokierunkowej, wykorzystując wskaźniki  $P_{nast}$ . Zapewnia to możliwość uporządkowanego dostępu do rekordów danych względem pola indeksującego. Można również uwzględnić wskaźnik  $P_{poprz}$ . W przypadku B<sup>+</sup>-drzewa na polu niebędącym polem klucza wymagane jest wprowadzenie dodatkowego poziomu pośredniego, podobnego do przedstawionego na rysunku 17.5, więc wskaźniki  $Pr$  są wskaźnikami blokowymi na bloki zawierające zbiór wskaźników rekordów na faktyczne rekordy w pliku danych, co omówiono w opisie rozwiązania trzeciego w podrozdziale 17.1.3.

Ze względu na fakt, że *wierzchołki wewnętrzne* B<sup>+</sup>-drzewa zawierają wartości wyszukiwania oraz wskaźniki drzewa bez żadnych wskaźników danych, można w nich pomieścić więcej wpisów niż ma to miejsce w przypadku podobnych B-drzew. Stąd dla tego samego rozmiaru bloku (wierzchołka) rząd  $p$  będzie większy w przypadku B<sup>+</sup>-drzewa niż B drzewa, co ilustruje przykład 6. Może to prowadzić do mniejszej liczby poziomów w przypadku B<sup>+</sup>-drzewa, co skraca czas wyszukiwania. Ze względu na fakt, że struktury wierzchołków wewnętrznych i liści B<sup>+</sup>-drzewa są różne, rząd  $p$  może być różny. Symbol  $p$  będzie oznaczał rząd *wierzchołków wewnętrznych*, zaś  $p_{\text{liść}}$  — rząd *wierzchołków liści*, który definiujemy jako maksymalną liczbę wskaźników danych w wierzchołku liścia.

**PRZYKŁAD 6:** W celu obliczenia rzędu  $p$  B<sup>+</sup>-drzewa założmy, że pole klucza wyszukiwania ma  $V = 9$  bajtów długości, rozmiar bloku dyskowego wynosi  $B = 512$  bajtów, wskaźnik na rekord ma  $P_r = 7$  bajtów długości, zaś wskaźnik na blok/drzewo ma  $P = 6$  bajtów długości. Wierzchołek wewnętrzny B<sup>+</sup>-drzewa może posiadać do  $p$  wskaźników drzewa i  $p-1$  wartości pola wyszukiwania. Muszą one mieścić się w pojedynczym bloku. Stąd mamy:

$$\begin{aligned}(p * P) + ((p - 1) * V) &\leq B \\ (p * 6) + ((p - 1) * 9) &\leq 512 \\ (15 * p) &\leq 521\end{aligned}$$

Możemy wybrać jako  $p$  największą wartość, która spełni powyższą nierówność, co każe nam wybrać  $p = 34$ . Jest to wartość większa od 23 dla B-drzewa (Czytelnikom pozostawiamy wyznaczenie rzędu B-drzewa dla wskaźników o tej samej wielkości), co oznacza większe zwielokrotnienie wyjściowe i większą liczbę wpisów w każdym wierzchołku wewnętrznym B<sup>+</sup>-drzewa niż w analogicznym B-drzewie. Wierzchołki liści B<sup>+</sup>-drzewa będą posiadały taką samą liczbę wartości i wskaźników oprócz tego, że wskaźnikami tymi są wskaźniki danych oraz wskaźnik następnego wierzchołka. Stąd rząd  $p_{\text{liść}}$  dla wierzchołków liści można obliczyć jako:

$$\begin{aligned}(p_{\text{liść}} * (P_r + V)) + P &\leq B \\ (p_{\text{liść}} * (7 + 9)) + 6 &\leq 512 \\ (16 * p_{\text{liść}}) &\leq 506\end{aligned}$$

Okazuje się, że każdy wierzchołek liścia może zawierać do  $p_{\text{liść}} = 31$  par wartość klucza/ wskaźnik danych, zakładając że wskaźniki danych są wskaźnikami rekordów.

Podobnie jak w przypadku B-drzew, mogą nam być potrzebne w każdym wierzchołku dodatkowe informacje w celu zaimplementowania algorytmów wstawiania i usuwania. Informacje te mogą uwzględniać rodzaj wierzchołka (wewnętrzny lub liścia), liczbę bieżących wpisów  $q$  w wierzchołku oraz wskaźniki na wierzchołek nadrzędny i z tego samego poziomu. Stąd, zanim wykonamy powyższe obliczenia dla  $p$  i  $p_{\text{liść}}$ , powinniśmy zredukować rozmiar bloku o ilość przestrzeni wymaganej dla takich informacji. Kolejny przykład ilustruje sposób obliczania liczby wpisów w B<sup>+</sup>-drzewie.

**PRZYKŁAD 7:** Założmy, że konstruujemy B<sup>+</sup>-drzewo na polu z przykładu 6. W celu obliczenia przybliżonej liczby wpisów w B<sup>+</sup>-drzewie zakładamy, że każdy wierzchołek jest zapełniony w 69 procentach. Każdy wierzchołek wewnętrzny będzie posiadał średnio  $34 * 0,69$ , czyli około 23 wskaźników, a zatem 22 wartości. Każdy wierzchołek liścia będzie posiadał

średnio  $0,69 \cdot p_{\text{liść}} = 0,69 \cdot 31$ , czyli około 21 wskaźników na rekordy danych. B<sup>+</sup>-drzewo będzie posiadać następujące średnie liczby wpisów na każdym kolejnym poziomie:

Korzeń:	1 wierzchołek	22 wpisy	23 wskaźniki
Poziom 1.:	23 wierzchołki	506 wpisów	529 wskaźników
Poziom 2.:	529 wierzchołków	11638 wpisów	12 167 wskaźników
Poziom liści:	12 167 wierzchołków	255 507 wpisów	

Dla rozmiaru bloku, rozmiaru wskaźnika oraz rozmiaru pola klucza wyszukiwania z przykładu 6. trzypoziomowe B<sup>+</sup>-drzewo przechowuje do 255 507 wskaźników rekordów ze średnim wypełnieniem wierzchołków na poziomie 69%. Warto zauważyć, że liście są tu traktowane inaczej niż wierzchołki wewnętrzne. Liczbę wskaźników danych w liściu wyznaczono na  $12\,167 \cdot 21$  (przy poziomie 69% wypełnienia liści, które mieszczą do 31 kluczy ze wskaźnikami danych). Porównaj to z 65 535 wpisami w analogicznym B-drzewie z przykładu 5. Ponieważ B-drzewo na wszystkich poziomach obejmuje dla każdego klucza wyszukiwania wskaźnik danych/rekordu, zwykle mieści mniej kluczy przy tej samej liczbie poziomów indeksu. Jest to główny powód, dla którego B<sup>+</sup>-drzewa są preferowane zamiast B-drzew jako indeksy plików baz danych. W większości SZBD, np. w Oracle, wszystkie indeksy są tworzone jako B<sup>+</sup>-drzewa.

**Wyszukiwanie, wstawianie i usuwanie w przypadku B<sup>+</sup>-drzew.** Algorytm 17.2 prezentuje procedurę użycia B<sup>+</sup>-drzewa jako struktury dostępowej w celu wyszukiwania rekordu. Algorytm 17.3 ilustruje procedurę wstawiania rekordu do pliku w przypadku struktury dostępowej opartej na B<sup>+</sup>-drzewie. W przypadku obu algorytmów zakładamy istnienie pola wyszukiwania będącego kluczem i w przypadku, gdy tak nie jest, należy je odpowiednio zmodyfikować. Wstawianie i usuwanie zilustrujemy za pomocą przykładu.

**Algorytm 17.2.** Wyszukiwanie rekordu z wartością pola klucza wyszukiwania K przy użyciu B<sup>+</sup>-drzewa

```

n ← blok zawierający wierzchołek korzenia B+-drzewa;
wczytaj blok n;
while (n nie jest wierzchołkiem liścia B+-drzewa) do
  begin
    q ← liczba wskaźników drzewa występujących w wierzchołku n;
    if K ≤ n.K1 (* n.K1 oznacza odwołanie do i-tej wartości pola wyszukiwania
      w wierzchołku n *)
    then n ← n.P1 (*n.P1 oznacza odwołanie do i-tego wskaźnika drzewa
      w wierzchołku n *)
    else if K > n.Kq-1
    then n ← n.Pq
    else begin
      wyszukaj wierzchołek n dla wpisu i, taki że n.Ki-1 < K ≤ n.Ki;
      n ← n.Pi
    end;
    wczytaj blok n
  end;
przeszukaj blok n względem wpisu (Ki, Pri) dla K = Ki; (* wyszukaj wierzchołek
liścia *)

```

```

if znaleziono
  then wczytaj blok pliku danych o adresie  $Pr_i$  i pobierz rekord
else rekord z wartością pola wyszukiwania  $K$  nie znajduje się w pliku danych;

```

**Algorytm 17.3.** Wstawianie rekordu z wartością pola klucza wyszukiwania  $K$  przy użyciu B<sup>+</sup>-drzewa rzędu  $p$

```

 $n \leftarrow$  blok zawierający wierzchołek korzenia B+-drzewa;
wczytaj blok  $n$ ; wyczyść stos  $S$ ;
while ( $n$  nie jest wierzchołkiem liścia B+-drzewa) do
  begin
    odłóż adres bloku  $n$  na stosie  $S$ ;
    (* stos  $S$  zawiera wierzchołki nadrzędne, które są potrzebne w razie dokonywania
    podziału *)
     $q \leftarrow$  liczba wskaźników drzewa występujących w wierzchołku  $n$ ;
    if  $K \leq n.K_1$  (*  $n.K_1$  oznacza odwołanie do  $i$ -tej wartości pola wyszukiwania
    w wierzchołku  $n$  *)
      then  $n \leftarrow n.P_1$  (*  $n.P_1$  oznacza odwołanie do  $i$ -tego wskaźnika drzewa
      w wierzchołku  $n$  *)
    else if  $K > n.K_{q-1}$ 
      then  $n \leftarrow n.P_q$ 
      else begin
        wyszukaj wierzchołek  $n$  dla wpisu  $i$ , taki że  $n.K_{i-1} < K \leq n.K_i$ ;
         $n \leftarrow n.P_i$ 
      end;
    wczytaj blok  $n$ 
  end;
przeszukaj blok  $n$  względem wpisu ( $K_i, Pr_i$ ) dla  $K = K_i$ ; (* wyszukaj wierzchołek
liścia  $n$  *)
if znaleziono
  then rekord już w pliku – nie można wstawić
else (* wstaw wpis do B+-drzewa wskazujący na rekord *)
  begin
    utwórz wpis ( $K, Pr$ ), gdzie  $Pr$  wskazuje na nowy rekord;
    if wierzchołek liścia  $n$  nie jest pełny
      then wstaw wpis ( $K, Pr$ ) na odpowiedniej pozycji w wierzchołku liścia  $n$ 
    else
      begin (* wierzchołek liścia  $n$  jest pełny i zawiera  $p_{liść}$  wskaźników na rekordy –
      dzielimy *)
        skopiuj  $n$  do  $temp$  (*  $temp$  jest rozszerzonym wierzchołkiem liścia umożliwiającym
        przechowanie dodatkowych wpisów *);
        wstaw wpis ( $K, Pr$ ) do  $temp$  na odpowiedniej pozycji;
        (* teraz  $temp$  zawiera  $p_{liść}+1$  wpisów postaci ( $K_i, Pr_i$ ) *)
         $new \leftarrow$  nowy pusty wierzchołek liścia drzewa;  $new.P_{nast} \leftarrow n.P_{nast}$ ;
         $j \leftarrow \lceil (p_{liść}+1)/2 \rceil$ ;
         $n \leftarrow$  pierwsze  $j$  wpisów w  $temp$  (aż do wpisu ( $K_j, Pr_j$ ));  $n.P_{nast} \leftarrow new$ ;
         $new \leftarrow$  pozostałe wpisy w  $temp$ ;  $K \leftarrow K_j$ ;
        (* teraz musimy przenieść wpis ( $K, New$ ) i wstawić do nadrzędnego wierzchołka
        wewnętrznego, jednak jeżeli wierzchołek nadrzędny będzie pełny, podział może
        ulec propagacji *)
        finished  $\leftarrow$  false;

```

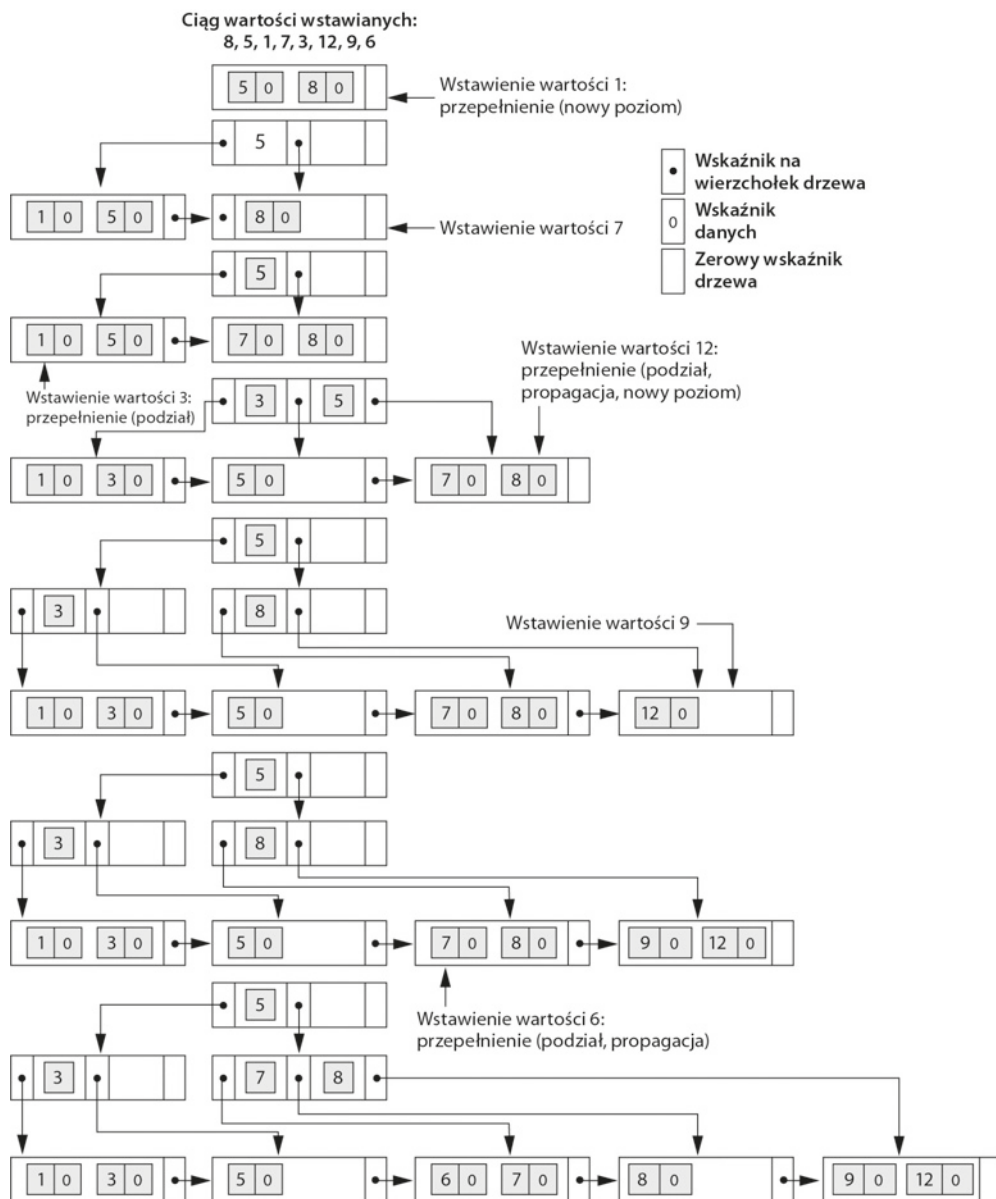


```

repeat
if stos S jest pusty
  then (* brak wierzchołka nadrzędnego; tworzony jest nowy korzeń drzewa *)
  begin
    root ← nowy pusty wierzchołek wewnętrzny drzewa;
    root ← <n, K, new>; finished ← true;
  end;
else
  begin
    n ← pobierz wartość ze stosu S;
    if wierzchołek wewnętrzny n nie jest pełny
    then
      begin (* wierzchołek nadrzędny nie jest pełny – bez podziału *)
        wstaw (K, new) na odpowiedniej pozycji w wierzchołku wewnętrznym n;
        finished ← true;
      end
    else
      begin (* wierzchołek wewnętrzny n jest zapełniony p wskaźnikami drzewa;
        wystąpił warunek przepełnienia; wykonujemy podział *)
        skopiuj n do temp (* temp jest rozszerzonym wierzchołkiem wewnętrznym *);
        wstaw (K, new) do temp na odpowiedniej pozycji;
        (* temp posiada teraz p+1 wskaźników drzewa *)
        new ← nowy pusty wierzchołek wewnętrzny drzewa;
        j ← ⌊((p+1)/2)⌋;
        n ← wpisy aż do wskaźnika drzewa Pj w temp;
        (* n zawiera <P1, K1 P2, K2, ..., Pj-1, Kj-1, Pj> *)
        new ← wpisy od wskaźnika drzewa Pj+1 w temp;
        (* new zawiera <Pj+1, Kj+1, ..., Kp-1, Pp, Kp, Pp+1> *)
        K ← Kj
        (* teraz musimy przenieść wpis (K, new) i wstawić go do nadrzędnego
        wierzchołka wewnętrznego *)
      end;
    end;
  until finished
end;
end;

```

Rysunek 17.12 ilustruje operację wstawiania rekordów do B<sup>+</sup>-drzewa rzędu  $p = 3$  oraz  $p_{\text{liść}} = 2$ . Po pierwsze, możemy zauważyć, że korzeń jest jedynym wierzchołkiem w drzewie, więc jest również liściem. Kiedy zostanie utworzony dodatkowy poziom, drzewo jest dzielone na wierzchołki wewnętrzne i wierzchołki liści. Należy zauważyć, że *każda wartość klucza musi występować na poziomie liści*, ponieważ wszystkie wskaźniki danych znajdują się na tym poziomie. Jednak tylko niektóre wartości występują w wierzchołkach wewnętrznych i sterują wyszukiwaniem. Należy również zauważyć, że *każda wartość występująca w wierzchołku wewnętrznym występuje także jako pierwsza od prawej strony wartość na poziomie liści poddrzewa*, na które wskazuje wskaźnik drzewa znajdujący się po lewej stronie wartości.



RYSUNEK 17.12. Przykład operacji wstawiania do B+-drzewa o parametrach  $p = 3$  oraz  $p_{liść} = 2$

Kiedy *wierzchołek liścia* jest pełny i wstawiamy nowy wpis, wierzchołek ulega *przepełnieniu* (ang. *overflow*) i musi zostać podzielony. Pierwsze  $j = \lceil ((p_{liść} + 1) / 2) \rceil$  wpisów z oryginalnego wierzchołka zostaje w nim zachowane, a pozostałe wpisy zostają przeniesione do nowego wierzchołka liścia.  $j$ -ta wartość wyszukiwania zostaje powielona w nadrzędnym rekordzie wewnętrznym i tworzy się w nim dodatkowy wskaźnik na nowy wierzchołek.

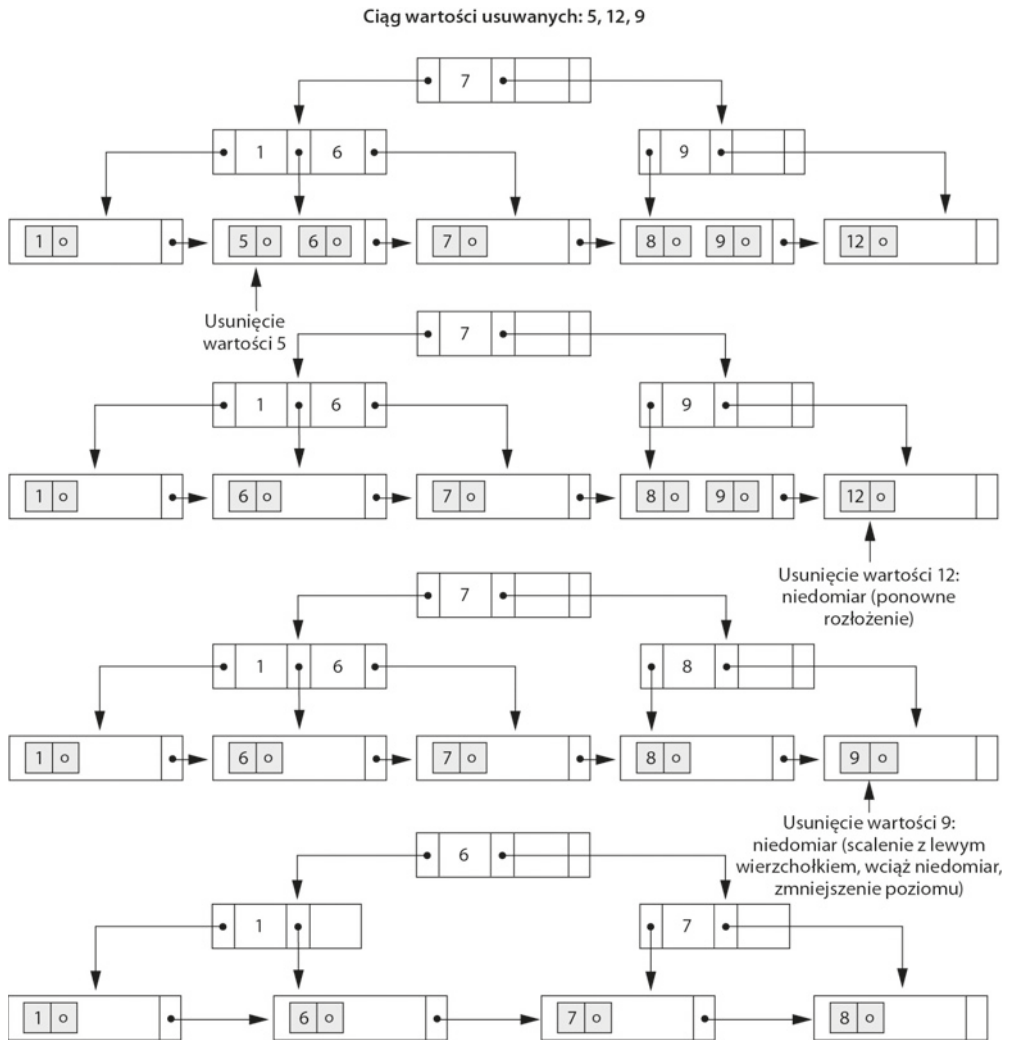
Wartości te muszą zostać wstawione do wierzchołka nadrzędnego w odpowiedniej kolejności. Jeżeli nadrzędny wierzchołek wewnętrzny jest pełny, nowa wartość powoduje również jego przepełnienie, więc także on musi zostać podzielony. Wpisy wierzchołka wewnętrznego aż do  $P_j$  —  $j$ -tego wskaźnika drzewa po wstawieniu nowej wartości i wskaźnika, gdzie  $j = \lfloor ((p+1)/2) \rfloor$  — są zachowywane, natomiast  $j$ -ta wartość wyszukiwania zostaje *przeniesiona*, a nie powielona, do wierzchołka nadrzędnego. Nowy wierzchołek wewnętrzny zawiera wpisy od  $P_{j+1}$  do końca wpisów wierzchołka (patrz algorytm 17.3). Operacja podziału może ulegać propagacji aż do korzenia, co powoduje utworzenie nowego wierzchołka korzenia i nowego poziomu  $B^+$ -drzewa.

Rysunek 17.13 ilustruje operacje usuwania z  $B^+$ -drzewa. Kiedy wpis jest usuwany, zawsze usuwa się go na poziomie liści. Jeżeli występuje on również w wierzchołku wewnętrznym, także stamtąd musi zostać usunięty. W takim przypadku wartość znajdująca się na lewo w wierzchołku liścia musi zastąpić bieżącą w wierzchołku wewnętrznym, ponieważ stanie się ona pierwszym od strony prawej wpisem w poddrzewie. Usuwanie może spowodować wystąpienie **niedomiaru** (ang. *underflow*) poprzez zredukowanie liczby wpisów w wierzchołku liścia poniżej wymaganego minimum. W takim przypadku próbujemy znaleźć **siostrzany** (ang. *sibling*) wierzchołek liścia — wierzchołek liścia znajdujący się bezpośrednio na lewo lub na prawo od wierzchołka z niedomiarem — i **ponownie rozłożyć** (ang. *redistribute*) wpisy pomiędzy wierzchołek a jego wierzchołek **siostrzany**, tak aby oba były zapełnione co najmniej w połowie. W przeciwnym razie wierzchołek zostaje scalony ze swoimi wierzchołkami siostrzanymi i liczba liści zostaje zmniejszona. Często stosowana metoda polega na podjęciu próby **ponownego rozłożenia** wpisów z użyciem lewego wierzchołka siostrzanego. Jeżeli jest to niemożliwe, dokonuje się próby ponownego rozłożenia z użyciem prawego wierzchołka siostrzanego. Jeżeli i to nie jest możliwe, trzy wierzchołki zostają scalone w dwa wierzchołki liści. W takim przypadku niedomiar może się propagować na wierzchołki **wewnętrzne**, ponieważ potrzebny jest o jeden wskaźnik drzewa oraz wartość wyszukiwania mniej. Może to powodować propagację i zmniejszenie liczby poziomów drzewa.

Należy zauważyć, że implementacja algorytmów wstawiania i usuwania może wymagać w przypadku każdego wierzchołka dostępu do wskaźników na wierzchołki nadrzędne oraz siostrzane lub użycia stosu, tak jak w algorytmie 17.3. Każdy wierzchołek powinien również zawierać liczbę zawartych w nim wpisów oraz swój typ (liść lub wewnętrzny). Inne rozwiązanie polega na zaimplementowaniu wstawiania i usuwania jako procedur rekurencyjnych<sup>13</sup>.

**Odmiany B-drzew i  $B^+$ -drzew.** Na zakończenie niniejszego rozdziału pokrótce omówimy istniejące odmiany B-drzew i  $B^+$ -drzew. W pewnych sytuacjach ograniczenie 5. dla B-drzewa (i dla wierzchołków wewnętrznych  $B^+$ -drzewa; wyjątkiem jest tu korzeń), które wymaga, aby każdy wierzchołek był co najmniej w połowie zapełniony, można zmienić tak, aby każdy wierzchołek był zapełniony co najmniej w dwóch trzecich. W takim przypadku

<sup>13</sup> Więcej szczegółów na temat algorytmów wstawiania i usuwania dla  $B^+$ -drzew znajdziesz w Ramakrishnan i Gehrke (2003).

RYSUNEK 17.13. Przykład operacji usuwania z B<sup>+</sup>-drzewa

B drzewo określa się mianem **B\*-drzewa** (ang. *B\*-tree*). Ogólnie rzecz biorąc, niektóre systemy pozwalają użytkownikowi na wybranie **współczynnika zapelnienia** (ang. *fill factor*) o wartości między 0,5 a 1,0, gdzie ta ostatnia oznacza, że wierzchołki B-drzewa (indeksu) mają być wypełnione w całości. Istnieje również możliwość określenia dwóch współczynników zapelnienia dla B<sup>+</sup>-drzewa: jednego dla poziomu liści i drugiego dla wierzchołków wewnętrznych. W momencie konstruowania indeksu każdy wierzchołek jest

wypełniany do mniej więcej poziomu określonego współczynnikiem zapełnienia. Część badaczy sugeruje, że warto złagodzić wymóg, aby każdy wierzchołek był zapełniony w połowie, i zamiast tego dopuszczać jego całkowite opróżnienie przed wykonaniem scale'nia, co upraszcza algorytm usuwania. Przeprowadzone symulacje pokazują, że nie powoduje to marnowania zbyt dużej ilości dodatkowej przestrzeni w przypadku losowego rozkładu operacji wstawiania i usuwania.

## 17.4. Indeksy na wielu kluczach

W dotychczasowej dyskusji zakładaliśmy, że klucze główne lub drugorzędne, poprzez które uzyskiwaliśmy dostęp do plików, były pojedynczymi atrybutami (polami). W wielu żądaniach związanych z pobieraniem lub aktualizowaniem danych uwzględnia się jednak wiele atrybutów. Jeżeli określona kombinacja atrybutów jest używana bardzo często, warto opracować strukturę dostępową w celu zapewnienia wydajnego dostępu względem wartości klucza, który jest połączeniem takich atrybutów.

Przykładowo, weźmy pod uwagę plik PRACOWNIK zawierający atrybuty NRDZ (numer działu), WIEK, ULICA, MIASTO, KODPOCZT, PENSJA oraz KOD\_SPECJALNOŚCI z kluczem PESEL. Rozważmy zapytanie: *wyświetl listę pracowników z działu o numerze 4, których wiek wynosi 59. Należy zauważyć, że ani atrybut NRDZ, ani WIEK nie jest atrybutem klucza, co oznacza, że wartość wyszukiwania dla każdego z nich wskazuje na wiele rekordów. Można uwzględnić następujące alternatywne strategie wyszukiwania:*

- (1) Zakładając, że pole NRDZ posiada indeks, a pole WIEK nie, uzyskujemy dostęp do rekordów posiadających wartość NRDZ = 4 przy użyciu indeksu, a następnie wybieramy spośród nich te rekordy, które spełniają warunek WIEK = 59.
- (2) Jeżeli pole WIEK jest indeksowane, a pole NRDZ nie, uzyskujemy dostęp do rekordów posiadających wartość WIEK = 59 przy użyciu indeksu, a następnie wybieramy spośród nich te rekordy, które spełniają warunek NRDZ = 4.
- (3) Jeżeli indeksy utworzono na obu polach NRDZ i WIEK, można wykorzystać oba. Każdy daje w wyniku zbiór rekordów lub zbiór wskaźników (na bloki lub rekordy). Przecięcie tych zbiorów rekordów lub wskaźników daje w wyniku rekordy lub wskaźniki, które spełniają oba warunki.

Wszystkie powyższe rozwiązania zwracają poprawne wyniki. Jednak jeżeli zbiory rekordów spełniających każdy warunek (NRDZ = 4 oraz WIEK = 59) oddzielnie są duże, a tylko kilka rekordów spełnia oba, wówczas żadna z tych technik nie zapewnia odpowiedniej wydajności działania. Warto też zauważyć, że zapytania takie jak „znajdź minimalny lub maksymalny wiek wśród wszystkich pracowników” można obsłużyć za pomocą samego indeksu pola WIEK, bez używania pliku danych. Jednak ustalenie maksymalnego lub minimalnego wieku dla warunku NRDZ = 4 za pomocą samego indeksu jest niemożliwe. Sam indeks nie wystarczy też do wyświetlenia wszystkich działów zatrudniających pracowników w wieku 59 lat. Istnieje wiele możliwości zapewnienia, aby kombinacja <NRDZ, WIEK> lub <WIEK, NRDZ> była traktowana jako klucz wyszukiwania składający się z wielu atrybutów. Poniżej pokrótce zostaną opisane takie techniki. Klucze zawierające wiele atrybutów będziemy określać mianem **kluczy złożonych** (ang. *composite keys*).

### 17.4.1. Indeks uporządkowany na wielu atrybutach

Dyskusja przedstawiona dotąd w niniejszym rozdziale ma zastosowanie także w przypadku, gdy utworzymy indeks na polu klucza wyszukiwania, które jest kombinacją  $\langle \text{NRDZ}, \text{WIEK} \rangle$ . W przedstawionym powyżej przykładzie klucz wyszukiwania jest parą wartości  $\langle 4, 59 \rangle$ . Ogólnie rzecz biorąc, jeżeli indeks zostanie utworzony na atrybutach  $\langle A_1, A_2, \dots, A_n \rangle$ , to wartości klucza wyszukiwania będą krotkami o  $n$  wartościach  $\langle v_1, v_2, \dots, v_n \rangle$ .

Leksykograficzne uporządkowanie wartości tych krotek określa porządek takiego złożonego klucza wyszukiwania. W omawianym przykładzie wszystkie klucze działów dla działu o numerze 3 poprzedzają klucze dla działu o numerze 4. Zatem  $\langle 3, n \rangle$  poprzedza  $\langle 4, m \rangle$  dla dowolnych wartości  $m$  i  $n$ . Rosnący porządek wartości klucza dla kluczy z  $\text{NRDZ} = 4$  dawałby:  $\langle 4, 18 \rangle$ ,  $\langle 4, 19 \rangle$ ,  $\langle 4, 20 \rangle$  itd. Uporządkowanie leksykograficzne działa podobnie do uporządkowania ciągów znaków. Indeks na kluczu złożonym o  $n$  atrybutach działu podobnie do innych indeksów omówionych dotąd w bieżącym rozdziale.

### 17.4.2. Mieszanie partycjonowane

Mieszanie partycjonowane stanowi rozszerzenie statycznego mieszania zewnętrznego (podrozdział 16.8.2) i pozwala na uzyskiwanie dostępu względem wielu kluczy. Jest ono odpowiednie wyłącznie w przypadku porównań równościowych. Zapytania zakresowe nie są obsługiwane. W przypadku mieszania partycjonowanego dla klucza składającego się z  $n$  komponentów funkcja mieszająca jest projektowana tak, aby zwracała wynik w  $n$  odrębnych adresach mieszania. Adres pakietu jest konkatenacją tych  $n$  adresów. Istnieje wówczas możliwość wyszukiwania wymaganego złożonego klucza poprzez przeglądanie pakietów odpowiadających częściom adresu, którym jesteśmy zainteresowani.

Przykładowo, weźmy pod uwagę złożony klucz wyszukiwania  $\langle \text{NRDZ}, \text{WIEK} \rangle$ . Jeżeli dla wartości  $\text{NRDZ}$  i  $\text{WIEK}$  utworzymy skróty w postaci adresów 3- i 5-bitowych, otrzymamy 8-bitowy adres pakietu. Założmy, że  $\text{NRDZ} = 4$  posiada adres mieszający 100, zaś  $\text{WIEK} = 59$  posiada adres mieszający 10101. W takiej sytuacji w celu wyszukania połączonej wartości wyszukiwania,  $\text{NRDZ} = 4$  oraz  $\text{WIEK} = 59$ , odwołujemy się do adresu pakietu 100 10101. W celu wyszukania tylko pracowników z atrybutem  $\text{WIEK} = 59$  należy przeszukać wszystkie pakiety (osiem), których adresami są 000 10101, 001 10101, ... itd. Korzyścią wynikającą z mieszania partycjonowanego jest to, że można je z łatwością rozszerzać na dowolną liczbę atrybutów. Adresy pakietów można zdefiniować tak, aby bardziej znaczące bity adresów odpowiadały atrybutom, do których częściej uzyskuje się dostęp. Główną wadą mieszania partycjonowanego jest to, że nie obsługuje ono zapytań zakresowych dla żadnych atrybutów składowych. Ponadto większość funkcji mieszających nie zachowuje rekordów zgodnie z kolejnością klucza używanego do mieszania. Dlatego dostęp do rekordów w porządku leksykograficznym na podstawie kombinacji atrybutów takich jak  $\langle \text{NRDZ}, \text{WIEK} \rangle$  używanych jako klucz nie jest ani prosty, ani wydajny.

### 17.4.3. Pliki matrycowe

Kolejnym rozwiązaniem jest zapewnienie organizacji pliku `PRACOWNIK` jako **pliku matrycowego** (ang. *grid file*). Jeżeli chcemy uzyskać dostęp do pliku na podstawie dwóch kluczy, na przykład  $\text{NRDZ}$  i  $\text{WIEK}$ , możemy skonstruować **matrycę** (ang. *grid array*) o jednej skali liniowej

(wymiarze) dla każdego z atrybutów wyszukiwania. Na rysunku 17.14 przedstawiono matrycę dla pliku PRACOWNIK z jedną skalą liniową dla atrybutu NRZDZ i drugą dla atrybutu WIEK. Skale tworzy się tak, aby osiągnąć jednorodny rozkład atrybutu. Zatem w prezentowanym przykładzie okazuje się, że skala liniowa dla atrybutu NRZDZ posiada wartości NRZDZ = 1, 2 połączone jako jedna wartość 0, zaś wartość NRZDZ = 5 odpowiada wartości 2 na tej skali. Podobnie, WIEK jest dzielony na swojej skali między wartości od 0 do 5 poprzez pogrupowanie zakresów wieku w taki sposób, aby otrzymać jednorodny rozkład pracowników względem ich wieku. Matryca przedstawiona dla tego pliku posiada w sumie 36 komórek. Każda komórka wskazuje na pewien adres pakietu, gdzie są przechowywane rekordy odpowiadające danej komórce. Na rysunku 17.14 przedstawiono również przypisanie komórek do pakietów (tylko częściowo).



RYСУNEK 17.14. Przykład matrycy na atrybutach NRZDZ i WIEK

Zatem nasze żądanie z warunkami NRZDZ = 4 oraz WIEK = 59 jest odwzorowywane na komórkę (1, 5) odpowiadającą matrycy. Rekordy dla tej kombinacji zostaną znalezione w odpowiednim pakiecie. Metoda ta jest szczególnie przydatna w przypadku zapytań zakresowych, które są odwzorowywane na zbiór komórek odpowiadających grupie wartości zgodnych ze skalami liniowymi. Jeśli zapytanie zakresowe polega na dopasowaniu warunku do komórek matrycy, można je przetworzyć za pomocą dostępu do pakietów odpowiadających tym komórkom. Przykładowo, zapytanie dla warunków NRZDZ  $\leq 5$  i WIEK  $> 40$  odpowiada danym z górnego pakietu z rysunku 17.14.

Z koncepcyjnego punktu widzenia, pojęcie pliku matrycowego może być stosowane względem dowolnej liczby kluczy wyszukiwania. W przypadku  $n$  kluczy wyszukiwania matryca posiada  $n$  wymiarów. Stąd matryca pozwala na podział pliku zgodnie z wymiarami atrybutów klucza wyszukiwania i zapewnia uzyskiwanie dostępu na podstawie kombinacji wartości z tych wymiarów. Pliki matrycowe sprawdzają się dobrze pod względem zmniejszenia czasu związanego z dostępem na podstawie wielu kluczy. Jednakże, wnoszą one narzut przestrzenny związany ze strukturą matrycy. Ponadto w przypadku plików dynamicznych koszty konserwacji wzrastają w związku z potrzebą częstych reorganizacji pliku<sup>14</sup>.

<sup>14</sup> Algorytmy wstawiania i usuwania dla plików matrycowych można znaleźć w pozycji Nievergelta i in. (1984).



## 17.5. Inne rodzaje indeksów

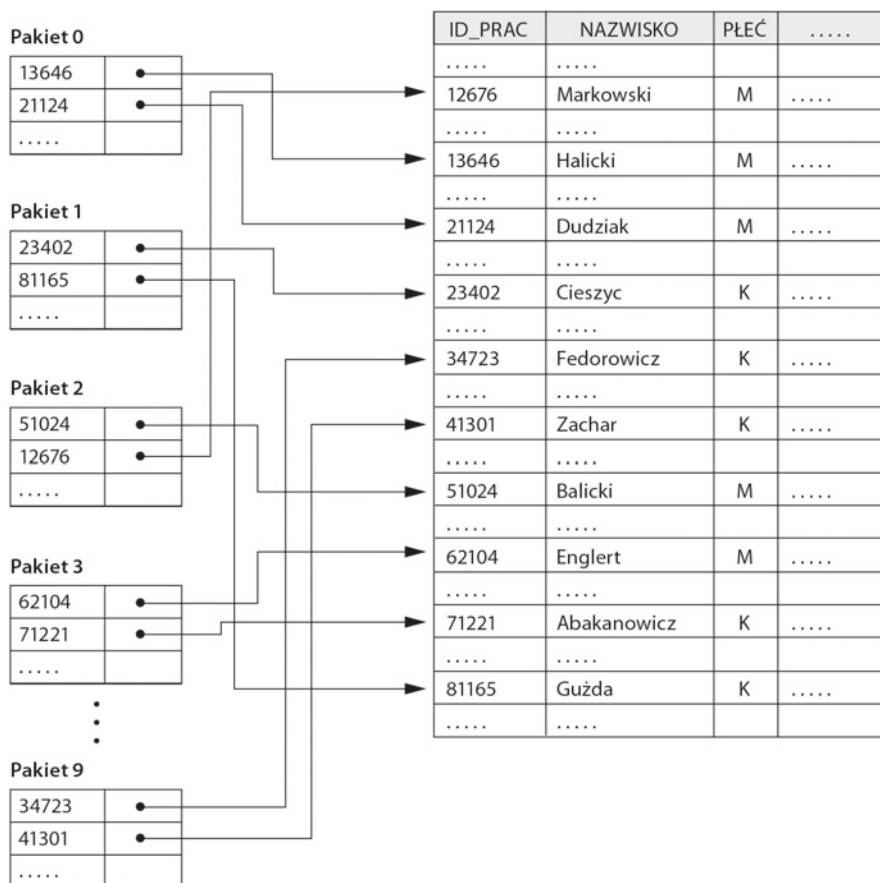
### 17.5.1. Indeksy oparte na mieszaniu

Istnieje również możliwość utworzenia struktur dostępowych podobnych do indeksów, opartych na technice *mieszania*. **Indeks oparty na mieszaniu** (ang. *hash index*) to struktura pomocnicza służąca do dostępu do plików na podstawie mieszania z użyciem klucza wyszukiwania innego niż ten, który określa podstawową organizację pliku danych. Wpisy indeksu mają postać  $\langle K, Pr \rangle$  (lub  $\langle K, P \rangle$ ), gdzie  $Pr$  to wskaźnik do rekordu zawierającego dany klucz, a  $P$  to wskaźnik do bloku obejmującego rekord z danym kluczem. Plik indeksu z tymi wpisami można organizować jako dynamicznie rozszerzający się plik przy użyciu jednej z technik opisanych w podrozdziale 16.8.3. Wyszukując wpis, korzystamy z mieszającego algorytmu wyszukiwania względem wartości  $K$ . Po znalezieniu wpisu wskaźnik  $Pr$  (lub  $P$ ) służy do zlokalizowania odpowiedniego rekordu w pliku danych. Na rysunku 17.15 przedstawiono oparty na mieszaniu indeks na polu `ID_PRAC`; jest to indeks pliku sekwencyjnego uporządkowanego według pola `IMIĘNAZWISKO`. Do pola `ID_PRAC` zastosowano podział na pakiety za pomocą funkcji mieszającej (suma cyfr pola `ID_PRAC` dzielona modulo przez 10). Przykładowo, w trakcie wyszukiwania `ID_PRAC` o wartości 51024 funkcja mieszająca zwraca numer pakietu 2. Najpierw należy uzyskać dostęp do tego pakietu. Zawiera on wpisy indeksu  $\langle 51024, Pr \rangle$ . Wskaźnik  $Pr$  prowadzi do konkretnego rekordu w pliku. W praktycznych zastosowaniach pakietów mogą być tysiące. Numer pakietu może zajmować kilka bitów długości i stosuje się do niego schematy oparte na katalogu, opisane w kontekście mieszania dynamicznego w punkcie 16.8.3. Jako indeksów można również używać innych struktur wyszukiwania.

### 17.5.2. Indeksy bitmapowe

**Indeks bitmapowy** to następna popularna struktura danych ułatwiająca obsługę zapytań wykorzystujących kilka kluczy. Indeksowanie bitmapowe jest stosowane do relacji zawierających dużą liczbę wierszy. W tej technice tworzony jest indeks na jednej lub kilku kolumnach, a wszystkie wartości lub zakresy wartości z tych kolumn są uwzględniane w indeksie. Indeks bitmapowy jest zwykle tworzony dla kolumn obejmujących stosunkowo niewielką liczbę unikatowych wartości. Aby zbudować indeks bitmapowy oparty na zbiorze rekordów z relacji, rekordy należy ponumerować od 0 do  $n$  za pomocą identyfikatora (rekordu lub wiersza), który można odwzorować na adres fizyczny obejmujący numer bloku i pozycję rekordu w tym bloku.

Indeks bitmapowy jest budowany dla **jednej konkretnej wartości** danego pola (kolumny relacji) i ma postać tablicy bitów. Dlatego dla danego pola utrzymywany jest jeden odrębny indeks bitmapowy (wektor), odpowiadający każdej unikatowej wartości z bazy. Wyobraź sobie indeks bitmapowy dla kolumny  $C$  i wartości  $V$  z tej kolumny. Gdy relacja obejmuje  $n$  wierszy, indeks ma  $n$  bitów. Bit  $i$ -ty ma wartość 1, jeśli w wierszu  $i$  kolumna  $C$  ma wartość  $V$ . W przeciwnym razie bit ma wartość 0. Jeśli  $C$  obejmuje zbiór  $m$  różnych wartości  $\langle v_1, v_2, \dots, v_m \rangle$ , należy utworzyć dla niej  $m$  indeksów bitmapowych. Na rysunku 17.16 pokazana jest relacja `PRACOWNIK` z kolumnami `ID_PRAC`, `NAZWISKO`, `PŁEĆ`, `KOD` i `POZIOM_PENSJI`, a także indeks bitmapowy dla kolumn `PŁEĆ` i `KOD`. W przykładowej bitmapie dla pola `PŁEĆ` i wartości  $K$  bity dla wierszy 1., 3., 4. i 7. są równe 1, a pozostałe to 0. Indeksy bitmapowe mogą mieć następujące zastosowania w zapytaniach:



RYSUNEK 17.15. Indeksowanie oparte na mieszaniu

- W zapytaniu  $C_1 = V_1$  bitmapa dla wartości  $V_1$  zwraca identyfikatory wierszy spełniających warunek.
- W zapytaniach  $C_1 = V_1$  i  $C_2 = V_2$  (wyszukiwanie z użyciem wielu kluczy) pobierane są dwie odpowiednie bitmapy i ustalana jest ich część wspólna (logiczne I), aby uzyskać zbiór identyfikatorów wierszy spełniających warunek. Aby uwzględnić warunki z  $k$  równościami, można wyznaczyć część wspólną dla  $k$  wektorów bitowych. Za pomocą indeksów bitmapowych można też obsługiwać złożone warunki I-LUB.
- W zapytaniach  $C_1 = V_1$  lub  $C_2 = V_2$ , lub  $C_3 = V_3$  (wyszukiwanie z użyciem wielu kluczy) należy pobrać trzy odpowiednie bitmapy dla trzech różnych atrybutów i wyznaczyć ich sumę (logiczne LUB), aby uzyskać zbiór identyfikatorów wierszy spełniających warunek.
- Aby pobrać liczbę wierszy spełniających warunek  $C_1 = V_1$ , należy zliczyć wpisy z odpowiedniego wektora bitowego mające wartość 1.
- Zapytania z negacją, np.  $C_1 \neg = V_1$ , można obsługiwać, wyznaczając logiczne dopełnienie odpowiedniej bitmapy.

## PRACOWNIK

ID_WIERSZA	ID_PRAC	NAZWISKO	PŁEĆ	KOD_PO CZ	POZIOM_PENSIJ
0	51024	Balicki	M	94040	..
1	23402	Cieszy c	K	30022	..
2	62104	Englert	M	19046	..
3	34723	Fedorowicz	K	30022	..
4	81165	Guzda	K	19046	..
5	13646	Halicki	M	19046	..
6	12676	Markowski	M	30022	..
7	41301	Zachar	K	94040	..

## Indeks bitmapowy dla pola PŁEĆ

M                      K  
10100110    01011001

## Indeks bitmapowy dla pola KOD\_PO CZ

KOD\_PO CZ 19046    KOD\_PO CZ 30022    KOD\_PO CZ 94040  
00101100                      01010010                      10000001

RYSUNEK 17.16. Indeksy bitmapowe uwzględniające płeć i kod pocztowy

Przyjrzyj się np. relacji PRACOWNIK z rysunku 17.16 z indeksami bitmapowymi PŁEĆ i KOD. Aby znaleźć pracowników z danymi PŁEĆ = K i KOD = 30022, należy wyznaczyć część wspólną dla bitmap 01011001 i 01010010, co daje identyfikatory wierszy 1. i 3. Pracowników, którzy nie mają kodu 94040, można uzyskać za pomocą dopełnienia wektora bitowego 10000001, co daje identyfikatory wierszy od 1. do 6. Jeśli założymy jednorodny rozkład wartości w danej kolumnie i przyjmiemy, że jedna kolumna ma pięć różnych wartości, a druga dziesięć, warunek złączenia tych dwóch kolumn będzie miał selektywność równą  $1/50$  ( $1/5 * 1/10$ ). Oznacza to, że trzeba pobrać tylko ok. 2% rekordów. Jeśli kolumna przyjmuje niewiele wartości, tak jak kolumna PŁEĆ na rysunku 17.16, warunek PŁEĆ = M zwróci zwykle 50% wierszy. W takich sytuacjach lepiej jest przeprowadzić kompletne skanowanie, niż posługiwać się indeksami bitmapowymi.

Indeksy bitmapowe są zwykle wydajne, jeśli chodzi o zajmowane przez nie miejsce. Jeśli plik zawiera milion wierszy (rekordów) o wielkości po 100 bajtów, każdy indeks bitmapowy zajmuje tylko jeden bit na wiersz, a więc wymaga miliona bitów (125 kilobajtów). Załóżmy, że relacja dotyczy miliona mieszkańców regionu posługujących się w sumie 200 kodami pocztowymi. Dwieście bitmap dla atrybutu KOD zajmuje 200 bitów (25 bajtów) na wiersz. Tak więc 200 bitmap zajmuje tylko 25% pamięci potrzebnej na plik danych, a pozwala precyzyjnie pobrać wszystkich mieszkańców obszaru o danym kodzie pocztowym i zwrócić odpowiednie identyfikatory wierszy.

Gdy rekordy są usuwane, zmiana numeracji wierszy i przesuwanie bitów w bitmapach są kosztowne. Aby uniknąć tych kosztów, można zastosować inną bitmapę — **bitmapę istnienia** (ang. *existence bitmap*). Zawiera ona bit 0 dla wierszy, które zostały usunięte, ale wciąż fizycznie istnieją, i 1 dla rzeczywiście istniejących wierszy. Gdy wiersz jest wstawiany do relacji, trzeba dodać wpis do wszystkich bitmap dla wszystkich kolumn, dla których istnieje indeks bitmapowy. Wiersze zwykle są dołączane do relacji, ale mogą też zastępować usunięte wiersze, aby zminimalizować koszty reorganizacji bitmap. Jednak ten proces i tak oznacza dodatkowe koszty związane z indeksowaniem.

Duże wektory bitowe są traktowane jako serie wektorów 32- lub 64-bitowych. Używane są przy tym odpowiednie operatory I, LUB i NIE, które przetwarzają 32- lub 64-bitowe wektory wejściowe za pomocą pojedynczej instrukcji. Dzięki temu operacje na wektorach bitowych są bardzo wydajne obliczeniowo.

**Bitmapy dla liści B+-drzew.** Bitmapy można stosować do liści indeksów opartych na B+-drzewach, a także do określania zbiorów rekordów zawierających konkretną wartość indeksowanego pola w liściu. Gdy B+-drzewo jest budowane na podstawie pola wyszukiwania niebędącego kluczem, liść — obok każdej wartości indeksowanego atrybutu — musi zawierać listę wskaźników rekordów. W przypadku wartości, które występują bardzo często (w dużym procencie relacji), zamiast wskaźników można zapisać indeks bitmapowy. Przykładowo, założmy, że w relacji o  $n$  wierszach wartość występuje w 10% rekordów. Wektor bitowy będzie zajmował  $n$  bitów i zawierał wartość 1 dla identyfikatorów wierszy obejmujących szukaną wartość. Wielkość tego wektora to  $n/8$ , czyli  $0,125n$  bajta. Jeśli wskaźnik rekordu zajmuje 4 bajty (32 bity),  $n/10$  wskaźników rekordów zajmie  $4 * n/10$ , czyli  $0,4n$  bajta. Ponieważ  $0,4n$  to ponad trzykrotnie więcej niż  $0,125n$ , lepiej jest przechowywać indeks bitmapowy niż wskaźniki rekordów. Dlatego dla szukanых wartości występujących częściej niż określony poziom (tu wynosi on  $1/32$ ) warto zastosować bitmapy jako skompresowany mechanizm składowania, który reprezentuje wskaźniki rekordów w B+-drzewach indeksujących pole niebędące kluczem.

### 17.5.3. Indeksowanie oparte na funkcji

W tym punkcie opiszemy nowy sposób indeksowania — **indeksowanie oparte na funkcji**, wprowadzone w relacyjnych SZBD firmy Oracle, a także w niektórych innych produktach komercyjnych<sup>15</sup>.

Podstawą tej techniki jest utworzenie indeksu w taki sposób, by wartość wykonania pewnej funkcji na polu lub na zbiorze pól była kluczem tego indeksu. W przykładach pokażemy, jak tworzyć i stosować takie indeksy.

**Przykład 1.** Następną instrukcją tworzy oparty na funkcji indeks dla tabeli PRACOWNIK. Ten indeks jest oparty na zapisanych wielkimi literami wartościach kolumny NAZWISKO. Wartości te mogą być wpisywane w dowolny sposób, ale w zapytaniach zawsze używany jest zapis z wielkimi literami.

```
CREATE INDEX INDEKS_WIELKIE ON PRACOWNIK (UPPER(NAZWISKO));
```

Ta instrukcja tworzy indeks oparty na funkcji UPPER(NAZWISKO), zwracającej nazwisko zapisane wielkimi literami. Na przykład wywołanie UPPER('Kował') zwraca wartość 'KOWAŁ'.

Indeksy oparte na funkcjach gwarantują, że system bazodanowy firmy Oracle użyje indeksu, zamiast przeprowadzać pełne skanowanie tabeli, nawet w sytuacji, gdy w predykcji wyszukiwania w zapytaniu używana jest funkcja. Przykładowo, dla poniższego zapytania zostanie użyty przedstawiony indeks:

```
SELECT IMIĘ, NAZWISKO
FROM   PRACOWNIK
WHERE  UPPER(NAZWISKO)= 'KOWAŁ'
```

<sup>15</sup> Autorem większości tego podrozdziału jest Rafi Ahmed.

Bez indeksu opartego na funkcji baza Oracle może wykonać pełne skanowanie tabeli, ponieważ indeks B+-drzewa jest przeszukiwany tylko bezpośrednio na podstawie wartości kolumny. Zastosowanie do kolumny jakiegokolwiek funkcji sprawia, że nie można korzystać z takiego indeksu.

**Przykład 2.** W tym przykładzie tabela PRACOWNIK zawiera dwa pola: PENSJA i PROWIZJA\_PROC, a indeks jest tworzony na podstawie sumy wartości tych pól.

```
CREATE INDEX INDEKS_DOCHÓD
ON PRACOWNIK(PENSJA + (PENSJA*PROWIZJA_PROC));
```

Dla następnego zapytania używany jest indeks INDEKS\_DOCHÓD, przy czym pola PENSJA i PROWIZJA\_PROC występują tu w innej kolejności niż w definicji indeksu.

```
SELECT IMIĘ, NAZWISKO
FROM PRACOWNIK
WHERE ((PENSJA*PROWIZJA_PROC) + PENSJA ) > 15000;
```

**Przykład 3.** Oto bardziej zaawansowany przykład wykorzystujący indeksowanie oparte na funkcjach do definiowania warunkowej unikatowości. Następną instrukcją tworzy dla tabeli ZAMÓWIENIA unikatowy indeks oparty na funkcji. Indeks ten uniemożliwia klientom skorzystanie z identyfikatora promocji („wielkiej wyprzedaży”) więcej niż raz. Instrukcja tworzy indeks złożony na polach ID\_KLIENTA i ID\_PROMOCJI oraz dopuszcza w indeksie tylko jeden wpis dla danego ID\_KLIENTA oraz ID\_PROMOCJI równego 2. W tym celu indeks jest deklarowany jako unikatowy.

```
CREATE UNIQUE INDEX promo_ix ON ZAMÓWIENIA
(CASE WHEN ID_PROMOCJI = 2 THEN ID_KLIENTA ELSE NULL END,
CASE WHEN ID_PROMOCJI = 2 THEN ID_PROMOCJI ELSE NULL END);
```

Warto zauważyć, że dzięki zastosowaniu instrukcji CASE z indeksu usuwane są wszystkie wiersze, w których ID\_PROMOCJI nie jest równy 2. Baza Oracle nie zapisuje w indeksie opartym na B+-drzewie żadnych wierszy, w których wszystkie klucze to wartości puste. Dlatego w tym przykładzie pola ID\_KLIENTA i ID\_PROMOCJI mają wartość NULL, chyba że pole ID\_PROMOCJI to 2. Wskutek tego ograniczenia indeksu są naruszone tylko wtedy, gdy ID\_PROMOCJI jest równy 2 i nastąpi próba wstawienia dwóch wierszy o tej samej wartości ID\_KLIENTA.

## 17.6. Ogólne zagadnienia związane z indeksami

### 17.6.1. Indeksy logiczne a fizyczne

Jak dotąd zakładaliśmy, że wpisy indeksu  $\langle K, Pr \rangle$  (lub  $\langle K, P \rangle$ ) zawsze zawierają wskaźnik fizyczny  $Pr$  (lub  $P$ ), który określa fizyczny adres rekordu na dysku jako numer bloku i przesunięcie. Jest to tak zwany **indeks fizyczny** (ang. *physical index*) i jego wadą jest to, że wskaźnik musi być zmieniony, jeżeli rekord zostanie przeniesiony do innej lokalizacji. Przykładowo, założmy, że podstawowa metoda organizacji pliku bazuje na mieszaniu liniowym lub mieszaniu rozszerzalnym. Wówczas za każdym razem, gdy pakiet jest

dzielony, niektóre rekordy są przydzielane do nowych pakietów, a zatem posiadają nowe adresy fizyczne. Gdyby na pliku utworzono indeks drugorzędny, wskaźniki na te rekordy musiałyby zostać znalezione i zaktualizowane, co stanowi niełatwe zadanie.

W celu poprawy tej sytuacji możemy użyć struktury określanej mianem **indeksu logicznego** (ang. *logical index*), w którym wpisy indeksu mają postać  $\langle K, K_p \rangle$ . Każdy wpis posiada jedną wartość  $K$  dla drugorzędnego pola indeksującego dopasowanego do wartości  $K_p$  pola używanego dla celów podstawowej organizacji pliku. Przeszukując indeks drugorzędny względem wartości  $K$ , program może zlokalizować odpowiednią wartość  $K_p$  i użyć jej w celu uzyskania dostępu do rekordu poprzez wykorzystanie podstawowej metody organizacji pliku (używając indeksu głównego, jeśli jest dostępny). Indeksy logiczne wprowadzają więc dodatkowy poziom pośredni między strukturą dostępową a danymi. Są one używane w przypadku, gdy można oczekiwać, że fizyczne adresy rekordów będą często zmieniane. Koszt wprowadzenia takiego poziomu pośredniego to konieczność wykonywania dodatkowego wyszukiwania w oparciu o metodę podstawowej organizacji pliku.

## 17.6.2. Tworzenie indeksu

W wielu relacyjnych SZBD występuje podobne polecenie tworzenia indeksu, choć nie jest ono częścią standardu SQL. Oto ogólna postać tego polecenia:

```
CREATE [ UNIQUE ] INDEX <nazwa indeksu>
ON <nazwa tabeli> ( <nazwa kolumny> [ <uporządkowanie> ]
{ . <nazwa kolumny> [ <uporządkowanie> ] } )
[ CLUSTER ] ;
```

Słowa kluczowe **UNIQUE** i **CLUSTER** są opcjonalne. Słowo kluczowe **CLUSTER** jest używane, gdy tworzony indeks ma też sortować rekordy z pliku danych według atrybutu indeksującego. Dlatego dodanie słowa kluczowego **CLUSTER** do atrybutu klucza (unikatowego) powoduje powstanie pewnego rodzaju indeksu głównego, a dodanie tego słowa do atrybutu niebędącego kluczem (nieunikatowego) skutkuje utworzeniem odmiany indeksu klastrowania. Wartością opcji *<uporządkowanie>* może być **ASC** (rosnąco) lub **DESC** (malejąco). Opcja ta określa, czy plik danych ma być uporządkowany rosnąco, czy malejąco według wartości atrybutu indeksującego. Ustawienie domyślne to **ASC**. Przykładowo, następująca instrukcja tworzy indeks klastrowania (rosnący) na niebędącym kluczem atrybucie **NRDZ** z pliku **PRACOWNIK**:

```
CREATE INDEX INDEKSNRDZ
ON PRACOWNIK (NRDZ)
CLUSTER;
```

**Proces tworzenia indeksu.** W wielu systemach indeks nie jest integralną częścią pliku danych, ale może być dynamicznie tworzony i usuwany. To dlatego często jest nazywany *strukturą dostępową*. Gdy spodziewasz się częstego dostępu do pliku na podstawie warunku wyszukiwania dotyczącego konkretnego pola, możesz zażądać od SZBD utworzenia na tym polu indeksu (takiego jak **INDEKSNRDZ** wcześniej). Zwykle tworzony jest wtedy indeks drugorzędny, aby uniknąć fizycznego porządkowania rekordów w pliku danych na dysku.



Główną zaletą indeksów drugorzędnych jest to, że (przynajmniej teoretycznie) można je tworzyć w połączeniu z *niemal dowolną podstawową organizacją rekordów*. Dlatego indeks drugorzędny można wykorzystać jako uzupełnienie innych, podstawowych metod dostępu, takich jak porządkowanie lub mieszanie. Można nawet stosować go do plików mieszanych. Gdy utworzysz na polu pliku indeks drugorzędny w postaci B<sup>+</sup>-drzewa, to jeśli plik jest duży i zawiera miliony rekordów, ani plik, ani indeks nie zmieszczą się w pamięci głównej. Wstawianie dużej liczby wpisów do indeksu odbywa się za pomocą **zbiorczego wczytywania** (ang. *bulk loading*) indeksu. Trzeba przetworzyć wszystkie rekordy pliku, aby utworzyć wpisy na poziomie liści drzewa. Te wpisy są następnie sortowane i zapełniane zgodnie z określonym współczynnikiem zapełnienia. Jednocześnie tworzone są inne poziomy indeksu. Dynamiczne tworzenie indeksów głównych i indeksów klastrowania jest dużo bardziej kosztowne i trudne, ponieważ rekordy z pliku danych trzeba fizycznie posortować na dysku zgodnie z kolejnością wyznaczaną przez pole indeksowania. Jednak niektóre systemy umożliwiają użytkownikom dynamiczne tworzenie indeksów dla plików dzięki sortowaniu pliku w trakcie budowania indeksu.

**Indeksowanie ciągów znaków.** Z indeksowaniem ciągów znaków związanych jest kilka problemów. Takie ciągi mogą być zmiennej długości (np. typ danych VARCHAR w języku SQL; patrz rozdział 6.) i mieć zbyt duży rozmiar, co ogranicza zwielokrotnienie wyjściowe. Jeśli indeks oparty na B<sup>+</sup>-drzewie ma korzystać z ciągów znaków jako klucza wyszukiwania, liczba kluczy na wierzchołek indeksu może być różna, co oznacza różne zwielokrotnienie wyjściowe. Niektóre wierzchołki mogą wymagać podziału z powodu zapełnienia i to niezależnie od liczby zapisanych w nich kluczy. Do łagodzenia tej sytuacji służy technika **kompresji z użyciem przedrostków**. Zamiast przechowywać w wierzchołkach pośrednich całe ciągi znaków, można zapisywać tylko przedrostek klucza wyszukiwania, pozwalający odróżniać oddzielne klucze tworzące poddrzewa. Przykładowo, jeśli kluczem wyszukiwania jest NAZWISKO i szukamy osoby o nazwisku Navathe, niebędący liściem wierzchołek może zawierać przedrostki Nac dla Nachamkin i Nay dla Nayuddin, będące dwoma kluczami po obu stronach wskaźnika do poddrzewa, do którego należy przejść.

### 17.6.3. Dostrajanie indeksów

Utworzone na początku indeksy trzeba czasem zmodyfikować. Wynika to z następujących przyczyn:

- Niektóre zapytania są wykonywane zbyt długo z powodu braku indeksu.
- Niektóre indeksy w ogóle nie są używane.
- Niektóre indeksy są zbyt często aktualizowane, ponieważ indeks dotyczy często modyfikowanego atrybutu.

Większość SZBD obejmuje instrukcje lub mechanizmy sprawdzania śladu, które pozwalają administratorom baz zażądać od systemu wyświetlenia sposobu wykonywania zapytań. Można w ten sposób zobaczyć, jakie operacje były wykonywane, w jakiej kolejności i jakie drugorzędne struktury dostępowe (indeksy) zostały użyte. Dzięki analizie takich planów wykonywania zapytań (to pojęcie opiszemy dokładniej w rozdziale 18.) można zdiagnozować przyczyny problemów. Na podstawie analiz związanych z dostrajaniem można usunąć niektóre indeksy i utworzyć nowe.



Z dostrajaniem związane są dynamiczna ocena wymagań, które czasem zmieniają się sezonowo lub w różnych porach miesiąca czy tygodnia, oraz modyfikacja indeksów i sposobów organizacji plików w celu maksymalizacji wydajności. Usuwanie i budowanie nowych indeksów oznacza koszty, które można uzasadnić poprawą wydajności. Na czas usuwania lub tworzenia indeksu aktualizowanie tabeli jest zwykle wstrzymywane. Trzeba uwzględnić tego rodzaju wstrzymanie świadczenia usług.

Oprócz usuwania lub tworzenia indeksów oraz przekształcania indeksów nieklastrowych w indeksy klastrowania i na odwrót można w celu poprawy wydajności zastosować **przebudowywanie indeksów**. Większość relacyjnych SZBD używa indeksów opartych na B<sup>+</sup>-drzewach. Jeśli klucz indeksowania jest często usuwany, strony indeksu mogą obejmować zmarnowaną przestrzeń, możliwą do odzyskania w ramach przebudowania indeksu. Ponadto zbyt wiele operacji wstawiania może skutkować przepełnieniem w indeksie klastrowania, co obniża wydajność. Przebudowanie indeksu klastrowania wymaga zmiany organizacji całej tabeli uporządkowanej według klucza używanego w danym indeksie.

Dostępne sposoby indeksowania, a także definiowania, tworzenia i reorganizacji indeksów zależą od systemu. W ramach przykładu rozważ indeksy rzadkie i zagęszczone opisane w podrozdziale 17.1. Indeks rzadki, np. indeks główny, obejmuje tylko jeden wskaźnik dla każdej strony (każdego bloku dyskowego) z pliku danych. Indeks zagęszczony, np. unikatowy indeks drugorzędny, obejmuje wskaźnik dla każdego rekordu. System Sybase udostępnia indeksy klastrowania w formie indeksów rzadkich opartych na B<sup>+</sup>-drzewach, a w systemie INGRES rzadkie indeksy klastrowania są plikami ISAM, natomiast zagęszczone indeksy klastrowania to B<sup>+</sup>-drzewa. W niektórych wersjach systemów Oracle i DB2 indeksem klastrowania musi być indeks zagęszczony, a administratorzy muszą sobie poradzić z tym ograniczeniem.

## 17.6.4. Dodatkowe kwestie związane ze składowaniem relacji i indeksów

**Używanie indeksów do zarządzania ograniczeniami i duplikatami.** Indeksy często są używane do wymuszania *ograniczeń klucza* na atrybucie. W trakcie przeszukiwania indeksu w celu wstawienia nowego rekordu można łatwo sprawdzić, czy inny rekord w danym pliku (a tym samym i w drzewie indeksu) ma tę samą wartość atrybutu klucza co nowy rekord. Jeśli tak jest, można odrzucić operację wstawiania.

Jeżeli indeks jest tworzony na polu niebędącym kluczem, mogą występować *duplikaty*. Ich obsługa to problem, z którym muszą sobie radzić producenci SZBD. Wpływa on na składowanie danych, a także na tworzenie indeksów i zarządzanie nimi. Rekordy danych z powtarzającym się kluczem mogą być zapisane w tym samym bloku lub obejmować grupę bloków, w których można umieścić wiele duplikatów. W niektórych systemach do rekordu dołączony jest identyfikator wiersza, dzięki czemu rekordy z powtarzającymi się kluczami mogą mieć własne, unikatowe identyfikatory. W takiej sytuacji w indeksie opartym na B<sup>+</sup>-drzewie można traktować kombinację <klucz, identyfikator\_wiersza> jako rzeczywisty klucz. Powstaje wtedy unikatowy indeks bez duplikatów. Usunięcie klucza *K* z takiego indeksu powoduje usunięcie wszystkich wystąpień *K*. Trzeba to uwzględnić w algorytmie usuwania danych.

W rzeczywistych SZBD usuwanie elementów z indeksów opartych na B<sup>+</sup>-drzewie jest obsługiwane w różny sposób, co pozwala poprawić wydajność i czas reakcji. Usunięte rekordy można oznaczyć jako skasowane, co pozwala pozostawić powiązane z nimi wpisy w indeksie do czasu, gdy proces odzyskiwania pamięci odzyska pamięć w pliku danych. Po procesie odzyskiwania pamięci indeks można na żywo odtworzyć.

**Pliki odwrócone i inne metody dostępu.** Plik, który posiada indeks drugorzędny na każdym ze swoich pól, często określa się mianem **pliku w pełni odwróconego** (ang. *fully inverted file*). Ze względu na fakt, że wszystkie indeksy są drugorzędne, nowe rekordy są wstawiane na końcu pliku. Zatem sam plik danych jest plikiem nieuporządkowanym (stertowym). Indeksy implementuje się zazwyczaj jako B<sup>+</sup>-drzewa, więc są one aktualizowane dynamicznie w celu odzwierciedlenia operacji wstawiania i usuwania rekordów. Niektóre komercyjne SZBD, takie jak Adabas firmy Software-AG, powszechnie wykorzystują taką metodę.

W podrozdziale 17.2 wspomniano o popularnej metodzie organizacji pliku stosowanej przez firmę IBM — ISAM. Inna metoda wykorzystywana przez tę firmę — **virtual storage access method (VSAM)** — przypomina strukturę dostępową B<sup>+</sup>-drzew i jest stosowana w wielu systemach komercyjnych.

**Używanie wskazówek indeksowania w zapytaniach.** SZBD takie jak Oracle umożliwiają stosowanie w zapytaniach wskazówek będących sugerowanymi alternatywami lub sygnałami dla procesora i optymalizatora zapytań, określającymi, jak wykonać zapytanie. Jeden z rodzajów wskazówek to wskazówki indeksowania. Sugerują one, jak wykorzystać indeks w celu usprawnienia wykonywania zapytania. Takie wskazówki mają postać specjalnych komentarzy (poprzedzonym symbolem +) i zastępują wszystkie decyzje optymalizatora, choć mogą zostać ignorowane przez optymalizator, jeśli są błędne, nieadekwatne lub niepoprawnie sformułowane. Nie będziemy tu szczegółowo omawiać wskazówek indeksowania, a jedynie zilustrujemy je za pomocą modelowych zapytań.

Przykładowo, aby pobrać numer PESEL, wynagrodzenie i numer działu pracowników zatrudnionych w działach o numerach mniejszych niż 10, można zastosować zapytanie:

```
SELECT /*+ INDEX (PRACOWNIK PRAC_NRDZ_INDEKS ) */ PESEL, PENSJA, NRDZ
FROM PRACOWNIK
WHERE NRDZ < 10;
```

To zapytanie obejmuje wskazówkę dotyczącą użycia prawidłowego indeksu PRAC\_NRDZ\_INDEKS (jest to indeks na atrybucie NRDZ relacji PRACOWNIK).

**Składowanie kolumnowe relacji.** Ostatnim trendem jest stosowanie składowania kolumnowego (ang. *column-based storage*) relacji jako alternatywy dla tradycyjnego sposobu przechowywania relacji w wierszach. Komercyjne relacyjne SZBD udostępniają oparte na B<sup>+</sup>-drzewach indeksy na kluczach głównych i drugorzędnych. Jest to wydajny mechanizm wspomagający dostęp do danych na podstawie różnych kryteriów wyszukiwania i umożliwiający zapis pojedynczych wierszy lub ich grup w celu optymalizacji systemu pod kątem zapisu. W hurtowniach danych (opisanych w rozdziale 29.), które są bazami danych przeznaczonymi tylko do odczytu, składowanie kolumnowe zapewnia wyjątkowe korzyści w zapytaniach wymagających samego odczytu. W relacyjnych SZBD ze składowaniem kolumnowym możliwe jest przechowywanie każdej kolumny danych osobno, co umożliwia poprawę wydajności w następujących obszarach:

- Pionowy podział baz na kolumny, co pozwala utworzyć dwukolumnową tabelę dla każdego atrybutu i uzyskiwać dostęp tylko do potrzebnych kolumn.
- Używanie indeksów kolumnowych (podobnych do opisanych w punkcie 17.5.2 indeksów bitmapowych) i indeksów złączeń na wielu tabelach, aby obsługiwać zapytania bez konieczności dostępu do tabel z danymi.
- Używanie perspektyw zmaterializowanych (patrz rozdział 7.), aby zapewnić obsługę zapytań dotyczących wielu kolumn.

Kolumnowe składowanie danych zapewnia dodatkową swobodę w tworzeniu indeksów, np. opisanych wcześniej indeksów bitmapowych. Ta sama kolumna może występować w wielu projekcjach tabeli, a indeksy można utworzyć dla każdej projekcji. Na potrzeby zapisu wartości w tej samej kolumnie opracowano strategie kompresji danych, ukrywanie wartości pustych, techniki kodowania słownikowego (gdzie różnym wartościom z kolumny przypisywane są krótsze kody) i techniki kodowania długości serii (ang. *run-length encoding* — RLE). Przykładowe systemy używające tych metod to MonetDB/X100, C-Store i Vertica. W niektórych popularnych systemach (takich jak Cassandra, Hbase i Hypertable) składowanie kolumnowe jest skutecznie stosowane w **bazach kolumnowych** (ang. *wide column-stores*). Składowanie danych w takich systemach jest wyjaśnione w kontekście systemów NOSQL omówionych w rozdziale 24.

## 17.7. Fizyczne projektowanie baz danych w przypadku baz relacyjnych

W niniejszym podrozdziale najpierw zostaną omówione czynniki związane z fizycznym projektowaniem baz danych, wpływające na wydajność działania aplikacji i transakcji. Następnie omówimy określone wskazówki dla relacyjnych SZBD w kontekście informacji przedstawionych w rozdziale 16. i wcześniejszych fragmentach niniejszego rozdziału.

### 17.7.1. Czynniki wpływające na fizyczny projekt bazy danych

Projektowanie fizyczne to działanie, którego celem jest nie tylko otrzymanie odpowiedniej struktury przechowywanych danych, ale również dokonanie tego w sposób gwarantujący wysoką wydajność działania. W przypadku danego koncepcyjnego schematu bazy danych istnieje wiele alternatywnych projektów fizycznych w ramach określonego SZBD. Nie jest możliwe podejmowanie wiążących decyzji w zakresie fizycznego projektu bazy danych i wykonywanie analiz wydajnościowych, dopóki nie zna się zapytań, transakcji i aplikacji, które będą działać w ramach bazy. Jest to **zestaw zadań** z konkretnego zbioru aplikacji systemu bazy danych. Administratorzy i projektanci baz danych muszą przeanalizować takie aplikacje, spodziewaną częstotliwość ich wywoływania, wszelkie więzy czasowe nałożone na ich wykonanie, spodziewaną częstotliwość występowania operacji aktualizacji oraz więzy unikatowości dotyczące atrybutów. Poniżej omówiono wszystkie te czynniki.

**A. Analiza zapytań i transakcji w bazie danych.** Przed rozpoczęciem tworzenia fizycznego projektu bazy danych trzeba mieć dobre wyobrażenie o planowanych sposobach użycia bazy danych poprzez zdefiniowanie w wysokopoziomowej postaci zapytań i transakcji, które

będą wykonywane na bazie. Dla każdego **zapytania pobierającego dane** należy określić następujące elementy:

- (1) Pliki (relacje), do których zapytanie będzie uzyskiwało dostęp.
- (2) Atrybuty, na których są określone warunki selekcji w zapytaniu.
- (3) To, czy warunkiem selekcji jest równość, nierówność czy warunek zakresowy.
- (4) Atrybuty, na których są określone warunki złączeń lub warunki łączące kilka tabel lub obiektów.
- (5) Atrybuty, których wartości będą zwracane przez zapytanie.

Atrybuty wymienione w punktach 2. i 4. są kandydatami do zdefiniowania struktur dostępowych takich jak indeksy, klucze mieszania i posortowany plik.

Dla każdej **transakcji lub operacji aktualizującej** należy określić następujące elementy:

- (1) Pliki, które będą aktualizowane.
- (2) Rodzaj operacji wykonywanej na każdym pliku (wstawianie, aktualizowanie lub usuwanie).
- (3) Atrybuty, na których są określone warunki selekcji dla operacji usuwania lub aktualizowania.
- (4) Atrybuty, których wartości będą zmieniane przez operację aktualizacji.

Ponownie, atrybuty wymienione w punkcie 3. są kandydatami do zdefiniowania struktur dostępowych, ponieważ służą one do lokalizowania aktualizowanych lub usuwanych rekordów. Z drugiej strony, atrybuty wymienione w punkcie 4. *nie powinny być wykorzystywane do definiowania struktur dostępowych*, gdyż ich modyfikowanie będzie wymagało aktualizacji także struktur dostępowych.

**B. Analiza spodziewanej częstotliwości wywoływania zapytań i transakcji.** Oprócz zidentyfikowania charakterystyk oczekiwanych zapytań i transakcji należy wziąć pod uwagę spodziewaną częstotliwość ich wykonywania. Takie informacje, wraz z informacjami o atrybutach zebranymi dla każdego zapytania i transakcji, są używane w celu utworzenie łącznej listy spodziewanej częstotliwości użycia wszystkich zapytań i transakcji. Wyraża się to w postaci spodziewanej częstotliwości użycia każdego atrybutu w każdym pliku jako atrybutu selekcji lub atrybutu złączenia w stosunku do wszystkich zapytań i transakcji. Ogólnie rzecz biorąc, w przypadku dużych obciążeń w zakresie przetwarzania ma zastosowanie nieformalna *reguła 80-20*, która określa, że około 80 procent czasu przetwarzania jest związane z tylko 20 procentami zapytań i transakcji. Dlatego też w praktyce rzadko zachodzi konieczność zbierania dokładnych statystyk i wskaźników wywołań dla wszystkich zapytań i transakcji — wystarczy określić około 20 procent najważniejszych z nich.

**C. Analiza więzów czasowych na zapytaniach i transakcjach.** Niektóre zapytania i transakcje mogą posiadać surowe ograniczenia co do wydajności ich działania. Przykładowo, transakcja może posiadać określone ograniczenie, mówiące o tym, że powinna kończyć działanie przed upływem 5 sekund w 95 procentach przypadków swoich wywołań i nigdy nie powinna być wykonywana dłużej niż 20 sekund. Takie więzy wydajnościowe określają dalsze priorytety wśród atrybutów będących kandydatami do tworzenia ścieżek dostępu. Atrybuty selekcji używane przez zapytania i transakcje w kontekście ograniczeń czasowych

stają się kandydatami o wysokim priorytecie do tworzenia podstawowych struktur dostępowych, ponieważ podstawowe struktury dostępne są zwykle najwydajniejsze, jeśli chodzi o lokalizowanie rekordów w plikach.

**D. Analiza spodziewanej częstotliwości operacji aktualizacji.** Dla pliku, który podlega częstym aktualizacjom, należy określić minimalną liczbę ścieżek dostępu, ponieważ aktualizacja samych ścieżek dostępu spowalnia operacje aktualizacji. Przykładowo, jeśli plik, do którego często wstawiane są rekordy, ma 10 indeksów na 10 różnych atrybutach, każdy z tych indeksów trzeba aktualizować za każdym razem, gdy wstawiany jest nowy indeks. Koszty aktualizowania 10 indeksów mogą spowolnić operacje wstawiania.

**E. Analiza więzów unikatowości na atrybutach.** Ścieżki dostępu powinny zostać określone na wszystkich atrybutach *klucza kandydującego* (lub zbiorach atrybutów), które są albo kluczem głównym, albo posiadają określone więzy unikatowości. Istnienie indeksu (lub innej ścieżki dostępu) sprawia, że wystarczy przeszukać tylko indeks w razie sprawdzania takich więzów, gdyż wszystkie wartości atrybutu występują w liściach indeksu. Przykładowo, jeśli w trakcie wstawiania nowego rekordu okaże się, że wartość atrybutu klucza tego rekordu *już istnieje w indeksie*, wstawianie nowego rekordu trzeba zablokować, ponieważ narusza on więzy unikatowości związane z danym atrybutem.

Po sporządzeniu listy takich informacji możemy odnieść się do decyzji dotyczących fizycznego projektu bazy danych, które zasadniczo dotyczą wyboru struktur składowania i ścieżek dostępu do plików bazy danych.

## 17.7.2. Decyzje dotyczące fizycznego projektu bazy danych

Większość systemów relacyjnych reprezentuje każdą relację główną jako fizyczny plik bazy danych. Opcje ścieżek dostępu umożliwiają określenie rodzaju pliku dla każdej relacji oraz atrybutów, na których należy zdefiniować indeksy (pojedyncze lub złożone). Najwyżej jeden z indeksów zdefiniowanych na każdym pliku może być indeksem głównym lub klastrowania. Można jednak utworzyć dowolną liczbę dodatkowych indeksów drugorzędnych.

**Decyzje projektowe dotyczące indeksów.** Atrybuty, których wartości występują w warunkach równościowych lub zakresowych (operacje selekcji), oraz te, które są kluczami lub występują w warunkach złączeń (operacje złączeniowe), wymagają określenia ścieżek dostępu — np. indeksów.

Wydajność wykonywania zapytań w dużej mierze zależy od tego, jakie istnieją indeksy lub schematy mieszania mogące przyspieszyć przetwarzanie operacji selekcji i złączeń. Z drugiej strony, w czasie operacji wstawiania, usuwania i aktualizowania istnienie indeksów wprowadza pewien narzut. Narzut ten musi być uzasadniony osiąganymi korzyściami pod względem zwiększenia wydajności działania zapytań i transakcji.

Podejmowane decyzje związane z projektem fizycznym należą do czterech kategorii:

- (1) **Określenie, czy należy indeksować dany atrybut.** Zgodnie z ogólnymi regułami tworzenia indeksu atrybut musi być kluczem (unikatowym) lub musi występować pewne zapytanie wykorzystujące dany atrybut albo w warunku selekcji (równość lub zakres wartości) albo w złączeniu. Jednym z czynników uzasadniających tworzenie wielu indeksów jest to, że niektóre zapytania mogą być przetwarzane poprzez przegląd samego indeksu, bez pobierania żadnych danych.

- (2) **Określenie, jaki atrybut (lub atrybuty) powinien być indeksowany.** Indeks można utworzyć na jednym lub (jeśli indeks jest złożony) wielu atrybutach. Jeżeli kilka atrybutów pochodzących z jednej relacji jest wspólnie uwzględnianych w kilku zapytaniach (na przykład (`styl_ubioru_#`, `kolor`) w przypadku bazy danych magazynu odzieżowego), wskazane jest utworzenie indeksu wieloattributowego. Kolejność atrybutów w takim indeksie musi odpowiadać zapytaniom. Przykładowo, w przypadku powyższego indeksu zakłada się, że zapytania będą bazować na uporządkowaniu kolorów według stylu ubiorów a nie odwrotnie.
- (3) **Określenie, czy należy utworzyć indeks klastrowania.** W przypadku każdej tabeli może istnieć co najwyżej jeden indeks główny lub klastrowania, ponieważ wiąże się to z fizycznym uporządkowaniem pliku względem danego atrybutu. W większości relacyjnych SZBD określa się to słowem kluczowym `CLUSTER` (jeżeli atrybut jest *kluczem*, tworzony jest *indeks główny*, natomiast *indeks klastrowania* jest tworzony w przypadku, gdy atrybut *nie jest kluczem*). Jeżeli tabela wymaga kilku indeksów, decyzja odnośnie do tego, który z nich powinien być indeksem głównym lub klastrowania, zależy od tego, czy jest konieczne zachowanie uporządkowania tabeli według danego atrybutu. Zapytania zakresowe wiele zyskują dzięki klastrowaniu. Jeżeli kilka atrybutów wymaga zapytań zakresowych, należy oszacować uzyskiwane względne korzyści przed podjęciem decyzji o tym, na którym atrybucie wykonać klastrowanie. Jeżeli odpowiedź dla zapytania ma być uzyskiwana tylko poprzez przegląd indeksu (bez pobierania rekordów danych), odpowiedni indeks *nie* powinien być indeksem klastrowania, gdyż największe korzyści z klastrowania uzyskuje się wówczas, gdy pobierane są właśnie rekordy danych. Indeks klastrowania można utworzyć na wielu atrybutach, jeśli zapytania zakresowe oparte na danym kluczu złożonym są przydatne do tworzenia raportów. Przykładowo, indeks na polach `KOD_POCZTOWY`, `ID_SKLEPU` i `ID_PRODUKTU` może być indeksem klastrowania dla danych sprzedażowych.
- (4) **Określenie, czy należy użyć indeksu mieszającego zamiast drzewiastego.** Ogólnie rzecz biorąc, relacyjne SZBD wykorzystują B<sup>+</sup>-drzewa w celu indeksowania. Jednakże, w niektórych systemach dostępny jest również schemat ISAM oraz indeksy mieszające. B<sup>+</sup>-drzewa obsługują zarówno zapytania równościowe, jak i zakresowe na atrybucie używanym jako klucz wyszukiwania. Indeksy mieszające sprawdzają się dobrze w przypadku warunków równościowych, szczególnie w czasie złączeń i znajdowania pasujących rekordów, jednak nie obsługują zapytań zakresowych.
- (5) **Określenie, czy w przypadku danego pliku należy użyć mieszania dynamicznego.** W przypadku plików, które często się zmieniają — to znaczy takich, które stale ulegają rozszerzaniu i zmniejszaniu — odpowiedni jest jeden ze schematów mieszania dynamicznego opisanych w podrozdziale 16.9. Obecnie większość komercyjnych relacyjnych SZBD nie oferuje takiej możliwości.

## 17.8. Podsumowanie

W niniejszym rozdziale przedstawiono metody organizacji plików związane z dodatkowymi strukturami dostępowymi, noszącymi nazwę indeksów. Służą one do zwiększania wydajności operacji pobierania rekordów z pliku danych. Takie struktury dostępowe mogą



być używane w *połączeniu* z podstawowymi metodami organizacji plików, omówionymi w rozdziale 16., których używa się w celu organizacji na dysku samych rekordów pliku.

Przedstawiono trzy rodzaje uporządkowanych indeksów jednopoziomowych: głównych, klastrowania i drugorzędnych. Każdy indeks określa się na polu pliku. Indeksy główny i klastrowania jest konstruowany z uwzględnieniem fizycznego uporządkowania pola pliku, natomiast indeks drugorzędny określa się na polach niebędących polami uporządkowania i jest on dodatkową strukturą dostępową poprawiającą wydajność zapytań oraz transakcji. Pole indeksu głównego musi być kluczem pliku, natomiast w przypadku indeksu klastrowania nie jest ono polem klucza. Indeks jednopoziomowy jest plikiem uporządkowanym i przeszukuje się go, wykorzystując wyszukiwanie binarne. W rozdziale pokazano również, w jaki sposób można konstruować indeksy wielopoziomowe w celu zwiększenia wydajności przeszukiwania indeksu. Przykładem jest tu popularna technika ISAM (ang. *index sequential access method*) firmy IBM, mająca postać indeksu wielopoziomowego opartego na konfiguracji cylindrów i ścieżek na dysku.

Następnie zaprezentowano, w jaki sposób można implementować indeksy jako B-drzewa i B<sup>+</sup>-drzewa, które są strukturami dynamicznymi pozwalającymi na dynamiczne rozszerzanie i zmniejszanie indeksu. Wierzchołki (bloki) tych struktur indeksowych są przechowywane jako wypełnione w połowie lub w całości dzięki odpowiedniemu działaniu algorytmów wstawiania i usuwania. W dalszej perspektywie stan wierzchołków stabilizuje się i średnio są one wypełnione w 69 procentach, co pozwala na zachowanie przestrzeni dla wstawień bez wymagania reorganizacji indeksu w przypadku większości operacji wstawiania. B<sup>+</sup>-drzewa mogą, ogólnie rzecz biorąc, przechowywać więcej wpisów w swoich wierzchołkach wewnętrznych niż B-drzewa, a stąd mogą mieć mniej poziomów lub zawierać więcej wpisów niż analogiczne B-drzewa.

Przedstawiono również ogólne omówienie metod dostępu do kluczy wielokrotnych i pokazano, w jaki sposób można konstruować indeks w oparciu o mieszające struktury danych. Dalej opisaliśmy **mieszanie partycjonowane**, będące wzbogaceniem mieszania zewnętrznego o obsługę wielu kluczy. Przedstawiliśmy też **pliki matrycowe**, w których dane są uporządkowane w pakiety według wielu wymiarów. Omówiliśmy również dość szczegółowo **indeksy mieszające**. Są to struktury drugorzędne zapewniające dostęp do plików za pomocą mieszania opartego na kluczu wyszukiwania innym niż ten stosowany do podstawowej organizacji danych. **Indeksy bitmapowe** to następny ważny rodzaj indeksów wykorzystywany do zapytań z użyciem wielu kluczy i dostosowany przede wszystkim do pól o małej liczbie unikatowych wartości. Bitmapy mogą też być używane w liściach indeksów opartych na B<sup>+</sup>-drzewach. Omówiliśmy również indeksy oparte na funkcjach, wprowadzane przez producentów baz relacyjnych, aby umożliwić tworzenie specjalnych indeksów opartych na funkcji zależnej od jednego lub kilku atrybutów.

Następnie przedstawiono pojęcie indeksu logicznego i porównano go z opisywanymi wcześniej indeksami fizycznymi. Indeksy logiczne tworzą dodatkowy poziom pośredni w indeksowaniu, co zwiększa swobodę przenoszenia rekordów na dysku. Opisaliśmy też tworzenie indeksów w języku SQL, proces wczytywania zbiorczego plików indeksów i indeksowanie ciągów znaków. Omówiliśmy warunki prowadzące do strojenia indeksów. Dalej przedstawiliśmy ogólne zagadnienia związane z indeksowaniem, w tym zarządzanie więzami, używanie indeksów odwróconych i stosowanie wskazówek indeksowania w zapytaniach.



Wspomnieliśmy też o składowaniu kolumnowym w relacjach, które staje się akceptowalną alternatywą dla składowania dużych baz danych i dostępu do nich. W końcowej części omówiliśmy fizyczne projektowanie relacyjnych baz danych, co wymaga podejmowania decyzji z zakresu składowania danych i dostępu do nich; zagadnienia te przedstawiliśmy w tym i poprzednim rozdziale. Ten fragment podzieliliśmy na opis czynników wpływających na projekt i decyzji określających, czy tworzyć indeks na podstawie danego atrybutu, jakie atrybuty uwzględnić w indeksie, czy tworzyć indeksy klastrowania, czy zwykłe, a także czy stosować indeksy mieszające i mieszanie dynamiczne.

## Pytania powtórkowe

- 17.1. Zdefiniuj następujące pojęcia: *pole indeksujące*, *pole klucza głównego*, *pole klastrowania*, *pole klucza drugorzędnego*, *zaczepienie bloku*, *indeks zagęszczony* oraz *indeks niezagęszczony (rzadki)*.
- 17.2. Jakie różnice występują między indeksem głównym, drugorzędnym a klastrowania? Jak różnice te wpływają na sposoby, w jakie implementuje się te indeksy? Które z nich są zagęszczone, a które nie?
- 17.3. Dlaczego można posiadać najwyżej jeden indeks główny lub indeks klastrowania na pliku, ale kilka indeksów drugorzędnych?
- 17.4. W jaki sposób indeksy wielopoziomowe zwiększają wydajność przeszukiwania pliku indeksu?
- 17.5. Czym jest rząd  $p$  B-drzewa? Opisz strukturę wierzchołków B-drzewa.
- 17.6. Czym jest rząd  $p$  B<sup>+</sup>-drzewa? Opisz strukturę wierzchołków wewnętrznych i wierzchołków liści B<sup>+</sup>-drzewa.
- 17.7. Czym różni się B-drzewo od B<sup>+</sup>-drzewa? Dlaczego B<sup>+</sup>-drzewo jest zwykle preferowane jako struktura dostępowa do pliku danych?
- 17.8. Wyjaśnij, jakie istnieją możliwości wyboru w zakresie uzyskiwania dostępu do pliku w oparciu o wiele kluczy wyszukiwania.
- 17.9. Czym jest mieszanie partycjonowane? W jaki sposób działa? Jakie wiążą się z nim ograniczenia?
- 17.10. Czym jest plik matrycowy? Jakie są jego zalety i wady?
- 17.11. Przedstaw przykład konstrukcji matrycy na dwóch atrybutach pewnego pliku.
- 17.12. Czym jest plik w pełni odwrócony? Czym jest indeksowany plik sekwencyjny?
- 17.13. W jaki sposób można użyć mieszania w celu skonstruowania indeksu?
- 17.14. Czym jest indeks bitmapowy? Utwórz relację z 2 kolumnami i 16 krotkami oraz przedstaw przykładowy indeks bitmapowy na jednej lub obu kolumnach.
- 17.15. Czym są indeksy oparte na funkcjach? Jaki jest dodatkowy cel ich używania?
- 17.16. Czym różnią się indeksy logiczne od fizycznych?
- 17.17. Czym jest składowanie kolumnowe w relacyjnych bazach danych?

## Ćwiczenia

17.18. Weźmy pod uwagę dysk o bloku rozmiaru  $B = 512$  bajtów. Wskaźnik bloku ma długość  $P = 6$  bajtów, zaś wskaźnik rekordu ma długość  $P_R = 7$  bajtów. Plik zawiera  $r = 30\,000$  rekordów *PRACOWNIK o stałej długości*. Każdy rekord posiada następujące pola: IMIĘNAZW (30 bajtów), PESEL (9 bajtów), KODDZIAŁU (9 bajtów), ADRES (40 bajtów), TELEFON (10 bajtów), DATAURODZENIA (8 bajtów), PŁEĆ (1 bajt), KODSTANOWISKA (4 bajty), PENSJA (4 bajty, wartość rzeczywista). Dodatkowy bajt jest używany jako znacznik usunięcia.

- a) Oblicz rozmiar rekordu  $R$  w bajtach.
- b) Oblicz współczynnik blokowy  $bfr$  oraz liczbę bloków pliku  $b$ , zakładając organizację niesegmentowaną.
- c) Załóżmy, że plik jest *uporządkowany* względem pola klucza PESEL i chcemy skonstruować *indeks główny* na tym polu. Oblicz: (i) współczynnik blokowy indeksu  $bfr_i$  (który jest również zwielokrotnieniem wyjściowym indeksu,  $fo$ ); (ii) liczbę wpisów na pierwszym poziomie indeksu oraz liczbę bloków na pierwszym poziomie indeksu; (iii) liczbę wymaganych poziomów, jeżeli wykorzystamy indeks wielopoziomowy; (iv) całkowitą liczbę bloków wymaganych przez indeks wielopoziomowy oraz (v) liczbę operacji dostępu do bloków wymaganych w celu wyszukania i pobrania rekordu z pliku (dla danej wartości pola PESEL) przy użyciu indeksu głównego.
- d) Załóżmy, że plik *nie jest uporządkowany* względem pola klucza PESEL i chcemy skonstruować *indeks drugorzędny* na tym polu. Powtórz poprzednie ćwiczenie (punkt c.) dla indeksu drugorzędnego i porównaj otrzymane wyniki z wynikami otrzymanymi dla indeksu głównego.
- e) Załóżmy, że plik *nie jest uporządkowany* względem pola niebędącego polem klucza KODDZIAŁU i chcemy skonstruować *indeks drugorzędny* na tym polu, wykorzystując rozwiązanie trzecie z podrozdziału 17.1.3, z dodatkowym poziomem pośrednim służącym do przechowywania wskaźników rekordów. Zakładamy, że istnieje 1000 różnych wartości pola KODDZIAŁU oraz że rozkładają się one równomiernie w rekordach *PRACOWNIK*. Oblicz: (i) współczynnik blokowy indeksu  $bfr_i$  (który jest również zwielokrotnieniem wyjściowym indeksu,  $fo$ ); (ii) liczbę bloków wymaganych na poziomie pośrednim przechowującym wskaźniki rekordów; (iii) liczbę wpisów na pierwszym poziomie indeksu oraz liczbę bloków na pierwszym poziomie indeksu; (iv) liczbę wymaganych poziomów, jeżeli wykorzystamy indeks wielopoziomowy; (v) całkowitą liczbę bloków wymaganych przez indeks wielopoziomowy i bloków używanych na dodatkowym poziomie pośrednim oraz (vi) przybliżoną liczbę operacji dostępu do bloków wymaganych w celu wyszukania i pobrania wszystkich rekordów z pliku posiadających określoną wartość pola KODDZIAŁU przy użyciu tego indeksu.
- f) Załóżmy, że plik jest *uporządkowany* względem pola niebędącego polem klucza KODDZIAŁU i chcemy skonstruować *indeks klastrowania* na tym polu, wykorzystując zaczepienie bloków (każda nowa wartość pola KODDZIAŁU rozpoczyna się na początku nowego bloku). Zakładamy, że istnieje 1000 różnych wartości pola KODDZIAŁU oraz że rozkładają się one równomiernie

w rekordach PRACOWNIK. Oblicz: (i) współczynnik blokowy indeksu  $bfr_i$  (który jest również zwielokrotnieniem wyjściowym indeksu,  $fo$ ); (ii) liczbę wpisów na pierwszym poziomie indeksu oraz liczbę bloków na pierwszym poziomie indeksu; (iii) liczbę wymaganych poziomów, jeżeli wykorzystamy indeks wielopoziomowy; (iv) całkowitą liczbę bloków wymaganych przez indeks wielopoziomowy oraz (v) przybliżoną liczbę operacji dostępu do bloków wymaganych do wyszukania i pobrania wszystkich rekordów z pliku posiadających określoną wartość pola KODDZIAŁU przy użyciu indeksu klastrowania (zakładamy, że wiele bloków w klastrze jest ułożonych w sposób ciągły).

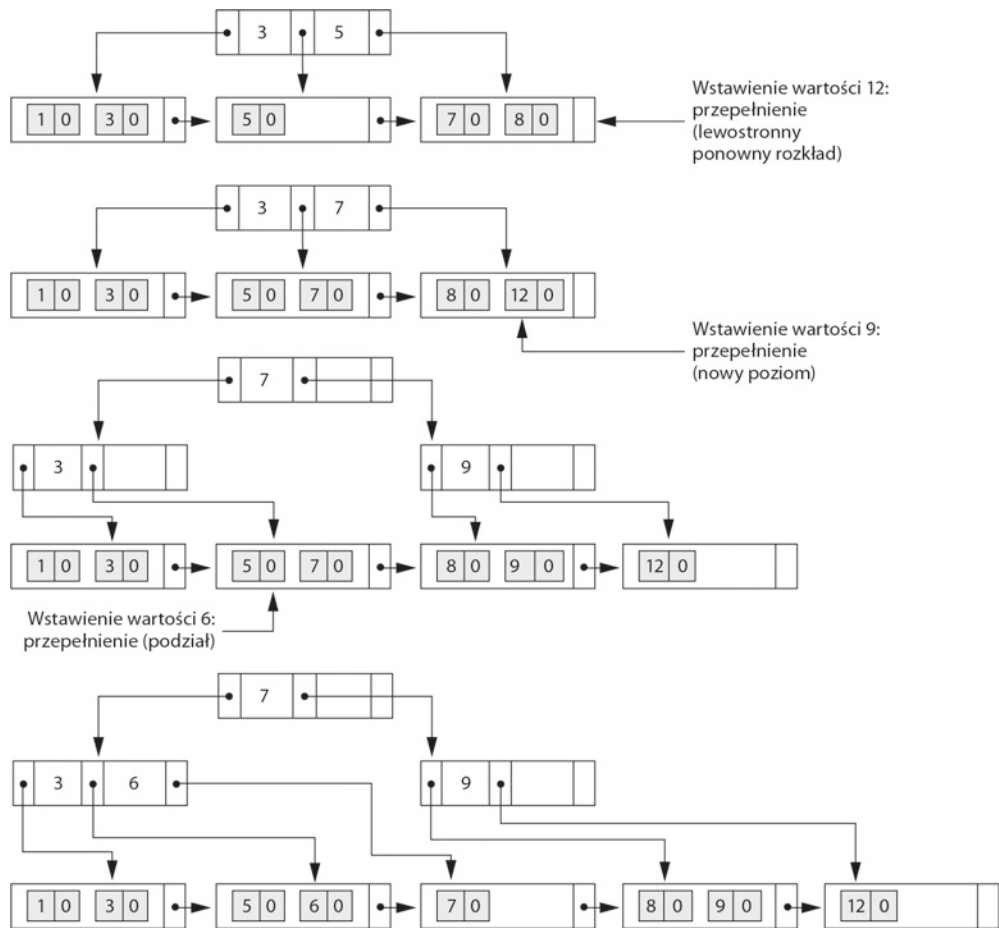
- g) Załóżmy, że plik *nie* jest uporządkowany względem pola klucza PESEL i chcemy skonstruować indeks oparty na strukturze dostępowej B<sup>+</sup>-drzewa na tym polu. Oblicz: (i) rzędy  $p$  i  $p_{liść}$  B<sup>+</sup>-drzewa; (ii) liczbę bloków poziomu liści wymaganych w przypadku, gdy bloki są zapełnione w przybliżeniu w 69 procentach (dla wygody możesz zaokrąglić wartości); (iii) liczbę wymaganych poziomów, jeżeli wierzchołki wewnętrzne również są zapełnione w przybliżeniu w 69 procentach (dla wygody możesz zaokrąglić wartości); (iv) całkowitą liczbę bloków wymaganych przez B<sup>+</sup>-drzewo oraz (v) liczbę operacji dostępu do bloków wymaganych do wyszukania i pobrania rekordu z pliku dla danej wartości pola PESEL przy użyciu B<sup>+</sup>-drzewa.
- h) Powtórz ćwiczenie z punktu g. dla B-drzewa zamiast B<sup>+</sup>-drzewa. Porównaj otrzymane wyniki.

- 17.19. Plik CZĘŚCI z polem CZĘŚĆ# jako polem klucza zawiera rekordy o następujących wartościach tego pola: 23, 65, 37, 60, 46, 92, 48, 71, 56, 59, 18, 21, 10, 74, 78, 15, 16, 20, 24, 28, 39, 43, 47, 50, 69, 75, 8, 49, 33, 38. Załóżmy, że wartości pola wyszukiwania są wstawiane w podanej kolejności do B<sup>+</sup>-drzewa rzędu  $p = 4$  oraz  $p_{liść} = 3$ . Pokaż, w jaki sposób drzewo będzie się rozszerzać i jaka będzie jego końcowa postać.
- 17.20. Powtórz ćwiczenie 14.15, ale tym razem użyj B-drzewa rzędu  $p = 4$ .
- 17.21. Załóżmy, że z B<sup>+</sup>-drzewa z ćwiczenia 17.19 zostały usunięte następujące wartości pola wyszukiwania w podanej kolejności: 65, 75, 43, 18, 20, 92, 59, 37. Pokaż, w jaki sposób drzewo będzie się zmniejszać i jaka będzie jego końcowa postać.
- 17.22. Powtórz ćwiczenie 17.21, ale tym razem użyj B-drzewa z ćwiczenia 17.20.
- 17.23. Algorytm 17.1 stanowi zarys procedury przeszukiwania niezagęszczanego wielopoziomowego indeksu głównego w celu pobrania rekordu pliku. Dostosuj ten algorytm do wymagań każdego z poniższych przypadków:
  - a) Wielopoziomowy indeks drugorzędny na nieuporządkowanym polu niebędącym kluczem pliku. Zakładamy, że używane jest trzecie rozwiązanie z podrzdziału 17.1.3, gdzie dodatkowy poziom pośredni służy do przechowywania wskaźników na poszczególne rekordy o odpowiedniej wartości pola indeksu.
  - b) Wielopoziomowy indeks drugorzędny na nieuporządkowanym polu klucza pliku.
  - c) Wielopoziomowy indeks klastrowania na uporządkowanym polu niebędącym polem klucza pliku.

- 17.24. Załóżmy, że na polach pliku niebędących polami klucza utworzono kilka indeksów drugorzędnych wykorzystując rozwiązanie trzecie z podrozdziału 17.1.3. Przykładowo, możemy posiadać indeksy na polach KODDZIAŁU, KODSTANOWISKA oraz PENSJA pliku PRACOWNIK z ćwiczenia 17.18. Opisz wydajny sposób wyszukiwania i pobierania rekordów spełniających złożony warunek selekcji dla tych pól, na przykład (KODDZIAŁU = 5 AND KODSTANOWISKA = 12 AND PENSJA = 50000), używając wskaźników rekordów z poziomu pośredniego.
- 17.25. Dostosuj algorytmy 17.2 i 17.3, stanowiące ogólną implementację operacji przeszukiwania i wstawiania dla B<sup>+</sup>-drzew, do charakterystyki B-drzew.
- 17.26. Istnieje możliwość zmodyfikowania algorytmu wstawiania dla B<sup>+</sup>-drzew w celu opóźnienia występowania przypadku, w którym musi zostać utworzony nowy poziom, poprzez sprawdzanie możliwości *redystrybucji* wartości pomiędzy wierzchołki liści. Na rysunku 17.17 przedstawiono, w jaki sposób można by tego dokonać dla przykładu z rysunku 17.12. Zamiast dzielić pierwszy od lewej strony liść w momencie wstawiania wartości 12, dokonujemy *lewostronnej redystrybucji*, przenosząc wartość 7 do wierzchołka liścia znajdującego się po lewej stronie (o ile wierzchołek ten posiada wolne miejsce). Rysunek 17.17 pokazuje, jak wyglądałoby drzewo w przypadku uwzględnienia redystrybucji (możesz też rozważyć *redystrybucję prawostronną*). Spróbuj zmodyfikować algorytm wstawiania do B<sup>+</sup>-drzewa tak, aby uwzględnić możliwość redystrybucji.
- 17.27. Przedstaw zarys algorytmu operacji usuwania z B<sup>+</sup>-drzewa.
- 17.28. Powtórz ćwiczenie 17.27 dla B-drzewa.

## Wybrane publikacje

**Indeksowanie.** B-drzewa i związane z nimi algorytmy wprowadzili Bayer i McCreight (1972). Pozycja Comer (1979) stanowi doskonałe studium B-drzew i ich historii, a także ich odmian. Knuth (1998) przedstawia szczegółową analizę wielu technik wyszukiwania, w tym B-drzew i pewnych ich odmian. Nievergelt (1974) omawia użycie drzew wyszukiwania binarnego jako metody organizacji plików. Podręczniki poświęcone strukturom plikowym, w tym pozycje Claybrooka (1983), Smitha i Barnesa (1987) oraz Salzberga (1988); podręcznik dotyczący algorytmów i struktur danych Wirtha (1985); a także podręcznik opisujący bazy danych Ramakrishnana i Gehrkego (2003) zawierają szczegółowe omówienie indeksowania i można w nich znaleźć algorytmy wyszukiwania, wstawiania i usuwania danych z B-drzew oraz B<sup>+</sup>-drzew. Larson (1981) analizuje indeksowane pliki sekwencyjne, zaś Held i Stonebraker (1978) porównują statyczne indeksy wielopoziomowe z dynamicznymi indeksami B-drzew. Lehman i Yao (1981) oraz Srinivasan i Carey (1991) dokonali dalszej analizy współbieżnego dostępu do struktur B-drzew. Książki Wiederholda (1987), Smitha i Barnesa (1987) oraz Salzberga (1988) omawiają między innymi wiele technik wyszukiwania opisanych w niniejszym rozdziale. Pliki matrycowe wprowadził Nievergelt (1984). Pobieranie z dopasowaniem częściowym, wykorzystujące mieszanie partycjonowane, zostało omówione w pozycjach Burkharda (1976, 1979).

RYSUNEK 17.17. Operacja wstawiania do B<sup>+</sup>-drzewa z lewostronnym ponownym rozkładem

Nowe techniki i zastosowania indeksów opartych na B<sup>+</sup>-drzewach omówiono w pracach Lanki i Maysa (1991), Zobela i in. (1992) oraz Faloutsosa i Jagadisha (1992). Mohan i Narang (1992) omawiają problematykę tworzenia indeksów. Wydajność działania różnych algorytmów operujących na B-drzewach i B<sup>+</sup>-drzewach omówiono w pracach Baeza-Yatesa i Larsona (1989) oraz Johnsona i Shasha'ego (1993). W pozycji Chana i in. (1992) zawarto omówienie zarządzania buforami. Kolumnowe bazy danych zostały zaproponowane w pozycji Stonebrakera i in. (2005) w kontekście systemu baz danych C-Store. Inną implementacją tego pomysłu jest system MonetDB/X100 Boncza i in. (2008). Abadi i in. (2008) opisali zalety baz kolumnowych w porównaniu z bazami wierszowymi w aplikacjach baz danych wymagających tylko odczytu.

**Fizyczne projektowanie baz danych.** Wiederhold (1987) omawia wszystkie etapy projektowania baz danych ze szczególnym uwzględnieniem projektów fizycznych. O'Neil i O'Neil (2001) prezentują szczegółowe omówienie projektowania fizycznego oraz kwestii dotyczących transakcji w kontekście komercyjnych relacyjnych SZBD. Navathe i Kerschberg (1986) omawiają wszystkie etapy projektowania baz danych i podkreślają rolę słowników danych. Rozen i Shasha (1991) oraz Carlis i March (1984) prezentują

różne modele problemów związanych z fizycznym projektowaniem baz danych. Shasha i Bonnet (2002) przedstawiają szczegółowe omówienie fizycznego projektowania baz danych. Pozycja Niemca (2008) to jedna z kilku dostępnych książek poświęconych zarządzaniu bazami Oracle i ich dostrajaniu. Schneider (2006) opisuje głównie projektowanie i dostrajanie baz MySQL.

# VIII

---

## Przetwarzanie i optymalizacja zapytań





## Strategie przetwarzania zapytań<sup>1</sup>

W niniejszym rozdziale zostaną omówione techniki wykorzystywane przez SZBD w celu przetwarzania wysokopoziomowych zapytań. Zapytanie wyrażone w wysokopoziomowym języku zapytań, takim jak SQL, musi najpierw zostać odczytane, poddane analizie składniowej i zweryfikowane<sup>2</sup>. **Czytnik** (ang. *scanner*) identyfikuje elementy języka — takie jak słowa kluczowe SQL, nazwy atrybutów oraz nazwy relacji — w tekście zapytania, natomiast **analizator składniowy** (ang. *parser*) sprawdza składnię zapytania w celu określenia, czy sformułowano je zgodnie z regułami składniowymi (regułami gramatyki) języka zapytań. Zapytanie musi również zostać **zweryfikowane** (ang. *validated*) poprzez sprawdzenie, czy wszystkie nazwy atrybutów i relacji są poprawnymi i semantycznie znaczącymi nazwami w schemacie określonej bazy danych, w której wykonywane jest zapytanie. Następnie tworzona jest wewnętrzna reprezentacja zapytania, zwykle stanowiąca drzewiastą strukturę danych określaną mianem **drzewa zapytania** (ang. *query tree*). Istnieje również możliwość reprezentowania zapytania przy użyciu grafowej struktury danych, określanej mianem **grafu zapytania** (ang. *query graph*); zwykle jest nim **skierowany graf acykliczny**. Następnie SZBD musi określić **strategię wykonania** (ang. *execution strategy*) dla pobrania wyników zapytania z plików bazy danych. Zapytanie zwykle posiada wiele możliwych strategii wykonania a proces wyboru najlepszej spośród nich określa się mianem **optymalizacji zapytania** (ang. *query optimization*).

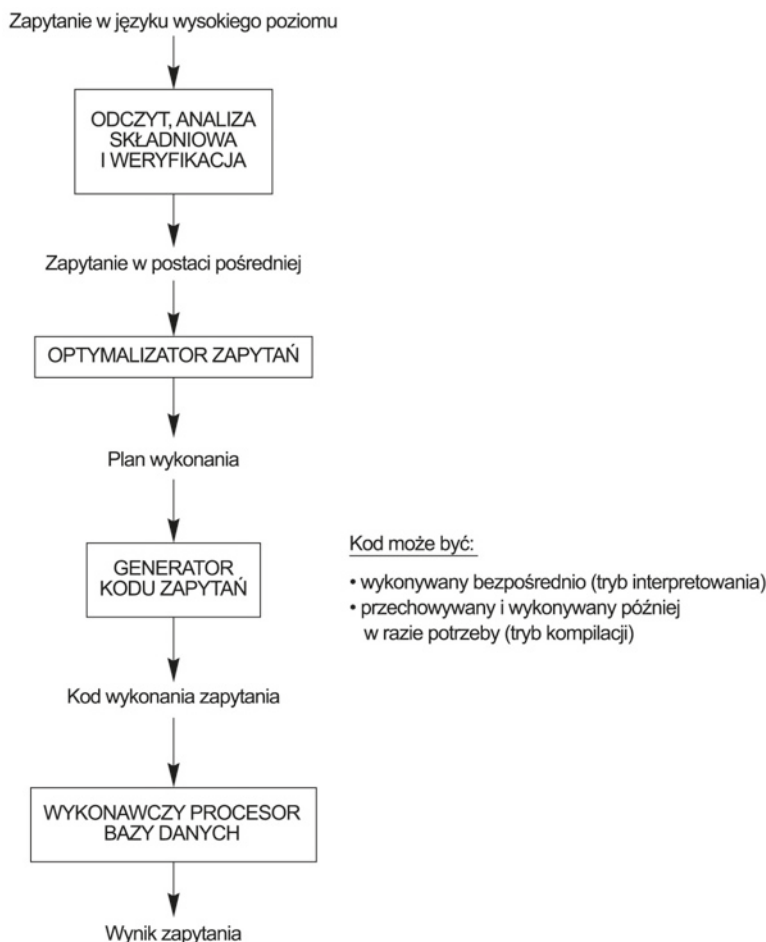
Szczegółowe omówienie optymalizacji zapytań odkładamy do następnego rozdziału. W tym rozdziale koncentrujemy się głównie na tym, jak zapytania są przetwarzane i jakie algorytmy są używane do wykonywania poszczególnych operacji w ramach zapytania. Na rysunku 18.1 przedstawiono kolejne etapy przetwarzania zapytania wysokopoziomowego. Zadaniem modułu **optymalizatora zapytań** (ang. *query optimizer*) jest utworzenie dobrego planu wykonania, zaś **generator kodu** (ang. *code generator*) generuje kod służący do wykonania tego planu. **Wykonawczy procesor bazy danych** (ang. *runtime database processor*) ma za zadanie wykonać kod zapytania, czy to w postaci skompilowanej, czy w trybie interpretowania, w celu uzyskania wyniku zapytania. W razie pojawienia się błędu czasu wykonania wykonawczy procesor bazy danych generuje komunikat o błędzie.

Pojęcie *optymalizacji* w rzeczywistości stanowi nieporozumienie, gdyż w pewnych przypadkach wybrany plan wykonania nie jest optymalną (najlepszą) strategią i stanowi jedynie *wystarczająco wydajną lub najlepszą dostępną strategię* dla wykonania zapytania.

---

<sup>1</sup> Dziękujemy Rafiemu Ahmedowi za wkład w aktualizację tego rozdziału.

<sup>2</sup> Nie będziemy tu omawiać fazy analizy składniowej i sprawdzania składni — materiał ten omawiają podręczniki poświęcone kompilatorom.



RYSUNEK 18.1. Typowe etapy przetwarzania wysokopoziomowego zapytania

Znalezienie optymalnej strategii jest zwykle zbyt czasochłonne, poza przypadkami najprostszych zapytań, i może wymagać dostępu do precyzyjnych i szczegółowych informacji dotyczących wielkości tabel lub rozkładu wartości kolumn; informacje te mogą nie być w całości dostępne w katalogu SZBD. Ponadto na podstawie predykatów z zapytania trzeba określać dodatkowe informacje, takie jak wielkość oczekiwanych wyników. Stąd być może lepszym określeniem dla *optymalizacji zapytań* byłoby *planowanie strategii wykonania*.

W przypadku niskopoziomowych nawigacyjnych języków baz danych w starych systemach — takich jak język DML systemów sieciowych lub język DL/1 systemów hierarchicznych — programista musi wybrać strategię wykonania zapytania, pisząc program bazodanowy. Jeżeli SZBD oferuje jedynie język nawigacyjny, istnieje *ograniczona sposobność* częstego wykorzystywania optymalizacji zapytań przez SZBD. Zamiast tego programista posiada możliwość wybrania strategii wykonania. Z drugiej strony, wysokopoziomowy język zapytań — taki jak SQL w przypadku relacyjnych SZBD (SZRBD) lub OQL

(patrz rozdział 12.) w przypadku obiektowych SZBD (SZOBD) — ze swej natury charakteryzuje się większą deklaratywnością, ponieważ określa, jakie są zamierzone wyniki zapytania, a nie identyfikuje szczegóły *sposobu* otrzymania wyników. Optymalizacja zapytań jest zatem konieczna w przypadku zapytań definiowanych w wysokopoziomowym języku zapytań.

Poniżej skoncentrujemy się na opisie optymalizacji zapytań w *kontekście SZRBD*, ponieważ wiele z opisywanych technik zaadaptowano do użytku w innym rodzaju systemach zarządzania bazami danych, np. w SZOBD<sup>3</sup>. Relacyjny SZBD musi w systematyczny sposób ocenić alternatywne strategie wykonania zapytania i wybrać strategię wystarczająco wydajną lub optymalną. Każdy SZBD posiada zwykle wiele ogólnych algorytmów użytkowania dostępu do bazy danych, które implementują operacje algebry relacyjnej, takie jak `SELECT` lub `JOIN` (patrz rozdział 8.), lub kombinacje tych operacji. Jedynie strategie wykonania, które można zaimplementować przy użyciu algorytmów dostępu SZBD i które pasują do określonego zapytania i *określonego projektu fizycznego bazy danych*, mogą być rozpatrywane przez moduł optymalizacji zapytań.

Oto struktura tego rozdziału: w podrozdziale 18.1 zostanie przedstawione ogólne omówienie sposobów translacji zapytań języka SQL do postaci zapytań algebry relacji i dodatkowych operacji, a następnie ich optymalizacji. Później, w podrozdziałach od 18.2 do 18.6, zostaną omówione algorytmy implementujące operacje relacyjne. W podrozdziale 18.7 omówimy strategię wykonywania nazywaną przetwarzaniem potokowym. Podrozdział 18.8 zawiera krótki przegląd strategii równoległego wykonywania operatorów. Podrozdział 18.9 to podsumowanie rozdziału.

W następnym rozdziale zostanie przedstawiony przegląd strategii optymalizacji zapytań. Można tu wyróżnić dwie główne techniki. Pierwsza z nich opiera się na **regułach heurystycznych** (ang. *heuristic rules*) w zakresie szeregowania operacji w ramach strategii wykonania zapytania. Reguły heurystyczne sprawdzają się w większości sytuacji, ale nie gwarantują poprawnego działania w każdym przypadku. Reguły zwykle określają kolejność wykonywania operacji w drzewie zapytania. Druga technika wiąże się z **szacowaniem kosztów** (ang. *cost estimating*) różnych strategii wykonania i wybraniem planu wykonania o najniższym szacowanym koszcie. Tematy omawiane w tym rozdziale wymagają opanowania materiału z kilku wcześniejszych rozdziałów. Potrzebna jest znajomość przede wszystkim rozdziałów poświęconych językowi SQL (rozdziały 6. i 7.), algebrze relacyjnej (rozdział 8.) oraz strukturom plików i indeksowaniu (rozdziały 16. i 17.). Należy też zauważyć, że przetwarzanie i optymalizacja zapytań to rozbudowane zagadnienie, a w tym i następnym rozdziale możemy przedstawić tylko wprowadzenie do podstawowych zasad i technik. W bibliografii tego i następnego rozdziału wymienionych jest kilka ważnych tekstów.

---

<sup>3</sup> Pewne problemy i techniki przetwarzania i optymalizacji zapytań są związane tylko z systemami SZOBD, jednak nie będziemy ich omawiać, gdyż w tym rozdziale prezentujemy tylko wprowadzenie do problematyki przetwarzania zapytań, a optymalizację przedstawiamy w rozdziale 19.

## 18.1. Translacja zapytań języka SQL do postaci wyrażeń algebry relacji i innych operacji

W praktyce SQL jest językiem zapytań używanym przez większość komercyjnych SZRBD. Zapytanie języka SQL jest najpierw tłumaczone na równoważne mu wyrażenie rozszerzonej algebry relacji — reprezentowane jako struktura danych drzewa zapytania — które podlega optymalizacji. Zwykle zapytania SQL są rozkładane na *bloki zapytania* (ang. *query blocks*), które stanowią podstawową jednostkę, jaka może być tłumaczona na operatory algebriczne i optymalizowana. **Blok zapytania** zawiera pojedyncze wyrażenie SELECT-FROM-WHERE, jak również klauzule GROUP BY i HAVING, jeżeli stanowią one część bloku. Dzięki temu zapytania zagnieżdżone występujące w zapytaniu zostają zidentyfikowane jako oddzielne bloki zapytań. Ze względu na fakt, że język SQL uwzględnia operatory agregujące — takie jak MAX, MIN, SUM i COUNT — operatory te również muszą zostać uwzględnione w ramach algebry rozszerzonej, co omówiono w podrozdziale 8.4.

Rozważmy następujące zapytanie SQL na relacji PRACOWNIK z rysunku 5.5:

```
SELECT NAZWISKO, IMIĘ
FROM   PRACOWNIK
WHERE  PENSJA > (SELECT MAX (PENSJA)
                  FROM   PRACOWNIK
                  WHERE  NRDZ=5);
```

To zapytanie pobiera nazwiska i imiona pracowników (z dowolnego działu firmy), których pensja jest wyższa od *najwyższej pensji z działu 5*. Zapytanie to zawiera podzapytanie zagnieżdżone, a zatem zostanie rozłożone na dwa bloki. Blok wewnętrzny ma postać:

```
(SELECT MAX (PENSJA)
FROM   PRACOWNIK
WHERE  NRDZ=5)
```

Ten fragment pobiera najwyższą pensję z działu 5. Blok zewnętrzny ma postać:

```
SELECT NAZWISKO, IMIĘ
FROM   PRACOWNIK
WHERE  PENSJA > C
```

gdzie C reprezentuje wynik zwrócony przez blok wewnętrzny. Blok wewnętrzny można przetłumaczyć do postaci wyrażenia rozszerzonej algebry relacji jako:

$$\mathfrak{S}_{\text{MAX PENSJA}}(\sigma_{\text{NRDZ}=5}(\text{PRACOWNIK}))$$

zaś blok zewnętrzny do postaci wyrażenia:

$$\pi_{\text{NAZWISKO, IMIĘ}}(\sigma_{\text{PENSJA}>C}(\text{PRACOWNIK}))$$

Następnie *optymalizator zapytań* wybiera plan wykonania dla każdego bloku. Należy zauważyć, że w powyższym przykładzie blok wewnętrzny musi zostać wykonany tylko raz w celu określenia maksymalnej pensji, używanej później — jako stała C — przez blok zewnętrzny. W punkcie 7.1.2 takie zapytanie nazwaliśmy *zagnieżdżonym blokiem podzapytania (nieskorelowanym z blokiem zewnętrznym)*. Znacznie większy problem pod wzglę-

dem optymalizacji stanowią bardziej skomplikowane *skorelowane zapytania zagnieżdżone* (patrz punkt 7.1.3), gdzie zmienna krotki z bloku zewnętrznego występuje w klauzuli WHERE bloku wewnętrznego. W zaawansowanych SZBD stosowanych jest wiele technik eliminowania zagnieżdżenia i optymalizowania skorelowanych podzapytań zagnieżdżonych.

### 18.1.1. Dodatkowe operatory połączeń częściowych i antyłączeń

Obecnie większość relacyjnych SZBD przetwarza zapytania języka SQL generowane przez różne aplikacje dla przedsiębiorstw. Te zapytania to: zapytania doraźne, standardowe zapytania zapuszkowane z parametrami i zapytania związane z generowaniem rekordów. Ponadto zapytania SQL pochodzą z aplikacji OLAP i dotyczą hurtowni danych (szczegółowe omówienie hurtowni danych zawiera rozdział 29.). Niektóre z tych zapytań są przekształcane na operacje nienależące do opisanej w rozdziale 8. standardowej algebry relacyjnej. Dwie często stosowane operacje to **złączenia częściowe** i **antyłączenia**. Warto zauważyć, że obie te operacje to rodzaje połączeń. Złączenia częściowe są zwykle używane do eliminowania zagnieżdżenia podzapytań EXISTS, IN i ANY<sup>4</sup>. Złączenia częściowe są tu reprezentowane za pomocą niestandardowej składni T1.X S= T2.Y, gdzie T1 to tabela lewostronna, a T2 to tabela prawostronna złączenia częściowego. Oto semantyka połączeń częściowych: wiersz T1 jest zwracany, gdy w T1.X znalezione zostanie dopasowanie do dowolnej wartości T2.Y; dalsze dopasowania nie są wyszukiwane. Stoi to w kontraście z wyszukiwaniem wszystkich możliwych dopasowań w połączeniu zewnętrznym.

Przyjrzyj się nieco zmodyfikowanej wersji schematu z rysunku 5.5:

```
PRACOWNIK(PESEL, DATAUR, ADRES, PŁEĆ, PENSJA, NRZD)
DZIAŁ(NUMERDZ, NAZWADZ, PESELKIER, KOD)
```

Dział jest tu lokalizowany na podstawie kodu pocztowego.

Zastanów się nad następującym zapytaniem:

```
Z(ZC): SELECT COUNT(*)
FROM DZIAŁ D
WHERE D.NUMERDZ IN ( SELECT E.NRZD
                     FROM PRACOWNIK E
                     WHERE E.PENSJA > 20000)
```

Występuje tu zapytanie zagnieżdżone łączane za pomocą łącznika IN.

Usuwanie zapytania zagnieżdżonego to **eliminowanie zagnieżdżenia**:

```
( SELECT E.NRZD
FROM PRACOWNIK WHERE E.PENSJA > 20000)
```

Prowadzi to do powstania następującego zapytania z operacją **złączenia częściowego**<sup>5</sup>, zapisywanego dalej za pomocą niestandardowej notacji „S=”:

<sup>4</sup> W pewnych sytuacjach, gdy duplikaty nie mają znaczenia, można też stosować złączenia wewnętrzne do eliminowania zagnieżdżenia podzapytań EXISTS i ANY.

<sup>5</sup> Warto zauważyć, że ten operator złączenia częściowego różni się od operatora stosowanego do przetwarzania zapytań rozproszonych.

```
SELECT COUNT(*)
FROM PRACOWNIK E, DZIAŁ D
WHERE D.NUMERDZ S= E.NRDZ and E.PENSJA > 20000;
```

To zapytanie zlicza działy zatrudniające pracowników zarabiających powyżej 20 000 złotych miesięcznie. Operacja polega tu na znalezieniu działu, którego atrybut `NUMERDZ` pasuje do wartości atrybutu `NRDZ` pracownika z tak wysoką pensją.

W algebrze stosowane są różne notacje. Często używana notacja jest pokazana poniżej.

Złączenie częściowe:



Teraz przyjrzyj się następnemu zapytaniu:

```
Z (AZ) : SELECT COUNT(*)
          FROM PRACOWNIK
          WHERE PRACOWNIK.NRDZ NOT IN (SELECT DZIAŁ.NUMERDZ
                                         FROM DZIAŁ
                                         WHERE KOD=30332)
```

To zapytanie zlicza pracowników, którzy *nie pracują* w działach o kodzie 30-332. Tu operacja polega na znalezieniu krotek pracowników, których atrybut `NRDZ` *nie pasuje* do wartości atrybutu `NUMERDZ` działów o podanym kodzie pocztowym. Interesuje nas tylko liczba takich pracowników, a wykonanie złączenia wewnętrznego dwóch tabel da oczywiście błędny wynik. W takiej sytuacji do eliminowania zagnieżdżenia używany jest operator **antyłączenia**.

Antyzłączenie służy do eliminowania zagnieżdżenia podzapytań `NOT EXISTS`, `NOT IN` i `ALL`. Antyzłączenie przedstawiamy za pomocą następującej niestandardowej składni:  $T1.x \text{ A} T2.y$ , gdzie  $T1$  to tabela lewostronna, a  $T2$  to tabela prawostronna antyzłączenia. Oto semantyka antyzłączeń: wiersz z  $T1$  jest odrzucany, gdy  $T1.x$  zostaje dopasowane do  $T2.y$ . Wiersz z  $T1$  jest zwracany, gdy  $T1.x$  nie pasuje do żadnej wartości  $T2.y$ .

Poniżej pokazany jest efekt eliminacji zagnieżdżenia. Antyzłączenie jest tu przedstawione za pomocą niestandardowego symbolu „A”:

```
SELECT COUNT(*)
FROM PRACOWNIK, DZIAŁ
WHERE PRACOWNIK.NRDZ A= DZIAŁ AND KOD=30332
```

W algebrze stosowane są różne notacje. Jedna z nich jest przedstawiona poniżej.

Antyzłączenie:





## 18.2. Algorytmy sortowania zewnętrznego

Algorytmy sortowania to jedne z najważniejszych algorytmów używanych w czasie przetwarzania zapytań. Przykładowo, kiedy zapytanie SQL określa klauzulę ORDER BY, wynik zapytania musi zostać posortowany. Sortowanie stanowi również kluczowy element algorytmów sortująco-scalających używanych w przypadku operacji JOIN i innych (takich jak UNION lub INTERSECTION) oraz w algorytmach eliminacji duplikatów wartości dla operacji PROJECT (kiedy zapytanie SQL określa opcję DISTINCT w klauzuli SELECT). W niniejszym podrozdziale omówimy jeden z takich algorytmów. Należy zauważyć, że sortowania można uniknąć, jeżeli istnieje odpowiedni indeks (np. główny lub klastrowania), umożliwiający uzyskiwanie uporządkowanego dostępu do rekordów.

Pojęcie **sortowania zewnętrznego** (ang. *external sorting*) odnosi się do algorytmów sortowania odpowiednich dla dużych plików rekordów składowanych na dysku, które nie mieszczą się w pamięci głównej, jak ma to miejsce w przypadku większości plików bazy danych<sup>6</sup>. Typowy algorytm sortowania zewnętrznego wykorzystuje **strategię sortująco-scalającą** (ang. *sort-merge strategy*), której działanie rozpoczyna się od posortowania niewielkich podplików — zwanych **jednostkami** (ang. *runs*) — pliku głównego, a następnie scalenia posortowanych jednostek, co powoduje utworzenie większych posortowanych podplików. Następnie scala się również i te podpliki. Algorytm sortująco-scalający, podobnie jak inne algorytmy bazodanowe, wymaga istnienia w pamięci *przestrzeni bufora*, gdzie jest wykonywane faktyczne sortowanie i scalanie jednostek. Podstawowy algorytm, przedstawiony poniżej na rysunku 18.2, składa się z dwóch faz: fazy sortowania oraz fazy scalania. Przestrzeń bufora w pamięci głównej to część **pamięci podręcznej SZBD**. Jest to obszar w pamięci głównej komputera kontrolowany przez SZBD. Przestrzeń bufora jest podzielona na pojedyncze **bufory** o wielkości równej rozmiarowi bloku dysku. Tak więc jeden bufor mieści dokładnie *jeden blok dysku*.

```

ustaw  $i \leftarrow 1$ ;
 $j \leftarrow b$ ; {rozmiar pliku w blokach}
 $k \leftarrow n_B$ ; {rozmiar bufora w blokach}
 $m \leftarrow \lceil (j/k) \rceil$ ; {liczba jednostek, z których każda mieści się w pliku}
{Faza sortowania}
while ( $i \leq m$ )
do {
    wczytaj kolejne  $k$  bloków pliku do bufora, a jeżeli pozostało mniej niż
         $k$  bloków, wczytaj pozostałe bloki;
    posortuj rekordy w buforze i zapisz je jako tymczasowy podplik;
     $i \leftarrow i+1$ ;
}
{Faza scalania: scalanie podplików dopóki nie zostanie 1 plik}

```

<sup>6</sup> Wewnętrzne algorytmy sortowania są odpowiednie dla sortowania struktur danych takich jak tabele i listy, które w całości mieszczą się w pamięci. Takie algorytmy są szczegółowo opisane w podręcznikach poświęconych algorytmom i strukturom danych. Te algorytmy to m.in.: techniki takie jak sortowanie szybkie, sortowanie przez kopcowanie, sortowanie bąbelkowe. Nie omawiamy ich w tym miejscu. SZBD działające w pamięci głównej, np. HANA, stosują własne techniki sortowania.

```

ustaw  $i \leftarrow 1$ ;
 $p \leftarrow \lceil \log_{k-1} m \rceil$  { $p$  jest liczbą powtórzeń fazy scalania}
 $j \leftarrow m$ ;
while ( $i \leq p$ )
do {
     $n \leftarrow 1$ ;
     $q \leftarrow \lceil (j/(k-1)) \rceil$ ; {liczba podplików zapisywanych w bieżącym przebiegu}
    while ( $n \leq q$ )
    do {
        wczytaj kolejne  $k-1$  podplików lub pozostałe podpliki (z poprzedniego
        przebiegu) po jednym bloku na raz;
        scal i zapisz jako nowy podplik po jednym bloku na raz;
         $n \leftarrow n+1$ ;
    }
     $j \leftarrow q$ ;
     $i \leftarrow i+1$ ;
}

```

RYSUNEK 18.2. Zarys algorytmu sortująco-scalającego dla celów sortowania zewnętrznego

W **fazie sortowania** (ang. *sorting phase*) jednostki (porcje) pliku, które mieszczą się w dostępnej przestrzeni bufora, są wczytywane do pamięci, sortowane przy użyciu algorytmu sortowania *wewnętrznego* i zapisywane z powrotem na dysku jako tymczasowe posortowane podpliki (lub jednostki). Rozmiar bloku i **liczba jednostek początkowych** ( $n_R$ ) zależą od **liczby bloków pliku** ( $b$ ) oraz **dostępnej przestrzeni bufora** ( $n_B$ ). Przykładowo, jeżeli  $n_B = 5$  bloków i rozmiar pliku  $b = 1024$  bloki, to  $n_R = \lceil (b/n_B) \rceil$ , czyli 205 jednostek początkowych, każda o rozmiarze 5 bloków (oprócz ostatniej, która liczy 4 bloki). Stąd po zakończeniu fazy sortowania 205 posortowanych jednostek jest przechowywanych jako tymczasowe podpliki na dysku.

W **fazie scalania** (ang. *merging phase*) posortowane jednostki są scalane w czasie jednego lub większej liczby **przebiegów** (ang. *passes*). Każdy przebieg może obejmować jeden lub więcej kroków. **Stopień scalenia** (ang. *degree of merging*)  $d_M$  jest liczbą jednostek, które można scalić ze sobą w każdym przebiegu. W każdym kroku potrzebny jest jeden blok bufora w celu przechowywania jednego bloku z każdej ze scalanych posortowanych jednostek i jeden dodatkowy bufor do przechowywania każdego bloku wyniku scalenia. Wynik to większy posortowany plik powstały po scaleniu kilku mniejszych posortowanych jednostek. Stąd  $d_M$  jest mniejszą spośród wartości  $(n_B-1)$  i  $n_R$ , zaś liczba przebiegów wynosi  $\lceil (\log_{d_M} n_R) \rceil$ . W naszym przykładzie  $n_B = 5$ , a  $d_M = 4$  (scalanie poczwórne), więc po zakończeniu pierwszego przebiegu 205 jednostek początkowych zostaje scalone po 4 w 52 jednostki. Te z kolei zostają scalone po 4 w 13 jednostek, później w 4 i wreszcie w 1 posortowany plik, co oznacza, że potrzebne są *cztery przebiegi*.

Wydajność algorytmu sortująco-scalającego można mierzyć w kategoriach liczby odczytów i zapisów bloków dysku (przenoszonych między dyskiem a pamięcią główną) potrzebnych do posortowania całego pliku. Przybliżeniem tego kosztu jest następujący wzór:

$$(2*b) + (2*b*(\log_{d_M} n_R))$$

Pierwszy składnik reprezentuje liczbę operacji dostępu do bloków dla fazy sortowania, gdyż dostęp do każdego bloku pliku odbywa się dwukrotnie — raz w celu wczytania go do pamięci i drugi raz w celu zapisania rekordów z powrotem na dysku po zakończeniu sortowania. Drugi składnik reprezentuje liczbę operacji dostępu do bloków w fazie scalania. W każdym przebiegu scalania liczba wczytywanych i zapisywanych bloków dysku jest w przybliżeniu równa początkowej liczbie bloków —  $b$ . Ponieważ przebiegów jest  $(\log_{d_M} n_R)$ , łączny koszt scalania wynosi  $(2*b*(\log_{d_M} n_R))$ .

Minimalna liczba potrzebnych buforów w pamięci głównej to  $n_B = 3$ , co oznacza  $d_M = 2$  i  $n_R = \lceil (b/3) \rceil$ . Minimalna wartość  $d_M = 2$  prowadzi do najniższej wydajności algorytmu równej:

$$(2*b)+(2*(b*(\log_2 n_R)))$$

W dalszych podrozdziałach omawiamy różne algorytmy operacji algebry relacyjnej (patrz rozdział 8.).

## 18.3. Algorytmy operacji selekcji

### 18.3.1. Możliwości implementacji operacji SELECT

Istnieje wiele algorytmów wykonywania operacji SELECT. Jest to w swej istocie operacja wyszukiwania w pliku dyskowym rekordów spełniających określony warunek. Niektóre z nich są uzależnione od posiadania przez plik określonych ścieżek dostępu i mogą mieć zastosowanie w przypadku tylko określonego rodzaju warunków selekcji. W niniejszym podrozdziale zostaną omówione niektóre z algorytmów implementacji operacji SELECT. W celu zilustrowania prowadzonego wykładu będziemy używać następujących operacji, określonych na relacyjnej bazie danych z rysunku 5.5:

```
(OP1):  $\sigma_{PESEL='65010912345'}$ (PRACOWNIK)
(OP2):  $\sigma_{NUMERDZ>5}$ (DZIAŁ)
(OP3):  $\sigma_{NRDZ=5}$ (PRACOWNIK)
(OP4):  $\sigma_{NRDZ=5 \text{ AND } PENSJA>30000 \text{ AND } PLEC='K'}$ (PRACOWNIK)
(OP5):  $\sigma_{PESELPRAC='65010912345' \text{ AND } NR\_PROJ=10}$ (PRACUJE_NAD)
```

(OP6): Zapytanie SQL:

```
SELECT *
FROM PRACOWNIK
WHERE NRDZ IN (3, 27, 49)
```

OP7: Zapytanie SQL (z punktu 17.5.3)

```
SELECT IMIE, NAZWISKO
FROM PRACOWNIK
WHERE ((PENSJA*PROWIZJA_PROC) + PENSJA) > 15000;
```

**Metody wyszukiwania w przypadku prostych operacji selekcji.** Do selekcji rekordów z pliku można używać wielu algorytmów. Są to tak zwane **przeglądy plików** (ang. *file scans*), ponieważ przeglądają one rekordy pliku w celu wyszukania i pobrania rekordów

spełniających warunek selekcji<sup>7</sup>. Jeżeli algorytm wyszukiwania uwzględnia użycie indeksu, jego przeszukiwanie określa się mianem **przeglądu indeksu** (ang. *index scan*). Poniższe metody wyszukiwania (od S1 do S6) stanowią przykłady niektórych algorytmów wyszukiwania, których można używać w celu zaimplementowania operacji selekcji:

- **S1. Wyszukiwanie liniowe (metoda siłowa):** Pobieramy *każdy rekord* z pliku i sprawdzamy, czy wartość jego atrybutu spełnia warunek selekcji. Ponieważ rekordy są pogrupowane w blokach dysku, każdy blok jest wczytywany do bufora w pamięci głównej, a następnie przeszukiwanie rekordów w bloku dysku odbywa się w pamięci głównej.
- **S2. Wyszukiwanie binarne:** Jeżeli warunek selekcji zawiera porównanie równościowe na atrybucie klucza, względem którego jest **uporządkowany** plik, może zostać wykorzystane wyszukiwanie binarne (które jest wydajniejsze od wyszukiwania liniowego). Odpowiednim przykładem jest tu operacja OP1, jeżeli atrybut PESEL jest atrybutem uporządkowania dla pliku PRACOWNIK<sup>8</sup>.
- **S3a. Użycie indeksu głównego:** Jeżeli warunek selekcji zawiera porównanie równościowe na **atrybucie klucza** z indeksem głównym — na przykład PESEL = '65010912345' w operacji OP1 — używamy indeksu głównego w celu pobrania rekordu. Należy zauważyć, że warunek ten powoduje wybranie (najwyżej) jednego rekordu.
- **S3b. Użycie klucza mieszającego:** Jeżeli warunek selekcji zawiera porównanie równościowe na **atrybucie klucza** z kluczem mieszającym — na przykład PESEL = '65010912345' w operacji OP1 — używamy klucza mieszającego w celu pobrania rekordu. Należy zauważyć, że warunek ten powoduje wybranie (najwyżej) jednego rekordu.
- **S4. Użycie indeksu głównego w celu pobrania wielu rekordów:** Jeżeli warunkiem porównania jest relacja >, >=, < lub <= na polu klucza z indeksem głównym — na przykład NUMERDZ > 5 w operacji OP2 — używamy indeksu w celu znalezienia rekordu spełniającego odpowiedni warunek równościowy (NUMERDZ = 5), a następnie pobieramy wszystkie kolejne rekordy z (uporządkowanego pliku). Dla warunku NUMERDZ < 5 pobieramy wszystkie wcześniejsze rekordy.
- **S5. Użycie indeksu klastrowania w celu pobrania wielu rekordów:** Jeżeli warunek selekcji zawiera porównanie równościowe na **atrybucie niebędącym polem klucza** z indeksem klastrowania — na przykład NRDZ = 5 w przypadku operacji OP3 — używamy indeksu w celu pobrania wszystkich rekordów spełniających ten warunek.
- **S6. Użycie indeksu drugorzędnego (B+-drzewa) na porównaniu równościowym:** ta metoda wyszukiwania może być użyta w celu pobrania pojedynczego rekordu, jeżeli pole indeksujące jest **kluczem** (posiada unikatowe wartości), lub w celu pobrania wielu rekordów, jeżeli pole indeksujące **nie jest**

<sup>7</sup> Operacja selekcji niekiedy jest określana jako **filtr**, ponieważ służy do odfiltrowywania rekordów w pliku, które *nie* spełniają warunku selekcji.

<sup>8</sup> Ogólnie rzecz biorąc, wyszukiwanie binarne nie jest wykorzystywane w zakresie przeszukiwania baz danych, gdyż pliki uporządkowane nie są używane, o ile nie posiadają również odpowiedniego indeksu głównego.

**kluczem.** Można jej również używać w przypadku porównań uwzględniających relacje  $>$ ,  $>=$ ,  $<$  lub  $<=$ . Zapytania z zakresami w warunkach selekcji (np.  $3000 \leq \text{PENSJA} \leq 4000$ ) są nazywane **zapytaniem zakresowym**. W zapytaniach zakresowych liście indeksu w postaci B+-drzewa zawierają uporządkowane wartości pola indeksowania, dlatego używana jest sekwencja tych liści odpowiadająca żadanemu zakresowi wartości danego pola i zwracane są wskaźniki do rekordów zgodnych z warunkiem.

- **S7a. Użycie indeksu bitmapowego** (patrz punkt 17.5.2): Jeżeli warunek selekcji obejmuje zestaw wartości atrybutu (np.  $\text{NRDZ IN (3, 27, 49)}$  w operacji 6.), można zastosować operacje LUB na bitmapach odpowiadających każdej z tych wartości, aby uzyskać zestaw identyfikatorów rekordów spełniających warunek. W tym przykładzie zadanie sprowadza się do wykonania operacji LUB na trzech bitmapach o długości odpowiadającej liczbie pracowników.
- **S7b. Użycie indeksu funkcyjnego** (patrz punkt 17.5.3): W operacji 7. warunek selekcji obejmuje wyrażenie  $((\text{PENSJA} * \text{PROWIZJA\_PROC}) + \text{PENSJA})$ . Jeśli zdefiniowany jest indeks funkcyjny (tak jak w punkcie 17.5.3):

```
CREATE INDEX income_ix
ON PRACOWNIK ((PENSJA * PROWIZJA_PROC) + PENSJA);
```

to można go zastosować do pobrania zgodnych z warunkiem rekordów pracowników. Zauważ, że dokładny zapis funkcji z instrukcji tworzącej indeks nie jest istotny.

W następnym rozdziale zostanie omówiony sposób określania wzorów służących do szacowania kosztów dostępu związanych z tymi metodami wyszukiwania pod względem liczby operacji dostępu do bloków oraz czasu dostępu. Metoda S1 (**wyszukiwanie liniowe**) ma zastosowanie w przypadku dowolnego pliku, ale wszystkie pozostałe metody są uzależnione od posiadania odpowiedniej ścieżki dostępu na atrybucie używanym w warunku selekcji. Metoda S2 (**wyszukiwanie binarne**) wymaga, by plik był posortowany według atrybutu wyszukiwania. Metody używające indeksu (S3a, S4, S5 i S6) są nazywane **wyszukiwaniem opartym na indeksie** i wymagają odpowiedniego indeksu na atrybucie używanym w wyszukiwaniu. Metody S4 i S6 mogą być używane w celu pobierania rekordów z określonego zakresu; określa się je mianem **zapytań zakresowych** (ang. *range queries*). Metoda S7a (**wyszukiwanie z użyciem indeksu bitmapowego**) nadaje się do pobierania danych, gdy atrybut musi pasować do zbioru podanych wartości. Metoda S7b (**wyszukiwanie z użyciem indeksu funkcyjnego**) jest odpowiednia, gdy dopasowanie odbywa się na podstawie zależnej od jednego lub kilku atrybutów funkcji używanej w indeksie funkcyjnym.

### 18.3.2. Metody wyszukiwania dla selekcji na podstawie warunku koniunktywnego

Jeżeli warunek operacji SELECT jest **warunkiem koniunktywnym** (ang. *conjunctive condition*), tzn. jeżeli składa się z kilku warunków prostych połączonych operatorami logicznymi AND, tak jak operacja OP4 przedstawiona powyżej, SZBD może użyć następujących dodatkowych metod w celu zaimplementowania operacji:

- **S8. Wybór koniunktywny przy użyciu pojedynczego indeksu:** Jeżeli atrybut związany z dowolnym **pojedynczym warunkiem prostym** w warunku koniunktywnym posiada ścieżkę dostępu, która pozwala na użycie jednej z metod od S2 do S6, używamy tego warunku w celu pobrania rekordów, a następnie sprawdzamy, czy każdy pobrany rekord *spełnia pozostałe warunki proste* z warunku koniunktywnego.
- **S9. Wybór koniunktywny przy użyciu indeksu złożonego:** Jeżeli w warunku koniunktywnym warunki równości dotyczą dwóch lub więcej atrybutów i na połączonych polach istnieje indeks złożony (lub mieszający) — na przykład, jeżeli indeks utworzono na kluczu złożonym (PESELPRAC, NR\_PROJ) pliku PRACUJE\_NAD dla operacji OP5 — możemy bezpośrednio użyć takiego indeksu.
- **S10. Wybór koniunktywny poprzez przecięcie zbiorów wskaźników na rekordy<sup>9</sup>:** Jeżeli na więcej niż jednym polu związanym z warunkami prostymi w warunku koniunktywnym istnieją indeksy drugorzędne (lub inne ścieżki dostępu) oraz jeżeli indeksy zawierają wskaźniki na rekordy (a nie wskaźniki na bloki), wówczas każdy indeks może zostać użyty w celu pobrania **zbioru wskaźników rekordów**, które spełniają pojedyncze warunki. **Przecięcie** (ang. *intersection*) tych zbiorów wskaźników rekordów daje w wyniku wskaźniki rekordów spełniające warunek koniunktywny, które są następnie używane w celu bezpośredniego pobrania tych rekordów. Jeżeli tylko niektóre warunki posiadają odpowiednie indeksy drugorzędne, każdy pobrany rekord jest dalej sprawdzany w celu określenia, czy spełnia pozostałe warunki<sup>10</sup>. W metodzie S10 przyjmujemy, że każdy z indeksów dotyczy *pola niebędącego kluczem* pliku, ponieważ jeśli jeden z warunków to równość z polem klucza, z całym warunkiem zgodny będzie tylko jeden rekord. Do selekcji koniunktywnej na podstawie kilku atrybutów można stosować także indeksy bitmapowe i funkcyjne opisane w metodzie S7. W selekcji koniunktywnej według kilku atrybutów do wynikowych bitmap stosowany jest operator AND, aby wygenerować listę identyfikatorów rekordów. To samo można zrobić, gdy jeden lub kilka zbiorów identyfikatorów rekordów uzyskano na podstawie indeksu funkcyjnego.

Zawsze, kiedy wybór określa pojedynczy warunek — tak, jak w przypadku operacji OP1, OP2 i OP3 — SZBD może jedynie sprawdzić, czy ścieżka dostępu istnieje na atrybucie związanym z tym warunkiem. Jeżeli istnieje ścieżka dostępu (np.: indeks, klucz mieszający, indeks bitmapowy lub posortowany plik), używana jest metoda odpowiadająca tej ścieżce. W przeciwnym razie można wykorzystać podejście oparte na metodzie siłowej wyszukiwania liniowego z metody S1. Optymalizacja zapytań dla operacji SELECT jest wymagana głównie w przypadku koniunktywnych warunków selekcji, kiedy *więcej niż jeden* z atrybutów występujących w warunku posiada ścieżkę dostępu. Optymalizator powinien wybrać ścieżkę dostępu, która pobiera *najmniejszą liczbę rekordów* w najbardziej wydajny sposób, szacując koszt różnych rozwiązań (patrz podrozdział 19.3) i wybierając metodę o najmniejszym szacowanym koszcie.

<sup>9</sup> Wskaźnik na rekord jednoznacznie identyfikuje rekord i oferuje dostęp do adresu rekordu na dysku. Stąd określa się go również mianem **identyfikatora rekordu**.

<sup>10</sup> Technika ta ma wiele odmian — na przykład, jeżeli indeksy są *indeksami logicznymi*, które przechowują wartości klucza głównego zamiast wskaźników na rekordy.



### 18.3.3. Metody wyszukiwania dla selekcji na podstawie alternatywy logicznej

W porównaniu z koniunktywnym warunkiem selekcji, warunek **alternatywy logicznej** (ang. *disjunctive condition*), gdzie warunki proste są połączone operatorem logicznym OR, a nie AND, jest znacznie trudniejszy w przetwarzaniu i optymalizacji. Przykładowo, rozważmy operację OP4':

(OP4') :  $\sigma_{NRDZ=5 \text{ OR } PENSJA>3000 \text{ OR } PLEC='K'}(PRACOWNIK)$

W przypadku takiego warunku rekordy spełniające warunek alternatywy logicznej stanowią **sumę teoriomnogościową** (ang. *union*) rekordów spełniających poszczególne warunki. Stąd, jeżeli *któryś* z warunków nie posiada ścieżki dostępu, jest się zmuszonym do wykorzystania metody siłowej wyszukiwania liniowego. Tylko wówczas, gdy ścieżka dostępu istnieje na *każdym* warunku, możemy zoptymalizować wybór, pobierając rekordy spełniające każdy z warunków (lub identyfikatory tych rekordów), a następnie zastosować operację *sumy teoriomnogościowej* w celu wyeliminowania duplikatów wartości.

Wszystkie metody od S1 do S7 można stosować do każdego warunku prostego generującego zbiór identyfikatorów rekordów. Optymalizator zapytań musi wybrać odpowiednią z nich w celu wykonania każdej operacji SELECT w zapytaniu. Taka optymalizacja wykorzystuje wzory określające koszt każdej możliwej metody dostępu, co zostanie omówione w podrozdziałach 19.4 i 19.5. Optymalizator wybiera metodę dostępu związaną z najmniejszym szacowanym kosztem.

### 18.3.4. Szacowanie selektywności warunku

Aby móc zminimalizować ogólny koszt wykonywania zapytań w kategoriach zużywanych zasobów i czasu reakcji, optymalizator zapytań pobiera wartościowe dane wejściowe z katalogu systemowego, gdzie znajdują się istotne informacje statystyczne na temat bazy.

**Informacje w katalogu bazy danych.** Typowy katalog relacyjnego SZBD zawiera następujące rodzaje informacji:

Dla każdej relacji (tabeli)  $r$  o schemacie  $R$  obejmującej  $r_R$  krotek:

- Liczbę wierszy (rekordów):  $|r(R)|$ . Liczbę tę określamy jako  $r_R$ .
- „Szerokość” relacji (czyli długość każdej krotki w relacji). Określamy ją jako  $R$ .
- Liczbę bloków zajmowanych przez relację w pamięci:  $b_R$ .
- Współczynnik blokowy  $bfr$ , czyli liczbę krotek na blok.

Dla każdego atrybutu  $A$  w relacji  $R$ :

- Liczbę różnych wartości  $A$  w  $R$ :  $LRW(A, R)$ .
- Wartości maksymalną i minimalną atrybutu  $A$  w  $R$ :  $\max(A, R)$  i  $\min(A, R)$ .

Zauważ, że w razie potrzeby można generować i przechowywać także wiele innych rodzajów statystyk. Jeśli istnieje indeks złożony na atrybutach  $\langle A, B \rangle$ , to istotna jest  $LRW(R, \langle A, B \rangle)$ . System stara się zapewniać, by te statystyki były możliwie aktualne. Jednak utrzymywanie pełnej aktualności jest uważane za zbędne, ponieważ koszty tego



w stosunkowo aktywnej bazie są zbyt wysokie. Do wielu z opisanych tu parametrów wrócimy w punkcie 19.3.2.

Kiedy optymalizator dokonuje wyboru między wieloma warunkami prostymi w koniunktywnym warunku selekcji, zwykle uwzględnia *selektywność* każdego z nich. **Selektywność** (ang. *selectivity*)  $s$  definiuje się jako współczynnik liczby rekordów (krotek), które spełniają warunek, do całkowitej liczby rekordów (krotek) w pliku (relacji), stąd  $s$  stanowi wartość z przedziału od 0 do 1. *Zerowa selektywność* oznacza, że żaden rekord nie spełnia warunku, zaś wartość 1 oznacza, że wszystkie rekordy spełniają warunek. Zwykle selektywność nie jest równa żadnej z tych skrajności i jest ułamkiem pozwalającym oszacować procent rekordów pobieranych z pliku.

Choć mogą nie być dostępne informacje o dokładnej selektywności wszystkich warunków, często w katalogu SZBD są przechowywane **oszacowania selektywności**, z których korzysta optymalizator. Przykładowo, w przypadku warunku równościowego na atrybucie klucza relacji  $r(R)$ ,  $s = 1/|r(R)|$ , gdzie  $|r(R)|$  oznacza liczbę krotek w relacji  $r(R)$ . W przypadku warunku równościowego na atrybucie o *odrębnych wartościach* wartość  $s$  można oszacować na podstawie wzoru  $(|r(R)|/i)/|r(R)|$  lub  $1/i$ , zakładając, że rekordy są **równomiernie rozłożone** między różne wartości. Przy takim założeniu  $|r(R)|/i$  rekordów spełnia warunek równościowy na takim atrybucie. Dla zapytań zakresowych z warunkiem selekcji zachodzą zależności:

$A \geq v$  przy założeniu równomiernego rozkładu;

$sl = 0$ , jeśli  $v > \max(A, R)$ ;

$sl = \max(A, r) - v / \max(A, R) - \min(A, R)$

Ogólnie rzecz biorąc, liczba rekordów spełniających warunek selekcji o selektywności  $s$  jest określana jako  $|r(R)| \cdot s$ . Im mniejszą wartość ma to oszacowanie, tym bardziej pożądane jest użycie danego warunku w celu pobrania rekordów. Dla atrybutów niebędących kluczem i mających LRW( $A, R$ ) różnych wartości rozkład wartości często nie jest równomierny.

Jeśli rozkład rekordów o poszczególnych różnych wartościach jest przechowywany przez SZBD w formie **histogramu**, można bardziej precyzyjnie oszacować liczbę rekordów spełniających dany warunek. Informacje z katalogu i histogramy omawiamy szczegółowo w punkcie 19.3.3.

## 18.4. Implementacja operacji JOIN

Operacja JOIN (złączenia) jest jedną z najbardziej czasochłonnych w przypadku przetwarzania zapytań. Wiele operacji złączeniowych spotykanych w zapytaniach jest typu EQUIJOIN (złączenie równościowe lub tożsamościowe) oraz NATURAL JOIN (złączenie naturalne), tak więc poniżej zajmiemy się tylko nimi, ponieważ prezentujemy tu tylko przegląd przetwarzania i optymalizacji zapytań. W dalszej części rozdziału pojęcie **złączenie** (ang. *join*) będzie się odnosiło do operacji EQUIJOIN (lub NATURAL JOIN).

Istnieje wiele możliwych sposobów implementacji **złączenia dwukierunkowego** (ang. *two-way join*), które oznacza złączenie dwóch plików. Złączenia uwzględniające więcej niż dwa pliki określa się mianem **złączeń wielokierunkowych** (ang. *multiway joins*).

Liczba możliwych sposobów wykonania złączeń wielokierunkowych rośnie bardzo szybko. W niniejszym podrozdziale zostaną omówione techniki implementacji *wyłącznie złączeń dwukierunkowych*. W celu zilustrowania prowadzonego wykładu będziemy się odwoływać do schematu relacyjnego z rysunku 5.5 — w szczególności do relacji PRACOWNIK, DZIAŁ i PROJEKT. Rozpatrywane algorytmy dotyczą operacji złączenia w postaci:

$$R \bowtie_{A=B} S$$

gdzie  $A$  i  $B$  są **atrybutami złączenia**, czyli zgodnymi w ramach domeny atrybutami relacji, odpowiednio,  $R$  i  $S$ . Omawiane metody można rozszerzyć na ogólniejsze formy złączeń. Przedstawimy cztery najczęściej wykorzystywane techniki wykonywania takiego złączenia, wykorzystując następujące operacje przykładowe:

(OP6): PRACOWNIK  $\bowtie_{NRDZ = NUMERDZ}$  DZIAŁ

(OP7): DZIAŁ  $\bowtie_{PESELKIEROWNIKA = PESEL}$  PRACOWNIK

### 18.4.1. Metody implementacji złączeń

- **J1. Złączenie pętli (bloków) zagnieżdżonych:** Jest to algorytm domyślny (siłowy), ponieważ nie wymaga żadnych specjalnych ścieżek dostępu do żadnego z plików uwzględnianych w złączeniu. Dla każdego rekordu  $t$  w pliku  $R$  (pętla zewnętrzna) pobieramy każdy rekord  $s$  z pliku  $S$  (pętla wewnętrzna) i sprawdzamy, czy oba rekordy spełniają warunek złączenia  $t[A] = s[B]$ <sup>11</sup>.
- **J2. Złączenie z pętlą pojedynczą (przy użyciu struktury dostępowej w celu pobrania pasujących rekordów):** Jeżeli na jednym z atrybutów — na przykład  $B$  w pliku  $S$  — podlegających złączeniu istnieje indeks (lub klucz mieszający), pobieramy każdy rekord  $t$  z pliku  $R$ , po jednym naraz (pętla dotycząca pliku  $R$ ), a następnie używamy struktury dostępowej (np. indeksu lub klucza mieszającego) w celu bezpośredniego pobrania wszystkich pasujących rekordów  $s$  z pliku  $S$ , spełniających warunek  $s[B] = t[A]$ .
- **J3. Złączenie sortująco-scalające:** Jeżeli rekordy plików  $R$  i  $S$  są *posortowane fizycznie* (uporządkowane) według wartości atrybutów złączenia, odpowiednio,  $A$  i  $B$ , możemy zaimplementować złączenie w najwydajniejszy z możliwych sposobów. Oba pliki są przeglądane równolegle w kolejności atrybutów złączenia i dopasowujemy rekordy posiadające te same wartości dla atrybutu  $A$  i  $B$ . Jeżeli pliki nie są posortowane, można tego dokonać przy użyciu techniki sortowania zewnętrznego (patrz podrozdział 18.2). W przypadku tej metody pary bloków pliku są po kolei kopiowane do buforów pamięci i rekordy każdego pliku przegląda się tylko raz (dopasowując do rekordów drugiego pliku) — chyba że zarówno  $A$ , jak i  $B$  są atrybutami niebędącymi kluczami. W takim przypadku należy nieco zmodyfikować działanie metody. W algorytmie 18.3(a) przedstawiono zarys działania złączenia sortująco-scalającego. Oznaczenie  $R(i)$  odnosi się do  $i$ -tego rekordu w pliku  $R$ . Odmiany złączenia sortująco-scalającego można użyć w przypadku, gdy indeksy drugorzędne istnieją na obu atrybutach złączenia. Indeksy dają możliwość uzyskania dostępu (przejrzenia) do rekordów w kolejności atrybutów złączenia, jednak same rekordy są

<sup>11</sup> W przypadku pliku dyskowego jest rzeczą oczywistą, że pętłe będą działały na blokach dyskowych, więc technikę tę nazywa się również *złączeniem bloków zagnieżdżonych*.

fizycznie rozrzucone po różnych blokach pliku, więc metoda ta może okazać się mało efektywna, gdyż każda operacja dostępu do rekordu może się wiązać z uzyskaniem dostępu do innego bloku.

- **J4. Partycjonowane złączenie mieszające (lub po prostu złączenie mieszające):**  
 Rekordy z plików  $R$  i  $S$  są dzielone na mniejsze pliki. Podział odbywa się na podstawie tej samej funkcji mieszającej  $h$  na atrybutach złączenia —  $A$  z pliku  $R$  (przy podziale pliku  $R$ ) oraz  $B$  z pliku  $S$  (przy podziale pliku  $S$ ). Początkowo, w pojedynczym przejściu przez plik liczący mniej rekordów (niech to będzie plik  $R$ ), używamy techniki mieszania na jego rekordach i zapisujemy je w pakietach pliku  $R$ . Jest to tak zwana **faza podziału** (ang. *partitioning phase*), gdyż rekordy pliku  $R$  zostają podzielone na pakiety mieszające. W najprostszej sytuacji zakładamy, że mniejszy plik po podziale w całości mieści się w pamięci głównej, dlatego podzielone podpliki pliku  $R$  są wszystkie przechowywane w tej pamięci. Zbiór rekordów o tej samej wartości  $h(A)$  jest umieszczany w tym samym pakiecie. Jest to **pakiet mieszający** w przechowywanej w pamięci głównej tabeli mieszającej. W drugiej fazie, noszącej nazwę **fazy sprawdzania** (ang. *probing phase*), w pojedynczym przejściu przez drugi plik ( $S$ ) używamy tej samej funkcji mieszającej  $h(B)$  na każdym z jego rekordów w celu *sprawdzenia* odpowiedniego pakietu i rekord ten zostaje połączony z wszystkimi odpowiadającymi mu rekordami z pliku  $R$ , znajdującymi się w danym pakiecie. W przypadku powyższego uproszczonego opisu złączenia mieszającego zakładamy, że mniejszy z dwóch plików *mieści się w całości w pakietach w pamięci* po zakończeniu pierwszej fazy. Poniżej zostanie omówiona ogólna odmiana złączenia mieszającego, gdzie nie jest wymagane przyjęcie takiego założenia. W praktyce techniki od J1 do J4 są implementowane poprzez uzyskiwanie dostępu do *całych bloków dyskowych* pliku, a nie pojedynczych rekordów. W zależności od dostępnej przestrzeni bufora w pamięci można dostosowywać liczbę bloków wczytywanych z pliku.

```
(a) posortuj krotki w pliku R według wartości atrybutu A; (*zakładamy, że plik R posiada
n krotek (rekordów) *)
posortuj krotki w pliku S według wartości atrybutu B; (*zakładamy, że plik S posiada
m krotek (rekordów) *)

ustaw i ← 1, j ← 1;
while (i ≤ n) and (j ≤ m)
do {
    if R(i)[A] > S(j)[B]
    then ustaw j ← j+1
    elseif R(i)[A] < S(j)[B]
    then ustaw i ← i+1
    else { (*R(i)[A] = S(j)[B], więc przekazujemy na wyjście dopasowaną krotkę *)
        prześlij połączoną krotkę <R(i), S(j)> do pliku T;
        (*przekazujemy pozostałe krotki pasujące do R(i) – o ile takie istnieją *)
        ustaw l ← j+1;
        while {l ≤ m} and (R(i)[A] = S(l)[B])
        do {
            prześlij połączoną krotkę <R(i), S(l)> do pliku T;
            ustaw l ← l+1;
        }
        (*przekazujemy pozostałe krotki pasujące do S(j) – o ile takie istnieją *)
```

```

        ustaw  $k \leftarrow i+1$ ;
        while  $\{k \leq n\}$  and  $(R(k)[A] = S(j)[B])$ 
        do {
            prześlij połączoną krotkę  $\langle R(k), S(j) \rangle$  do pliku  $T$ ;
            ustaw  $k \leftarrow k+1$ ;
        }
        ustaw  $i \leftarrow k$ ;  $j \leftarrow 1$ ;
    }
}

```

(b) utwórz krotkę  $t[\langle \text{lista atrybutów} \rangle]$  w pliku  $T'$  dla każdej krotki  $t$  w pliku  $R$   
*(\*plik  $T'$  zawiera wynik projekcji przed eliminacją duplikatów \*)*

if  $\langle \text{lista atrybutów} \rangle$  zawiera klucz pliku  $R$

then  $T \leftarrow T'$

else {

posortuj krotki w pliku  $T'$ ;

ustaw  $i \leftarrow 1$ ;  $j \leftarrow 2$ ;

while  $i \leq n$

do {

prześlij krotkę  $T'[i]$  do pliku  $T$ ;

while  $T'[i] = T'[j]$  and  $j \leq n$  do  $j \leftarrow j+1$ ; *(\*eliminacja duplikatów \*)*

$i \leftarrow j$ ;  $j \leftarrow i+1$ ;

}

}

*(\*plik  $T$  zawiera wynik projekcji po wyeliminowaniu duplikatów \*)*

(c) posortuj krotki w plikach  $R$  i  $S$ , używając tych samych unikatowych atrybutów sortowania;

ustaw  $i \leftarrow 1$ ,  $j \leftarrow 1$ ;

while  $(i \leq n)$  and  $(j \leq m)$

do {

if  $R(i) > S(j)$

then {

prześlij  $S(j)$  do pliku  $T$ ;

ustaw  $j \leftarrow j+1$ ;

}

elseif  $R(i) < S(j)$

then {

prześlij  $R(i)$  do pliku  $T$ ;

ustaw  $i \leftarrow i+1$ ;

}

else ustaw  $j \leftarrow j+1$ ; *(\*  $R(i) = S(j)$ , więc pomijamy jedną ze zduplikowanych krotek \*)*

if  $(i \leq n)$  then dodajemy krotki od  $R(i)$  do  $R(n)$  do pliku  $T$ ;

if  $(j \leq m)$  then dodajemy krotki od  $S(j)$  do  $S(m)$  do pliku  $T$ ;

(d) posortuj krotki w plikach  $R$  i  $S$  używając tych samych unikatowych atrybutów sortowania;

ustaw  $i \leftarrow 1$ ,  $j \leftarrow 1$ ;

while  $(i \leq n)$  and  $(j \leq m)$

do {

if  $R(i) > S(j)$

```

then ustaw  $j \leftarrow j+1$ ;
elseif  $R(i) < S(j)$ 
then ustaw  $i \leftarrow i+1$ ;
else {
    prześlij  $R(i)$  do pliku  $T$ ; (*  $R(i) = S(j)$ , więc przekazujemy krotkę na wyjście *)
    ustaw  $i \leftarrow i+1$ ,  $j \leftarrow j+1$ ;
}
}

```

(e) posortuj krotki w plikach  $R$  i  $S$ , używając tych samych unikatowych atrybutów sortowania;

```

ustaw  $i \leftarrow 1$ ,  $j \leftarrow 1$ ;
while ( $i \leq n$ ) and ( $j \leq m$ )
do {
    if  $R(i) > S(j)$ 
    then ustaw  $j \leftarrow j+1$ ;
    elseif  $R(i) < S(j)$ 
    then {
        prześlij  $R(i)$  do pliku  $T$ ; (*  $R(i)$  nie posiada dopasowania w  $S(j)$ , więc
                                przekazujemy  $R(i)$  na wyjście *)
        ustaw  $i \leftarrow i+1$ ;
    }
    else ustaw  $i \leftarrow i+1$ ,  $j \leftarrow j+1$ ;
}
if ( $i \leq n$ ) then dodajemy krotki od  $R(i)$  do  $R(n)$  do pliku  $T$ ;

```

RYСУNEK 18.3. Implementacja operacji JOIN, PROJECT, UNION, INTERSECTION oraz SET DIFFERENCE przy użyciu metody sortująco-scalającej, gdzie plik  $R$  posiada  $n$  krotek, zaś plik  $S$  posiada  $m$  krotek.

(a) Implementacja operacji  $T \leftarrow R \bowtie_{A=B} S$ . (b) Implementacja operacji  $T \leftarrow \pi_{\langle \text{lista atrybutów} \rangle}(R)$ . (c) Implementacja operacji  $T \leftarrow R \cup S$ . (d) Implementacja operacji  $T \leftarrow R \cap S$ . (e) Implementacja operacji  $T \leftarrow R - S$

## 18.4.2. Wpływ dostępnej przestrzeni bufora i pliku używanego w pętli zewnętrznej na wydajność operacji złączenia w pętli zagnieżdżonej

Dostępna przestrzeń bufora ma istotny wpływ na różne algorytmy złączeniowe. Rozważmy najpierw metodę pętli zagnieżdżonych (J1). Raz jeszcze badając operację OP6 zakładamy, że liczba buforów dostępnych w pamięci głównej dla implementacji złączenia wynosi  $n_B = 7$  bloków (buforów). Przyjmujemy, że każdy bufor w pamięci ma wielkość równą wielkości bloku dysku. Dla zobrazowania sytuacji przyjmijmy, że plik DZIAŁ składa się z  $r_D = 50$  rekordów przechowywanych w  $b_D = 10$  blokach dyskowych oraz że plik PRACOWNIK składa się z  $r_P = 6000$  rekordów przechowywanych w  $b_P = 2000$  bloków dyskowych. Korzystnym rozwiązaniem jest wczytywanie do pamięci naraz jak największej liczby bloków z pliku, zawierających rekordy wykorzystywane w pętli zewnętrznej. Zauważ, że gdy przechowywany jest jeden blok do odczytu z pliku z pętli wewnętrznej i jeden blok do zapisu do pliku wyjściowego, w relacji z pętli zewnętrznej dostępnych jest do odczytu  $n_B - 2$  bloków. Algorytm może następnie wczytywać po jednym bloku z pliku pętli wewnętrznej i używać jego rekordów

w celu **sprawdzania** (czyli przeszukiwania) bloków pętli zewnętrznej w pamięci pod względem występowania pasujących do siebie rekordów. Redukuje to całkowitą liczbę operacji dostępu do bloków. Dodatkowy blok bufora jest potrzebny w celu przechowywania rekordów wynikowych po dokonaniu złączenia i zawartość takiego bloku bufora jest dołączana do **pliku wynikowego** — pliku dyskowego, który zawiera wynik złączenia — w momencie, gdy zostanie zapełniony. Taki blok bufora jest później wykorzystywany ponownie w celu przechowywania dodatkowych rekordów wynikowych.

W przypadku złączenia w pętlach zagnieżdżonych znaczenie ma to, który plik zostanie wybrany w celu obsługi pętli zewnętrznej, a który pętli wewnętrznej. W przypadku, gdy dla obsługi pętli zewnętrznej zostanie wybrany plik `PRACOWNIK`, każdy jego blok zostanie odczytany raz, zaś cały plik `DZIAŁ` (każdy jego blok) będzie odczytywany *za każdym razem*, gdy wczytamy  $(n_B - 2)$  bloków pliku `PRACOWNIK`. Otrzymujemy następujące dane:

Całkowita liczba bloków, do których uzyskuje się dostęp (do odczytu) w przypadku pliku zewnętrznego =  $b_P$

Liczba załadowań  $(n_B - 2)$  bloków pliku zewnętrznego do pamięci głównej =  $\lceil b_P / (n_B - 2) \rceil$

Całkowita liczba bloków, do których uzyskuje się dostęp (do odczytu) w przypadku pliku wewnętrznego =  $b_D * \lceil b_P / (n_B - 2) \rceil$

Stąd otrzymujemy następującą wartość całkowitej liczby operacji dostępu do bloków:

$$b_P + (\lceil b_P / (n_B - 2) \rceil * b_D) = 2000 + (\lceil 2000 / 5 \rceil * 10) = 6000 \text{ operacji dostępu do bloków}$$

Z drugiej strony, jeżeli w pętli zewnętrznej wykorzystamy rekordy pliku `DZIAŁ`, to przez symetrię otrzymamy następującą całkowitą liczbę operacji dostępu do bloków:

$$b_D + (\lceil b_D / (n_B - 2) \rceil * b_P) = 10 + (\lceil 10 / 5 \rceil * 2000) = 4010 \text{ operacji dostępu do bloków}$$

Algorytm złączeniowy wykorzystuje bufor w celu przechowywania złączonych rekordów pliku wynikowego. Po zapełnieniu bufora zostaje on zapisany na dysku i wykorzystany ponownie<sup>12</sup>.

Jeżeli plik wynikowy operacji złączenia posiada  $b_{WYN}$  bloków dyskowych, każdy z nich jest zapisywany raz, więc należy uwzględnić dodatkowe  $b_{WYN}$  operacji dostępu do bloków (w celu zapisu) w przypadku przedstawionych powyżej wzorów w celu oszacowania całkowitego kosztu operacji. Tak samo będzie w przypadku wzorów opracowanych dla innych algorytmów złączeniowych. Jak pokazuje omówiony przykład, w przypadku złączenia w pętlach zagnieżdżonych korzystne jest użycie jako pliku pętli zewnętrznej pliku o *mniej* liczbie bloków.

### 18.4.3. Wpływ współczynnika selekcji złączenia na wydajność tej operacji

Kolejnym czynnikiem wpływającym na wydajność złączenia, szczególnie w przypadku metody pętli pojedynczej J2, jest odsetek rekordów występujących w pliku, które zostaną poddane złączeniu z rekordami z drugiego pliku. Jest to **współczynnik selekcji złączenia**<sup>13</sup>

<sup>12</sup> Jeżeli zarezerwujemy dwa bufory dla pliku wynikowego, można wykorzystać podwójne buforowanie w celu przyspieszenia działania algorytmu (patrz podrozdział 16.3).

<sup>13</sup> Różni się on od *selektywności złączenia*, która zostanie omówiona w rozdziale 19.

(ang. *join selection factor*) pliku w kontekście złączenia równościowego z innymi plikiem. Współczynnik ten zależy od określonego warunku złączenia równościowego między dwoma plikami. W celu zilustrowania problemu weźmy pod uwagę operację OP7, która określa złączenie każdego rekordu pliku DZIAŁ z rekordem pliku PRACOWNIK dla kierownika danego działu. W tym przypadku każdy rekord pliku DZIAŁ (w prezentowanym przykładzie jest ich 50) powinien zostać złączony z *jednym* rekordem z pliku PRACOWNIK i wiele rekordów tego pliku nie zostanie poddane złączeniu (5950 z nich oznacza pracowników nie będących kierownikami działów).

Założmy, że indeks drugorzędny istnieje zarówno na atrybucie PESEL pliku PRACOWNIK, jak i PESELKIEROWNIKA pliku DZIAŁ, gdzie liczba poziomów zagłębienia indeksów wynosi, odpowiednio,  $x_{PESEL} = 4$  oraz  $x_{PESELKIEROWNIKA} = 2$ . Mamy dwie możliwości zaimplementowania metody J2. Pierwsza z nich polega na pobraniu wszystkich rekordów z pliku PRACOWNIK, a następnie użyciu indeksu na atrybucie PESELKIEROWNIKA pliku DZIAŁ w celu znalezienia odpowiadającego rekordu w tym pliku. W takim przypadku nie zostanie znaleziony żaden pasujący rekord dla pracowników, którzy nie są kierownikami działów. Liczba operacji dostępu do bloków w takim przypadku wynosi w przybliżeniu:

$$b_P + (r_P * (x_{PESELKIEROWNIKA} + 1)) = 2000 + (6000 * 3) = 20000 \text{ operacji dostępu do bloków}$$

Drugie rozwiązanie polega na pobraniu każdego rekordu z pliku DZIAŁ i użyciu indeksu na polu PESEL pliku PRACOWNIK w celu znalezienia rekordu odpowiedniego kierownika w tym pliku. W takim przypadku każdy rekord pliku DZIAŁ będzie posiadał odpowiadający mu jeden rekord w pliku PRACOWNIK. Liczba operacji dostępu do bloków w takim przypadku wynosi w przybliżeniu:

$$b_D + (r_D * (x_{PESEL} + 1)) = 10 + (50 * 5) = 260 \text{ operacji dostępu do bloków}$$

Drugie rozwiązanie jest wydajniejsze, gdyż współczynnik selekcji złączenia pliku DZIAŁ *względem warunku złączenia*  $PESEL = PESELKIEROWNIKA$  wynosi 1 (złączany jest każdy rekord z pliku DZIAŁ), natomiast współczynnik selekcji złączenia dla pliku PRACOWNIK *względem tego samego warunku złączenia* wynosi (50/6000), czyli około 0,008 (złączanych jest tylko 0,8% rekordów z pliku PRACOWNIK). W przypadku metody J2 w zewnętrznej pętli złączenia należy użyć mniejszego pliku lub pliku, który posiada dopasowanie dla każdego rekordu (to znaczy pliku o wyższym współczynniku selekcji złączenia). Istnieje również możliwość utworzenia indeksu specjalnie w celu wykonania operacji złączenia, jeżeli taki jeszcze nie istnieje.

Złączenie sortująco-scalające J3 jest dość wydajne, kiedy oba pliki są już posortowane według ich atrybutów złączenia. Każdy z nich wymaga wykonania tylko jednego przebiegu. Stąd liczba operacji dostępu do bloków jest równa sumie liczby bloków w obu plikach. W przypadku tej metody zarówno operacja OP6, jak i OP7 wymaga wykonania  $b_P + b_D = 2000 + 10 = 2010$  operacji dostępu do bloków. Jednak oba pliki muszą być uporządkowane według atrybutów złączenia. Jeżeli nie jest tak w przypadku jednego lub obu z nich, mogą zostać posortowane specjalnie w celu wykonania złączenia. Jeżeli uznamy, że koszt wykonania sortowania pliku zewnętrznego wynosi  $(b \log_2 b)$  operacji dostępu do bloków i jeżeli oba pliki muszą zostać posortowane, to całkowity koszt złączenia sortująco-scalającego można określić jako  $(b_P + b_D + b_P \log_2 b_P + b_D \log_2 b_D)^{14}$ .

<sup>14</sup> Można użyć dokładniejszych wzorów z podrozdziału 19.5, jeżeli znamy liczbę dostępnych buforów dla sortowania.



### 18.4.4. Ogólna postać partycjonowanego złączenia mieszającego

Metoda złączenia mieszającego J4 również jest dość wydajna. W tym przypadku każdy plik wymaga wykonania tylko jednego przebiegu bez względu na to, czy jest posortowany. Jeżeli tabela mieszająca dla mniejszego pliku w całości mieści się w pamięci głównej po wykonaniu mieszania (partycjonowania) na jego atrybucie złączenia, implementacja jest prosta. Jednak jeśli części pliku mieszającego trzeba przechowywać na dysku, metoda nieco się komplikuje i zaproponowano wiele jej odmian w celu zwiększenia wydajności działania. Poniżej zostaną omówione dwie takie techniki o udowodnionej wydajności: *partycjonowane złączenie mieszające* oraz jego odmiana — *hybrydowe złączenie mieszające*.

W przypadku algorytmu **partycjonowanego złączenia mieszającego** (ang. *partition hash join*) każdy plik jest najpierw dzielony na  $M$  partycji przy użyciu **funkcji mieszającej partycjonowania** (ang. *partitioning hash function*) na atrybutach złączenia. Następnie dokonuje się złączenia każdej pary partycji. Przykładowo, załóżmy, że poddajemy złączeniu relacje  $R$  i  $S$  na atrybutach złączenia  $R.A$  oraz  $S.B$ .

$$R \bowtie_{A=B} S$$

W trakcie **fazy podziału** relacja  $R$  jest dzielona na  $M$  partycji  $R_1, R_2, \dots, R_M$ , zaś relacja  $S$  na  $M$  partycji  $S_1, S_2, \dots, S_M$ . Właściwością każdej pary odpowiadających sobie partycji  $R_i$  oraz  $S_i$  jest to, że rekordy z  $R_i$  *muszą zostać poddane złączeniu tylko z rekordami partycji  $S_i$*  i odwrotnie. Właściwość tę zapewnia się przy wykorzystaniu *tej samej funkcji mieszającej* w celu dokonania podziału obu plików względem ich atrybutów złączenia — atrybutu  $A$  dla relacji  $R$  i atrybutu  $B$  dla relacji  $S$ . Minimalna liczba buforów przechowywanych w pamięci potrzebnych w **fazie podziału** wynosi  $M+1$ . Każdy z plików  $R$  i  $S$  jest dzielony niezależnie. W trakcie podziału pliku przydzielanych jest  $M$  buforów w pamięci. Przechowują one rekordy przypisywane po mieszanii do każdej partycji. Jeden dodatkowy bufor jest potrzebny do przechowywania pojedynczych bloków dzielonego pliku wejściowego. Kiedy bufor zostanie zapełniony, jego zawartość jest dołączana do **podpliku dyskowego** (ang. *disk subfile*), przechowującego dane określonej partycji. Faza podziału składa się z *dwóch iteracji*. Po pierwszej z nich pierwszy plik  $R$  zostaje podzielony na podpliki  $R_1, R_2, \dots, R_M$ , gdzie wszystkie rekordy przydzielone do tego samego bufora znajdują się w tej samej partycji. Po drugiej iteracji podobnie zostaje podzielony drugi plik  $S$ .

W drugiej fazie, noszącej nazwę **fazy złączenia lub sprawdzania** (ang. *joining (probing) phase*), potrzebnych jest  $M$  iteracji. W czasie iteracji  $i$  zostają złączone dwie partycje  $R_i$  oraz  $S_i$ . Minimalna liczba buforów potrzebnych dla iteracji  $i$  jest liczbą bloków występujących w mniejszej z dwóch partycji, na przykład  $R_i$ , oraz dwóch dodatkowych buforów. Jeżeli w trakcie iteracji  $i$  wykorzystamy złączenie pętli zagnieżdżonych, rekordy należące do mniejszej partycji  $R_i$  zostaną skopiowane do buforów pamięci. Następnie wszystkie bloki z drugiej partycji  $S_i$  są odczytywane — po jednym naraz — i każdy rekord jest wykorzystywany w celu **sprawdzenia** (czyli przeszukania) partycji  $R_i$  pod względem występowania pasujących rekordów. Wszelkie takie pasujące rekordy są poddawane złączeniu i zapisywane do pliku wynikowego. W celu zwiększenia wydajności procesu sprawdzania w pamięci często wykorzystuje się *tabelę mieszającą umieszczoną w pamięci* w celu przechowywania rekordów z partycji  $R_i$ , używając *innej* funkcji mieszającej niż funkcja partycjonowania<sup>15</sup>.

<sup>15</sup> Jeżeli funkcja mieszająca użyta w celu partycjonowania zostanie wykorzystana ponownie, wszystkie rekordy partycji ponownie znajdą się w tym samym pakiecie.

Koszt takiego partycjonowanego złączenia mieszającego można w omawianym przykładzie oszacować jako  $3 \cdot (b_R + b_S) + b_{WYN}$ , gdyż każdy rekord jest odczytywany raz i zapisywany z powrotem na dysku raz w czasie fazy podziału. W czasie fazy złączania (sprawdzania) każdy rekord jest odczytywany po raz drugi w celu wykonania złączenia. *Główna trudność* tego algorytmu wiąże się z zapewnieniem, aby funkcja mieszająca partycjonowania była **jednorodna** (ang. *uniform*), to znaczy, aby rozmiary partycji były niemal równe. Jeżeli funkcja partycjonowania charakteryzuje się **rozkładem skośnym** (ang. *skewed*), czyli jest niejednorodna, wówczas pewne partycje mogą być zbyt duże, aby pomieścić się w dostępnej przestrzeni pamięciowej dla drugiej fazy złączania.

Należy zauważyć, że jeżeli dostępna przestrzeń bufora w pamięci  $n_B > (b_R + 2)$ , gdzie  $b_R$  jest liczbą bloków dla *mniejszego* z dwóch plików poddawanych złączeniu, na przykład  $R$ , wówczas nie ma powodu przeprowadzania podziału, gdyż w takiej sytuacji złączenie można w całości wykonać w pamięci, używając pewnej odmiany złączenia w pętlach zagnieźdzonych bazując na mieszaniu i sprawdzaniu. Przykładowo, załóżmy, że wykonujemy operację złączeniową OP6, którą powtarzamy poniżej:

(OP6): PRACOWNIK  $\bowtie$  NR0Z = NUMERDZ DZIAŁ

W tym przykładzie mniejszym plikiem jest DZIAŁ, a zatem, jeżeli liczba dostępnych buforów w pamięci  $n_B$  jest większa od  $(b_D + 2)$ , cały plik DZIAŁ można wczytać do pamięci głównej i zorganizować w ramach tabeli mieszającej na atrybucie złączenia. Każdy blok pliku PRACOWNIK jest następnie wczytywany do bufora i każdy jego rekord w takim buforze zostaje poddany działaniu funkcji mieszającej względem swojego atrybutu złączenia i użyty w celu *sprawdzenia* odpowiedniego pakietu w pamięci w tabeli mieszającej pliku DZIAŁ. Jeżeli zostanie znaleziony pasujący rekord, rekordy są poddawane złączeniu i rekord(y) wynikowy jest zapisywany do bufora wynikowego, a w później do pliku wynikowego na dysku. Koszt pod względem operacji dostępu do bloków wynosi zatem  $(b_D + b_P)$  plus  $b_{WYN}$  — czyli koszt zapisania pliku wynikowego.

### 18.4.5. Hybrydowe złączanie mieszające

**Hybrydowy algorytm złączenia mieszającego** stanowi odmianę partycjonowanego złączenia mieszającego, gdzie *faza złączenia dla jednej z partycji* jest uwzględniona w *fazie podziału*. W celu zilustrowania takiego rozwiązania załóżmy, że rozmiar bufora dyskowego wynosi jeden blok dyskowy, że jest *dostępnych*  $n_B$  takich buforów oraz że używana funkcja mieszająca ma postać  $h(K) = K \bmod M$ , tak że tworzonych jest  $M$  partycji, gdzie  $M < n_B$ . Dla zilustrowania problemu przyjmijmy, że wykonujemy operację złączenia OP6. W *pierwszym przebiegu* fazy podziału, kiedy hybrydowy algorytm złączenia mieszającego dzieli mniejszy z dwóch plików (w przypadku operacji OP6 — DZIAŁ), algorytm dzieli przestrzeń bufora między  $M$  partycji, tak aby wszystkie bloki *pierwszej partycji* pliku DZIAŁ w całości znajdowały się w pamięci głównej. W przypadku każdej z pozostałych partycji zostaje przydzielony tylko pojedynczy bufor w pamięci, którego rozmiar wynosi jeden blok. Reszta partycji jest zapisywana na dysku, tak jak w przypadku zwykłego partycjonowanego złączenia mieszającego. Stąd po zakończeniu *pierwszego przebiegu fazy podziału* pierwsza partycja pliku DZIAŁ znajduje się w całości w pamięci głównej, natomiast każda z pozostałych partycji pliku DZIAŁ znajduje się w podpliku na dysku.

W czasie drugiego przebiegu fazy podziału dzielone są rekordy z drugiego, większego pliku (w przypadku operacji OP6 — PRACOWNIK) poddawanego złączeniu. Jeżeli wynik działania funkcji mieszającej na rekordzie powoduje jego wstawienie do *pierwszej partycji*, zostaje on złączony z odpowiednim rekordem z pliku DZIAŁ i złączone rekordy są zapisywane do bufora wynikowego (a później na dysku). Jeżeli wykonanie funkcji mieszającej na rekordzie pliku PRACOWNIK powoduje jego wstawienie do partycji innej niż pierwsza, jest on dzielony normalnie i zapisywany na dysku. Stąd po zakończeniu drugiego przebiegu fazy podziału wiadomo, że wszystkie rekordy wstawione do pierwszej partycji zostały złączone. W tym momencie występuje  $M-1$  par partycji na dysku. Dlatego też w czasie drugiej fazy **złączania (sprawdzania)** zamiast  $M$  wymagane jest  $M-1$  iteracji. Celem jest złączenie w czasie fazy podziału tylu rekordów, aby zmniejszyć koszt ich przechowywania na dysku i ponownego wczytywania w czasie fazy złączania.

## 18.5. Algorytmy operacji projekcji i teoriomnogościowych

Operacja projekcji (PROJECT)  $\pi_{\langle \text{lista atrybutów} \rangle}(R)$  z algebry relacyjnej powoduje, że po projekcji  $R$  na kolumny z listy atrybutów wszystkie powtórzenia są eliminowane. Wynik jest traktowany jak zbiór krotek. Jednak zapytanie SQL:

```
SELECT PENSJA
FROM   PRACOWNIK
```

generuje listę pensji wszystkich pracowników. Jeśli jest 10 000 pracowników i tylko 80 różnych wysokości pensji, tworzona jest jedna kolumna wynikowa z 10 000 krotek. Ta operacja jest wykonywana w wyniku prostego wyszukiwania liniowego za pomocą prostego przebiegu przez tabelę.

Uzyskanie rzeczywistego efektu działania operatora  $\pi_{\langle \text{lista atrybutów} \rangle}(R)$  z algebry relacyjnej jest proste, jeżeli  $\langle \text{lista atrybutów} \rangle$  zawiera klucz relacji  $R$ , ponieważ w takim przypadku wynik operacji ma tę samą liczbę krotek co relacja  $R$ , ale zawiera w każdej krotce tylko wartości atrybutów należących do  $\langle \text{lista atrybutów} \rangle$ . Jeżeli  $\langle \text{lista atrybutów} \rangle$  nie zawiera klucza relacji  $R$ , *trzeba wyeliminować duplikaty krotek*. Dokonuje się tego zwykle przez posortowanie wyniku operacji, a następnie usunięcie duplikatów krotek, które występują wówczas obok siebie. Na rysunku 18.3(b) przedstawiono zarys odpowiedniego algorytmu. W celu wyeliminowania duplikatów krotek można również wykorzystać mieszanie: w miarę jak każdy rekord zostaje poddany działaniu funkcji mieszającej i wstawiony do pakietu pliku mieszającego w pamięci, sprawdza się rekordy już występujące w pakiecie — jeżeli wystąpi duplikat, jest on pomijany. Warto przypomnieć w tym miejscu, że w przypadku zapytań SQL domyślnym zachowaniem jest nieeliminowanie duplikatów z wyniku zapytania. Powtórzenia są eliminowane tylko wtedy, gdy stosowane jest słowo kluczowe DISTINCT.

Operacje teoriomnogościowe — UNION, INTERSECTION, SET DIFFERENCE oraz CARTESIAN PRODUCT — są czasem kosztowne w implementacji, ponieważ UNION, INTERSECTION, MINUS i SET DIFFERENCE to operatory zbiorów i zawsze muszą zwracać wyniki bez duplikatów.

W szczególności, operacja iloczynu kartezjańskiego  $R \times S$  jest dość kosztowna, ponieważ jej wynik zawiera rekord dla każdej kombinacji rekordów z relacji  $R$  i  $S$ . Ponadto każdy rekord wyniku zawiera wszystkie atrybuty relacji  $R$  i  $S$ . Jeżeli relacja  $R$  posiada  $n$  rekordów i  $j$  atrybutów, zaś relacja  $S$  —  $m$  rekordów i  $k$  atrybutów, relacja wynikowa będzie zawierała  $n*m$  rekordów oraz  $j+k$  atrybutów. Stąd istotną rzeczą jest unikanie operacji iloczynu kartezjańskiego i zastępowanie jej równoważnymi operacjami w czasie optymalizacji zapytań. Pozostałe trzy operacje teoriomnogościowe — UNION (suma), INTERSECTION (przecięcie) oraz SET DIFFERENCE (różnica)<sup>16</sup> — mają zastosowanie tylko w przypadku relacji **zgodnych co do typu**, które posiadają taką samą liczbę atrybutów i te same dziedziny atrybutów. Często stosowanym sposobem implementowania tych operacji jest użycie odmian **techniki sortująco-scalającej**: dwie relacje zostają posortowane względem tych samych atrybutów i jednokrotne przejście każdej relacji wystarczy do utworzenia wyniku. Przykładowo, operację UNION w postaci  $R \cup S$  można zrealizować, przeglądając i scalając oba posortowane pliki równolegle i, w razie znalezienia tej samej krotki w obu relacjach, zachowując w scalonym wyniku tylko jedną z nich. W przypadku operacji INTERSECTION  $R \cap S$ , w scalonym wyniku zachowujemy tylko te krotki, które występują w *obu relacjach*. Rysunki od 18.3(c) do 18.3(e) stanowią zarysy implementacji tych operacji poprzez sortowanie i scalanie. Nie uwzględniono w nich pewnych szczegółów działania.

W celu realizacji operacji UNION, INTERSECTION oraz SET DIFFERENCE można również użyć techniki **mieszania**. Jedna tabela jest najpierw przeglądana i dzielona na przechowywaną w pamięci tabelę mieszającą z pakietami, po czym rekordy z drugiej tabeli są przeglądane jeden po drugim i sprawdzane względem odpowiedniej partycji. Przykładowo, w celu zrealizowania operacji  $R \cup S$ , najpierw poddajemy działaniu funkcji mieszającej (dzielimy) rekordy relacji  $R$ , a później poddajemy działaniu funkcji mieszającej (sprawdzamy) rekordy relacji  $S$ , ale nie wstawiamy zduplikowanych rekordów do pakietów. W celu zrealizowania operacji  $R \cap S$  najpierw dzielimy rekordy relacji  $R$ , zapisując je w pliku mieszającym. Następnie, poddając działaniu funkcji mieszającej każdy rekord relacji  $S$ , sprawdzamy, czy w pakiecie można znaleźć identyczny rekord z relacji  $R$ ; jeśli okaże się, że tak, dodajemy go do pliku wynikowego. W celu zrealizowania operacji  $R - S$  najpierw poddajemy działaniu funkcji mieszającej rekordy relacji  $R$ , zapisując je w pakietach pliku mieszającego. Przeglądając każdy rekord relacji  $S$ , sprawdzamy, czy w pakiecie można znaleźć identyczny, i ewentualnie usuwamy go z pakietu.

### 18.5.1. Stosowanie antyżyłczeń w operacji SET DIFFERENCE (EXCEPT lub MINUS w języku SQL)

Operator MINUS języka SQL jest przekształcany w antyżyłczenie (omówione w podrozdziale 18.1) w opisany dalej sposób. Załóżmy, że chcesz ustalić, które działy ze schematu z rysunku 5.5 nie mają pracowników. Zapytanie:

```
SELECT NUMERDZ FROM DZIAŁ MINUS SELECT NRDZ FROM PRACOWNIK;
```

można przekształcić na następującą postać:

<sup>16</sup> W przypadku języka SQL operacja SET DIFFERENCE nosi nazwę EXCEPT.

```
SELECT DISTINCT DZIAŁ.NUMERDZ
FROM   DZIAŁ, PRACOWNIK
WHERE  DZIAŁ.NUMERDZ A= PRACOWNIK.NRDZ
```

Posługujemy się tu niestandardową notacją dla antyzłączeń, A=, przy czym relacja DZIAŁ znajduje się po lewej stronie antyzłączenia, a relacja PRACOWNIK po prawej.

W języku SQL występują dwie odmiany operacji na zbiorach. Operacje UNION, INTERSECTION i EXCEPT lub MINUS (słowa kluczowe języka SQL używane dla operacji SET DIFFERENCE) dotyczą tradycyjnych zbiorów, bez duplikatów w wynikach. Operacje UNION ALL, INTERSECTION ALL i EXCEPT ALL dotyczą wielozbiorów. Dla bazy z rysunku 5.5 rozważmy zapytanie wyszukujące wszystkie działy zatrudniające pracowników i kontrolujące zarządzanie przynajmniej jednym projektem. Można to zapisać tak:

```
SELECT NRDZ FROM PRACOWNIK
INTERSECT ALL
SELECT NR_DZ FROM PROJEKT
```

Wyznaczanie części wspólnej nie eliminuje tu duplikatów wartości NRDZ z relacji PRACOWNIK. Jeśli wszyscy z 10 000 pracowników są przydzieleni do działów, dla których w relacji PROJEKT występuje jakiś projekt, wynikiem będzie lista wszystkich 10 000 numerów działów — w tym z duplikatami. Zadanie można też wykonać za pomocą opisanego w podrozdziale 18.1 złączenia częściowego:

```
SELECT DISTINCT PRACOWNIK.NRDZ
FROM   DZIAŁ, PRACOWNIK
WHERE  PRACOWNIK.NRDZ S= DZIAŁ.NUMERDZ
```

Jeśli operator INTERSECT jest używany z członem ALL, potrzebny jest dodatkowy krok eliminowania duplikatów spośród pobranych numerów działów.

## 18.6. Implementacja operacji agregujących oraz złączeń różnego rodzaju

### 18.6.1. Implementacja operacji agregujących

Operacje agregujące (MIN, MAX, COUNT, AVERAGE, SUM) zastosowane względem całej tabeli można wykonywać poprzez przegląd tabeli lub użycie odpowiedniego indeksu, o ile taki jest dostępny. Rozważmy następujące przykładowe zapytanie SQL:

```
SELECT MAX(PENSJA)
FROM   PRACOWNIK;
```

Jeżeli na atrybucie PENSJA relacji PRACOWNIK istnieje indeks (rosnący) oparty na B+drzewie, optymalizator może zdecydować o jego użyciu w celu wyszukania największej wartości poprzez przechodzenie do *pierwszego od prawej strony* wskaźnika w każdym wierzchołku indeksu od korzenia do skrajnie prawego liścia. Wierzchołek będzie zawierał największą wartość atrybutu PENSJA jako swój *ostatni* wpis. W większości przy-

padków rozwiązanie to jest wydajniejsze od pełnego przeglądu tabeli PRACOWNIK, gdyż nie ma potrzeby faktycznego pobierania żadnych rekordów. Funkcję MIN można obsłużyć w podobny sposób, z tą różnicą, że wybieramy wskaźnik *pierwszy od strony lewej*. Wierzchołek zawiera najmniejszą wartość atrybutu PENSJA jako swój *pierwszy* wpis.

Indeksu można również użyć dla funkcji agregujących AVERAGE i SUM, ale tylko jeśli jest to **indeks zagęszczony**, to znaczy, że występuje w nim wpis dla każdego rekordu z pliku głównego. W takim przypadku odpowiednie obliczenia dotyczą wartości z indeksu. W przypadku **indeksu niezagęszczonego** musi być użyta faktyczna liczba rekordów związanych z każdym wpisem w indeksie w celu zapewniania poprawności obliczeń. Jest to możliwe, jeśli w każdym wpisie indeksu zapisana jest *liczba rekordów powiązanych z każdą wartością* tego indeksu. W funkcji agregującej COUNT można w podobny sposób ustalić liczbę wartości na podstawie indeksu. Funkcja COUNT(\*) może też być stosowana do całej relacji; ponieważ liczba rekordów przechowywanych aktualnie w każdej relacji jest zwykle zapisywana w katalogu, wynik można wtedy pobrać bezpośrednio z katalogu.

Jeśli w zapytaniu użyje się klauzuli GROUP BY, operator agregujący musi być zastosowany oddzielnie względem każdej grupy krotek podzielonych na podstawie atrybutu grupowania. Stąd tabela musi najpierw zostać podzielona na podzbiory krotek, gdzie każda partycja (grupa) posiada tę samą wartość dla atrybutów grupowania. W takim przypadku obliczenia stają się bardziej skomplikowane. Weźmy pod uwagę następujące zapytanie:

```
SELECT   NRDZ, AVG(PENSJA)
FROM     PRACOWNIK
GROUP BY NRDZ;
```

Często stosowana technika w przypadku podobnych zapytań polega na użyciu najpierw albo **sortowania**, albo **mieszania** na atrybutach grupujących w celu podziału pliku na odpowiednie grupy. Następnie algorytm wylicza funkcję agregującą dla krotek w każdej grupie, które posiadają te same wartości atrybutów grupowania. W powyższym przykładowym zapytaniu zbiór krotek relacji PRACOWNIK dla każdego numeru działu zostaje zgrupowany razem w ramach partycji i zostaje wyliczona średnia pensja dla każdej grupy.

Należy zauważyć, że jeżeli na atrybucie (atrybutach) grupowania istnieje **indeks klastrowania** (patrz rozdział 17.), wówczas rekordy są *już podzielone* (pogrupowane) na odpowiednie podzbiory. W takiej sytuacji konieczne jest jedynie wykonanie obliczeń dla każdej grupy.

## 18.6.2. Implementacja różnego rodzaju złączeń

Oprócz standardowych złączeń (nazywanych w języku SQL INNER JOIN) występują też często stosowane inne odmiany złączeń. Opiszemy pokrótce trzy z nich: złączenia zewnętrzne, złączenia częściowe i antyzłączenia.

**Złączenia zewnętrzne.** W podrozdziale 6.4 przedstawiono *operację złączenia zewnętrznego* z jej trzema odmianami: lewostronnego złączenia zewnętrznego, prawostronnego złączenia zewnętrznego oraz pełnego złączenia zewnętrznego. Ponadto w rozdziale 5. omówiono, w jaki sposób operacje te można określać w języku SQL. Poniżej przedstawiono przykład operacji lewostronnego złączenia zewnętrznego w zapisie SQL:



```
SELECT P.NAZWISKO, P.IMIĘ, D.NAZWADZ
FROM (PRACOWNIK P LEFT OUTER JOIN DZIAL D ON P.NRDZ = D.NUMERDZ);
```

Wynikiem tego zapytania jest tabela imion i nazwisk pracowników z nazwami działów, w których pracują. Przypomina ono wynik złączenia zwykłego (wewnętrznego), z tą różnicą, że jeżeli krotka relacji PRACOWNIK (krotka z lewej relacji) *nie posiada przypisanego działu*, imię i nazwisko pracownika i tak pojawia się w tabeli wynikowej, ale wartością nazwy działu w takich krotkach będzie *null*. Złączenie zewnętrzne można traktować jak połączenie złączenia wewnętrznego i antyzłączenia.

Złączenie zewnętrzne można określić, modyfikując jeden z algorytmów złączeniowych, takich jak złączenia w pętłach zagnieżdżonych lub złączenia z pętlą pojedynczą. Przykładowo, w celu określenia *lewostronnego* złączenia zewnętrznego używamy lewej relacji jako pętli zewnętrznej lub pętli pojedynczej, ponieważ każda krotka z tej relacji musi się znaleźć w wyniku. Jeżeli w drugiej relacji znajdują się pasujące krotki, tworzone jest złączenie krotek i są one zapisywane do wyniku. Jednakże jeżeli nie zostanie znaleziona żadna pasująca krotka, ta pierwsza i tak jest uwzględniana w wyniku, ale uzupełnia się ją wartościami *null*. W celu określania złączeń zewnętrznych można również rozszerzyć algorytm złączenia sortująco-scalającego lub mieszejącego.

Teoretycznie złączenie zewnętrzne można też przeprowadzić za pomocą kombinacji operacji algebry relacji. Przykładowo, lewostronne złączenie zewnętrzne przedstawione powyżej jest równoważne następującej sekwencji operacji relacyjnych:

- (1) Określamy złączenie wewnętrzne tabel PRACOWNIK i DZIAŁ.

$$TEMP1 \leftarrow \pi_{NAZWISKO, IMIĘ, NAZWADZ}(PRACOWNIK \bowtie_{NRDZ = NUMERDZ} DZIAŁ)$$

- (2) Znajdujemy w tabeli PRACOWNIK krotki, które nie występują w wyniku złączenia wewnętrznego.

$$TEMP2 \leftarrow \pi_{NAZWISKO, IMIĘ}(PRACOWNIK) - \pi_{NAZWISKO, IMIĘ}(TEMP1)$$

Operację MINUS można przeprowadzić, wykonując antyzłączenie między relacjami PRACOWNIK i TEMP1 na atrybutach NAZWISKO i IMIĘ, co opisano w punkcie 18.5.2.

- (3) Uzupełniamy każdą krotkę w TEMP2 polem NAZWADZ o wartości *null*.

$$TEMP2 \leftarrow TEMP2 \times NULL$$

- (4) Wykonujemy operację UNION na TEMP1 i TEMP2 w celu otrzymania wyniku operacji LEFT OUTER JOIN.

$$WYNIK \leftarrow TEMP1 \cup TEMP2$$

Koszt złączenia zewnętrznego w powyższym przypadku stanowi sumę kosztów odpowiednich etapów działań (złączenia wewnętrznego, projekcji, różnicy i sumy). Jednak warto zauważyć, że krok 3. można wykonać w czasie tworzenia relacji tymczasowej w kroku 2., to znaczy uzupełnić po prostu każdą wynikową krotkę wartością *null*. Ponadto, w etapie 4. wiemy, że dwa operandy sumy są rozłączne (nie posiadają wspólnych krotek), więc nie ma potrzeby przeprowadzania eliminacji duplikatów. Dlatego preferowaną metodą jest połączenie złączenia wewnętrznego i antyzłączenia zamiast wykonywania przedstawionych kroków, ponieważ algebraiczna projekcja, po której następuje obliczanie różnicy, skutkuje wielokrotnym zapisywaniem i przetwarzaniem tabel tymczasowych.



Prawostronne złączenie zewnętrzne można przekształcić w lewostronne, przedstawiając operandy. Dlatego złączenie lewostronne nie wymaga odrębnego omawiania. **Pełne złączenie zewnętrzne** wymaga obliczenia wyniku złączenia wewnętrznego i uzupełnienia wyniku dodatkowymi krotkami powstałymi na bazie niedopasowanych krotek z relacji używanych jako lewy i prawy operand. Pełne złączenie zewnętrzne jest zwykle obliczane za pomocą rozszerzonych algorytmów złączania sortująco-scalającego lub złączania mieszającego, uwzględniających niedopasowane krotki.

**Implementowanie złączeń częściowych i antyzłączeń.** W podrozdziale 18.1 opisaliśmy te rodzaje złączeń jako operacje, na które odwzorowywane są niektóre zapytania z podzapytaniami zagnieżdżonymi. Celem jest możliwość wykonania pewnej odmiany złączenia zamiast wielokrotnego przetwarzania podzapytań. W takich sytuacjach stosowanie złączenia wewnętrznego jest błędem, ponieważ dla każdej krotki z relacji zewnętrznej w złączeniu wewnętrznym wyszukiwane są wszystkie możliwe dopasowania do relacji wewnętrznej. W złączeniu częściowym wyszukiwanie kończy się zaraz po znalezieniu pierwszego dopasowania i pobraniu krotki z relacji zewnętrznej. W antyzłączeniu koniec wyszukiwania następuje po znalezieniu pierwszego dopasowania i odrzuceniu krotki z relacji zewnętrznej. Oba te rodzaje złączeń można zaimplementować jako rozszerzenie algorytmów złączania opisanych w podrozdziale 18.4.

**Implementowanie złączeń nierównościowych.** Operację złączania można też wykonać, gdy warunkiem złączania jest nierówność. W rozdziale 6. nazwaliśmy tę operację złączeniem theta. Ten mechanizm działa dla warunków obejmujących dowolne operatory, np.:  $<$ ,  $>$ ,  $\geq$ ,  $\leq$ ,  $\neq$ . Także wtedy można stosować prawie wszystkie z opisanych metod złączania; wyjątkiem są algorytmy oparte na mieszaniu.

## 18.7. Łączenie operacji poprzez mechanizm potokowy

Zapytanie określone w języku SQL zwykle jest przekształcane na wyrażenie algebry relacji, które jest *sekwencją operacji relacyjnych*. Jeżeli będziemy wykonywać po jednej operacji naraz, musimy generować pliki tymczasowe na dysku w celu przechowywania wyników takich działań tymczasowych, co wnosi spory narzut. Przetwarzanie zapytania za pomocą generowania i zapisywania każdego tymczasowego wyniku oraz przekazywania go jako argumentu do następnego operatora to **przetwarzanie z materializacją**. Każdy tymczasowy zmaterializowany wynik jest zapisywany na dysku i zwiększa ogólny koszt przetwarzania zapytania.

Generowanie i przechowywanie dużych plików tymczasowych na dysku jest czasochłonne i w wielu przypadkach może być niepotrzebne, gdyż pliki te są natychmiast używane jako dane wejściowe dla kolejnej operacji. W celu zredukowania liczby plików tymczasowych często generuje się kod wykonania zapytania, który odpowiada algorytmom łączenia operacji w zapytaniu.

Przykładowo, złączenie może być kombinacją dwóch operacji SELECT na plikach wejściowych i końcowej operacji PROJECT na pliku wynikowym zamiast wykonywania wszystkich operacji złączenia oddzielnie. Wszystkie operacje można zaimplementować w ramach jednego algorytmu z dwoma plikami wejściowymi i jednym plikiem wyjściowym.

Zamiast tworzyć cztery pliki tymczasowe, wykorzystujemy algorytm bezpośrednio i otrzymujemy tylko jeden plik wynikowy.

W podrozdziale 19.1 omówiono, w jaki sposób heurystyczna optymalizacja algebry relacji może posłużyć do zgrupowania wykonywanych operacji. Łączenie kilku operacji w jedną i unikanie zapisu tymczasowych wyników na dysku to tak zwane przetwarzanie **potokowe** (ang. *pipelining*) lub **strumieniowe** (ang. *stream-based processing*).

Często kod wykonania zapytania tworzy się dynamicznie w celu zaimplementowania wielu operacji. Wygenerowany kod tworzenia zapytania łączy kilka algorytmów, które odpowiadają poszczególnym operacjom. W rezultacie tworzone są krotki pochodzące z jednej operacji, które wykorzystujemy jako dane wejściowe dla dalszych operacji. Przykładowo, jeżeli operacja złączenia występuje po dwóch operacjach selekcji na relacji bazowej, krotki stanowiące wynik każdej operacji selekcji są przekazywane jako dane wejściowe dla algorytmu złączeniowego w ramach **strumienia** lub **potoku**. To podejście jest nazywane **przetwarzaniem potokowym**. Ma ono dwie charakterystyczne zalety:

- Pozwala uniknąć dodatkowych kosztów i opóźnień powodowanych zapisem tymczasowych wyników na dysku.
- Możliwość szybkiego rozpoczęcia generowania wyników, gdy operator podstawowy jest łączony z wybranymi operatorami opisanymi w następnym punkcie, oznaczając, że przetwarzanie potokowe pozwala rozpocząć generowanie krotek wynikowych w trakcie operowania na pozostałych tabelach pośrednich potoku.

### 18.7.1. Iteratory używane do implementowania operacji fizycznych

Różne algorytmy wykonujące operacje algebraiczne wczytują dane wejściowe w postaci jednego lub kilku plików, przetwarzają te dane i generują plik wyjściowy w postaci relacji. Jeśli operacja jest zaimplementowana w taki sposób, że zwraca po jednej krotce, można ją traktować jak **iterator**. Przykładowo, możesz zaprojektować opartą na krotkach implementację złączenia w pętlach zagnieżdżonych, która generuje dane wyjściowe krotka po krotce. Iteratory działają inaczej niż materializacja, gdzie całe relacje są tworzone jako wyniki tymczasowe i zapisywane na dysku lub w pamięci głównej, a następnie ponownie wczytywane przez następny algorytm. Plan zapytania obejmujący drzewo zapytania można wykonać, wywołując iteratory w określonej kolejności. W danym momencie aktywnych może być wiele iteratorów, co powoduje przekazywanie wyników w górę drzewa wykonania i pozwala uniknąć dodatkowego składowania tymczasowych wyników. Interfejs iteratora zwykle obejmuje następujące metody:

- (1) `Open()`. Ta metoda inicjuje operator, przydzielając bufor na jego dane wejściowe i wyjściowe. Inicjuje też struktury danych potrzebne danemu operatorowi. Ponadto służy do przekazywania argumentów takich jak warunki selekcji potrzebne do wykonania operacji. Metoda `Open()` jest wywoływana w celu pobrania potrzebnych argumentów.
- (2) `Get_Next()`. Ta metoda wywołuje metodę `Get_next()` każdego z argumentów wejściowych i uruchamia kod specyficzny dla operacji wykonywanej na danych wejściowych. Zwracana jest następna wygenerowana krotka wyjściowa, a stan

iteratora jest aktualizowany, aby śledzić ilość przetworzonych danych wejściowych. Gdy nie ma już dalszych krotek do zwrócenia, w buforze wyjściowym umieszczana jest specjalna wartość.

- (3) `Close()`. Ta metoda kończy iterowanie po wygenerowaniu wszystkich możliwych krotek albo po zwróceniu wymaganej (lub żądanej) liczby krotek. Ponadto wywołuje metodę `Close()` argumentów przekazanych do iteratora.

Każdy iterator można traktować jak klasę z implementacją trzech wymienionych metod, które można stosować do każdej instancji danej klasy. Jeśli implementowany operator pozwala na całkowite przetworzenie krotki po jej otrzymaniu, można w wydajny sposób zastosować strategię potokową. Jeżeli jednak krotki wejściowe wymagają sprawdzania w wielu przebiegach, dane wejściowe trzeba pobierać w formie zmaterializowanej relacji. Wtedy większość pracy wykonuje metoda `Open()` i nie można osiągnąć pełnych korzyści przetwarzania potokowego. Niektóre operatory fizyczne nie są zgodne z interfejsem iteratora, dlatego mogą nie obsługiwać przetwarzania potokowego.

Iteratory można też stosować do metod dostępowych. Dostęp do B<sup>+</sup>-drzewa lub indeksu opartego na mieszanii można potraktować jak działanie funkcji implementowanej jako iterator. Taka funkcja generuje sekwencję krotek zgodnych z warunkiem selekcji przekazanym do metody `Open()`.

## 18.8. Algorytmy równoległego przetwarzania zapytań

W rozdziale 2. wspomnieliśmy o kilku wersjach architektur klient-serwer, w tym o architekturach dwu- i trzywarstwowych. Istnieje też inna wersja, **architektura równoległych baz danych**, popularna w aplikacjach intensywnie przetwarzających dane. Omówimy ją szczegółowo w rozdziale 23. razem z rozproszonymi bazami danych, big data i nowymi technologiami NOSQL.

Na potrzeby równoległych baz danych zaproponowano trzy podstawowe podejścia. Są one powiązane z trzema różnymi konfiguracjami sprzętowymi procesorów i drugorzędnych mechanizmów składowania (dysków) wspomagającymi równoległość. W architekturze z **pamięcią współdzieloną** wiele procesorów jest połączonych z siecią i ma dostęp do obszaru wspólnej pamięci głównej. Każdy procesor ma dostęp do całej przestrzeni adresowej z wszystkich maszyn. Dostęp do lokalnej pamięci głównej i podręcznej jest szybszy. Dostęp do wspólnej pamięci jest wolniejszy. Ta architektura jest narażona na zakłócenia, ponieważ wraz z dodawaniem procesorów zwiększa się rywalizacja o wspólną pamięć. Architektura drugiego rodzaju to **architektura z dyskami współdzielonymi**. W tym podejściu każdy procesor ma własną pamięć niedostępną dla innych procesorów. Jednak każda maszyna ma dostęp do wszystkich dysków za pomocą sieci (przy czym nie każdy procesor ma własny dysk). W podrozdziale 16.11 omówiliśmy dwa rodzaje drugorzędnych mechanizmów składowania stosowanych w korporacjach. Zarówno sieci SAN, jak i technologia NAS wykorzystują architekturę z dyskami współdzielonymi i umożliwiają przetwarzanie równoległe. Jednak jednostki transferu danych są w nich inne. W sieciach SAN dane są przesyłane między dyskami i procesorami w formie bloków lub stron. Technologia NAS działa jak serwer plików przesyłający dane za pomocą protokołu transferu plików. W takich systemach dodawanie procesorów skutkuje zwiększeniem rywalizacji o ograniczoną przepustowość sieci.

Opisane trudności doprowadziły do tego, że **architektura bez zasobów współdzielonych** stała się najczęściej wykorzystywaną architekturą w systemach równoległych baz danych. W tej architekturze każdy procesor używa własnej pamięci głównej i dyskowej. Gdy procesor A żąda danych zlokalizowanych na dysku  $D_B$  podłączonym do procesora B, procesor A przesyła w sieci żądanie jako komunikat do procesora B, który uzyskuje dostęp do własnego dysku  $D_B$  i przesyła dane siecią w komunikacie do procesora A. Równoległe bazy danych z architekturą bez zasobów współdzielonych są stosunkowo tanie w budowie. Obecnie standardowe procesory są łączone w ten sposób w szafach serwerowych, a kilka szaf może być powiązanych siecią zewnętrzną. Każdy procesor ma własną pamięć główną i pamięć dyskową.

Architektura bez zasobów współdzielonych umożliwia osiągnięcie równoległości w przetwarzaniu zapytań na trzech opisanych dalej poziomach: poszczególnych operatorów, pojedynczych zapytań i wielu zapytań. Z badań wynika, że dzięki przydziałowi większej liczby procesorów i dysków możliwe jest **liniowe przyspieszenie** przetwarzania, czyli liniowe skrócenie czasu wykonywania operacji. **Liniowe skalowanie** polega na zapewnianiu stałej wydajności dzięki zwiększaniu liczby procesorów i dysków proporcjonalnie do wielkości danych. Oba te zjawiska są pośrednimi celami przetwarzania równoległego.

### 18.8.1. Równoległość na poziomie operatorów

W operacjach, które można zaimplementować za pomocą algorytmów równoległych, jedną z głównych strategii jest podział danych między dyski. **Podział poziomy** relacji polega na podziale krotek między dyski na podstawie wybranej metody partycjonowania. Gdy używanych jest  $n$  dysków, przypisywanie  $i$ -tej krotki do dysku  $i \bmod n$  to **partycjonowanie cykliczne**. W **partycjonowaniu zakresowym** krotki są (w miarę możliwości) rozdzielane równomiernie w wyniku podziału zakresu wartości wybranego atrybutu. Przykładowo, krotki z relacji PRACOWNIK mogą zostać przypisane do 10 dysków dzięki podziałowi wieku na 10 przedziałów: 22 – 25, 26 – 28, 29 – 30 itd., tak aby każdy przedział obejmował mniej więcej 10% łącznej liczby pracowników. Partycjonowanie zakresowe to trudna operacja, wymagająca dobrego zrozumienia rozkładu danych według atrybutu używanego w klauzuli zakresowej. Zakresy używane do partycjonowania są reprezentowane za pomocą **wektora zakresów**. W **mieszaniu partycjonowanym** krotka  $i$  jest przypisywana do dysku  $h(i)$ , gdzie  $h$  to funkcja haszująca. Dalej pokrótce opiszemy, jak projektowane są algorytmy równoległe dla poszczególnych operacji.

**Sortowanie.** Jeśli do danych zastosowano partycjonowanie zakresowe (np. na podstawie wieku) na  $n$  dysków dla  $n$  procesorów, to w celu posortowania całej relacji według wieku można równoległe posortować osobno każdą partycję, a następnie połączyć wyniki. Może to skutkować  $n$ -krotnym skróceniem czasu sortowania. Jeśli relacja została podzielona na partycje w inny sposób, możliwe są następujące rozwiązania:

- Ponowny podział relacji na partycje za pomocą partycjonowania zakresowego według atrybutu, który ma być użyty do sortowania, i późniejsze sortowanie każdej partycji z osobna oraz złączenie wyników w opisany wcześniej sposób.
- Zastosowanie równoległej wersji zewnętrznego algorytmu sortująco-skalającego (patrz rysunek 18.2).

**Selekcja.** Jeśli w selekcji na podstawie jakiegoś warunku używana jest równość  $\langle A = v \rangle$ , a ten sam atrybut  $A$  zastosowano do partycjonowania zakresowego, selekcję można przeprowadzić tylko z użyciem partycji obejmującej  $v$ . W innych sytuacjach selekcja jest wykonywana równolegle we wszystkich procesorach, a wyniki są scalane. Jeśli warunek selekcji to  $v_1 \leq A \leq v_2$ , a do partycjonowania zakresowego używany jest atrybut  $A$ , zakres wartości  $(v_1, v_2)$  musi obejmować określoną liczbę partycji. Operację selekcji trzeba wtedy równolegle wykonać tylko w odpowiednich procesorach.

**Projekcja i eliminowanie duplikatów.** Projekcję bez eliminowania duplikatów można uzyskać, wykonując tę operację równolegle wraz z wczytywaniem danych z każdej partycji. Aby usunąć duplikaty, można posortować krotki i wyeliminować duplikaty. Do sortowania można wykorzystać dowolną z wymienionych technik (w zależności od sposobu partycjonowania danych).

**Złączanie.** Podstawowa zasada równoległego złączania polega na podziale złączanych relacji (np.  $R$  i  $S$ ) w taki sposób, że złączenie jest dzielone na  $n$  mniejszych, wykonywanych równolegle w  $n$  procesorach. Wyniki są następnie sumowane. Dalej opiszemy różne związane z tym wyniki:

a. **Równościowe złączanie partycjonowane.** Jeśli relacje  $R$  i  $S$  są podzielone na  $n$  partycji z  $n$  procesorów w taki sposób, że partycje  $r_i$  i  $s_i$  są przydzielone do tego samego procesora  $P_i$ , złączenie można obliczyć lokalnie — pod warunkiem, że jest to złączenie równościowe lub naturalne. Warto zauważyć, że jeden klucz złączania nie może występować w różnych partycjach (konieczne jest więc partycjonowanie w sensie znanym z teorii zbiorów). Ponadto atrybut używany w warunku złączenia musi spełniać jeden z następujących warunków:

- Jest też używany do partycjonowania zakresowego, a zakresy dla poszczególnych partycji są takie same dla  $R$  i  $S$ .
- Jest też używany do podziału na  $n$  partycji za pomocą mieszania partycjonowanego. Dla  $R$  i  $S$  używana musi być ta sama funkcja mieszająca. Jeśli rozkład wartości atrybutu złączania w  $R$  i  $S$  jest różny, trudno utworzyć wektor zakresów pozwalający na jednorodny podział  $R$  i  $S$  na takie równe partycje. W idealnych warunkach wielkość  $|r_i| + |s_i|$  powinna być identyczna dla wszystkich partycji  $i$ . W przeciwnym razie, gdy dane są zanadto nierównomiernie rozłożone, nie da się osiągnąć pełnych korzyści z przetwarzania równoległego. Złączanie lokalne w każdym procesorze można wykonać za pomocą dowolnej z opisanych technik: sortowanie przez scalanie, pętle zagnieżdżone i mieszanie.

b. **Złączanie nierównościowe z partycjonowaniem i replikacją.** Jeśli warunkiem złączania jest nierówność ( $<$ ,  $\leq$ ,  $>$ ,  $\geq$ ,  $\neq$  itd.), nie da się podzielić  $R$  i  $S$  na partycje w taki sposób, by  $i$ -ta partycja  $R$  ( $r_i$ ) była złączana tylko z  $j$ -tą partycją  $S$  ( $s_j$ ). Takie złączenie można wykonać równolegle na dwa sposoby:

- *Przypadek asymetryczny.* Podział relacji  $R$  za pomocą jednego ze schematów partycjonowania, replikacja jednej z relacji (np.  $S$ ) na potrzeby wszystkich  $n$  partycji i złączanie  $r_i$  z całą  $S$  w procesorze  $P_i$ . Ta metoda jest zalecana, gdy  $S$  jest znacznie mniejsze od  $R$ .
- *Przypadek symetryczny.* W tej ogólnej metodzie, dostosowanej do złączeń dowolnego rodzaju,  $R$  i  $S$  są podzielone na partycje.  $R$  jest podzielona na  $n$  części, a  $S$  — na  $m$  porcji. Do złączania równoległego używanych jest  $m * n$

procesorów. Partycje są w odpowiedni sposób replikowane, tak by procesory od  $P_{0,0}$  do  $P_{n-1,m-1}$  (w sumie  $m * n$  procesorów) mogły lokalnie przeprowadzać złączenie. Procesor  $P_{i,j}$  łączy  $r_i$  z  $s_j$  za pomocą dowolnej techniki złączania. System replikuje partycję  $r_i$  do procesorów  $P_{i,0}, P_{i,1}, \dots, P_{i,m-1}$ . Podobnie partycja  $s_j$  jest replikowana do procesorów  $P_{0,j}, P_{1,j}, \dots, P_{m-1,j}$ . Ogólnie partycjonowanie z replikacją powoduje wyższe koszty niż samo partycjonowanie. Dlatego przy złączaniu równościowym partycjonowanie z replikacją jest bardziej kosztowne.

**c. Równoległe partycjonowane złączanie mieszające.** Partycjonowane złączanie mieszające w wersji z algorytmu J4 z podrozdziału 18.4 można wykonywać równolegle. Pomysł polega na tym, że jeśli  $R$  i  $S$  to duże relacje, to nawet po podziale każdej z nich na  $n$  partycji (gdzie  $n$  to liczba procesorów) złączenie lokalne w każdym procesorze i tak może być kosztowne. Złączenie przebiega wtedy w opisany dalej sposób (zakładamy, że  $s$  jest mniejsze niż  $r$ ):

1. Za pomocą funkcji mieszającej  $h1$  na atrybucie złączania odwzorowujemy każdą krotkę relacji  $r$  i  $s$  na jeden z  $n$  procesorów. Niech  $r_i$  i  $s_i$  będą partycjami przydzielonymi za pomocą mieszania do  $P_i$ . Najpierw należy wczytać krotki relacji  $s$  z każdego procesora na jego dysk lokalny, a następnie odwzorować je na odpowiedni procesor za pomocą funkcji  $h1$ .
2. W każdym procesorze  $P_i$  krotki z  $S$  otrzymane w kroku 1. są dzielone na partycje za pomocą innej funkcji mieszającej,  $h2$ , na  $k$  pakietów. Ten krok jest identyczny z etapem partycjonowania z algorytmu mieszania partycjonowanego J4 z podrozdziału 18.4.
3. Z dysku lokalnego każdego procesora wczytywanych jest  $r$  krotek, po czym są one odwzorowywane na odpowiednie procesory za pomocą funkcji mieszającej  $h1$ . Gdy krotki są otrzymywane w poszczególnych procesorach, procesor dzieli je na  $k$  pakietów za pomocą funkcji mieszającej  $h2$  z kroku 2. Ten proces odpowiada etapowi sprawdzania z algorytmu J4.
4. Procesor  $P_i$  uruchamia algorytm mieszania partycjonowanego lokalnie dla partycji  $r_i$  i  $s_i$ , wykonując etap złączania dla  $k$  pakietów (zgodnie z opisem algorytmu J4), i generuje wynik złączenia.

Wyniki z wszystkich procesorów  $P_i$  są obliczane niezależnie od siebie, po czym wyznaczana jest suma w celu uzyskania ostatecznego wyniku.

**Agregacja.** Operacje agregacji z grupowaniem są wykonywane po podziale danych na partycje na podstawie atrybutu grupowania, po czym w każdym procesorze lokalnie uruchamiana jest funkcja agregująca oparta na dowolnych algorytmach przeznaczonych dla jednego procesora. Można wykorzystać partycjonowanie zakresowe lub mieszanie partycjonowane.

**Operacje na zbiorach.** Jeśli będące argumentami relacje  $R$  i  $S$  są podzielone na partycje za pomocą tej samej funkcji mieszającej, sumę, część wspólną i różnicę zbiorów można obliczać w każdym procesorze równoległe. Jeżeli partycjonowanie relacji wykonano za pomocą różnych metod, konieczny może być ponowny podział  $R$  i  $S$  za pomocą identycznej funkcji mieszającej.



## 18.8.2. Równoległość w jednym zapytaniu

Opisaliśmy, jak wykonywać poszczególne operacje za pomocą podziału danych między wiele procesorów i uruchamiania operacji równolegle w tych procesorach. Plan wykonywania zapytania można przedstawić w formie grafu operacji. Jednym ze sposobów na równoległe wykonywanie zapytania jest zastosowanie równoległego algorytmu dla każdej operacji z zapytania; dane wejściowe poszczególnych operacji są wtedy odpowiednio dzielone na partycje. Inna możliwość równoległego wykonywania zadań związana jest z przetwarzaniem drzewa operatorów, gdzie niektóre operacje można wykonywać równolegle, ponieważ nie są od siebie zależne. Takie operacje można wykonywać w różnych procesorach. Jeśli dane wyjściowe z jednej operacji można generować krotka po krotce i przekazywać do następnego operatora, uzyskujemy **równoległość potokową**. Operator, który nie generuje danych wyjściowych do czasu pobrania wszystkich danych wejściowych, **blokuje potok**.

## 18.8.3. Równoległość w wielu zapytaniach

Równoległość w wielu zapytaniach dotyczy jednoczesnego przetwarzania wielu zapytań. W architekturach bez zasobów współdzielonych lub ze współdzielonym dyskiem trudno jest ją osiągnąć. Trzeba wtedy koordynować blokady, rejestrowanie zdarzeń i inne mechanizmy działające w procesorach (patrz rozdziały z części 9. poświęcone przetwarzaniu transakcji). Należy też unikać sprzecznych aktualizacji tych samych danych przez wiele procesorów. Konieczna jest **spójność pamięci podręcznej**, gwarantująca, że procesor aktualizujący stronę zawiera w buforze jej najnowszą wersję. Niezbędne jest też współdziałanie protokołów zapewniania spójności pamięci podręcznej i kontroli równoległości (patrz rozdział 21.).

Głównym celem równoległości na poziomie wielu zapytań jest skalowanie (zwiększenie ogólnej szybkości przetwarzania zapytań lub transakcji dzięki zwiększeniu liczby procesorów). Ponieważ jednoprocessorowe systemy wielodostępne są projektowane w celu kontroli równoległego przetwarzania transakcji i zwiększenia ich przepustowości (patrz rozdział 21.), systemy baz danych używające równoległej architektury z pamięcią współdzieloną pozwalają łatwiej osiągnąć równoległość bez wprowadzania istotnych zmian.

Z tego omówienia wynika, że można przyspieszyć przetwarzanie zapytań dzięki wykonywaniu różnych operacji (takich jak sortowanie, selekcja, projekcja, złączanie i agregacja) w równoległy sposób. Dodatkowe przyspieszenie można uzyskać, wykonując niezależne fragmenty drzewa zapytań równolegle w różnych procesorach. Trudno jest jednak osiągnąć równoległość w jednym zapytaniu w równoległych architekturach bez zasobów współdzielonych. Jednym z obszarów, w których przewagę ma architektura z dyskiem współdzielonym, jest jej większy zakres zastosowań, ponieważ (w odróżnieniu od architektury bez zasobów współdzielonych) nie wymaga ona przechowywania danych podzielonych na partycje. Obecnie systemy SAN i NAS cechują się tą zaletą. Na ogólną szybkość wpływ ma wiele parametrów, takich jak liczba procesorów i pojemność buforów. Szczegółowe omawianie tych parametrów wykracza poza zakres rozdziału.



## 18.9. Podsumowanie

W niniejszym rozdziale przedstawiono omówienie technik używanych przez SZBD do przetwarzania wysokopoziomowych zapytań. Najpierw omówiono, w jaki sposób zapytania SQL są tłumaczone na wyrażenia algebry relacji. Przedstawiono działanie złączeń częściowych i antyzłączeń, na które odwzorowywane są określone zapytania zagnieżdżone w celu uniknięcia standardowych złączeń wewnętrznych. Omówiono sortowanie zewnętrzne, często potrzebne w trakcie przetwarzania zapytań w celu uporządkowania krotek relacji w ramach agregacji, eliminowania duplikatów itd. Uwzględniono różne rodzaje selekcji i opisano algorytmy selekcji prostej, opartej na jednym atrybucie, i selekcji złożonej, z klauzulami koniunkcyjnymi i dysjunkcyjnymi. Przedstawiono wiele technik dotyczących różnych rodzajów selekcji, w tym wyszukiwanie liniowe i binarne, używanie indeksu opartego na B+drzewach, indeksy bitmapowe, indeks klastrowania i indeks funkcyjny. Opisano też selektywność warunków i typowe informacje umieszczane w katalogu SZBD. Następnie omówiono szczegółowo złączanie i zaproponowano algorytmy złączania w pętlach zagnieżdżonych, złączania w pętlach zagnieżdżonych z użyciem indeksu, złączania sortująco-scalającego i złączania mieszającego.

Pokazano, jak pojemność bufora, współczynnik selekcji złączenia oraz wybór relacji wewnętrznej i zewnętrznej wpływają na wydajność algorytmów złączania. Opisano też hybrydowy algorytm mieszający, który pozwala uniknąć części kosztów zapisu w fazie złączania. Omówiono algorytmy projekcji i operacji na zbiorach, a także algorytmy agregacji. Dalej objaśniono algorytmy różnych typów złączeń, w tym złączeń zewnętrznych, częściowych, antyzłączeń i złączeń nierównośćowych. Opisano też, jak łączyć operacje w trakcie przetwarzania zapytań, aby wykonywać je potokowo lub strumieniowo (zamiast stosowania materializacji). Pokazano, jak implementować operatory za pomocą iteratorów. W końcowej części omówiono strategię przetwarzania zapytań i pokrótce wprowadzono trzy rodzaje architektur równoległych systemów baz danych. Dalej krótko opisano równoległość na poziomie pojedynczych operacji, w pojedynczych zapytaniach i na poziomie wielu zapytań.

## Pytania powtórkowe

- 18.1. Omów powody, dla których przed przeprowadzeniem optymalizacji dokonuje się konwersji zapytań SQL do postaci zapytań algebry relacyjnej.
- 18.2. Omów złączenia częściowe i antyzłączenia jako operacje, na które można odwzorować zapytania zagnieżdżone. Przedstaw przykładowe złączenia obu tych rodzajów.
- 18.3. Jak sortowane są duże tabele, które nie mieszczą się w pamięci? Opisz ogólną procedurę.
- 18.4. Omów różne algorytmy implementacji każdego z poniższych operatorów relacyjnych oraz warunki, w których każdy z nich może zostać użyty: SELECT, JOIN, PROJECT, UNION, INTERSECT, SET DIFFERENCE, CARTESIAN PRODUCT.
- 18.5. Podaj przykłady zapytań z selekcją koniunktywną i dysjunktywną. Wyjaśnij możliwość występowania wielu sposobów wykonywania takich zapytań.

- 18.6. Omów różne sposoby eliminowania duplikatów w trakcie przetwarzania zapytania `SELECT DISTINCT <atrybut>`.
- 18.7. Jak implementowane są operacje agregujące?
- 18.8. Jak implementowane są złączenia zewnętrzne i nierównościami?
- 18.9. Czym jest iterator? Jakie metody są częścią iteratora?
- 18.10. Jakie trzy rodzaje architektur równoległych występują w systemach baz danych? Która z nich jest stosowana najczęściej?
- 18.11. Opisz równoległe implementacje złączeń.
- 18.12. Czym jest równoległość na poziomie pojedynczych zapytań i wielu zapytań? Którą z nich trudniej jest uzyskać w architekturze bez zasobów współdzielonych? Dlaczego?
- 18.13. W jakich warunkach niemożliwe jest równoległe potokowe wykonywanie sekwencji operacji?

## Ćwiczenia

- 18.14. Rozważ zapytania języka SQL Z1, Z8, Z1B, Z4 oraz Z27 z rozdziału 6. i Z27 z rozdziału 7.
  - a) Narysuj co najmniej dwa drzewa zapytań, które mogą reprezentować *każde* z tych zapytań. W jakim przypadku należałoby użyć każdego z tych drzew?
  - b) Narysuj początkowe drzewo zapytania dla każdego z tych zapytań, a następnie pokaż, w jaki sposób drzewo zapytań jest optymalizowane przez algorytm przedstawiony w podrozdziale 18.7.
  - c) W przypadku każdego zapytania porównaj własne drzewo zapytania z punktu a. z początkowym i końcowym drzewem zapytania z punktu b.
- 18.15. Plik liczący 4096 bloków ma być posortowany przy dostępnej przestrzeni buforów liczącej 64 bloki. Ile przebiegów będzie potrzebne w trakcie fazy scalania zewnętrznego algorytmu sortowania przez scalanie?
- 18.16. Czy indeks niezagęszczony może zostać użyty w implementacji operatora agregacji? Dlaczego tak lub nie? Przedstaw przykład.
- 18.17. Rozszerz algorytm złączenia sortująco-scalającego, tak aby zaimplementować operację lewostronnego złączenia zewnętrznego.

## Wybrane publikacje

Odwołania do literatury z zakresu przetwarzania i optymalizacji zapytań przedstawimy w końcowej części rozdziału 19. Dlatego wybrane publikacje z rozdziału 19. dotyczą tego i następnego rozdziału. Trudno jest oddzielić literaturę dotyczącą strategii i algorytmów przetwarzania zapytań od literatury omawiającej optymalizację.

## Optymalizacja zapytań

W tym rozdziale<sup>1</sup> zakładamy, że Czytelnik zna już opisane w poprzednim rozdziale strategie przetwarzania zapytań w relacyjnych SZBD. Celem optymalizacji zapytań jest wybór najlepszej strategii ich wykonywania. Jak już wspomniano, określenie *optymalizacja* jest mylące, ponieważ wybrany plan wykonania nie zawsze jest optymalny. Podstawowym celem jest uzyskanie najwydajniejszego i ekonomicznego planu z wykorzystaniem dostępnych informacji o schemacie i zawartości używanych relacji. Plan ten należy uzyskać w rozsądnym czasie. Zatem odpowiedni opis **optymalizacji zapytań** brzmi tak: aktywność przeprowadzana przez optymalizator zapytań w SZBD w celu wyboru najlepszej dostępnej strategii wykonywania zapytania.

Oto struktura tego rozdziału: w podrozdziale 19.1 opisana jest notacja odwzorowywania zapytań z języka SQL na drzewa i grafy zapytań. W większości relacyjnych SZBD wewnętrzną reprezentacją zapytania jest drzewo. Przedstawimy tu heurystyki przekształcania zapytań w wydajniejszą równoważną postać. Opiszemy też ogólną procedurę stosowania takich heurystyk. W podrozdziale 19.2 omówimy przekształcanie zapytań w plany wykonania, a także objaśnimy optymalizację zagnieżdżonych podzapytań. Zaprezentujemy ponadto przykłady przekształcania zapytań w dwóch scenariuszach: skalania perspektyw w zapytaniach z grupowaniem i transformacji stosowanych w hurtowniach danych zapytań wykorzystujących schemat gwiazdy. Pokrótkę opiszemy również perspektywy zmaterializowane. Podrozdział 19.3 jest poświęcony omówieniu selektywności i szacowania wielkości wyników. Przedstawimy w nim optymalizację kosztową. Wrócimy też do informacji z katalogu systemowego (opisanych wcześniej w punkcie 18.3.4) i zaprezentujemy histogramy. W podrozdziałach 19.4 i 19.5 przedstawimy modele określania kosztów selekcji i złączania. W punkcie 19.5.3 znajdziesz szczegółowe omówienie ważnego problemu kolejności złączania. W podrozdziale 19.6 opiszemy przykład optymalizacji kosztowej. Podrozdział 19.7 zawiera omówienie dodatkowych kwestii związanych z optymalizacją zapytań. Podrozdział 19.8 dotyczy optymalizacji zapytań w hurtowniach danych. Podrozdział 19.9 to omówienie optymalizacji zapytań w bazach Oracle. W podrozdziale 19.10 pokrótce opiszemy optymalizację zapytań semantycznych. Rozdział zakończymy podsumowaniem w podrozdziale 19.11.

---

<sup>1</sup> Dziękujemy Rafiemu Ahmedowi za istotny wkład w powstanie tego rozdziału.

## 19.1. Drzewa zapytań i heurystyki optymalizacji zapytań

W niniejszym podrozdziale zostaną omówione techniki optymalizacji wykorzystujące reguły heurystyczne w celu modyfikowania wewnętrznej reprezentacji zapytania — zwykle przechowywanego w postaci struktury danych drzewa zapytania lub grafu zapytania — w celu zwiększania oczekiwanej wydajności działania. Skaner i analizator składniowy (parser) zapytania w języku SQL najpierw generuje *początkową reprezentację wewnętrzną*, która jest później optymalizowana zgodnie z regułami heurystycznymi. To prowadzi do powstania *zoptymalizowanej reprezentacji zapytania*, odpowiadającej strategii wykonywania zapytania. Następnie generuje się plan wykonania zapytania w celu wykonania grup operacji w oparciu o istniejące ścieżki dostępu do plików związanych z danym zapytaniem.

Jedną z głównych **reguł heurystycznych** (ang. *heuristic rule*) polega na stosowaniu operacji SELECT i PROJECT *przed* operacją JOIN lub innymi operacjami binarnymi. Wynika to z faktu, że rozmiar pliku wynikowego operacji binarnej — takiej jak JOIN — jest zwykle funkcją multiplikatywną rozmiarów plików wejściowych. Operacje SELECT i PROJECT redukują rozmiar pliku, a więc powinny być stosowane *przed* złączeniem i innymi operacjami binarnymi.

W podrozdziale 19.1.1 ponownie zostanie przedstawiona notacja drzew i grafów zapytań, wprowadzona wcześniej w kontekście algebry relacyjnej i rachunku relacji w punktach 8.3.5 i 8.6.5. Mogą one być używane jako podstawa struktur danych używanych jako wewnętrzna reprezentacja zapytań. *Drzewo zapytania* jest wykorzystywane do reprezentowania wyrażeń *algebry relacyjnej* lub rozszerzonej algebry relacyjnej, natomiast *graf zapytania* jest wykorzystywany do reprezentowania wyrażeń *rachunku relacyjnego*. Następnie, w podrozdziale 19.1.2, zostanie omówiony sposób stosowania heurystycznych reguł optymalizacji w celu konwertowania drzewa zapytania do postaci **równoważnego drzewa zapytania**, które reprezentuje inne wyrażenie algebry relacji, bardziej wydajne pod względem wykonania, ale dające w efekcie ten sam wynik co oryginalne. Zostanie również omówiona kwestia równoważności różnych wyrażeń algebry relacji. Wreszcie, w podrozdziale 19.1.3, zostanie omówiona problematyka generowania planów wykonania zapytań.

### 19.1.1. Notacja drzew zapytań i grafów zapytań

**Drzewo zapytań** (ang. *query tree*) to drzewiasta struktura danych, odpowiadająca wyrażeniu algebry relacji. Reprezentuje ona relacje wejściowe zapytania jako *wierzchołki-liście* w drzewie, zaś operacje relacyjne jako wierzchołki wewnętrzne. Wykonanie drzewa zapytania polega na wykonywaniu operacji wierzchołków wewnętrznych w momencie, gdy staną się dostępne ich operandy, a następnie zastępowaniu takich wierzchołków wewnętrznymi relacjami, będącymi wynikiem wykonania danej operacji. Wykonywanie operacji *rozpoczyna się od liści* (reprezentujących wejściowe relacje bazy danych z zapytania), a *kończy na korzeniu* (odpowiadającym ostatniej operacji z zapytania). Wykonywanie kończy się w momencie dotarcia do korzenia i utworzenia relacji wynikowej dla zapytania.

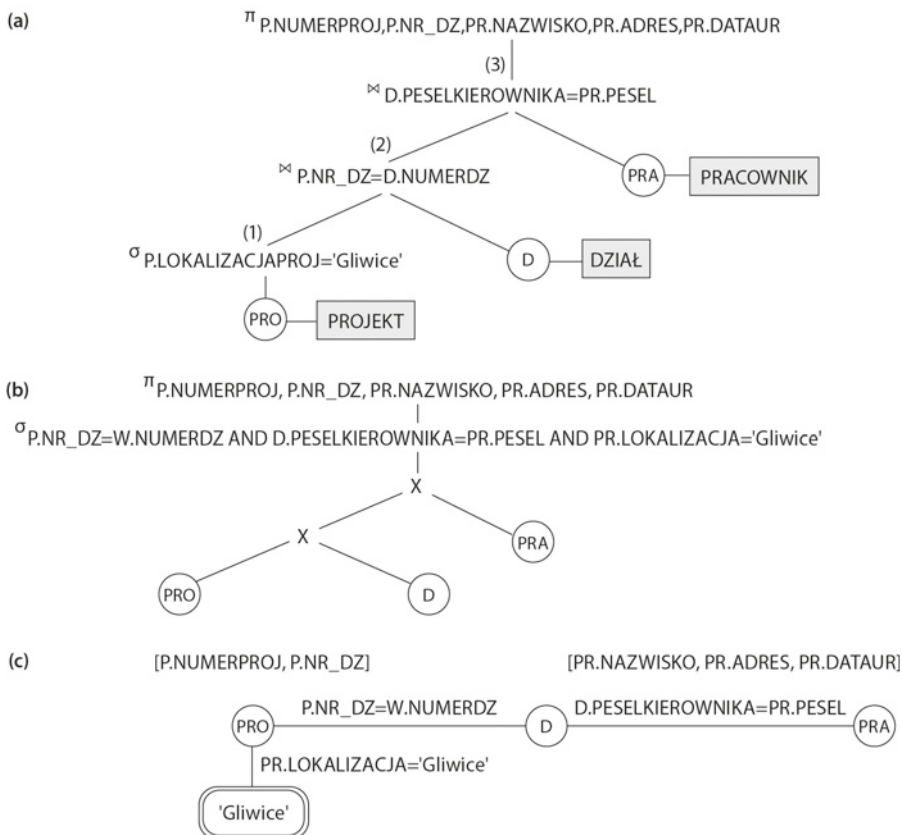
Na rysunku 19.1(a) przedstawiono drzewo zapytania dla zapytania Z2 z rozdziałów od 6. do 8. Dla każdego projektu zlokalizowanego w *Gliwicach* pobieramy jego numer, numer działu nadzorującego oraz nazwisko, adres i datę urodzenia kierownika działu. Zapytanie to określono na schemacie relacji FIRMA z rysunku 5.5 i odpowiada ono następującemu wyrażeniu algebry relacji:

$$\pi_{\text{NUMERPROJ, NR\_DZ, NAZWISKO, ADRES, DATAUR}}(((\sigma_{\text{LOKALIZACJAPROJ} = \text{'GLIWICE'}}(\text{PROJEKT})) \bowtie \text{NR\_DZ} = \text{NUMERDZ}(\text{DZIAŁ})) \bowtie \text{PESELKIEROWNIKA} = \text{PESEL}(\text{PRACOWNIK}))$$

Odpowiada to następującemu zapytaniu SQL:

Z2: **SELECT** PRO.NUMERPROJ, PRO.NR\_DZ, PRA.NAZWISKO, PRA.ADRES, PRA.DATAUR  
**FROM** PROJEKT **AS** PRO, DZIAŁ **AS** D, PRACOWNIK **AS** PRA  
**WHERE** PRO.NR\_DZ = D.NUMERDZ **AND** D.PESELKIEROWNIKA = PRA.PESEL **AND**  
 PRO.LOKALIZACJAPROJ = 'GLIWICE';

Na rysunku 19.1(a) trzy relacje PROJEKT, DZIAŁ i PRACOWNIK są reprezentowane przez wierzchołki-liście P, D i PR, zaś *operacje algebry relacji* wyrażenia są reprezentowane przez wierzchołki wewnętrzne drzewa. W momencie wykonania tego zapytania wierzchołek oznaczony jako (1) na rysunku 19.1(a) musi zostać wykonany przed wierzchołkiem (2), ponieważ pewne krotki wynikowe operacji (1) muszą się stać dostępne, nim będzie można rozpocząć wykonywanie operacji (2). Podobnie wierzchołek (2) musi zostać wykonany i zwrócić wyniki przed wykonaniem wierzchołka (3) itd.



RYSUNEK 19.1. Dwa drzewa zapytań dla zapytania Z2. (a) Drzewo zapytania odpowiadające wyrażeniu algebry relacji dla zapytania Z2. (b) Początkowe (kanoniczne) drzewo zapytań dla zapytania SQL Z2. (c) Graf zapytania dla zapytania Z2

Jak widać, drzewo zapytania reprezentuje określoną kolejność wykonania operacji związanych z zapytaniem. Bardziej naturalną reprezentacją zapytania jest notacja **grafu zapytań** (ang. *query graph*). Na rysunku 19.1(c) przedstawiono graf zapytań dla zapytania Z2. Relacje w tym zapytaniu są reprezentowane przez **wierzchołki relacji** (ang. *relation nodes*), które przedstawia się jako pojedyncze okręgi. Wartości stałe, zwykle pochodzące z warunków selekcji zapytania, są reprezentowane jako **wierzchołki stałe** (ang. *constant nodes*), które przedstawia się jako podwójne okręgi lub owale. Warunki selekcji i złączenia są reprezentowane jako **krawędzie** (ang. *edges*) grafu, co pokazano na rysunku 19.1(c). Wreszcie atrybuty pobierane z każdej relacji są przedstawiane jako nawiasy kwadratowe umieszczone nad relacją.

Reprezentacja grafu zapytań nie określa kolejności, w jakiej mają być wykonywane operacje. Każdemu zapytaniu odpowiada tylko jeden graf<sup>2</sup>. Chociaż niektóre techniki optymalizacji opierano na grafach zapytań (np. w SZBD INGRES), obecnie preferuje się drzewa zapytań, ponieważ w praktyce optymalizator musi określić kolejność wykonania operacji, a grafy zapytań tego nie umożliwiają.

### 19.1.2. Heurystyczna optymalizacja drzew zapytań

Ogólnie rzecz biorąc, wiele różnych wyrażeń algebry relacji — a więc również różnych drzew zapytań — może być **semantycznie równoważnych**. Oznacza to, że odpowiadają one *temu samemu zapytaniu i generują te same wyniki*<sup>3</sup>.

Analizator składni zapytań zwykle generuje standardowe **początkowe drzewo zapytania** (ang. *initial query tree*) odpowiadające zapytaniu SQL bez dokonywania żadnej optymalizacji. Przykładowo, w przypadku zapytania z operacjami selekcji, projekcji i złączenia, takiego jak Z2, drzewo początkowe przedstawiono na rysunku 19.1(b). Najpierw jest stosowany iloczyn kartezjański relacji wymienionych w klauzuli FROM, później są stosowane warunki selekcji i złączenia klauzuli WHERE, a jeszcze później operacja projekcji na atrybutach klauzuli SELECT. Takie **kanoniczne drzewo zapytania** reprezentuje wyrażenie algebry relacji, które jest *bardzo niewydajne w razie bezpośredniego wykonania*, ze względu na występowanie operacji CARTESIAN PRODUCT ( $\times$ ). Na przykład, jeżeli relacje PROJEKT, DZIAŁ i PRACOWNIK posiadają rozmiary rekordów wynoszące 100, 50 i 150 bajtów oraz zawierają 100, 20 oraz 5000 krotek, to wynik iloczynu kartezjańskiego zawierałby 10 milionów krotek o rozmiarze rekordu wynoszącym 300 bajtów. Jednak drzewo zapytania z rysunku 19.1(b) stanowi prostą standardową postać, która może być łatwo utworzona na podstawie zapytania w języku SQL, lecz nigdy nie jest wykonywane. W tym momencie zadaniem heurystycznego optymalizatora zapytań jest przekształcenie początkowego drzewa zapytania do postaci **końcowego drzewa zapytania** (ang. *final query tree*), którego wykonanie będzie wydajne.

Optymalizator musi zawierać reguły *równoważności między wyrażeniami algebry relacji*, które mogą być stosowane względem drzewa początkowego w celu przekształcenia go na końcowe, zoptymalizowane drzewo zapytania. Najpierw zostanie przedstawione niefor-

<sup>2</sup> Stąd graf zapytania odpowiada wyrażeniu *rachunku relacyjnego* (patrz punkt 8.6.5).

<sup>3</sup> Zapytanie można również określić na różne sposoby w wysokopoziomym języku zapytań, takim jak SQL (patrz rozdziały 7. i 8.).

malne omówienie sposobu przekształcania drzew zapytań za pomocą metod heurystycznych. Następnie zajmiemy się omówieniem ogólnych reguł przekształceń i pokażemy, w jaki sposób można ich używać w przypadku algebraicznego optymalizatora heurystycznego.

**Przykład przekształcenia zapytania.** Weźmy pod uwagę następujące zapytanie  $Z$  wykonywane na bazie danych z rysunku 5.5: *Znajdź nazwiska pracowników urodzonych po roku 1957, którzy pracują nad projektem o nazwie 'Wodnik'.* Zapytanie to można w języku SQL zapisać jako:

```
Q: SELECT PRA.NAZWISKO
   FROM PRACOWNIK PRA, PRACUJE_NAD PN, PROJEKT PRO
   WHERE PRO.NAZWAPROJ = 'WODNIK' AND PRO.NUMERPROJ = PN.NR_PROJ
      AND PRA.PESELPRAC = PN.PESEL AND PRA.DATAUR > '1957-12-31';
```

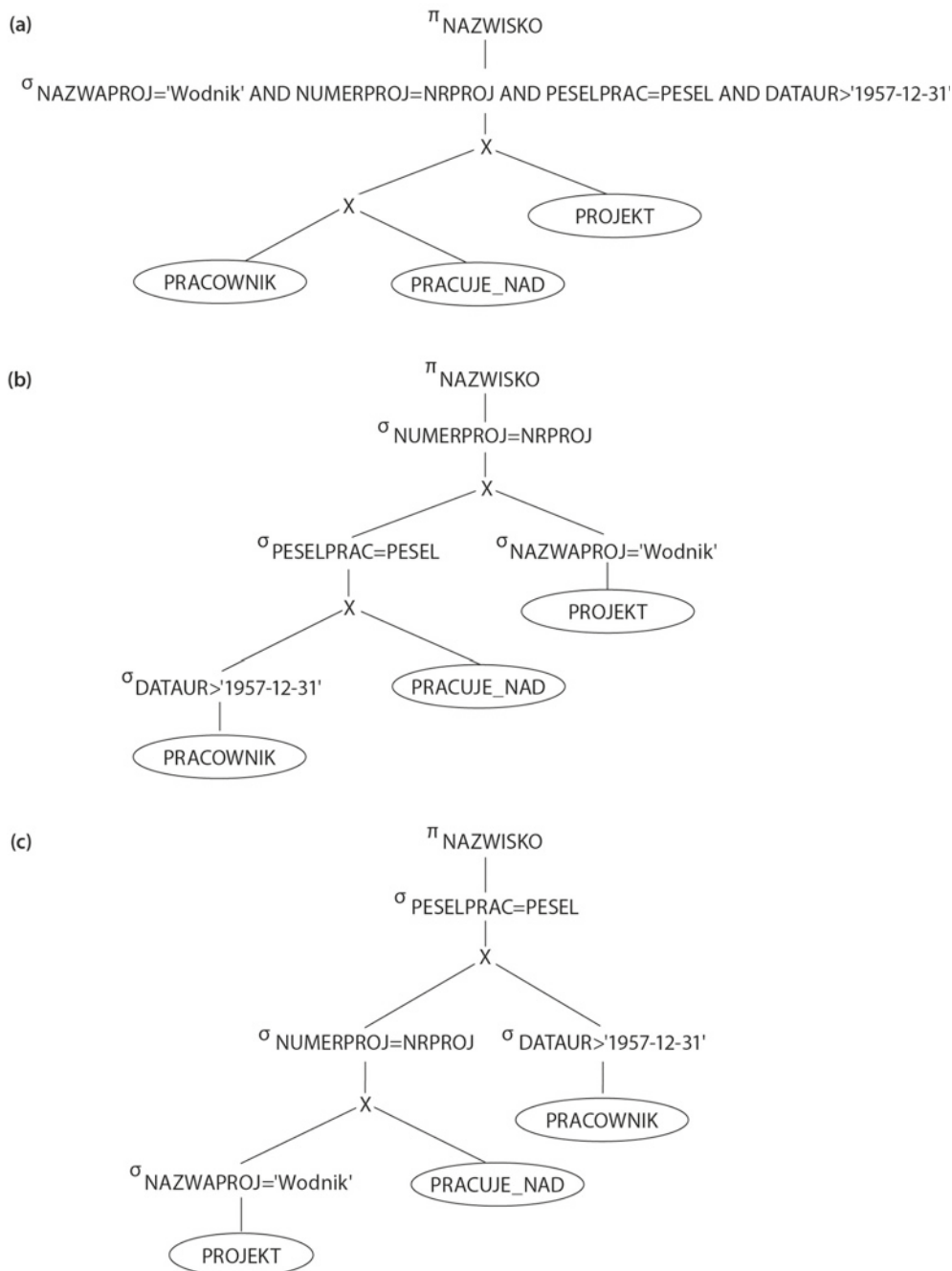
Początkowe drzewo zapytania dla zapytania  $Z$  przedstawiono na rysunku 19.2(a). Bezpośrednie wykorzystanie tego drzewa tworzy bardzo duży plik zawierający iloczyn kartezjański całych plików PRACOWNIK, PRACUJE\_NAD oraz PROJEKT. To dlatego początkowe drzewo zapytania nigdy nie jest wykonywane, ale zostaje przekształcone w równoważne drzewo, którego wykonanie jest wydajne. To konkretne zapytanie potrzebuje tylko jednego rekordu z relacji PROJEKT — dla projektu 'Wodnik' — i tylko tych rekordów z relacji PRACOWNIK, w których data urodzenia jest późniejsza od '1957-12-31'. Na rysunku 19.2(b) przedstawiono ulepszoną wersję drzewa zapytania, które najpierw stosuje operacje SELECT w celu zredukowania liczby krotek występujących w iloczynie kartezjańskim.

Dalsze usprawnienie osiąga się zamieniając pozycjami relacje PRACOWNIK i PROJEKT, co przedstawiono na rysunku 19.2(c). Wykorzystywana tu jest informacja o tym, że NUMERPROJ jest atrybutem klucza w relacji PROJEKT i stąd operacja SELECT wykonywana na relacji PROJEKT pobiera tylko jeden rekord. Drzewo zapytania można jeszcze ulepszyć, zastępując operacje CARTESIAN PRODUCT, po których występuje warunek złączenia, operacjami JOIN, co przedstawiono na rysunku 19.2(d). Kolejne usprawnienie polega na zachowaniu tylko atrybutów wymaganych przez kolejne operacje z relacji pośrednich poprzez uwzględnienie operacji PROJECT ( $\pi$ ) na jak najwcześniejszym etapie przetwarzania drzewa zapytania, co przedstawiono na rysunku 19.2(e). Redukuje to liczbę atrybutów (kolumn) relacji pośrednich, natomiast operacje SELECT redukują liczbę krotek (rekordów).

Jak pokazują przedstawione przykłady, drzewo zapytania można przekształcać krok po kroku do postaci równoważnego drzewa zapytania, którego wykonanie jest wydajniejsze. Jednak należy się upewnić, czy etapy przekształceń zawsze prowadzą do równoważnego drzewa zapytania. W tym celu optymalizator zapytań musi wiedzieć, które reguły przekształceń pozwalają na *zachowanie takiej równoważności*. Poniżej zostaną omówione pewne reguły przekształceń.

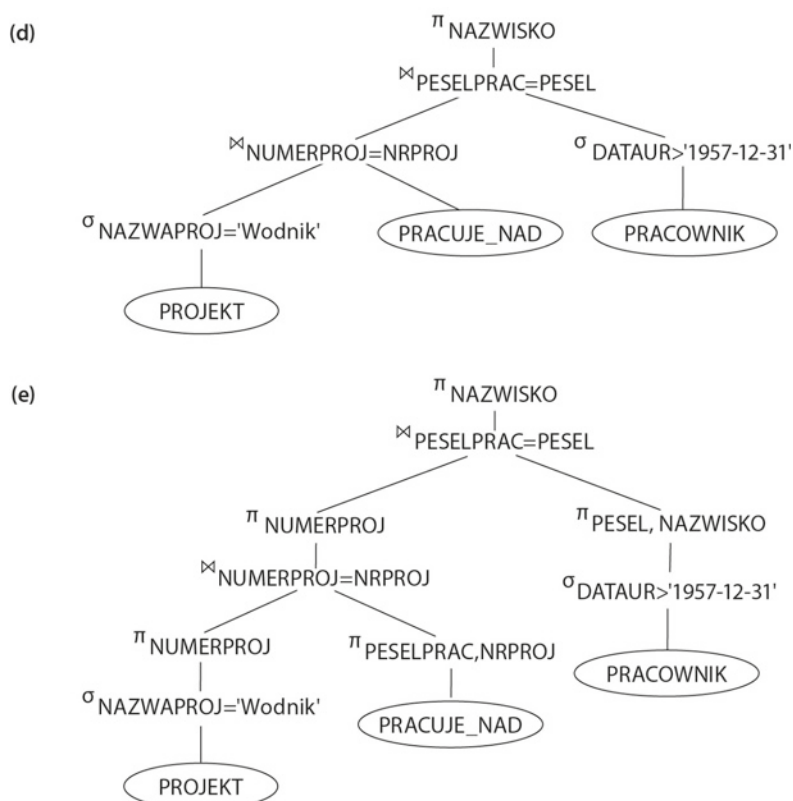
**Ogólne reguły przekształceń dla operacji algebry relacji.** Istnieje wiele reguł przekształcania operacji algebry relacji na równoważne im inne operacje. Na potrzeby optymalizacji zapytań będziemy zainteresowani znaczeniem operacji oraz relacjami wynikowymi. Zatem jeżeli dwie relacje posiadają ten sam zbiór atrybutów w *różnej kolejności*, ale obie reprezentują te same informacje, traktujemy je jako równoważne. W podrozdziale 5.1.2 podano alternatywną definicję *relacji*, która kolejność atrybutów traktuje jako nieistotną. Będziemy używać właśnie tej definicji. Poniżej zostaną określone pewne reguły przekształceń, przydatne w przypadku optymalizacji zapytań, bez przedstawiania dowodu ich poprawności.





RYSUNEK 19.2. Etapy konwersji drzewa zapytania w czasie optymalizacji heurystycznej.

(a) Początkowe (kanoniczne) drzewo zapytania dla zapytania SQL o nazwie Z. (b) Przeniesienie operacji SELECT w dół DRZEWA. (c) Zastosowanie bardziej restrykcyjnej operacji SELECT jako pierwszej.



RYSUNEK 19.2. — ciąg dalszy (d) Zastąpienie operacji CARTESIAN PRODUCT i SELECT operacją JOIN. (e) Przeniesienie operacji PROJECT w dół drzewa

- (1) **Sekwencja operacji  $\sigma$ .** Koniunktywny warunek selekcji można rozbić na sekwencję pojedynczych operacji  $\sigma$ :

$$\sigma_{c1} \text{ AND } c2 \text{ AND } \dots \text{ AND } c_n(R) \equiv \sigma_{c1} (\sigma_{c2} (\dots (\sigma_{c_n}(R)) \dots))$$

- (2) **Przemienność operacji  $\sigma$ .** Operacja  $\sigma$  jest przemienna:

$$\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$$

- (3) **Sekwencja operacji  $\pi$ .** W przypadku sekwencji operacji  $\pi$  można zignorować je wszystkie oprócz ostatniej:

$$\pi_{List a1} (\pi_{List a2} (...(\pi_{List an} (R))...)) \equiv \pi_{List a1} (R)$$

- (4) **Przemienność operacji  $\sigma$  i  $\pi$ .** Jeżeli warunek selekcji  $c$  uwzględnia jedynie atrybuty  $A_1, \dots, A_n$  z listy projekcji, to można zamienić kolejność ich wykonania:

$$\pi_{A1, A2, \dots, An}(\sigma_C(R)) \equiv \sigma_C(\pi_{A1, A2, \dots, An}(R))$$

- (5) **Przemienność operacji  $\bowtie$  (oraz  $\times$ ).** Operacja  $\bowtie$  jest przemienna, podobnie jak operacja  $\times$ :

$$R \bowtie_r S \equiv S \bowtie_r R$$

$$R \times S \equiv S \times R$$

Należy zauważyć, że chociaż kolejność atrybutów może nie być taka sama w relacjach wynikowych obu złączeń (lub iloczynów kartezjańskich), to „znaczenie” pozostaje to samo, ponieważ kolejność atrybutów nie jest istotna w przypadku alternatywnej definicji relacji.

- (6) **Przemienność operacji  $\sigma$  i  $\bowtie$  (lub  $\times$ ).** Jeżeli wszystkie atrybuty w warunku selekcji  $c$  uwzględniają atrybuty tylko jednej z relacji podlegających złączeniu — na przykład  $R$  — obie operacje można zamienić w następujący sposób:

$$\sigma_c (R \bowtie S) \equiv (\sigma_c (R)) \bowtie S$$

Alternatywnie, jeżeli warunek selekcji  $c$  można zapisać jako  $(c1 \text{ AND } c2)$ , gdzie warunek  $c1$  uwzględnia tylko atrybuty relacji  $R$ , zaś warunek  $c2$  uwzględnia tylko atrybuty relacji  $S$ , to operacje można zamienić w sposób następujący:

$$\sigma_c (R \bowtie S) \equiv (\sigma_{c1} (R)) \bowtie (\sigma_{c2} (S))$$

Te same reguły mają zastosowanie w przypadku, gdy operację  $\bowtie$  zastąpi się operacją  $\times$ .

- (7) **Przemienność operacji  $\pi$  i  $\bowtie$  (lub  $\times$ ).** Załóżmy, że lista projekcji ma postać  $L = \{A_1, \dots, A_n, B_1, \dots, B_m\}$ , gdzie  $A_1, \dots, A_n$  są atrybutami relacji  $R$ , zaś  $B_1, \dots, B_m$  są atrybutami relacji  $S$ . Jeżeli warunek złączenia  $c$  uwzględnia tylko atrybuty z listy  $L$ , to obie operacje można zamienić w sposób następujący:

$$\pi_L (R \bowtie S) \equiv (\pi_{A_1, \dots, A_n} (R)) \bowtie (\pi_{B_1, \dots, B_m} (S))$$

Jeżeli warunek złączenia  $c$  zawiera dodatkowe atrybuty niewystępujące na liście  $L$ , muszą one zostać dodane do listy projekcji i potrzebna jest dodatkowa operacja  $\pi$ . Przykładowo, jeżeli w warunku złączenia  $c$  są uwzględnione atrybuty  $A_{n+1}, \dots, A_{n+k}$  relacji  $R$  oraz atrybuty  $B_{m+1}, \dots, B_{m+p}$  relacji  $S$ , ale nie występują one na liście projekcji  $L$ , to operacje można zamienić w następujący sposób:

$$\pi_L (R \bowtie S) \equiv \pi_L ((\pi_{A_1, \dots, A_n, A_{n+1}, \dots, A_{n+k}} (R)) \bowtie (\pi_{B_1, \dots, B_m, B_{m+1}, \dots, B_{m+p}} (S)))$$

W przypadku operacji  $\times$  nie występuje żaden warunek  $c$ , więc ma zastosowanie pierwsza reguła przekształcenia, gdy zastąpimy operator  $\bowtie$  operatorem  $\times$ .

- (8) **Przemienność operacji teoriomnogościowych.** Operacje teoriomnogościowe  $\cup$  i  $\cap$  są przemienne, ale nie jest tak w przypadku operacji  $-$ .
- (9) **Łączność operacji  $\bowtie$ ,  $\times$ ,  $\cup$  i  $\cap$ .** Wymienione cztery operacje są indywidualnie łączne, to znaczy, jeżeli symbol  $\theta$  oznacza którąś z tych czterech operacji (w ramach całego wyrażenia), to otrzymujemy:

$$(R \theta S) \theta T \equiv R \theta (S \theta T)$$

- (10) **Przemienność operacji  $\sigma$  z operacjami teoriomnogościowymi.** Operacja  $\sigma$  jest przemienna z operacjami  $\cup$ ,  $\cap$  i  $-$ . Jeżeli symbol  $\theta$  oznacza którąś z tych trzech operacji (w ramach całego wyrażenia), to otrzymujemy:

$$\sigma_c (R \theta S) \equiv (\sigma_c (R)) \theta (\sigma_c (S))$$

- (11) **Operacja  $\pi$  jest przemienna z operacją  $\cup$ :**

$$\pi_L (R \cup S) \equiv (\pi_L (R)) \cup (\pi_L (S))$$

- (12) **Konwersja sekwencji operacji  $(\sigma, \times)$  na operację  $\bowtie$ .** Jeżeli warunek  $c$  operacji  $\sigma$ , który występuje po operatorze  $\times$ , odpowiada warunkowi złączenia, to można przekonwertować sekwencję operacji  $(\sigma, \times)$  na operację  $\bowtie$  w następujący sposób:

$$(\sigma_c(R \times S)) \equiv (R \bowtie_c S)$$

- (13) **Stosowanie  $\sigma$  razem z różnicą zbiorów.**

$$\sigma_c(R - S) = \sigma_c(R) - \sigma_c(S)$$

Operację  $\sigma$  można też zastosować do tylko jednej relacji:

$$\sigma_c(R - S) = \sigma_c(R) - S$$

- (14) **Stosowanie  $\sigma$  do tylko jednego argumentu operatora  $\cap$ .**

Jeśli w warunku  $\sigma_c$  wszystkie atrybuty pochodzą z relacji  $R$ , to:

$$\sigma_c(R \cap S) = \sigma_c(R) \cap S$$

- (15) **Różne trywialne transformacje.**

Jeśli  $S$  jest pusta, to  $R \cup S = R$ .

Jeżeli warunek  $c$  w  $\sigma_c$  jest spełniony dla całej relacji  $R$ , to  $\sigma_c(R) = R$ .

Istnieją również inne możliwości przekształceń. Przykładowo, warunek selekcji lub złączenia  $c$  można przekonwertować na warunek mu równoważny, używając następujących reguł algebry Boole'a (praw DeMorgana):

$$\text{NOT } (c1 \text{ AND } c2) \equiv (\text{NOT } c1) \text{ OR } (\text{NOT } c2)$$

$$\text{NOT } (c1 \text{ OR } c2) \equiv (\text{NOT } c1) \text{ AND } (\text{NOT } c2)$$

Dodatkowe przekształcenia omówione w rozdziałach 4., 5. i 6. nie będą tu powtarzane. Poniżej omówimy, w jaki sposób można używać przekształceń w przypadku optymalizacji heurystycznej.

**Zarys algorytmu heurystycznej optymalizacji algebraicznej.** Poniżej możemy przedstawić zarys etapów algorytmu, który wykorzystuje niektóre z podanych powyżej reguł w celu przekształcenia początkowego drzewa zapytania do postaci drzewa końcowego, którego wykonanie jest wydajniejsze (w większości przypadków). Algorytm będzie prowadził do przekształceń podobnych do omówionych w przykładzie z rysunku 19.2. Poniżej wymieniono kolejne etapy działania algorytmu.

- (1) Używamy reguły 1. w celu rozbicia każdej operacji SELECT z warunkami koniunktywnymi na sekwencję operacji SELECT. Daje to większą swobodę w zakresie przesuwania operacji SELECT w dół różnych gałęzi drzewa.
- (2) Używając reguł 2., 4., 6. oraz 10., 13. i 14., dotyczących przemienności operacji SELECT z innymi operacjami, przesuwamy operację SELECT jak najniżej w drzewie zapytania, na ile pozwalają na to atrybuty uwzględnione w warunku selekcji. Jeśli w warunku występują atrybuty z *tylko jednej tabeli* (co oznacza, że jest to *warunek selekcji*), operacja jest przenoszona do liścia reprezentującego daną tabelę. Jeżeli warunek obejmuje atrybuty z dwóch tabel (jest więc *warunkiem złączenia*), zostaje przeniesiony w dół drzewa w miejsce, gdzie te dwie tabele są łączone.
- (3) Używając reguł 5. i 9., dotyczących przemienności i łączności operacji binarnych, zmieniamy rozmieszczenie liści drzewa zgodnie z następującymi kryteriami. Najpierw ustawiamy relacje wierzchołków-liści z najbardziej restrykcyjnymi

operacjami SELECT, tak aby w reprezentacji drzewa zapytania były one wykonywane jako pierwsze. Definicja *najbardziej restrykcyjnych* operacji SELECT może oznaczać, że albo są to te operacje, które dają w wyniku relację o najmniejszej liczbie krotek, albo o najmniejszym rozmiarze bezwzględny<sup>4</sup>. Kolejną możliwością jest zdefiniowanie najbardziej restrykcyjnej operacji SELECT jako tej, która charakteryzuje się najmniejszą selektywnością. Jest to bardziej praktyczne rozwiązanie, gdyż oszacowania selektywności często są dostępne w katalogu SZBD. Następnie upewniamy się, że uporządkowanie wierzchołków-liści nie powoduje wystąpienia operacji iloczynu kartezjańskiego. Przykładowo, jeżeli dwie relacje z najbardziej restrykcyjnymi operacjami SELECT nie posiadają warunku bezpośredniego złączenia, może okazać się pożądane dokonanie zmiany uporządkowania wierzchołków-liści w celu uniknięcia występowania operacji iloczynu kartezjańskiego<sup>5</sup>.

- (4) Używając reguły 12., łączymy operację CARTESIAN PRODUCT z kolejną operacją SELECT w drzewie w celu utrzymania operacji JOIN, jeżeli warunek reprezentuje warunek złączenia.
- (5) Używając reguł 3., 4., 7. i 11. dotyczących łączenia w sekwencje operacji PROJECT i przemienności operacji PROJECT z innymi operacjami, rozbijamy i przesuwamy jak najdalej listy atrybutów projekcji w dół drzewa, w razie potrzeby tworząc nowe operacje PROJECT. Po wykonaniu każdej operacji PROJECT powinny zostać zachowane tylko te atrybuty, które są potrzebne w wyniku zapytania oraz w kolejnych operacjach w drzewie zapytania.
- (6) Identyfikujemy poddrzewa reprezentujące grupy operacji, które mogą być wykonane przez pojedynczy algorytm.

W omawianym przykładzie na rysunku 19.2(b) przedstawiono drzewo z rysunku 19.2(a) po wykonaniu 1. i 2. etapu algorytmu. Na rysunku 19.2(c) przedstawiono drzewo po wykonaniu etapu 3., na rysunku 19.2(d) — po wykonaniu etapu 4., zaś na rysunku 19.2(e) — po wykonaniu etapu 5. W etapie 6. możemy w ramach jednego algorytmu zgrupować razem operacje z poddrzewa, którego korzeniem jest operacja  $\pi_{\text{PESELPRAC}}$ . Możemy również zgrupować pozostałe operacje w ramach kolejnego poddrzewa, w którym krotki będące wynikiem działania pierwszego algorytmu zastępują poddrzewo, którego korzeniem jest operacja  $\pi_{\text{PESELPRAC}}$ , ponieważ pierwsze grupowanie oznacza, że to poddrzewo jest wykonywane jako pierwsze.

**Podsumowanie problematyki heurystycznej optymalizacji algebraicznej.** Główna zasada określa, że najpierw należy stosować operacje, które redukują rozmiar wyników pośrednich. Chodzi tu między innymi o jak najwcześniejsze wykonanie operacji SELECT w celu zredukowania liczby krotek oraz operacji PROJECT w celu zredukowania liczby atrybutów. Dokonuje się tego, przesuując operacje SELECT i PROJECT jak najdalej w dół drzewa. Ponadto, najbardziej restrykcyjne operacje SELECT i JOIN, czyli takie, które dają w wyniku relacje o najmniejszej liczbie krotek lub najmniejszym rozmiarze bezwzględny, powinny być wy-

<sup>4</sup> Można wykorzystywać dowolną z tych definicji, ponieważ opisywane reguły mają charakter heurystyczny.

<sup>5</sup> Należy zauważyć, że iloczyn kartezjański jest w pewnych przypadkach dopuszczalny — na przykład wówczas, gdy każda relacja posiada tylko jedną krotkę, ponieważ dla każdej z nich został wcześniej określony warunek selekcji bazujący na polu klucza.

konane przed innymi podobnymi operacjami. Zapewnia się to, reorganizując wierzchołki liście drzewa, unikając występowania iloczynów kartezjańskich i odpowiednio dostosowując pozostałą część drzewa.

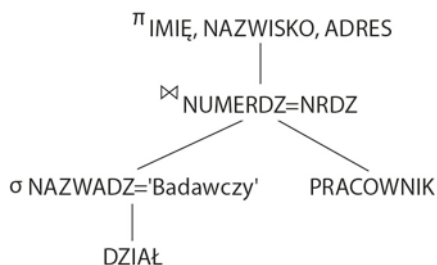
## 19.2. Wybór planów wykonania zapytań

### 19.2.1. Różne sposoby wykonywania zapytań

Plan wykonania dla wyrażenia algebry relacyjnej reprezentowanego jako drzewo zapytania uwzględnia informacje dotyczące metod dostępu istniejących dla każdej relacji, jak również algorytmy używane w celu obliczania operatorów relacyjnych reprezentowanych w drzewie. Jako przykład można rozważyć zapytanie Z1 z rozdziału 7., dla którego odpowiednie wyrażenie algebry relacyjnej ma następującą postać:

$$\pi_{\text{IMIE}, \text{NAZWISKO}, \text{ADRES}}(\sigma_{\text{NAZWADZ} = \text{'BADAWCZY'}}(\text{DZIAŁ}) \bowtie_{\text{NUMERDZ} = \text{NRDZ}} \text{PRACOWNIK})$$

Drzewo zapytania przedstawiono na rysunku 19.3. W celu przekształcenia go na plan wykonania optymalizator może wybrać przeszukiwanie według indeksu dla operacji SELECT na tabeli DZIAŁ (zakładając, że taki istnieje), oparty na indeksie algorytm złączenia pętli zagnieżdżonych dla operacji złączenia, przetwarzający w pętli rekordy z wyniku operacji SELECT na tabeli DZIAŁ (przy założeniu istnienia indeksu na atrybucie NRDZ relacji PRACOWNIK), oraz przegląd wyniku operacji JOIN dla operatora PROJECT. Ponadto podejście związane z wykonaniem zapytania może określać wykonywanie obliczeń materializowanych lub potokowych, przy czym zwykle preferowane są obliczenia potokowe (jeśli są wykonalne).



RYСУNEK 19.3. Drzewo zapytania dla zapytania Z1

W przypadku **obliczeń materializowanych** (ang. *materialized evaluation*) wynik operacji jest przechowywany jako relacja tymczasowa (to znaczy, wynik jest *fizycznie materializowany*). Przykładowo, operacja złączenia może zostać wyliczona i cały wynik zachowany jest jako relacja tymczasowa, która później jest wczytywana jako dane wejściowe przez algorytm wyliczający operację PROJECT i która daje w wyniku tabelę wynikową zapytania. Z drugiej strony, w przypadku **obliczeń potokowych** (ang. *pipelined evaluation*), w miarę jak są tworzone wynikowe operacji, są one bezpośrednio przekazywane do kolejnej operacji w sekwencji zapytań. Potokową strategię przetwarzania zapytań opisano w podrozdziale 18.7. Przykładowo, w miarę jak wybrane krotki relacji DZIAŁ są tworzone przez operację SELECT, są one umieszczane w buforze. Algorytm operacji JOIN

pobiera następnie krotki z bufora i krotki stanowiące wynik operacji JOIN są za pomocą potoku przekazywane do algorytmu operacji projekcji. Korzyścią metody potokowej są oszczędności wynikające z uniknięcia konieczności zapisywania wyników pośrednich na dysku oraz wczytywania ich z powrotem dla kolejnej operacji.

W podrozdziale 19.1 opisano możliwość przekształcania drzew zapytań w równoważne drzewa, pozwalające na wydajniejsze przetwarzanie zapytania ze względu na czas wykonania i wykorzystanie zasobów. Istnieją też bardziej złożone transformacje zapytań służące do optymalizacji, a raczej do „usprawniania” przetwarzania. Te transformacje można stosować w sposób heurystyczny lub oparty na kosztach.

W punktach 7.1.2 i 7.1.3 napisano, że podzapytania zagnieżdżone mogą występować w klauzuli WHERE, a także w klauzuli FROM zapytań w języku SQL. Jeśli w klauzuli WHERE blok wewnętrzny zawiera referencję do relacji używanej w bloku zewnętrznym, powstaje skorelowane zapytanie zagnieżdżone. Gdy w klauzuli FROM używane jest zapytanie w celu zdefiniowania relacji wynikowej lub pochodnej, traktowanej jako relacja w zapytaniu zewnętrznym, powstaje odpowiednik perspektywy. Optymalizator obsługuje oba te rodzaje podzapytań zagnieżdżonych — przekształca je i ponownie zapisuje całe zapytanie. W dwóch następnych punktach przedstawimy za pomocą przykładów dwa rodzaje przekształcania i ponownego zapisywania zapytań. Te dwa rodzaje nazywamy optymalizacją podzapytań zagnieżdżonych i scalaniem podzapytań (perspektyw). W podrozdziale 19.8 wrócimy do tego zagadnienia w kontekście hurtowni danych i przedstawimy optymalizację z przekształceniami w schemacie gwiazdy.

### 19.2.2. Optymalizacja podzapytań zagnieżdżonych

Zapytania zagnieżdżone omówiliśmy w punkcie 7.1.2. Przyjrzyj się następującemu zapytaniu:

```
SELECT PRA1.IMIE, E1.NAZWISKO
FROM   PRACOWNIK PRA1
WHERE  PRA1.PENSJA =( SELECT MAX(PENSJA)
                       FROM PRACOWNIK PRA2)
```

W tym zapytaniu zagnieżdżonym występuje blok zapytania wewnątrz zewnętrznego bloku zapytania. Wykonanie całego zapytania wymaga przetworzenia najpierw zapytania zagnieżdżonego, co daje pojedynczą wartość w postaci maksymalnej pensji M z relacji PRACOWNIK. Następnie wykonywany jest blok zewnętrzny z warunkiem selekcji PENSJA = M. Maksymalną pensję można uzyskać jako najwyższą wartość indeksu na atrybucie PENSJA (jeśli taki istnieje) lub z katalogu (jeżeli jest aktualny). Zapytanie zewnętrzne jest przetwarzane na podstawie tego samego indeksu. Jeśli indeks nie istnieje, w obu zapytaniach potrzebne jest wyszukiwanie liniowe.

Skorelowane zagnieżdżone zapytania języka SQL opisano w punkcie 7.1.3. W podzapytaniu skorelowanym zapytanie wewnętrzne obejmuje referencję do zapytania zewnętrznego, przy czym używana jest jedna lub kilka zmiennych. To podzapytanie działa jak funkcja zwracająca zbiór wartości dla każdej wartości zmiennej lub kombinacji takich zmiennych.

Założmy, że w bazie z rysunku 5.5 zmodyfikowaliśmy relację DZIAŁ w następujący sposób:

```
DZIAŁ (NUMERDZ, NAZWADZ, PESELKIEROWNIKA, DATAPRZEJKIEROWNICTWA, KOD)
```



Przyjrzyj się następującemu zapytaniu:

```
SELECT IMIĘ, NAZWISKO, PENSJA
FROM PRACOWNIK PRA
WHERE EXISTS ( SELECT *
                FROM DZIAŁ D
                WHERE D.NUMERDZ = PRA.NRDZ AND D.KOD=30332);
```

Tu podzapytanie zagnieżdżone przyjmuje jako parametr wartość PRA.NRDZ, określającą dział, gdzie pracownik jest zatrudniony, i zwraca wartość TRUE lub FALSE jako funkcję zależną od tego, czy kod pocztowy lokalizacji tego działu to 30-332. Naiwna strategia przetwarzania tego zapytania polega na tym, by wykonać wewnętrzne podzapytanie zagnieżdżone dla każdej krotki z relacji zewnętrznej. Jest to niewydajne. Gdy tylko to możliwe, optymalizator języka SQL próbuje przekształcić zapytania z podzapytaniami zagnieżdżonymi w operacje złączenia. Złączenie można następnie wykonać za pomocą jednej z metod opisanych w podrozdziale 18.4. Przedstawione wcześniej zapytanie można przekształcić na następującą postać:

```
SELECT IMIĘ, NAZWISKO, PENSJA
FROM PRACOWNIK PRA, DZIAŁ D
WHERE D.NUMERDZ = PRA.NRDZ AND D.KOD=30332
```

Proces usuwania zagnieżdżonego zapytania oraz przekształcania zapytań zewnętrznego i wewnętrznego w jeden blok to **eliminowanie zagnieżdżenia**. Tu stosowane jest złączenie wewnętrzne, ponieważ D.NUMERDZ jest unikatowy, a złączenie jest równościowe. To gwarantuje, że krotka z relacji PRACOWNIK pasuje do najwyżej jednej krotki relacji DZIAŁ. W rozdziale 7. pokazano, że w zapytaniu Z16 z podzapytaniem połączonym łącznikiem IN także wyeliminowano zagnieżdżenie i utworzono zapytanie w postaci jednego bloku ze złączeniem. W języku SQL zapytania obejmujące podzapytanie zagnieżdżone połączone łącznikiem IN lub ANY można zwykle przekształcić w zapytanie z jednym blokiem. Inne techniki to np. tworzenie na podstawie podzapytań tymczasowych tabel wynikowych i używanie ich w złączeniach.

Ponownie prezentujemy tu przykładowe zapytanie z podrozdziału 18.1. Zauważ, że operator IN to odpowiednik operatora =ANY.

```
Q (SJ) :
SELECT COUNT(*)
FROM DZIAŁ D
WHERE D.NUMERDZ IN ( SELECT PRA.NRDZ
                     FROM PRACOWNIK PRA
                     WHERE PRA.PENSJA >20000)
```

Także tu optymalizator ma dwie możliwości:

- (1) Przetworzyć podzapytanie zagnieżdżone dla każdej krotki zewnętrznej, co jest niewydajne.
- (2) Wyeliminować zagnieżdżenie za pomocą **złączenia częściowego**, co jest znacznie wydajniejsze niż rozwiązanie 1. W podrozdziale 18.1 zastosowano tę możliwość, aby wprowadzić i zdefiniować operator złączenia częściowego. Zauważ, że przy eliminowaniu zagnieżdżenia tego podzapytania, czyli przy tworzeniu zapytania w formie pojedynczego bloku, *nie można* stosować złączenia wewnętrznego,

ponieważ krotka z relacji DZIAŁ może pasować do więcej niż jednej krotki z relacji PRACOWNIK, co da błędny wynik. Łatwo zobaczyć, że podzapytanie zagnieżdżone działa jak **filtr**, dlatego (w odróżnieniu od złączenia wewnętrznego) nie może wygenerować większej liczby wierszy niż zawiera ich tabela DZIAŁ. Złączenie częściowe symuluje to działanie.

Proces, który opisaliśmy jako **eliminowanie zagnieżdżenia**, można też nazwać **eliminowaniem powiązania** (ang. *decorrelation*). W podrozdziale 18.1 przedstawiliśmy inny przykład, gdzie zastosowano łącznik NOT IN. Ten łącznik został za pomocą operacji **antyłączenia** przekształcony w zapytanie w formie jednego bloku. Optymalizacja złożonych podzapytań zagnieżdżonych jest trudna i wymaga stosowania skomplikowanych technik. Dwie takie techniki omówimy w punkcie 19.2.3. Eliminowanie zagnieżdżenia to wartościowa technika optymalizacji, często stosowana w optymalizatorach języka SQL.

### 19.2.3. Scalanie podzapytań (perspektyw)

W niektórych sytuacjach podzapytanie występuje w klauzuli FROM zapytania i działa jak relacja pochodna, podobna do używanej w zapytaniu zdefiniowanej perspektywy. Takie podzapytanie z klauzuli FROM można nazwać perspektywą wewnątrzwierszową. Czasem perspektywa zdefiniowana wcześniej jako odrębne zapytanie jest używana jako jedna z relacji będących argumentami nowego zapytania. Wtedy transformację zapytania można nazwać scalaniem perspektyw lub podzapytań. Opisane tu techniki scalania perspektyw dotyczą zarówno perspektyw wewnątrzwierszowych, jak i zdefiniowanych.

Przyjrzyj się trzem następującym relacjom:

```
PRAC (PESEL, IM, NAZ, NRDZ)
DZIAŁ (NRDZ, NAZWADZ, NAZWKIER, IDBUD)
BUD (IDBUD, PIĘTRA, ADR, TEL)
```

Znaczenie tych relacji nie wymaga wyjaśnień. Ostatnia relacja reprezentuje budynki, w których dział jest zlokalizowany. TEL to numer telefonu portierni budynku.

W następnym zapytaniu w klauzuli FROM używana jest perspektywa wewnątrzwierszowa. Pobiera ona dla pracowników o imieniu Jan nazwisko, adres i numer telefonu budynku, w którym dana osoba pracuje:

```
SELECT P.IM, V.ADR, V.TEL
FROM PRAC P, ( SELECT D.NRDZ, D.NAZWADZ, B.ADR, B.TEL
                FROM DZIAŁ D, BUD B
                WHERE D.IDBUD = B.IDBUD ) V
WHERE V.NRDZ = P.NRDZ AND P.IM = "Jan";
```

To zapytanie łączy tabelę PRAC z perspektywą V, która zawiera adres i numer telefonu budynku, w którym pracownik pracuje. Perspektywa ta łączy dwie tabele: DZIAŁ i BUD. To zapytanie można wykonać, najpierw tymczasowo materializując perspektywę, a następnie łącząc ją z tabelą PRAC. Optymalizator może tylko określić kolejność złączania: P, V lub V, P. Na potrzeby obliczania perspektywy możliwa kolejność złączania to D, B lub B, D. Tak więc łączna liczba możliwych sposobów złączania to tylko 4. Ponadto niemożliwe jest złączanie P, V na podstawie indeksu, ponieważ w V nie występuje indeks na atrybucie złączenia NRDZ. Operacja **scalania perspektywy** scala tabele z perspektywy z tabelami z bloku zapytania zewnętrznego i daje następujące zapytanie:

```
SELECT P.IM, B.ADR, B.TEL
FROM PRAC P, DZIAŁ D, BUD B
WHERE D.IDBUD = B.IDBUD AND D.NRDZ = P.NRDZ AND P.IM = "Jan";
```

W bloku scalonego zapytania w klauzuli FROM występują trzy tabele, co daje 8 możliwych kolejności złączania. Dostępne są też indeksy na atrybutach NRDZ z tabeli DZIAŁ i IDBUD z tabeli BUD; te indeksy można wykorzystać w złączaniu pętli zagnieżdżonej z użyciem indeksu, co wcześniej było niemożliwe. Jako ćwiczenie dla Czytelnika pozostawiamy opracowanie i porównanie planów wykonania zapytania ze scalaniem i bez scalania.

Perspektywy zawierające operacje selekcji, projekcji i złączania są zwykle uważane za proste i zawsze można do nich zastosować opisane scalanie perspektyw. Scalanie przezważnie zapewnia dodatkowe możliwości i skutkuje lepszym planem wykonania niż w scenariuszu bez scalania. Czasem możliwe stają się też inne optymalizacje, np. pominięcia tabeli w zapytaniu zewnętrznym, jeśli jest ona używana w perspektywie. W niektórych sytuacjach (gdy perspektywa jest złożona i obejmuje operacje na zbiorach takie jak DISTINCT, OUTER JOIN, AGGREGATION lub GROUP BY) scalanie perspektyw może prowadzić do błędów. Dalej opiszemy możliwe scenariusze scalania perspektyw, gdy używana jest operacja GROUP BY.

**Scalanie perspektyw, gdy używana jest operacja GROUP BY.** Gdy w perspektywie obok wspomnianych wcześniej selekcji, projekcji i złączeń występują dodatkowe konstrukty, scalanie w przedstawiony sposób może być niepożądane. Opóźnienie operacji GROUP BY do momentu po złączeniu może zapewniać korzyść w postaci zmniejszenia ilości danych uwzględnianych w późniejszych złączeniach. Optymalizator zwykle uwzględnia plany wykonania ze scalaniem i bez scalania, a także porównuje ich koszty, aby ustalić sensowność scalania. Przedstawimy to na przykładzie.

Przyjrzyj się następującym relacjom:

```
SPRZEDAŻ (IDKLI, IDPROD, DATA, SZTUKSPRZED)
KLIENT (IDKLI, NAZWKLI, KRAJ, EMAILKLI)
PRODUKT (IDPROD, NAZWAPROD, SZTUKDOST)
```

Zapytanie: wymień klientów z Francji, którzy kupili więcej niż 50 sztuk produktu „Pierścień\_234”, można zapisać w przedstawiony poniżej sposób.

Utworzenie perspektywy w celu zliczenia kupionych sztuk dla par <IDKLI, IDPROD>.

```
CREATE VIEW CP_BOUGHT_VIEW AS
SELECT SUM (S.SZTUKSPRZED) as KUPIONE, S.IDKLI, S.IDPROD
FROM SPRZEDAŻ S
GROUP BY S.IDKLI, S.IDPROD;
```

Zapytanie używające tej perspektywy wygląda tak:

```
ZG: SELECT K.IDKLI, K.NAZWKLI, K.EMAILKLI
FROM KLIENT K, PRODUKT P, PERSP_KUPIONE_KP P1
WHERE P.IDPROD = P1.IDPROD AND K.IDKLI = P1.IDKLI AND P1.KUPIONE > 50
AND NAZWAPROD = "Pierścień_234" AND K.KRAJ = "Francja";
```

Można najpierw przetworzyć perspektywę P1 i tymczasowo zmaterializować jej wyniki, a następnie wykonać zapytanie ZG, używając zmaterializowanej perspektywy jako jednej z tabel złączenia. Dzięki scalaniu zapytanie przyjmuje postać:

```

ZT: SELECT K.IDKLI, K.NAZWKLI, K.EMAILKI
FROM KLIENT K, PRODUKT P, SPRZEDAŻ S
WHERE P.IDPROD = S.IDPROD AND K.IDKLI = S.IDKLI AND
      NAZWAPROD = "Pierścień_234" AND K.KRAJ = "Francja"
GROUP BY P.IDPROD, P.ROWID, K.ROWID, K.IDKLI, K.NAZWKLI, K.EMAILKI
HAVING SUM (S.SZTUKSPRZED) > 50;

```

Po scaleniu wyników zapytanie ZT jest dużo wydajniejsze i mniej kosztowne do wykonania. Oto przyczyny: przed scaleniem perspektywa P1 grupuje dane według całej tabeli SPRZEDAŻ i materializuje wynik, co jest kosztowne. W przekształconym zapytaniu grupowanie dotyczy złączenia trzech tabel. W tej operacji uwzględniana jest jedna krotka z tabeli PRODUKT, co skutkuje znaczącym odfiltrowaniem danych z tabeli SPRZEDAŻ. Złączenie w zapytaniu ZT po przekształceniach może być trochę bardziej kosztowne niż w pierwotnej wersji, ponieważ uwzględniana jest cała relacja SPRZEDAŻ zamiast zagregowanej perspektywy PERSP\_KUPIONE\_KP z zapytania ZG. Warto jednak zauważyć, że operacja GROUP BY z P1 generuje tabelę, której liczność nie jest o wiele niższa niż w relacji SPRZEDAŻ. Jest tak, ponieważ grupowanie opiera się na atrybutach <IDKLI, IDPROD>, których powtarzalność w relacji SPRZEDAŻ może być niska. Zwróć też uwagę na zastosowanie atrybutów P.ROWID i K.ROWID, dotyczących unikatowych identyfikatorów wierszy dodawanych w celu zachowania równoważności z pierwotnym zapytaniem. Przypominamy, że decyzja o scaleniu perspektyw z operacją GROUP BY musi zostać podjęta przez optymalizator na podstawie oszacowania kosztów.

## 19.2.4. Perspektywy zmaterializowane

Perspektywy omówiono w podrozdziale 7.3, gdzie opisano także ich materializowanie. Perspektywa jest zdefiniowana w bazie w formie zapytania, a **perspektywa zmaterializowana** przechowuje wyniki takiego zapytania. Używanie perspektyw zmaterializowanych w celu uniknięcia niektórych obliczeń z zapytania to następna technika optymalizowania zapytań. Widok zmaterializowany może być przechowywany albo tymczasowo, aby umożliwić przetwarzanie innych zapytań z jego użyciem, albo trwale, jak często ma to miejsce w hurtowniach danych (patrz rozdział 29.). Widok zmaterializowany obejmuje dane pochodne, ponieważ jego zawartość można obliczyć w wyniku wykonania zapytania definiującego ten widok. Podstawowa idea związana z materializacją jest taka, że znacznie łatwiej jest wczytać go, gdy jest potrzebny, i kierować zapytania do niego, niż obliczać go ponownie od podstaw. Jeśli perspektywa obejmuje kosztowne operacje, takie jak złączenie, agregacja itd., pozwala uzyskać znaczne oszczędności.

Rozważ np. perspektywę V2 z podrozdziału 7.3. Jest ona zdefiniowana jako relacja za pomocą złączenia relacji DZIAŁ i PRACOWNIK. Dla każdego działu wyznaczane są łączna liczba pracowników i łączne wynagrodzenie wypłacane pracownikom tego działu. Jeśli te informacje są często potrzebne w raportach i zapytaniach, perspektywę można trwale zapisać. Perspektywa zmaterializowana może obejmować dane powiązane tylko z fragmentem lub podwyrażeniem zapytania. Dlatego potrzebny jest zaawansowany algorytm, aby zastąpić wyłącznie odpowiednie fragmenty zapytania jedną lub kilkoma perspektywami zmaterializowanymi i obliczyć pozostałe części zapytania w standardowy sposób. W podrozdziale 7.3 wymieniliśmy też trzy strategie aktualizowania (odświeżania) perspektyw:

- Natychmiastowa aktualizacja, co powoduje odświeżanie perspektywy bezpośrednio po aktualizacji dowolnej relacji używanej w perspektywie.
- „Leniwa” aktualizacja, kiedy to perspektywa jest odświeżana tylko na żądanie.
- Okresowa (inaczej: odroczone) aktualizacja, kiedy to perspektywa jest aktualizowana później, np. w regularnych odstępach czasu.

Gdy stosowana jest natychmiastowa aktualizacja, trzeba ponosić wysokie koszty zapewnienia aktualności perspektywy w reakcji na zmiany (wstawianie, usuwanie lub modyfikowanie danych) w dowolnej z relacji podstawowych. Przykładowo, usunięcie pracownika z bazy, zmiana jego wynagrodzenia lub zatrudnienie nowej osoby wpływają na krótką reprezentującą ten dział w perspektywie, co wymaga natychmiastowej aktualizacji perspektywy P2 z podrozdziału 7.3. Czasem takie aktualizacje są wykonywane ręcznie za pomocą programów, które po zmianach relacji podstawowej modyfikują wszystkie perspektywy zdefiniowane z użyciem tej relacji. Oczywiście nie ma wtedy gwarancji, że uwzględnione zostaną wszystkie perspektywy. Można też wykorzystać wyzwalacze (patrz podrozdział 7.2) aktywowane po aktualizacji relacji podstawowej i wprowadzające odpowiednie zmiany w perspektywach zmaterializowanych. Proste i naiwne podejście polega na ponownym obliczaniu całej perspektywy po każdej aktualizacji dowolnej tabeli podstawowej, co powoduje nieakceptowalne koszty. Dlatego obecnie w większości relacyjnych SZBD stosuje się przyrostową aktualizację perspektyw, co opiszemy dalej.

**Przyrostowa aktualizacja perspektyw.** Podstawowa zasada przyrostowej aktualizacji perspektyw polega na tym, że zamiast tworzyć perspektywę od podstaw, można ją modyfikować przyrostowo, uwzględniając tylko zmiany wprowadzone od ostatniego jej utworzenia i aktualizacji. Sztuka polega na precyzyjnym ustaleniu zmian w zmaterializowanym widoku na podstawie zestawu wstawionych lub usuniętych krotek relacji podstawowej. Poniżej opisane są ogólne sposoby przyrostowego aktualizowania perspektyw obejmujących złączenia, selekcję, projekcję i kilka rodzajów agregacji. W kontekście modyfikacji można potraktować te techniki jak połączenie usunięcia dawnej krotki i wstawienia nowej. Załóżmy, że używana jest perspektywa  $V$  oparta na relacjach  $R$  i  $S$ . Ich instancje to  $v, r$  i  $s$ .

**Złączenia.** Jeśli perspektywa obejmuje złączenie wewnętrzne relacji  $r$  i  $s$ ,  $v_{\text{dawna}} = r \bowtie s$  oraz w  $r$  wstawiony został nowy zestaw krotek  $r_i$ , to nowa wartość perspektywy zawiera  $(r \cup r_i) \bowtie s$ . Przyrostową zmianę perspektywy można obliczyć jako  $v_{\text{nowa}} = r \bowtie s \cup r_i \bowtie s$ . Podobnie usunięcie zbioru krotek  $r_d$  z  $r$  daje nową perspektywę  $v_{\text{nowa}} = r \bowtie s - r_d \bowtie s$ . Gdy dane są dodawane lub usuwane w  $s$ , powstają podobne symetryczne wyrażenia.

**Selekcja.** Jeśli perspektywa jest zdefiniowana jako  $V = \sigma_C R$  z warunkiem selekcji  $C$ , to po wstawieniu do  $r$  zestawu krotek  $r_i$  perspektywę można zmodyfikować tak:  $v_{\text{nowa}} = v_{\text{dawna}} \cup \sigma_C r_i$ . Po usunięciu z  $r$  krotek  $r_d$  otrzymujemy:  $v_{\text{nowa}} = v_{\text{dawna}} - \sigma_C r_d$ .

**Projekcja.** W porównaniu z opisaną wcześniej strategią projekcja wymaga dodatkowej pracy. Załóżmy, że perspektywa jest zdefiniowana tak:  $V = \pi_{PŁEĆ, PENSJA} R$ , gdzie  $R$  to relacja PRACOWNIK. Przyjmijmy też, że pary  $\langle PŁEĆ, PENSJA \rangle$  z pensją 5000 występują w  $r$  w trzech różnych krotkach:  $t_5 \rightarrow \langle M, 5000 \rangle$ ,  $t_{17} \rightarrow \langle M, 5000 \rangle$  i  $t_{23} \rightarrow \langle K, 5000 \rangle$ . Perspektywa  $v$  zawiera więc dwie krotki:  $\langle M, 5000 \rangle$  i  $\langle F, 5000 \rangle$ , uzyskane z trzech krotek z  $r$ . Usunięcie z  $r$  krotki  $t_5$  nie wpływa na perspektywę. Jednak usunięcie krotki  $t_{23}$  wymaga wyeliminowania z perspektywy krotki  $\langle F, 5000 \rangle$ . Podobnie wstawienie do relacji  $r$  nowej krotki  $t_{77}$  z parą  $\langle M, 5000 \rangle$  nie wpływa na perspektywę. Tak więc aktualizacja perspektyw z projekcją wymaga

zarządzania nie tylko kolumnami tej perspektywy, ale i licznikiem. W tym przykładzie początkowe wartości licznika to 2 dla  $\langle M, 5000 \rangle$  i 1 dla  $\langle F, 5000 \rangle$ . Za każdym razem, gdy wstawienie danych do relacji podstawowej tworzy dane do perspektywy, licznik należy zwiększyć. Po usunięciu z relacji podstawowej krotki reprezentowanej w perspektywie licznik jest zmniejszany. Gdy liczba krotek w perspektywie dojdzie do zera, daną krotkę należy usunąć z perspektywy. Gdy nowo wstawiona krotka powoduje dodanie krotki do perspektywy, licznik jest ustawiany na 1. Zauważ, że w tym omówieniu przyjmujemy, iż w definicji perspektywy używana jest instrukcja `SELECT DISTINCT`, odpowiadająca operacji projekcji ( $\pi$ ). W projekcji z użyciem wielozbiorów (bez modyfikatora `DISTINCT`) też można wykorzystać liczniki. Istnieje możliwość wyświetlania krotki z perspektywy tyle razy, ile wskazuje licznik (w sytuacji, gdy perspektywa musi być traktowana jak wielozbiór).

**Część wspólna.** Jeśli perspektywa jest zdefiniowana jako  $V = R \cap S$ , to należy sprawdzić, czy wstawiona nowa krotka  $r_i$  występuje w relacji  $s$ . Gdy tak jest, krotkę należy wstawić do  $v$ . W przeciwnym razie krotka nie jest wstawiana. Po usunięciu krotki  $r_d$  należy sprawdzić, czy występuje ona w perspektywie  $v$ . Jeśli tak jest, krotka jest usuwana z perspektywy.

**Agregacja (GROUP BY).** Na potrzeby omówienia agregacji założymy, że wykonywana jest operacja `GROUP BY` na kolumnie  $G$  z relacji  $R$ , a perspektywa obejmuje człon `SELECT G funkcja_agregująca(A)`. Perspektywa jest wtedy wynikiem zastosowania funkcji agregującej do atrybutu  $A$ . Odpowiada to następującej operacji (patrz punkt 8.4.2):

$$G \bowtie_{\text{funkcja-agregująca}}(A)$$

Poniżej omawiamy kilka funkcji agregujących:

- **Zliczanie.** Na potrzeby zliczania krotek w każdej grupie po wstawieniu do  $r$  nowej krotki o wartości  $G = g_1$  należy zwiększyć odpowiedni licznik o 1 (jeśli  $g_1$  już znajduje się w perspektywie). Jeżeli perspektywa nie obejmuje jeszcze krotki o wartości  $g_1$ , do perspektywy wstawiana jest nowa krotka  $\langle g_1, 1 \rangle$ . W momencie usuwania krotki o wartości  $G = g_1$  odpowiedni licznik jest zmniejszany o 1. Jeśli licznik dla  $g_1$  spadnie do wartości 0, krotka zostanie usunięta z perspektywy.
- **Suma.** Założymy, że perspektywa zawiera parę  $(G, \text{suma}(A))$  i że istnieje licznik dla każdej grupy z perspektywy. Jeśli do relacji  $r$  wstawiana jest krotka z wartościami  $(g_1, x_1)$  kolumn  $R.G$  i  $R.A$ , a perspektywa nie zawiera wpisu dla  $g_1$ , do perspektywy dodawana jest nowa krotka  $\langle g_1, x_1 \rangle$  z licznikiem ustawionym na 1. Jeżeli w dawnej perspektywie znajduje się już wpis dla  $g_1$  —  $\langle g_1, s_1 \rangle$ , jest on modyfikowany do postaci  $\langle g_1, s_1 + x_1 \rangle$ , a powiązany licznik jest zwiększany o 1. Gdy z relacji podstawowej usuwana jest krotka z atrybutami  $R.G$  i  $R.A$  o wartości  $\langle g_1, x_1 \rangle$ , to jeśli licznik dla grupy  $g_1$  jest równy 1, krotka dla tej grupy zostanie usunięta z perspektywy. Jeżeli ten licznik ma wartość większą niż 1, zostaje zmniejszony o 1, a suma  $s_1$  jest zmniejszana do  $s_1 - x_1$ .
- **Średnia.** Wartości tej funkcji agregującej nie da się aktualizować niezależnie, bez przechowywania sumy i licznika oraz obliczania średniej jako sumy podzielonej przez liczbę elementów. Dlatego w celu obliczania nowej średniej konieczne jest przechowywanie i przyrostowe aktualizowanie sumy oraz licznika w opisany wcześniej sposób.
- **Maksimum i minimum.** Tu opiszemy tylko maksimum (minimum jest obliczane w symetryczny sposób). Dla każdej grupy aktualizowana jest kombinacja  $(g, \text{maks}(a), \text{licznik})$ , gdzie  $\text{maks}(a)$  reprezentuje maksymalną wartość atrybutu  $R.A$  w relacji



podstawowej. Jeśli wstawiana krotka ma wartość tego atrybutu mniejszą lub równą względem  $\text{maks}(a)$ , zwiększany jest jedynie licznik grupy. Jeżeli wartość jest większa niż  $\text{maks}(a)$ , trzeba ustawić maksimum w perspektywie na nową wartość i zwiększyć licznik. Gdy w trakcie usuwania krotki wartość  $R.A$  jest mniejsza niż  $\text{maks}(a)$ , zmniejszany jest jedynie licznik. Jeśli wartość  $R.A$  jest równa  $\text{maks}(a)$ , licznik należy zmniejszyć o 1, a usuwana krotka obejmuje wartość maksymalną atrybutu  $A$ . Dlatego w grupie trzeba obliczyć nowe maksimum atrybutu  $A$ , co wymaga dużo pracy. Gdy licznik przyjmie wartość 0, grupa jest usuwana z perspektywy, ponieważ usuwana krotka była ostatnią z tej grupy.

Opisaliśmy materializację przyrostową jako technikę optymalizacji w zakresie aktualizowania perspektyw. Jednak widoki zmaterializowane mogą też zmniejszać ilość pracy w niektórych zapytaniach. Przykładowo, jeśli zapytanie obejmuje komponent  $R \bowtie S$  lub  $\pi_1 R$  dostępny jako perspektywa, można tak zmodyfikować to zapytanie, aby korzystało z perspektywy. Nie trzeba wtedy wykonywać zbędnych obliczeń. Czasem zdarza się odwrotna sytuacja — w zapytaniu  $Z$  używana jest perspektywa  $V$  zmaterializowana do postaci  $v$ . Załóżmy, że ta perspektywa zawiera złączenie  $R \bowtie S$ . Jednak dla  $v$  nie istnieją struktury dostępne, np. indeksy. Przyjmijmy, że dostępne są indeksy na niektórych atrybutach (np.  $A$  z relacji  $R$ ), a zapytanie  $Z$  obejmuje warunek selekcji oparty na  $A$ . W takich sytuacjach zamiast zapytania z użyciem perspektywy korzystniejsze byłoby użycie indeksu składowej relacji. Perspektywa jest wtedy zastępowana definiującym ją zapytaniem, a relacja reprezentująca perspektywę zmaterializowaną w ogóle nie jest używana.

## 19.3. Wykorzystanie selektywności w optymalizacji kosztowej

Optymalizator zapytań nie powinien polegać wyłącznie na regułach heurystycznych lub przekształcaniu zapytań. Należy również uwzględniać oszacowania i porównywać koszty wykonania zapytania przy użyciu różnych strategii wykonania i algorytmów, wybierając strategię o *najniższym oszacowanym koszcie*. Aby takie podejście spełniło swoje zadanie, wymagane jest posiadanie prawidłowych *oszacowań kosztów*, tak aby różne strategie można było porównywać rzetelnie i w sposób realistyczny. Poza tym należy ograniczyć liczbę rozpatrywanych strategii wykonania. W przeciwnym razie zbyt wiele czasu będzie trzeba poświęcić na oszacowanie kosztów związanych z różnymi strategiami wykonania. Stąd podejście to bardziej nadaje się w przypadku **zapytań kompilowanych** (ang. *compiled queries*), gdzie optymalizacja jest przeprowadzana w czasie kompilacji i kod wynikowej strategii wykonania jest przechowywany i wykonywany bezpośrednio w czasie uruchomienia. W przypadku **zapytań interpretowanych** (ang. *interpreted queries*), gdzie cały proces przedstawiony na rysunku 18.1 występuje w czasie ich przetwarzania, pełna optymalizacja może wydłużyć czas odpowiedzi. Zapytania kompilowane wymagają bardziej rozbudowanych mechanizmów optymalizacji, natomiast w przypadku zapytań interpretowanych najlepiej sprawdzają się metody optymalizacji częściowe, zajmujące mniej czasu.



Takie podejście określa się mianem **kosztowej optymalizacji zapytań** (ang. *cost-based query optimization*)<sup>6</sup> i wykorzystuje ono tradycyjne techniki optymalizacji przeszukujące *przestrzeń rozwiązań* problemu w celu znalezienia rozwiązania, które będzie minimalizować funkcję kosztu. Funkcje kosztu używane w optymalizacji zapytań to oszacowania, a nie dokładne funkcje, więc optymalizator może wybrać strategię wykonania zapytania, która nie będzie optymalna (bezwzględnie najlepsza). W podrozdziale 19.3.1 zostaną omówione składowe koszty wykonywania zapytań. W podrozdziale 19.3.2 zostaną omówione rodzaje informacji wymaganych przez funkcje kosztu. Informacje takie są przechowywane w katalogu SZBD. W podrozdziale 19.3.3 opiszemy histogramy służące do przechowywania szczegółowych informacji o rozkładzie wartości ważnych atrybutów.

Proces podejmowania decyzji w trakcie optymalizowania zapytań nie jest prosty i związanych jest z nim wiele wyzwań. W abstrakcyjnej postaci kosztową optymalizację zapytań można opisać tak:

- Dla danego podwyrażenia w zapytaniu zastosowanie może mieć wiele reguł tworzenia wyrażeń równoważnych. Proces wprowadzania wyrażeń równoważnych przebiega kaskadowo. Nie występuje tu punkt końcowy lub ostateczna konwergencja. Trudno jest wykonać ten proces w sposób wydajny ze względu na pamięć.
- Niezbędne jest stosowanie miar ilościowych do oceny poszczególnych możliwości. Stosując wymogi dotyczące pamięci i czasu oraz sprowadzając je do wspólnej miary nazywanej kosztem, można zaprojektować metodę optymalizacji.
- Można zaprojektować odpowiednie strategie wyszukiwania, zachowujące rozwiązania o najniższym koszcie i eliminujące bardziej kosztowne możliwości.
- Zakresem optymalizacji zapytania jest zwykle blok zapytania. Różne ścieżki dostępu do tabel i indeksów, kolejność złączeń, metody złączania, metody grupowania i inne techniki oznaczają możliwości, spośród których optymalizator zapytań musi wybierać.
- W globalnej optymalizacji zapytań zakresem optymalizacji jest wiele bloków zapytań<sup>7</sup>.

### 19.3.1. Składowe koszty wykonywania zapytań

Koszt wykonania zapytania uwzględnia następujące składowe:

- (1) **Koszt dostępu do drugorzędnych mechanizmów składowania danych.** Jest to koszt przenoszenia (przy odczycie i zapisie) bloków danych między drugorzędnym mechanizmem składowania a buforami pamięci głównej. Inna nazwa to *koszt dyskowych operacji wejścia-wyjścia*. Koszt wyszukiwania rekordów w pliku zależy od typu struktur dostępu utworzonych na danym pliku, takich jak uporządkowanie, mieszanie i indeksy główne lub drugorzędne. Ponadto na koszt związany z dostępem mają wpływ czynniki takie jak to, czy bloki pliku znajdują się obok siebie w ramach tego samego cylindra dysku, czy też są rozrzucone po dysku.

---

<sup>6</sup> Podejście to zostało użyte po raz pierwszy w przypadku optymalizatora eksperymentalnego w SZBD firmy IBM o nazwie SYSTEM R.

<sup>7</sup> W tym rozdziale nie omawiamy optymalizacji globalnej w tym sensie. Szczegółowe informacje znajdziesz w Ahmed i in. (2006).

- (2) **Koszt składowania.** Jest to koszt przechowywania wszelkich plików pośrednich generowanych w ramach strategii wykonania zapytania.
- (3) **Koszt obliczeniowy.** Jest to koszt dokonywania obliczeń w pamięci na buforach danych w czasie wykonywania zapytania. Do takich operacji należy wyszukiwanie i sortowanie rekordów, scalanie rekordów w ramach złączeń oraz wykonywanie obliczeń na wartościach pól. Inna nazwa to *koszty pracy procesora*.
- (4) **Koszt zużycia pamięci.** Jest to koszt zależny od liczby buforów pamięci potrzebnych w czasie wykonywania zapytania.
- (5) **Koszt komunikacji.** Jest to koszt związany z przesłaniem zapytania i jego wyników z bazy danych do węzła lub terminalu, z którego zostało przesłane żądanie. W rozproszonych bazach danych (patrz rozdział 23.) obejmuje on też koszt przenoszenia tabel i wyników między różnymi komputerami w trakcie przetwarzania zapytania.

W przypadku dużych baz danych główny nacisk kładzie się na minimalizację kosztu dostępu do drugorzędnych mechanizmów składowania danych. Proste funkcje kosztu ignorują inne czynniki i porównują różne strategie wykonywania zapytań w kontekście liczby bloków przesyłanych między dyskiem a pamięcią główną. W przypadku mniejszych baz danych, gdzie większość danych w plikach związanych z zapytaniem może być w całości przechowywana w pamięci, nacisk kładzie się na minimalizację kosztu obliczeniowego. Z kolei w przypadku rozproszonych baz danych (patrz rozdział 23.) należy również minimalizować koszt komunikacji. Uwzględnienie wszystkich składowych kosztu w jednej (ważonej) funkcji kosztu jest niełatwym zadaniem ze względu na trudności związane z przypisaniem odpowiednich wag do składowych kosztu. Z tego powodu niektóre funkcje kosztu uwzględniają tylko jeden czynnik — dostęp do dysku. W kolejnym punkcie zostaną omówione pewne informacje potrzebne do formułowania funkcji kosztu.

### 19.3.2. Informacje z katalogu używane w funkcjach kosztu

W celu oszacowania kosztów różnych strategii wykonania należy przechowywać wszelkie potrzebne w tym celu informacje. Mogą one być przechowywane w katalogu SZBD, do którego ma dostęp optymalizator zapytań. Po pierwsze, musimy znać rozmiar każdego pliku. W przypadku pliku, którego wszystkie rekordy mają ten sam typ, potrzebna jest **liczba rekordów (krotek) ( $r$ )**, (średni) **rozmiar rekordu ( $R$ )** oraz **liczba bloków ( $b$ )** (lub ich bliskie oszacowania). Może również być potrzebny **współczynnik blokowy ( $bfr$ )** pliku. Te czynniki zostały wymienione w punkcie 18.3.4 i korzystaliśmy z nich w celu zilustrowania różnych algorytmów implementujących operacje relacyjne. Należy także przechowywać informacje o *podstawowej organizacji pliku*. Rekordy pliku mogą być *nieuporządkowane*, *uporządkowane* względem wartości atrybutu z indeksem głównym lub klastrowania albo bez niego, lub podlegać *mieszaniu* (statycznemu lub dynamicznemu) względem atrybutu klucza. Przechowuje się także informacje o wszystkich indeksach drugorzędnych oraz atrybutach indeksujących. **Liczba poziomów ( $x$ )** każdego indeksu wielopoziomowego (głównego, drugorzędnego lub klastrowania) jest potrzebna w przypadku funkcji oszacowujących liczbę operacji dostępu do bloków występujących w czasie wykonania zapytania. W przypadku niektórych funkcji kosztu potrzebna jest **liczba bloków indeksu pierwszego poziomu ( $b_1$ )**.

Kolejnym istotnym parametrem jest **liczba odrębnych wartości**  $LOW(A, R)$  atrybutu w relacji  $R$  oraz **selektywność** ( $sl$ ) atrybutu, stanowiąca odsetek rekordów spełniających warunek równościowy na atrybucie. Pozwala to na oszacowanie **liczności selekcji** ( $s = sl \cdot r$ ) atrybutu, która jest *średnią* liczbą rekordów spełniających równościowy warunek selekcji na danym atrybucie.

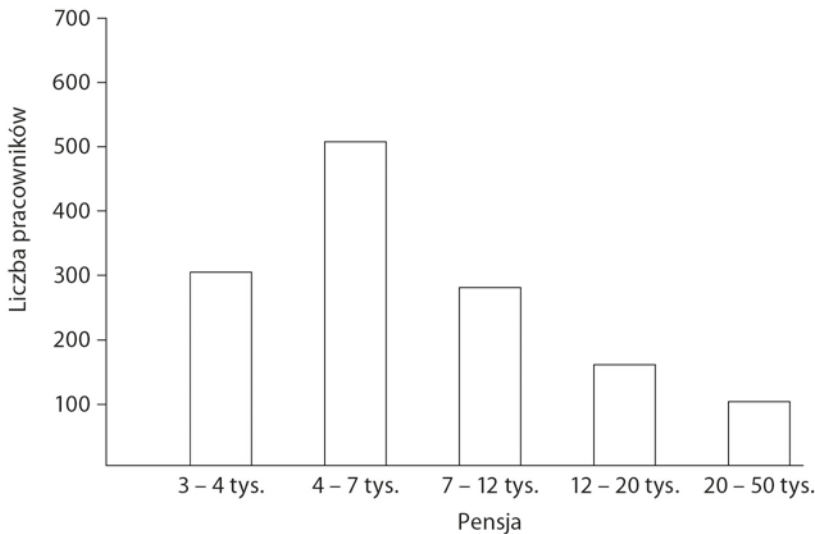
Informacje takie jak liczba poziomów indeksu są łatwe do przechowywania, ponieważ nie zmieniają się zbyt często. Jednakże inne informacje mogą ulegać częstym zmianom, na przykład liczba rekordów  $r$  w pliku zmienia się za każdym razem, gdy wstawiamy lub usuwamy rekord. Optymalizator zapytań musi posiadać w miarę dokładne, ale niekoniecznie ściśle i aktualne wartości tych parametrów, oszacowując koszt różnych strategii wykonania. Ważną pomocą w przewidywaniu wielkości wyników zapytań są dobre szacunki rozkładu wartości. W tym celu większość systemów przechowuje histogramy.

### 19.3.3. Histogramy

**Histogramy** to tabele lub struktury danych przechowywane przez SZBD i obejmujące informacje o rozkładzie danych. W większości SZBD przechowywane są histogramy dotyczące najważniejszych atrybutów. Gdy histogram jest niedostępny, najlepiej przyjąć założenie, że atrybut ma rozkład równomierny od najniższej do najwyższej wartości. Histogramy dzielą wartości atrybutu na ważne przedziały (nazywane pakietami) i przechowują łączną liczbę rekordów danej relacji należących do poszczególnych pakietów. Mogą też obejmować liczbę różnych wartości w każdym pakiecie. Czasem przyjmuje się pośrednio założenie, że różne wartości w pakiecie mają rozkład równomierny. Wszystkie wspomniane założenia są uproszczeniami i rzadko są prawdziwe. Dlatego zawsze przydatne jest tworzenie szczegółowego histogramu (z dużą liczbą pakietów). Popularnych jest kilka typów histogramów. W histogramach **o równej szerokości** zakres wartości jest podzielony na równe podzakresy. W histogramach **o równej wysokości** pakiety są tworzone w taki sposób, że każdy obejmuje mniej więcej tę samą liczbę rekordów. Histogramy o równej wysokości są uznawane za bardziej przydatne, ponieważ przechowują mniejszą liczbę często występujących wartości w jednym pakiecie i większą liczbę rzadziej pojawiających się wartości w innym. Dlatego założenie równomiernego rozkładu wartości w pakiecie jest tu spełnione z większym prawdopodobieństwem. Na rysunku 19.4 pokazany jest przykładowy histogram z informacjami o pensjach w firmie. Ten histogram dzieli zakres pensji na pięć pakietów. Mogą one odpowiadać ważnym podzakresom, z których prawdopodobnie korzystają będą zapytania, ponieważ poszczególne pakiety dotyczą pracowników różnych typów. Pokazany histogram nie ma ani równej szerokości, ani równej wysokości.

## 19.4. Funkcje kosztu dla operacji SELECT

Poniżej zostaną przedstawione funkcje kosztu dla algorytmów selekcji od S1 do S8 z punktu 18.3.1 w kontekście *liczby przesyłanych bloków* między pamięcią a dyskiem. Algorytm S9 obejmuje część wspólną wskaźników na rekordy pobrane w inny sposób (np. za pomocą algorytmu S6), dlatego funkcja kosztu dla tego algorytmu jest oparta na



Rysunek 19.4. Histogram wynagrodzeń w relacji PRACOWNIK

kosztach z algorytmu S6. Pokazane funkcje kosztu stanowią oszacowania, które pomijają czas obliczeniowy, koszt składowania i inne czynniki. Przypomnijmy notację stosowaną od tego miejsca we wzorach:

$C_{Si}$  — koszt dla metody  $Si$  mierzony w dostęпах do bloków.

$r_X$  — liczba rekordów (krotek) w relacji  $X$ .

$b_X$  — liczba bloków zajmowanych przez relację  $X$  (stosujemy też zapis  $b$ ).

$bfr_X$  — współczynnik blokowy (czyli liczba rekordów na blok) w relacji  $X$ .

$sl_A$  — selektywność atrybutu  $A$  dla danego warunku.

$sA$  — licznosc selekcji dla pobieranego atrybutu ( $=sl_A \cdot r$ ).

$sA$  — liczba poziomów indeksu na atrybucie  $A$ .

$b_{1A}$  — liczba bloków pierwszego poziomu indeksu na atrybucie  $A$ .

$LOW(A, X)$  — liczba różnych wartości atrybutu  $A$  w relacji  $X$ .

*Uwaga:* gdy stosujemy opisaną notację we wzorach, pomijamy nazwy relacji i atrybutów, jeśli są one oczywiste.

- **S1. Wyszukiwanie liniowe (metoda siłowa).** Przeszukujemy wszystkie bloki pliku w celu pobrania wszystkich rekordów spełniających warunek selekcji. Stąd  $C_{S1a} = b$ . W przypadku warunku równościowego na kluczu średnio przeszukiwana jest tylko połowa bloków pliku zanim zostanie znaleziony rekord, więc  $C_{S1b} = (b/2)$ , jeżeli rekord zostanie znaleziony. Jeśli jednak żaden rekord nie spełnia warunku, to  $C_{S1b} = b$ .
- **S2. Wyszukiwanie binarne.** W tym przypadku uzyskujemy dostęp do w przybliżeniu  $C_{S2} = \log_2 b + \lceil (s/bfr) \rceil - 1$  bloków pliku. Wartość ta redukuje się do  $\log_2 b$ , jeżeli warunek równościowy dotyczy atrybutu unikatowego (klucza), ponieważ wówczas  $s = 1$ .

- **S3a. Użycie indeksu głównego w celu pobrania jednego rekordu.** W przypadku indeksu głównego pobieramy o jeden blok więcej niż wynosi liczba poziomów indeksu. Stąd koszt wynosi o jeden blok więcej niż liczba poziomów indeksu:  
 $C_{S3a} = x + 1$ .
- **S3b. Użycie klucza mieszającego w celu pobrania jednego rekordu.** W przypadku mieszania w większości sytuacji potrzebny jest dostęp tylko do jednego bloku. Funkcja kosztu ma postać  $C_{S3b} = 1$  dla mieszania statycznego lub liniowego oraz 2 dla mieszania rozszerzalnego (patrz podrozdział 16.8).
- **S4. Użycie indeksu uporządkowania w celu pobrania wielu rekordów.** Jeżeli warunkiem porównania jest relacja  $>$ ,  $>=$ ,  $<$  lub  $<=$  na polu klucza z indeksem uporządkowania, warunek spełnia około połowa rekordów. Daje to funkcję kosztu  $C_{S4} = x + (b/2)$ . Jest to bardzo zgrubne oszacowanie i choć w przeciętnym przypadku może okazać się poprawne, czasem bywa bardzo niedokładne. Bardziej precyzyjne szacunki są możliwe, jeśli zapisany jest rozkład rekordów w postaci histogramu.
- **S5. Użycie indeksu klastrowania w celu pobrania wielu rekordów.** Na każdym poziomie indeksu potrzebny jest dostęp do jednego bloku, aby uzyskać adres pierwszego należącego do klastra bloku pliku z dysku. Dla danego warunku równościowego na atrybucie indeksowania  $s$  rekordów spełnia ten warunek, gdzie  $s$  jest licznoscią selekcji atrybutu indeksującego. Oznacza to, że dostęp jest uzyskiwany do  $\lceil (s/bfr) \rceil$  bloków pliku, co daje  $C_{S5} = x + \lceil (s/bfr) \rceil$ .
- **S6. Użycie drugorzędnego indeksu (B+-drzewa).** Gdy używany jest drugorzędny indeks na atrybucie klucza (unikatowym) i równościowy warunek selekcji (czyli  $<\text{atrybut}=\text{wartość}>$ ), koszt wynosi  $x+1$  dostępu do bloku. Dla drugorzędnego indeksu na atrybucie niebędącym kluczem (nieunikatowym)  $s$  rekordów spełnia warunek równościowy, gdzie  $s$  to liczność selekcji dla atrybutu indeksującego. Jednakże, ze względu na fakt, że indeks nie jest indeksem klastrowania, każdy rekord może znajdować się w innym bloku, więc oszacowanie kosztu (dla najgorszego przypadku) wynosi  $C_{S6a} = x + 1 + s$ . Dodatkowe 1 związane jest z blokiem zawierającym wskaźniki na rekordy po przeszukaniu indeksu (patrz rysunek 17.5). W zapytaniach zakresowych jeżeli warunkiem porównania jest  $>$ ,  $>=$ ,  $<$  lub  $<=$  i zakłada się, że warunek spełnia połowa rekordów pliku, to (bardzo zgrubnie) uzyskiwany jest dostęp do połowy bloków pierwszego poziomu indeksu oraz do połowy rekordów pliku poprzez indeks. Oszacowanie kosztu dla takiego przypadku wynosi w przybliżeniu  $C_{S6b} = x + (b_{I1}/2) + (r/2)$ . Składnik  $r/2$  można uściślić, jeżeli są dostępne lepsze oszacowania selektywności w postaci histogramu. Koszt  $C_{S6b}$  może być bardzo wysoki. Dla warunku zakresowego takiego jak  $v1 < A < v2$  liczność selekcji,  $s$ , trzeba ustalić według histogramu lub w domyślny sposób, przy założeniu rozkładu równomiernego. Koszt jest wtedy obliczany na podstawie tego, czy  $A$  jest kluczem oraz czy istnieje indeks w postaci B+-drzewa na  $A$ . Obliczenia dla różnych warunków pozostawiamy jako ćwiczenie dla Czytelników.
- **S7. Wybór koniunktywny.** Można użyć metody S1 lub jednej z omówionych powyżej metod od S2 do S6. W tym drugim przypadku wykorzystujemy jeden warunek w celu pobrania rekordów, a następnie sprawdzamy w buforze pamięci, czy każdy pobrany rekord spełnia pozostałe warunki w koniunkcji. Jeśli istnieje wiele indeksów, wyszukiwanie z użyciem każdego z nich może dać zestaw wskaźników na rekordy (identyfikatorów rekordów) w buforach pamięci głównej.

Część wspólną zbiorów wskaźników rekordów (opisaną w S9) można obliczyć w pamięci głównej, a następnie pobrać wynikowe rekordy na podstawie ich identyfikatorów.

- **S8. Wybór koniunktywny przy użyciu indeksu złożonego.** Tak jak w przypadku S3a, S5 lub S6a, w zależności od rodzaju indeksu.
- **S9. Wybór przy użyciu indeksu bitmapowego** (patrz punkt 17.5.2). Jeśli możliwe jest ograniczenie selekcji do zbioru warunków równościowych, z których każdy to porównanie atrybutu z wartością (np.  $A=\{7, 13, 17, 55\}$ ), to potrzebny jest dostęp do wektora bitowego dla każdej wartości. Taki wektor zajmuje  $r$  bitów, czyli  $r/8$  bajtów. W jednym bloku można zmieścić wiele wektorów bitowych. Jeśli  $s$  rekordów spełnia warunek, potrzebny jest dostęp do  $s$  bloków z rekordami danych.
- **S10. Wybór przy użyciu indeksu funkcyjnego** (patrz punkt 17.5.3). To podejście jest podobne do S6, przy czym indeks jest oparty na funkcji zależnej od wielu atrybutów. Jeśli ta funkcja występuje w klauzuli SELECT, można wykorzystać powiązany indeks.

**Przykład użycia funkcji kosztu.** W przypadku optymalizatora zapytań często wylicza się różne dostępne strategie wykonania zapytania i szacuje koszt związany z każdą z nich. W celu wydajnego znajdowania optymalnych oszacowań kosztu można używać techniki optymalizacji, takiej jak programowanie dynamiczne, nie musząc rozpatrywać wszystkich możliwych strategii wykonania. **Programowanie dynamiczne** to technika optymalizacji<sup>8</sup>, w której podproblemy są rozwiązywane tylko raz. Tę technikę można stosować, gdy problem da się podzielić na podproblemy, które same składają się z podproblemów. Do podejścia wykorzystującego programowanie dynamiczne wrócimy przy omawianiu kolejności złączeń w punkcie 19.5.5. Nie będziemy tu omawiać algorytmów optymalizacji, a raczej użyjemy prostego przykładu w celu zilustrowania, w jaki sposób mogą być wykorzystywane oszacowania kosztu.

### 19.4.1. Przykład optymalizacji selekcji na podstawie wzorów szacowania kosztów

Założmy, że plik PRACOWNIK z rysunku 5.5 zawiera  $r_P = 10\,000$  rekordów przechowywanych w  $b_P = 2000$  blokach dyskowych o współczynniku blokowym  $bfr_P = 5$  rekordów na blok oraz następujących ścieżkach dostępu:

- (1) Indeks klastrowania na atrybucie PENSJA liczący  $x_{PENSJA} = 3$  poziomy i charakteryzujący się średnią licznością selekcji  $s_{PENSJA} = 20$ . Daje to selektywność równą  $sl_{PENSJA} = 20/10000 = 0,002$ .
- (2) Drugorzędny indeks na atrybucie klucza PESEL, gdzie  $x_{PESEL} = 4$  ( $s_{PESEL} = 1$ ,  $sl_{PESEL} = 0,0001$ ).
- (3) Drugorzędny indeks na atrybucie niebędącym kluczem NRDZ, gdzie  $x_{NRDZ} = 2$  i liczba bloków pierwszego poziomu indeksu wynosi  $b_{I1NRDZ} = 4$ . Istnieje  $LOW(NRDZ, PRACOWNIK) = 125$  odrębnych wartości atrybutu NRDZ, więc jego selektywność

<sup>8</sup> Szczegółowe omówienie programowania dynamicznego jako techniki optymalizacji znajdziesz w podręcznikach poświęconych algorytmom (np. Corman i in., 2003).



jest równa  $s_{NRDZ} = (1/LOW(NRDZ, PRACOWNIK)) = 0,008$ , a liczność selekcji wynosi  $s_{NRDZ} = (r_P \cdot s_{NRDZ}) = (r_P/LOW(NRDZ, PRACOWNIK)) = 80$ .

- (4) Drugorzędny indeks na atrybucie PŁEĆ, gdzie  $x_{PŁEĆ} = 1$ . Istnieją  $LOW(PŁEĆ, PRACOWNIK) = 2$  wartości atrybutu PŁEĆ, więc średnia liczność selekcji wynosi  $s_{PŁEĆ} = (r_P/LOW(PŁEĆ, PRACOWNIK)) = 5000$ . Warto zauważyć, że jeśli odsetek kobiet i mężczyzn nie jest w przybliżeniu równy, przydatny byłby tu histogram z procentami pracowników płci męskiej i żeńskiej.

Użycie funkcji kosztu zilustrujemy przy użyciu następujących przykładów:

(OP1):  $\sigma_{PESEL='65010912345'}(PRACOWNIK)$

(OP2):  $\sigma_{NRDZ>5}(PRACOWNIK)$

(OP3):  $\sigma_{NRDZ=5}(PRACOWNIK)$

(OP4):  $\sigma_{NRDZ=5 \text{ AND } PENSJA>3000 \text{ AND } PŁEĆ='K'}(PRACOWNIK)$

Koszt siłowego rozwiązania S1 (przeszukiwania liniowego lub przeglądu pliku) oszacujemy jako  $C_{S1a} = b_P = 2000$  (dla selekcji względem atrybutu niebędącego kluczem) lub  $C_{S1b} = (b_P/2) = 1000$  (średni koszt dla selekcji względem atrybutu klucza). W przypadku operacji OP1 możemy użyć albo metody S1, albo metody S6a. Oszacowanie kosztu dla tej drugiej wynosi  $C_{S6a} = x_{PESEL} + 1 = 4 + 1 = 5$  i stanowi ona lepszy wybór niż metoda S1, której średni koszt to  $C_{S1b} = 1000$ . W przypadku operacji OP2 możemy użyć albo metody S1 (jej szacowany koszt to  $C_{S1a} = 2000$ ), albo metody S6b (o szacowanym koszcie  $C_{S6b} = x_{NRDZ} + (b_{I1NRDZ}/2) + (r_P/2) = 2 + (4/2) + (10000/2) = 5004$ ), tak więc wybieramy podejście siłowe dla operacji OP2. W przypadku operacji OP3 możemy użyć metody S1 (o szacowanym koszcie  $C_{S1a} = 2000$ ) lub metody S6a (o szacowanym koszcie  $C_{S6a} = x_{NRDZ} + s_{NRDZ} = 2 + 80 = 82$ ), więc wybieramy metodę S6a.

Wreszcie rozpatrujemy metodę OP4, która posiada koniunktywny warunek selekcji. Musimy oszacować koszt użycia każdej z trzech składowych warunku selekcji związany z pobieraniem rekordów oraz wykorzystania podejścia siłowego. W tym ostatnim przypadku oszacowanie kosztu to  $C_{S1a} = 2000$ . Użycie jako pierwszego warunku ( $NRDZ = 5$ ) daje oszacowanie  $C_{S6a} = 82$ . Użycie jako pierwszego warunku ( $PENSJA > 3000$ ) daje oszacowanie kosztu  $C_{S4} = x_{PENSJA} + (b_P/2) = 3 + (2000/2) = 1003$ . Użycie jako pierwszego warunku ( $PŁEĆ = 'K'$ ) daje oszacowanie kosztu  $C_{S6a} = x_{PŁEĆ} + s_{PŁEĆ} = 1 + 5000 = 5001$ . Optymalizator wybiera więc metodę S6a opartą na indeksie drugorzędnym na atrybucie NRDZ, ponieważ wiąże się ona z najniższym oszacowaniem kosztu. Warunek ( $NRDZ = 5$ ) jest używany w celu pobrania rekordów, zaś pozostała część warunku koniunktywnego ( $PENSJA > 3000$  AND  $PŁEĆ = 'K'$ ) jest sprawdzana pod względem każdego wybranego rekordu po ich pobraniu do pamięci. Do wyniku operacji dołączane są tylko rekordy spełniające dodatkowe warunki. Przyjrzyj się warunkowi  $NRDZ=5$  z operacji OP3. Atrybut NRDZ ma 125 wartości, dlatego odpowiedni będzie indeks oparty na B+-drzewie. Gdyby jednak użyć atrybutu KOD relacji PRACOWNIK i warunku  $KOD=30332$ , to jeśli w bazie występuje tylko pięć kodów pocztowych, można zastosować indeksy bitmapowe do sprawdzenia, czy rekordy są zgodne z warunkiem. Przy założeniu rozkładu jednorodnego  $s_{KOD}=2000$ . To oznacza koszt 2000 przy zastosowaniu indeksów bitmapowych.



## 19.5. Przykłady funkcji kosztu dla operacji JOIN

W celu określania wystarczająco dokładnych funkcji kosztu dla operacji JOIN musimy posiadać dostęp do oszacowań rozmiaru (liczby krotek) pliku *wynikającego* z wykonania operacji JOIN. Informacje takie są zwykle przechowywane jako współczynnik rozmiaru (liczby krotek) wynikowego pliku złączenia do rozmiaru pliku iloczynu kartezjańskiego, jeżeli oba zostaną wykorzystane względem tych samych plików wejściowych, i określa się go mianem **selektywności złączenia** (ang. *join selectivity*) *js*. Jeżeli liczbę krotek relacji *R* oznaczymy jako  $|R|$ , to otrzymamy:

$$js = |(R \bowtie S)| / |(R \times S)| = |(R \bowtie S)| / (|R| \cdot |S|)$$

Jeżeli nie występuje żaden warunek złączenia *c*, to  $js = 1$  i złączenie staje się równoważne operacji CARTESIAN PRODUCT. Jeżeli warunków złączenia nie spełniają żadne krotki relacji, to  $js = 0$ . Ogólnie rzecz biorąc,  $0 \leq js \leq 1$ . W przypadku złączenia, w którym warunek *c* jest porównaniem równościowym  $R.A = S.B$ , otrzymujemy dwa następujące przypadki szczególne:

- (1) Jeżeli *A* jest kluczem relacji *R*, to  $|(R \bowtie_c S)| \leq |S|$ , więc  $js \leq (1/|R|)$ . Jest tak, ponieważ każdy rekord z pliku *S* jest złączany z najwyżej jednym rekordem z pliku *R*, gdyż *A* jest kluczem relacji *R*. Specjalny scenariusz ma miejsce, gdy atrybut *B* jest *kluczem obcym* relacji *S* powiązanym z *kluczem głównym* *A* relacji *R*. Ponadto jeśli dla klucza obcego *B* nie obowiązuje ograniczenie NOT NULL, to  $js = (1/|R|)$ , a plik wynikowy złączenia będzie zawierał  $|S|$  rekordów.
- (2) Jeżeli *B* jest kluczem relacji *S*, to  $|(R \bowtie_c S)| \leq |R|$ , więc  $js \leq (1/|S|)$ .

Z tego wynika **prosty wzór** do obliczania selektywności złączeń:

$$js = 1 / \text{MAKS}(\text{LOW}(A, R), \text{LOW}(B, S))$$

Posiadanie oszacowania selektywności złączenia dla często występujących warunków złączenia pozwala optymalizatorowi zapytań na oszacowanie rozmiaru pliku wynikowego po wykonaniu operacji złączenia, dla danych rozmiarów dwóch plików wejściowych. Ten rozmiar to **liczność złączenia** (ang. *join cardinality* — *jc*).

$$jc = |(R \bowtie_c S)| = js \cdot |R| \cdot |S|$$

Możemy teraz przedstawić pewne przykładowe *przybliżone* funkcje kosztu służące do szacowania kosztu niektórych algorytmów złączeń podanych w podrozdziale 18.4. Operacje złączenia mają postać

$$R \bowtie_A =_B S$$

gdzie *A* i *B* są zgodnymi w ramach dziedziny atrybutami relacji, odpowiednio, *R* i *S*. Przyjmujemy, że relacja *R* posiada  $b_R$  bloków, zaś relacja *S* —  $b_S$  bloków.

- **J1. Złączenie pętli zagnieżdżonych.** Załóżmy, że jako pętli zewnętrznej używamy relacji *R*. Wówczas otrzymujemy przedstawioną poniżej funkcję kosztu służącą do oszacowania liczby operacji dostępu do bloków dla tej metody, zakładając występowanie *trzech buforów pamięci*. Zakładamy, że współczynnik blokowy pliku wynikowego wynosi  $bfr_{RS}$  oraz że znamy selektywność złączenia:

$$C_{J1} = b_R + (b_R \cdot b_S) + ((js \cdot |R| \cdot |S|) / bfr_{RS})$$

Ostatni składnik powyższego wzoru jest kosztem zapisania pliku wynikowego na dysku. Wzór ten można zmodyfikować tak, aby uwzględnić różne liczby buforów w pamięci, zgodnie z dyskusją z podrozdziału 19.4. Jeśli w pamięci głównej dostępnych jest  $n_B$  bloków buforów na wykonanie złączenia, wzór na koszt to:

$$C_{J1} = b_R + \lceil (b_R/n_B - 2) \rceil b_S + ((j_S \cdot |R| \cdot |S|)/bfr_{RS})$$

- **J2. Złączenie z pętlą pojedynczą (przy użyciu struktury dostępowej w celu pobrania pasujących rekordów).** Jeżeli na atrybucie złączenia  $B$  relacji  $S$  istnieje indeks o  $x_B$  poziomach, to możemy pobrać każdy rekord  $s$  z relacji  $R$ , a następnie użyć tego indeksu w celu pobrania wszystkich pasujących rekordów  $t$  relacji  $S$ , które spełniają warunek  $t[B] = s[A]$ . Koszt zależy od rodzaju indeksu. W przypadku indeksu drugorzędowego, gdzie  $s_B$  jest liczącością selekcji dla atrybutu złączenia  $B$  relacji  $S^9$ , otrzymujemy:

$$C_{J2a} = b_R + (|R| \cdot (x_B + s_B)) + ((j_S \cdot |R| \cdot |S|)/bfr_{RS})$$

W przypadku indeksu klastrowania, gdzie  $s_B$  jest liczącością selekcji atrybutu  $B$ , otrzymujemy:

$$C_{J2b} = b_R + (|R| \cdot (x_B + (s_B/bfr_B))) + ((j_S \cdot |R| \cdot |S|)/bfr_{RS})$$

W przypadku indeksu głównego, otrzymujemy:

$$C_{J2c} = b_R + (|R| \cdot (x_B + 1)) + ((j_S \cdot |R| \cdot |S|)/bfr_{RS})$$

Jeżeli na jednym z atrybutów złączenia istnieje **klucz mieszający**, na przykład na atrybucie  $B$  relacji  $S$ , to otrzymujemy:

$$C_{J2d} = b_R + (|R| \cdot h) + ((j_S \cdot |R| \cdot |S|)/bfr_{RS})$$

gdzie  $h \geq 1$  jest średnią liczbą operacji dostępu do bloków w celu pobrania rekordu dla danej wartości klucza mieszania. Zwykle szacuje się, że  $h$  jest równe 1 dla mieszania statycznego i liniowego oraz 2 dla mieszania rozszerzalnego. Są to optymistyczne szacunki. W praktyce  $h$  wynosi zwykle od 1,2 do 1,5.

- **J3. Złączenie sortująco-scalające:** Jeżeli pliki są już posortowane względem wartości atrybutów złączenia, funkcja kosztu dla tej metody ma postać:

$$C_{J3a} = b_R + b_S + ((j_S \cdot |R| \cdot |S|)/bfr_{RS})$$

Jeżeli zachodzi konieczność posortowania plików, należy uwzględnić koszt takiego sortowania. W tym celu można wykorzystać wzory z podrozdziału 18.2.

- **J4. Partycjonowane złączenie mieszające (lub po prostu złączenie mieszające):** Rekordy z plików  $R$  i  $S$  są dzielone na mniejsze pliki. Podział każdego pliku odbywa się z użyciem tej samej funkcji mieszającej  $h$  dla atrybutu złączania  $A$  z relacji  $R$  (przy podziale pliku  $R$ ) i  $B$  z relacji  $S$  (przy podziale pliku  $S$ ). W podrozdziale 18.4 pokazano, że koszt takiego złączenia wynosi w przybliżeniu:

$$C_{J4} = 3 \cdot (b_R + b_S) + ((j_S \cdot |R| \cdot |S|)/bfr_{RS})$$

<sup>9</sup> Liczącość selekcji została zdefiniowana jako średnia liczba rekordów spełniających warunek równościowy na atrybucie, która jest średnią liczbą rekordów posiadających tę samą wartość dla atrybutu, a stąd złączanych w ramach pojedynczego rekordu w innym pliku.

### 19.5.1. Selektywność i liczność złączeń częściowych i antyzłączeń

Omówimy tu te dwie ważne operacje stosowane do eliminowania zagnieżdżenia w niektórych zapytaniach. W podrozdziale 18.1 pokazaliśmy przykładowe podzapytania przekształcane w te operacje. Celem stosowania tych operacji jest uniknięcie niepotrzebnego wysiłku związanego z kompletnym porównywaniem dwóch tabel parami na podstawie warunku złączenia. Rozważmy teraz selektywność i liczność dwóch wymienionych typów złączeń.

#### Złączenie częściowe

```
SELECT COUNT(*)
FROM T1
WHERE T1.X IN (SELECT T2.Y
               FROM T2);
```

Wyeliminowanie zagnieżdżenia z tego zapytania prowadzi do złączenia częściowego. W poniższym zapytaniu używamy niestandardowej notacji złączeń częściowych —  $S=$ .

```
SELECT COUNT(*)
FROM T1, T2
WHERE T1.X  $S=$  T2.Y;
```

Selektywność tego złączenia częściowego wynosi:

$$js = \min(1, \text{LOW}(Y, T2) / \text{LOW}(X, T1))$$

Liczność przedstawionego złączenia częściowego jest równa:

$$jc = |T1| \cdot js$$

**Antyzłączenie.** Przyjrzyj się następującemu zapytaniu:

```
SELECT COUNT (*)
FROM T1
WHERE T1.X NOT IN (SELECT T2.Y
                  FROM T2);
```

Wyeliminowanie zagnieżdżenia w tym zapytaniu prowadzi do powstania antyzłączenia<sup>10</sup>. W poniższym zapytaniu używamy niestandardowej notacji antyzłączeń —  $A=$ .

```
SELECT COUNT(*)
FROM T1, T2
WHERE T1.X  $A=$  T2.Y;
```

Selektywność tego antyzłączenia wynosi:

$$js = 1 - \min(1, \text{LOW}(T2.y) / \text{LOW}(T1.x))$$

Liczność przedstawionego złączenia częściowego jest równa:

$$jc = |T1| \cdot js$$

---

<sup>10</sup> Warto zauważyć, że aby zastosować antyzłączenie w podzapytaniu NOT IN, oba atrybuty złączenia —  $T1.X$  i  $T2.Y$  — muszą mieć wartości różne od NULL. Szczegółowe omówienie tej kwestii znajdziesz w Bellamkonda i in. (2009).

## 19.5.2. Przykład optymalizacji złączenia na podstawie funkcji kosztu

Założmy, że posiadamy plik PRACOWNIK opisany w przykładzie z poprzedniego podrozdziału i przyjmujemy, że plik DZIAŁ z rysunku 5.5 zawiera  $r_D = 125$  rekordów przechowywanych w  $b_D = 13$  blokach dyskowych. Rozważmy operacje złączeniowe:

(OP6): PRACOWNIK ⋈<sub>NRDZ=NUMERDZ</sub> DZIAŁ

(OP7): DZIAŁ ⋈<sub>PESELKIEROWNIKA=PESEL</sub> PRACOWNIK

Założmy, że posiadamy indeks główny na polu NUMERDZ pliku DZIAŁ z  $x_{\text{NUMERDZ}} = 1$  poziomem oraz indeks drugorzędny na polu PESELKIEROWNIKA pliku DZIAŁ o liczności selekcji  $s_{\text{PESELKIEROWNIKA}} = 1$  oraz  $x_{\text{PESELKIEROWNIKA}} = 2$  poziomach. Zakładamy, że selektywność złączenia dla operacji OP6 wynosi  $j_{\text{OP6}} = (1/|DZIAŁ|) = 1/125^{11}$ , ponieważ NUMERDZ jest kluczem dla DZIAŁ. Zakładamy również, że współczynnik blokowy dla wynikowego pliku złączenia  $bfr_{PD} = 4$  rekordy na blok. Możemy oszacować koszt w najgorszym przypadku operacji JOIN OP6, używając odpowiednich metod J1 i J2 w sposób następujący:

- (1) Wykorzystując metodę J1 z plikiem PRACOWNIK jako pętlą zewnętrzną:

$$C_{J1} = b_P + (b_P \cdot b_D) + ((j_{\text{OP6}} \cdot r_P \cdot r_D) / bfr_{PD}) = \\ 2000 + (2000 \cdot 13) + (((1/125) \cdot 10\,000 \cdot 125) / 4) = 30\,500$$

- (2) Wykorzystując metodę J1 z plikiem DZIAŁ jako pętlą zewnętrzną:

$$C_{J1} = b_D + (b_P \cdot b_D) + ((j_{\text{OP6}} \cdot r_P \cdot r_D) / bfr_{PD}) = 13 + (13 \cdot 2000) + (((1/125) \cdot 10\,000 \cdot 125) / 4) \\ = 28\,513$$

- (3) Wykorzystując metodę J2 z plikiem PRACOWNIK jako pętlą zewnętrzną:

$$C_{J2c} = b_P + (r_P \cdot (x_{\text{NUMERDZ}} + 1)) + ((j_{\text{OP6}} \cdot r_P \cdot r_D) / bfr_{PD}) = \\ 2000 + (10\,000 \cdot 2) + (((1/125) \cdot 10\,000 \cdot 125) / 4) \\ = 24\,500$$

- (4) Wykorzystując metodę J2 z plikiem DZIAŁ jako pętlą zewnętrzną:

$$C_{J2a} = b_D + (r_D \cdot (x_{\text{NRDZ}} + s_{\text{NRDZ}})) + ((j_{\text{OP6}} \cdot r_P \cdot r_D) / bfr_{PD}) = \\ 13 + (125 \cdot (2 + 80)) + (((1/125) \cdot 10\,000 \cdot 125) / 4) \\ = 12\,763$$

- (5) Wykorzystując metodę J4:

$$C_{J4} = 3 \cdot (b_D + b_P) + ((j_{\text{OP6}} \cdot r_P \cdot r_D) / bfr_{PD}) = 3 \cdot (13 + 2000) + 2500 = 8539$$

Przypadek 5. charakteryzuje się najniższym oszacowaniem kosztu, dlatego właśnie on zostanie wybrany. Należy zauważyć, że gdyby dla wykonania złączenia było dostępnych 15 (lub więcej) buforów pamięci zamiast tylko trzech, 13 z nich można by wykorzystać w celu przechowania całej relacji DZIAŁ (relacji z pętli zewnętrznej) w pamięci, jeden jako bufor wyniku, a jeden do przechowywania po jednym bloku pliku PRACOWNIK (relacji z pętli wewnętrznej). Koszt dla przypadku 2. zostałby znacznie zredukowany do  $b_P + b_D + ((j_{\text{OP6}} \cdot r_P \cdot r_D) / bfr_{PD})$ , czyli 4513, zgodnie z omówieniem zawartym w podrozdziale 18.4.

<sup>11</sup> Warto zauważyć podobieństwo z innym wzorem:  $1/\text{maks}(\text{LOW}(\text{NRDZ}, \text{PRACOWNIK}), \text{LOW}(\text{NUMERDZ}, \text{DZIAŁ})) = 1/\text{maks}(125, 125) = 1/125$ .

Gdyby dostępna była inna liczba buforów pamięci, np.  $n_B = 10$ , koszt dla przypadku 2. byłby równy tyle co w obliczeniach poniżej (także ta wartość zapewnia wyższą wydajność niż dla przypadku 4.):

$$\begin{aligned} C_{J1} &= b_R + \lceil (b_D/n_B - 2) \rceil b_E + ((j_s \cdot |R| \cdot |S|)/bfr_{RS}) \\ &= 13 + (\lceil 13/8 \rceil 2000) + ((1/125) \cdot 10000 \cdot 125/4) \\ &= 13 + (2 \cdot 2000) + 2500 = 6513 \end{aligned}$$

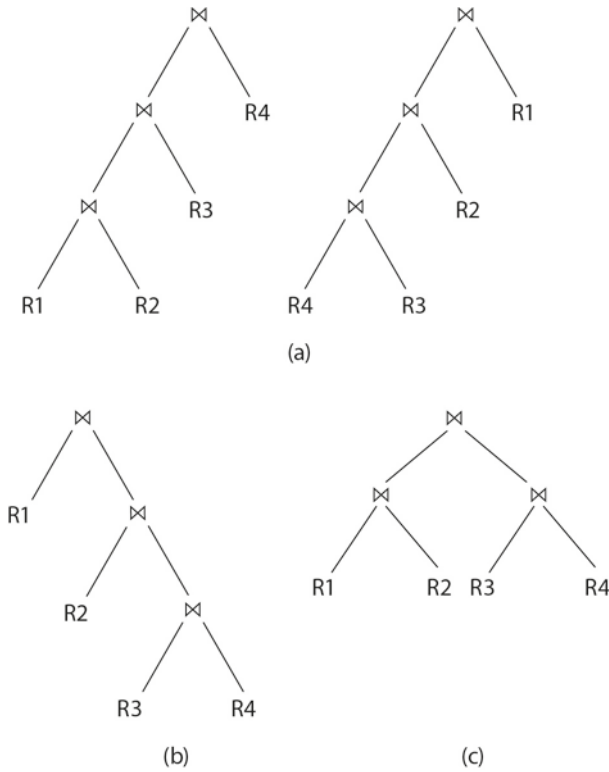
W ramach ćwiczenia Czytelnik powinien dokonać podobnej analizy dla operacji OP7.

### 19.5.3. Zapytania dotyczące wielu relacji i porządkowanie złączeń

Wśród reguł przekształceń algebraicznych z podrozdziału 19.1.2 występuje reguła przemienności oraz reguła łączności dla operacji złączenia. Dzięki tym regułom można utworzyć wiele równoważnych wyrażeń złączeniowych. W rezultacie liczba alternatywnych drzew zapytań gwałtownie wzrasta w miarę zwiększania liczby złączeń występujących w zapytaniu. Ogólnie rzecz biorąc, zapytanie złączające  $n$  relacji posiada  $n-1$  operacji złączenia, a więc może posiadać dużą liczbę różnych porządków złączeń. Dla bloku zapytania obejmującego  $n$  relacji jest  $n!$  kolejności złączeń (ta liczba uwzględnia iloczyny kartezyjańskie). Oszacowanie kosztu każdego możliwego drzewa złączeń dla zapytania o dużej liczbie złączeń wymaga poświęcenia znacznej ilości czasu ze strony optymalizatora zapytań. Dlatego też potrzebne jest opracowanie pewnej metody „odsiewania” możliwych drzew zapytań. Optymalizatory zapytań zwykle ograniczają strukturę drzewa zapytania do postaci drzew lewostronnie zagłębionych (lub prawostronnie zagłębionych). Drzewo **lewostronnie zagłębione** (ang. *left-deep tree*) jest drzewem binarnym, w którego przypadku prawy potomek każdego wierzchołka niebędącego wierzchołkiem liścia zawsze stanowi relację podstawową. Optymalizator wybiera określone drzewa lewostronnie zagłębione o najniższym szacowanym koszcie. Na rysunku 19.5(a) przedstawiono dwa przykłady drzew lewostronnie zagłębionych (warto zauważyć, że drzewa z rysunku 19.2 także są drzewami lewostronnie zagłębionymi). **Drzewo prawostronnie zagłębione** to drzewo binarne, w którym każdy lewy potomek każdego wierzchołka liścia odpowiada relacji podstawowej (rysunek 19.5(b)).

**Drzewo krzaczaste** to drzewo binarne, w którym lewy lub prawy potomek wierzchołka wewnętrznego może być wierzchołkiem wewnętrznym. Na rysunku 19.5(b) pokazano drzewo prawostronnie zagłębione, a na rysunku 19.5(c) znajduje się drzewo krzaczaste z czterema relacjami podstawowymi. Większość optymalizatorów zapytań preferuje drzewa lewostronnie zagłębione jako drzewa złączeń i wybiera jedną z  $n!$  możliwych kolejności złączeń, gdzie  $n$  to liczba relacji. Kolejność złączeń omówimy szczegółowo w punktach 19.5.4 i 19.5.5. Drzewo lewostronnie zagłębione ma jeden kształt, a możliwych kolejności złączania  $N$  tabel w takim drzewie jest  $N!$ . Z kolei kształt drzewa krzaczastego zależy od pokazanej poniżej funkcji rekurencyjnej, gdzie  $S(n)$  jest zdefiniowane tak:  $S(1) = 1$ .

$$S(n) = \sum_{i=1}^{n-1} S(i) \cdot S(n-1)$$



RYSUNEK 19.5. (a) Dwa drzewa zapytań lewostronnie zagłębione. (b) Drzewo zapytań prawostronnie zagłębione. (c) Krzaczaste drzewo zapytań

Ten rekurencyjny wzór na  $S(n)$  można objaśnić w następujący sposób:  $i$  oznacza liczbę liści w lewym poddrzewie ( $i$  należy do przedziału od 1 do  $N - 1$ ); liście te można uporządkować na  $S(i)$  sposobów. Podobnie pozostałych  $N - i$  liści w prawym poddrzewie można uporządkować na  $S(N - i)$  sposobów. Liczba permutacji w drzewie krzaczastym wynosi:

$$P(n) = n! \cdot S(n) = (2n - 2)! / (n - 1)!$$

W tabeli 19.1 pokazano liczbę możliwych drzew zagłębionych lewostronnie (lub prawostronnie) i drzew krzaczastych dla złączeń obejmujących do siedmiu relacji.

TABELA 19.1. Liczba permutacji drzew zagłębionych lewostronnie i drzew krzaczastych dla N relacji

Liczba relacji (N)	Liczba drzew zagłębionych lewostronnie (N!)	Liczba kształtów drzew krzaczastych (S(N))	Liczba drzew krzaczastych ((2N-2)!/(N-1)!)
2	2	1	2
3	6	2	12
4	24	5	120
5	120	14	1680
6	720	42	30 240
7	5040	132	665 280

Z tabeli 19.1 jasno wynika, że jeśli uwzględniane mają być wszystkie możliwe drzewa krzaczaste, przestrzeń rozwiązań szybko staje się niemożliwa do przetworzenia. W niektórych sytuacjach, np. dla skomplikowanych wersji schematu płatka śniegu (patrz podrozdział 29.3), zaproponowano techniki uwzględniania drzew krzaczastych<sup>12</sup>.

W przypadku drzew lewostronnie zagłębianych prawy potomek jest traktowany jako relacja wewnętrzna (w czasie wykonywania złączenia pętli zagnieżdżonych) lub jako relacja stosowana do sprawdzania (w trakcie wykonywania złączenia pętli zagnieżdżonych z użyciem indeksu). Jedną z zalet wiążących się z drzewami lewostronnie (lub prawostronnie) zagłębianymi jest to, że poddają się one przetwarzaniu potokowemu, co omówiono w podrozdziale 18.7. Przykładowo, weźmy pod uwagę drzewo lewostronnie zagłębiane z rysunku 19.5(a) i założmy, że algorytmem złączenia jest metoda pętli pojedynczej z użyciem indeksu. W takim przypadku blok dyskowy krotek relacji zewnętrznej jest używany w celu sprawdzania relacji wewnętrznej pod względem występowania pasujących krotek. W miarę jak jest tworzony wynikowy blok krotek ze złączenia relacji R1 i R2, może on być używany do sprawdzania relacji R3. Podobnie, w miarę jak jest tworzony wynikowy blok krotek z tego złączenia, może on być używany do sprawdzania relacji R4. Kolejną zaletą drzew lewostronnie (lub prawostronnie) zagłębianych jest to, że posiadanie relacji podstawowej jako jednej z danych wejściowych każdego złączenia pozwala optymalizatorowi na wykorzystanie wszelkich ścieżek dostępu na takiej relacji, które mogą być przydatne do wykonania złączenia.

Jeżeli zamiast technik potokowych wykorzystywana jest materializacja (patrz podrozdziały 18.7 i 19.2), wyniki złączenia mogą być materializowane i przechowywane jako relacje tymczasowe. Kluczowe znaczenie pod względem uporządkowania złączeń z punktu widzenia optymalizatora ma znalezienie takiego uporządkowania, które redukuje rozmiar wyników tymczasowych, gdyż wyniki tymczasowe (przetwarzane potokowo lub materializowane) są używane przez kolejne operatory, a więc wpływają na koszt wykonania związany z tymi operatorami.

#### 19.5.4. Optymalizacja fizyczna

Dla każdego logicznego planu zapytania opartego na opisanych wcześniej heurystykach operacje wymagają podjęcia dodatkowych decyzji związanych z ich wykonaniem przez konkretny algorytm na poziomie fizycznym. Jest to tzw. optymalizacja fizyczna. Jeśli jest ona oparta na względnym koszcie zastosowania każdej możliwej implementacji, nazywamy ją kosztową optymalizacją fizyczną. Dwie grupy technik podejmowania takich decyzji można na ogólnym poziomie określić podejściami typu góra-dół i dół-góra. W podejściu **góra-dół** poczynawszy od góry drzewa, analizowane są możliwości implementacji każdej operacji i na każdym etapie wybierane jest najlepsze rozwiązanie. W podejściu **dół-góra** analizujemy operacje, przechodząc w górę drzewa, oceniając możliwości fizycznego wykonania i wybierając najlepszą możliwość na każdym etapie. Teoretycznie oba podejścia sprowadzają się do oceny całej przestrzeni rozwiązań implementacyjnych w celu zminimalizowania kosztów przetwarzania. Jednak strategia dół-góra w naturalny sposób nadaje się do przetwarzania potokowego, dlatego jest stosowana w komercyjnych relacyjnych SZBD. Do najważniejszych

---

<sup>12</sup> Reprezentatywny scenariusz z użyciem drzew krzaczastych opisano w Ahmed i in. (2014).



fizycznych decyzji należy określienie kolejności operacji złączania, co pokrótce opisujemy w punkcie 19.5.5. Na etapie optymalizacji fizycznej stosowane są pewne heurystyki, dzięki czemu żmudne obliczenia kosztów są zbędne. Oto niektóre z tych heurystyk:

- Na potrzeby selekcji tam, gdzie to możliwe, należy stosować skanowanie indeksu.
- Dla koniunktywnych warunków selekcji należy w pierwszej kolejności zastosować wybór o najmniejszej liczności.
- Jeśli relacje są już posortowane według atrybutów dopasowywanych w złączeniu, należy przedkładać złączanie sortująco-scalające nad inne metody złączania.
- Przy obliczaniu sumy i części wspólnej więcej niż dwóch relacji należy posługiwać się regułą łączności. Relacje trzeba analizować rosnąco według szacowanej liczności.
- Jeśli jeden z argumentów złączenia posiada indeks na atrybucie złączania, należy zastosować ten argument jako relację wewnętrzną.
- Jeśli lewostronna relacja jest mała, a prawostronna duża i posiada indeks na kolumnie złączania, należy spróbować złączania pętli zagnieżdżonej z użyciem indeksu.
- Należy uwzględniać tylko te kolejności złączania, gdzie nie występują iloczyny kartezyjańskie; ewentualnie wszystkie złączenia muszą być wykonywane przed wyznaczeniem iloczynu kartezyjańskiego.

To tylko niektóre ze stosowanych przez optymalizator heurystyk z poziomu fizycznego. Jeśli liczba relacji jest niewielka (zwykle poniżej sześciu), przez co liczba możliwości implementacji jest ograniczona, większość optymalizatorów stosuje bezpośrednio optymalizację kosztową zamiast heurystyk.

### 19.5.5. Określanie kolejności złączeń za pomocą programowania dynamicznego

W punkcie 19.5.3 napisano, że istnieje wiele możliwych sposobów uporządkowania  $n$  relacji w złączeniu  $n$ -elementowym. Już dla  $n = 5$ , co w praktyce zdarza się nierzadko, występuje 120 permutacji dla drzew zagłębionych lewostronnie i 1680 dla drzew krzaczastych. Ponieważ drzewa krzaczaste skutkują poważnym rozszerzeniem przestrzeni rozwiązań, zwykle preferowane są drzewa zagłębione lewostronnie (względem drzew krzaczastych i zagłębionych prawostronnie). Mają one kilka zalet. Po pierwsze, dobrze sprawdzają się dla standardowych algorytmów złączania, w tym dla algorytmów z pętlą zagnieżdżoną, algorytmów z pętlą zagnieżdżoną opartych na indeksie i innych algorytmów jednoprzebiegowych. Po drugie, pozwalają generować **plany w pełni potokowe** (czyli takie, w których wszystkie złączenia można wykonać w ramach potoków). Warto zauważyć, że tabele wewnętrzne zawsze trzeba materializować, ponieważ w algorytmach złączania cała tabela wewnętrzna jest potrzebna w trakcie dopasowywania atrybutu złączania. W drzewach zagłębionych prawostronnie uzyskanie potrzebnego efektu nie jest możliwe.

Standardowym sposobem przetwarzania permutacji relacji ze złączenia jest zachłanna heurystyczna technika nazywana programowaniem dynamicznym. **Programowanie dynamiczne** to technika optymalizacji<sup>13</sup>, gdzie podproblemy są rozwiązywane tylko raz.

---

<sup>13</sup> Szczegółowe omówienie programowania dynamicznego jako techniki optymalizacji znajdziesz w podręczniku poświęconym algorytmom, np. Corman i in. (2003).

Można ją stosować, gdy problem da się podzielić na podproblemy, które same też składają się z podproblemów. Oto cechy typowego algorytmu programowania dynamicznego<sup>14</sup>:

- (1) Określana jest struktura optymalnego rozwiązania.
- (2) Wartość optymalnego rozwiązania jest definiowana rekurencyjnie.
- (3) Optymalne rozwiązanie jest obliczane, a jego wartość uzyskuje się metodą dół-góra.

Warto zauważyć, że rozwiązanie uzyskane za pomocą tej procedury jest optymalne lokalnie, a nie bezwzględnie. Aby zobaczyć, jak zastosować programowanie dynamiczne do wyboru kolejności złączania, rozważmy problem porządkowania pięcioelementowego złączenia relacji  $r_1, r_2, r_3, r_4, r_5$ . Ten problem ma 120 ( $=5!$ ) rozwiązań w postaci drzewa zagłębionego lewostronnie. W idealnych warunkach można oszacować i porównać koszt każdego z tych drzew oraz wybrać najlepsze z nich. W programowaniu dynamicznym problem jest dzielony na łatwiejsze do wykonania fragmenty. Wiadomo, że dla trzech relacji istnieje tylko sześć możliwych drzew zagłębionych lewostronnie. Warto zauważyć, że w celu przetworzenia wszystkich możliwych rozwiązań w postaci drzew krzaczastych należy uwzględnić ich 12. Złączenie można więc rozbić w następujący sposób:

$$r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4 \bowtie r_5 = (r_1 \bowtie r_2 \bowtie r_3) \bowtie r_4 \bowtie r_5$$

Sześć ( $=3!$ ) możliwych rozwiązań dla  $(r_1 \bowtie r_2 \bowtie r_3)$  można następnie połączyć z sześcioma sposobami złączenia wyniku pierwszych relacji (nazwijmy go tymcz1) z pozostałymi relacjami:

$$(\text{tymcz1} \bowtie r_2 \bowtie r_3)$$

Uwzględnienie sześciu możliwości uzyskania tymcz1 i dla każdej z nich sześciu sposobów wykonania drugiego złączenia ( $\text{tymcz1} \bowtie r_2 \bowtie r_3$ ) daje przestrzeń rozwiązań o wielkości  $6 \cdot 6 = 36$ . To dlatego programowanie dynamiczne można zastosować do wykonywania czegoś w rodzaju zachłannej optymalizacji. Wybierany jest wtedy „optymalny” plan uzyskania tymcz1, po czym algorytm *nie wraca* do tego planu. Tak więc przestrzeń rozwiązań jest ograniczona do tylko sześciu możliwości analizowanych dla drugiego złączenia. Zatem łączna liczba analizowanych możliwości wynosi  $6 + 6$  zamiast 120 ( $=5!$ ) z nieheurystycznego, kompletnego podejścia.

Przy określaniu najlepszego ogólnego porządku złączania ważna jest też kolejność generowania wyników złączenia, ponieważ w złączaniu sortująco-scalającym z następną relacją ta kolejność jest istotna. Kolejność korzystna dla następnego złączenia jest uznawana za **obiecuującą kolejność złączania**. To podejście po raz pierwszy zaproponowano w oprogramowaniu System R w IBM Research<sup>15</sup>. System R na potrzeby określania obiecujących kolejności uwzględnia (obok atrybutów złączania z późniejszego złączenia) też atrybuty grupowania z dalszej instrukcji GROUP BY i porządek sortowania w korzeniu drzewa. Przykładowo, w opisanym wcześniej scenariuszu obiecujące kolejności złączania relacji tymcz1 to te, które pasują do atrybutów złączania potrzebnych do złączenia tej relacji z  $r_4$  lub  $r_5$ . Algorytm programowania dynamicznego można rozwinąć, aby uwzględnił najlepsze kolejności złączania dla każdej obiecującej kolejności sortowania. Liczba podzbiorów  $n$  relacji wynosi  $2^n$  (dla  $n = 5$  otrzymujemy 32; dla  $n = 10$  mamy 1024, co też jest akcep-

<sup>14</sup> Na podstawie rozdziału 16. książki Corman i in. (2003).

<sup>15</sup> Zapoznaj się z klasyczną pracą z tego obszaru: Selinger i in. (1979).

towalne), a liczba obiecujących kolejności złączania jest niewielka. Złożoność rozszerzonego algorytmu programowania dynamicznego, który wyznacza optymalną permutację złączeń w drzewie zagłęzionym lewostronnie, wynosi  $O(3^n)$ .

## 19.6. Przykład ilustrujący kosztową optymalizację zapytań

W celu zilustrowania techniki kosztowej optymalizacji zapytań weźmiemy pod uwagę zapytanie Z2 i jego drzewo zapytania przedstawione na rysunku 19.1(a).

```
Z2: SELECT NUMERPROJ, NR_DZ, NAZWISKO, ADRES, DATAUR
      FROM PROJEKT, DZIAŁ, PRACOWNIK
      WHERE NR_DZ = NUMERDZ AND PESELKIEROWNIKA = PESEL AND
            LOKALIZACJAPROJ = 'GLIWICE';
```

Załóżmy, że posiadamy informacje na temat relacji, przedstawione na rysunku 19.6. Statystyki WARTOŚĆ\_NISKA oraz WARTOŚĆ\_WYSOKA zostały znormalizowane dla zwiększenia jasności. Co do drzewa z rysunku 19.1(a), zakładamy, że reprezentuje wynik procesu algebraicznej optymalizacji heurystycznej oraz początek optymalizacji kosztowej (w omawianym przykładzie zakładamy, że optymalizator heurystyczny nie przesuwając operacji projekcji w dół drzewa).

Pierwsza optymalizacja kosztowa polega na rozważeniu porządku złączeń. Jak wspomniano powyżej, zakładamy, że optymalizator rozpatruje tylko drzewa lewostronnie zagłęzione, więc potencjalnymi porządkami złączeń (bez iloczynu kartezjańskiego) są:

1. PROJEKT  $\bowtie$  DZIAŁ  $\bowtie$  PRACOWNIK
2. DZIAŁ  $\bowtie$  PROJEKT  $\bowtie$  PRACOWNIK
3. DZIAŁ  $\bowtie$  PRACOWNIK  $\bowtie$  PROJEKT
4. PRACOWNIK  $\bowtie$  DZIAŁ  $\bowtie$  PROJEKT

Zakładamy, że operacja selekcji została już zastosowana względem relacji PROJEKT. Jeżeli przyjmiemy, że stosowane jest podejście oparte na materializacji, wówczas dla każdej operacji złączenia tworzona jest nowa relacja tymczasowa. Zbadamy koszt złączeń w przypadku 1. Pierwsze złączenie dotyczy relacji PROJEKT i DZIAŁ. Musi zostać określona zarówno metoda złączenia, jak i metody dostępu dla relacji wejściowych. Ze względu na fakt, że relacja DZIAŁ nie posiada żadnego indeksu (według rysunku 19.6), jedyną możliwą metodą dostępu jest przegląd tabeli (czyli wyszukiwanie liniowe). Relacja PROJEKT będzie związana z wykonaniem operacji selekcji przed operacją złączenia, więc istnieją dwie możliwości: przegląd tabeli (wyszukiwanie liniowe) lub wykorzystanie indeksu PROJ\_LOKP, optymalizator musi zatem porównać ich szacowane koszty. Informacje statystyczne dotyczące indeksu PROJ\_LOKP (patrz rysunek 19.6) mówią, że liczba poziomów indeksu  $x$  wynosi 2 (korzeń plus poziom liści). Indeks jest nieunikatowy (ponieważ atrybut LOKALIZACJAPROJ nie jest kluczem relacji PROJEKT), więc optymalizator zakłada jednorodny rozkład danych i oszacowuje liczbę wskaźników rekordów dla każdej wartości

(a)

NAZWA_TABELI	NAZWA_KOLUMNY	LICZBA_ODRĘBNYCH	WARTOŚĆ_NISKA	WARTOŚĆ_WYSOKA
PROJEKT	LOKALIZACJAPROJ	200	1	200
PROJEKT	NUMERPROJ	2000	1	2000
PROJEKT	NR_DZ	50	1	50
DZIAŁ	NUMERDZ	50	1	50
DZIAŁ	PESELKIEROWNIKA	50	1	50
PRACOWNIK	PESEL	10000	1	10000
PRACOWNIK	NRDZ	50	1	50
PRACOWNIK	PENSJA	500	1	500

(b)

NAZWA_TABELI	LICZBA_WIERSZY	BLOKI
PROJEKT	2000	100
DZIAŁ	50	5
PRACOWNIK	10000	2000

(c)

NAZWA_INDEKSU	UNIKATOWOŚĆ	POZIOMB*	BLOKI_LIŚCI	ODRĘBNE_KLUCZE
PROJ_LOKP	NIEUNIKATOWY	1	4	200
PRAC_PESEL	UNIKATOWY	1	50	10000
PRAC_PENS	NIEUNIKATOWY	1	50	500

\* POZIOMB określa liczbę poziomów bez poziomu liści

RYSUNEK 19.6. Przykładowe dane statystyczne dla relacji z zapytania Z2. (a) Informacje o kolumnach. (b) Informacje o tabelach. (c) Informacje o indeksach

atrybutu LOKALIZACJAPROJ na 10. Wartość tę otrzymuje się na podstawie danych z rysunku 19.6, wykonując mnożenie  $\text{SELEKTYWNOŚĆ} \cdot \text{LICZBA\_WIERSZY}$ , gdzie SELEKTYWNOŚĆ jest szacowana jako  $1/\text{LICZBA\_ODRĘBNYCH}$ . Tak więc koszt użycia indeksu i uzyskania dostępu do rekordów można oszacować na 12 operacji dostępu do bloków (2 dla indeksu i 10 dla bloków danych). Koszt przeglądania tabeli szacuje się na 100 operacji dostępu do bloków, więc, zgodnie z oczekiwaniami, dostęp przy użyciu indeksu jest wydajniejszy.

W przypadku podejścia wykorzystującego materializację tworzony jest tymczasowy plik TEMP1 o rozmiarze 1 bloku w celu przechowywania wyniku operacji selekcji. Rozmiar pliku oblicza się określając współczynnik blokowy przy użyciu wzoru  $\text{LICZBA\_WIERSZY} / \text{BLOKI}$ , co daje  $2000/100$ , czyli 20 wierszy na blok. Stąd 10 wierszy wybranych z relacji PROJEKT zmieści się w pojedynczym bloku. Teraz możemy obliczyć szacowany koszt pierwszego złączenia. Będziemy rozpatrywać tylko metodę pętli zagnieżdżonych, gdzie relacją zewnętrzną jest plik tymczasowy TEMP1, zaś relacją wewnętrzną jest relacja DZIAŁ. Ze względu na fakt, że cały plik TEMP1 mieści się w dostępnej przestrzeni bufora, musimy wczytać każdy z pięciu bloków tabeli DZIAŁ tylko raz, tak więc koszt złączenia jest związany z dostępem do sześciu bloków oraz kosztem zapisania tymczasowego pliku wynikowego TEMP2. Optymalizator musiałby określić rozmiar pliku TEMP2. Atrybut złączenia NUMERDZ jest kluczem relacji DZIAŁ, więc dowolna wartość pola NR\_DZ w pliku TEMP1 zostanie

złączona z najwyżej jednym rekordem z relacji DZIAŁ, a stąd liczba wierszy w pliku TEMP2 będzie równa liczbie wierszy w pliku TEMP1, czyli 10. Optymalizator określa rozmiar rekordu dla pliku TEMP2 oraz liczbę bloków wymaganych do przechowania tych 10 rekordów. Dla uproszczenia zakładamy, że współczynnik blokowy dla pliku TEMP2 wynosi pięć wierszy na blok, więc w sumie w celu przechowania pliku TEMP2 potrzebne są dwa bloki.

Wreszcie należy oszacować koszt ostatniego złączenia. Możemy wykorzystać złączenie z pętlą pojedynczą na pliku TEMP2, ponieważ w takim przypadku można wykorzystać indeks PRAC\_PESEL (patrz rysunek 19.6) w celu sprawdzenia i zlokalizowania pasujących rekordów z relacji PRACOWNIK. Zatem metoda złączenia wiązałaby się z wczytaniem każdego bloku pliku TEMP2 i wyszukaniem każdej z pięciu wartości pola PESELKIEROWNIKA przy użyciu indeksu PRAC\_PESEL. Każde przeglądanie indeksu wymaga dostępu do korzenia, do liści oraz do bloku danych ( $x+1$ , gdzie liczba poziomów  $x$  wynosi 2). Tak więc 10 przeglądów wymaga 30 operacji dostępu do bloków. Dodanie dwóch operacji dostępu dla pliku TEMP2 daje w sumie 32 operacje dostępu do bloków w przypadku tego złączenia.

W przypadku końcowej projekcji zakładamy użycie przetwarzania potokowego w celu utworzenia wyniku końcowego, co nie wymaga dodatkowych operacji dostępu do bloków, więc całkowity koszt dla kolejności złączeń określonej w punkcie 1. jest sumą kosztów określonych powyżej. Optymalizator szacuje następnie w podobny sposób koszty dla pozostałych trzech kolejności złączeń i wybiera tę, która wiąże się z najniższym kosztem. Wykonanie takich obliczeń pozostawiamy jako ćwiczenie dla Czytelnika.

## 19.7. Dodatkowe zagadnienia związane z optymalizacją zapytań

W tym podrozdziale omówimy kilka istotnych kwestii, których nie zdołaliśmy poruszyć wcześniej.

### 19.7.1. Wyświetlanie planu wykonania zapytania uzyskanego przez system

Większość komercyjnych relacyjnych SZBD umożliwia wyświetlanie wygenerowanego przez optymalizator zapytań planu wykonania, dzięki czemu administratorzy baz danych mogą zobaczyć takie plany i spróbować zrozumieć decyzje podjęte przez optymalizator<sup>16</sup>. Typową składnią jest jedna z odmian wyrażenia EXPLAIN <zapytanie>.

- W bazach **Oracle** używany jest zapis:

```
EXPLAIN PLAN FOR  
<zapytanie w języku SQL>
```

---

<sup>16</sup> Tu przedstawiamy tylko sam mechanizm, bez opisywania składniowych szczegółów z poszczególnych systemów.

Zapytanie może obejmować instrukcje INSERT, DELETE i UPDATE. Dane wyjściowe trafiają do tabeli PLAN\_TABLE. Należy napisać odpowiednie zapytanie w języku SQL w celu odczytu danych z tej tabeli. Oracle udostępnia ponadto dwa skrypty: *UTLXPLS.SQL* i *UTLXPLP.SQL*, wyświetlające dane z tabeli PLAN\_TABLE oraz opisujące sekwencyjne i równoległe wykonanie zapytania.

- W bazach **IBM DB2** używany jest zapis:

```
EXPLAIN PLAN SELECTION [dodatkowe opcje] FOR <zapytanie w języku SQL>
```

Nie ma tu tabeli z planami. Polecenie PLAN SELECTION oznacza, że na etapie wyboru planu należy wczytać tabele objaśniające z odpowiednimi wyjaśnieniami. To samo polecenie można wykorzystać do objaśniania instrukcji w języku XQuery.

- W bazach **SQL Server** używany jest zapis:

```
SET SHOWPLAN_TEXT ON lub
SET SHOWPLAN_XML ON lub
SET SHOWPLAN_ALL ON
```

Te instrukcje były stosowane przed wprowadzeniem języka TRANSACT-SQL. Dlatego plan jest prezentowany w formie tekstowej, w formacie XML lub jako pełny tekst (tym możliwościom odpowiadają podane opcje).

- W bazach **PostgreSQL** używany jest zapis:

```
EXPLAIN [zestaw opcji] <zapytanie>
```

Dostępne opcje to: ANALYZE, VERBOSE, COSTS, BUFFERS, TIMING itd.

## 19.7.2. Szacowanie wielkości wyników dla innych operacji

W podrozdziałach 19.4 i 19.5 opisano operacje selekcji i złączania oraz szacowanie wyników zapytań obejmujących takie operacje. Tu analizujemy szacowanie wielkości wyników innych operacji.

**Projekcja.** Dla projekcji w postaci  $\pi_{LISTA}(R)$  zapisanej jako SELECT <lista atrybutów> FROM R szacowaną liczbą krotek w wyniku jest  $|R|$  (ponieważ SQL traktuje wynik jako wielozbiór). Jeśli używana jest opcja DISTINCT, wielkość wyniku operacji  $\pi_A(R)$  to  $LOW(A, R)$ .

**Operacje na zbiorach.** Jeśli argumenty przy obliczaniu części wspólnej, sumy lub różnicy zbiorów to efekt selekcji danych z tej samej relacji, wówczas operacje można zapisać jako koniunkcję, dysjunkcję lub negację. Przykładowo, operację  $\sigma_{c1}(R) \cap \sigma_{c2}(R)$  można zapisać jako  $\sigma_{c1 \text{ AND } c2}$ , a  $\sigma_{c1}(R) \cup \sigma_{c2}(R)$  jako  $\sigma_{c1 \text{ OR } c2}$ . Wielkość wyniku można oszacować na podstawie selektywności warunków  $c1$  i  $c2$ . Inna możliwość to oszacowanie górnego ograniczenia dla operacji  $r \cap s$  na minimum wielkości  $r$  i  $s$ ; dla operacji  $r \cup s$  górnym ograniczeniem jest suma wielkości  $r$  i  $s$ .

**Agregacja.** Wielkością wyniku operacji  $G \overset{\text{funkcja-agregująca}}{\bowtie} (A) R$  jest  $LOW(G, R)$ , ponieważ występuje jedna grupa dla każdej unikatowej wartości  $G$ .

**Złączenie zewnętrzne.** Wielkość wyniku operacji  $R \text{ LEFT OUTER JOIN } S \text{ to } |R \bowtie S|$  plus  $|R \text{ antyzłączenie } S|$ . Podobnie dla operacji  $R \text{ FULL OUTER JOIN } S$  otrzymujemy  $|r \bowtie s|$  plus  $|r \text{ antyzłączenie } s|$  plus  $|s \text{ antyzłączenie } r|$ . Szacowanie selektywności antyzłączeń opisaliśmy w punkcie 19.5.1.



### 19.7.3. Zapis planu w pamięci podręcznej

W rozdziale 2. wspomnieliśmy o użytkownikach parametrycznych, którzy wielokrotnie uruchamiają te same zapytania lub transakcje, ale z innymi zestawami parametrów. Przykładowo, kasjer używa numeru konta i kodu funkcji do sprawdzenia stanu danego rachunku. Aby wielokrotnie uruchamiać takie zapytania lub transakcje, optymalizator oblicza najlepszy plan, gdy zapytanie jest przesyłane po raz pierwszy, i zapisuje go w pamięci podręcznej w celu późniejszego użycia. Zachowywanie planu i ponowne użytkowanie go to **zapis planu w pamięci podręcznej**. Gdy zapytanie zostanie ponownie przesłane z innymi stałymi jako parametrami, ten sam plan jest używany jeszcze raz z nowymi parametrami. Możliwe, że w pewnych sytuacjach konieczne będzie zmodyfikowanie tego planu. Przykładowo, jeśli zapytanie generuje raport na podstawie zakresu dat lub kont, to w zależności od ilości danych stosowane mogą być różne strategie. W technice nazywanej **parametryczną optymalizacją zapytań** zapytanie jest optymalizowane bez części wartości parametrów, a optymalizator zwraca liczbę planów dla różnych możliwych zbiorów wartości. Wszystkie te informacje są zapisywane w pamięci podręcznej. Gdy zapytanie jest przesyłane, parametry z niego są porównywane z parametrami z różnych planów i używany jest najmniej kosztowny z planów zgodnych z zapytaniem.

### 19.7.4. Optymalizacja z wykorzystaniem pierwszych $k$ wyników

Gdy oczekiwane są duże dane wyjściowe z zapytania, czasem użytkownikowi wystarczy pierwszych  $k$  wyników ustalonych według jakiegoś sposobu sortowania. W niektórych relacyjnych SZBD występuje **klauzula limitu**  $K$  elementów, ograniczająca wielkość wyników do tego rozmiaru. Podobnie można stosować wskazówki nakazujące optymalizatorowi ograniczenie generowania wyników. Próba wygenerowania całego zbioru wyników i późniejszego wyświetlenia tylko  $k$  pierwszych po ich posortowaniu to naiwne i niewydatne podejście. Jedną z sugerowanych strategii polega na generowaniu posortowanych wyników, tak aby można było zakończyć przetwarzanie po  $K$  krotkach. Zaproponowano też inne strategie, np. wprowadzanie dodatkowych warunków selekcji na podstawie szacowanej najwyższej wartości. Szczegółowe omówienie tego zagadnienia wykracza poza zakres tej książki. Więcej informacji znajdziesz w podanej bibliografii.

## 19.8. Przykład optymalizacji zapytań w hurtowniach danych

W tym podrozdziale przedstawimy następny przykład zastosowania przekształceń zapytań jako techniki optymalizacji wykonywania zapytań. W podrozdziale 19.2 zapoznałeś się z przykładami przekształcania zapytań. Dotyczyły one podzapytań zagnieżdżonych i wykorzystano tam heurystyki zamiast optymalizacji kosztowej. Przedstawione scalanie podzapytań (perspektyw) można uznać za technikę przekształcania heurystycznego, jednak w scalaniu perspektyw z instrukcją `GROUP BY` używana jest też optymalizacja kosztowa. W tym podrozdziale omawiamy wykorzystanie kosztów do przekształcania zapytań z użyciem schematu gwiazdy w hurtowniach danych. Takie zapytania są powszechnie stosowane



w hurtowniach danych opartych na schemacie gwiazdy. Omówienie tego schematu znajdziesz w podrozdziale 29.3.

Omawianą procedurę nazwiemy **optymalizacją z przekształceniami w schemacie gwiazdy**. Schemat gwiazdy obejmuje zestaw tabel. Jego nazwa wynika z podobieństwa kształtu schematu do gwiazdy; w środku znajduje się jedna lub kilka tabel (relacji) faktów powiązanych z wieloma tabelami (relacjami) wymiarów. Tabela faktów zawiera informacje na temat związków (np. sprzedaży) między różnymi tabelami wymiarów (np. z danymi o klientach, częściach, dostawcach, kanale sprzedaży, roku) i kolumnami wartości (np. sprzedane sztuki). Rozważmy pokazane dalej przykładowe zapytanie ZSTAR. Załóżmy, że W1, W2 i W3 to aliasy tabel wymiarów WYM1, WYM2 i WYM3 z kluczami głównymi W1.KG, W2.KG i W3.KG. W tabeli faktów FAKT (alias F) występują powiązane z tymi wymiarami atrybuty klucza obcego: F.KO1, F.KO2 i F.KO3. Z użyciem tych atrybutów definiowane są złączenia. Zapytanie tworzy grupy na podstawie atrybutów W1.X i W2.Y i generuje sumę dla atrybutu wartości (patrz podrozdział 29.3) F.M z tabeli faktów F. Dla atrybutów A, B i C z WYM1, WYM2 i WYM3 obowiązują pewne warunki:

Zapytanie ZSTAR:

```
SELECT W1.X, W2.Y, SUM (F.M)
FROM   FAKT F, WYM1 W1, WYM2 W2, WYM3 W3
WHERE  F.KO1 = W1.KG and F.KO2 = W2.KG and F.KO3 = W3.KG and
       W1.A > 5 and W2.B < 77 and W3.C = 11
GROUP BY W1.X, W2.Y
```

Tabela faktów jest zwykle bardzo duża w porównaniu z tabelami wymiarów. ZSTAR to typowe zapytanie dla schematu gwiazdy, a bardzo duża tabela faktów jest przeważnie złączana z kilkoma małymi tabelami wymiarów. Zapytanie może ponadto obejmować jednotabelowe predykaty filtrujące dotyczące innych kolumn tabel wymiarów. Takie predykaty są przeważnie restrykcyjne. Połączenie takich filtrów (np. W1.A > 5 w pokazanym zapytaniu) pomaga znacznie zmniejszyć ilość przetwarzanych danych z tabeli faktów. Zapytania tego rodzaju zwykle grupują dane na podstawie kolumn z tabel wymiarów i agregują informacje z użyciem kolumn wartości z tabeli faktów.

Celem optymalizacji z przekształceniami w schemacie gwiazdy jest dostęp tylko do ograniczonego zbioru danych z tabeli faktów i uniknięcie pełnego przeglądu tabeli. Możliwe są dwie takie optymalizacje: (A) klasyczne przekształcenie w schemacie gwiazdy i (B) przekształcenie z użyciem indeksu bitmapowego. Obie te optymalizacje są wykonywane na podstawie porównania kosztów pierwotnego i przekształconego zapytania.

#### A. Klasyczne przekształcenia w schemacie gwiazdy.

W tej optymalizacji najpierw obliczany jest iloczyn kartezyjański tabel wymiarów (po zastosowaniu do każdej z tych tabel filtrów takich jak W1.A > 5). Warto zauważyć, że zwykle dla tabel wymiarów nie są używane predykaty złączenia. Iloczyn kartezyjański jest następnie złączany z tabelą faktów z wykorzystaniem indeksów w formie B-drzew (jeśli są dostępne) na kluczach złączania z tabeli faktów.

#### B. Przekształcenia z użyciem indeksu bitmapowego.

W tej optymalizacji wymagane są indeksy bitmapowe<sup>17</sup> na używanych w zapytaniu kluczach złączania z tabeli faktów. Przykładowo, dla zapytania ZSTAR niezbędne są

<sup>17</sup> W niektórych sytuacjach klucze tworzące indeks w formie B-drzewa można przekształcić na bitmapy. Tu nie omawiamy tej techniki.

indeksy bitmapowe (patrz punkt 17.5.2) na atrybutach FAKT.K01, FAKT.K02 i FAKT.K03. Każdy bit w takiej bitmapie odpowiada wierszowi z tabeli faktów. Bit jest równy 1, jeśli wartość atrybutu będącego kluczem występuje w danym wierszu tabeli faktów. Zapytanie ZSTAR jest przekształcane w pokazane poniżej zapytanie Z2STAR:

### Z2STAR:

```
SELECT W1.X, W2.Y, SUM (F.M)
FROM FACT F, WYM1 W1, WYM2 W2
WHERE F.K01 = W1.KG and F.K02 = W2.KG and W1.A > 5 and W2.B < 77 and
      F.K01 IN (SELECT W1.KG
                FROM   WYM1 W1
                WHERE  W1.A > 5) AND
      F.K02 IN (SELECT W2.KG
                FROM   WYM2 W2
                WHERE  W2.B < 77) AND
      F.K03 IN (SELECT W3.KG
                FROM   WYM3 W3
                WHERE  W3.C = 11)
GROUP BY W1.X, W2.Y;
```

Przekształcenie z użyciem indeksu bitmapowego powoduje dodanie predykatów dotyczących tabel wymiarów. Warto zauważyć, że podzapytania dodane w Z2STAR można traktować jak operacje sprawdzania przynależności do zbioru (np. F.K01 IN (5, 9, 12, 13, 29 ...)).

Gdy wykonywane są operacje AND i OR na bitmapach dotyczących wartości kluczy występujących w podzapytaniach dla wymiarów, z tabeli faktów trzeba pobrać tylko odpowiednie wiersze. Jeśli predykaty filtrujące tabele wymiarów i wyznaczanie części wspólnej w trakcie złączania tabeli faktów z tabelami wymiarów spowodują odfiltrowanie dużego podzbioru wierszy tabeli faktów, opisana optymalizacja jest znacznie wydajniejsza od siłowego przeglądania całej tabeli faktów.

Oto operacje wykonywane w zapytaniu Z2STAR w celu dostępu do tabeli FAKT i złączenia jej:

- (1) W wyniku iteracyjnego sprawdzania wartości kluczy z podzapytań dotyczących wymiarów otrzymywane są bitmapy dla danej wartości klucza z indeksu bitmapowego tabeli FAKT.
- (2) Na potrzeby podzapytania bitmapy pobrane dla różnych wartości kluczy są scalane (operacja OR).
- (3) Scalone bitmapy dla wszystkich podzapytań dotyczących wymiarów są łączone za pomocą operacji AND. Oznacza to, że wyznaczana jest koniunkcja złączeń.
- (4) Na podstawie końcowej bitmapy generowane są identyfikatory krotek z tabeli FAKT.
- (5) Wiersze z tabeli FAKT są bezpośrednio pobierane za pomocą identyfikatorów krotek.

**Ponowne złączanie.** Bitmapy z podzapytań pozwalają przefiltrować tabelę faktów na podstawie predykatów filtrujących dotyczących tabel wymiarów. Dlatego konieczne może być ponowne złączenie tabel wymiarów z odpowiednimi wierszami z tabeli faktów za pomocą pierwotnych predykatów złączeń. Ponownego złączania tabel wymiarów można uniknąć, jeśli kolumny pobrane z podzapytania są unikatowe, a kolumny z tabeli wymiarów nie występują w klauzulach SELECT i GROUP BY. Warto zauważyć, że w zapytaniu Z2STAR tabela WYM3

nie jest ponownie złączana z tabelą FAKT, ponieważ nie jest używana w klauzulach SELECT i GROUP BY, a klucz WYM3.KG jest unikatowy.

## 19.9. Optimalizacja zapytań w bazach Oracle<sup>18</sup>

Ten podrozdział zawiera obszernie omówienie różnych funkcji z zakresu przetwarzania zapytań w bazach Oracle, w tym optymalizacji, wykonywania i analiz zapytań<sup>19</sup>.

### 19.9.1. Optymalizator fizyczny

Optymalizator fizyczny z baz Oracle działa na podstawie kosztów i został wprowadzony w Oracle 7.1. Zasięg jego działania to jeden blok zapytania. Optymalizator fizyczny sprawdza różne ścieżki dostępu do tabel i indeksów, algorytmy wykonywania operatorów, kolejności złączania, metody złączania, techniki równoległego wykonywania zapytania itd. Na tej podstawie wybiera plan wykonania o szacunkowo najniższych kosztach. Szacowany koszt wykonania zapytania to liczba proporcjonalna do oczekiwanego czasu potrzebnego na wykonanie zapytania z użyciem danego planu wykonania.

Optymalizator fizyczny oblicza koszt na podstawie statystyk dotyczących obiektów (np.: liczności elementów w tabelach, liczby różnych wartości w kolumnie, najwyższej i najniższej wartości w kolumnie, rozkładu wartości kolumn), szacowanego wykorzystania zasobów (np. operacji wejścia-wyjścia i czasu procesora) oraz zapotrzebowania na pamięć. Szacowany koszt to wewnętrzna miara w przybliżeniu odpowiadająca czasowi wykonywania zapytania i wymaganym zasobom. Celem optymalizacji kosztowej w bazach Oracle jest znalezienie kompromisu między najkrótszym czasem wykonania a najniższym zużyciem zasobów.

### 19.9.2. Globalny optymalizator zapytań

W tradycyjnych relacyjnych SZBD optymalizacja zapytań obejmuje dwie odrębne fazy optymalizacji logicznej i fizycznej. Natomiast w bazach Oracle używany jest globalny optymalizator zapytań, w którym etapy przekształceń logicznych i optymalizacji fizycznej są zintegrowane w celu wygenerowania optymalnego planu wykonania dla całego drzewa zapytania. Architektura przetwarzania zapytań w bazach Oracle jest przedstawiona na rysunku 19.7.

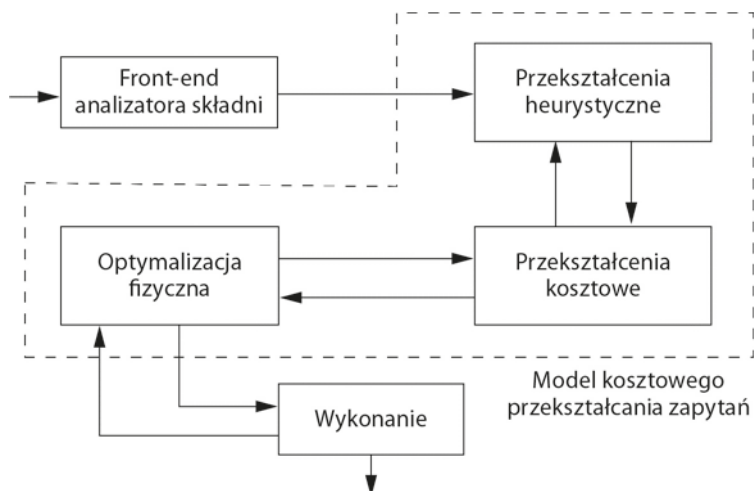
Bazy Oracle wykonują wiele przekształceń zapytań, co skutkuje zmianami i transformacją zapytań od użytkowników na równoważną, ale potencjalnie lepszą postać. Wykonywane są przekształcenia heurystyczne i kosztowe. Model kosztowych przekształceń zapytań<sup>20</sup> wprowadzony w Oracle 10g zapewnia wydajne mechanizmy do eksploracji przestrzeni stanów wygenerowanych w wyniku zastosowania jednego lub wielu przekształceń. W przekształceniach kosztowych instrukcja w języku SQL, która może obejmować wiele

---

<sup>18</sup> Autorem tego podrozdziału jest Rafi Ahmed z Oracle Corporation.

<sup>19</sup> Obsługę analiz wprowadzono w Oracle 10.2.

<sup>20</sup> Opisany w pracy Ahmeda i in. (2006).



RYSUNEK 19.7. Model kosztowego przekształcania zapytań (oparty na: Ahmed i in., 2006)

bloków zapytania, jest kopiowana i modyfikowana, a optymalizator fizyczny oblicza jej koszt. Ten proces jest powtarzany wielokrotnie. Za każdym razem stosowany jest nowy zestaw przekształceń (mogą być one zależne od siebie). Ostatecznie wybierane jest jedno lub kilka przekształceń, które są stosowane do pierwotnej instrukcji w języku SQL (jeśli prowadzą do optymalnego planu wykonania zapytania). Aby poradzić sobie z eksplozją kombinatoryczną, model kosztowego przekształcania zapytań zapewnia wydajne strategie przeszukiwania przestrzeni stanów z różnymi przekształceniami.

Dostępność ogólnego modelu przekształceń kosztowych umożliwiła dodanie innych innowacyjnych transformacji do bogatego repertuaru technik przekształcania zapytań dostępnego w bazach Oracle. Do najważniejszych z tych przekształceń należą: scalanie podzapytań (w klauzuli `FROM` zapytania) obejmujących instrukcje `GROUP BY` i `DISTINCT`, eliminowanie zagnieżdżonych podzapytań, przenoszenie predykatów, eliminowanie powtarzających się podwyrażeń, przenoszenie predykatów złączenia w dół, rozwijanie instrukcji `OR`, łączenie podzapytań, faktoryzacja złączeń (ang. *join factorization*), eliminowanie podzapytań za pomocą funkcji okna, przekształcenia w schemacie gwiazdy, przenoszenie instrukcji `GROUP BY` i krzaczaste drzewa złączenia<sup>21</sup>.

Model przekształceń kosztowych w Oracle 10g to dobry przykład zaawansowanego podejścia do optymalizowania zapytań w języku SQL.

### 19.9.3. Optymalizacja adaptacyjna

Optymalizator fizyczny w bazach Oracle działa adaptacyjnie i używa informacji zwrotnych z poziomu wykonania w celu ulepszania wcześniejszych decyzji. Optymalizator wybiera optymalny plan wykonania danej instrukcji języka SQL, używając modelu kosztowego, opartego na statystykach obiektowych (np. liczba wierszy lub rozkład wartości w kolumnie) i systemowych (np. przepustowość operacji wejścia-wyjścia w podsystemie składowania

<sup>21</sup> Więcej szczegółów zawierają prace Ahmeda i in. (2006, 2014).

danych). Optymalność ostatecznego planu wykonania zależy przede wszystkim od dokładności używanych statystyk, a także od złożoności samego modelu. W Oracle pętla informacji zwrotnych z rysunku 19.7 tworzy pomost między silnikiem wykonawczym a optymalizatorem fizycznym. Ten pomost zapewnia cenne informacje statystyczne, umożliwiające optymalizatorowi fizycznemu ocenę skutków podjętych decyzji oraz dokonywanie lepszych wyborów w trakcie aktualnego wykonywania zapytania i późniejszych jego uruchomień. Przykładowo, na podstawie oszacowanej liczności elementów w tabeli optymalizator może wybrać metodę złączania pętli zagnieżdżonej z użyciem indeksu. Jednak w fazie wykonania może się okazać, że rzeczywista liczność elementów znacznie różni się od szacunków. Te informacje mogą spowodować, że fizyczny optymalizator zmieni decyzję i dynamicznie zrezygnuje ze złączania z użyciem indeksu na rzecz złączania mieszającego.

### 19.9.4. Przetwarzanie tablicowe

Jednym z ważnych braków w implementacjach języka SQL jest brak obsługi przetwarzania z użyciem tablic  $N$ -wymiarowych. Firma Oracle opracowała rozszerzenia na potrzeby analityki i mechanizmów OLAP. Te rozszerzenia zostały zintegrowane z silnikiem relacyjnego SZBD Oracle<sup>22</sup>. Potrzebę stosowania zapytań OLAP wyjaśnimy w trakcie omawiania hurtowni danych w rozdziale 29. Wspomniane rozszerzenia języka SQL związane z obliczeniami z użyciem tablic na potrzeby złożonego modelowania i optymalizacji obejmują struktury dostępowe i strategie wykonania pozwalające wydajnie przetwarzać takie obliczenia. Klauzula obliczeń (jej omawianie wykracza poza zakres tej książki) sprawia, że relacyjny SZBD Oracle może traktować tabelę jak wielowymiarową tablicę i stosować do niej zestaw wzorów. Te wzory zastępują liczne złączenia i operacje UNION, które trzeba wykonywać dla analogicznych obliczeń w obecnym języku ANSI SQL (ANSI — *American National Standards Institute*). Klauzula obliczeń nie tylko ułatwia tworzenie aplikacji, ale też umożliwia relacyjnemu SZBD Oracle skuteczniejszą optymalizację.

### 19.9.5. Wskazówki

Ciekawym dodatkiem do optymalizatora zapytań w bazach Oracle jest możliwość podawania przez programistę aplikacji wskazówek dla optymalizatora (nazywanych też *uwagami* lub *dyrektywami*). Wskazówki są umieszczane w tekście instrukcji w języku SQL. Często wykorzystuje się je w tych rzadkich sytuacjach, kiedy optymalizator wybiera nieoptymalny plan. Związane jest to z tym, że programista aplikacji czasem może chcieć zmienić decyzję optymalizatora z powodu błędnego oszacowania kosztów lub liczności elementów. Rozważmy np. tabelę PRACOWNIK z rysunku 5.6. W kolumnie PŁEĆ tej tabeli znajdują się tylko dwie różne wartości. Jeśli liczba pracowników wynosi 10 000, optymalizator z powodu braku histogramu dla kolumny PŁEĆ przyjmie równomierny rozkład danych i oszacuje, że liczba mężczyzn i kobiet jest równa. Jeżeli istnieje indeks drugorzędny, prawie na pewno nie zostanie użyty. Jeśli jednak programista aplikacji wie, że mężczyzn jest tylko 100, może zastosować wskazówkę w zapytaniu w języku SQL z warunkiem PŁEĆ='M' w klauzuli WHERE, aby zasugerować użycie powiązanego indeksu w trakcie przetwarzania zapytania. Dla różnych operacji można stosować różne rodzaje wskazówek. Oto niektóre z nich:

---

<sup>22</sup> Więcej szczegółów znajdziesz w tekście Witkowskiego i in. (2003).

- Ścieżka dostępu do danej tabeli.
- Kolejność złączania w bloku zapytania.
- Konkretna metoda złączania tabel.
- Aktywowanie lub blokowanie przekształceń.

### 19.9.6. Zarysy

W relacyjnym SZBD Oracle zarysy (ang. *outlines*) służą do zachowywania planów wykonania instrukcji lub zapytań w języku SQL. Zarysy są implementowane i przedstawiane jako kolekcja wskazówek, ponieważ wskazówki łatwo jest przenosić i zrozumieć. Oracle udostępnia rozbudowany zestaw wskazówek, dający na tyle duże możliwości, by określić dowolny, nawet najbardziej złożony plan wykonania. Gdy zarys jest używany na etapie optymalizowania instrukcji w języku SQL, wskazówki są stosowane na odpowiednich etapach przez optymalizator (i inne komponenty). Każda instrukcja przetwarzana przez optymalizator z baz Oracle automatycznie powoduje wygenerowanie zarysu, który można wyświetlić razem z planem wykonania. Zarysy są używane np. do zapewniania stabilności planu, analiz typu „co jeśli” i eksperymentów z zakresu wydajności.

### 19.9.7. Zarządzanie planami wykonywania instrukcji języka SQL

Plany wykonywania instrukcji języka SQL mają istotny wpływ na ogólną wydajność systemów baz danych. Nowe statystyki optymalizatora, zmiany w parametrach konfiguracyjnych, aktualizacje oprogramowania, wprowadzenie nowych technik optymalizowania i przetwarzania zapytań i obciążenie zasobów sprzętowych to niektóre z wielu czynników, które mogą spowodować wygenerowanie przez optymalizator zapytań w bazach Oracle nowego planu wykonania tych samych zapytań lub instrukcji w języku SQL. Choć większość zmian w planach wykonania jest korzystna lub nieszkodliwa, niektóre plany mogą się okazać nieoptymalne i wywierać negatywny wpływ na wydajność systemu.

W Oracle 11g wprowadzono nowy mechanizm — zarządzanie planami wykonywania instrukcji języka SQL (ang. *SQL plan management* — SPM)<sup>23</sup>. Służy on do zarządzania planami wykonania zbiorów zapytań lub procesów roboczych. Mechanizm SPM zapewnia stabilną i optymalną wydajność zbiorów instrukcji w języku SQL, ponieważ zapobiega wykonywaniu nowych nieoptymalnych planów, a jednocześnie pozwala wykonywać nowe plany, jeśli zweryfikowano, że ich wydajność jest wyższa niż pierwotnych. Obejmuje złożony mechanizm zarządzania planami wykonywania zbiorów instrukcji języka SQL, dla których użytkownik włączył SPM. Przechowuje też wcześniejsze plany wykonywania w postaci zarysów powiązanych z tekstem instrukcji języka SQL oraz porównuje wydajność pierwotnego i nowego planu wykonywania danej instrukcji, zanim umożliwi zastosowanie danego planu przez użytkownika. Mechanizm SPM można skonfigurować tak, aby działał automatycznie. Można też stosować go ręcznie do jednej lub kilku instrukcji języka SQL.

---

<sup>23</sup> Patrz Ziauddin i in. (2008).

## 19.10. Semantyczna optymalizacji zapytań

Zaproponowano również odmienne podejście do kwestii optymalizacji, noszące nazwę **semantycznej optymalizacji zapytań** (ang. *semantic query optimization*). Technika ta, której można używać w połączeniu z wcześniej omówionymi technikami, wykorzystuje więzy określone na schemacie bazy danych — takie jak atrybuty unikatowe i inne bardziej złożone warunki — w celu zmodyfikowania zapytania do postaci innego, którego wykonanie będzie wydajniejsze. Nie będziemy tego zagadnienia omawiać szczegółowo, a tylko zilustrujemy je na prostym przykładzie. Weźmy pod uwagę poniższe zapytanie SQL:

```
SELECT P.NAZWISKO, R.NAZWISKO
FROM   PRACOWNIK AS P, PRACOWNIK AS R
WHERE  P.PESELPRZEŁOŻ = R.PESEL AND P.PENSJA > R.PENSJA
```

Powyższe zapytanie pobiera nazwiska pracowników, którzy zarabiają więcej od swoich przełożonych. Załóżmy, że posiadamy więzy na schemacie bazy danych określające, że żaden pracownik nie może zarabiać więcej od swojego przełożonego. Jeżeli semantyczny optymalizator zapytań sprawdza występowanie takich więzów, nie musi wykonywać zapytania w ogóle, ponieważ wiadomo wówczas, że jego wynik będzie pusty. Może to pozwolić zaoszczędzić wiele czasu, o ile sprawdzanie więzów da się przeprowadzać w wydajny sposób. Jednak przeszukiwanie wielu więzów w celu znalezienia tych, które mają zastosowanie w przypadku danego zapytania i które mogą je semantycznie optymalizować, również może okazać się czasochłonne.

Rozważmy następujący przykład:

```
SELECT NAZWISKO, PENSJA
FROM   PRACOWNIK, DZIAŁ
WHERE  PRACOWNIK.NRDZ = DZIAŁ.NUMERDZ and
PRACOWNIK.PENSJA>10000
```

W tym przykładzie atrybuty są pobierane z tylko jednej relacji: PRACOWNIK. Warunek selekcji także dotyczy tej relacji. Występują tu jednak ograniczenia integralności odwołań: PRACOWNIK.NRDZ to klucz obcy powiązany z kluczem głównym DZIAŁ.NUMERDZ. Dlatego to zapytanie można przekształcić, usuwając z zapytania relację DZIAŁ i unikając dzięki temu złączenia wewnętrznego:

```
SELECT NAZWISKO, PENSJA
FROM   PRACOWNIK
WHERE  PRACOWNIK.NRDZ IS NOT NULL and PRACOWNIK.PENSJA>10000
```

Przekształcenia tego rodzaju są oparte na semantyce związku klucz główny-klucz obcy, który tworzy więzy między dwiema relacjami.

Wraz z pojawieniem się w systemach baz danych reguł aktywnych i dodatkowych meta-danych (patrz rozdział 26.) techniki semantycznej optymalizacji zapytań są stopniowo wprowadzane w SZBD.



## 19.11. Podsumowanie

W poprzednim rozdziale przedstawiono strategie przetwarzania zapytań stosowane w relacyjnych SZBD. Omówiono algorytmy dla różnych standardowych operatorów relacyjnych, w tym selekcji, projekcji i złączania. Opisano też inne rodzaje złączeń, w tym złączenia zewnętrzne, złączenia częściowe i antyzłączenia, a także agregację i sortowanie zewnętrzne. W tym rozdziale naszym celem było skupienie się na technikach optymalizacji zapytań stosowanych w relacyjnych SZBD. W podrozdziale 19.1 wprowadzono notację drzew zapytań i grafów oraz opisano heurystyczne sposoby optymalizacji zapytań. Te sposoby to reguły heurystyczne i techniki algebraiczne pozwalające poprawić wydajność wykonywania zapytań. Pokazano, w jaki sposób drzewa zapytań reprezentujące wyrażenia algebry relacyjnej można heurystycznie optymalizować, przedstawiając wierzchołki i przekształcając drzewo w inne, równoważne, które umożliwia wydajniejsze wykonanie zapytania. Przedstawiono też reguły przekształceń z zachowaniem równoważności i procedury systematycznego stosowania tych reguł do drzewa zapytań. W podrozdziale 19.2 opisano inne plany wykonywania zapytań, wykorzystujące przetwarzanie potokowe i materializację. Dalej wprowadzono przekształcanie zapytań języka SQL w celu optymalizacji zagnieżdżonych podzapytań. Ponadto pokazano na przykładach scalanie podzapytań z klauzuli `FROM`, działających podobnie jak relacje pochodne lub perspektywy. Opisano też technikę materializowania perspektyw.

W podrozdziale 19.3 dość szczegółowo omówiono kosztowe optymalizowanie zapytań. Opisano tam przechowywane w katalogach informacje, z których korzysta optymalizator zapytań. Przedstawiono również histogramy przechowujące rozkład ważnych atrybutów. W podrozdziałach 19.4 i 19.5 pokazano, jak tworzone są funkcje kosztów dla algorytmów dostępu do baz danych w operacjach selekcji i złączania. W podrozdziale 19.6 za pomocą przykładu przedstawiono, jak te funkcje są używane do szacowania kosztów wykonywania różnych strategii. W podrozdziale 19.7 opisano różne dodatkowe zagadnienia, takie jak wyświetlanie planów zapytania, szacowanie wielkości wyników, zapisywanie planu w pamięci podręcznej i optymalizację z użyciem pierwszych  $k$  wyników. Podrozdział 19.8 poświęcono omówieniu optymalizacji typowych zapytań w hurtowniach danych. Przedstawiono przykład kosztowego przekształcania zapytań w hurtowniach danych w tzw. schemacie gwiazdy. W podrozdziale 19.9 zaprezentowano szczegółowe omówienie optymalizatora zapytań w bazach Oracle, w którym używane są liczne dodatkowe techniki. Dokładny ich opis wykracza poza zakres tego tekstu. Na zakończenie, w podrozdziale 19.10, wspomniano o technice semantycznej optymalizacji zapytań, gdzie na podstawie semantyki lub ograniczeń odwołań można uprościć zapytanie albo całkowicie uniknąć dostępu do danych lub wykonywania zapytania.

## Pytania powtórkowe

- 19.1. Czym jest plan wykonania zapytania?
- 19.2. Co rozumiemy pod pojęciem *optymalizacji heurystycznej*? Omów podstawowe zasady heurystyczne stosowane w czasie optymalizacji zapytań.
- 19.3. W jaki sposób drzewo zapytań reprezentuje wyrażenie algebry relacyjnej? Co rozumiemy przez wykonanie drzewa zapytania? Omów reguły przekształcania drzew zapytań i zidentyfikuj, kiedy każda reguła powinna zostać zastosowana w czasie optymalizacji.

- 19.4. Ile różnych kolejności wykonania złączeń istnieje w przypadku zapytania zawierającego 10 relacji?
- 19.5. Co rozumiemy pod pojęciem *kosztowej optymalizacji zapytań*?
- 19.6. Czym jest optymalizacja oparta na programowaniu dynamicznym? Jak ta metoda jest stosowana w trakcie optymalizacji zapytań?
- 19.7. Jakie problemy są związane z utrzymywaniem widoków zmaterializowanych?
- 19.8. Opisz różnice między *przetwarzaniem potokowym* a *materializacją*.
- 19.9. Omów składowe koszty dla funkcji używanej w celu oszacowania kosztu zapytania. Które składowe są częściej wykorzystywane jako podstawa funkcji kosztu?
- 19.10. Omów różne rodzaje parametrów, które są używane w funkcjach kosztu. Gdzie są przechowywane te informacje?
- 19.11. Czym są złączenia częściowe i antyzłączenia? Czym są powiązane z nimi selektywność złączenia i liczność wyniku złączenia? Podaj odpowiednie wzory.
- 19.12. Wymień funkcje kosztu dla metod SELECT i JOIN omówionych w podrozdziałach 19.4 i 19.5.
- 19.13. Jakie specjalne mechanizmy optymalizacji baz Oracle nie zostały omówione w tym rozdziale?
- 19.14. Co rozumiemy pod pojęciem *semantycznej optymalizacji zapytań*? W jaki sposób różni się ono od innych technik optymalizacji zapytań?

## Ćwiczenia

- 19.15. Określ funkcje kosztu algorytmów PROJECT, UNION, INTERSECTION, SET DIFFERENCE oraz CARTESIAN PRODUCT omówionych w podrozdziale 19.4.
- 19.16. Określ funkcje kosztu dla algorytmu składającego się z dwóch operacji SELECT, operacji JOIN oraz końcowej operacji PROJECT w kontekście funkcji kosztu dla poszczególnych operacji.
- 19.17. Napisz w pseudojęzyku algorytm reprezentujący procedurę ustalania kolejności złączeń za pomocą programowania dynamicznego.
- 19.18. Oblicz funkcje kosztu dla różnych opcji wykonania operacji JOIN OP7, omówionej w podrozdziale 19.4.
- 19.19. Opracuj wzory dla algorytmu hybrydowego złączenia mieszającego na obliczanie rozmiaru bufora dla pierwszego pakietu. Opracuj ściślejsze wzory szacowania kosztu dla tego algorytmu.
- 19.20. Oszacuj koszt operacji OP6 i OP7 przy użyciu wzorów opracowanych w ćwiczeniu 19.19.
- 19.21. Porównaj koszt dwóch różnych planów zapytań dla następującego zapytania:

$\sigma_{\text{PENSJA} > 4000}(\text{PRACOWNIK} \bowtie_{\text{NRDZ} = \text{NUMERDZ}} \text{DZIAŁ})$

Użyj statystyk bazodanowych z ćwiczenia 19.6.

## Wybrane publikacje

Znajdziesz tu publikacje dotyczące przetwarzania i optymalizacji zapytań. Algorytmy i strategie przetwarzania zapytań opisano w poprzednim rozdziale, trudno jest jednak oddzielić literaturę poświęconą tym zagadnieniom od prac z zakresu optymalizacji. Dlatego bibliografia z tych dziedzin została połączona.

Szczegółowy algorytm optymalizacji metodą algebry relacyjnej przedstawia pozycja Smitha i Chana (1975). Praca doktorska Kooi (1980) prezentuje podstawy technik optymalizacji zapytań. Praca Jarkego i Kocha (1984) prezentuje taksonomię technik optymalizacji zapytań i zawiera bibliografię prac poświęconych temu zagadnieniu. W pracy Graefego (1993) omówiono optymalizację zapytań w systemach bazodanowych i zawarto bogatą bibliografię.

Whang (1985) omawia optymalizację zapytań w systemie OBE (Office-By-Example), który jest oparty na systemie QBE. Optymalizację kosztową wprowadzono w eksperymentalnym SZBD SYSTEM R i omówiono go w pracy Astrahana i in. (1976). Selinger i in. (1979) w klasycznym tekście omawia optymalizację złączeń wielokierunkowych w systemie SYSTEM R. Algorytmy złączeniowe omawia pozycja Gotlieba (1975), Blasgena i Eswarana (1976) oraz Whanga i in. (1982). Algorytmy mieszające w implementacji złączeń opisano i przeanalizowano między innymi w pracy DeWitta i in. (1984), Bratbergsengena (1984), Shapiro (1986), Kitsuregawy i in. (1989) oraz Blakeleya i Martina (1990). Blakeley i in. (1986) opisali utrzymywanie perspektyw zmaterializowanych. Chaudhari i in. (1995) omówili optymalizację zapytań z użyciem perspektyw zmaterializowanych. Metody znajdowania poprawnych kolejności złączeń prezentują pozycje Ioannidisa i Kanga (1990) oraz Swamiego i Gupty (1989). Omówienie skutków stosowania drzew złączeń lewostronnie zagłębionych i drzew krzaczastych zawiera praca Ioannidisa i Kanga (1991). Kim (1982) omawia przekształcenia zagnieżdżonych zapytań SQL do postaci reprezentacji kanonicznych. Optymalizację funkcji agregujących omawia pozycja Kluga (1982) i Muralikrishna'ego (1992). Chaudhari i Shim (1994) opisali optymalizację zapytań z instrukcją GROUP BY. Yan i Larson (1995) omówili wczesną i leniwą agregację. Salzberg I in. (1990) opisuje szybki algorytm sortowania zewnętrznego. Szacowanie rozmiaru relacji tymczasowych ma podstawowe znaczenie dla optymalizacji zapytań. Schematy szacowania oparte na próbkowaniu przedstawiono w pracy Hasa i in. (1995), Hasa i Swamiego (1995) oraz Liptona i in. (1990). Wykorzystanie repozytorium bazodanowego i bardziej szczegółowych statystyk w postaci histogramów stanowi temat pracy Muralikrishna'ego i DeWitta (1988) oraz Poosala'ego i in. (1996). Galindo-Legaria i Joshi (2001) opisali optymalizację podzapytań zagnieżdżonych i agregacji.

O'Neil i Graefe (1995) omówili złączenia wielu tabel z użyciem indeksów bitmapowych. Kim i in. (1985) omawia zaawansowane zagadnienia z zakresu optymalizacji zapytań. Semantyczną optymalizację zapytań omawia pozycja Kinga (1981) oraz Malleya i Zdonicka (1986). Wyniki prac nad semantyczną optymalizacją zapytań można znaleźć w pozycji Chakravarthy'ego i in. (1990), Shenoya i Ozsoyoglu (1989) oraz Siegela i in. (1992). Graefe i McKenna (1993) opracowali optymalizator zapytań Volcano oparty na regułach równoważności zapytań. Podejście z Volcano i późniejszego narzędzia Cascades (Graefe, 1995) jest podstawą optymalizacji zapytań w systemie SQL Server Microsoftu. Carey i Kossman (1998) oraz Bruno i in. (2002) zaprezentowali techniki optymalizowania zapytań dotyczących  $k$  pierwszych wyników. Galindo-Legaria i in. (2004) opisali przetwarzanie i optymalizowanie aktualizacji w bazach danych.

Ahmed i in. (2006) omówili kosztowe przekształcanie zapytań w bazach Oracle i zaprezentowali wartościowy przegląd architektury globalnej optymalizacji zapytań w systemie Oracle 10g. Ziaudin i in. (2008) opisali wymuszanie modyfikowania przez optymalizator planu wykonywania zapytania. Objaśnili mechanizm SPM z baz Oracle stabilizujący wydajność. Bellamkonda i in. (2009) przedstawili dodatkowe techniki optymalizacji zapytań. Ahmed i in. (2014) przeanalizowali zalety drzew krzaczastych w porównaniu z innymi sposobami wykonywania zapytań. Witkowski i in. (2003) opisali obsługę obliczeń z wykorzystaniem tablic  $N$ -wymiarowych w dziedzinie analityki; funkcja ta została zintegrowana z silnikiem relacyjnego SZBD Oracle.



# IX

---

**Przetwarzanie transakcji, sterowanie  
współbieżne i odtwarzanie baz danych**





## Wprowadzenie do problematyki i teorii przetwarzania transakcji

Pojęcie *transakcji* stanowi mechanizm opisu logicznych jednostek przetwarzania danych w bazie. Systemy **przetwarzania transakcji** (ang. *transaction processing systems*) to systemy dużych baz danych z setkami równolegle pracujących użytkowników, którzy wykonują transakcje w tych bazach danych. Przykładami są tu systemy rezerwacji biletów lotniczych, systemy bankowe, systemy przetwarzania kart kredytowych, sklepy internetowe, giełdy, kasy w supermarketach i wiele innych podobnych systemów. Wymagają one wysokiej dostępności i krótkich czasów odpowiedzi dla setek równolegle pracujących użytkowników. W niniejszym rozdziale zostaną przedstawione pojęcia związane z systemami przetwarzania transakcji. Zostanie zdefiniowane pojęcie samej transakcji, służącej do reprezentowania logicznej jednostki przetwarzania w bazie danych, która musi zostać wykonana w całości, aby móc zapewnić poprawność działania systemu. Transakcje są zwykle implementowane w programie komputerowym obejmującym kierowane do baz polecenia (np.: pobierania, wstawiania, usuwania i aktualizowania danych). Niektóre podstawowe techniki programowania baz danych przedstawiono w rozdziałach 10. i 11.

W tym rozdziale skoncentrujemy się na podstawowych pojęciach i teoriach potrzebnych do zapewnienia poprawnego wykonywania transakcji. Zostaną omówione problemy sterowania współbieżnego, które występują w sytuacji, gdy wiele transakcji wygenerowanych przez wielu użytkowników koliduje ze sobą w sposób powodujący generowanie niepoprawnych wyników. Zostanie również omówiona kwestia powrotu do stanu normalnego w razie niepowodzenia transakcji.

Oto struktura rozdziału: w podrozdziale 20.1 w nieformalny sposób zostanie omówione, dlaczego techniki sterowania współbieżnego i odtwarzania są konieczne w systemach bazodanowych. W podrozdziale 20.2 zostanie wprowadzone pojęcie *transakcji* i omówimy dodatkowe pojęcia związane z ich przetwarzaniem w systemach bazodanowych. Podrozdział 20.3 zostanie poświęcony pojęciom niepodzielności, zachowania spójności, izolacji oraz trwałości — tak zwanym właściwościom ACID — których zachowanie jest pożądane w przypadku przetwarzania transakcji. W podrozdziale 20.4 zostanie wprowadzone pojęcie harmonogramów (historii) wykonywania transakcji oraz opisane mechanizmy *odtworzenia* harmonogramów. Podrozdział 20.5 stanowi omówienie pojęcia *szeregowania* równolegle wykonywanych transakcji, które może być używane w celu definiowania poprawnych sekwencji (harmonogramów) wykonania współbieżnych transakcji. W podrozdziale 20.6 Czytelnik znajdzie omówienie poleceń obsługi transakcji w języku SQL; wprowadzimy tu także pojęcie poziomów izolacji. Podrozdział 20.7 zawiera podsumowanie rozdziału.

Dwa kolejne rozdziały będą stanowiły bardziej szczegółowe omówienie metod i technik używanych w celu obsługi przetwarzania transakcyjnego. Rozdział 21. będzie opisywał podstawowe techniki sterowania współbieżnego, zaś w rozdziale 22. zostanie przedstawione ogólne omówienie technik odtwarzania baz danych.

## 20.1. Wprowadzenie do problematyki przetwarzania transakcji

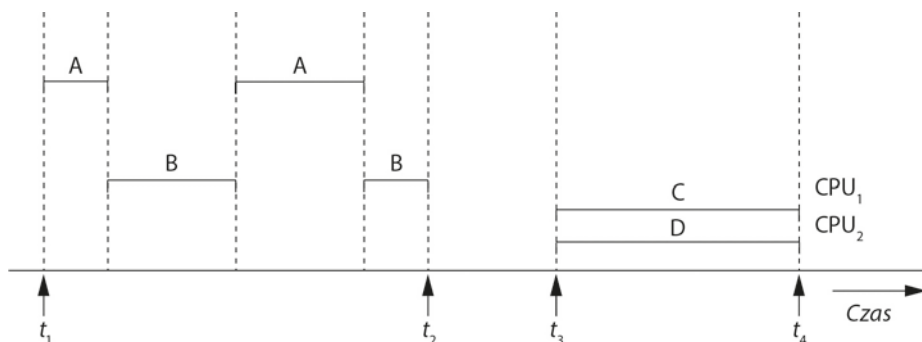
W niniejszym podrozdziale zostaną omówione pojęcia współbieżnego wykonywania transakcji oraz odtwarzania w razie niepowodzenia transakcji. W podrozdziale 20.1.1 zostanie przedstawione porównanie systemów baz danych jedno- i wieloużytkownikowych oraz omówienie kwestii tego, w jaki sposób przebiega współbieżne wykonywanie transakcji w systemach wieloużytkownikowych. W podrozdziale 20.1.2 zostanie zdefiniowane pojęcie transakcji i przedstawiony prosty model ich wykonywania w oparciu o operacje odczytu i zapisu w bazach danych, który jest używany w celu zdefiniowania i sformalizowania mechanizmów sterowania współbieżnego oraz pojęć dotyczących odtwarzania. W podrozdziale 20.1.3 zostaną przedstawione nieformalne przykłady, dowodzące, że techniki przetwarzania współbieżnego są potrzebne w przypadku systemów wieloużytkownikowych. Wreszcie w podrozdziale 20.1.4 zostanie wyjaśnione, dlaczego wymagane jest opracowanie odpowiednich technik, pozwalających na odtwarzanie baz w przypadku awarii systemu i niepowodzeń transakcji; Czytelnik znajdzie w nim omówienie różnych przypadków niepowodzenia transakcji w czasie ich wykonywania.

### 20.1.1. Systemy jedno- i wieloużytkownikowe

Jednym z kryteriów klasyfikowania systemów bazodanowych jest dokonywanie tego według liczby użytkowników, którzy mogą z systemu korzystać **współbieżnie** (ang. *concurrently*), czyli w tym samym czasie. SZBD jest **jednoużytkownikowy** (ang. *single-user*), jeżeli w danym momencie może z niego korzystać tylko jeden użytkownik, natomiast w przypadku, gdy z systemu może korzystać wielu użytkowników (a więc również uzyskiwać dostęp do bazy danych) współbieżnie, jest on systemem **wieloużytkownikowym** (ang. *multiuser*). Systemy jednoużytkownikowe są w dużej mierze ograniczone do systemów używanych w komputerach osobistych. Większość pozostałych SZBD to systemy wieloużytkownikowe. Przykładowo, system rezerwacji biletów lotniczych jest współbieżnie używany przez setki użytkowników i pracowników biur podróży. Systemy używane w bankach, agencjach ubezpieczeniowych, na giełdach, w supermarketach i innych podobnych instytucjach również są wykorzystywane przez wielu użytkowników. W takich systemach zwykle setki lub tysiące użytkowników operują na bazie danych i jednocześnie generują transakcje.

Wielu użytkowników może uzyskiwać dostęp do bazy danych (i używać systemów komputerowych) równocześnie dzięki pojęciu **wieloprogramowości** (ang. *multiprogramming*), które pozwala pojedynczemu komputerowi na wykonywanie w tym samym czasie wielu programów (lub **procesów**). Jeden procesor główny (CPU) może w danym momencie

wykonywać najwyżej jeden proces. Jednak **wieloprogramowe systemy operacyjne** wykonują pewne polecenia związane z jednym procesem, następnie zawieszają wykonywanie tego procesu i wykonują polecenia kolejnego procesu itd. Proces zostaje wznowiony w miejscu jego zawieszenia, kiedy znów nadejdzie jego kolej wykorzystania procesora. Stąd współbieżne wykonywanie procesów polega w rzeczywistości na **przeplataniu** (ang. *interleave*), co przedstawia rysunek 20.1, na którym dwa procesy A i B są wykonywane współbieżnie dzięki przeplotowi. Przeplot powoduje, że procesor pracuje także wówczas, gdy proces wymaga wykonania operacji wejścia-wyjścia (I/O), takich jak odczyt bloku z dysku. Procesor jest przełączany na wykonywanie kolejnego procesu, zamiast oczekiwać bezczynnie na zakończenie operacji I/O. Przeplot zapobiega również opóźnianiu procesów przez inne, długotrwałe procesy.



RYSUNEK 20.1. Przetwarzanie z przeplotem a przetwarzanie równoległe transakcji współbieżnych

Jeżeli system komputerowy posiada kilka procesorów sprzętowych, możliwe jest **równoległe przetwarzanie** (ang. *parallel processing*) wielu procesów, co przedstawiono w postaci procesów C i D na rysunku 20.1. Większość zagadnień teoretycznych dotyczących sterowania współbieżnego w bazach danych opracowano w kontekście **współbieżności z przeplotem** (ang. *interleaved concurrency*), więc w dalszej części niniejszego rozdziału przyjmujemy właśnie taki model. W przypadku wieloużytkownikowych SZBD przechowywane dane stanowią główne zasoby, do których można uzyskiwać współbieżny interaktywny dostęp za pomocą aplikacji stale pobierających informacje z bazy danych i modyfikujących je.

## 20.1.2. Transakcje, elementy baz danych, operacje odczytu i zapisu oraz bufory SZBD

**Transakcja** (ang. *transaction*) jest wykonywanym programem, który tworzy logiczną jednostkę przetwarzania w bazie danych. Transakcja składa się z jednej lub wielu operacji dostępu do bazy danych — może tu chodzić o operacje wstawiania, usuwania, modyfikowania (aktualizowania) i pobierania danych. Operacje bazodanowe tworzące transakcję mogą być albo zawarte w programie aplikacji, albo określone interaktywnie za pomocą języka wysokiego poziomu, takiego jak SQL. Jednym ze sposobów określania zakresu transakcji jest wyróżnienie jawnych instrukcji **begin transaction** oraz **end transaction** w programie aplikacji. W takim przypadku wszystkie operacje dostępu do bazy danych występujące między

tymi instrukcjami są traktowane jako tworzące jedną transakcję. Jeden program aplikacji może zawierać wiele transakcji, jeżeli określono kilka zakresów transakcji. Jeśli operacje bazodanowe należące do transakcji nie aktualizują bazy danych, a tylko pobierają dane, o transakcji mówi się, że jest **transakcją tylko do odczytu** (ang. *read-only transaction*). Pozostałe transakcje są **transakcjami do odczytu i zapisu** (ang. *read-write transaction*).

*Model bazy danych* używany w celu wyjaśnienia pojęć dotyczących przetwarzania transakcji jest prosty w porównaniu z modelami omawianymi wcześniej w książce (np. relacyjnym lub obiektowym). **Baza danych** jest zasadniczo reprezentowana jako zbiór *nazwanych elementów danych*. Rozmiar elementu danych określa się mianem jego **ziarnistości** (ang. *granularity*). **Elementem danych** może być *rekord w bazie danych* lub większa jednostka, taka jak cały *blok dyskowy*, a także mniejsze jednostki — np. *wartość pojedynczego pola (atrybutu)* rekordu z bazy danych. Jednak omawiane pojęcia są niezależne od ziarnistości elementów danych; dotyczą one wszystkich elementów danych. Każdy taki element ma *unikatową nazwę*, przy czym zwykle nie jest ona używana przez programistę. Nazwa ta umożliwia *unikatowe zidentyfikowanie każdego elementu danych*. Przykładowo, jeśli element danych to jeden blok dyskowy, jako jego nazwę można wykorzystać adres tego bloku. Gdy używany jest jeden rekord, nazwą może być identyfikator rekordu. W przypadku użycia uproszczonego modelu bazy danych podstawowe operacje dostępu do bazy, które mogą należeć do transakcji, to:

- `odczytaj_element(X)`: wczytuje element bazy danych o nazwie *X* do zmiennej programowej. W celu uproszczenia naszej notacji zakładamy, że *zmienna programowa również nosi nazwę X*.
- `zapisz_element(X)`: zapisuje wartość zmiennej programowej *X* w elemencie bazy danych o nazwie *X*.

Jak stwierdzono w rozdziale 16., podstawową jednostką transferu danych z dysku do pamięci głównej jest jeden blok (jedna strona). Wykonanie polecenia `odczytaj_element(X)` wiąże się z wykonaniem następujących działań:

- (1) Znalezienie adresu bloku dyskowego, który zawiera element *X*.
- (2) Skopiowanie bloku dyskowego do bufora w pamięci głównej (o ile blok ten nie znajduje się już w którymś z buforów pamięci głównej). Wielkość bufora jest równa wielkości bloku dysku.
- (3) Skopiowanie elementu *X* z bufora do zmiennej programowej o nazwie *X*.

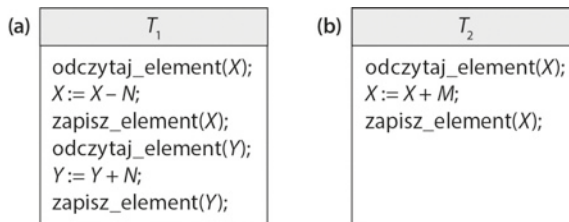
Wykonanie polecenia `zapisz_element(X)` wiąże się z wykonaniem następujących działań:

- (1) Znalezienie adresu bloku dyskowego, który zawiera element *X*.
- (2) Skopiowanie tego bloku dyskowego do bufora w pamięci głównej (o ile blok ten nie znajduje się już w którymś z buforów pamięci głównej).
- (3) Skopiowanie elementu *X* ze zmiennej programowej o nazwie *X* do odpowiedniej lokalizacji w buforze.
- (4) Przeniesienie zaktualizowanego bloku z bufora z powrotem na dysk (od razu lub z pewnym opóźnieniem).

Etap 4. jest tym, który dokonuje faktycznej aktualizacji bazy danych na dysku. W pewnych przypadkach bufor nie jest od razu zapisywany na dysku, na wypadek, gdyby miały w nim być dokonywane dodatkowe zmiany. Zazwyczaj decyduje o tym, kiedy należy zapisać zmo-

dyfikowany blok dyskowy znajdujący się w buforze pamięci głównej, podejmuje menedżer odtwarzania systemu zarządzania bazą danych w porozumieniu z odpowiednimi mechanizmami systemu operacyjnego. SZBD zazwyczaj przechowuje w pamięci głównej **pamięć podręczną bazy danych** z wieloma **buforami danych**. Każdy bufor zwykle obejmuje zawartość jednego bloku bazy danych z dysku, a blok ten zawiera przetwarzane elementy bazy danych. W momencie, gdy wszystkie te bufora zostaną zajęte, a do pamięci musi zostać skopiowany kolejny blok bazy danych, wykorzystywana jest pewna **strategia zastępowania buforów** w celu wybrania buforów, które zostaną zastąpione. Popularną strategią zastępowania buforów jest **LRU** (ang. *least recently used*, czyli najdłużej nieużywany). Jeśli wybrany bufor był wcześniej modyfikowany, musi zostać zapisany z powrotem na dysku, nim będzie mógł być wykorzystany ponownie<sup>1</sup>. Istnieją też strategie zastępowania buforów specyficzne dla SZBD. Kilka z nich pokrótce omówimy w punkcie 20.2.4.

Transakcja zawiera operacje `odczytaj_element` i `zapisz_element`, które służą uzyskiwaniu dostępu do bazy danych i jej aktualizowaniu. Na rysunku 20.2 przedstawiono przykłady dwóch bardzo prostych transakcji. **Zbiór odczytu** (ang. *read-set*) transakcji to zbiór wszystkich elementów, które są odczytywane przez transakcję, zaś **zbiór zapisu** (ang. *write-set*) to zbiór wszystkich elementów zapisywanych przez transakcję. Przykładowo, zbiór odczytu  $T_1$  z rysunku 20.2 ma postać  $\{X, Y\}$ , zaś zbiór zapisu również ma postać  $\{X, Y\}$ .



RYСУNEK 20.2. Dwie przykładowe transakcje. (a) Transakcja  $T_1$ ; (b) Transakcja  $T_2$

Mechanizmy sterowania współbieżnością oraz odtwarzania w głównej mierze są związane z poleceniami dostępu do bazy danych w transakcjach. Transakcje wygenerowane przez różnych użytkowników mogą być wykonywane współbieżnie i aktualizować te same elementy danych. Jeżeli takie współbieżne wykonywanie *nie jest kontrolowane*, może to prowadzić do problemów, takich jak niespójność bazy danych. W kolejnym podrozdziale zostaną nieformalnie przedstawione niektóre ze spotykanych problemów.

### 20.1.3. Uzasadnienie potrzeby stosowania sterowania współbieżnego

W przypadku wykonywania transakcji w niekontrolowany sposób można napotkać kilka problemów. Poniżej zilustrujemy je na przykładzie znacznie uproszczonej bazy rezerwacji biletów lotniczych, w której dla każdego lotu jest przechowywany jeden rekord. Każdy rekord zawiera między innymi *liczbę zarezerwowanych miejsc* na dany lot jako *nazwany*

<sup>1</sup> Nie będziemy w tym miejscu omawiać ogólnych strategii zastępowania buforów, gdyż zwykle omawiają je podręczniki poświęcone systemom operacyjnym.

(z możliwością jednoznacznej identyfikacji) *element danych*. Na rysunku 20.2(a) przedstawiono transakcję  $T_1$ , która *przenosi*  $N$  rezerwacji z jednego lotu, którego liczba zarezerwowanych miejsc jest przechowywana w elemencie bazy danych o nazwie  $X$ , do innego lotu, którego liczba zarezerwowanych miejsc jest przechowywana w elemencie bazy danych o nazwie  $Y$ . Na rysunku 20.2(b) przedstawiono prostszą transakcję  $T_2$ , która jedynie *rezerwuje*  $M$  miejsc na pierwszy lot ( $X$ ), występujący w transakcji  $T_1$ <sup>2</sup>. W celu uproszczenia przykładu nie prezentujemy dodatkowych części transakcji, takich jak sprawdzanie, czy lot posiada wystarczającą liczbę wolnych miejsc, przed zarezerwowaniem dodatkowych.

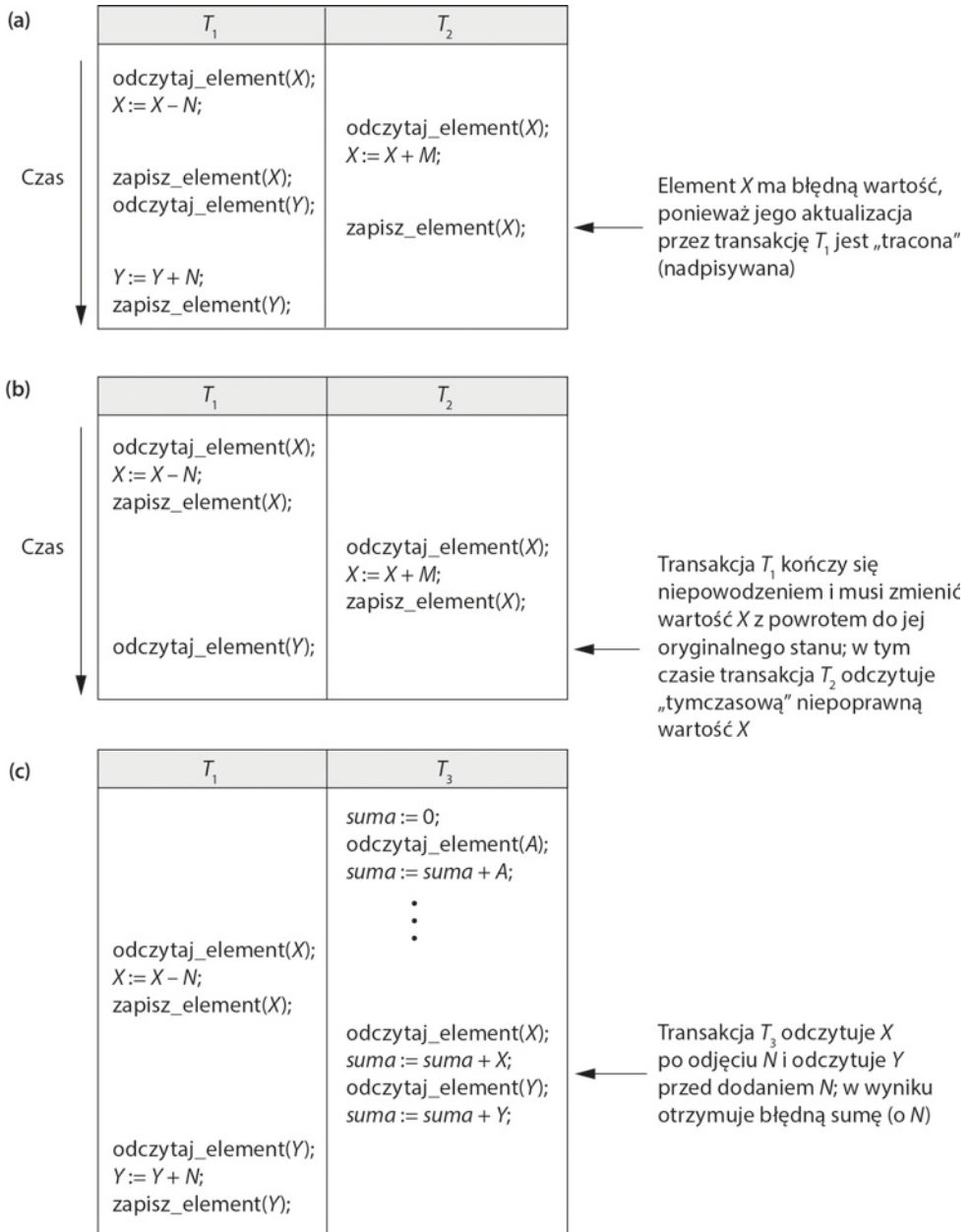
Kiedy zostanie napisany program uzyskujący dostęp do bazy danych, numery lotów, ich daty oraz liczby miejsc możliwych do zarezerwowania są określane jako parametry. Stąd ten sam program może być używany w celu wykonywania *wielu różnych transakcji*, z których każda dotyczy innego lotu i różnej liczby rezerwowanych miejsc. Dla celów sterowania współbieżnego transakcję traktuje się jako *określone wykonanie* programu dla konkretnej daty, lotu oraz liczby miejsc. Na rysunkach 20.2(a) i (b) transakcje  $T_1$  oraz  $T_2$  stanowią *określone wykonania* programów, które odwołują się do określonych lotów o liczbach miejsc przechowywanych w bazie w elementach danych  $X$  i  $Y$ . Poniżej zostaną omówione rodzaje problemów, jakie mogą wystąpić w przypadku takich dwóch transakcji wykonywanych współbieżnie.

**Problem utraconej aktualizacji.** Problem ten występuje w przypadku, gdy dwie transakcje uzyskujące dostęp do tych samych elementów bazy danych są związane z przeplothem ich operacji w sposób sprawiający, że wartości pewnych elementów bazy danych stają się błędne. Załóżmy, że transakcje  $T_1$  i  $T_2$  uruchomiono mniej więcej w tym samym momencie, oraz przyjmijmy, że ich działanie podlega przepłutowi, tak jak pokazano to na rysunku 20.3(a). W takim przypadku końcowa wartość elementu  $X$  jest niepoprawna, ponieważ transakcja  $T_2$  odczytuje wartość  $X$  *zanim* transakcja  $T_1$  zmieni ją w bazie danych, a stąd zaktualizowana wartość będąca wynikiem działania transakcji  $T_1$  zostaje utracona. Przykładowo, jeżeli na początku  $X = 80$  (początkowo istniało 80 rezerwacji na lot),  $N = 5$  (transakcja  $T_1$  przenosi 5 rezerwacji miejsc z lotu  $X$  na lot  $Y$ ) oraz  $M = 4$  (transakcja  $T_2$  rezerwuje 4 miejsca na lot  $X$ ), to końcowym wynikiem powinno być  $X = 79$ , jednak w wyniku przepłotu operacji przedstawionych na rysunku 20.3(a)  $X = 84$ , ponieważ aktualizacja transakcji  $T_1$  usuwająca pięć miejsc z lotu  $X$  została *utracona*.

**Problem aktualizacji tymczasowej (odczytu zmodyfikowanego).** Problem ten występuje wówczas, gdy jedna transakcja aktualizuje element bazy danych, a następnie transakcja ta z pewnego powodu kończy się niepowodzeniem (patrz punkt 20.1.4). Do zaktualizowanego elementu uzyskuje dostęp inna transakcja, zanim zostanie on zmieniony z powrotem na swoją oryginalną wartość. Na rysunku 20.3(b) przedstawiono przykład, gdzie transakcja  $T_1$  aktualizuje element  $X$ , a następnie ulega uszkodzeniu przed swoim zakończeniem, więc system musi zmienić wartość  $X$  z powrotem do jej oryginalnej postaci. Jednak zanim zostanie to zrobione, transakcja  $T_2$  odczytuje *tymczasową* wartość  $X$ , która nie zostanie na stałe zachowana w bazie danych ze względu na niepowodzenie transakcji  $T_1$ . Wartość elementu  $X$  odczytana przez transakcję  $T_2$  nosi nazwę *danych zmodyfikowanych* (ang. *dirty*

<sup>2</sup> W przypadku podobnego, częściej używanego przykładu, zakłada się wykorzystanie bazy danych banku z jedną transakcją, dokonującą przelewu pewnej kwoty z konta  $X$  na konto  $Y$ , oraz drugą transakcją, dokonującą wpłaty na konto  $X$ .





RYSTONEK 20.3. Niektóre problemy występujące w przypadku, gdy wykonywanie współbieżne nie podlega kontroli. (a) Problem utraconej aktualizacji. (b) Problem aktualizacji tymczasowej. (c) Problem błędnego podsumowania

data), ponieważ została ustalona przez transakcję, która nie została zakończona i zatwierdzona. Dlatego problem ten określa się również mianem *problemu odczytu zmodyfikowanego* (ang. *dirty read problem*).



**Problem błędnej sumy.** Jeżeli jedna transakcja oblicza wartość funkcji agregującej podsumowania na wielu rekordach, kiedy inna transakcja aktualizuje niektóre z tych rekordów, funkcja agregująca może obliczyć pewne wartości przed tym, jak zostaną zaktualizowane, a inne już po ich aktualizacji. Przykładowo, załóżmy, że transakcja  $T_3$  oblicza całkowitą liczbę rezerwacji na wszystkie loty. W tym samym czasie jest wykonywana transakcja  $T_1$ . Jeżeli wystąpią operacje podlegające przeplotowi z rysunku 20.3(c), wynik transakcji  $T_3$  będzie błędny o wartość  $N$ , ponieważ transakcja  $T_3$  odczyta wartość  $X$  po tym, jak zostanie odjęte  $N$  miejsc, ale wartość  $Y$  zostanie odczytana przed dodaniem  $N$  miejsc.

**Problem odczytu niepowtarzalnego.** Kolejny problem, jaki może wystąpić, to **odczyt niepowtarzalny** (ang. *unrepeatable read*), gdzie transakcja  $T$  odczytuje element dwukrotnie i element ten zostaje zmieniony przez inną transakcję  $T'$  między tymi dwoma odczytami. Stąd transakcja  $T$  otrzymuje *różne wartości* dla swoich dwóch odczytów tego samego elementu. Może tak się zdarzyć na przykład wówczas, gdy w czasie transakcji rezerwowania miejsc klient wykona zapytanie odnośnie do dostępności miejsc na kilka lotów. Kiedy klient zdecyduje się na określony lot, transakcja ponownie odczytuje liczbę miejsc na dany lot przed zakończeniem rezerwacji. Może wtedy wczytać inną wartość.

## 20.1.4. Uzasadnienie potrzeby odtwarzania

Kiedy transakcja zostaje przekazana do wykonania przez SZBD, staje się on odpowiedzialny za zapewnienie, aby albo wszystkie operacje wykonywane w ramach transakcji zostały zakończone z powodzeniem i efekty ich działania zostały na stałe zapisane w bazie danych, albo transakcja nie miała wpływu na bazę danych i inne transakcje. W pierwszym przypadku mówimy, że transakcja została **zatwierdzona**, a w drugim scenariuszu transakcja jest **anulowana**. SZBD nie może dopuścić do sytuacji, w której pewne operacje transakcji  $T$  zostaną uwzględnione w bazie danych, a inne nie. To *cała transakcja* jest logiczną jednostką przetwarzania w bazie danych. Jeśli transakcja **zawiedzie** po wykonaniu tylko niektórych operacji, operacje te trzeba wycofać i nie mogą one powodować trwałych skutków.

**Rodzaje awarii.** Występujące awarie, ogólnie rzecz biorąc, dzieli się na awarie transakcyjne, systemowe i nośników. Można wyróżnić kilka powodów niepowodzenia transakcji w trakcie jej wykonywania.

- (1) **Awaria komputera (załamanie systemu).** W systemie występuje błąd sprzętowy, programowy lub sieciowy w czasie wykonywania transakcji. Awarie sprzętowe są zwykle awariami nośników — na przykład pamięci głównej.
- (2) **Błąd transakcyjny lub systemowy.** Pewna operacja w ramach transakcji może spowodować jej uszkodzenie, na przykład przepełnienie wartości całkowitej lub dzielenie przez zero. Uszkodzenie transakcji może również wystąpić ze względu na błędną wartość parametru lub logiczny błąd programistyczny<sup>3</sup>. Ponadto użytkownik może przerwać transakcję w czasie jej wykonywania.
- (3) **Błędy lokalne lub wyjątki wykryte przez transakcję.** W czasie wykonywania transakcji mogą wystąpić pewne czynniki wymuszające jej anulowanie, na przykład wymagane przez nią dane mogą okazać się niedostępne. Należy zauważyć,

---

<sup>3</sup> Ogólnie rzecz biorąc, transakcja powinna zostać dokładnie przetestowana w celu zapewnienia, aby nie zawierała logicznych błędów programistycznych (ang. *bugs*).

że wyjątek<sup>4</sup>, taki jak niewystarczający stan konta w bankowej bazie danych, może spowodować, że transakcja w rodzaju wypłaty z konta zostanie anulowana. Taki wyjątek można zaprogramować w samej transakcji i wówczas nie stanowi on sytuacji awaryjnej.

- (4) **Wymuszenie sterowania współbieżnego.** Wykorzystywana metoda sterowania współbieżnego (patrz rozdział 21.) może zdecydować o anulowaniu transakcji, kiedy narusza warunki szeregowania (patrz podrozdział 20.5) lub kiedy kilka transakcji znajdzie się w stanie zakleszczenia (patrz punkt 21.1.3). Transakcje anulowane z powodu naruszenia warunków szeregowania lub zakleszczeń są zwykle później automatycznie uruchamiane ponownie.
- (5) **Awaria dysku.** Pewne bloki dyskowe mogą utracić zapisane na nich dane ze względu na błąd operacji odczytu lub zapisu albo awarię głowicy odczytująco-zapisującej. Może się to zdarzyć w czasie operacji odczytu lub zapisu w ramach transakcji.
- (6) **Problemy i katastrofy fizyczne.** Chodzi tu o bardzo długą listę problemów, do których zaliczamy utratę zasilania lub awarię systemu chłodzenia, pożar, włamanie, sabotaż, omyłkowe nadpisanie dysków lub taśm bądź zamontowanie przez operatora nieodpowiedniej taśmy.

Awarie typu 1., 2., 3. i 4. występują częściej niż awarie typu 5. lub 6. W razie wystąpienia którejś z tych pierwszych system musi zachować wystarczającą ilość informacji, aby można było przeprowadzić odtworzenie do stanu normalnego. Awarie dyskowe i inne zdarzenia katastroficzne nie występują zbyt często, jeśli jednak tak się zdarzy, głównym zadaniem jest właśnie przeprowadzenie odtwarzania. Zagadnienia te stanowią treść rozdziału 22.

Pojęcie transakcji ma fundamentalne znaczenie w przypadku wielu technik sterowania współbieżnego oraz odtwarzania po awarii.

## 20.2. Pojęcia dotyczące transakcji i systemu

W niniejszym podrozdziale zostaną omówione dodatkowe pojęcia mające związek z przetwarzaniem transakcji. W punkcie 20.2.1 zostaną opisane różne stany, w jakich może się znaleźć transakcja, oraz zostaną omówione dodatkowe operacje potrzebne w procesie przetwarzania transakcji. W punkcie 20.2.2 zostanie omówiony dziennik systemowy, w którym są przechowywane informacje potrzebne w procesie odtwarzania. Wreszcie w punkcie 20.2.3 zostanie opisane pojęcie punktów zatwierdzenia oraz kwestia ich istotności w zakresie przetwarzania transakcji. Punkt 20.2.4 zawiera krótkie omówienie strategii zastępowania buforów w SZBD.

### 20.2.1. Stany transakcji i dodatkowe operacje

Transakcja jest niepodzielną jednostką działań, które albo muszą zostać wykonane w całości, albo w ogóle. Dla celów odtwarzania system musi śledzić, kiedy transakcje są inicjowane i kończone, oraz zatwierdzać je lub anulować (patrz podrozdział 20.2.3). Stąd menedżer odtwarzania SZBD śledzi następujące operacje:

---

<sup>4</sup> Wyjątki, jeśli są poprawnie oprogramowane, nie powodują uszkodzenia transakcji.

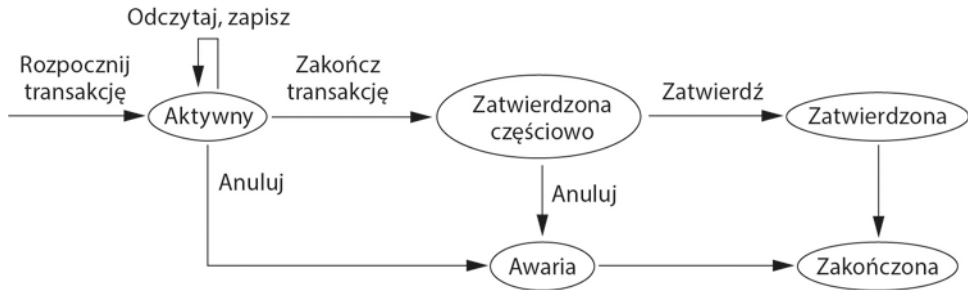
- **ROZPOCZNIJ\_TRANSAKCJĘ** (**BEGIN\_TRANSACTION**). Jest to oznaczenie początku wykonywania transakcji.
- **ODCZYTAJ** (**READ**) lub **ZAPISZ** (**WRITE**). Są to operacje odczytu i zapisu przeprowadzane na elementach bazy danych, wykonywane jako część transakcji.
- **ZAKOŃCZ\_TRANSAKCJĘ** (**END\_TRANSACTION**). Jest to oznaczenie sytuacji, w której wszystkie operacje **ODCZYTAJ** i **ZAPISZ** zostały zakończone oraz zakończono wykonywanie transakcji. Jednak w tym momencie konieczne może okazać się sprawdzenie, czy zmiany wprowadzone przez transakcję mogą zostać na stałe zastosowane względem bazy danych (zatwierdzone) lub czy transakcja musi zostać anulowana ze względu na naruszenie reguł szeregowania (patrz podrozdział 20.5), bądź z innego powodu.
- **ZATWIERDŹ\_TRANSAKCJĘ** (**COMMIT\_TRANSACTION**). Jest to oznaczenie *udanego zakończenia* transakcji w taki sposób, że wszelkie zmiany (aktualizacje) dokonane przez transakcję mogą być bezpiecznie **zatwierdzone** (ang. *committed*) w bazie danych i nie zostaną wycofane.
- **WYCOFAJ** (**ROLLBACK**), inaczej **ANULUJ** (**ABORT**). Jest to oznaczenie *nieudanego zakończenia* transakcji w taki sposób, że wszelkie zmiany spowodowane przez transakcję w bazie danych muszą zostać **wycofane**.

Na rysunku 20.4 przedstawiono diagram przejść stanów opisujący, w jaki sposób transakcja przechodzi przez swoje stany wykonania. Transakcja przechodzi do **stanu aktywnego** (ang. *active state*) tuż po rozpoczęciu wykonywania i mogą wówczas być wykonywane operacje **ODCZYTAJ** i **ZAPISZ**. W momencie kończenia transakcji przechodzi ona do **stanu zatwierdzenia częściowego** (ang. *partially committed state*). Na tym etapie protokół sterowania współbieżnego musi przeprowadzić dodatkowe testy, aby sprawdzić, czy transakcja może zostać zatwierdzona. Ponadto protokół odtwarzania musi zapewnić, aby awaria systemu nie sprawiła, że wprowadzone zmiany nie będą mogły zostać zarejestrowane w sposób trwały (zwykle odbywa się to poprzez rejestrowanie zmian w dzienniku systemowym, co zostanie omówione w kolejnym punkcie)<sup>5</sup>. W razie sukcesu takiego sprawdzenia o transakcji mówi się, że osiągnęła swój punkt zatwierdzenia i przechodzi ona do **stanu zatwierdzenia** (ang. *committed state*). Punkty zatwierdzenia omówiono bardziej szczegółowo w podrozdziale 20.2.3. Po zatwierdzeniu transakcji jej wykonanie jest kończone i wszelkie wprowadzone zmiany muszą zostać trwale zapisane w bazie danych (nawet po wystąpieniu awarii systemu).

Jednak transakcja może również przejść do **stanu awaryjnego** (ang. *failed state*), jeżeli jedno ze sprawdzeń zakończy się niepowodzeniem lub transakcja zostanie anulowana będąc w stanie aktywnym. Transakcja może zostać wówczas wycofana w celu anulowania efektów związanych z wykonanymi operacjami **ZAPISZ** na bazie danych. **Stan zakończenia** (ang. *terminated state*) odpowiada sytuacji, gdy transakcja opuszcza system. Informacje dotyczące transakcji, przechowywane w tabelach systemowych w czasie jej działania, zostają usunięte w momencie jej zakończenia. Transakcje zakończone niepowodzeniem lub anulowane mogą być *ponownie wykonane* w późniejszym okresie — automatycznie lub w wyniku ponownego wywołania przez użytkownika — jako zupełnie nowe transakcje.

---

<sup>5</sup> Optymistyczny mechanizm sterowania współbieżnego (patrz podrozdział 21.4) wymaga również, aby na tym etapie zostały przeprowadzone pewne sprawdzenia zapewniające, że transakcja nie będzie miała wpływu na inne wykonywane transakcje.



RYSUNEK 20.4. Diagram przejść stanów ilustrujący stany związane z wykonywaniem transakcji

## 20.2.2. Dziennik systemowy

W celu umożliwienia odtwarzania po awariach wpływających na transakcje system przechowuje **dziennik**<sup>6</sup> (ang. *log*) w celu śledzenia wszystkich operacji związanych z transakcjami, które mają wpływ na wartości elementów bazy danych. Informacje te mogą być potrzebne w celu umożliwienia odtworzenia po awarii. Dziennik to przechowywany na dysku plik sekwencyjny, do którego można tylko dodawać dane, więc nie mają na niego wpływu żadne awarie oprócz dyskowych lub katastroficznych. Zwykle bufor w pamięci głównej (jeden lub kilka), tzw. bufor dziennika, przechowują najnowszy fragment tego pliku. Dlatego wpisy dziennika są najpierw umieszczane w buforze dziennika w pamięci głównej. Gdy **bufor dziennika** się zapełni lub gdy spełnione zostaną inne warunki, *bufor jest dołączany na końcu pliku dziennika na dysku*. Ponadto dziennik jest okresowo archiwizowany na nośniku kopii zapasowych (taśmie) w celu zabezpieczenia się przed katastroficznymi awariami. Poniżej zostaną wymienione rodzaje wpisów — określanych mianem **rekordów dziennika** (ang. *log records*) — zapisywanych w dzienniku oraz związane z nimi akcje. W przypadku tych wpisów symbol *T* odnosi się do unikatowego **identyfikatora transakcji**, który jest generowany automatycznie przez system i używa się go w celu identyfikowania transakcji:

- (1) [rozpocznij\_transakcję, *T*]. Wskazuje, że transakcja *T* rozpoczęła wykonywanie.
- (2) [zapisz\_element, *T*, *X*, *stara\_wartość*, *nowa\_wartość*]. Wskazuje, że transakcja *T* zmieniała wartość elementu bazy danych *X* z wartości *stara\_wartość* na *nowa\_wartość*.
- (3) [odczytaj\_element, *T*, *X*]. Wskazuje, że transakcja *T* odczytała wartość elementu bazy danych *X*.
- (4) [zatwierdź, *T*]. Wskazuje, że transakcja *T* została zakończona powodzeniem, i potwierdza, że jej efekty mogą zostać zatwierdzone (trwale zapisane) w bazie danych.
- (5) [anuluj, *T*]. Wskazuje, że transakcja *T* została anulowana.

Protokoły odtwarzania unikające kaskadowych operacji wycofania (patrz podrozdział 20.4.2) — czyli niemal wszystkie protokoły — *nie wymagają*, aby operacje odczytu były rejestrowane w dzienniku. Jednakże, jeżeli dziennik jest używany również w innych celach — takich, jak nadzór (śledzenie wszelkich działań wykonywanych w bazie danych) — wówczas można uwzględnić także takie wpisy. Ponadto pewne protokoły odtwarzania

<sup>6</sup> Dziennik (ang. *log*) określa się niekiedy mianem *dziennika SZBD* (ang. *DBMS journal*).

wymagają prostszych wpisów dotyczących operacji zapisu, nie zawierających wartości *nowa\_wartość* (patrz podrozdział 20.4.2).

Należy zauważyć, że zakładamy, iż wszystkie trwałe zmiany w bazie danych występują w transakcjach, więc pojęcie odtworzenia po uszkodzeniu transakcji odnosi się albo do cofnięcia, albo ponowienia operacji transakcji indywidualnie na podstawie dziennika. W razie awarii systemu można dokonać odtworzenia do spójnego stanu bazy danych poprzez sprawdzenie dziennika i wykorzystanie jednej z technik opisanych w rozdziale 22. Ze względu na fakt, że dziennik zawiera rekord dla każdej operacji zapisu, która zmieniła wartość pewnego elementu bazy danych, istnieje możliwość **cofnięcia** efektów takich operacji transakcji *T* poprzez przejrzanie dziennika i odtworzenie wartości wszystkich elementów zmienionych przez operację zapisu transakcji *T* na wartości określone przez parametr *stara\_wartość*. Może również być konieczne **ponowienie** operacji transakcji, jeżeli wszystkie jej aktualizacje zostaną zarejestrowane w dzienniku, ale wystąpi uszkodzenie zanim będzie możliwe upewnienie się, że wszystkie wartości *nowa\_wartość* zostały trwałe zapisane w bazie danych na dysku<sup>7</sup>.

### 20.2.3. Punkt zatwierdzenia transakcji

Transakcja *T* osiąga swój **punkt zatwierdzenia** wówczas, gdy wszystkie jej operacje uzyskujące dostęp do bazy danych zostaną z powodzeniem wykonane oraz wyniki działania wszystkich operacji transakcji zostaną zarejestrowane w dzienniku. Za punktem zatwierdzenia o transakcji mówi się, że została **zatwierdzona**, a jej efekty uznaje się za *trwale zapisane* w bazie danych. Następnie transakcja zapisuje w dzienniku rekord zatwierdzenia [zatwierdź, *T*]. Jeżeli nastąpi awaria systemu, należy przeszukać wstecz dziennik w poszukiwaniu wszystkich transakcji *T*, które zapisały rekord [rozpocznij\_transakcję, *T*] w dzienniku, ale nie zapisały rekordu [zatwierdź, *T*]. Może się okazać, że transakcje te trzeba *wycofać* (ang. *rollback*) w celu usunięcia skutków ich działania na bazie danych w czasie procesu odtwarzania. Transakcje, które zapisały swój rekord zatwierdzenia w dzienniku, musiały również zapisać w nim wszystkie swoje operacje zapisu, więc efekty ich działania na bazie danych można *ponowić* na podstawie rekordów dziennika.

Należy zauważyć, że plik dziennika musi być przechowywany na dysku. Zgodnie z dyskusją zawartą w rozdziale 16., aktualizacja pliku dyskowego wiąże się ze skopiowaniem odpowiedniego bloku pliku z dysku do bufora w pamięci głównej, zaktualizowaniem go tutaj oraz skopiowaniem z bufora na dysk. Jak już wspomniano, często stosuje się rozwiązanie polegające na przechowywaniu jednego lub większej liczby bloków pliku dziennika w buforach pamięci głównej (w **buforach dziennika**) do momentu, aż zostaną zapełnione wpisami dziennika, a następnie na zapisaniu ich z powrotem na dysku tylko raz, zamiast dokonywać zapisu za każdym razem, gdy do dziennika dodawany jest wpis. Pozwala to zaoszczędzić na czasie związanym z wieloma operacjami zapisu na dysku tego samego bloku pliku dziennika. W momencie wystąpienia awarii systemu tylko wpisy dziennika *zapisane z powrotem na dysku* są uwzględniane w procesie odtwarzania, ponieważ zawartość pamięci głównej może zostać utracona. Stąd też *zanim* transakcja osiągnie swój punkt zatwierdzenia, część dziennika nie zapisana dotąd na dysku musi zostać poddana tej operacji. Proces ten nosi nazwę **zapisu wymuszonego** (ang. *force-writing*) bufora dziennika przed zatwierdzeniem transakcji.

---

<sup>7</sup> Operacje cofania i ponawiania omówiono bardziej szczegółowo w rozdziale 19.

## 20.2.4. Strategie zastępowania bufora specyficzne dla SZBD

Pamięć podręczna SZBD przechowuje strony dysku zawierające informacje, które aktualnie są przetwarzane w buforach pamięci głównej. Jeśli wszystkie bufony tej pamięci podręcznej są zajęte, a nowe strony dysku trzeba wczytać do pamięci głównej, potrzebna jest **strategia zastępowania buforów**, aby wybrać konkretne bufony do zastąpienia. Dalej opiszemy niektóre strategie tego typu zaprojektowane specjalnie na potrzeby systemów baz danych.

**Metody podziału na dziedziny.** W SZBD występują różne rodzaje stron dyskowych: strony indeksu, strony pliku danych, strony pliku dziennika itd. W tej metodzie pamięć podręczna SZBD jest dzielona na odrębne dziedziny (zbiory buforów). Każda dziedzina dotyczy stron dysku jednego rodzaju, a zastępowanie stron w ramach każdej dziedziny odbywa się za pomocą podstawowego algorytmu LRU. Choć to rozwiązanie zwykle zapewnia wyższą wydajność niż sama technika LRU, jest to *algorytm statyczny*, który nie dostosowuje się do dynamicznie zmieniającego się obciążenia, ponieważ liczba dostępnych buforów w każdej dziedzinie jest stała. Zaproponowano kilka wersji strategii zastępowania stron metodą podziału na dziedziny, gdzie dodano dynamiczne funkcje równoważenia obciążenia. Przykładowo, algorytm **GRU** (ang. *Group LRU*) przypisuje każdej dziedzinie priorytet i w pierwszej kolejności wybiera do zastąpienia strony z dziedziny o najniższym priorytecie. Inna metoda dynamicznie zmienia liczbę buforów w każdej dziedzinie na podstawie aktualnego obciążenia.

**Metoda aktywnego zbioru (ang. *hot set*).** Ten algorytm zastępowania stron jest przydatny w zapytaniach, które muszą wielokrotnie przeglądać zbiory stron — np. wtedy, gdy wykonywane jest złączenie metodą pętli zagnieżdżonych (patrz rozdział 18.). Jeśli plik z pętli wewnętrznej zostanie w całości wczytany do buforów pamięci głównej i nie będzie zastępowany (stanie się zbiorem aktywnym), złączenie będzie wydajne, ponieważ dla każdej strony z pliku z pętli zewnętrznej trzeba przejrzeć wszystkie rekordy z pliku w pętli wewnętrznej, aby dopasować rekordy. Metoda aktywnego zbioru określa dla każdego algorytmu przetwarzania zestaw stron dysku, które są często używane, i nie zastępuje go do momentu zakończenia przetwarzania.

**Metoda DBMIN.** W tej strategii zastępowania stron stosowany jest model **QLSM** (ang. *query locality set model*), który określa wzorzec używanych stron w każdym algorytmie dla poszczególnych rodzajów operacji na bazie danych. Różne algorytmy dla relacyjnych operacji takich jak **SELECT** i **JOIN** omówiono w rozdziale 18. Na podstawie sposobu dostępu do danych, cech pliku i używanego algorytmu model QLSM pozwala oszacować liczbę buforów pamięci głównej potrzebnych dla każdego pliku z operacji. Strategia DBMIN oblicza za pomocą modelu QLSM **zbiór lokalny** dla każdej instancji pliku używanego w zapytaniu (niektóre zapytania mogą korzystać z tego samego pliku dwukrotnie, otrzymywany jest więc zbiór lokalny dla każdej potrzebnej instancji pliku). Następnie strategia przydziela każdej instancji pliku z zapytania odpowiednią liczbę buforów. Robi to na podstawie zbioru lokalnego danej instancji. Zbiór lokalny jest podobny do *zbioru roboczego*, używanego w strategiach zastępowania stron przez procesy systemu operacyjnego. Jednak zbiorów lokalnych może być wiele — po jednym dla każdej instancji pliku występującej w zapytaniu.



## 20.3. Pożądane właściwości transakcji

Transakcje powinny charakteryzować się kilkoma właściwościami. Często określa się je mianem **właściwości ACID** (od pierwszych liter odpowiednich terminów w języku angielskim) i powinny one być wymuszane przez mechanizm sterowania współbieżnego oraz metody odtwarzania SZBD. Poniżej wymieniono te właściwości.

- **Niepodzielność** (ang. *atomicity*). Transakcja jest niepodzielną jednostką przetwarzania — jest wykonywana albo w całości, albo wcale.
- **Zachowanie spójności** (ang. *consistency preservation*). Transakcja powinna zachowywać spójność. Oznacza to, że jeśli zostaje wykonana od początku do końca bez zakłóceń ze strony innych transakcji, przenosi bazę danych z jednego stanu do innego.
- **Izolacja** (ang. *isolation*). Transakcja powinna wyglądać tak, jakby była wykonywana w izolacji od innych transakcji (nawet jeśli jednocześnie wykonywanych jest wiele transakcji). Oznacza to, że wykonywanie transakcji nie powinno kolidować ze współbieżnym wykonywaniem innych transakcji.
- **Trwałość** (ang. *durability*). Zmiany zastosowane względem bazy danych przez zatwierdzone transakcje muszą być trwałe. Zmiany te nie mogą zostać utracone w wyniku jakiegokolwiek awarii.

Właściwość *niepodzielności* wymaga, aby transakcja była wykonywana do końca. Jej zapewnienie leży w gestii *podsystemu odtwarzania transakcji* SZBD. Jeżeli z pewnego powodu, takiego jak awaria systemu, transakcja nie zakończy swojego działania, mechanizm odtwarzania musi zapewnić wycofanie wszelkich efektów jej działania w bazie danych. Z kolei operacje zapisu z zatwierdzonej transakcji muszą zostać ostatecznie zapisane na dysku.

Odpowiedzialność za zachowanie *spójności* powszechnie zrzuca się na programistów, którzy piszą programy bazodanowe, lub na moduł SZBD, który wymusza więzy integralności. Warto przypomnieć, że **stan bazy danych** jest zbiorem wszystkich przechowywanych elementów danych (wartości) w bazie danych w danym punkcie czasu. **Stan spójny** (ang. *consistent state*) bazy danych spełnia więzy określone w jej schemacie, jak również wszelkie inne więzy, które powinny być zachowane. Program bazodanowy powinien być napisany w sposób gwarantujący, że jeżeli baza danych znajduje się w spójnym stanie przed wykonaniem transakcji, będzie również w spójnym stanie po *zakończeniu* wykonywania transakcji, zakładając, że *nie występują kolizje z innymi transakcjami*.

*Izolacja* jest wymuszana przez *podsystem sterowania współbieżnego* SZBD<sup>8</sup>. Jeżeli żadna transakcja nie udostępnia wprowadzanych zmian innym transakcjom do momentu swojego zatwierdzenia, wymuszana jest jedna z form izolacji, która rozwiązuje problem aktualizacji tymczasowej oraz eliminuje kaskadowe operacje wycofania (patrz rozdział 22.). Nie rozwiązuje to jednak niektórych innych problemów.

Za zachowanie właściwości *trwałości* odpowiedzialny jest *podsystem odtwarzania* SZBD. W następnym podrozdziale znajdziesz wprowadzenie do zapewniania trwałości i niepodzielności przez protokoły odtwarzania; bardziej szczegółowe omówienie tego zagadnienia zawiera rozdział 22.

---

<sup>8</sup> Protokoły sterowania współbieżnego zostaną omówione w rozdziale 21.



**Poziomy izolacji.** Podjęto próbę zdefiniowania **poziomu izolacji** (ang. *level of isolation*) transakcji. O transakcji mówi się, że posiada zerowy poziom izolacji, jeżeli nie nadpisuje zmodyfikowanych odczytów transakcji znajdujących się na wyższych poziomach. Izolacja poziomu pierwszego charakteryzuje się brakiem utraconych aktualizacji, zaś izolacja poziomu drugiego nie posiada ani utraconych aktualizacji, ani zmodyfikowanych odczytów. Wreszcie izolacja poziomu trzeciego (określana również mianem *prawdziwej izolacji*) oprócz właściwości poziomu drugiego posiada również odczyty powtarzalne<sup>9</sup>. Inny rodzaj izolacji to **izolacja snapshotów**. Jest ona podstawą kilku stosowanych w praktyce metod sterowania współbieżnego. Izolację snapshotów opiszemy w podrozdziale 20.6, a także w rozdziale 21. (podrozdział 21.4).

## 20.4. Charakteryzowanie harmonogramów na podstawie możliwości odtwarzania

Kiedy transakcje są wykonywane współbieżnie w technice przeplotu, kolejność wykonania operacji związanych z różnymi transakcjami określa się mianem **harmonogramu** (ang. *schedule*) lub **historii**. W niniejszym podrozdziale najpierw zostanie zdefiniowane pojęcie harmonogramu, a później zostaną omówione rodzaje harmonogramów umożliwiające odtwarzanie w razie wystąpienia awarii. W podrozdziale 20.5 harmonogramy zostaną opisane w kontekście nakładania się na siebie różnych transakcji, co prowadzi do zdefiniowania pojęcia szeregowalności i harmonogramów szeregowalnych.

### 20.4.1. Harmonogramy (historie) transakcji

**Harmonogram** (ang. *schedule*), inaczej **historia**,  $S$  zbioru  $n$  transakcji  $T_1, T_2, \dots, T_n$  jest uporządkowaniem operacji transakcji. Operacje z różnych transakcji mogą się przeplatać w harmonogramie  $S$ . Jednak dla każdej transakcji  $T_i$  należącej do harmonogramu  $S$  operacje tej transakcji w  $S$  muszą występować w tej samej kolejności, w jakiej występują w  $T_i$ . Uporządkowanie operacji w harmonogramie  $S$  ma *uporządkowanie całkowite* (ang. *total ordering*), co oznacza, że dla każdej pary operacji z harmonogramu jedna musi występować przed drugą. Z teoretycznego punktu widzenia możliwe jest występowanie harmonogramów, których operacje tworzą *uporządkowanie częściowe* (ang. *partial ordering*), na razie jednak zakładamy, że operacje mają uporządkowanie całkowite.

Dla celów odtwarzania i sterowania współbieżnego jesteśmy zainteresowani głównie operacjami odczytaj\_element oraz zapisz\_element, jak również operacjami zatwierdź oraz anuluj. Skrócona notacja opisu harmonogramów wykorzystuje symbole  $r$ ,  $w$ ,  $c$  oraz  $a$  dla operacji odczytaj\_element, zapisz\_element, zatwierdź oraz anuluj, oraz dodaje, jako indeks dolny, identyfikator transakcji (jej numer) do każdej operacji w harmonogramie. W przypadku zastosowania takiej notacji element bazy danych  $X$ , który jest odczytywany lub zapisywany, występuje w nawiasie po symbolu  $r$  lub  $w$ . W niektórych harmonogramach

---

<sup>9</sup> Składnia poziomów izolacji używana w języku SQL (patrz podrozdział 20.6) ściśle odpowiada tym poziomom.

przedstawiamy tylko operacje *odczytu* i *zapisu*, natomiast w innych uwzględniamy też dodatkowe operacje, np. zatwierdzania lub anulowania. Harmonogram z rysunku 20.3(a), który będziemy oznaczać symbolem  $S_a$ , można zapisać w takiej notacji następująco:

$$S_a: r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); w_1(Y);$$

Podobnie, harmonogram z rysunku 20.3(b), który będziemy oznaczać symbolem  $S_b$ , może zostać zapisany w poniższy sposób, jeżeli założymy, że transakcja  $T_1$  została anulowana po wykonaniu operacji `odczytaj_element(Y)`:

$$S_b: r_1(X); w_1(X); r_2(X); w_2(X); r_1(Y); a_1;$$

**Operacje powodujące konflikt w harmonogramie.** Dwie operacje w harmonogramie są w stanie **konfliktu**, jeżeli spełniają wszystkie trzy z następujących warunków: (1) należą do *różnych transakcji*; (2) uzyskują dostęp do *tego samego elementu X*; (3) *przynajmniej jedna* z operacji jest operacją `zapisz_element(X)`. Przykładowo, w przypadku harmonogramu  $S_a$  operacje  $r_1(X)$  oraz  $w_2(X)$  są w konflikcie, podobnie jak operacje  $r_2(X)$  i  $w_1(X)$  oraz operacje  $w_1(X)$  i  $w_2(X)$ . Jednakże operacje  $r_1(X)$  i  $r_2(X)$  nie są w konflikcie, gdyż obie są operacjami odczytu. Operacje  $w_2(X)$  i  $w_1(Y)$  również nie są w konflikcie, gdyż operują na odrębnych elementach danych  $X$  i  $Y$ . Z kolei operacje  $r_1(X)$  i  $w_1(X)$  nie są w konflikcie, ponieważ należą do tej samej transakcji.

Oto intuicyjne ujęcie: dwie operacje powodują konflikt, jeśli zmiana ich kolejności może skutkować zmianą wyniku. Przykładowo, jeśli zmienimy kolejność dwóch operacji ( $r_1(X); w_2(X)$ ) na ( $w_2(X); r_1(X)$ ), to wartość  $X$  wczytywana przez transakcję  $T_1$  zmieni się, ponieważ w drugim uporządkowaniu wartość  $X$  jest odczytywana przez  $r_1(X)$  *po* modyfikacjach w operacji  $w_2(X)$ ; w pierwszym uporządkowaniu wartość zostaje wczytana *przed* wprowadzeniem zmian. Jest to **konflikt odczytu i zapisu**. Innym rodzajem problemu jest **konflikt zapisu i zapisu**, występujący w sytuacji, gdy zmieniana jest kolejność dwóch operacji ( $w_1(X); w_2(X)$ ) na ( $w_2(X); w_1(X)$ ). W takim konflikcie *ostatnia wartość X* będzie inna, ponieważ w jednej sytuacji zostanie zapisana przez transakcję  $T_2$ , a w drugiej — przez  $T_1$ . Warto zauważyć, że dwie operacje odczytu nie powodują konfliktu, ponieważ zmiana ich kolejności nie skutkuje zmianą wyniku.

W dalszej części podrozdziału omówimy teoretyczne definicje dotyczące harmonogramów. O harmonogramie  $S$  składającym się z  $n$  transakcji  $T_1, T_2, \dots, T_n$ , mówimy, że jest **harmonogramem pełnym** (ang. *total schedule*), jeżeli spełnione są następujące warunki:

- (1) Operacje harmonogramu  $S$  są dokładnie tymi samymi operacjami, które występują w transakcjach  $T_1, T_2, \dots, T_n$ , wliczając w to operację zatwierdzenia lub anulowania jako ostatnią dla każdej transakcji.
- (2) Dla dowolnej pary operacji należących do tej samej transakcji  $T_i$  kolejność ich występowania w harmonogramie  $S$  jest taka sama jak kolejność w transakcji  $T_i$ .
- (3) Dla dowolnych dwóch transakcji konfliktowych jedna z nich musi występować w harmonogramie przed drugą<sup>10</sup>.

<sup>10</sup> Z teoretycznego punktu widzenia nie jest konieczne określanie kolejności występowania par operacji *bezkonfliktowych*.

Warunek 3. pozwala, aby dwie *operacje bezkonfliktowe* występowały w harmonogramie w dowolnej kolejności, co prowadzi do definicji harmonogramu jako **uporządkowania częściowego** operacji pochodzących z  $n$  transakcji<sup>11</sup>. Jednakże dla dowolnej pary operacji konfliktowych (warunek 3.) oraz dla dowolnej pary operacji należących do tej samej transakcji (warunek 2.) musi zostać określone uporządkowanie pełne. Warunek 1. określa po prostu, że wszystkie operacje należące do transakcji muszą występować w harmonogramie pełnym. Ze względu na fakt, że każda transakcja jest albo zatwierdzana, albo anulowana, harmonogram pełny na końcu *nie zawiera żadnych transakcji aktywnych*.

Ogólnie rzecz biorąc, trudno spotkać harmonogramy pełne w systemach przetwarzania transakcji, ponieważ nowe transakcje są ciągle generowane w systemie. Stąd przydatną rzeczą jest zdefiniowanie pojęcia **zatwierdzonego rzutowania** (ang. *committed projection*)  $C(S)$  harmonogramu  $S$ , które zawiera tylko te operacje z  $S$ , które należą do transakcji zatwierdzonych, to znaczy transakcji  $T_i$ , których operacje zatwierdzenia  $c_i$  występują w harmonogramie  $S$ .

## 20.4.2. Charakterystyka harmonogramów według możliwości odtwarzania

W przypadku pewnych harmonogramów odtwarzanie po awarii transakcji i systemu jest łatwe, natomiast w przypadku innych proces ten może być dość złożony. W niektórych sytuacjach poprawne odtworzenie stanu po awarii w ogóle nie jest możliwe. Dlatego istotną rzeczą jest scharakteryzowanie rodzajów harmonogramów, w przypadku których *odtworzenie jest możliwe*, jak również tych, w przypadku których jest to *względnie prosty proces*. Takie charakterystyki nie określają faktycznego algorytmu odtwarzania, a jedynie stanowią próbę teoretycznego opisu różnych rodzajów harmonogramów.

Po pierwsze, chcemy zapewnić, aby w momencie zatwierdzenia transakcji  $T$  już *nigdy* nie było konieczne jej wycofanie. To gwarantuje, że właściwość trwałości transakcji nie zostanie naruszona (patrz podrozdział 20.3). Harmonogramy, które teoretycznie spełniają ten wymóg, określa się mianem *harmonogramów odtwarzalnych* (ang. *recoverable schedules*), zaś te, które go nie spełniają — **nieodtworzalnych**, i dlatego nie powinno się pozwalać na ich występowanie w SZBD. Oto warunek **odtworzalności harmonogramu**: harmonogram  $S$  jest odtwarzalny, jeżeli żadna transakcja  $T$  w harmonogramie  $S$  nie jest zatwierdzana do momentu, aż wszystkie transakcje  $T'$ , które zapisały element odczytywany przez transakcję  $T$ , zostaną zatwierdzone. Transakcja  $T$  **czyta** z transakcji  $T'$  w harmonogramie  $S$ , jeżeli pewien element  $X$  jest najpierw zapisywany przez  $T'$ , a później odczytywany przez  $T$ . Ponadto, transakcja  $T'$  nie powinna zostać anulowana zanim transakcja  $T$  odczyta element  $X$  i nie powinny występować żadne transakcje zapisujące  $X$  po dokonaniu zapisu przez  $T'$ , a przed wykonaniem odczytu przez  $T$  (chyba że takie transakcje zostaną anulowane zanim transakcja  $T$  odczyta element  $X$ ).

---

<sup>11</sup> W praktyce większość harmonogramów posiada pełne uporządkowanie operacji. W razie wykorzystania przetwarzania równoległego istnieje teoretyczna możliwość posiadania harmonogramów z częściowo uporządkowanymi operacjami bezkonfliktowymi.

Jak się okaże, harmonogramy odtwarzalne czasem wymagają złożonego procesu odtwarzania, jednak jeśli przechowywanych jest wystarczająco dużo informacji (w dzienniku), dla każdego takiego harmonogramu można opracować algorytm odtwarzania. Harmonogramy (częściowe)  $S_a$  i  $S_b$  opisane w poprzednim podrozdziale są odtwarzalne, gdyż spełniają przedstawioną powyżej definicję. Weźmy pod uwagę poniższy harmonogram  $S_a'$ , który jest taki sam jak harmonogram  $S_a$ , z tą różnicą, że dodano do niego dwie operacje zatwierdzenia:

$$S_a': r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); c_2; w_1(Y); c_1;$$

Harmonogram  $S_a'$  jest odtwarzalny, pomimo że występuje w nim problem utraty aktualizacji; można sobie z nim poradzić za pomocą teorii szeregowańności (patrz podrozdział 20.5). Jednakże weźmy pod uwagę dwa (częściowe) harmonogramy  $S_c$  oraz  $S_d$ :

$$S_c: r_1(X); w_1(X); r_2(X); r_1(Y); w_2(X); c_2; a_1;$$

$$S_d: r_1(X); w_1(X); r_2(X); r_1(Y); w_2(X); w_1(Y); c_1; c_2;$$

$$S_e: r_1(X); w_1(X); r_2(X); r_1(Y); w_2(X); w_1(Y); a_1; a_2;$$

Harmonogram  $S_c$  nie jest odtwarzalny, ponieważ transakcja  $T_2$  odczytuje element  $X$  z transakcji  $T_1$ , a następnie transakcja  $T_2$  zostaje zatwierdzona przed transakcją  $T_1$ . Jeżeli transakcja  $T_1$  zostanie anulowana po wykonaniu operacji  $c_2$  w harmonogramie  $S_c$ , wówczas wartość elementu  $X$  odczytywana przez transakcję  $T_2$  nie jest już prawidłowa i transakcja  $T_2$  musi zostać anulowana już *po* jej zatwierdzeniu, co sprawia, że harmonogram *nie jest odtwarzalny*. Aby harmonogram mógł być odtwarzalny, operacja  $c_2$  w harmonogramie  $S_c$  musi zostać zawieszona do momentu, aż zostanie zatwierdzona transakcja  $T_1$ , co przedstawiono w formie harmonogramu  $S_d$ . Jeżeli zamiast zatwierdzenia transakcja  $T_1$  zostanie anulowana, wówczas powinna również zostać anulowana transakcja  $T_2$ , co przedstawiono w postaci harmonogramu  $S_e$ , ponieważ odczytywana przez nią wartość  $X$  nie jest już wówczas prawidłowa. W harmonogramie  $S_e$  anulowanie  $T_2$  jest akceptowalne, ponieważ transakcja ta nie została jeszcze zatwierdzona (w nieodtwarzalnym harmonogramie  $S_c$  jest inaczej).

W przypadku harmonogramu odtwarzalnego żadna zatwierdzona transakcja nigdy nie musi być wycofywana, dzięki czemu wymóg mówiący, że zatwierdzona transakcja jest trwała, nie jest naruszany. Jednak istnieje możliwość wystąpienia zjawiska **wycofania kaskadowego** (ang. *cascading rollback*), inaczej **anulowania kaskadowego** (ang. *cascading abort*), kiedy to *niezatwierdzona* transakcja musi zostać wycofana, ponieważ odczytała element z transakcji, która zakończyła się niepowodzeniem. Przedstawiono to w ramach harmonogramu  $S_e$ , gdzie transakcja  $T_2$  musi zostać wycofana, ponieważ odczytała element  $X$  z transakcji  $T_1$ , która później została anulowana.

Ze względu na fakt, że wycofywanie kaskadowe może być czasochłonne, gdyż może dotyczyć wielu transakcji (patrz rozdział 22.), istotną rzeczą jest scharakteryzowanie harmonogramów, w których zjawisko to nie może wystąpić. O harmonogramie mówi się, że jest **bezkaskadowy** (ang. *cascadeless*), czyli że **unikaj wycofywania kaskadowego**, jeżeli każda transakcja w harmonogramie odczytuje tylko te elementy, które zostały zapisane przez zatwierdzone transakcje. W takim przypadku wszystkie odczytywane elementy nie mogą zostać usunięte, więc nie występuje wycofywanie kaskadowe. W celu spełnienia tego kryterium polecenie  $r_2(X)$  w harmonogramach  $S_d$  i  $S_e$  musi zostać wstrzymane do momentu, aż zostanie zatwierdzona (lub anulowana) transakcja  $T_1$ , przez co transakcja  $T_2$  zostanie

opóźniona, co zapewni niewystępowanie wycofywania kaskadowego w razie anulowania transakcji  $T_1$ .

Wreszcie można wyróżnić trzeci, bardziej restrykcyjny rodzaj harmonogramu, noszący nazwę **harmonogramu ścisłego** (ang. *strict schedule*), gdzie transakcje nie mogą *ani odczytywać, ani zapisywać* elementu  $X$  do momentu, aż zostanie zatwierdzona (lub anulowana) ostatnia transakcja, która go zapisała. Harmonogramy ścisłe upraszczają proces odtwarzania. W ich przypadku proces cofnięcia operacji `zapisz_element( $X$ )` anulowanej transakcji polega na zwykłym odtworzeniu **obrazu pierwotnego** (ang. *before image*, BFIM), czyli wartości *stara\_wartość*, elementu danych  $X$ . Ta prosta procedura zawsze działa poprawnie w przypadku harmonogramów ścisłych, jednak może się nie sprawdzać w przypadku harmonogramów odtwarzalnych lub bezkaskadowych. Weźmy na przykład pod uwagę harmonogram  $S_f$ :

$S_f: w_1(X, 5); w_2(X, 8); a_1;$

Założmy, że początkową wartością  $X$  było 9, co jest obrazem pierwotnym przechowywanym w dzienniku systemowym wraz z operacją  $w_1(X, 5)$ . Jeżeli transakcja  $T_1$  zostanie anulowana, jak ma to miejsce w harmonogramie  $S_f$ , procedura odtwarzania przywraca ją obraz pierwotny anulowanej operacji zapisu przywróci wartość  $X$  na 9, mimo że została ona już zmieniona na wartość 8 przez transakcję  $T_2$ , co potencjalnie prowadzi do niepoprawnych wyników. Chociaż harmonogram  $S_f$  jest bezkaskadowy, nie jest on harmonogramem ścisłym, gdyż pozwala, aby transakcja  $T_2$  zapisała element  $X$ , nawet jeśli transakcja  $T_1$ , która jako ostatnia zapisała element  $X$ , nie została jeszcze zatwierdzona (lub anulowana). Problem ten nie występuje w przypadku harmonogramów ścisłych.

Warto zauważyć, że każdy harmonogram ścisły jest bezkaskadowy, a wszystkie harmonogramy bezkaskadowe są odtwarzalne. Załóżmy, że występuje  $i$  transakcji  $T_1, T_2, \dots, T_i$  obejmujących  $n_1, n_2, \dots, n_i$  operacji. Po utworzeniu zbioru *wszystkich możliwych harmonogramów* tych transakcji można podzielić je na dwa rozłączne zbiory: odtwarzalne i nieodtwarzalne. Harmonogramy bezkaskadowe to podzbiór harmonogramów odtwarzalnych, a harmonogramy ścisłe tworzą podzbiór harmonogramów bezkaskadowych. Tak więc wszystkie harmonogramy ścisłe są bezkaskadowe, a wszystkie harmonogramy bezkaskadowe są odtwarzalne.

Większość protokołów odtwarzania dopuszcza tylko harmonogramy ścisłe, dlatego sam proces odtwarzania nie jest skomplikowany (patrz rozdział 22.).

## 20.5. Charakterystyka harmonogramów według ich szeregowności

W poprzednim podrozdziale scharakteryzowaliśmy harmonogramy pod względem ich właściwości odtwarzalności. Poniżej zostaną scharakteryzowane rodzaje harmonogramów uważanych za *poprawne* w przypadku wykonywania transakcji współbieżnych. Takie harmonogramy nazywamy *szeregownymi*. Założmy, że dwaj użytkownicy — obsługujący stanowiska rezerwacji biletów lotniczych — przesyłają do SZBD transakcje  $T_1$  i  $T_2$  z rysunku 20.2 mniej więcej w tym samym momencie. Jeżeli nie jest dozwolony przeplot operacji, istnieją tylko dwie możliwości:

- (1) Wykonanie wszystkich operacji transakcji  $T_1$  (po kolei), a następnie wszystkich operacji transakcji  $T_2$  (po kolei).
- (2) Wykonanie wszystkich operacji transakcji  $T_2$  (po kolei), a następnie wszystkich operacji transakcji  $T_1$  (po kolei).

Oba te rozwiązania (*harmonogramy szeregowe*) przedstawiono na rysunkach, odpowiednio, 20.5(a) i 20.5(b). Jeżeli dopuszczalny jest przeplot operacji, istnieje wiele możliwych kolejności, w jakich system może wykonać poszczególne operacje tych transakcji. Dwa możliwe harmonogramy przedstawiono na rysunku 20.5(c). Pojęcie **szeregowalności harmonogramów** (ang. *serializability of schedules*) jest używane w celu identyfikowania, które harmonogramy są poprawne w przypadku, gdy wykonanie transakcji dopuszcza przeplot ich operacji. W niniejszym podrozdziale zostanie zdefiniowane pojęcie szeregowalności oraz omówione sposoby jej stosowania w praktyce.

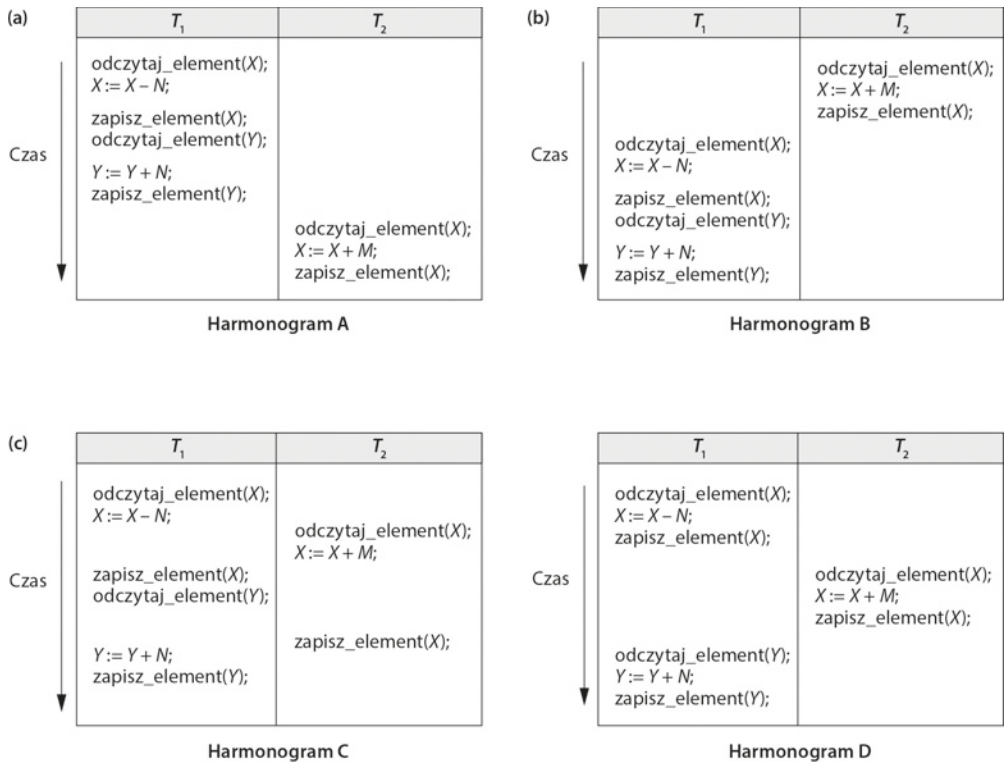
### 20.5.1. Harmonogramy szeregowe, nieszeregowe oraz konfliktowo-szeregowalne

Harmonogramy  $A$  i  $B$  z rysunków 20.5(a) i (b) określa się mianem *szeregowych* (ang. *serial*), ponieważ operacje każdej transakcji są wykonywane po kolei, bez przeplotu z operacjami innych transakcji. W przypadku harmonogramu szeregowego całe transakcje są wykonywane po kolei: najpierw  $T_1$ , potem  $T_2$  na rysunku 20.5(a) oraz najpierw  $T_2$ , potem  $T_1$  na rysunku 20.5(b). Harmonogramy  $C$  i  $D$  z rysunku 20.5(c) określa się mianem *niezeregowych* (ang. *nonserial*), ponieważ każda sekwencja stanowi przeplot operacji należących do różnych transakcji.

Ujmując rzecz formalnie, harmonogram  $S$  jest **szeregowy**, jeżeli dla każdej transakcji  $T$  należącej do harmonogramu wszystkie operacje transakcji  $T$  są wykonywane po kolei w ramach harmonogramu. W przeciwnym razie harmonogram jest **niezeregowy**. Stąd w przypadku harmonogramu szeregowego tylko jedna transakcja w danym momencie jest aktywna — zatwierdzenie (lub anulowanie) aktywnej transakcji inicjuje wykonanie kolejnej. W takim przypadku nie występuje przeplot. Jeżeli transakcje traktujemy jako *niezależne*, to możemy bezpiecznie założyć, że *każdy harmonogram szeregowy jest poprawny*. Możemy tak postąpić, ponieważ co do każdej transakcji zakłada się, że jest poprawna, jeżeli jest wykonywana samodzielnie (zgodnie z właściwością *zachowania spójności* z podrozdziału 20.3). Dlatego też nie ma znaczenia, która transakcja jest wykonywana jako pierwsza. O ile każda z nich zostanie wykonana w całości bez zakłóceń ze strony operacji innych transakcji, w bazie danych otrzymujemy poprawny wynik końcowy.

Problem związany z harmonogramami szeregowymi polega na tym, że ograniczają one współbieżność i przeplot wykonywania operacji. W przypadku harmonogramu szeregowego, jeżeli transakcja oczekuje na zakończenie operacji wejścia-wyjścia, nie można przełączyć procesora na wykonywanie innej transakcji, co powoduje marnowanie wartościowego czasu procesora. Ponadto, jeżeli pewna transakcja  $T$  trwa dość długo, inne transakcje muszą oczekiwać na zakończenie przez transakcję  $T$  wszystkich operacji, zanim będą mogły rozpocząć swoje działanie. Stąd harmonogramy szeregowe w praktyce traktuje się jako *niemożliwe do zaakceptowania*. Jeśli jednak zdołamy ustalić, jakie inne harmonogramy są *równoważne* szeregowemu, można pozwolić na ich wykonanie.





RYSUNEK 20.5. Przykłady harmonogramów szeregowych i nieszeregowych dotyczących transakcji  $T_1$  i  $T_2$ . (a) Harmonogram szeregowy: najpierw transakcja  $T_1$ , potem  $T_2$ . (b) Harmonogram szeregowy: najpierw transakcja  $T_2$ , potem  $T_1$ . (c) Dwa harmonogramy nieszegowe C i D z operacjami podlegającymi przepłutowi

W celu zilustrowania prowadzonego wykładu weźmy pod uwagę rysunek 20.5 i załóżmy, że początkowe wartości elementów danych bazy to  $X = 90$  i  $Y = 90$  oraz że  $N = 3$ , zaś  $M = 2$ . Po wykonaniu transakcji  $T_1$  i  $T_2$  możemy oczekiwać, że wartości te będą wynosić  $X = 89$  oraz  $Y = 93$ , zgodnie ze znaczeniem transakcji. Oczywiście wykonanie któregoś z harmonogramów szeregowych A lub B daje poprawny wynik. Rozważmy teraz harmonogramy nieszegowe C i D. Harmonogram C (taki sam jak na rysunku 20.3(a)) daje w wyniku  $X = 92$  oraz  $Y = 93$ , co oznacza, że wartość  $X$  jest niepoprawna. Z kolei harmonogram D daje poprawne wyniki.

Harmonogram C daje błędny wynik dlatego, że w jego przypadku występuje *problem utraconej aktualizacji*, omówiony w podrozdziale 20.1.3. Transakcja  $T_2$  odczytuje wartość  $X$  zanim zostanie ona zmieniona przez transakcję  $T_1$ , więc w bazie danych zostaje odzwierciedlony tylko efekt transakcji  $T_2$ . Efekt działania transakcji  $T_1$  zostaje *utracony* — jest nadpisywany przez transakcję  $T_2$ , co prowadzi do powstania nieprawidłowego wyniku elementu  $X$ . Jednak niektóre harmonogramy nieszegowe dają poprawny, spodziewany wynik, na przykład harmonogram D. Chcemy określić, które harmonogramy nieszegowe *zawsze* dają poprawny wynik, a które mogą dawać wyniki błędne. Pojęciem używanym w celu charakteryzowania harmonogramów pod tym względem jest ich szeregowalność.



Oto definicja *harmonogramu szeregowalnego*: harmonogram  $S$  składający się z  $n$  transakcji jest **szeregowalny** (ang. *serializable*), jeżeli jest *równoważny pewnemu harmonogramowi szeregowemu* tych samych  $n$  transakcji. Wkrótce zostanie zdefiniowane pojęcie *równoważności harmonogramów*. Należy zauważyć, że istnieje  $n!$  możliwych harmonogramów szeregowych dla  $n$  transakcji i wiele więcej harmonogramów nieszeregowych. Można określić dwie rozłączne grupy harmonogramów nieszeregowych: takich, które są równoważne jednemu (lub większej liczbie) harmonogramowi szeregowemu, a stąd są szeregowalne, oraz takich, które nie są równoważne *jakiemukolwiek* harmonogramowi szeregowemu, a stąd nie są szeregowalne.

Stwierdzenie, że nieszeregowy harmonogram  $S$  jest szeregowalny, oznacza tyle, co stwierdzenie, że jest on poprawny, ponieważ jest równoważny harmonogramowi szeregowemu, uważanemu za poprawny. Otwarta pozostaje kwestia następująca: kiedy dwa harmonogramy uważa się za *równoważne*?

Istnieje kilka sposobów definiowania równoważności harmonogramów. Najprostsza, ale jednocześnie najmniej zadowalająca, definicja równoważności harmonogramów polega na porównaniu efektów harmonogramów w bazie danych. O dwóch harmonogramach mówi się, że są **równoważne co do wyniku** (ang. *result equivalent*), jeżeli dają ten sam końcowy stan bazy danych. Jednakże dwa różne harmonogramy mogą przypadkowo dawać ten sam stan końcowy. Przykładowo, na rysunku 20.6 harmonogramy  $S_1$  i  $S_2$  dają w wyniku ten sam końcowy stan bazy danych, jeżeli wykona się je dla początkowej wartości  $X = 100$ , ale dla innych wartości początkowych  $X$  harmonogramy te *nie* będą równoważne co do wyniku. Poza tym takie dwa harmonogramy wykonują różne transakcje, więc bez wątpienia nie należy ich traktować jako równoważne. Dlatego też sama równoważność wynikowa nie może być używana w celu definiowania równoważności harmonogramów. Najbezpieczniejsze i najbardziej ogólne podejście do kwestii zdefiniowania równoważności harmonogramów polega na skupieniu się na operacjach wczytywania i zapisu w transakcji oraz nieprzyjmowaniu żadnych założeń co do rodzajów operacji związanych z transakcjami. Aby dwa harmonogramy były równoważne, operacje stosowane względem każdego elementu danych, na który mają wpływ te harmonogramy, powinny być zastosowane do tych elementów w obu harmonogramach *w tej samej kolejności*. Ogólnie rzecz biorąc, wykorzystuje się dwie definicje równoważności harmonogramów: *równoważności konfliktowej* (ang. *conflict equivalence*) oraz *równoważności perspektywicznej* (ang. *view equivalence*). Poniżej zostanie omówiona równoważność konfliktowa, której używa się częściej.

$S_1$	$S_2$
odczytaj_element( $X$ ); $X := X + 10$ ; zapisz_element( $X$ );	odczytaj_element( $X$ ); $X := X * 1.1$ ; zapisz_element( $X$ );

RYSUNEK 20.6. Dwa harmonogramy, które są równoważne co do wyniku w przypadku wartości początkowej  $X = 100$ , ale ogólnie nie są równoważne co do wyniku

**Równoważność konfliktowa dwóch harmonogramów.** O dwóch harmonogramach mówimy, że są **równoważne konfliktowo** (ang. *conflict equivalent*), jeżeli kolejność dowolnych dwóch *operacji konfliktowych* jest taka sama w obu harmonogramach. W podrozdziale

20.4.1 stwierdzono, że o dwóch operacjach w harmonogramie mówi się, że są *konfliktowe*, jeżeli należą do różnych transakcji, uzyskują dostęp do tego samego elementu bazy danych i przynajmniej jedna z tych dwóch operacji jest operacją `zapisz_element` (drugą może być `odczytaj_element`). Jeżeli dwie operacje konfliktowe zostaną zastosowane w *różnej kolejności* w dwóch harmonogramach, uzyskany efekt może być różny w bazie danych lub w odniesieniu do innych transakcji w danym harmonogramie, a stąd harmonogramy te nie są równoważne konfliktowo. Przykładowo, jeżeli — jak opisano w punkcie 20.4.1 — operacje odczytu i zapisu występują w kolejności  $r_1(X)$ ,  $w_2(X)$  w harmonogramie  $S_1$  oraz w odwróconej kolejności  $w_2(X)$ ,  $r_1(X)$  w harmonogramie  $S_2$ , wartość odczytana przez operację  $r_1(X)$  może być w obu harmonogramach różna. Podobnie, jeżeli dwie operacje zapisu występują w kolejności  $w_1(X)$ ,  $w_2(X)$  w harmonogramie  $S_1$  oraz w odwróconej kolejności  $w_2(X)$ ,  $w_1(X)$  w harmonogramie  $S_2$ , to kolejna operacja  $r(X)$  w obu harmonogramach potencjalnie może odczytać odmienne wartości, a jeśli są to ostatnie operacje zapisu elementu  $X$  w ramach harmonogramu, końcowa wartość  $X$  w bazie danych będzie różna.

**Harmonogramy szeregowalne.** Wykorzystując pojęcie równoważności konfliktowej, harmonogram  $S$  definiujemy jako **szeregowalny konfliktowo**<sup>12</sup> (ang. *conflict serializable*), jeżeli jest on (konfliktowo) równoważny pewnemu szeregowemu harmonogramowi  $S'$ . W takim przypadku można reorganizować *niekonfliktowe* operacje w harmonogramie  $S$  do momentu, aż utworzy się równoważny harmonogram szeregowy  $S'$ . Zgodnie z taką definicją harmonogram  $D$  z rysunku 20.5(c) jest równoważny szeregowemu harmonogramowi  $A$  z rysunku 20.5(a). W obu tych harmonogramach operacja `odczytaj_element( $X$ )` transakcji  $T_2$  odczytuje wartość  $X$  zapisaną przez transakcję  $T_1$ , natomiast pozostałe operacje `odczytaj_element` odczytują wartości z bazy danych z jej stanu początkowego. Ponadto transakcja  $T_1$  jest ostatnią transakcją zapisującą element  $Y$ , zaś transakcja  $T_2$  jest ostatnią transakcją zapisującą element  $X$  w obu harmonogramach. Ze względu na fakt, że  $A$  jest harmonogramem szeregowym, zaś  $D$  jest harmonogramem równoważnym harmonogramowi  $A$ ,  $D$  jest *harmonogramem szeregowalnym*. Należy zauważyć, że operacje  $r_1(Y)$  oraz  $w_1(Y)$  harmonogramu  $D$  nie pozostają w konflikcie z operacjami  $r_2(X)$  oraz  $w_2(X)$ , gdyż uzyskują dostęp do różnych elementów danych. Możemy więc przenieść operacje  $r_1(Y)$ ,  $w_1(Y)$  przed operacje  $r_2(X)$ ,  $w_2(X)$ , co pozwala na uzyskanie równoważnego harmonogramu szeregowego  $T_1, T_2$ .

Harmonogram  $C$  z rysunku 20.5(c) nie jest równoważny żadnemu z dwóch możliwych harmonogramów szeregowych  $A$  i  $B$ , a stąd *nie jest szeregowalny*. Próba reorganizacji harmonogramu  $C$  w celu znalezienia równoważnego harmonogramu szeregowego kończy się niepowodzeniem, ponieważ operacje  $r_2(X)$  i  $w_1(X)$  pozostają w konflikcie, co oznacza, że nie możemy przenieść operacji  $r_2(X)$  w dół w celu otrzymania równoważnego harmonogramu szeregowego  $T_1, T_2$ . Podobnie, ze względu na fakt, że operacje  $w_1(X)$  oraz  $w_2(X)$  pozostają w konflikcie, nie można przenieść operacji  $w_1(X)$  w dół w celu otrzymania równoważnego harmonogramu szeregowego  $T_2, T_1$ .

Kolejną, bardziej skomplikowaną definicją równoważności — określaną mianem *równoważności perspektywicznej* — która prowadzi do wprowadzenia pojęcia *szeregowalności perspektywicznej* (ang. *view serializability*), omówiono w podrozdziale 20.5.4.

<sup>12</sup> Pojęcia *szeregowalny* będziemy używać w znaczeniu szeregowalności konfliktowej. Inna, praktycznie stosowana definicja szeregowalności (patrz podrozdział 20.6) określa, że oznacza to posiadanie odczytów powtarzalnych, brak odczytów zmodyfikowanych oraz brak rekordów fantomowych (patrz punkt 22.7.1, gdzie omówiono fantomy).

## 20.5.2. Sprawdzanie występowania szeregowalności konfliktowej harmonogramu

Istnieje prosty algorytm określania szeregowalności konfliktowej harmonogramu. Większość metod sterowania współbieżnego w rzeczywistości *nie* przeprowadza testów szeregowalności. Zamiast tego opracowuje się raczej protokoły lub reguły gwarantujące szeregowalność harmonogramów. Niektóre metody gwarantują szeregowalność w większości sytuacji, ale nie dają pełnej pewności. Pozwala to zmniejszyć koszty sterowania współbieżnego. Poniżej zostanie omówiony algorytm testowania szeregowalności konfliktowej, co pozwoli Czytelnikowi na lepsze zrozumienie takich protokołów sterowania współbieżnego, które zostaną omówione w rozdziale 21.

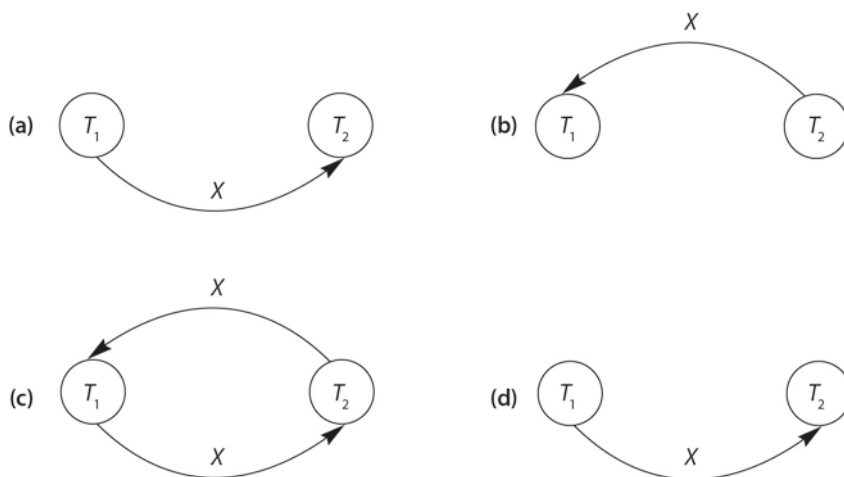
Algorytm 20.1 może być używany w celu sprawdzania harmonogramu pod względem występowania szeregowalności konfliktowej. Sprawdza on jedynie operacje `odczytaj_element` oraz `zapisz_element` występujące w harmonogramie w celu skonstruowania **grafu pierwszeństwa** (ang. *precedence graph*), inaczej **grafu szeregowania** (ang. *serialization graph*), który jest **grafem skierowanym**  $G = (N, E)$ , składającym się ze zbioru wierzchołków  $N = \{T_1, T_2, \dots, T_n\}$  oraz zbioru krawędzi skierowanych  $E = \{e_1, e_2, \dots, e_m\}$ . Dla każdej transakcji  $T_i$  w harmonogramie istnieje jeden wierzchołek w grafie. Każda krawędź  $e_i$  ma postać  $(T_j \rightarrow T_k)$ , gdzie  $1 \leq j \leq n$ ,  $1 \leq k \leq n$  oraz  $T_j$  jest **wierzchołkiem początkowym** krawędzi  $e_i$ , zaś  $T_k$  jest **wierzchołkiem końcowym** krawędzi  $e_i$ . Krawędź taka z  $T_j$  do  $T_k$  jest tworzona przez algorytm wówczas, gdy w  $T_j$  i  $T_k$  występuje para operacji konfliktowych i jedna z operacji transakcji  $T_j$  występuje w harmonogramie *przed* pewną operacją konfliktową transakcji  $T_k$ .

**Algorytm 20.1.** Testowanie szeregowalności konfliktowej harmonogramu  $S$

- (1) Dla każdej transakcji  $T_i$  należącej do harmonogramu  $S$  utwórz w grafie pierwszeństwa wierzchołek opatrzony etykietą  $T_i$ .
- (2) Dla każdego przypadku w harmonogramie  $S$ , gdy transakcja  $T_j$  wykonuje operację `odczytaj_element( $X$ )` po wykonaniu przez transakcję  $T_i$  operacji `zapisz_element( $X$ )`, utwórz w grafie pierwszeństwa krawędź  $(T_i \rightarrow T_j)$ .
- (3) Dla każdego przypadku w harmonogramie  $S$ , gdy transakcja  $T_j$  wykonuje operację `zapisz_element( $X$ )` po wykonaniu przez transakcję  $T_i$  operacji `odczytaj_element( $X$ )`, utwórz w grafie pierwszeństwa krawędź  $(T_i \rightarrow T_j)$ .
- (4) Dla każdego przypadku w harmonogramie  $S$ , gdy transakcja  $T_j$  wykonuje operację `zapisz_element( $X$ )` po wykonaniu przez transakcję  $T_i$  operacji `zapisz_element( $X$ )`, utwórz w grafie pierwszeństwa krawędź  $(T_i \rightarrow T_j)$ .
- (5) Harmonogram  $S$  jest szeregowalny wtedy i tylko wtedy, gdy graf pierwszeństwa nie zawiera cykli.

Graf pierwszeństwa jest konstruowany zgodnie z algorytmem 20.1. Jeżeli występuje w nim cykl, harmonogram  $S$  nie jest (konfliktowo) szeregowalny. Jeżeli cykl nie występuje, harmonogram  $S$  jest szeregowalny. **Cykl** (ang. *cycle*) w grafie skierowanym jest **ciągami krawędzi**  $C = ((T_j \rightarrow T_k), (T_k \rightarrow T_p), \dots, (T_i \rightarrow T_j))$  o tej własności, że wierzchołek początkowy każdej krawędzi (oprócz pierwszej) jest taki sam jak wierzchołek końcowy poprzedniej krawędzi oraz wierzchołek początkowy pierwszej krawędzi jest taki sam jak wierzchołek końcowy ostatniej krawędzi (ciąg krawędzi zaczyna się i kończy w tym samym wierzchołku).

W grafie pierwszeństwa krawędź wiodąca z wierzchołka  $T_i$  do  $T_j$  oznacza, że transakcja  $T_i$  musi występować przed transakcją  $T_j$  w dowolnym harmonogramie szeregowym równoważnym harmonogramowi  $S$ , ponieważ dwie operacje konfliktowe występują w harmonogramie w takiej kolejności. Jeżeli w grafie pierwszeństwa nie występuje żaden cykl, można utworzyć **równoważny harmonogram szeregowy** (ang. *equivalent serial schedule*)  $S'$ , który jest równoważny harmonogramowi  $S$ , porządkując transakcje należące do  $S$  w sposób następujący: kiedy w grafie pierwszeństwa występuje krawędź wiodąca z wierzchołka  $T_i$  do  $T_j$ , wierzchołek  $T_i$  musi występować przed wierzchołkiem  $T_j$  w równoważnym harmonogramie szeregowym  $S'$ <sup>13</sup>. Należy zauważyć, że krawędzie  $(T_i \rightarrow T_j)$  w grafie pierwszeństwa mogą opcjonalnie zostać opatrzone etykietami z nazwami elementów danych, które spowodowały utworzenie danej krawędzi. Na rysunku 20.7 przedstawiono takie etykiety. W trakcie wykrywania cykli etykiety nie mają znaczenia.

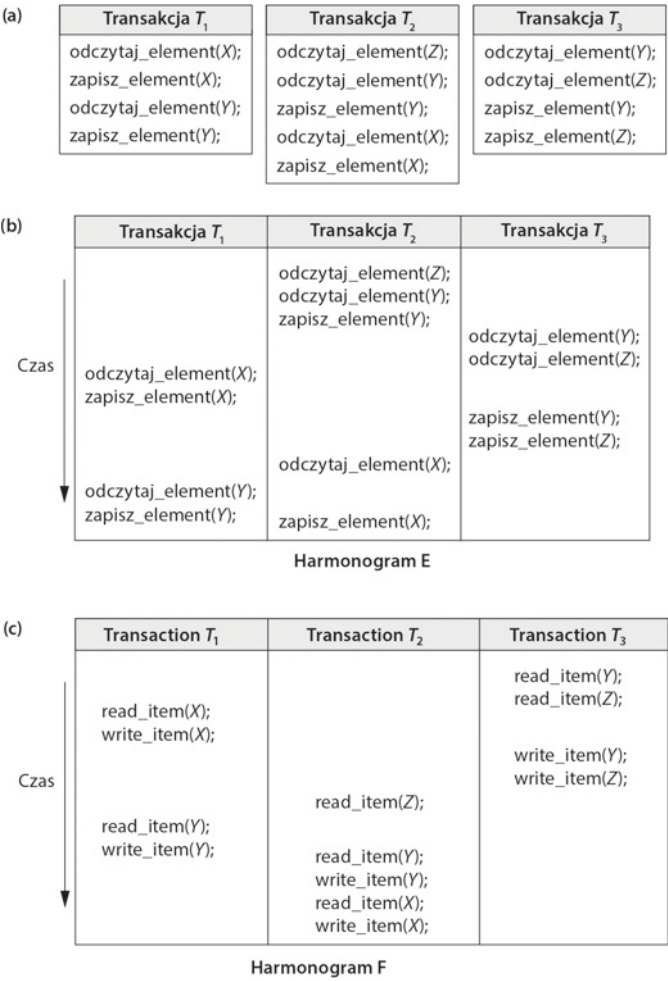


RYSUNEK 20.7. Konstrukcja grafów pierwszeństwa dla harmonogramów od A do D z rysunku 20.5 w celu sprawdzenia szeregowalności konfliktowej. (a) Graf pierwszeństwa dla harmonogramu szeregowego A. (b) Graf pierwszeństwa dla harmonogramu szeregowego B. (c) Graf pierwszeństwa dla harmonogramu C (nieszeregowalnego). (d) Graf pierwszeństwa dla harmonogramu D (szeregowalnego, równoważnego harmonogramowi A)

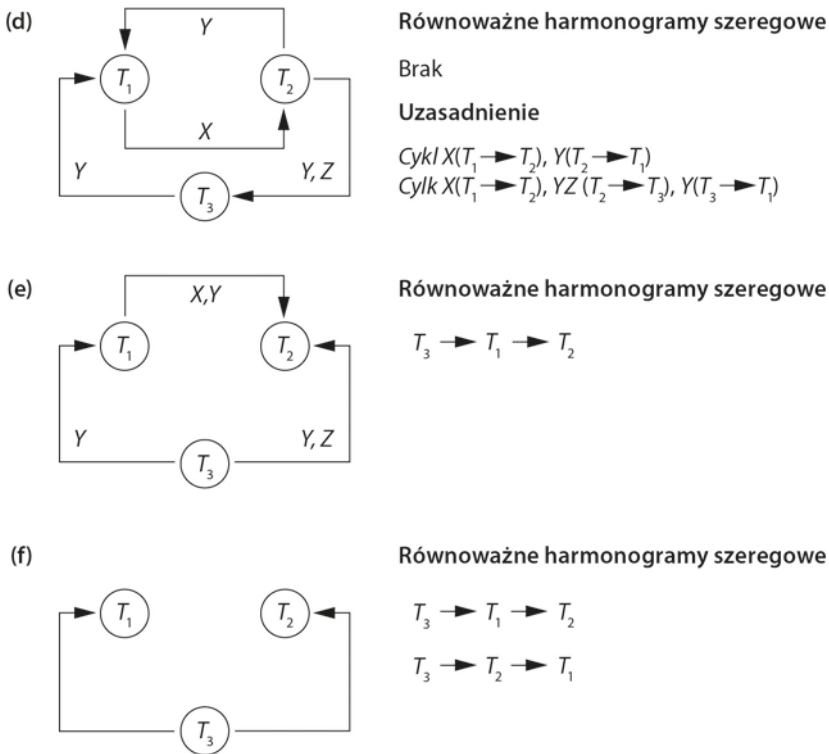
Ogólnie rzecz biorąc, kilka harmonogramów szeregowych może być równoważnych harmonogramowi  $S$ , jeżeli graf pierwszeństwa tego harmonogramu nie zawiera cykli. Jednak jeżeli graf pierwszeństwa zawiera cykl, z łatwością można wykazać, że nie da się utworzyć żadnego równoważnego harmonogramu szeregowego, więc harmonogram  $S$  nie jest szeregowalny. Grafy pierwszeństwa utworzone dla harmonogramów od A do D z rysunku 20.5 przedstawiono na rysunkach od 20.7(a) do 20.7(d). Graf dla harmonogramu C zawiera cykl, więc nie jest szeregowalny. Graf dla harmonogramu D nie zawiera cykli, więc jest szeregowalny i w równoważnym harmonogramie szeregowym po transakcji  $T_1$  występuje transakcja  $T_2$ . Grafy dla harmonogramów A i B, zgodnie z oczekiwaniami, nie zawierają cykli, ponieważ są one *szeregowy*, a stąd również szeregowalne.

<sup>13</sup> Taki proces porządkowania wierzchołków grafu acyklicznego określa się mianem sortowania topologicznego.

Kolejny przykład, uwzględniający trzy transakcje, przedstawiono na rysunku 20.8. Na rysunku 20.8(a) pokazano operacje `odczytaj_element` i `zapisz_element` każdej transakcji. Dwa harmonogramy *E* i *F* dla tych transakcji przedstawiono na rysunkach, odpowiednio, 20.8(b) i 20.8(c), zaś grafy pierwszeństwa dla harmonogramów *E* i *F* przedstawiono w częściach (d) i (e). Harmonogram *E* nie jest szeregowalny, ponieważ odpowiedni graf pierwszeństwa zawiera cykle. Harmonogram *F* jest szeregowalny i harmonogram mu równoważny przedstawiono na rysunku 20.8(e). Chociaż dla harmonogramu *F* istnieje tylko jeden równoważny harmonogram szeregowy, w ogólnym przypadku może występować *wiele równoważnych harmonogramów szeregowych* dla harmonogramu szeregowalnego. Na rysunku 20.8(f) przedstawiono graf pierwszeństwa reprezentujący harmonogram o dwóch równoważnych harmonogramach szeregowych. Aby znaleźć równoważny harmonogram szeregowy, zacznij od węzła, do którego nie prowadzą żadne krawędzie wejściowe, a następnie upewnij się, że kolejność węzłów dla poszczególnych krawędzi nie została zmieniona.



RYSUNEK 20.8. Inny przykład sprawdzania szeregowalności. (a) Operacje odczytu i zapisu trzech transakcji  $T_1$ ,  $T_2$  oraz  $T_3$ . (b) Harmonogram E. (c) Harmonogram F



RYSUNEK 20.8. (ciąg dalszy) Inny przykład sprawdzania szeregowalności. (d) Graf pierwszeństwa dla harmonogramu E. (e) Graf pierwszeństwa dla harmonogramu F. (f) Graf pierwszeństwa z dwoma równoważnymi harmonogramami szeregowymi

### 20.5.3. Wykorzystywanie szeregowalności do sterowania współbieżnego

Jak napisano powyżej, stwierdzenie, że harmonogram  $S$  jest (konfliktowo) szeregowalny — to znaczy, że harmonogram  $S$  jest (konfliktowo) równoważny harmonogramowi szeregowemu — oznacza tyle, co stwierdzenie, że jest on poprawny. Jednakże harmonogram *szeregowalny* nie jest tym samym, co harmonogram *szeregowy*. Harmonogram szeregowy reprezentuje niewydajny sposób przetwarzania, ponieważ nie jest wówczas dozwolony przeplot żadnych operacji należących do różnych transakcji. Może to prowadzić do niskiego poziomu wykorzystania mocy procesora, kiedy transakcje oczekują na zakończenie dyskowych operacji wejścia-wyjścia lub na zakończenie innych transakcji, co ma bardzo negatywny wpływ na szybkość przetwarzania. Harmonogram szeregowalny daje korzyści płynące z wykonania współbieżnego przy zachowaniu poprawności rozwiązania. W praktyce testowanie szeregowalności harmonogramów jest dość trudne. Przeplot operacji należących do współbieżnie wykonywanych transakcji (które na poziomie systemu operacyjnego zwykle są wykonywane jako procesy) jest zazwyczaj określany przez menedżera procesów systemu operacyjnego, który przydziela zasoby wszystkim procesom. Czynniki takie jak obciążenie systemu, czas dostarczenia transakcji oraz priorytety procesów przy-



czynią się do kolejności wykonywania operacji w harmonogramie. Stąd trudno jest z góry określić w celu zapewnienia szeregowalności, w jaki sposób będzie przebiegał przeplot operacji harmonogramu.

Jeżeli transakcje wykonuje się w dowolnych momentach, a później sprawdza wynikowy harmonogram pod względem jego szeregowalności, trzeba anulować wyniki jego działania, jeżeli okaże się, że nie jest szeregowalny. Jest to poważny problem, który sprawia, że takie podejście jest niepraktyczne. W przypadku większości komercyjnych SZBD projektuje się **protokoły** (zbiory reguł), które w przypadku ich przestrzegania przez *każdą* transakcję lub wymuszania przez podsystem sterowania współbieżnego SZBD zapewniają szeregowalność *wszystkich harmonogramów, do których należą transakcje*. Niektóre protokoły mogą w rzadkich sytuacjach dopuszczać harmonogramy nieszegrowalne, aby zmniejszyć koszty sterowania współbieżnego (patrz podrozdział 20.6).

Istnieje jeszcze jeden problem. Kiedy transakcje są generowane w systemie przez cały czas, trudno jest określić, kiedy harmonogram rozpoczyna, a kiedy kończy działanie. Teorię szeregowalności można dostosować do radzenia sobie z tym problemem, nakazując rozpatrywanie tylko zatwierdzonego rzutowania harmonogramu  $S$ . Zgodnie z tym, co stwierdzono w punkcie 20.4.1, *zatwierdzone rzutowanie*  $C(S)$  harmonogramu  $S$  uwzględnia tylko te jego operacje, które należą do zatwierdzonych transakcji. Teoretycznie można zdefiniować harmonogram  $S$  jako szeregowalny, jeżeli jego zatwierdzone rzutowanie  $C(S)$  jest równoważne pewnemu harmonogramowi szeregowemu, gdyż SZBD gwarantuje wyłącznie transakcje zatwierdzone.

W rozdziale 21. zostaną omówione różne protokoły sterowania współbieżnego, które gwarantują szeregowalność. Najczęściej stosowana technika, określana mianem *blokowania dwufazowego* (ang. *two-phase locking*), bazuje na blokowaniu elementów danych w celu zapobieżenia nachodzeniu na siebie współbieżnie wykonywanych transakcji oraz na wymuszeniu dodatkowego warunku, którego spełnienie gwarantuje szeregowalność. Rozwiązanie takie jest stosowane w niektórych komercyjnych SZBD. Omówimy też protokół oparty na *izolacji snapshotów*, gwarantujący szeregowalność w większości sytuacji, choć nie we wszystkich. Jest on stosowany w niektórych komercyjnych SZBD, ponieważ generuje niższe koszty niż protokół blokowania dwufazowego. Zaproponowano również inne protokoły<sup>14</sup>, do których należy *porządkowanie według znaczników czasu* (ang. *timestamp ordering*), gdzie do każdej transakcji przypisuje się unikatowy znacznik czasu i protokół zapewnia, że wszelkie operacje konfliktowe zostaną wykonane w kolejności zgodnej ze znacznikami czasu transakcji. Występują również *protokoły wielowersyjne* (ang. *multiversion protocols*), które są oparte na przechowywaniu wielu wersji elementów danych, oraz *protokoły optymistyczne* (nazywane również protokołami *certyfikacji* lub *walidacji*), które sprawdzają możliwe naruszenia szeregowalności po zakończeniu transakcji, ale przed dopuszczeniem ich zatwierdzenia.

---

<sup>14</sup> Jak dotąd nie były one jednak zbyt często używane w praktyce; większość systemów wykorzystuje taką czy inną odmianę protokołu blokowania dwufazowego.



## 20.5.4. Równoważność perspektywiczna i szeregowa

W podrozdziale 20.5.1 zdefiniowano pojęcia równoważności konfliktowej harmonogramów oraz szeregowa konfliktowej. Inną, mniej restrykcyjną definicją równoważności harmonogramów jest *równoważność perspektywiczna* (ang. *view equivalence*). Prowadzi ona do innej definicji szeregowa, znanej jako *szeregowa perspektywiczna*. O dwóch harmonogramach  $S$  i  $S'$  mówi się, że są **równoważne perspektywnie**, jeżeli spełnione są następujące trzy warunki:

- (1) Ten sam zbiór transakcji należy do harmonogramów  $S$  i  $S'$  i harmonogramy te zawierają te same operacje związane z tymi transakcjami.
- (2) Dla dowolnej operacji  $r_i(X)$  transakcji  $T_i$  w harmonogramie  $S$ , jeżeli wartość  $X$  odczytana przez pewną operację zostanie zapisana przez inną operację  $w_i(X)$  transakcji  $T_j$  (lub jeżeli jest to oryginalna wartość  $X$  sprzed rozpoczęcia harmonogramu), ten sam warunek musi być spełniony dla wartości  $X$  odczytanej przez operację  $r_i(X)$  transakcji  $T_i$  w harmonogramie  $S'$ .
- (3) Jeżeli operacja  $w_k(Y)$  transakcji  $T_k$  jest ostatnią operacją zapisującą element  $Y$  w ramach transakcji  $S$ , to operacja  $w_k(Y)$  transakcji  $T_k$  musi również być ostatnią operacją zapisującą element  $Y$  w ramach transakcji  $S'$ .

Idea równoważności perspektywicznej polega na tym, że o ile każda operacja odczytu danej transakcji odczytuje wynik tej samej operacji zapisu w obu harmonogramach, operacje zapisu każdej z tych transakcji muszą dawać te same wyniki. Stąd o operacjach odczytu mówi się, że *mają dostęp do tej samej perspektywy* w obu harmonogramach. Warunek 3. pozwala zapewnić, że końcowa operacja zapisu każdego elementu danych jest taka sama w obu harmonogramach, więc stan bazy danych powinien być taki sam po ich zakończeniu. O harmonogramie  $S$  mówi się, że jest **szeregowa perspektywnie**, jeżeli jest on równoważny perspektywnie harmonogramowi szeregowemu.

Definicje szeregowa konfliktowej oraz szeregowa perspektywicznej są podobne, jeżeli warunek znany jako **ograniczone założenie co do zapisu** (ang. *constrained write assumption*) lub **braku zapisów zamaskowanych** jest spełniony dla wszystkich transakcji w ramach harmonogramu. Warunek ten określa, że każda operacja zapisu  $w_i(X)$  transakcji  $T_i$  jest poprzedzona operacją odczytu  $r_i(X)$  transakcji  $T_i$  oraz że wartość zapisana przez operację  $w_i(X)$  transakcji  $T_i$  zależy tylko od wartości  $X$  odczytanej przez operację  $r_i(X)$ . Przyjmuje się, że obliczenie nowej wartości  $X$  jest funkcją  $f(X)$  opartą na starej wartości  $X$  odczytanej z bazy danych. **Zapis zamaskowany** (ang. *blind write*) to operacja zapisu elementu  $X$  w transakcji  $T$  niezależna od dawnej wartości  $X$ , dlatego taki zapis nie jest poprzedzony odczytem  $X$  w transakcji  $T$ .

Jednakże definicja szeregowa perspektywicznej jest mniej restrykcyjna od szeregowa konfliktowej w przypadku **nieograniczonego założenia co do zapisu** (ang. *unconstrained write assumption*), gdzie wartość zapisywana przez operację  $w_i(X)$  transakcji  $T_i$  może być niezależna od jej starej wartości pochodzącej z bazy danych. Jest tak, gdy *zapis zamaskowany* jest dozwolony; taką sytuację ilustruje poniższy harmonogram  $S_g$  trzech transakcji  $T_1$ :  $r_1(X)$ ;  $w_1(X)$ ;  $T_2$ :  $w_2(X)$ ; oraz  $T_3$ :  $w_3(X)$ :

$S_g$ :  $r_1(X)$ ;  $w_2(X)$ ;  $w_1(X)$ ;  $w_3(X)$ ;  $c_1$ ;  $c_2$ ;  $c_3$ ;

W przypadku harmonogramu  $S_g$  operacje  $w_2(X)$  i  $w_3(X)$  są zapisami zamaskowanymi, gdyż transakcje  $T_2$  i  $T_3$  nie odczytują wartości elementu  $X$ . Harmonogram  $S_g$  jest szeregowalny perspektywicznie, gdyż jest równoważny perspektywicznie harmonogramowi szeregowemu postaci  $T_1, T_2, T_3$ . Jednakże harmonogram  $S_g$  nie jest szeregowalny konfliktowo, gdyż nie jest równoważny konfliktowo żadnemu harmonogramowi szeregowemu (w ramach ćwiczenia Czytelnik powinien zbudować graf szeregowalności dla  $S_g$  i sprawdzić go pod kątem cykli). Wykazano, że każdy harmonogram szeregowalny konfliktowo jest również szeregowalny perspektywicznie, ale nie odwrotnie, czego dowodzi powyższy przykład. Istnieje algorytm sprawdzający, czy harmonogram  $S$  jest szeregowalny perspektywicznie. Udowodniono jednak, że problem testowania szeregowalności perspektywicznej jest NP-łożony, co oznacza, że znalezienie wydajnego algorytmu rozwiązania tego problemu, który działałby w czasie wielomianowym, jest mało prawdopodobne.

## 20.5.5. Inne rodzaje równoważności harmonogramów

Szeregowalność harmonogramów jest niekiedy uważana za zbyt restrykcyjną jako warunek zapewnienia poprawności współbieżnego wykonania. Pewne aplikacje mogą tworzyć poprawne harmonogramy, spełniające mniej surowe warunki niż szeregowalność konfliktowa lub perspektywiczna. Przykładem mogą tu być transakcje znane jako **transakcje Winien-Ma** (ang. *debit-credit transactions*), które występują na przykład w przypadku depozytów i podejmowania kwot z konta bankowego. Semantyka operacji Winien-Ma opiera się na tym, że aktualizują one wartość elementu danych  $X$  albo przez odjęcie, albo dodanie do bieżącej wartości elementu danych. Ze względu na fakt, że operacje dodawania i odejmowania są przemienne, to znaczy, mogą być stosowane w dowolnej kolejności, istnieje możliwość utworzenia poprawnych harmonogramów, które nie są szeregowalne. Przykładowo, weźmy pod uwagę następujące dwie transakcje, z których każda może być wykorzystana do przelania pewnej kwoty między dwoma kontami bankowymi:

$T_1: r_1(X); X := X - 10; w_1(X); r_1(Y); Y := Y + 10; w_1(Y);$

$T_2: r_2(Y); Y := Y - 20; w_2(Y); r_2(X); X := X + 20; w_2(X);$

Weźmy pod uwagę następujący nieszeregowalny harmonogram  $S_h$  dla tych dwóch transakcji:

$S_h: r_1(X); w_1(X); r_2(Y); w_2(Y); r_1(Y); w_1(Y); r_2(X); w_2(X);$

Posiadając dodatkową wiedzę (**semantykę**) o tym, że operacje występujące między każdą operacją  $r_i(I)$  a  $w_i(I)$  są przemienne, wiemy, że kolejność wykonania sekwencji składających się z operacji (odczyt, aktualizacja, zapis) nie ma znaczenia, o ile każda taka sekwencja określonej transakcji  $T_i$  na określonym elemencie  $I$  nie zostanie przerwana przez operacje konfliktowe. Stąd harmonogram  $S_h$  uważa się za poprawny, pomimo że nie jest szeregowalny. Obecnie toczą się prace nad rozszerzeniem teorii sterowania współbieżnego w celu uwzględnienia przypadków, w których szeregowalność jest uważana za zbyt restrykcyjny warunek poprawności harmonogramów. Ponadto w aplikacjach z pewnych dziedzin, np. w projektowaniu wspomaganim komputerowo (ang. *computer-aided design* — CAD) w obszarze złożonych systemów takich jak samoloty, transakcje są długotrwałe. Na potrzeby takich aplikacji zaproponowano mniej restrykcyjne modele sterowania współbieżnego służące zachowaniu spójności baz danych. Jeden z tych modeli dotyczy *spójności ostatecznej* (ang. *eventual consistency*). Spójność ostateczną opiszemy w kontekście rozproszonych baz danych w rozdziale 23.

## 20.6. Obsługa transakcji w języku SQL

W tym podrozdziale przedstawimy krótkie wprowadzenie do obsługi transakcji w języku SQL. Z tym zagadnieniem związanych jest wiele szczegółów, a w nowych standardach występuje więcej poleceń z zakresu przetwarzania transakcji. Podstawowa definicja transakcji SQL jest podobna do już zdefiniowanego pojęcia transakcji. Oznacza to, że jest to logiczna jednostka pracy, której niepodzielność jest zagwarantowana. Pojedyncza instrukcja SQL jest zawsze traktowana jako niepodzielna — albo kończy swoje działanie bez błędu, albo ulega uszkodzeniu i pozostawia bazę w niezmiennym stanie.

W języku SQL nie istnieje jawna instrukcja `Rozpocznij_Transakcję`. Inicjalizacja transakcji jest przeprowadzana niejawnie w momencie występowania określonych instrukcji SQL. Jednakże każda transakcja musi posiadać jawnie określoną instrukcję zakończenia, którą jest `COMMIT` lub `ROLLBACK`. Każda transakcja posiada określone cechy. Określa się je za pomocą instrukcji SQL `SET TRANSACTION`. Cechami tymi są: *tryb dostępu*, *rozmiar obszaru diagnostycznego* oraz *poziom izolacji*.

**Tryb dostępu** (ang. *access level*) można określić jako `READ ONLY` lub `READ WRITE`. Wartością domyślną jest `READ WRITE`, chyba że określono poziom izolacji `READ UNCOMMITTED` (patrz niżej), kiedy to przyjmuje się występowanie trybu `READ ONLY`. Tryb `READ WRITE` pozwala na wykonywanie instrukcji aktualizowania, wstawiania, usuwania i tworzenia. Tryb `READ ONLY` służy wyłącznie do pobierania danych.

**Rozmiar obszaru diagnostycznego** (ang. *diagnostic area size*), `DIAGNOSTIC SIZE n`, określa wartość całkowitą  $n$ , wskazującą liczbę warunków, które mogą być jednocześnie przechowywane w obszarze diagnostycznym. Warunki takie określają informacje zwrotne (błędy lub wyjątki) skierowane do użytkownika lub programu przez ostatnio wykonanych  $n$  poleceń SQL.

**Poziom izolacji** (ang. *isolation level*) określa się przy użyciu instrukcji `ISOLATION LEVEL <poziom_isolacji>`, gdzie wartością `<poziom_isolacji>` może być `READ UNCOMMITTED`, `READ COMMITTED`, `REPEATABLE READ` lub `SERIALIZABLE`<sup>15</sup>. Domyślnym poziomem izolacji jest `SERIALIZABLE`, aczkolwiek pewne systemy jako wartości domyślnej używają `READ COMMITTED`. Użycie pojęcia `SERIALIZABLE` (szeregowalny) w tym przypadku opiera się na niedopuszczeniu do odczytów zmodyfikowanych, odczytów niepowtarzalnych oraz fantomów<sup>16</sup>, i stąd nie jest równoważne szeregowalności, którą zdefiniowaliśmy wcześniej w podrozdziale 20.5. Jeżeli transakcja jest wykonywana na niższym poziomie izolacji niż `SERIALIZABLE`, może występować jedno lub więcej spośród trzech poniższych naruszeń:

- (1) **Odczyt zmodyfikowany.** Transakcja  $T_1$  może odczytać wartość zaktualizowaną przez transakcję  $T_2$ , której jeszcze nie zatwierdzono. Jeżeli transakcja  $T_2$  ulegnie uszkodzeniu i zostanie anulowana, okaże się, że transakcja  $T_1$  odczytała nieistniejącą i niepoprawną wartość.

---

<sup>15</sup> Są one podobne do *poziomów izolacji* omówionych skrótowo pod koniec podrozdziału 20.3.

<sup>16</sup> Problemy odczytów zmodyfikowanych i odczytów niepowtarzalnych omówiono w podrozdziale 20.1.3. Fantomy zostaną omówione w podrozdziale 22.7.1.

- (2) **Odczyt niepowtarzalny.** Transakcja  $T_1$  może odczytać określoną wartość z tabeli. Jeżeli inna transakcja  $T_2$  zaktualizuje później tę wartość, a transakcja  $T_1$  odczyta ją ponownie, będzie to inny odczyt.
- (3) **Fantomy.** Transakcja  $T_1$  może odczytać zbiór wierszy z tabeli, na przykład w oparciu o pewien warunek określony w klauzuli `WHERE` zapytania SQL. Załóżmy, że transakcja  $T_2$  wstawia nowy wiersz, który również spełnia warunek klauzuli `WHERE` użytej przez transakcję  $T_1$ , do tabeli używanej przez  $T_1$ . Rekord  $r$  jest nazywany **fantomowym**, ponieważ nie istniał w momencie rozpoczęcia transakcji  $T_1$ , ale jest dostępny w momencie jej zakończenia. Transakcja  $T_1$  może (choć nie zawsze tak się dzieje) zobaczyć wiersz fantomowy, który wcześniej nie istniał. Jeśli równoważna kolejność szeregową to wykonanie najpierw  $T_1$ , a później  $T_2$ , rekord  $r$  nie powinien być widoczny. Jeżeli jednak kolejność to najpierw  $T_2$ , a później  $T_1$ , rekord fantomowy powinien zostać uwzględniony w wynikach przekazywanych do  $T_1$ . Gdy system nie może zagwarantować poprawności działania, nie obsługuje problemu rekordów fantomowych.

W tabeli 20.1 zawarto podsumowanie możliwych naruszeń różnych poziomów izolacji. Wpis „tak” oznacza, że naruszenie jest możliwe, zaś wpis „nie”, że nie jest ono możliwe. Poziom `READ UNCOMMITTED` jest najmniej restrykcyjny, a poziom `SERIALIZABLE` — najbardziej wymagający i pozwala uniknąć wszystkich trzech opisanych wcześniej problemów.

TABELA 20.1. Możliwe naruszenia w oparciu o poziomy izolacji zdefiniowane w języku SQL

Poziom izolacji	Rodzaj naruszenia		
	Odczyt zmodyfikowany	Odczyt niepowtarzalny	Fantom
READ UNCOMMITTED	tak	tak	tak
READ COMMITTED	nie	tak	tak
REPEATABLE READ	nie	nie	tak
SERIALIZABLE	nie	nie	nie

Przykładowa transakcja SQL może mieć następującą postać:

```
EXEC SQL WHENEVER SQLERROR GOTO UNDO;
EXEC SQL SET TRANSACTION
    READ WRITE
    DIAGNOSTIC SIZE 5
    ISOLATION LEVEL SERIALIZABLE;
EXEC SQL INSERT INTO PRACOWNIK (IMIE, NAZWISKO, PESEL, NRDZ, PENSJA)
    VALUES ('JAN', 'KOWALSKI', '69110204312', 2, 35000);
EXEC SQL UPDATE PRACOWNIK
    SET PENSJA = PENSJA * 1.1 WHERE NRDZ = 2;
EXEC SQL COMMIT;
GOTO KONIEC;
UNDO: EXEC SQL ROLLBACK;
KONIEC: ...;
```

Powyższa transakcja najpierw wstawia nowy wiersz do tabeli `PRACOWNIK`, a następnie aktualizuje pensję wszystkich pracowników pracujących w dziale o numerze 2. W razie wystąpienia błędu którejś z instrukcji SQL cała transakcja zostaje wycofana. Implikuje to, że

każda wartość pensji zaktualizowana przez tę transakcję zostałaby przywrócona do swojej wcześniejszej postaci oraz że nowo wstawiony wiersz zostałby usunięty.

Jak pokazano, język SQL oferuje wiele funkcji związanych z obsługą transakcji. Administrator bazy danych lub programiści mogą wykorzystywać dostępne opcje w celu podjęcia próby usprawnienia wydajności transakcji poprzez zmniejszenie ograniczeń co do szeregowalności, o ile będzie to dopuszczalne z punktu widzenia aplikacji.

**Izolacja snapshotów.** Następnym poziom izolacji, izolacja snapshotów, jest stosowany w niektórych komercyjnych SZBD. Istnieją protokoły sterowania współbieżnego oparte na tym podejściu. Podstawowa zasada **izolacji snapshotów** polega na tym, że transakcja widzi wczytywane elementy danych w postaci zatwierdzonych wartości elementów ze *snapshotu bazy danych* (stanu bazy) z momentu rozpoczęcia tej transakcji. Izolacja snapshotów gwarantuje, że problem rekordów fantomowych nie wystąpi, ponieważ transakcja (lub instrukcja) w bazie będzie miała dostęp tylko do rekordów zatwierdzonych w momencie rozpoczęcia danej transakcji. Wszystkie operacje wstawiania, usuwania lub aktualizacji rozpoczęte po uruchomieniu transakcji nie będą w niej widoczne. W rozdziale 21. opiszemy protokół sterowania współbieżnego oparty na tym rozwiązaniu.

## 20.7. Podsumowanie

W niniejszym rozdziale omówiono pojęcia z obszaru SZBD dotyczące przetwarzania transakcji. W podrozdziale 20.1 przedstawiono pojęcie transakcji bazodanowej oraz operacji dotyczących przetwarzania transakcji. Porównano systemy jedno- i wieloużytkownikowe, a następnie przedstawiono przykłady tego, w jaki sposób niekontrolowane wykonywanie współbieżnych transakcji w systemie wieloużytkownikowym może prowadzić do powstawania błędnych wyników i wartości w bazie danych (punkt 20.1.1). Omówiono także różne rodzaje awarii, które mogą wystąpić w trakcie wykonywania transakcji (punkt 20.1.4).

Następnie (w podrozdziale 20.2) przedstawiono typowe stany, przez które przechodzi transakcja w czasie swojego wykonywania, i omówiono kilka pojęć używanych w zakresie metod odtwarzania i sterowania współbieżnego. Dziennik systemowy (punkt 20.2.2) śledzi operacje dostępu do bazy danych i system wykorzystuje te informacje w celu przywracania do stanu normalnego po awarii. Transakcja albo kończy się powodzeniem i osiąga swój punkt zatwierdzenia, albo ulega uszkodzeniu i musi zostać wycofana. Zmiany wprowadzone przez zatwierdzoną transakcję (punkt 20.2.3) są trwale zapisywane w bazie danych. W podrozdziale 20.3 przedstawiono ogólny przegląd pożądanych właściwości transakcji, a ściślej: niepodzielności, zachowania spójności, izolacji oraz trwałości. Określa się je często mianem właściwości ACID.

W punkcie 20.4.1 zdefiniowano harmonogram (inaczej, historię) jako sekwencję wykonania operacji kilku transakcji z możliwym przeplotem. Harmonogramy scharakteryzowano pod względem ich odtwarzalności (punkt 20.4.2). Harmonogramy odtwarzalne zapewniają, że po zatwierdzeniu transakcji nigdy nie zachodzi potrzeba anulowania takiej operacji. Harmonogramy bezkaskadowe dodają warunek zapewniający, że żadna anulowana transakcja nie będzie wymagała kaskadowego anulowania innych transakcji. Harmonogramy ścisłe oferują jeszcze silniejszy warunek, który dopuszcza prosty schemat

odtworzenia polegający na przywróceniu starych wartości elementów, które zostały zmienione przez anulowaną transakcję.

W podrozdziale 20.5 zdefiniowano równoważność harmonogramów i pokazano, że każdy harmonogram szeregowalny jest równoważny pewnemu harmonogramowi szeregowemu. Zdefiniowano także pojęcia równoważności konfliktowej i równoważności perspektywicznej. Harmonogram szeregowalny uważa się za poprawny. W punkcie 20.5.2 przedstawiono algorytmy testowania (konfliktowej) szeregowalności harmonogramów. Wyjaśniono, dlaczego testowanie szeregowalności jest niepraktyczne w rzeczywistych systemach, choć może być używane w celu definiowania i weryfikowania protokołów sterowania współbieżnego (punkt 20.5.3). W punktach 20.5.4 i 20.5.5 omówiono pokrótce mniej restrykcyjne definicje równoważności harmonogramów. Wreszcie w podrozdziale 20.6 przedstawiono ogólne omówienie sposobów praktycznego wykorzystania pojęć dotyczących transakcji w języku SQL, a także izolację snapshotów używaną w różnych komercyjnych SZBD.

## Pytania powtórkowe

- 20.1. Co rozumiemy przez współbieżne wykonywanie transakcji bazodanowych w systemie wieloużytkownikowym? Omów, dlaczego sterowanie współbieżnością jest konieczne i podaj nieformalne przykłady.
- 20.2. Omów różne rodzaje awarii. Co rozumiemy pod pojęciem awarii katastroficznej?
- 20.3. Omów działania podejmowane przez operacje `odczytaj_element` i `zapisz_element` w bazie danych.
- 20.4. Narysuj diagram stanów i omów typowe stany, przez które przechodzi transakcja w czasie swojego wykonania.
- 20.5. W jakim celu jest używany dziennik systemowy? Jakie typowe rodzaje rekordów występują w dzienniku systemowym? Czym są punkty zatwierdzenia i na czym polega ich istotność?
- 20.6. Omów właściwości niepodzielności, trwałości, izolacji i zachowania spójności transakcji bazodanowych.
- 20.7. Czym jest harmonogram (historia)? Zdefiniuj pojęcia harmonogramów odtwarzalnych, bezkaskadowych i ścisłych oraz porównaj je pod względem ich odtwarzalności.
- 20.8. Omów różne wskaźniki równoważności transakcji. Jaka istnieje różnica między równoważnością konfliktową a równoważnością perspektywiczną?
- 20.9. Czym jest harmonogram szeregowy? Czym jest harmonogram szeregowalny? Dlaczego harmonogram szeregowy uważa się za poprawny? Dlaczego harmonogram szeregowalny uważa się za poprawny?
- 20.10. Jaka jest różnica między ograniczonym a nieograniczonym założeniem co do zapisu? Które z nich jest bardziej realistyczne?
- 20.11. Omów, w jaki sposób szeregowalność jest wykorzystywana w celu wymuszania sterowania współbieżnego w systemie bazodanowym. Dlaczego czasem szeregowalność uważa się za zbyt restrykcyjny wskaźnik poprawności harmonogramów?



- 20.12. Opisz cztery poziomy izolacji języka SQL. Omów też izolację snapshotów i jej wpływ na problem rekordów fantomowych.
- 20.13. Zdefiniuj naruszenia powodowane przez każdą z następujących operacji: odczyt zmodyfikowany, odczyt niepowtarzalny, fantomy.

## Ćwiczenia

- 20.14. Zmień transakcję z rysunku 20.2(b) na postać:

```
odczytaj_element(X);
X := X+M;
if X > 90 then exit
else zapisz_element(X);
```

Omów końcowy wynik różnych harmonogramów z rysunków 20.3(a) i (b), gdzie  $M = 2$  oraz  $N = 2$ , w kontekście następujących pytań. Czy dodanie powyższego warunku zmienia wynik końcowy? Czy wynik spełnia warunki założonej reguły spójności (określającej, że pojemność  $X$  wynosi 90)?

- 20.15. Powtórz ćwiczenie 20.14, dodając sprawdzenie w transakcji  $T_1$ , tak aby  $Y$  nie przekraczało 90.
- 20.16. Dodaj operację zatwierdzenia na końcu każdej z transakcji  $T_1$  i  $T_2$  z rysunku 20.2. Następnie wymień wszystkie możliwe harmonogramy dla zmodyfikowanych transakcji. Określ, które z nich są odtwarzalne, które bezkaskadowe, a które ścisłe.
- 20.17. Wymień wszystkie możliwe harmonogramy dla transakcji  $T_1$  i  $T_2$  z rysunku 20.2 oraz określ, które z nich są szeregowe (poprawne), a które nie.
- 20.18. Ile harmonogramów *szeregowych* istnieje dla trzech transakcji z rysunku 20.8(a)? Czym one są? Jaka jest całkowita liczba możliwych harmonogramów?
- 20.19. Napisz program tworzący wszystkie możliwe harmonogramy dla trzech transakcji z rysunku 20.8(a) i określający, które z tych harmonogramów są szeregowe (poprawne), a które nie. Dla każdego harmonogramu szeregowego (poprawnego) program powinien go wyświetlać i wymieniać wszystkie równoważne harmonogramy szeregowe.
- 20.20. Dlaczego w języku SQL wymagana jest jawna instrukcja końca transakcji, a nie jest wymagana instrukcja początku?
- 20.21. Opisz sytuacje, w których każdy z różnych poziomów izolacji mógłby być przydatny w zakresie przetwarzania transakcji.
- 20.22. Które z poniższych harmonogramów są (konfliktowo) szeregowe? Dla każdego harmonogramu szeregowego określ równoważne harmonogramy szeregowe.
- $r_1(X); r_3(X); w_1(X); r_2(X); w_3(X);$
  - $r_1(X); r_3(X); w_3(X); w_1(X); r_2(X);$
  - $r_3(X); r_2(X); w_3(X); r_1(X); w_1(X);$
  - $r_3(X); r_2(X); r_1(X); w_3(X); w_1(X);$



- 20.23. Weźmy pod uwagę trzy transakcje  $T_1$ ,  $T_2$  i  $T_3$  oraz harmonogramy  $S_1$  i  $S_2$  przedstawione poniżej. Narysuj grafy szeregowości (pierwszeństwa) dla harmonogramów  $S_1$  i  $S_2$  oraz określ, czy każdy z nich jest szeregowalny, czy nie. Jeżeli harmonogram jest szeregowalny, zapisz równoważny(e) harmonogram(y) szeregowy(e).

$$T_1: r_1(X); r_1(Z); w_1(X);$$

$$T_2: r_2(Z); r_2(Y); w_2(Z); w_2(Y);$$

$$T_3: r_3(X); r_3(Y); w_3(Y);$$

$$S_1: r_1(X); r_2(Z); r_1(Z); r_3(X); r_3(Y); w_1(X); w_3(Y); r_2(Y); w_2(Z); w_2(Y);$$

$$S_2: r_1(X); r_2(Z); r_3(X); r_1(Z); r_2(Y); r_3(Y); w_1(X); w_2(Z); w_3(Y); w_2(Y);$$

- 20.24. Weźmy pod uwagę poniższe harmonogramy  $S_3$ ,  $S_4$  i  $S_5$ . Określ, czy każdy harmonogram jest ścisły, bezkaskadowy, odtwarzalny lub nieodtworzalny (określ najsurowszy warunek odtwarzalności spełniany przez każdy z harmonogramów).

$$S_3: r_1(X); r_2(Z); r_1(Z); r_3(X); r_3(Y); w_1(X); c_1; w_3(Y); c_3; r_2(Y); w_2(Z); w_2(Y); c_2;$$

$$S_4: r_1(X); r_2(Z); r_1(Z); r_3(X); r_3(Y); w_1(X); w_3(Y); r_2(Y); w_2(Z); w_2(Y); c_1; c_2; c_3;$$

$$S_5: r_1(X); r_2(Z); r_3(X); r_1(Z); r_2(Y); r_3(Y); w_1(X); c_1; w_2(Z); w_3(Y); w_2(Y); c_3; c_2;$$

## Wybrane publikacje

Szeregowość i powiązane z nią pomysły z zakresu utrzymywania spójności w bazach danych przedstawiono w pracy Graya i in. (1975). Pojęcie transakcji omówiono w pozycji Graya (1981). W 1998 r. Gray otrzymał cioną nagrodę ACM Turing Award za pracę nad transakcjami bazodanowymi i implementowaniem transakcji w relacyjnych SZBD. Pozycja Bernsteina, Hadzilacosa i Goodmana (1988) skupia się na technikach sterowania współbieżnego i odtwarzania zarówno w scentralizowanych, jak i rozproszonych systemach bazodanowych. Stanowi doskonałe źródło informacji. Papadimitriou (1986) oferuje spojrzenie z bardziej teoretycznej perspektywy. Obszerna, ponad 1000-stronicowa książka Graya i Reutera (1993) prezentuje spojrzenie z perspektywy praktycznej na pojęcia i techniki przetwarzania transakcji. Elmagarmid (1992) prezentuje kolekcję artykułów naukowych dotyczących przetwarzania transakcji z zaawansowanych aplikacji. Obsługa transakcji w języku SQL została opisana w książce Date'a i Darwena (1997). Szeregowość perspektywiczna została zdefiniowana w książce Yannakakisa (1984). Odtwarzalność harmonogramów i niezawodność baz danych stanowią treść pozycji Hadzilacosa (1983, 1988). Chou i DeWitt (1985) opisali strategię zastępowania buforów. Ports i Grittner (2012) omówili izolację snapshotów.

## Techniki sterowania współbieżnego

W niniejszym rozdziale zostanie omówionych wiele technik sterowania współbieżnego, które są wykorzystywane w celu zapewnienia właściwości niekolidowania, czyli izolacji, transakcji wykonywanych współbieżnie. Większość tych technik zapewnia szeregowalność harmonogramów (patrz podrozdział 21.5) przy użyciu **protokołów sterowania współbieżnego** (to znaczy, zestawów reguł) gwarantujących szeregowalność. Jeden z ważnych zbiorów protokołów — *protokoły blokowania dwufazowego* — stosuje technikę **blokowania** (ang. *locking*) elementów danych w celu zapobieżenia współbieżnemu uzyskiwaniu dostępu do elementów przez wiele transakcji. W podrozdziale 18.1 i punkcie 21.3.2 zostanie opisanych kilka protokołów blokowania. Protokoły z blokowaniem są wykorzystywane w niektórych komercyjnych SZBD, jednak uważa się, że generują wysokie koszty. Inna grupa protokołów sterowania współbieżnego wykorzystuje **znaczniki czasowe** (ang. *timestamps*). Znacznik czasowy jest unikatowym identyfikatorem każdej transakcji, generowanym przez system. Wartości znaczników czasowych są generowane zgodnie z kolejnością momentów rozpoczęcia transakcji. Protokoły sterowania współbieżnego, które wykorzystują uporządkowanie według znaczników czasu w celu zapewnienia szeregowalności, zostaną opisane w podrozdziale 21.2. W podrozdziale 21.3 zostaną omówione **wielowersyjne** (ang. *multiversion*) protokoły sterowania współbieżnego, które wykorzystują wiele wersji tych samych elementów danych. Jeden z takich protokołów obok kolejności opartej na znacznikach czasu uwzględnia dodatkowo wersje (punkt 21.3.1). Inny protokół oprócz znaczników czasu stosuje blokowanie dwufazowe (punkt 21.3.2). W podrozdziale 21.4 zostanie przedstawiony protokół bazujący na pojęciu **walidacji** (inaczej **certyfikacji**) transakcji po wykonaniu jej operacji. Tego rodzaju protokoły określa się mianem **protokołów optymistycznych** (ang. *optimistic protocol*). Także w nich przyjmuje się, że może istnieć wiele wersji elementu danych. W podrozdziale 21.4 opisano też protokół oparty na **izolacji snapshotów**, w którym można stosować techniki podobne do metod wykorzystujących walidację i wiele wersji. Protokoły tego typu są używane w wielu komercyjnych SZBD i w niektórych sytuacjach generują niższe koszty niż protokoły oparte na blokadach.

Kolejnym czynnikiem wpływającym na sterowanie współbieżne jest **ziarnistość** (ang. *granularity*) elementów danych — to znaczy, jaką część bazy danych reprezentuje dany element. Element może być tak mały jak wartość pojedynczego atrybutu (pola) lub tak duży jak blok dyskowy, a nawet cały plik lub baza danych. Ziarnistość elementów oraz protokołów sterowania współbieżnego z użyciem wielu ziarnistości (jest on rozszerzeniem blokowania dwufazowego) to temat podrozdziału 21.5. W podrozdziale 21.6 zostaną omówione kwestie związane ze sterowaniem współbieżnym, które występują w przypadku użycia indeksów do przetwarzania transakcji. Wreszcie w podrozdziale 21.7 zostaną omówione pewne dodatkowe zagadnienia związane ze sterowaniem współbieżnym. Podrozdział 21.8 zawiera podsumowanie rozdziału.

Jeżeli interesuje Cię głównie wprowadzenie do technik sterowania współbieżnego opartych na blokadach, wystarczy przeczytać podrozdziały 21.1, 21.5, 21.6 i 21.7 oraz opcjonalnie punkt 21.3.2.

## 21.1. Techniki blokowania dwufazowego dla celów sterowania współbieżnego

Niektóre z podstawowych technik używanych do sterowania współbieżnym wykonywaniem transakcji są oparte na pojęciu blokowania elementów danych. **Blokada** (ang. *lock*) jest zmienną związaną z elementem danych, która opisuje stan tego elementu pod względem możliwości działań, jakie mogą być na nim wykonywane. Ogólnie rzecz biorąc, w bazie danych na każdy element danych przypada jedna blokada. Blokadę są używane jako środek synchronizowania dostępu współbieżnych transakcji do elementów bazy danych. W punkcie 21.1.1 zostaną omówione cechy i rodzaje blokad. Następnie, w punkcie 21.1.2, zostaną przedstawione protokoły wykorzystujące blokowanie w celu zagwarantowania szeregowalności harmonogramów transakcji. Wreszcie w punkcie 21.1.3 zostaną omówione dwa problemy związane z użyciem blokad — zakleszczenie i zgłodzenie — oraz sposoby radzenia sobie z nimi w protokołach sterowania współbieżnego.

### 21.1.1. Rodzaje blokad i systemowe tabele blokad

W sterowaniu współbieżnym używa się kilku rodzajów blokad. W celu stopniowego prezentowania kolejnych pojęć dotyczących blokowania, najpierw omówimy blokady binarne, które są proste, ale jednocześnie zaledwie restrykcyjne w kontekście sterowania współbieżnego w bazach danych i w praktyce w zasadzie się ich nie stosuje. Następnie zostaną omówione *blokadę dzielone i wyłączne* (blokadę do odczytu i do zapisu), które oferują bardziej ogólne możliwości w zakresie blokowania i są używane w praktycznych schematach blokowania w bazach danych. W punkcie 21.3.2 zostanie opisana *blokada certyfikacji* oraz pokazane, w jaki sposób można jej używać w celu zwiększenia wydajności działania protokołów blokujących.

**Blokady binarne.** **Blokada binarna** (ang. *binary block*) może posiadać dwa **stany** lub **wartości**: *zablokowany* i *odblokowany* (dla uproszczenia — 1 i 0). Z każdym elementem  $X$  bazy danych jest związana odrębna blokada. Jeżeli wartość blokady dla  $X$  wynosi 1, do elementu  $X$  *nie mogą uzyskać dostępu* operacje bazodanowe. Jeżeli wartością blokady dla  $X$  jest 0, dostęp jest możliwy. Po uzyskaniu dostępu wartość blokady jest ustawiana na 1. Bieżącą wartość (stan) blokady związanej z elementem  $X$  określamy jako  $BLOKADA(X)$ .

W przypadku blokad binarnych używa się dwóch operacji, `blokuj_element` i `odblokuj_element`. Transakcja żąda dostępu do elementu  $X$ , wykonując najpierw operację `blokuj_element(X)`. Jeżeli  $BLOKADA(X) = 1$ , transakcja musi czekać. Jeżeli  $BLOKADA(X) = 0$ , zostaje ustawiona na wartość 1 (transakcja **blokuje** element) i transakcja może uzyskać dostęp do elementu  $X$ . Po zakończeniu operacji na elemencie  $X$  transakcja wykonuje operację `odblokuj_element(X)`, która ustawia wartość  $BLOKADA(X)$  na 0 (**odblokuje** element), więc

do elementu  $X$  mogą uzyskiwać dostęp inne transakcje. Stąd blokada binarna wymusza **wzajemne wykluczenie** (ang. *mutual exclusion*) na elemencie danych. Opis operacji `blokuj_element(X)` oraz `odblokuj_element(X)` przedstawiono na rysunku 21.1.

```

blokuj_element(X)
B: if BLOKADA(X) = 0 (* element jest odblokowany *)
    then BLOKADA(X) ← 1; (* zablokowanie elementu *)
    else
        begin
            czekaj (dopóki BLOKADA(X) ≠ 0
                    i menedżer blokad uruchomi transakcję);
            go to B;
        end;
odblokuj_element(X)
BLOKADA(X) ← 0; (* odblokowanie elementu *)
if jakieś transakcje oczekują
    then uruchom którąś z oczekujących transakcji;

```

RYSunek 21.1. Operacje blokowania i odblokowania w przypadku blokad binarnych

Należy zauważyć, że operacje `blokuj_element` i `odblokuj_element` muszą zostać zaimplementowane jako jednostki niepodzielne (znane jako **sekcje krytyczne** (ang. *critical sections*) w systemach operacyjnych). Oznacza to, że nie powinien być dozwolony przeplot w momencie rozpoczęcia operacji blokowania lub odblokowania aż do czasu ich zakończenia. Na rysunku 21.1 polecenie oczekiwania w ramach operacji `blokuj_element(X)` jest zwykle implementowane przez umieszczenie transakcji w kolejce oczekiwania na element  $X$ , dopóki nie zostanie on odblokowany i transakcja będzie mogła uzyskać do niego dostęp. Inne transakcje, które również chcą uzyskać dostęp do elementu  $X$ , są umieszczane w tej samej kolejce. Stąd polecenie oczekiwania jest traktowane jako występujące poza operacją `blokuj_element`.

Implementacja blokady binarnej jest dość prosta. Wszystko, czego potrzeba, to zmienna dwuwartościowa, `BLOKADA`, związana z każdym elementem danych  $X$  w bazie. W swojej najprostszej postaci każda blokada może być rekordem złożonym z trzech pól: *<nazwa elementu danych, BLOKADA, transakcja blokująca>* i towarzyszącą mu kolejką dla transakcji, które oczekują na uzyskanie dostępu. W **tabeli blokad** (ang. *lock table*), którą można zaimplementować jako tablicę mieszającą, system musi przechowywać *tylko rekordy dla elementów, które są w danej chwili zablokowane*. Elementy niezajmujące się w tabeli blokad są traktowane jako niezablokowane. SZBD posiada **podsystem menedżera blokad**, odpowiedzialny za śledzenie i sterowanie dostępem do blokad.

W razie użycia prostego schematu blokowania opisanego powyżej każda transakcja musi przestrzegać następujących reguł:

- (1) Transakcja  $T$  musi wykonać operację `blokuj_element(X)` przed wykonaniem jakichkolwiek operacji `odczytaj_element(X)` lub `zapisz_element(X)`.
- (2) Transakcja  $T$  musi wykonać operację `odblokuj_element(X)` po zakończeniu wszystkich operacji `odczytaj_element(X)` lub `zapisz_element(X)`.

- (3) Transakcja  $T$  nie wykonuje operacji `blokuj_element( $X$ )`, jeżeli już trzyma blokadę na elemencie  $X$ <sup>1</sup>.
- (4) Transakcja  $T$  nie wykonuje operacji `odblokuj_element( $X$ )`, chyba że trzyma już blokadę na elemencie  $X$ .

Reguły te może wymuszać moduł menedżera blokad SZBD. Między operacjami `blokuj_element( $X$ )` oraz `odblokuj_element( $X$ )` w ramach transakcji  $T$  o transakcji tej mówi się, że **trzyma blokadę** na elemencie  $X$ . Najwyżej jedna transakcja może trzymać blokadę na określonym elemencie. Stąd żadne dwie transakcje nie mogą uzyskać współbieżnego dostępu do tego samego elementu.

**Blokady dzielone i wyłączne (odczytu i zapisu).** Omówiony powyżej binarny schemat blokowania jest zbyt restrykcyjny w przypadku elementów bazy danych, ponieważ tylko jedna transakcja może trzymać blokadę na danym elemencie. Należy pozwolić, aby kilka transakcji mogło uzyskiwać dostęp do tego samego elementu  $X$ , jeżeli wszystkie uzyskują dostęp *tylko w celu odczytu*. Wynika to z tego, że operacje odczytu tego samego elementu przez różne transakcje są *bezkonfliktowe* (patrz punkt 21.4.1). Jednakże jeżeli transakcja ma zapisać element  $X$ , musi posiadać wyłączny dostęp do tego elementu. Używa się tu innego rodzaju blokady, noszącej nazwę **blokady wielotrybowej** (ang. *multiple-mode lock*). W przypadku takiego schematu — określanego mianem blokad **dzielonych i wyłącznych** (ang. *shared/exclusive*) lub **odczytu i zapisu** (ang. *read/write*) — występują trzy operacje blokowania: `blokuj_do_odczytu( $X$ )`, `blokuj_do_zapisu( $X$ )` oraz `odblokuj( $X$ )`. Blokada związana z elementem  $X$ , `BLOKADA( $X$ )`, posiada teraz trzy możliwe stany: *zablokowany\_do\_odczytu*, *zablokowany\_do\_zapisu* oraz *odblokowany*. **Element zablokowany do odczytu** (ang. *read-locked item*) jest również określanym mianem **elementu zablokowanego ze współużytkowaniem** (ang. *shared-locked item*), ponieważ inne transakcje mogą odczytywać taki element, natomiast **element zablokowany do zapisu** (ang. *write-locked item*) jest również określanym mianem **elementu zablokowanego na wyłączność** (ang. *exclusive-locked item*), ponieważ tylko jedna transakcja trzyma blokadę na takim elemencie.

Jedną z metod implementacji wymienionych operacji jest przechowywanie informacji o liczbie transakcji trzymających dzieloną (do odczytu) blokadę na elemencie w tabeli blokad. Przechowywana jest też lista identyfikatorów transakcji trzymających blokadę współużytkowaną. Każdy rekord takiej tabeli zawiera cztery pola: *<nazwa elementu danych, BLOKADA, liczba\_odczytów, transakcja( $e$ )\_blokująca( $e$ )>*. Ponownie, w celu zaoszczędzenia przestrzeni, system przechowuje w tabeli blokad rekordy tylko dla elementów zablokowanych. Wartością (stanem) zmiennej `BLOKADA` jest albo *zablokowany\_do\_odczytu*, albo *zablokowany\_do\_zapisu* po odpowiednim ich zakodowaniu (jeżeli założymy, że w tabeli blokad nie są przechowywane żadne informacje o elementach niezablokowanych). Jeżeli `BLOKADA( $X$ ) = zablokowany_do_zapisu`, wartością pola *transakcja( $e$ )\_blokująca( $e$ )* jest *pojedyncza transakcja* trzymająca wyłączną (do zapisu) blokadę na elemencie  $X$ . Jeżeli `BLOKADA( $X$ ) = zablokowany_do_odczytu`, wartością pola *transakcja( $e$ )\_blokująca( $e$ )* jest lista transakcji trzymających dzieloną (do odczytu) blokadę na elemencie  $X$ . Trzy operacje

<sup>1</sup> Regułę tę można pominąć, jeżeli zmodyfikujemy operację `blokuj_element( $X$ )` z rysunku 21.1 tak, aby w przypadku, gdy element jest w danej chwili zablokowany przez *transakcję żądającą*, blokada była przyznawana.

`blokuj_do_odczytu(X)`, `blokuj_do_zapisu(X)` oraz `odblokuj(X)` opisano na rysunku 21.2<sup>2</sup>. Tak jak poprzednio każda z trzech operacji powinna być traktowana jako niepodzielna — nie powinien być dozwolony żaden przeplot od momentu rozpoczęcia operacji aż do jej zakończenia poprzez przyznanie blokady lub umieszczenie transakcji w kolejce oczekującej.

```

blokuj_do_odczytu(X)
B: if BLOKADA(X) = "odblokowany"
    then begin BLOKADA(X)←"zablokowany_do_odczytu";
        liczba_odczytów(X)←1;
    end
    else if BLOKADA(X) = "zablokowany_do_odczytu"
        then liczba_odczytów(X)← liczba_odczytów(X)+1;
    else begin
        czekaj (dopóki BLOKADA(X)="odblokowany" i
            menedżer blokad uruchomi transakcję);
        go to B;
    end;
blokuj_do_zapisu(X)
B: if BLOKADA(X) = "odblokowany"
    then BLOKADA(X)←"zablokowany_do_zapisu";
    else begin
        czekaj (dopóki BLOKADA(X)="odblokowany" i
            menedżer blokad uruchomi transakcję);
        go to B;
    end;
odblokuj(X)
if BLOKADA(X)="zablokowany_do_zapisu"
    then begin
        BLOKADA(X)←"odblokowany";
        uruchom którąś z oczekujących transakcji, o ile takie istnieją;
    end
    else if BLOKADA(X)="zablokowany_do_odczytu"
        then begin
            liczba_odczytów(X)←liczba_odczytów(X)-1;
            if liczba_odczytów(X)=0
                then begin
                    BLOKADA(X)←"odblokowany";
                    uruchom którąś z oczekujących transakcji, o ile takie
                        istnieją;
                end;
        end;
end;

```

RYSUNEK 21.2. Operacje blokowania i odblokowania w przypadku blokad dwutrybowych (odczytu-zapisu lub dzielonych-wyłącznych)

<sup>2</sup> Przedstawione algorytmy nie pozwalają na *rozszerzanie* lub *zawężanie* blokad, co opisano w dalszej części podrozdziału. Czytelnik może je rozbudować w celu umożliwienia wykonywania tych operacji.

W przypadku użycia schematu blokowania dzielonego i wyłącznego, system musi wymuszać stosowanie się do następujących reguł:

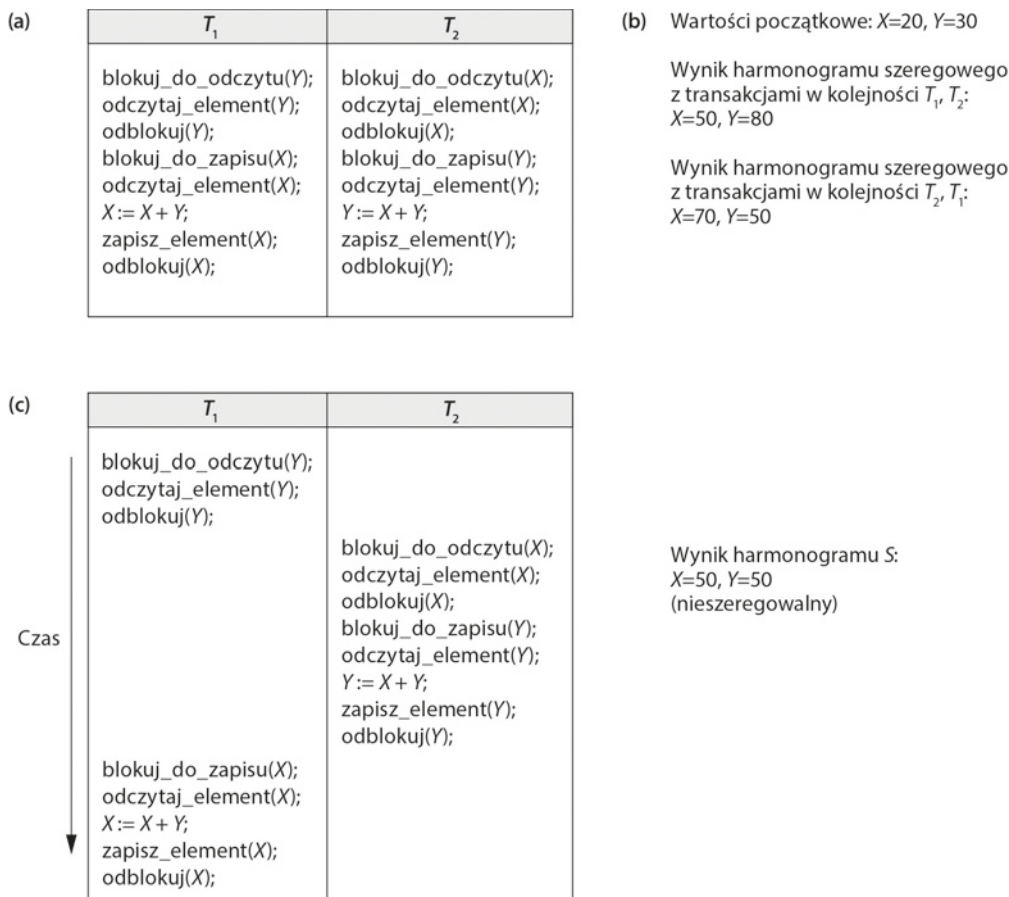
- (1) Transakcja  $T$  musi wykonać operację `blokuj_do_odczytu(X)` lub `blokuj_do_zapisu(X)` przed wykonaniem jakichkolwiek operacji `odczytaj_element(X)`.
- (2) Transakcja  $T$  musi wykonać operację `blokuj_do_zapisu(X)` przed wykonaniem jakichkolwiek operacji `zapisz_element(X)`.
- (3) Transakcja  $T$  musi wykonać operację `odblokuj(X)` po wykonaniu wszystkich operacji `blokuj_do_odczytu(X)` i `blokuj_do_zapisu(X)`<sup>3</sup>.
- (4) Transakcja  $T$  nie wykonuje operacji `blokuj_do_odczytu(X)`, jeżeli już trzyma blokadę dzieloną (do odczytu) lub blokadę wyłączną (do zapisu) na elemencie  $X$ . Regułę tę można złagodzić, co zostanie opisane poniżej.
- (5) Transakcja  $T$  nie wykonuje operacji `blokuj_do_zapisu(X)`, jeżeli już trzyma blokadę dzieloną (do odczytu) lub blokadę wyłączną (do zapisu) na elemencie  $X$ . Regułę tę można złagodzić, co zostanie opisane poniżej.
- (6) Transakcja  $T$  nie wykonuje operacji `odblokuj(X)`, chyba że trzyma blokadę dzieloną (do odczytu) lub blokadę wyłączną (do zapisu) na elemencie  $X$ .

**Konwersje blokad (rozszerzanie i zawężanie).** Niekiedy pożądanym jest złagodzenie warunków 4. i 5. w celu dopuszczenia **konwersji blokad** (ang. *lock conversion*). Oznacza to, że transakcja, która już trzyma blokadę na elemencie  $X$ , może w pewnych warunkach **przekonwertować** blokadę z jednego stanu do drugiego. Przykładowo, istnieje możliwość, aby transakcja  $T$  wykonała operację `blokuj_do_odczytu(X)`, a następnie dokonała **rozszerzenia** (ang. *update*) blokady poprzez wykonanie operacji `blokuj_do_zapisu(X)`. Jeżeli transakcja  $T$  jest jedyną trzymającą blokadę do odczytu na elemencie  $X$  w momencie wywołania operacji `blokuj_do_zapisu(X)`, blokada może zostać rozszerzona. W przeciwnym razie transakcja ta musi czekać. Istnieje również możliwość, aby transakcja  $T$  wykonała operację `blokuj_do_zapisu(X)`, a następnie **zawęziła** (ang. *downgrade*) blokadę poprzez wykonanie operacji `blokuj_do_odczytu(X)`. W przypadku, gdy wykorzystuje się rozszerzanie i zawężanie blokad, tabela blokad musi zawierać identyfikatory transakcji w strukturze rekordu dla każdej blokady (w ramach pola *transakcja(e)\_blokująca(e)*) w celu przechowywania informacji o tym, które transakcje trzymają blokady na danym elemencie. Aby umożliwić rozszerzanie i zawężanie blokad, opis operacji `blokuj_do_odczytu(X)` i `blokuj_do_zapisu(X)` z rysunku 21.2 należy odpowiednio zmienić. Pozostawiono to jako ćwiczenie dla Czytelnika.

Samo użycie w transakcjach blokad binarnych lub blokad odczytu-zapisu *nie gwarantuje szeregowałości* harmonogramów. Na rysunku 21.3 przedstawiono przykład, w którym wymienione wcześniej reguły blokowania są spełnione, ale można otrzymać harmonogram nieszegrowalny. Dzieje się tak dlatego, że na rysunku 21.3(a) elementy  $Y$  w transakcji  $T_1$  oraz  $X$  w transakcji  $T_2$  zostały *odblokowane zbyt wcześnie*. Pozwala to, aby wystąpił harmonogram podobny do przedstawionego na rysunku 21.3(c), który nie jest szeregowa-  
walny, a stąd daje błędne wyniki. W celu zagwarantowania szeregowałości należy stosować się do *dotodkowego protokołu* dotyczącego rozmieszczenia operacji blokowania i odblokowania w każdej transakcji. Najlepiej znanym z takich protokołów jest blokowanie dwufazowe, które zostanie opisane poniżej.

<sup>3</sup> Regułę tę można złagodzić w celu umożliwienia transakcji odblokowania elementu, a następnie ponownego jego zablokowania.





RYSUNEK 21.3. Transakcje niezgodne z regułami blokowania dwufazowego. (a) Dwie transakcje  $T_1$  i  $T_2$ . (b) Wyniki możliwych harmonogramów szeregowych transakcji  $T_1$  i  $T_2$ . (c) Nieszeregowalny harmonogram S wykorzystujący blokady

## 21.1.2. Gwarantowanie szeregowości blokowania dwufazowego

O transakcji mówimy, że jest zgodna z **protokołem blokowania dwufazowego** (ang. *two-phase locking protocol*), jeżeli *wszystkie* operacje blokowania (blokuj\_do\_odczytu, blokuj\_do\_zapisu) poprzedzają *pierwszą* operację odblokowania w transakcji<sup>4</sup>. Taką transakcję można podzielić na dwie fazy: **(pierwszą) fazę rozszerzania** (ang. *expanding phase*), w czasie której mogą być nakładane nowe blokady na elementy, ale żadne z nich nie mogą być zdejmowane, oraz **(drugą) fazę kurczenia** (ang. *shrinking phase*), w czasie której mogą być zwalniane istniejące blokady, ale nie mogą być nakładane nowe. Jeżeli dopuszczalna jest

<sup>4</sup> Nie ma to związku z protokołem zatwierdzania dwufazowego wykorzystywanego w celach odtwarzania w rozproszonych bazach danych (patrz rozdział 23.).

konwersja blokad, wówczas rozszerzanie blokad (z „do odczytu” na „do zapisu”) musi być wykonywane w czasie fazy rozszerzania, zaś zawężanie blokad (z „do zapisu” na „do odczytu”) musi być wykonywane w czasie fazy kurczenia.

Transakcje  $T_1$  i  $T_2$  z rysunku 21.3(a) nie są zgodne z protokołem blokowania dwufazowego. Jest tak dlatego, że operacja `blokuj_do_zapisu(X)` występuje po operacji `odblokuj(Y)` w transakcji  $T_1$  i, podobnie, operacja `blokuj_do_zapisu(Y)` występuje po operacji `odblokuj(X)` w transakcji  $T_2$ . Jeżeli wymusi się blokowanie dwufazowe, transakcje można przepisać jako  $T_1'$  i  $T_2'$ , co pokazano na rysunku 21.4. Tym razem harmonogram przedstawiony na rysunku 21.3(c) nie jest dozwolony dla transakcji  $T_1'$  i  $T_2'$  (z ich zmodyfikowaną kolejnością operacji blokowania i odblokowania) w przypadku reguł blokowania opisanych w podrozdziale 21.1.1. Dzieje się tak dlatego, że transakcja  $T_1'$  wykonuje swoją operację `blokuj_do_zapisu(X)` *przed* odblokowaniem elementu  $Y$ . W konsekwencji, kiedy transakcja  $T_2'$  wykonuje operację `blokuj_do_odczytu(X)`, jest zmuszona czekać do momentu, aż transakcja  $T_1'$  zwolni blokadę, wykonując operację `odblokuj(X)` w ramach harmonogramu. Może to jednak prowadzić do zakleszczenia (patrz punkt 21.1.3).

$T_1'$	$T_2'$
<code>blokuj_do_odczytu(Y);</code> <code>odczytaj_element(Y);</code> <code>blokuj_do_zapisu(X);</code> <code>odblokuj(Y)</code> <code>odczytaj_element(X);</code> $X := X + Y;$ <code>zapisz_element(X);</code> <code>odblokuj(X);</code>	<code>blokuj_do_odczytu(X);</code> <code>odczytaj_element(X);</code> <code>blokuj_do_zapisu(Y);</code> <code>odblokuj(X)</code> <code>odczytaj_element(Y);</code> $Y := X + Y;$ <code>zapisz_element(Y);</code> <code>odblokuj(Y);</code>

RYСУNEK 21.4. Transakcje  $T_1'$  i  $T_2'$ , takie same, jak transakcje  $T_1$  i  $T_2$  z rysunku 21.3, ale zgodne z protokołem blokowania dwufazowego. Warto zauważyć, że mogą one spowodować zakleszczenie

Można udowodnić, że jeżeli *każda* transakcja w harmonogramie jest zgodna z protokołem blokowania dwufazowego, harmonogram *na pewno jest szeregowalny*, co pozwala uniknąć konieczności sprawdzania szeregowalności harmonogramów. Mechanizm blokowania poprzez wymuszenie reguł blokowania dwufazowego wymusza również szeregowalność.

Blokowanie dwufazowe może ograniczyć poziom współbieżności związanej z harmonogramem. Wynika to z faktu, że transakcja  $T$  może nie być w stanie zwolnić elementu  $X$  po zakończeniu korzystania z niego, jeżeli musi później zablokować dodatkowy element  $Y$ , czyli kiedy transakcja  $T$  musi zablokować dodatkowy element  $Y$  zanim będzie go potrzebowała, tak aby mogła zwolnić element  $X$ . Stąd element  $X$  musi pozostać zablokowany przez transakcję  $T$  do momentu, aż wszystkie elementy wymagane przez transakcję dla operacji odczytu i zapisu zostaną zablokowane. Dopiero wtedy transakcja  $T$  będzie mogła zwolnić element  $X$ . Jednocześnie inna transakcja, próbująca uzyskać dostęp do elementu  $X$ , może być zmuszona do oczekiwania, choć transakcja  $T$  skończyła już działania na tym elemencie. Zatem jeśli element  $Y$  zostanie zablokowany wcześniej niż to konieczne, inna transakcja, która próbuje uzyskać do niego dostęp, jest zmuszona czekać, pomimo że transakcja  $T$  nie korzysta jeszcze z elementu  $Y$ . Jest to cena, jaką płaci się za zagwarantowanie szeregowalności wszystkich harmonogramów bez konieczności ich sprawdzania.

Choć protokół blokowania dwufazowego gwarantuje szeregowalność (w tym sensie, że każdy dozwolony harmonogram jest szeregowalny), nie *wszystkie możliwe* harmonogramy szeregowalne są w nim dozwolone (niektóre takie harmonogramy są w tym protokole zabronione).

**Blokowanie dwufazowe podstawowe, konserwatywne, ścisłe i rygorystyczne.** Istnieje wiele różnych odmian blokowania dwufazowego (ang. *Two-Phase Locking, 2PL*). Opisana powyżej technika jest znana jako **podstawowe blokowanie dwufazowe** (ang. *basic 2PL*). Odmiana znana jako **konserwatywne blokowanie dwufazowe** (ang. *conservative 2PL, static 2PL*) wymaga, aby transakcja zablokowała wszystkie elementy, do których uzyskuje dostęp, *przed rozpoczęciem swojego wykonywania* poprzez; potrzebne jest do tego **wcześniejsze zadeklarowanie zbioru odczytów** (ang. *read-set*) oraz **zbioru zapisów** (ang. *write-set*). W podrozdziale 21.1.2 stwierdzono, że **zbiór odczytów** transakcji jest zbiorem wszystkich elementów, które odczytuje, zaś **zbiór zapisów** transakcji jest zbiorem elementów, które zapisuje. Jeżeli któryś z wymaganych elementów nie może zostać zablokowany, transakcja nie blokuje żadnego elementu. Zamiast tego czeka, aż będzie możliwe zablokowanie wszystkich elementów. Konserwatywne blokowanie dwufazowe, co zostanie omówione w podrozdziale 21.1.3, jest *protokołem pozwalającym na uniknięcie problemu zakleszczenia*. Jednak jego stosowanie w praktyce jest trudne ze względu na konieczność wcześniejszego deklarowania zbioru odczytów i zapisów, co w niektórych sytuacjach nie jest możliwe.

W praktyce najpopularniejszą odmianą blokowania dwufazowego jest **ścisłe blokowanie dwufazowe** (ang. *strict 2PL*), które gwarantuje występowanie ścisłych harmonogramów (patrz podrozdział 21.4). W przypadku tej odmiany transakcja  $T$  nie zwalnia żadnej ze swoich wyłącznych (do zapisu) blokad *do momentu* jej zatwierdzenia lub anulowania. Stąd żadna inna transakcja nie może odczytywać lub zapisywać elementu zapisywanego przez  $T$ , chyba że transakcja ta została zatwierdzona, co prowadzi do powstania ścisłego harmonogramu odtwarzania. Ścisłe blokowanie dwufazowe nie zapobiega występowaniu zakleszczeń. Bardziej restrykcyjną odmianą ścisłego blokowania dwufazowego jest **rygorystyczne blokowanie dwufazowe** (ang. *rigorous 2PL*), które również gwarantuje występowanie harmonogramów ścisłych. W tym przypadku transakcja  $T$  nie zwalnia żadnych swoich blokad (wyłącznych lub dzielonych) do momentu, aż zostanie zatwierdzona lub anulowana, dzięki czemu jest łatwiejsza w implementacji od ścisłego blokowania dwufazowego.

Warto zwrócić uwagę na różnice między ścisłym a rygorystycznym blokowaniem dwufazowym. W tym pierwszym do czasu zatwierdzenia transakcji utrzymywane są blokady zapisu. W tym drugim utrzymywane są wszystkie blokady. Należy też zwrócić uwagę na różnice między konserwatywnym a rygorystycznym blokowaniem dwufazowym. To pierwsze musi zablokować wszystkie swoje elementy *zanim rozpocznie działania*, więc po rozpoczęciu transakcji znajduje się ona w fazie kurczenia. Z kolei w tym drugim przypadku żaden element nie jest odblokowywany do momentu, *aż zostaną zakończone działania* (poprzez zatwierdzenie lub anulowanie), więc transakcja aż do końca znajduje się w swojej fazie rozszerzania.

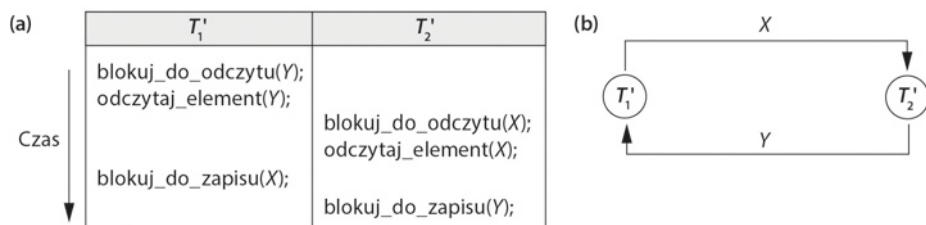
W wielu przypadkach sam **podsystem sterowania współbieżnego** jest odpowiedzialny za generowanie operacji `blokuj_do_odczytu` i `blokuj_do_zapisu`. Przykładowo, założmy, że system ma wymuszać protokół ścisłego blokowania dwufazowego. Wówczas, kiedy

transakcja  $T$  wykonuje operację `odczytaj_element(X)`, system wywołuje operację `blokuj_do_odczytu(X)` w imieniu transakcji  $T$ . Jeżeli stanem zmiennej `BLOKADA(X)` jest *zablokowany\_do\_zapisu*, nałożony przez pewną inną transakcję  $T'$ , to system umieszcza transakcję  $T$  w kolejce oczekującej na element  $X$ . W przeciwnym wypadku operacja `blokuj_do_odczytu(X)` może zostać wykonana, co pozwala na wykonanie operacji `odczytaj_element(X)` transakcji  $T$ . Z drugiej strony, jeżeli transakcja  $T$  wykonuje operację `zapisz_element(X)`, system wywołuje operację `blokuj_do_zapisu(X)` w imieniu transakcji  $T$ . Jeżeli stanem zmiennej `BLOKADA(X)` jest *zablokowany\_do\_zapisu* lub *zablokowany\_do\_odczytu* nałożony przez pewną inną transakcję  $T'$ , to system umieszcza transakcję  $T$  w kolejce oczekującej na element  $X$ . Jeżeli stanem zmiennej `BLOKADA(X)` jest *zablokowany\_do\_odczytu* i transakcja  $T$  jest jedyną transakcją trzymającą blokadę do odczytu na elemencie  $X$ , system rozszerza blokadę do postaci *zablokowany\_do\_zapisu* i pozwala na wykonanie operacji `zapisz_element(X)` przez transakcję  $T$ . Wreszcie, jeżeli stanem zmiennej `BLOKADA(X)` jest *odblokowany*, system akceptuje operację `blokuj_do_zapisu(X)` i pozwala na wykonanie operacji `zapisz_element(X)`. Po każdej operacji system musi odpowiednio *zaktualizować swoją tabelę blokad*.

Blokowanie powoduje wysokie koszty, ponieważ każda operacja odczytu lub zapisu jest poprzedzona kierowanym do systemu żądaniem przyznania blokady. Ponadto, użycie blokad może powodować dwa dodatkowe problemy: zakleszczenie i zagłodzenie. Zostaną one omówione poniżej.

### 21.1.3. Problem zakleszczenia i zagłodzenia

**Zakleszczenie** (ang. *deadlock*) występuje wówczas, gdy *każda* transakcja  $T$  ze zbioru *dwóch lub więcej transakcji* oczekuje na pewien element zablokowany przez inną transakcję  $T'$  z tego zbioru. Stąd każda transakcja ze zbioru znajduje się w kolejce oczekującej i czeka na inną transakcję ze zbioru na zwolnienie blokady. Jednak ponieważ ta inna transakcja także czeka, nigdy nie zwolni blokady. Adekwatny prosty przykład przedstawiono na rysunku 21.5(a), gdzie dwie transakcje  $T_1'$  i  $T_2'$  znajdują się w stanie zakleszczenia w ramach harmonogramu częściowego. Transakcja  $T_1'$  znajduje się w kolejce oczekującej na element  $X$  zablokowany przez transakcję  $T_2'$ , natomiast transakcja  $T_2'$  znajduje się w kolejce oczekującej na element  $Y$  zablokowany przez transakcję  $T_1'$ . W takim przypadku ani transakcja  $T_1'$ , ani  $T_2'$ , ani żadna inna nie może uzyskać dostępu do elementów  $X$  i  $Y$ .



RYСУNEK 21.5. Ilustracja problemu zakleszczenia. (a) Harmonogram częściowy transakcji  $T_1'$  i  $T_2'$ , która znajduje się w stanie zakleszczenia. (b) Graf oczekiwania dla harmonogramu częściowego z (a)

**Protokoły zapobiegania zakleszczeniom.** Jednym ze sposobów unikania zakleszczeń jest użycie **protokołu zapobiegania zakleszczeniom** (ang. *deadlock prevention protocol*)<sup>5</sup>. Jeden z takich protokołów, używany w przypadku konserwatywnego blokowania dwufazowego, wymaga, aby każda transakcja *z góry blokowała wszystkie wymagane elementy* (co zwykle nie jest zbyt realnym założeniem) — jeżeli któryś z elementów nie może zostać zarezerwowany, żaden nie podlega blokadzie. Zamiast tego transakcja odczekuje pewien czas, a następnie ponownie podejmuje próbę zablokowania wszystkich wymaganych elementów. Takie rozwiązanie, oczywiście, w jeszcze większym stopniu ogranicza współbieżność. Drugi protokół, również ograniczający współbieżność, jest związany z *uporządkowaniem wszystkich elementów* w bazie danych i zapewnieniem, aby transakcja wymagająca kilku elementów blokowała je w odpowiedniej kolejności. Wymaga to, aby programista (lub system) pamiętał o wybranym porządku elementów, co również nie jest praktycznym rozwiązaniem w kontekście baz danych.

Zaproponowano wiele schematów zapobiegania zakleszczeniom, które służą podejmowaniu decyzji odnośnie do tego, co należy zrobić z transakcją związaną z prawdopodobnym zakleszczeniem: czy powinna zostać zablokowana i ma czekać, czy może powinna zostać anulowana lub wywłaszczona, a anulowana druga transakcja? Techniki te wykorzystują pojęcie **znacznika czasu transakcji** (ang. *transaction timestamp*)  $TS(T)$ , który jest unikatowym identyfikatorem przypisanym do każdej transakcji. Znaczniki czasu bazują zwykle na kolejności, w jakiej uruchomiono transakcje. Stąd, jeżeli transakcja  $T_1$  rozpoczyna działanie przed transakcją  $T_2$ , to  $TS(T_1) < TS(T_2)$ . Należy zauważyć, że starsza transakcja posiada *mniej* wartość znacznika czasu. Dwa schematy zapobiegające zakleszczeniom noszą nazwy *czekaj-kończ* oraz *zakończ-czekaj*. Załóżmy, że transakcja  $T_i$  próbuje zablokować element  $X$ , ale nie jest w stanie tego zrobić, ponieważ element  $X$  jest zablokowany przez inną transakcję  $T_j$  z blokadą konfliktową. Reguły stosowane przez te schematy są następujące:

- **Czekaj-kończ** (ang. *wait-die*): jeżeli  $TS(T_i) < TS(T_j)$ , to (transakcja  $T_i$  jest starsza od  $T_j$ )  $T_i$  może czekać, w przeciwnym wypadku (transakcja  $T_i$  jest młodsza od  $T_j$ ) anulujemy transakcję  $T_i$  ( $T_i$  kończy swoje działanie) i ponawiamy ją później z *tym samym znacznikiem czasu*.
- **Zakończ-czekaj** (ang. *wound-wait*): jeżeli  $TS(T_i) < TS(T_j)$ , to (transakcja  $T_i$  jest starsza od  $T_j$ ) anulujemy transakcję  $T_j$  (transakcja  $T_i$  kończy transakcję  $T_j$ ), a później ponawiamy ją z *tym samym znacznikiem czasu*. W przeciwnym wypadku (transakcja  $T_i$  jest młodsza od  $T_j$ ) transakcja  $T_i$  może czekać.

W przypadku schematu *czekaj-kończ* starsza transakcja może *czekać na młodszą transakcję*, natomiast żądanie młodszej transakcji o element trzymany przez starszą transakcję powoduje anulowanie tej pierwszej i jej ponowienie. W przypadku schematu *zakończ-czekaj* jest odwrotnie: młodsza transakcja *może czekać na starszą*, natomiast żądanie starszej transakcji o element trzymany przez młodszą transakcję powoduje *wywłaszczenie* (ang. *preempt*) tej ostatniej i jej anulowanie. Oba schematy sprowadzają się do anulowania *młodszej* (później rozpoczętej) z dwóch transakcji, które *mogą być związane*

<sup>5</sup> Protokoły te generalnie nie są używane w praktyce ze względu na nierealistyczne założenia lub możliwe koszty. Bardziej praktyczne jest stosowanie (omówionych w dalszych punktach) technik wykrywania zakleszczeń i limitów czasowych.

z zakleszczeniem. Założenie jest takie, że to rozwiązanie powoduje mniejsze marnotrawstwo zasobów. Można wykazać, że te dwie techniki *pozwalają na uniknięcie zakleszczenia*, gdyż w przypadku schematu czekaj-kończ transakcje oczekują tylko na młodsze transakcje, więc nie są tworzone żadne cykle, natomiast w przypadku schematu zakończ-czekaj transakcje oczekują wyłącznie na starsze transakcje i także nie są tworzone żadne cykle. Jednakże obie te techniki mogą powodować niepotrzebne anulowanie i ponawianie transakcji, mimo że w rzeczywistości mogą one *wcale nie powodować zakleszczenia*.

Kolejna grupa protokołów zapobiegających zakleszczeniom nie wymaga użycia znaczników czasu. Należą do niej algorytmy **nieoczekiwania** (ang. *no waiting, NW*) oraz **oczekiwania ostrożnego** (ang. *cautious waiting, CW*). W przypadku **algorytmu nieoczekiwania**, jeżeli transakcja nie może założyć blokady, jest natychmiast anulowana, a później ponawiana po upływie określonego czasu bez sprawdzania, czy zakleszczenie rzeczywiście wystąpiłoby, czy nie. W tym scenariuszu żadna transakcja nie oczekuje, dlatego zakleszczenie nie występuje. Ze względu na fakt, że schemat ten może powodować niepotrzebne anulowanie i ponawianie transakcji, zaproponowano algorytm **oczekiwania ostrożnego** w celu zredukowania liczby niepotrzebnych operacji anulowania i ponawiania. Załóżmy, że transakcja  $T_i$  próbuje zablokować element  $X$ , ale nie jest w stanie tego zrobić ze względu na fakt, że element  $X$  został zablokowany przez pewną inną transakcję  $T_j$  z blokadą konfliktową. Reguły oczekiwania ostrożnego są następujące:

- **Oczekiwanie ostrożne** (ang. *cautious waiting*): jeżeli transakcja  $T_j$  nie jest blokowana (nie czeka na pewien inny zablokowany element), to blokowana jest transakcja  $T_i$  i może ona czekać. W przeciwnym wypadku transakcja  $T_i$  jest anulowana.

Można wykazać, że oczekiwanie ostrożne zapobiega powstawaniu zakleszczeń, ponieważ żadna transakcja nie oczekuje na inną zablokowaną transakcję. Biorąc pod uwagę czas  $b(T)$ , w którym każda blokowana transakcja  $T$  została zablokowana, jeżeli dwie transakcje —  $T_i$  i  $T_j$  — zostaną zablokowane i transakcja  $T_i$  czeka na  $T_j$ , to  $b(T_i) < b(T_j)$ , gdyż transakcja  $T_i$  może czekać na  $T_j$  tylko wówczas, gdy  $T_j$  nie jest blokowana. Stąd czasy blokowania tworzą uporządkowanie zupełne na wszystkich blokowanych transakcjach, więc nie mogą występować żadne cykle powodujące zakleszczenie.

**Wykrywanie zakleszczeń.** Innym podejściem do kwestii zakleszczeń jest **wykrywanie zakleszczeń** (ang. *deadlock detection*), gdzie system sprawdza, czy stan zakleszczenia rzeczywiście występuje. Rozwiązanie to jest atrakcyjne, jeżeli wiadomo, że między transakcjami rzadko występują kolizje, to znaczy, jeżeli różne transakcje rzadko uzyskują dostęp do tych samych elementów w tym samym czasie. Może tak być w przypadku, gdy transakcje są krótkie i każda z nich blokuje tylko kilka elementów lub kiedy obciążenie transakcyjne jest niewielkie. Z drugiej strony, jeżeli transakcje są długie i wykorzystują wiele elementów lub jeśli obciążenie transakcyjne jest duże, lepszym rozwiązaniem może okazać się użycie schematu zapobiegania zakleszczeniom.

Prostym sposobem wykrywania stanu zakleszczenia przez system jest skonstruowanie i przechowywanie **grafu oczekiwania** (ang. *wait-for graph*). Dla każdej wykonywanej transakcji w grafie oczekiwania jest tworzony wierzchołek. Kiedy transakcja  $T_i$  czeka na



zablokowanie elementu  $X$ , który został zablokowany przez transakcję  $T_j$ , w grafie jest tworzona krawędź skierowana ( $T_i \rightarrow T_j$ ). Kiedy transakcja  $T_j$  zwalnia blokadę(y) na elementach, na które czeka transakcja  $T_i$ , krawędź jest z grafu usuwana. Stan zakleszczenia występuje wtedy i tylko wtedy, gdy graf oczekiwania zawiera cykl. Jednym z problemów związanych z takim podejściem jest kwestia określenia tego, *kiedy* system powinien sprawdzać występowanie zakleszczeń. Jedną z możliwości to sprawdzanie wystąpienia cyklu każdorazowo po dodaniu krawędzi do grafu oczekiwania. Może to jednak powodować nadmierne koszty. Można stosować takie kryteria jak liczba współbieżnie wykonywanych transakcji lub okres, przez jaki kilka transakcji czeka na zablokowanie elementów. Na rysunku 21.5(b) pokazany jest graf oczekiwania dla (częściowego) harmonogramu z rysunku 21.5(a).

Jeżeli system znajduje się w stanie zakleszczenia, pewne związane z nim transakcje muszą zostać anulowane. Wybór tego, które transakcje należy anulować, określa się mianem **wyboru ofiary** (ang. *victim selection*). Ogólnie rzecz biorąc, algorytm taki powinien unikać wyboru transakcji, które działały przez dłuższy czas i które wykonały wiele operacji aktualizowania, i starać się wybierać takie, które nie dokonały wielu zmian (czyli młodsze transakcje).

**Limity czasu.** Kolejnym prostym schematem radzenia sobie z zakleszczeniami jest użycie **limitów czasu** (ang. *timeouts*). Metoda ta sprawdza się w praktyce ze względu na wiążący się z nią niski narzut oraz prostotę. W jej przypadku, jeżeli transakcja oczekuje przez okres czasu dłuższy od zdefiniowanego dla systemu limitu, system przyjmuje, że transakcja uległa zakleszczeniu i anuluje ją — bez względu na to, czy zakleszczenie rzeczywiście wystąpiło, czy nie.

**Zagłodzenie.** Kolejnym problemem, który może wystąpić w przypadku użycia blokowania jest **zagłodzenie** (ang. *starvation*), o którym mówimy wówczas, gdy transakcja nie może kontynuować działania przez nieokreślony czas, podczas gdy inne transakcje w systemie są wykonywane normalnie. Może się tak zdarzyć w przypadku, gdy schemat oczekiwania na zablokowane elementy nie jest bezstronny i daje pewnym transakcjom pierwszeństwo w porównaniu z innymi. Jednym z rozwiązań problemu zagłodzenia jest użycie bezstronnego schematu oczekiwania, takiego jak kolejka **pierwszy zgłoszony — pierwszy obsłużony** (ang. *first-come-first-served*). Transakcje mogą blokować element w kolejności, w jakiej oryginalnie zażądały blokady. Kolejny schemat pozwala, aby pewne transakcje miały wyższy priorytet od innych, ale zwiększa priorytet każdej transakcji w miarę, jak oczekuje na swoją kolej, do momentu, aż osiągnie najwyższy priorytet i zostanie wykonana. Zagłodzenie może również występować ze względu na wybór ofiary, jeżeli algorytm wybiera jako ofiarę ciągle tę samą transakcję, co powoduje jej anulowanie i uniemożliwia ukończenie. W celu uniknięcia tego problemu algorytm może wykorzystywać wyższe priorytety dla transakcji, które zostały wielokrotnie anulowane. Opisane wcześniej schematy czekaj-koncz i zakończ-czekaj nie powodują zagłodzenia, ponieważ gdy ponownie uruchamiają anulowaną transakcję, przypisywany jest jej pierwotny znacznik czasu. Z tego powodu prawdopodobieństwo anulowania tej samej transakcji jest niskie.



## 21.2. Sterowanie współbieżne w oparciu o uporządkowanie według znaczników czasu

Użycie blokad w połączeniu z protokołem blokowania dwufazowego gwarantuje szeregowalność harmonogramów. Harmonogramy szeregowalne utworzone przez blokowanie dwufazowe posiadają swoje równoważne harmonogramy szeregowo oparte na porządku, w jakim wykonywane transakcje blokują elementy. Jeżeli transakcja wymaga elementu, który już został zablokowany, może zostać zmuszona do oczekiwania aż element ten zostanie zwolniony. Niektóre transakcje mogą zostać anulowane lub ponownie uruchomione z powodu zakleszczenia. Inne podejście, również gwarantujące szeregowalność, polega na użyciu znaczników czasu transakcji w celu określenia kolejności ich wykonywania dla równoważnego harmonogramu szeregowego. W podrozdziale 21.2.1 zostaną omówione znaczniki czasu, zaś w podrozdziale 21.2.2 przedstawimy, w jaki sposób jest wymuszana szeregowalność poprzez porządkowanie transakcji w oparciu o ich znaczniki.

### 21.2.1. Znaczniki czasu

Jak już wspomniano, **znacznik czasu** (ang. *timestamp*) jest unikatowym identyfikatorem stworzonym przez SZBD w celu identyfikowania transakcji. Zazwyczaj wartości znaczników czasu są przypisywane w kolejności, w jakiej transakcje są przesyłane do systemu, więc znacznik czasu można postrzegać jako *czas rozpoczęcia transakcji*. Znacznik czasu transakcji  $T$  będziemy określać jako **TS( $T$ )**. Techniki sterowania współbieżnego oparte na porządkowaniu znaczników czasu nie wykorzystują blokad, stąd *nie mogą tu występować zakleszczenia*.

Znaczniki czasu mogą być generowane na kilka sposobów. Jedną z możliwości jest użycie licznika, który jest zwiększany za każdym razem, gdy jego wartość jest przypisywana do transakcji. W przypadku takiego schematu znaczniki czasu transakcji są numerowane jako 1, 2, 3, .... Licznik komputerowy posiada ograniczoną wartość maksymalną, więc system musi okresowo zerować licznik, kiedy przez pewien krótki okres czasu nie są wykonywane żadne transakcje. Innym sposobem implementacji znaczników czasu jest użycie bieżącej wartości daty i czasu zegara systemowego i zapewnienie, aby żadne dwa znaczniki nie zostały wygenerowane z taką samą wartością.

### 21.2.2. Algorytm uporządkowania według znaczników czasu

Idea działania tego schematu opiera się na równoważnym uporządkowaniu szeregowym transakcji na podstawie ich znaczników czasu. Harmonogram obejmujący takie transakcje jest szeregowalny, a *jedyny dozwolony równoważny mu harmonogram szeregowy* zawiera transakcje w kolejności zgodnej ze znacznikami czasu. Jest to tak zwane **porządkowanie według znaczników czasu** (ang. *timestamp ordering, TO*). Należy zwrócić uwagę na różnice w porównaniu z blokowaniem dwufazowym, gdzie harmonogram był szeregowalny dzięki temu, że był równoważny pewnemu harmonogramowi szeregowemu, dopuszczalnemu w ramach protokołu blokowania. Jednak w przypadku porządkowania według znaczników czasu harmonogram jest równoważny *określonemu uporządkowaniu szeregowemu* odpowiadającemu porządkowi znaczników czasu transakcji. Algorytm dopuszcza przeplot

operacji z transakcji, ale musi zapewniać, aby dla każdego elementu, do którego uzyskują dostęp *operacje konfliktowe* w harmonogramie, kolejność, w jakiej ten dostęp jest uzyskiwany, nie naruszała kolejności znaczników czasu. W tym celu algorytm kojarzy z każdym elementem  $X$  bazy danych dwa znaczniki czasu TS:

- (1)  $TS\_odczytu(X)$ . **Znacznik czasu odczytu** (ang. *read timestamp*) elementu  $X$  to znacznik czasu o najwyższej wartości spośród wszystkich znaczników transakcji, które z powodzeniem odczytały element  $X$ , to znaczy  $TS\_odczytu(X) = TS(T)$ , gdzie  $T$  jest *najmłodszą* transakcją, która z powodzeniem odczytała element  $X$ .
- (2)  $TS\_zapisu(X)$ . **Znacznik czasu zapisu** (ang. *write timestamp*) elementu  $X$  to znacznik czasu o najwyższej wartości spośród wszystkich znaczników transakcji, które z powodzeniem zapisały element  $X$ , to znaczy  $TS\_zapisu(X) = TS(T)$ , gdzie  $T$  jest *najmłodszą* transakcją, która z powodzeniem zapisała element  $X$ . Dalej zobaczysz, że w tym algorytmie  $T$  jest ostatnią transakcją zapisującą element  $X$ .

**Podstawowe porządkowanie według znaczników czasu.** Kiedy pewna transakcja  $T$  podejmuje próbę wykonania operacji  $odczytaj\_element(X)$  lub  $zapisz\_element(X)$ , algorytm **podstawowego** porządkowania według znaczników czasu porównuje znacznik transakcji  $T$  ze znacznikami  $TS\_odczytu(X)$  oraz  $TS\_zapisu(X)$  w celu zapewnienia, że kolejność znaczników czasu wykonania transakcji nie jest naruszona. Jeśli jednak nastąpi naruszenie, transakcja  $T$  jest anulowana i ponawiana jako nowa transakcja z *nowym znacznikiem czasu*. Jeżeli transakcja  $T$  zostanie anulowana i wycofana, każda transakcja  $T_1$ , która mogła korzystać z wartości zapisanej przez  $T$ , również musi zostać wycofana. Podobnie, każda transakcja  $T_2$ , która mogła korzystać z wartości zapisanej przez  $T_1$ , musi zostać wycofana itd. Taki efekt określa się mianem **wycofania kaskadowego** i jest ono jednym z problemów związanych z podstawowym porządkowaniem według znaczników czasu, gdyż utworzone harmonogramy nie zawsze muszą być odtwarzalne. Należy zastosować *dotodatkowy protokół* w celu zapewnienia, że harmonogramy będą odtwarzalne, bezkaskadowe lub ścisłe. Najpierw opiszemy algorytm podstawowego porządkowania według znaczników czasu. Algorytm sterowania współbieżnego musi sprawdzać, czy operacje konfliktowe naruszają porządek znaczników czasu w następujących dwóch przypadkach:

- (1) Gdy transakcja  $T$  wykonuje operację  $zapisz\_element(X)$ , wykonywane są następujące testy:
  - a. Jeżeli  $TS\_odczytu(X) > TS(T)$  lub  $TS\_zapisu(X) > TS(T)$ , anulujemy i wycofujemy transakcję  $T$  oraz odrzucamy operację. Należy tak postąpić dlatego, że pewna *młodsza* transakcja o znaczniku czasu większym niż  $TS(T)$  — a więc występująca *po* transakcji  $T$  w uporządkowaniu według znaczników — odczytała już lub zapisała wartość elementu  $X$ , zanim transakcja  $T$  miała szansę zapisać  $X$ , co narusza uporządkowanie według znaczników czasu.
  - b. Jeżeli warunek z punktu a. nie występuje, wykonujemy operację  $zapisz\_element(X)$  transakcji  $T$  i ustawiamy wartość  $TS\_zapisu(X)$  na  $TS(T)$ .
- (2) Gdy transakcja  $T$  wykonuje operację  $odczytaj\_element(X)$ , przeprowadzane są następujące testy:
  - a. Jeżeli  $TS\_zapisu(X) > TS(T)$ , anulujemy i wycofujemy transakcję  $T$  i odrzucamy operację. Należy tak postąpić dlatego, że pewna *młodsza* transakcja o znaczniku czasu większym niż  $TS(T)$  — a więc występująca *po* transakcji  $T$  w uporządkowaniu według znaczników — zapisała już wartość elementu  $X$ , zanim transakcja  $T$  miała szansę odczytać  $X$ .

- b. Jeżeli  $TS\_zapisu(X) \leq TS(T)$ , wykonujemy operację  $odczytaj\_element(X)$  transakcji  $T$  i ustawiamy wartość  $TS\_odczytu(X)$  na *większą* spośród wartości  $TS(T)$  lub bieżącej wartości  $TS\_odczytu(X)$ .

Zatem kiedy algorytm podstawowego porządkowania według znaczników czasu wykryje dwie *operacje konfliktowe* występujące w niepoprawnej kolejności, odrzuca późniejszą z tych operacji, anulując transakcję, która ją wykonała. Harmonogramy tworzone przez algorytm podstawowego porządkowania według znaczników czasu są więc zawsze *szeregowalne konfliktowo*. Jak wcześniej wspomniano, zakleszczenie nie występuje w przypadku porządkowania według znaczników czasu, jednak może wystąpić cykliczny restart (a stąd zagłodzenie) w przypadku, gdy transakcja jest stale anulowana i ponawiana.

**Ścisłe porządkowanie według znaczników czasu.** Odmianą metody podstawowej jest **ścisłe** (ang. *strict*) porządkowanie według znaczników czasu, które zapewnia, że harmonogramy są zarówno **ścisłe** (łatwe do odtworzenia), jak i (konfliktowo) szeregowalne. W przypadku tej odmiany transakcja  $T$ , taka że  $TS(T) > TS\_zapisu(X)$ , która wykonuje operację  $odczytaj\_element(X)$  lub  $zapisz\_element(X)$ , *opóźnia* ją do momentu, aż transakcja  $T'$ , która *zapisła* wartość  $X$  (a stąd  $TS(T') = TS\_zapisu(X)$ ), zostanie zatwierdzona lub anulowana. W celu zaimplementowania takiego algorytmu konieczne jest zasymulowanie blokady elementu  $X$ , który został zapisany przez transakcję  $T'$ , do momentu, aż transakcja ta zostanie zatwierdzona lub anulowana. Algorytm ten *nie powoduje zakleszczeń*, gdyż transakcja  $T$  czeka na transakcję  $T'$  tylko wówczas, gdy  $TS(T) > TS(T')$ .

**Reguła zapisu Thomasa.** Odmiana algorytmu podstawowego porządkowania według znaczników czasu, znana jako **reguła zapisu Thomasa** (ang. *Thomas's write rule*), nie wymaga szeregowalności konfliktowej, ale odrzuca mniejszą liczbę operacji zapisu, modyfikując sprawdzenia operacji  $zapisz\_element(X)$  następująco:

- (1) Jeżeli  $TS\_odczytu(X) > TS(T)$ , to anulujemy i wycofujemy transakcję  $T$  oraz odrzucamy operację.
- (2) Jeżeli  $TS\_zapisu(X) > TS(T)$ , nie wykonujemy operacji zapisu, ale kontynuujemy przetwarzanie. Postępujemy tak dlatego, że pewna transakcja o wyższej wartości znacznika niż  $TS(T)$ , a stąd występująca w uporządkowaniu po transakcji  $T$ , *zapisła* już wartość  $X$ . Dlatego też musimy zignorować operację  $zapisz\_element(X)$  transakcji  $T$ , ponieważ jest ona nieaktualna. Należy zauważyć, że wszelkie konflikty wynikające z takiej sytuacji zostałyby wykryte w punkcie 1.
- (3) Jeżeli nie występuje ani sytuacja z punktu 1., ani z punktu 2., wykonujemy operację  $zapisz\_element(X)$  transakcji  $T$  i ustawiamy  $TS\_zapisu(X)$  na  $TS(T)$ .

## 21.3. Techniki wielowersyjnego sterowania współbieżnego

Inne protokoły sterowania współbieżnego przechowują stare wartości elementu danych, kiedy jest on aktualizowany. Są to tak zwane techniki **wielowersyjnego sterowania współbieżnego** (ang. *multiversion concurrency control*), ponieważ przechowywanych jest kilka wersji (wartości) elementów. Kiedy transakcja wymaga dostępu do elementu, wybierana

jest *odpowiednia* wersja w celu zachowania szeregowalności bieżąco wykonywanego harmonogramu (o ile to możliwe). Jednym z powodów przechowywania kilku wersji jest to, że pewne operacje odczytu, które zostałyby odrzucone w przypadku innych technik, tu mogą być akceptowane poprzez odczytanie *starszej wartości* elementu w celu zachowania szeregowalności. Kiedy transakcja zapisuje element, zapisuje *nową wersję*, a stara wersja jest zachowywana. Niektóre algorytmy wielowersyjnego sterowania współbieżnego wykorzystują pojęcie szeregowalności perspektywicznej zamiast szeregowalności konfliktowej.

Oczywistą wadą technik wielowersyjnych są zwiększone wymagania dotyczące przestrzeni przechowywania wielu wersji elementów bazy danych. W niektórych sytuacjach starsze wersje i tak muszą być przechowywane, na przykład w celach odtwarzania. Niektóre aplikacje bazodanowe wymagają przechowywania starszych wersji w celu tworzenia historii zmian wartości elementów danych. Przypadkiem krańcowym jest *czasowa baza danych* (ang. *temporal database*) — patrz podrozdział 26.2 — która śledzi wszystkie zmiany oraz momenty, w których te zmiany zachodzą. W takich przypadkach nie ma problemu związanego z dodatkowym narzutem co do przestrzeni składowania danych w zakresie technik wielowersyjnych, gdyż starsze wersje i tak muszą być przechowywane.

Zaproponowano kilka schematów wielowersyjnego sterowania współbieżnego. Poniżej zostaną omówione dwa z nich — jeden bazujący na porządkowaniu znaczników czasu, drugi — na blokowaniu dwufazowym. Ponadto metoda sterowania współbieżnego oparta na walidacji (patrz podrozdział 21.4) też wymaga przechowywania wielu wersji, a stosowaną w kilku systemach komercyjnych technikę *izolacji snapshotów* (patrz podrozdział 21.4) można zaimplementować, przechowując starsze wersje elementów w pamięci tymczasowej.

### 21.3.1. Technika wielowersyjna oparta na porządkowaniu według znaczników czasu

W przypadku tej metody przechowuje się kilka wersji  $X_1, X_2, \dots, X_k$  każdego elementu danych  $X$ . Dla *każdej wersji* przechowywana jest wartość tej wersji  $X_i$  oraz następujące dwa znaczniki czasu:

- (1)  $TS\_odczytu(X_i)$ . **Znacznik czasu odczytu** elementu  $X_i$  jest największą wartością spośród wszystkich znaczników transakcji, które z powodzeniem odczytały daną wersję elementu  $X_i$ .
- (2)  $TS\_zapisu(X_i)$ . **Znacznik czasu zapisu** elementu  $X_i$  jest znacznikiem czasu transakcji, która zapisała wartość danej wersji elementu  $X_i$ .

Kiedy transakcja  $T$  może wykonać operację `zapisz_element( $X$ )`, tworzona jest nowa wersja  $X_{k+1}$  elementu  $X$  i zarówno  $TS\_zapisu(X_{k+1})$ , jak i  $TS\_odczytu(X_{k+1})$  ustawia się na wartość  $TS(T)$ . Z kolei kiedy transakcja  $T$  może odczytać wartość wersji  $X_i$ , wartość  $TS\_odczytu(X_i)$  jest ustawiana na większą spośród wartości  $TS\_odczytu(X_i)$  lub  $TS(T)$ .

W celu zapewnienia szeregowalności stosuje się następujące reguły:

- (1) Jeżeli transakcja  $T$  wykonuje operację `zapisz_element( $X$ )` i wersja  $i$  elementu  $X$  posiada najwyższą wartość  $TS\_zapisu(X_i)$  spośród wszystkich wersji elementu  $X$  oraz jest *mniej* lub *równa*  $TS(T)$ , a ponadto  $TS\_odczytu(X_i) > TS(T)$ , wówczas

anulujemy i wycofujemy transakcję  $T$ . W przeciwnym razie tworzymy nową wersję  $X_i$  elementu  $X$  z wartościami  $TS\_odczytu(X_i) = TS\_zapisu(X_i) = TS(T)$ .

- (2) Jeżeli transakcja  $T$  wykonuje operację  $odczytaj\_element(X)$ , znajdujemy wersję  $i$  elementu  $X$ , która posiada najwyższą wartość  $TS\_zapisu(X_i)$  spośród wszystkich wersji elementu  $X$  oraz *jest mniejsza lub równa*  $TS(T)$ . Następnie zwracamy wartość  $X_i$  do transakcji  $T$  i ustawiamy wartość  $TS\_odczytu(X_i)$  na większą spośród wartości  $TS(T)$  lub bieżącej  $TS\_odczytu(X_i)$ .

Przypadek 2. pokazuje, że operacja  $odczytaj\_element(X)$  *zawsze kończy się powodzeniem*, gdyż znajduje odpowiednią wersję elementu  $X_i$  w oparciu o wartość  $TS\_zapisu$  różnych istniejących wersji elementu  $X$ . Jednakże w przypadku 1. transakcja  $T$  może zostać anulowana i wycofana. Dzieje się tak w sytuacji, gdy transakcja  $T$  próbuje zapisać wersję elementu  $X$ , która powinna być zostać odczytana przez inną transakcję  $T'$ , której znacznikiem czasu jest  $TS\_odczytu(X_i)$ . Transakcja  $T'$  odczytała już jednak wersję  $X_i$ , która została zapisana przez transakcję ze znacznikiem czasu równym  $TS\_zapisu(X_i)$ . W razie wystąpienia takiego konfliktu transakcja  $T$  jest wycofywana, a w przeciwnym razie jest tworzona nowa wersja elementu  $X$ , zapisywana przez transakcję  $T$ . Należy zauważyć, że jeżeli transakcja  $T$  zostanie wycofana, może wystąpić wycofywanie kaskadowe. Stąd, w celu zapewnienia odtwarzalności, transakcja  $T$  nie powinna mieć możliwości zatwierdzenia, dopóki wszystkie transakcje, które zapisały pewną wersję odczytaną przez  $T$ , nie zostaną zatwierdzone.

### 21.3.2. Wielowersyjne blokowanie dwufazowe z użyciem blokad certyfikujących

W przypadku takiego wielotrybowego schematu blokowania można wyróżnić *trzy tryby blokowania* elementu: do odczytu, do zapisu oraz do *certyfikacji*, zamiast tylko dwóch (do odczytu i do zapisu), jak miało to miejsce wcześniej. Stąd stan  $BLOKADA(X)$  elementu  $X$  może mieć wartość  $zablokowany\_do\_odczytu$ ,  $zablokowany\_do\_zapisu$ ,  $zablokowany\_do\_certyfikacji$  oraz  $odblokowany$ . W standardowym schemacie blokowania, uwzględniającym tylko blokady do odczytu i zapisu (patrz podrozdział 21.1.1), blokada do zapisu jest blokadą wyłączną. Związki między blokadami do odczytu i zapisu w schemacie standardowym można opisywać za pomocą **tabeli zgodności blokad** (ang. *lock compatibility table*), przedstawionej na rysunku 21.6(a). Wpis o wartości *tak* oznacza, że jeżeli transakcja  $T$  trzyma na elemencie  $X$  blokadę o typie określonym w nagłówku kolumny oraz jeżeli transakcja  $T'$  zażąda na tym samym elemencie blokady o typie określonym w nagłówku wiersza, to transakcja  $T'$  może *uzyskać blokadę*, ponieważ tryby blokowania są zgodne. Z drugiej strony wpis o wartości *nie* oznacza, że blokady nie są zgodne, więc transakcja  $T'$  *musi poczekać na zwolnienie* blokady przez transakcję  $T$ .

W przypadku standardowego schematu blokowania kiedy transakcja uzyskuje blokadę do zapisu na elemencie, żadna inna transakcja nie może uzyskać dostępu do tego elementu. Idea działania schematu wielowersyjnego blokowania dwufazowego polega na pozwoleniu, aby inne transakcje  $T'$  odczytywały element  $X$  w czasie, gdy pojedyncza transakcja  $T$  posiada blokadę do zapisu na tym elemencie. Osiąga się to, dopuszczając występowanie

(a)	Odczyt	Zapis
Odczyt	Tak	Nie
Zapis	Nie	Nie

(b)	Odczyt	Zapis	Certyfikacja
Odczyt	Tak	Tak	Nie
Zapis	Tak	Nie	Nie
Certyfikacja	Nie	Nie	Nie

RYСУNEK 21.6. Tabele zgodności blokad. (a) Tabela zgodności blokad dla schematu blokowania odczytu-zapisu. (b) Tabela zgodności blokad dla schematu blokowania odczytu-zapisu-certyfikacji

dwóch wersji każdego elementu  $X$ . Jedna z wersji (**zatwierdzona**) musi zawsze być taką, która została zapisana przez zatwierdzoną transakcję. Druga wersja  $X'$  (**lokalna**) jest tworzona w sytuacji, gdy transakcja  $T$  uzyskuje blokadę do zapisu na danym elemencie. Inne transakcje mogą wciąż odczytywać *zatwierdzoną wersję* elementu  $X$ , kiedy transakcja  $T$  posiada blokadę do zapisu. Transakcja  $T$  może zapisać wartość  $X'$  zgodnie z potrzebą bez wpływu na wartość zatwierdzonej wersji elementu  $X$ . Jednakże kiedy transakcja  $T$  jest gotowa do zatwierdzenia, musi uzyskać **blokadę do certyfikacji** (ang. *certify lock*) na wszystkich elementach, na których w danej chwili posiada blokady do zapisu, zanim będzie mogła zostać zatwierdzona; jest to inna forma **rozszerzania blokady**. Blokada do certyfikacji nie jest zgodna z blokadami do odczytu, więc transakcja może zostać zmuszona do opóźnienia swojego zatwierdzenia w związku z koniecznością uzyskania blokady do certyfikacji do momentu, aż wszystkie jej elementy zablokowane do zapisu zostaną zwolnione przez transakcje odczytujące. Kiedy blokady do certyfikacji (będące blokadami wyłącznymi) zostaną uzyskane, zatwierdzona wersja elementu  $X$  jest ustawiana na wartość wersji  $X'$ , wersja  $X'$  jest usuwana, a następnie zostaje zwolniona blokada do certyfikacji. Tabela zgodności blokad dla takiego schematu została przedstawiona na rysunku 21.6(b).

W przypadku opisywanego schematu wielowersyjnego blokowania dwufazowego operacje odczytu mogą być wykonywane współbieżnie z pojedynczą operacją zapisu — nie dopuszczają tego standardowe schematy blokowania dwufazowego. Wszystko dzieje się kosztem tego, że zatwierdzenie transakcji może zostać opóźnione do momentu uzyskania przez nią wyłącznych blokad do certyfikacji na *wszystkich elementach*, które aktualizuje. Można wykazać, że taki schemat pozwala uniknąć anulowania kaskadowego, gdyż transakcje mogą odczytywać tylko tę wersję elementu  $X$ , która została zapisana przez transakcję zatwierdzoną. Jednak występuje tu problem zakleszczenia; rozwiązaniem tego problemu jest stosowanie różnych odmian technik opisanych w podrozdziale 21.1.3.



## 21.4. Sterowanie współbieżne z użyciem technik walidacyjnych (optymistycznych) i izolacji snapshotów

W przypadku wszystkich dotąd omówionych technik sterowania współbieżnego określone sprawdzenia były przeprowadzane *przed* wykonaniem operacji w bazie danych. Przykładowo, w przypadku blokowania sprawdzenie dotyczy określenia, czy element, do którego jest uzyskiwany dostęp, nie jest zablokowany. W przypadku porządkowania według znaczników czasu porównuje się znacznik czasu transakcji ze znacznikami czasu odczytu i zapisu danego elementu. Takie sprawdzenia generują dodatkowe koszty wykonywania transakcji, co powoduje ogólne spowolnienie działań.

W przypadku **technik optymistycznego sterowania współbieżnego** (ang. *optimistic concurrency control techniques*), znanych również jako techniki **walidacyjne** lub **certyfikacyjne**, w czasie wykonywania transakcji nie są przeprowadzane *żadne sprawdzenia*. Kilka zaproponowanych metod sterowania współbieżnego wykorzystuje technikę walidacyjną. W punkcie 21.4.1 zostanie opisany tylko jeden taki schemat. Następnie, w punkcie 21.4.2, omówimy techniki sterowania współbieżnego oparte na **izolacji snapshotów**. W implementacjach tych metod sterowania współbieżnego stosowane są różne rozwiązania oparte na walidacji i kontrolowaniu wersji. Używane są też znaczniki czasu. W niektórych z tych metod mogą występować anomalie naruszające szeregowalność, jednak ponieważ koszty są tu zwykle niższe niż w blokowaniu dwufazowym, metody te są używane w kilku relacyjnych SZBD.

### 21.4.1. Walidacyjne (optymistyczne) sterowanie współbieżne

W jego przypadku aktualizacje uwzględnione w ramach transakcji *nie* są bezpośrednio stosowane względem elementów bazy danych do momentu, aż transakcja osiągnie swój koniec i przejdzie *walidację*. W czasie jej wykonywania wszystkie aktualizacje są stosowane względem *kopii lokalnych* elementów danych, przechowywanych dla celów transakcji<sup>6</sup>. Na końcu wykonania transakcji, w **fazie walidacji**, następuje sprawdzenie, czy któreś z aktualizacji transakcji nie naruszyły szeregowalności. System musi przechowywać pewne informacje potrzebne w fazie walidacji. Jeżeli okaże się, że szeregowalność nie została naruszona, transakcja zostaje zatwierdzona, a baza danych zaktualizowana na podstawie kopii lokalnych. W przeciwnym razie transakcja jest anulowana i ponawiana w późniejszym okresie.

Można wyróżnić trzy fazy takiego protokołu sterowania współbieżnego:

- (1) **Faza odczytu.** Transakcja może odczytywać wartości zatwierdzonych elementów danych z bazy. Jednak aktualizacje są stosowane tylko względem kopii lokalnych (wersji) elementów danych przechowywanych w obszarze roboczym transakcji.
- (2) **Faza walidacji.** Wykonywane jest sprawdzenie, mające na celu zapobieżenie naruszeniu szeregowalności w przypadku, gdy aktualizacje transakcji zostaną zastosowane względem bazy danych.

---

<sup>6</sup> Warto zauważyć, że można to postrzegać jako przechowywanie wielu wersji elementów.



- (3) **Faza zapisu.** W razie powodzenia fazy walidacji aktualizacje transakcji są stosowane względem bazy danych. W przeciwnym razie aktualizacje są odrzucane, a transakcja ponawiana.

Idea funkcjonowania schematu optymistycznego sterowania współbieżnego polega na wykonywaniu wszystkich sprawdzeń razem, dzięki czemu wykonywanie transakcji wiąże się z minimalnym narzutem do momentu osiągnięcia fazy walidacji. Jeżeli między transakcjami występuje niewiele konfliktów, większość z nich jest określana jako poprawne. Jeśli jednak konflikty są częste, wiele transakcji wykonanych do końca spotyka się z odrzuceniem swoich wyników i musi być ponawianych. W takiej sytuacji techniki optymistyczne nie sprawdzają się zbyt dobrze. Techniki te określa się mianem *optymistycznych*, ponieważ zakładamy w ich przypadku, że występuje niewiele konfliktów, a więc nie ma potrzeby wykonywania sprawdzeń w czasie wykonywania transakcji. W wielu rodzajach obciążenia roboczego z przetwarzaniem transakcji to założenie jest zwykle prawdziwe.

Protokół optymistyczny, który opisujemy, wykorzystuje znaczniki czasu transakcji i wymaga, aby system przechowywał zbiory zapisów i zbiory odczytów transakcji. Ponadto dla każdej transakcji muszą być przechowywane informacje o czasie *początkowym* i *końcowym* trzech omawianych faz. Warto przypomnieć, że zbiór\_zapisów transakcji jest zbiorem elementów, które są przez nią zapisywane, zaś zbiór\_odczytów to zbiór elementów przez nią odczytywanych. W fazie walidacji transakcji  $T_i$  protokół sprawdza, czy  $T_i$  nie jest w konflikcie z którąś z zatwierdzonych transakcji lub z innymi transakcjami znajdującymi się w fazie walidacji. Faza walidacji transakcji  $T_i$  służy do sprawdzenia dla *każdej* takiej transakcji  $T_j$ , która albo została zatwierdzona, albo znajduje się w fazie walidacji, czy spełniony jest *jeden* z następujących warunków:

- (1) Transakcja  $T_j$  kończy swoją fazę zapisu zanim transakcja  $T_i$  rozpocznie swoją fazę odczytu.
- (2) Transakcja  $T_i$  rozpoczyna swoją fazę zapisu po tym, jak transakcja  $T_j$  zakończyła swoją fazę zapisu, oraz zbiór\_odczytów transakcji  $T_i$  nie posiada żadnych elementów wspólnych ze zbiorem\_zapisów transakcji  $T_j$ .
- (3) Zarówno zbiór\_odczytów, jak i zbiór\_zapisów transakcji  $T_i$  nie posiada żadnych elementów wspólnych ze zbiorem\_zapisów transakcji  $T_j$  i transakcja ta kończy swoją fazę odczytu zanim zrobi to transakcja  $T_i$ .

W czasie walidacji transakcji  $T_i$  najpierw sprawdzany jest pierwszy warunek dla każdej transakcji  $T_j$ , gdyż jest on najprostszy do zweryfikowania. Tylko wówczas, gdy warunek (1) nie jest spełniony, dokonuje się sprawdzenia warunku (2) i tylko wówczas, gdy warunek (2) nie jest spełniony, dokonuje się sprawdzenia warunku (3) — najbardziej skomplikowanego. Jeżeli jakikolwiek z tych trzech warunków jest spełniony, to kolizja nie występuje i walidacja transakcji  $T_i$  kończy się powodzeniem. Jeżeli nie jest spełniony *żaden* z tych trzech warunków, walidacja transakcji  $T_i$  kończy się niepowodzeniem (ponieważ  $T_i$  i  $T_j$  mogą naruszać szeregowałość) i jest ona anulowana i ponowiona w późniejszym okresie, gdyż oznacza to, że *mogła* wystąpić kolizja z  $T_j$ .

## 21.4.2. Sterowanie współbieżne oparte na izolacji snapshotów

W podrozdziale 20.6 wyjaśniono, że według podstawowej definicji **izolacja snapshotów** polega na tym, że transakcja widzi wczytywane elementy danych zgodnie z zatwierdzonymi wartościami tych elementów ze *snapshotu bazy danych* (inaczej: stanu bazy danych) z momentu rozpoczęcia transakcji. Izolacja snapshotów gwarantuje ochronę przed problemem rekordów fantomowych, ponieważ transakcja w bazie danych (lub, w niektórych sytuacjach, instrukcja) zobaczy tylko rekordy zatwierdzone w bazie w chwili uruchomienia tej transakcji. Operacje wstawiania, usuwania lub aktualizowania wykonane po rozpoczęciu transakcji nie będą w niej widoczne. Ponadto izolacja snapshotów chroni przed problemami odczytu zmodyfikowanego i odczytu niepowtarzalnego. Jednak gdy sterowanie współbieżne odbywa się na podstawie izolacji snapshotów, mogą wystąpić pewne anomalie naruszające szeregowalność. Choć są one rzadkie, bardzo trudno jest je wykryć i mogą skutkować utratą spójności lub uszkodzeniem baz. Zainteresowani Czytelnicy znajdą w bibliografii z końca rozdziału prace ze szczegółowym omówieniem rzadkich rodzajów anomalii, jakie mogą występować.

W tym modelu operacje odczytu nie wymagają stosowania blokad odczytu do elementów. Zmniejsza to dodatkowe koszty powiązane z blokowaniem dwufazowym. Jednak zapis wymaga blokad zapisu. Dlatego w transakcjach z dużą liczbą odczytów wydajność tego rozwiązania jest znacznie wyższa niż wydajność blokowania dwufazowego. W momencie zapisu system musi przechowywać starsze wersje zaktualizowanych elementów w **tymczasowej pamięci wersji**. Przechowywane są też znaczniki czasu utworzenia tych wersji. Jest to konieczne, aby transakcja, która rozpoczęła pracę przed zapisaniem elementu, mogła wczytać wartość (wersję) tego elementu, jaka znajdowała się w snapshotcie bazy w momencie uruchomienia transakcji.

Na potrzeby śledzenia wersji dla zaktualizowanych elementów przechowywane są wskaźniki do listy nowych wersji elementów z *pamięci tymczasowej*. Dzięki temu w każdej transakcji można wczytać odpowiedni element. Elementy są usuwane z pamięci podręcznej, gdy nie są już potrzebne. Konieczna jest więc metoda decydowania o tym, kiedy usuwać zbędne wersje.

Odmiany tej metody są używane w kilku komercyjnych i otwartych SZBD, w tym w systemach Oracle i PostGRES. Jeśli użytkownicy wymagają gwarancji szeregowalności, programiści i inżynierowie oprogramowania muszą rozwiązać problemy z anomaliami naruszającymi szeregowalność. Wymaga to przeanalizowania zestawu transakcji i ustalenia, jakie rodzaje anomalii mogą wystąpić, oraz dodania chroniących przed nimi testów. Może to zwiększać obciążenie programistów w porównaniu z sytuacją, gdy używany jest SZBD wymuszający szeregowalność w każdej sytuacji.

Zaproponowana została odmiana izolacji snapshotów (IS), **szeregowalna izolacja snapshotów** (SIS), którą zaimplementowano w wybranych SZBD używających IS jako podstawowej metody sterowania współbieżnego. Przykładowo, nowe wersje SZBD PostGRES umożliwiają użytkownikom wybór między podstawową IS a SIS. Kompromis polega tu na zagwarantowaniu pełnej szeregowalności za pomocą SIS lub zaakceptowaniu rzadkich anomalii i zyskaniu wyższej wydajności dzięki podstawowej IS. Zainteresowani użytkownicy dokładne omówienie tych zagadnień znajdą w pracach z bibliografii z końcowej części rozdziału.

## 21.5. Ziarnistość elementów danych i blokowanie z wieloma poziomami ziarnistości

W przypadku wszystkich technik sterowania współbieżnego zakłada się, że baza danych składa się z wielu nazwanych elementów danych. Elementem bazy danych może być:

- rekord bazodanowy;
- wartość pola rekordu bazodanowego;
- blok dyskowy;
- cały plik;
- cała baza danych.

Wybór elementu danych może mieć wpływ na wydajność sterowania współbieżnego i odtwarzania. W podrozdziale 21.5.1 zostaną omówione pewne kompromisy podejmowane w przypadku wyboru różnych poziomów ziarnistości używanych w celu blokowania, zaś w podrozdziale 21.5.2 zostanie opisany schemat blokowania z wieloma poziomami ziarnistości, gdzie poziom ziarnistości (rozmiar elementów danych) można dynamicznie zmieniać.

### 21.5.1. Kwestie dotyczące poziomu ziarnistości w przypadku blokowania

Rozmiar elementów danych często określa się mianem **ziarnistości elementu danych** (ang. *data item granularity*). Pojęcie *drobnoziarnistości* (ang. *fine granularity*) odnosi się do niewielkich rozmiarów elementów, natomiast pojęcie *gruboziarnistości* (ang. *coarse granularity*) odnosi się do dużych rozmiarów elementów. W przypadku dokonywania wyboru rozmiaru elementów danych należy uwzględnić pewne kompromisy. Rozmiar elementów danych będziemy rozpatrywać w kontekście blokowania, choć podobne uwagi można poczynić w przypadku innych technik sterowania współbieżnego.

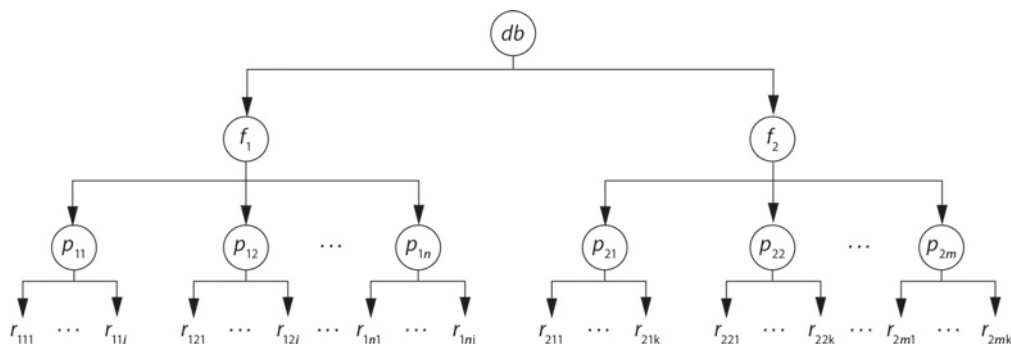
Po pierwsze, należy zauważyć, że im większy rozmiar elementu danych, tym niższy dozwolony poziom współbieżności. Przykładowo, jeżeli rozmiarem elementu danych jest blok dyskowy, transakcja *T*, która musi zablokować rekord *B*, musi jednocześnie zablokować cały blok dyskowy *X*, zawierający rekord *B*, ponieważ blokada jest związana z całym elementem danych (blokiem). Jeżeli inna transakcja *S* zechce zablokować inny rekord *C* znajdujący się w tym samym bloku *X* w trybie blokady konfliktowej, jest zmuszona czekać. Gdyby rozmiarem elementu danych był rekord, transakcja *S* mogłaby kontynuować działania, ponieważ blokowałaby inny element danych (rekord).

Z drugiej strony, im mniejszy rozmiar elementu danych, tym w bazie znajduje się ich większa liczba. Ze względu na fakt, że każdy element jest związany z blokadą, system posiada dużą liczbę blokad aktywnych, które musi obsługiwać menedżer blokad. Wykonywana jest większa liczba operacji blokowania i odblokowania, co wiąże się z większym narzutem. Ponadto, tabela blokad wymaga większej przestrzeni. W przypadku znaczników czasu wymagana jest przestrzeń służąca do przechowywania znaczników *TS\_odczytu* i *TS\_zapisu* dla każdego elementu danych, co wiąże się z podobnym narzutem dotyczącym obsługi dużej liczby elementów.

Biorąc pod uwagę konieczność podejmowania opisanych powyżej kompromisów, pojawia się oczywiste pytanie: jaki jest najlepszy rozmiar elementu danych? Odpowiedź brzmi: *zależy to od rodzaju wykonywanych transakcji*. Jeżeli typowe transakcje uzyskują dostęp do niewielkiej liczby rekordów, korzystnym rozwiązaniem jest ustalenie rozmiaru elementu danych na pojedynczy rekord. Z drugiej strony, jeżeli typowe transakcje uzyskują dostęp do wielu rekordów w tym samym pliku, być może lepszym rozwiązaniem będzie określenie ziarnistości na poziomie bloku lub pliku, tak aby transakcje mogły wszystkie takie rekordy traktować jako jeden (a najwyżej kilka) elementów danych.

## 21.5.2. Blokowanie z wieloma poziomami ziarnistości

Ze względu na fakt, że najlepszy rozmiar ziarnistości zależy od rodzaju transakcji, odpowiednie wydaje się, aby system bazy danych obsługiwał wiele poziomów ziarnistości, gdzie poziom ten może być różny w przypadku różnych transakcji. Na rysunku 21.7 przedstawiono prostą hierarchię ziarnistości dla bazy zawierającej dwa pliki, z których każdy składa się z kilku stron, a każda strona — z kilku rekordów. Można tym zilustrować protokół **blokowania dwufazowego z wieloma poziomami ziarnistości** (ang. *multiple granularity level 2PL*) z trybami blokad na wyłączność i dzielonych, gdzie blokady można zażądać na każdym poziomie. Jednak w celu zapewnienia wydajnej obsługi takiego protokołu potrzebne są dodatkowe rodzaje blokad.



Rysunek 21.7. Hierarchia ziarnistości ilustrująca blokowanie z wieloma poziomami ziarnistości

Weźmy pod uwagę następujący scenariusz, uwzględniający tylko blokady dzielone i wyłączne, odwołujący się do przykładu z rysunku 21.7. Załóżmy, że transakcja  $T_1$  chce zaktualizować *wszystkie rekordy* w pliku  $p_1$  i zgłasza żądanie, otrzymując blokadę wyłączną na pliku  $p_1$ . Następnie wszystkie strony pliku  $p_1$  (od  $s_{11}$  do  $s_{1n}$ ), wraz z zawartymi w nich rekordami, zostają zablokowane w trybie wyłącznym. Jest to korzystne z punktu widzenia transakcji  $T_1$ , ponieważ ustawienie pojedynczej blokady na pliku jest wydajniejsze od ustawiania  $n$  blokad na poziomie strony lub blokowania każdego rekordu oddzielnie. Załóżmy teraz, że inna transakcja  $T_2$  chce jedynie odczytać rekord  $r_{1nj}$  ze strony  $s_{1n}$  pliku  $p_1$ . Transakcja ta zgłasza żądanie względem dzielonej blokady na poziomie rekordu na rekordzie  $r_{1nj}$ . Jednak system bazodanowy (to znaczy menedżer transakcji, a dokładniej menedżer blokad) musi zweryfikować zgodność żądanej blokady z blokadą już ustawioną. Jednym ze sposobów zapewniających takie działanie jest przejście drzewa od

liścia  $r_{1nj}$  przez wierzchołki  $s_{1n}$  i  $p_1$  do  $bd$ . Jeżeli na któryś z tych elementów nałożono blokadę konfliktową, żądanie zablokowania rekordu  $r_{1nj}$  jest odrzucane, transakcja  $T_2$  jest blokowana i musi czekać. Takie przechodzenie byłoby całkiem wydajne.

Co jednak w przypadku, gdyby żądanie transakcji  $T_2$  nadeszło *przed* żądaniem transakcji  $T_1$ ? W takim przypadku dzielona blokada na poziomie rekordu zostaje przyznana transakcji  $T_2$  na rekordzie  $r_{1nj}$ , ale kiedy pojawia się żądanie transakcji  $T_1$  o blokadę na poziomie pliku, trudnym zadaniem dla menedżera blokad jest sprawdzenie w kontekście konfliktów blokad wszystkich wierzchołków (stron i rekordów), które są potomkami wierzchołka  $p_1$ . Byłoby to bardzo niewydajne rozwiązanie, stojące w sprzeczności z rozwiązaniem polegającym na wykorzystaniu blokad o wielu poziomach ziarnistości.

W celu zapewnienia praktyczności blokowania z wieloma poziomami ziarnistości potrzebne jest wprowadzenie dodatkowych rodzajów blokad, określanych mianem **blokad intencyjnych** (ang. *intention locks*). Idea ich działania polega na umożliwieniu transakcjom określania, jakiego rodzaju blokady (dzielonej lub wyłącznej) będą wymagały od jednego z potomków danego wierzchołka. Istnieją trzy rodzaje blokad intencyjnych:

- (1) *Dzielone-intencyjne* (ang. *intention-shared, IS*), które wskazują, że dzielona blokada(y) będzie potrzebna na którymś z wierzchołków podrzędnych.
- (2) *Wyłączne-intencyjne* (ang. *intention-exclusive, IX*), które wskazują, że wyłączna blokada(y) będzie potrzebna na którymś z wierzchołków podrzędnych.
- (3) *Dzielone-intencyjne-wyłączne* (ang. *shared-intention-exclusive, SIX*), które wskazują, że bieżący rekord jest zablokowany w trybie dzielnym, ale na którymś z wierzchołków podrzędnych będzie potrzebna blokada wyłączna.

Tabelę zgodności trzech blokad intencyjnych oraz blokad dzielonych i wyłącznych przedstawiono na rysunku 21.8. Oprócz wprowadzenia trzech rodzajów blokad intencyjnych należy także zastosować odpowiedni protokół blokowania. Protokół **blokowania z wieloma poziomami ziarnistości** (ang. *multiple granularity protocol, MGL*) określa następujące reguły:

- (1) Należy trzymać się zgodności blokad (zgodnie z rysunkiem 21.8).
- (2) Korzeń drzewa musi zostać zablokowany jako pierwszy w dowolnym trybie.
- (3) Wierzchołek  $N$  może zostać zablokowany przez transakcję  $T$  w trybie  $S$  lub  $IS$  tylko wtedy, gdy wierzchołek nadrzędny dla  $N$  został już zablokowany przez transakcję  $T$  w trybie  $IS$  lub  $IX$ .
- (4) Wierzchołek  $N$  może zostać zablokowany przez transakcję  $T$  w trybie  $X$ ,  $IX$  lub  $SIX$  tylko wtedy, gdy wierzchołek nadrzędny dla  $N$  został już zablokowany przez transakcję  $T$  w trybie  $IX$  lub  $SIX$ .
- (5) Transakcja  $T$  może zablokować wierzchołek tylko wtedy, gdy nie zwolniła blokady żadnego wierzchołka (w celu wymuszenia stosowania protokołu blokowania dwufazowego).
- (6) Transakcja  $T$  może odblokować wierzchołek  $N$  tylko wtedy, gdy żaden z potomków wierzchołka  $N$  nie jest zablokowany przez transakcję  $T$ .

	IS	IX	S	SIX	X
IS	Tak	Tak	Tak	Tak	Nie
IX	Tak	Tak	Nie	Nie	Nie
S	Tak	Nie	Tak	Nie	Nie
SIX	Tak	Nie	Nie	Nie	Nie
X	Nie	Nie	Nie	Nie	Nie

Rysunek 21.8. Macierz zgodności blokad dla blokowania z wieloma poziomami ziarnistości

Reguła 1. określa po prostu, że blokady konfliktowe nie mogą być przyznawane. Reguły 2., 3. i 4. określają sytuacje, w których transakcja może zablokować dany wierzchołek w dowolnym z trybów blokowania. Reguły 5. i 6. protokołu MGL wymuszają reguły blokowania dwufazowego w celu utworzenia harmonogramów szeregowlanych. Blokowanie *rozpoczyna się od korzenia* i odbywa się w dół drzewa aż do momentu napotkania wierzchołka, który wymaga blokady. Usuwanie blokady *rozpoczyna się od zablokowanego wierzchołka* i odbywa się w górę drzewa aż do momentu odblokowania korzenia. W celu zilustrowania protokołu MGL z hierarchią bazy danych z rysunku 21.7 weźmy pod uwagę następujące trzy transakcje:

- (1) Transakcja  $T_1$  podejmuje próbę aktualizacji rekordu  $r_{111}$  oraz rekordu  $r_{211}$ .
- (2) Transakcja  $T_2$  podejmuje próbę aktualizacji wszystkich rekordów strony  $s_{12}$ .
- (3) Transakcja  $T_3$  podejmuje próbę odczytania rekordu  $r_{11j}$  oraz całego pliku  $p_2$ .

Na rysunku 21.9 przedstawiono możliwy harmonogram szeregowalny dla tych trzech transakcji. Pokazano na nim jedynie operacje blokowania i odblokowywania. Notacja  $\langle \text{rodzaj\_blokad} \rangle (\langle \text{element} \rangle)$  jest używana w celu przedstawienia operacji blokowania w harmonogramie.

Protokół blokowania z wieloma poziomami ziarnistości jest szczególnie przydatny w przypadku przetwarzania różnych rodzajów transakcji: (1) krótkich transakcji, uzyskujących dostęp tylko do kilku elementów (rekordów lub pól), (2) długich transakcji, uzyskujących dostęp do całych plików. W takim środowisku podobny protokół pozwala na zmniejszenie ilości blokad transakcji oraz zmniejszenie narzutu związanego z mechanizmami blokowania w porównaniu z podejściem wykorzystującym blokowanie na jednym poziomie.

## 21.6. Użycie blokad dla celów sterowania współbieżnego w przypadku indeksów

Blokowanie dwufazowe może również być stosowane względem indeksów w postaci B-drzew i B+-drzew (patrz rozdział 19.), gdzie wierzchołki indeksu odpowiadają stronom dyskowym. Jednakże, trzymanie blokad na stronach indeksów aż do fazy kurczenia protokołu blokowania dwufazowego mogłoby powodować zbyt wiele blokad transakcji. Dzieje

$T_1$	$T_2$	$T_3$
IX(db) IX( $f_1$ )	IX(db)	IS(db) IS( $f_1$ ) IS( $p_{11}$ )
IX( $p_{11}$ ) X( $r_{111}$ )	IX( $f_1$ ) X( $p_{12}$ )	S( $r_{111}$ )
IX( $f_2$ ) IX( $p_{21}$ ) X( $p_{211}$ )		S( $f_2$ )
odblokuj( $r_{211}$ ) odblokuj( $p_{21}$ ) odblokuj( $f_2$ )	odblokuj( $p_{12}$ ) odblokuj( $f_1$ ) odblokuj(db)	
odblokuj( $r_{111}$ ) odblokuj( $p_{11}$ ) odblokuj( $f_1$ ) odblokuj(db)		odblokuj( $r_{111}$ ) odblokuj( $p_{11}$ ) odblokuj( $f_1$ ) odblokuj( $f_2$ ) odblokuj(db)

RYSUNEK 21.9. Operacje blokowania ilustrujące harmonogram szeregowalny

się tak dlatego, że przeszukiwanie indeksu zawsze *rozpoczyna się od korzenia*, więc jeżeli transakcja chce wstawić rekord (operacja zapisu), korzeń zostaje zablokowany w trybie wyłącznym, a więc wszystkie inne konfliktowe żądania blokad względem indeksu muszą czekać do momentu, aż transakcja wejdzie w fazę kurczenia. Powoduje to zablokowanie innych transakcji, które chciałyby uzyskać dostęp do indeksu, więc w praktyce należy zastosować inne podejście do kwestii blokowania indeksów.

Struktura drzewiasta indeksu może być wykorzystana w przypadku opracowywania schematu sterowania współbieżnego. Przykładowo, w czasie wykonywania przeszukiwania indeksu (operacja odczytu) ścieżka w drzewie jest pokonywana od korzenia do liścia. Kiedy zostanie uzyskany dostęp do wierzchołka niższego poziomu, wierzchołki wyższego poziomu nie są już ponownie używane. Tak więc kiedy na wierzchołku podrzędnym zostanie uzyskana blokada do odczytu, blokada na wierzchołku nadrzędnym może zostać zwolniona. Po drugie, kiedy względem wierzchołka liścia jest stosowana operacja



wstawienia (to znaczy, kiedy jest wstawiany klucz i wskaźnik), wówczas określony wierzchołek liścia musi zostać zablokowany w trybie wyłącznym. Jednakże jeśli taki wierzchołek nie jest pełny, operacja wstawiania nie powoduje zmian w wierzchołkach indeksu wyższego poziomu, co implikuje, że nie muszą one być blokowane w trybie wyłącznym.

Konserwatywne podejście do kwestii wstawiania polegałoby na zablokowaniu wierzchołka korzenia w trybie wyłącznym, a następnie na uzyskaniu dostępu do odpowiedniego wierzchołka potomnego korzenia. Jeżeli wierzchołek potomny nie jest pełny, wówczas można zwolnić blokadę na wierzchołku korzenia. Podejście takie może być stosowane wzdłuż całej ścieżki aż do liścia, który zazwyczaj znajduje się trzy lub cztery poziomy poniżej korzenia. Choć trzymane są blokady wyłączne, są one dość szybko zwalniane. Alternatywnym, bardziej **optymistycznym podejściem** byłoby żądanie i trzymanie blokad *dzielonych* na wierzchołkach wiodących do wierzchołka liścia, na który byłaby nakładana blokada *wyłączna*. Jeżeli operacja wstawiania powoduje podział liścia, wstawianie jest propagowane na wyższy poziom wierzchołka(ów). Następnie blokady na wierzchołku(ach) wyższego poziomu mogą być rozszerzane do trybu wyłącznego.

Inne podejście do kwestii blokowania indeksów polega na użyciu odmiany B<sup>+</sup>-drzewa, noszącego nazwę **drzewa B-powiązań** (ang. *B-link tree*). W przypadku takiego drzewa wierzchołki siostrzane znajdujące się na tym samym poziomie są połączone. Pozwala to na użycie blokad dzielonych w przypadku żądań dostępu do stron i wymaga zwolnienia blokady przed uzyskaniem dostępu do wierzchołka. W przypadku operacji wstawiania blokada dzielona na wierzchołku jest rozszerzana do blokady wyłącznej. Jeżeli występuje podział wierzchołka, wierzchołek nadrzędny musi zostać ponownie zablokowany w trybie wyłącznym. Jedną z komplikacji wiąże się z operacjami wyszukiwania wykonywanymi współbieżnie z aktualizowaniem. Założmy, że współbieżna operacja aktualizacji jest związana z taką samą ścieżką jak ścieżka operacji wyszukiwania i powoduje wstawienie nowego wpisu do wierzchołka liścia. Ponadto założmy, że owo wstawienie powoduje podział wierzchołka liścia. Po zakończeniu wstawiania proces wyszukiwania jest wznowiany i podąża zgodnie ze wskaźnikami do odpowiedniego liścia, ale tylko po to, by stwierdzić, że poszukiwanego klucza nie ma tam, ponieważ operacja podziału przeniosła go do nowego wierzchołka liścia, który jest *prawym wierzchołkiem siostrzanym* oryginalnego wierzchołka liścia. Jednakże proces wyszukiwania wciąż może zakończyć się powodzeniem, jeżeli podąży on za wskaźnikiem (łączem) z oryginalnego wierzchołka liścia do jego prawego wierzchołka siostrzanego, do którego został przeniesiony poszukiwany klucz.

Obsługa przypadku usuwania, gdzie scalane są dwa lub więcej wierzchołków z drzewa indeksu, również stanowi część protokołu współbieżności drzewa B-powiązań. W takim przypadku trzymane są blokady na scalanych wierzchołkach, jak również blokada na wierzchołku nadrzędnym dwóch scalanych wierzchołków.

## 21.7. Inne kwestie związane ze sterowaniem współbieżnym

W niniejszym podrozdziale zostaną omówione pewne dodatkowe kwestie związane ze sterowaniem współbieżnym. W punkcie 21.7.1 zostaną omówione problemy związane z wstawianiem i usuwaniem rekordów oraz tak zwany *problem rekordów fantomowych*, który

może występować w przypadku wstawiania rekordów. Problem ten został opisany w podrozdziale 20.6 jako potencjalnie wymagający sterowania współbieżnego. Następnie, w punkcie 21.7.2, zostaną omówione problemy, które mogą występować w przypadku, gdy transakcja przekazuje na monitor pewne dane wyjściowe przed swoim zatwierdzeniem, a później jest anulowana.

### 21.7.1. Wstawianie, usuwanie i rekordy fantomowe

Kiedy nowy element danych jest **wstawiany** do bazy, oczywiście nie można uzyskać do niego dostępu do czasu, aż zostanie utworzony, a operacja wstawiania zakończona. W środowisku z blokowaniem blokada na elemencie może zostać utworzona i ustawiona w trybie wyłącznym (do zapisu). Blokadę tę można zwolnić w tym samym czasie, gdy są zwalniane inne blokady do zapisu w oparciu o używany protokół sterowania współbieżnego. W przypadku protokołu opartego na znacznikach czasu znaczniki odczytu i zapisu nowego elementu są ustawiane na wartość znacznika transakcji tworzącej.

Weźmy także pod uwagę **operację usuwania**, która jest stosowana względem istniejącego elementu danych. W przypadku protokołów blokujących ponownie należy uzyskać blokadę wyłączną (do zapisu), zanim transakcja będzie mogła usunąć element. W przypadku porządkowania według znaczników czasu protokół musi zapewniać, aby żadna późniejsza transakcja nie odczytała ani nie zapisała elementu przed pozwoleniem na jego usunięcie.

Sytuacja znana jako **problem fantomowy** (ang. *phantom problem*) może wystąpić wówczas, gdy nowy rekord wstawiany przez pewną transakcję  $T$  spełnia warunek nałożony na rekordy przez pewną inną transakcję  $T'$ . Przykładowo, załóżmy, że transakcja  $T$  wstawia nowy rekord do relacji PRACOWNIK z atrybutem NRZD = 5, natomiast transakcja  $T'$  uzyskuje dostęp do wszystkich rekordów relacji PRACOWNIK, dla których NRZD = 5 (na przykład w celu zsumowania wszystkich wartości pola PENSJA, aby obliczyć budżet personalny dla działu o numerze 5). Jeżeli równoważnym porządkiem szeregowym jest kolejność transakcji  $T, T'$ , to transakcja  $T'$  musi odczytać nowy rekord relacji PRACOWNIK i uwzględnić wartość jego pola PENSJA w obliczeniach sumy. Dla równoważnego uporządkowania transakcji  $T, T'$  nowa pensja nie zostałaby uwzględniona. Należy jednak zauważyć, że choć transakcje podlegają konfliktowi logicznemu, w tym drugim przypadku nie istnieje między nimi żaden rekord (element danych) wspólny, gdyż transakcja  $T'$  mogła zablokować wszystkie rekordy o wartości atrybutu NRZD = 5 *zanim* transakcja  $T$  wstawiła nowy rekord. Dzieje się tak dlatego, że rekord powodujący konflikt jest **rekordem fantomowym** (ang. *phantom record*), który nagle pojawił się w bazie danych w momencie wstawienia. Jeżeli inne operacje w ramach obu transakcji podlegają konfliktowi, konflikt związany z rekordem fantomowym może nie zostać rozpoznany przez protokół sterowania współbieżnego.

Jednym z rozwiązań problemu rekordów fantomowych jest użycie **blokowania indeksowego** (ang. *index locking*), zgodnie z omówieniem zawartym w podrozdziale 21.6. W rozdziale 19. stwierdzono, że indeks zawiera wpisy z wartością atrybutu oraz zbiór wskaźników na wszystkie rekordy w pliku o danej wartości. Przykładowo, indeks na polu NRZD pliku PRACOWNIK zawierałby wpis dla każdej odrębnej wartości NRZD oraz zbiór wskaźników na wszystkie rekordy pliku PRACOWNIK z taką wartością. Jeżeli wpis indeksu zostanie

zablokowany zanim stanie się możliwy dostęp do danego rekordu, można wówczas wykryć konflikt związany z rekordem fantomowym. Dzieje się tak dlatego, że transakcja  $T'$  zażądałaby blokady do odczytu na *wpisie indeksu* dla  $NRDZ = 5$ , zaś transakcja  $T$  zażądałaby blokady do zapisu na tym samym wpisie *zanim* byłoby możliwe założenie blokad na faktycznych rekordach. Ze względu na fakt, że wystąpi konflikt blokad indeksu, pozwala to na wykrycie konfliktu rekordu fantomowego.

Bardziej ogólna technika, znana jako **blokowanie predykatowe** (ang. *predicate locking*), blokuje dostęp do wszystkich rekordów spełniających dowolny *predykat* (warunek) w podobny sposób, jednak blokady predykatowe okazały się trudne w wydajnej implementacji. Jeśli sterowanie współbieżne jest oparte na izolacji snapshotów (patrz punkt 21.4.2), transakcja wczytująca elementy używa snapshotu bazy danych z momentu rozpoczęcia tej transakcji. Wszystkie później wstawione rekordy nie będą pobierane przez tę transakcję.

### 21.7.2. Transakcje interaktywne

Kolejny problem występuje w sytuacji, gdy interaktywne transakcje odczytują dane wejściowe i zapisują dane wyjściowe na interaktywne urządzenie, takie jak ekran monitora, zanim zostaną zatwierdzone. Problem polega na tym, że użytkownik może do transakcji  $T$  wstawić wartość elementu danych, opartą na pewnej wartości zapisanej na ekran przez transakcję  $T'$ , która mogła zostać niezatwierdzona. Taka zależność między transakcjami  $T$  i  $T'$  nie może zostać uwzględniona w modelu metody systemu sterowania współbieżnego, gdyż jest ona oparta tylko na interakcji użytkownika z dwiema transakcjami.

Rozwiązanie tego problemu polega na opóźnieniu przekazywania danych wyjściowych transakcji na ekran do momentu ich zatwierdzenia.

### 21.7.3. Zatrzaski

Blokady nakładane na krótki okres czasu określa się zwykle mianem **zatrzasków** (ang. *latches*). Zatrzaski nie są zgodne z tradycyjnym protokołem sterowania współbieżnego, takim jak blokowanie dwufazowe. Przykładowo, zatrzask może być używany w celu zagwarantowania fizycznej spójności strony w czasie jej zapisywania z bufora na dysk. Dla strony jest przydzielany zatrzask, zostaje ona zapisana na dysku, a następnie zatrzask jest zwalniany.

## 21.8. Podsumowanie

W niniejszym rozdziale omówiono techniki SZBD dotyczące sterowania współbieżnego. Rozpoczęto od omówienia w podrozdziale 21.1 protokołów blokowania, które są zdecydowanie najczęściej stosowanymi w praktyce. W punkcie 21.1.2 opisano protokół blokowania dwufazowego oraz wiele jego odmian: podstawowe blokowanie dwufazowe, ścisłe blokowanie dwufazowe, konserwatywne blokowanie dwufazowe oraz rygorystyczne blokowanie dwufazowe. Odmiana ścisła i rygorystyczna są używane najczęściej ze względu na swoje lepsze właściwości dotyczące odtwarzania. W punkcie 21.1.1 wprowadzono pojęcia blokad dzielonych (do odczytu) i wyłącznych (do zapisu) oraz pokazano, w jaki sposób blokowanie może zagwarantować szeregowalność w przypadku jego użycia w połączeniu z regułą blo-

kowania dwufazowego. Przedstawiono również różne techniki radzenia sobie z problemem zakleszczenia, który może występować w przypadku blokowania (punkt 21.1.3). W praktyce często wykorzystuje się limity czasowe i techniki wykrywania zakleszczeń (grafy oczekiwania). Można też zastosować protokoły zapobiegania zakleszczeniu, np. protokoły nieoczekiwania i oczekiwania ostrożnego.

Następnie przedstawiono inne protokoły sterowania współbieżnego. Chodzi tu o protokół porządkowania na podstawie znaczników czasu (podrozdział 21.2), który zapewnia szeregowalność w oparciu o kolejność znaczników. Znaczniki czasu są unikatowymi, generowanymi przez system identyfikatorami. Omówiono regułę zapisu Thomasa, która pozwala na zwiększenie wydajności, ale nie gwarantuje szeregowalności. Przedstawiono także protokół ścisłego porządkowania według znaczników czasu. Następnie omówiono dwa protokoły wielowersyjne (podrozdział 21.3), w których przypadku zakłada się, że starsze wersje elementów danych mogą być przechowywane w bazie. Jedną z takich technik, noszącą nazwę wielowersyjnego blokowania dwufazowego (używana w praktyce), zakłada, że dla każdego elementu mogą istnieć dwie jego wersje, i próbuje zwiększyć poziom współbieżności, zapewniając zgodność blokad odczytu i zapisu (za cenę wprowadzenia dodatkowego trybu blokady certyfikującej). Przedstawiono również protokół wielowersyjny bazujący na porządkowaniu według znaczników czasu. W punkcie 21.4.1 omówiono przykład protokołu optymistycznego, znanego również jako protokół certyfikacji lub walidacji.

Następnie, w punkcie 21.4.2, opisano metody sterowania współbieżnego oparte na izolacji snapshotów. Są one stosowane w kilku SZBD z powodu niskich kosztów. Podstawowa metoda izolacji snapshotów dopuszcza w rzadkich sytuacjach harmonogramy nieszerogowalne z powodu pewnych trudnych do wykrycia anomalii. Mogą one skutkować uszkodzeniem bazy danych. Niedawno opracowano odmianę tej techniki, szeregowalną izolację snapshotów, która gwarantuje szeregowalne harmonogramy.

W dalszej części rozdziału, w podrozdziale 21.5, skupiono się na istotnych kwestiach natury praktycznej, związanych z ziarnistością elementów danych. Opisano protokół blokowania o wielu poziomach ziarnistości, który pozwala na dokonywanie zmian ziarnistości (rozmiaru elementu) w oparciu o bieżące transakcje w celu zwiększenia wydajności sterowania współbieżnego. W podrozdziale 21.6 omówiono istotną kwestię praktyczną, związaną z opracowywaniem protokołów blokowania dla indeksów, tak aby nie stały się one przeszkodą w zakresie dostępu współbieżnego. Wreszcie w podrozdziale 21.7 przedstawiono problem rekordów fantomowych oraz problemy związane z transakcjami interakcyjnymi oraz pokrótce opisano pojęcie zatrząsków i różnice występujące między nimi a blokadami.

## Pytania powtórkowe

- 21.1. Czym jest protokół blokowania dwufazowego? W jaki sposób gwarantuje on szeregowalność?
- 21.2. Jakie istnieją odmiany protokołu blokowania dwufazowego? Dlaczego często preferuje się ścisłe lub rygorystyczne blokowanie dwufazowe?
- 21.3. Omów problemy zakleszczenia i zagłodzenia oraz różne podejścia do kwestii radzenia sobie z nimi.

- 21.4. Porównaj blokady binarne z blokadami wyłącznymi i dzielonymi. Dlaczego te drugie są preferowanym rozwiązaniem?
- 21.5. Opisz protokoły czekaj-kończ i zakończ-czekaj zapobiegające zakleszczeniom.
- 21.6. Opisz metodę oczekiwania ostrożnego, nieoczekiwania oraz protokoły limitów czasowych zapobiegające zakleszczeniom.
- 21.7. Czym jest znacznik czasu? W jaki sposób jest on generowany przez system?
- 21.8. Omów protokół porządkowania według znaczników czasu w zakresie sterowania współbieżnego. Czym różni się ściśle porządkowanie według znaczników czasu od porządkowania podstawowego?
- 21.9. Omów dwie techniki wielowersyjnego sterowania współbieżnego. Czym jest blokada certyfikująca? Jakie są zalety i wady używania takich blokad?
- 21.10. Czym różnią się techniki optymistycznego sterowania współbieżnego od innych technik sterowania współbieżnego? Dlaczego nazywa się je również technikami walidacji lub certyfikacji? Omów typowe fazy metody optymistycznego sterowania współbieżnego.
- 21.11. Czym jest izolacja snapshotów? Jakie są wady i zalety metod sterowania współbieżnego opartych na izolacji snapshotów?
- 21.12. Jaki wpływ na wydajność sterowania współbieżnego ma ziarnistość elementów danych? Jakie czynniki wpływają na wybór poziomu ziarnistości elementów danych?
- 21.13. Jakiego rodzaju blokady są wymagane w przypadku operacji wstawiania i usuwania?
- 21.14. Na czym polega blokowanie z wieloma poziomami ziarnistości? W jakich sytuacjach jest używane?
- 21.15. Czym są blokady intencyjne?
- 21.16. Kiedy używa się zatrząsków?
- 21.17. Czym jest rekord fantomowy? Omów problemy powodowane przez rekordy fantomowe w zakresie sterowania współbieżnego.
- 21.18. W jaki sposób blokowanie indeksowe rozwiązuje problem rekordów fantomowych?
- 21.19. Czym jest blokada predykatowa?

## Ćwiczenia

- 21.20. Udowodnij, że protokół podstawowego blokowania dwufazowego gwarantuje szeregowalność konfliktową harmonogramów (*podpowiedź*: wykaż, że jeżeli graf szeregowalności dla harmonogramu zawiera cykl, wówczas co najmniej jedna z transakcji należących do harmonogramu nie spełnia warunków protokołu blokowania dwufazowego).
- 21.21. Zmodyfikuj struktury danych dla blokad wielotrybowych oraz algorytmy dla operacji `blokuj_do_odczytu(X)`, `blokuj_do_zapisu(X)` oraz `odblokuj(X)`, tak aby było możliwe rozszerzanie i zawężanie blokad (*podpowiedź*: blokada musi sprawdzać identyfikator(y) transakcji, na której jest założona ta blokada, o ile takie istnieją).

- 21.22. Udowodnij, że ściśle blokowanie dwufazowe gwarantuje harmonogramy ściśle.
- 21.23. Udowodnij, że protokoły czekaj-kończ i zakończ-czekaj pozwalają uniknąć problemów zakleszczenia i zagłodzenia.
- 21.24. Udowodnij, że oczekiwanie ostrożne pozwala uniknąć zakleszczeń.
- 21.25. Zastosuj algorytm porządkowania na podstawie znaczników czasu względem harmonogramów z rysunków 21.8(b) oraz (c) i określ, czy algorytm ten pozwala na wykonanie tych harmonogramów.
- 21.26. Powtórz ćwiczenie 21.25, ale użyj metody porządkowania z wielowersyjnymi znacznikami czasu.
- 21.27. Dlaczego blokowanie dwufazowe nie jest używane jako metoda sterowania współbieżnego w przypadku indeksów takich jak B<sup>+</sup>-drzewa?
- 21.28. Macierz zgodności z rysunku 21.8 pokazuje, że blokady IS oraz IX są zgodne. Wyjaśnij, dlaczego.
- 21.29. Protokół MGL określa, że transakcja  $T$  może odblokować wierzchołek  $N$  tylko wówczas, gdy żaden z potomków wierzchołka  $N$  nie jest wciąż blokowany przez transakcję  $T$ . Pokaż, że bez takiego warunku protokół MGL byłby niepoprawny.

## Wybrane publikacje

Protokół blokowania dwufazowego oraz pojęcie blokad predykatowych zaproponowano w pozycji Eswarana i in. (1976). Bernstein i in. (1987), Gray i Reuter (1993) oraz Papadimitriou (1986) omawiają techniki sterowania współbieżnego i odtwarzania. Kumar (1996) omawia wydajność metod sterowania współbieżnego. Blokowanie omówiono w pozycji Graya i in. (1975), Liena i Weinbergera (1978), Kedema i Silbershatza (1980) oraz Kortha (1983). Zakleszczenia i grafy oczekiwania sformalizował Holt (1972), zaś schematy czekaj-kończ i zakończ-czekaj przedstawiono w pozycji Rosenkrantz i in. (1978). Oczekiwanie ostrożne omówiono w pracy Hsu i Zhanga (1992). Helal i in. (1993) porównuje różne podejścia do problemu blokowania.

Techniki sterowania współbieżnego oparte na znacznikach czasu omówiono w pozycji Bernsteina i Goodmana (1980) oraz Reeda (1983). Optymistyczne sterowanie współbieżne omawia pozycja Kunga i Robinsona (1981) oraz Bassiouniego (1988). Papadimitriou i Kanellakis (1979) oraz Bernstein i Goodman (1983) omawiają techniki wersji wielokrotnych. Porządkowanie z wielowersyjnymi znacznikami czasu zaproponowano w pozycjach Reeda (1978, 1983), zaś wielowersyjne blokowanie dwufazowe omówiono w pozycji Laia i Wilkinsona (1984). Metodę blokowania z wieloma poziomami ziarnistości zaproponowano w pozycji Graya i in. (1975), zaś efekty ziarnistości blokowania przeanalizowano w pracy Riesa i Stonebrakera (1977). Bhargava i Reidl (1988) prezentują podejście polegające na dynamicznym wybieraniu spośród wielu metod sterowania współbieżnego i odtwarzania. Metody sterowania współbieżnego dla indeksów przedstawiono w pracy Lehmana i Yao (1981) oraz Shasha'ego i Goodmana (1988). Studium wydajności różnych algorytmów sterowania współbieżnego wykorzystujących B<sup>+</sup>-drzewa zawarto w pracy Srinivasana i Careya (1991).

Anomalie, jakie może powodować podstawowa izolacja snapshotów, są opisane m.in. w pracach Feketego i in. (2004), Jorwekara i in. (2007) oraz Portsa i Grittnera (2012).

Modyfikowanie izolacji snapshotów w celu zapewnienia jej szeregowalności omówiono w pozycjach Cahilla i in. (2008), Feketego i in. (2005), Revilaka i in. (2011) oraz Portsa i Grittnera (2012).

Inne badania nad sterowaniem współbieżnym dotyczą między innymi sterowania współbieżnego opartego na semantyce (Badrinath i Ramamritham, 1992), modeli transakcji dla działań długotrwałych (Dayal i in., 1991) oraz wielopoziomowego zarządzania transakcjami (Hasse i Weikum, 1991).



## Techniki odtwarzania baz danych

W niniejszym rozdziale zostaną omówione pewne techniki, których można używać w celu odtwarzania baz danych po awariach. W podrozdziale 20.1.4 omówiono już różne przyczyny awarii, takie jak załamanie systemu i błędy transakcji. W podrozdziale 20.2 przedstawiono także wiele pojęć wykorzystywanych przez procesy odtwarzania, takich jak dziennik systemowy i punkty zatwierdzenia.

W tym rozdziale prezentujemy dodatkowe zagadnienia związane z protokołami odtwarzania baz danych i przegląd różnych algorytmów z tego obszaru. W podrozdziale 22.1 zostanie przedstawiony ogólny zarys typowych procedur odtwarzania oraz podział algorytmów odtwarzania na różne kategorie. Następnie omówimy kilka pojęć dotyczących odtwarzania, w tym rejestrowanie zapisów z wyprzedzeniem, aktualizacje bezpośrednie i przesłanianie oraz proces wycofywania (anulowania) efektów działania niepełnych lub zakończonych niepowodzeniem transakcji. W podrozdziale 22.2 zostaną przedstawione techniki odtwarzania bazujące na *aktualizacjach odroczonech*, znane również jako techniki NO-UNDO/REDO. W podrozdziale 22.3 zostaną omówione techniki bazujące na *aktualizacjach natychmiastowych*, gdzie dane można aktualizować na dysku w trakcie wykonywania transakcji. Zaliczamy do nich algorytmy UNDO/REDO oraz UNDO/NO-REDO. W podrozdziale 22.4 zostanie również omówiona technika znana jako przesłanianie lub stronicowanie z przesłanianiem, którą można zaklasyfikować do algorytmów klasy NO-UNDO/↪NO-REDO. Przykład praktycznego schematu odtwarzania w SZBD, noszący nazwę ARIES, zostanie przedstawiony w podrozdziale 22.5. W podrozdziale 22.6 omówimy pobieżnie kwestie odtwarzania w systemach wielu baz danych. Wreszcie w podrozdziale 22.7 zostaną omówione techniki odtwarzania po awariach katastroficznych. Podrozdział 22.8 zawiera podsumowanie rozdziału.

W prezentowanym materiale będziemy kłaść nacisk na koncepcyjne opisanie kilku różnych podejść do problemu odtwarzania. Opisy funkcji odtwarzania występujące w określonych systemach Czytelnik może znaleźć w pozycjach wymienionych w bibliografii, w internecie oraz w podręcznikach określonych systemów. Techniki odtwarzania są często blisko związane z mechanizmami sterowania współbieżnego. Określone techniki najlepiej jest stosować wraz z konkretnymi metodami sterowania współbieżnego. Poniższe omówienie stanowi próbę przedstawienia pojęć dotyczących odtwarzania niezależnie od jakichkolwiek mechanizmów współbieżnych.

## 22.1. Pojęcia związane z odtwarzaniem

### 22.1.1. Zarys problematyki odtwarzania i podział algorytmów odtwarzania na odrębne kategorie

Odtwarzanie po awariach transakcji zazwyczaj oznacza, że baza danych jest *przywracana* do ostatniego spójnego stanu, tuż sprzed wystąpienia awarii. W tym celu system musi przechowywać informacje dotyczące zmian związanych z elementami danych, wprowadzonych przez różne transakcje. Informacje te są zwykle przechowywane w **dzienniku systemowym** (ang. *system log*), zgodnie z opisem zawartym w podrozdziale 21.2.2. Typową strategię odtwarzania można nieformalnie opisać w sposób następujący:

- (1) Jeżeli wystąpi poważne uszkodzenie całego obszaru bazy danych z powodu awarii katastroficznej, na przykład awarii dysku, metoda odtwarzania przywraca starą kopię bazy danych, która została *zarchiwizowana* na nośniku kopii bezpieczeństwa (zwykle na taśmie lub innym przechowywanym w trybie offline nośniku pamięci masowej o dużej pojemności), i rekonstruuje bardziej aktualny stan bazy poprzez ponowne zastosowanie, czyli *ponowienie* (ang. *redoing*), operacji zatwierdzonych transakcji na podstawie *zarchiwizowanego* dziennika, aż do punktu wystąpienia awarii.
- (2) Kiedy baza danych nie zostanie uszkodzona fizycznie, ale znajdzie się w niespójnym stanie ze względu na wystąpienie niekatastroficznej awarii typu od 1. do 4. z podrozdziału 21.1.4, stosowana strategia polega na zidentyfikowaniu wszelkich zmian, które spowodowały niespójność w bazie. Przykładowo, transakcja, która zaktualizowała elementy bazy danych na dysku, ale nie została zatwierdzona, może wymagać wycofania zmian poprzez *odwołanie* (ang. *undoing*) pewnych operacji. Może również okazać się konieczne *ponowienie* (ang. *redo*) pewnych operacji w celu przywrócenia spójnego stanu bazy danych — np. w sytuacji, jeśli transakcja zatwierdziła niektóre operacje zapisu, ale nie została jeszcze wprowadzona na dysku. W przypadku awarii niekatastroficznych nie jest potrzebna pełna kopia bezpieczeństwa bazy. Zamiast tego do określania odpowiednich działań na potrzeby odtwarzania bazy wykorzystuje się wpisy zawarte w czynnym dzienniku systemowym.

Z koncepcyjnego punktu widzenia można wyróżnić dwie główne techniki odtwarzania po niekatastroficznych awariach transakcji: aktualizacje odroczone oraz aktualizacje natychmiastowe. Techniki **aktualizacji odroczonej** (ang. *deferred update*) nie powodują fizycznej aktualizacji bazy danych na dysku do momentu *po* zatwierdzeniu transakcji. Wtedy aktualizacje są rejestrowane w bazie danych. Przed osiągnięciem punktu zatwierdzenia wszystkie aktualizacje transakcji są rejestrowane w lokalnym obszarze roboczym transakcji (lub w buforach) utrzymywanym przez SZBD (jest to pamięć podręczna SZBD w pamięci głównej; patrz punkt 20.2.4). Przed zatwierdzeniem aktualizacje są najpierw trwale rejestrowane w dzienniku na dysku, a następnie, po zatwierdzeniu, zapisywane w bazie danych na podstawie zawartości buforów z pamięci głównej. Jeżeli transakcja zakończy się niepowodzeniem przed osiągnięciem punktu zatwierdzenia, nie zmienia bazy danych pod żadnym względem, więc operacja UNDO (odwołania) nie jest potrzebna. Może

okazać się konieczne wykonanie operacji REDO (ponowienia) działań operacji zatwierdzonej transakcji na podstawie informacji z dziennika, ponieważ ich efekt mógł zostać niezarejestrowany w bazie. Stąd aktualizacje odroczone określa się mianem **algorytmu NO-UNDO/REDO**. Technikę tę omówimy w podrozdziale 22.2.

W przypadku techniki **aktualizacji natychmiastowych** (ang. *immediate update*) baza danych *może być aktualizowana* przez pewne operacje transakcji *zanim* osiągnie swój punkt zatwierdzenia. Jednakże, operacje takie są zwykle rejestrowane w dzienniku *na dysku* poprzez zapis wymuszony *zanim* zostaną zastosowane względem bazy danych, co umożliwia odtwarzanie. Jeżeli transakcja ulegnie awarii po zarejestrowaniu pewnych zmian w bazie danych, ale przed osiągnięciem punktu zatwierdzenia, efekty jej operacji na bazie danych muszą zostać odwołane, to znaczy, transakcje te muszą zostać wycofane. W ogólnym przypadku aktualizacji natychmiastowych w czasie odtwarzania może być konieczne wykonanie zarówno operacji UNDO, jak i REDO. Taka technika, znana jako **algorytm UNDO/REDO**, wymaga obu rodzajów operacji i jest najczęściej stosowana w praktyce. Odmiana tego algorytmu, w której przypadku wszystkie aktualizacje są rejestrowane w bazie danych *przed* zatwierdzeniem transakcji, wymaga tylko operacji UNDO, więc określa się ją mianem **algorytmu UNDO/NO-REDO**. Techniki te zostaną omówione w podrozdziale 22.3.

Operacje UNDO i REDO muszą być **powtarzalne** (inaczej: idempotentne). Oznacza to, że ich wielokrotne wykonanie jest równoważne jednokrotnemu. W rzeczywistości cały proces odtwarzania powinien być powtarzalny, ponieważ jeśli system ulegnie awarii w trakcie odtwarzania, w następnej próbie odtworzenia bazy operacje UNDO i REDO mogą zostać uruchomione na niektórych operacjach zapisu elementu już wykonanych w pierwszej próbie odtworzenia bazy. Efekt rekonstrukcji po awarii systemu *w trakcie odtwarzania* powinien być taki sam jak w sytuacji, gdy *w czasie odtwarzania nie było awarii!*

## 22.1.2. Zapisywanie w pamięci podręcznej (buforowanie) bloków dyskowych

Proces odtwarzania jest często blisko związany z funkcjami systemu operacyjnego, w szczególności chodzi tu o buforowanie stron dyskowych w pamięci podręcznej SZBD przechowywanej w pamięci głównej. W typowej sytuacji jedna lub więcej stron dyskowych zawierających elementy danych, które podlegają aktualizacji, jest **buforowanych** (ang. *cached*) w pamięci głównej i aktualizowanych właśnie w pamięci przed zapisaniem na dysku. Za buforowanie stron dyskowych zwykle odpowiedzialny jest system operacyjny, jednak ze względu na istotność tego elementu z punktu widzenia wydajności procedur odtwarzania ich obsługą zajmuje się SZBD, który wywołuje niskopoziomowe podprogramy systemu operacyjnego (patrz punkt 20.2.4).

Ogólnie rzecz biorąc, wygodnie jest rozpatrywać odtwarzanie w kontekście stron dyskowych (bloków) bazy danych. Zazwyczaj zbiór buforów przechowywanych w pamięci, zwany **pamięcią podręczną SZBD** (ang. *DBMS cache*) znajduje się pod kontrolą SZBD i służy do przechowywania tych buforów. **Katalog** (ang. *directory*) pamięci podręcznej jest wykorzystywany do przechowywania informacji o tym, które elementy bazy danych znajdują się w buforach<sup>1</sup>. Może to być tabela zawierająca wpisy postaci <adres strony

---

<sup>1</sup> Przypomina to nieco pojęcie *tabel stronicowania* używanych przez system operacyjny.

dyskowej, lokalizacja bufora>. Kiedy SZBD ma wykonać pewne działania na elemencie danych, najpierw sprawdza katalog pamięci podręcznej w celu określenia, czy strona dyskowa zawierająca ten element znajduje się w pamięci podręcznej. Jeżeli tak nie jest, element musi zostać zlokalizowany na dysku i odpowiednie strony dyskowe muszą zostać skopiowane do pamięci podręcznej. Może okazać się konieczne **zastąpienie** (ang. *replace*), inaczej **opróżnienie** (ang. *flush*), niektórych buforów w celu zapewnienia wolnego miejsca dla nowego elementu (patrz punkt 20.2.4).

Wpisy w katalogu pamięci podręcznej SZBD obejmują dodatkowe informacje związane z zarządzaniem buforami. Z każdym buforem w pamięci podręcznej związany jest **bit modyfikacji** (ang. *dirty bit*), który może być uwzględniony we wpisie w katalogu, określając, czy bufor został zmodyfikowany. Kiedy strona jest wczytywana z dysku do bufora pamięci podręcznej, katalog pamięci podręcznej jest aktualizowany nowym adresem strony dyskowej i bit modyfikacji jest zerowany. Kiedy bufor ulegnie modyfikacji, bit modyfikacji odpowiedniego wpisu w katalogu zostaje ustawiony na jeden. W katalogu przechowywane są też inne informacje, np. identyfikatory transakcji, które zmodyfikowały bufor. Kiedy zawartość bufora jest zastępowana (opróżniana), najpierw musi zostać z powrotem zapisana w odpowiedniej stronie dyskowej, ale tylko wówczas, gdy *wartością bitu modyfikacji jest 1*.

Potrzebny jest również inny bit, określany mianem **bitu zajętości** (ang. *pin-unpin bit*). Strona w pamięci podręcznej jest **zajęta** (bit ma wartość 1), jeżeli w danym momencie nie może być zapisana z powrotem na dysku. Protokół odtwarzania może np. zablokować zapis na dysk niektórych stron bufora do czasu zatwierdzenia transakcji, która zmodyfikowała dany bufor.

W przypadku opróżniania zmodyfikowanego bufora i zapisywania go na dysku można zastosować dwie główne strategie. Pierwsza z nich to **aktualizacja bezpośrednia** (ang. *in-place updating*), która zapisuje bufor z powrotem *w tej samej oryginalnej lokalizacji na dysku*, co powoduje nadpisanie na dysku starych wartości wszelkich zmienionych elementów danych<sup>2</sup>. Stąd przechowywana jest pojedyncza kopia każdego bloku dyskowego bazy danych. Druga strategia, znana jako **przesłanianie** (ang. *shadowing*), polega na zapisywaniu zaktualizowanego bufora w innej lokalizacji na dysku, dzięki czemu można przechowywać wiele wersji elementów danych. W praktyce to podejście stosuje się rzadko.

Ogólnie rzecz biorąc, stara wartość elementu danych sprzed aktualizacji jest określana mianem **obrazu pierwotnego** (ang. *before image, BFIM*), zaś nową wartość po aktualizacji określa się mianem **obrazu końcowego** (ang. *after image, AFIM*). W przypadku przesłaniania na dysku mogą być przechowywane zarówno obrazy BFIM, jak AFIM, dzięki czemu nie jest bezwzględnie konieczne przechowywanie dziennika w celu umożliwiania odtwarzania. Odtwarzanie oparte na technice przesłaniania zostanie pokrótce omówione w podrozdziale 22.4.

---

<sup>2</sup> Aktualizacja bezpośrednia jest rozwiązaniem najczęściej stosowanym w praktyce.

### 22.1.3. Rejestrowanie zapisów z wyprzedzeniem, technika zabierania oraz wymuszania

W przypadku użycia metody aktualizacji bezpośredniej konieczne jest wykorzystanie dziennika w celach odtwarzania (patrz podrozdział 22.2.2). W takim przypadku mechanizm odtwarzania musi zapewniać, aby obraz pierwotny elementu danych został zarejestrowany w odpowiednim wpisie dziennika oraz aby wpis ten został zapisany na dysku zanim obraz pierwotny zostanie nadpisany obrazem końcowym w bazie danych na dysku. Taki proces ogólnie określa się mianem **rejestrowania zapisów z wyprzedzeniem** (ang. *write-ahead logging*). Jest on niezbędny do tego, aby można było odwołać operację, jeśli jest to potrzebne w trakcie odtwarzania bazy. Zanim opiszemy protokół rejestrowania zapisów z wyprzedzeniem, musimy wprowadzić rozróżnienie między dwoma rodzajami informacji zawartych we wpisach dziennika dla polecenia zapisu: informacje wymagane przez operację odwołania oraz informacje wymagane przez operację ponowienia. **Wpis dziennika typu ponowienia** (ang. *REDO-type log entry*) uwzględnia **nową wartość** (AFIM) elementu zapisanego przez operację, gdyż jest ona potrzebna w celu *ponowienia* efektów działań operacji na podstawie informacji zawartych w dzienniku (poprzez ustawienie wartości elementu w bazie danych na wartość AFIM). **Wpis dziennika typu odwołania** (ang. *UNDO-type log entry*) uwzględnia **starą wartość** (BFIM) elementu, gdyż jest ona potrzebna w celu *odwołania* efektów działań operacji na podstawie informacji zawartych w dzienniku (poprzez ustawienie wartości elementu w bazie danych na wartość BFIM). W przypadku algorytmu UNDO/REDO oba typy wpisów dziennika są łączone. Ponadto, kiedy możliwe jest wycofywanie kaskadowe (patrz punkt 22.1.5), wpisy odczytują element w dzienniku są traktowane jako wpisy typu odwołania.

Jak wcześniej wspomniano, pamięć podręczna SZBD zawiera w pamięci głównej buforowane bloki dyskowe bazy danych, do których należą nie tylko *bloki danych*, ale również *bloki indeksów* i *bloki dziennika*. Kiedy zapisywany jest rekord dziennika, jest on przechowywany w bieżącym bloku dziennika w pamięci podręcznej SZBD. Dziennik jest po prostu sekwencyjnym (umożliwiającym tylko dołączanie danych) plikiem dyskowym, a pamięć podręczna SZBD może zawierać w buforach w pamięci głównej kilka bloków dziennika (na przykład ostatnie  $n$ ), które są zapisywane na dysku. Kiedy następuje aktualizacja bloku danych — przechowywanego w pamięci podręcznej SZBD — odpowiedni rekord dziennika zostaje dopisany do ostatniego bloku dziennika w pamięci podręcznej. W przypadku wykorzystania podejścia zapisu z wyprzedzeniem, bufor (bloki) dziennika zawierające odpowiednie rekordy dziennika dla określonej aktualizacji bloku danych *najpierw muszą zostać zapisane na dysku*, zanim będzie można to samo zrobić z samym blokiem danych.

Do standardowej terminologii związanej z odtwarzaniem należą pojęcia **zabierania** (ang. *steal/ no-steal*) oraz **wymuszania** (ang. *force/no-force*), które określają, *kiedy* strona bazy danych z pamięci podręcznej może zostać zapisana na dysku:

- (1) Jeżeli strona pamięci podręcznej zaktualizowana przez transakcję *nie może* zostać zapisana na dysku przed zatwierdzeniem transakcji, jest to tak zwane **podejście bez zabierania** (ang. *no steal approach*). To, czy strona może zostać zapisana na dysku, określa bit zajętości. W przeciwnym wypadku, jeżeli protokół pozwala na zapisywanie zaktualizowanego bufora *przed* zatwierdzeniem transakcji, jest to **podejście z zabieraniem** (ang. *steal*). Jest ono wykorzystywane wówczas,

gdy menedżer pamięci podręcznej SZBD (bufora) potrzebuje ramki bufora dla innej transakcji i zastępuje istniejącą stronę, która została zaktualizowana, ale której transakcja nie została zatwierdzona. *Reguła unikania zabierania* polega na tym, że w trakcie odtwarzania nigdy nie jest konieczne odwoływanie, ponieważ aktualizacje wprowadzane przez transakcję są zapisywane na dysku dopiero po jej zatwierdzeniu.

- (2) Jeżeli wszystkie strony zaktualizowane przez transakcję są natychmiast zapisywane na dysku *przed* jej zatwierdzeniem, jest to tak zwane **podejście z wymuszeniem** (ang. *force approach*). W przeciwnym wypadku mówimy o **podejściu bez wymuszenia** (ang. *no-force approach*). *Reguła wymuszania* polega na tym, że w trakcie odtwarzania nigdy nie jest konieczne ponawianie, ponieważ wszystkie aktualizacje wprowadzane przez transakcję są zapisywane na dysku przed jej zatwierdzeniem.

Schemat odtwarzania z aktualizacjami odroczonymi z podrozdziału 22.2 wykorzystuje *podejście bez zabierania* (NO-UNDO), jednak typowe systemy bazodanowe stosują strategię *zabierania bez wymuszenia*. Jednak w typowych systemach baz danych stosowana jest strategia *z zabieraniem i bez wymuszania* (UNDO/REDO). *Korzyściami z podejściem zabierania* jest to, że unika się potrzeby używania bardzo dużej przestrzeni bufora w celu przechowywania wszystkich zaktualizowanych stron w pamięci. *Korzyściami strategii bez wymuszania* jest to, że zaktualizowana strona zatwierdzonej transakcji wciąż może znajdować się w buforze, kiedy inna transakcja chce ją zaktualizować, co eliminuje koszt operacji wejścia-wyjścia związanej z ponownym odczytem strony z dysku. Może to nieść ze sobą znaczne oszczędności pod względem liczby operacji wejścia-wyjścia, kiedy określona strona jest często aktualizowana przez wiele transakcji.

W celu dopuszczenia odtwarzania w przypadku aktualizacji bezpośrednich, odpowiednie wpisy wymagane dla celów odtwarzania muszą być trwale zarejestrowane na dysku, zanim zmiany będą mogły być zastosowane względem bazy danych. Przykładowo, weźmy pod uwagę następujący protokół **rejestrowania zapisów z wyprzedzeniem** (ang. *write-ahead logging, WAL*) w przypadku algorytmu odtwarzania wymagającego zarówno operacji UNDO, jak i REDO.

- (1) Obraz pierwotny elementu nie może zostać nadpisany swoim obrazem końcowym w bazie danych na dysku, dopóki nie zostaną zapisane na dysku wszystkie rekordy dziennika typu UNDO dla transakcji aktualizującej — aż do momentu bieżącego.
- (2) Operacja zatwierdzenia transakcji nie może zostać zakończona, dopóki nie zostaną zapisane na dysku wszystkie rekordy dziennika operacji typu REDO i UNDO dla danej transakcji.

W celu ułatwienia procesu odtwarzania, podsystem odtwarzania SZBD może być zmuszony do przechowywania wielu list związanych z transakcjami przetwarzanymi w systemie. Chodzi tu o listę **transakcji aktywnych**, które zostały uruchomione, ale nie zostały zatwierdzone, oraz o listy wszystkich **transakcji zatwierdzonych i anulowanych** od czasu wykonania ostatniego punktu kontrolnego (patrz kolejny punkt). Przechowywanie takich list usprawnia proces odtwarzania.



## 22.1.4. Punkty kontrolne w dzienniku systemowym oraz tworzenie przybliżonych punktów kontrolnych

Kolejnym rodzajem wpisów w dzienniku są **punkty kontrolne** (ang. *checkpoint*)<sup>3</sup>. Rekord postaci [punkt\_kontrolny, lista aktywnych transakcji] jest okresowo zapisywany w dzienniku w momencie, gdy system zapisuje w bazie danych na dysku wszystkie buforzy SZBD, które uległy modyfikacji. W konsekwencji wszystkie transakcje, które w dzienniku posiadają swoje wpisy [zatwierdź, T] przed wpisem [punkt\_kontrolny], nie muszą podlegać *ponowieniu* swoich operacji zapisu w przypadku awarii systemu, gdyż wszystkie ich aktualizacje są rejestrowane w bazie danych na dysku w czasie wykonywania punktu kontrolnego. W ramach tworzenia punktu kontrolnego do rekordu takiego punktu dołączana jest lista identyfikatorów transakcji aktywnych w danym momencie. Dzięki temu w trakcie odtwarzania bazy takie transakcje można łatwo zidentyfikować.

Menedżer odtwarzania SZBD musi podjąć decyzję, w jakich odstępach czasu będzie wykonywany punkt kontrolny. Odstępy te można mierzyć w jednostkach czasu — na przykład co  $m$  minut — lub pod względem liczby  $t$  zatwierdzonych transakcji od czasu wykonania ostatniego punktu kontrolnego, gdzie wartości  $m$  lub  $t$  są parametrami systemu. Wykonanie punktu kontrolnego polega na przeprowadzeniu następujących działań:

- (1) Tymczasowe zawieszenie wykonywania transakcji.
- (2) Wymuszony zapis na dysku wszystkich buforów pamięci głównej, które zostały zmodyfikowane.
- (3) Zapisanie rekordu [punkt\_kontrolny] w dzienniku i wymuszenie zapisu dziennika na dysku.
- (4) Wznowienie wykonywania transakcji.

W konsekwencji wykonania etapu 2. rekord punktu kontrolnego w dzienniku może również zawierać dodatkowe informacje, takie jak lista identyfikatorów transakcji aktywnych oraz lokalizacje (adresy) pierwszego i ostatniego rekordu w dzienniku dla każdej transakcji aktywnej. Może to ułatwić proces odwoływania operacji transakcji w przypadku, gdy okaże się, że transakcja musi zostać wycofana.

Czas potrzebny do wymuszonego zapisania wszystkich zmodyfikowanych buforów może opóźnić wykonywanie transakcji ze względu na etap 1. W praktyce jest to nieakceptowalne. W celu zredukowania tego opóźnienia często stosuje się technikę określaną mianem tworzenia **przybliżonych punktów kontrolnych** (ang. *fuzzy checkpointing*). W przypadku tej techniki system może wznowić przetwarzanie transakcji po zapisaniu rekordu [punkt\_kontrolny] w dzienniku bez konieczności oczekiwania na zakończenie etapu 2. Jednakże dopóki etap ten nie zostanie zakończony, poprzedni rekord [punkt\_kontrolny] powinien być traktowany jako obowiązujący. W tym celu system przechowuje wskaźnik na ten punkt kontrolny, który wskazuje na poprzedni rekord [punkt\_kontrolny] w dzienniku. Po zakończeniu etapu 2. wskaźnik ten jest zmieniany i wskazuje na nowy punkt kontrolny w dzienniku.

---

<sup>3</sup> Pojęcie *punktu kontrolnego* było w pewnych systemach, takich jak DB2, używane w celu opisania bardziej restrykcyjnych sytuacji. Używano go również w literaturze w celu opisania całkowicie odmiennych pojęć.



## 22.1.5. Wycofywanie transakcji i wycofywanie kaskadowe

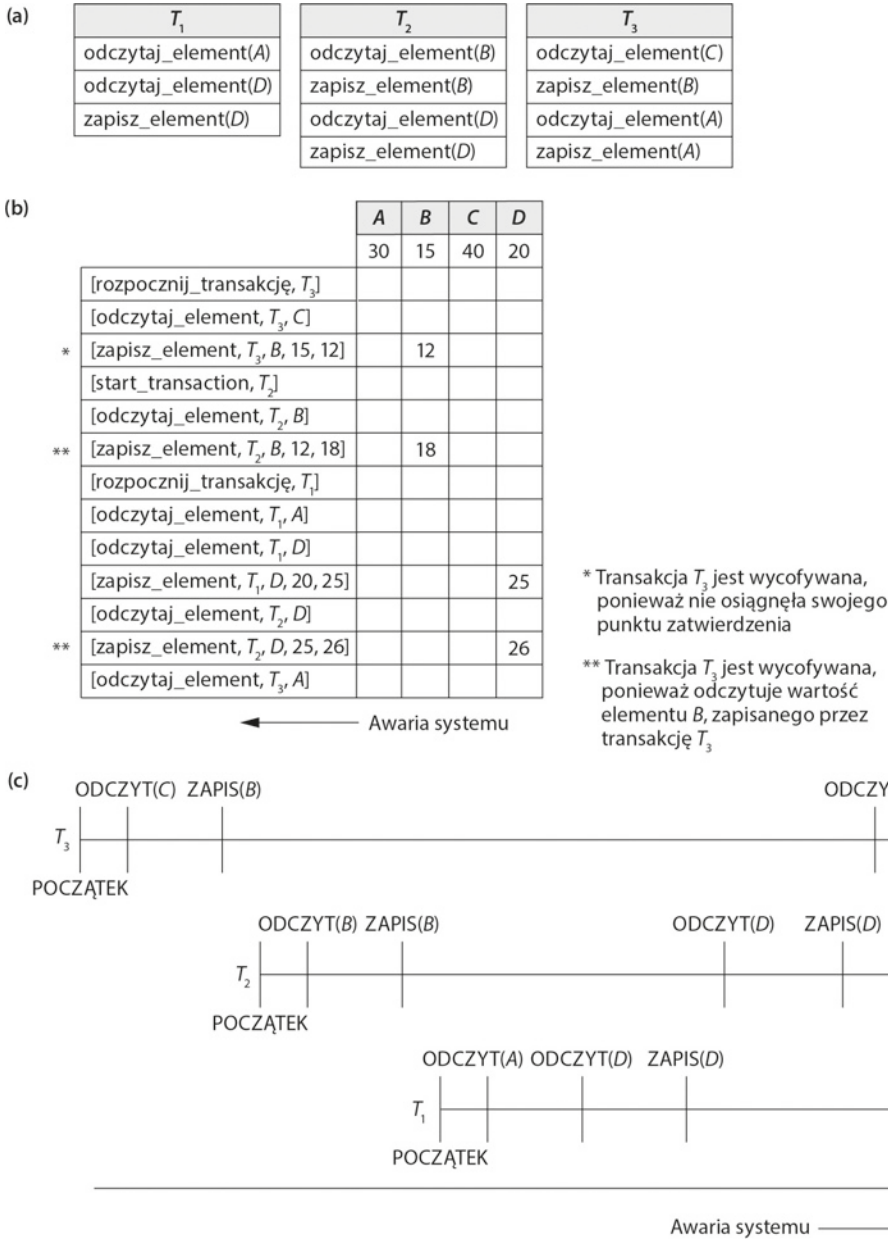
Jeżeli transakcja ulegnie z jakiegoś powodu awarii po zaktualizowaniu bazy danych, ale przed zatwierdzeniem, może być konieczne jej **wycofanie** (ang. *rollback*). Jeżeli wartości jakichś elementów danych zostały przez transakcję zmienione i zapisane w bazie danych na dysku, muszą zostać odtworzone do swoich poprzednich wartości (BFIM). W celu odtworzenia starych wartości elementów danych wykorzystuje się wpisy dziennika typu UNDO.

Jeżeli transakcja  $T$  zostanie wycofana, każda transakcja  $S$ , która w czasie jej wykonywania odczytała wartość pewnego elementu danych  $X$  zapisywanego przez  $T$ , również musi zostać wycofana. Podobnie, kiedy transakcja  $S$  jest wycofywana, każda transakcja  $R$ , która odczytała wartość pewnego elementu danych  $Y$  zapisywanego przez  $S$ , również musi zostać wycofana. Takie zjawisko określa się mianem **wycofywania kaskadowego** (ang. *cascading rollback*) i może ono wystąpić w sytuacji, gdy protokół odtwarzania zapewnia harmonogramy *odtworzalne*, ale nie zapewnia harmonogramów *ściśłych* lub *bezkaskadowych* (patrz podrozdział 20.4.2). Co zrozumiałe, wycofywanie kaskadowe może być dość złożone i czasochłonne. Z tego powodu niemal wszystkie mechanizmy odtwarzania projektuje się tak, aby wycofywanie kaskadowe *nigdy nie było wymagane*.

Na rysunku 22.1 przedstawiono przykład, kiedy wycofywanie kaskadowe jest konieczne. Operacje odczytu i zapisu trzech oddzielnych transakcji przedstawiono na rysunku 22.1(a). Rysunek 22.1(b) przedstawia dziennik systemowy w momencie awarii systemu dla konkretnego harmonogramu wykonania tych transakcji. Wartości elementów danych  $A$ ,  $B$ ,  $C$  i  $D$ , używane przez transakcje, przedstawiono na prawo od wpisów dziennika. Zakładamy, że oryginalne wartości elementów, zawarte w pierwszym wierszu, to  $A = 30$ ,  $B = 15$ ,  $C = 40$  oraz  $D = 20$ . W momencie awarii systemu transakcja  $T_3$  nie osiągnęła swojego końca i musi zostać wycofana. Operacje zapisu tej transakcji, oznaczone na rysunku 22.1(b) znakiem \*, są tymi operacjami, które w czasie wycofywania transakcji należy odwołać. Na rysunku 22.1(c) w graficznej postaci przedstawiono operacje różnych transakcji na osi czasu.

Teraz musimy sprawdzić występowanie wycofywania kaskadowego. Na podstawie rysunku 22.1(c) możemy stwierdzić, że transakcja  $T_2$  odczytuje wartość elementu  $B$ , którą zapisała transakcja  $T_3$ ; można to również określić badając wpisy dziennika. Ze względu na fakt, że transakcja  $T_3$  jest wycofywana, transakcja  $T_2$  również musi zostać wycofana. Operacje zapisu transakcji  $T_2$ , oznaczone w dzienniku symbolem \*\*, są tymi, które należy odwołać. Warto zauważyć, że w czasie wycofywania transakcji konieczne jest odwołanie tylko operacji zapisz\_element. Operacje odczytaj\_element są rejestrowane w dzienniku tylko w celu określenia, czy zachodzi potrzeba wycofania kaskadowego innych transakcji.

W praktyce kaskadowe wycofywanie transakcji *nigdy* nie jest wymagane, ponieważ praktyczne metody odtwarzania gwarantują harmonogramy *bezkaskadowe* lub *ściśle*. Stąd nie istnieje również potrzeba rejestrowania w dzienniku jakichkolwiek operacji odczytaj\_element, ponieważ są one potrzebne tylko w celu określenia konieczności wycofania kaskadowego.



RYSUNEK 22.1. Ilustracja wycofania kaskadowego (procesu, który nigdy nie występuje w przypadku harmonogramów ścisłych lub bezkaskadowych). (a) Operacje odczytu i zapisu trzech transakcji. (b) Dziennik systemowy w momencie wystąpienia awarii. (c) Stan operacji przed wystąpieniem awarii

### 22.1.6. Działania transakcji niewpływające na bazy danych

Transakcja zwykle powoduje działania, które *nie wpływają* na bazę danych — np. generuje i wyświetla komunikaty lub raporty na podstawie pobranych z bazy informacji. Jeśli transakcja ulegnie awarii przed ukończeniem pracy, możliwe, że użytkownik nie powinien zobaczyć raportów, ponieważ transakcja nie została zakończona. Jeżeli wygenerowane zostaną błędne raporty, w ramach odtwarzania bazy trzeba poinformować użytkownika o tym, że dane są nieprawidłowe (ponieważ użytkownik może na podstawie raportu wykonać operacje wpływające na bazę). Dlatego raporty należy generować dopiero *po dojsściu transakcji do punktu zatwierdzenia*. Standardowa metoda radzenia sobie z opisywanymi działaniami polega na wywoływaniu poleceń, które generują raporty, ale wykonywaniu ich w zadaniach wsadowych, przetwarzanych dopiero po dojsściu transakcji do punktu zatwierdzenia. Po awarii transakcji zadania wsadowe są anulowane.

## 22.2. Techniki odtwarzania NO-UNDO/REDO oparte na aktualizacjach odroczonech

Idea działania technik aktualizacji odroczonech polega na odłożeniu wszelkich bieżących aktualizacji bazy danych do momentu udanego zakończenia działania transakcji, kiedy ta osiągnie swój punkt zatwierdzenia<sup>4</sup>.

W czasie wykonywania transakcji aktualizacje są rejestrowane tylko w dzienniku oraz w buforach pamięci podręcznej. Kiedy transakcja osiągnie swój punkt zatwierdzenia, a dziennik zostanie poddany zapisowi wymuszonemu na dysku, aktualizacje są rejestrowane w bazie danych. Jeżeli transakcja ulegnie awarii przed osiągnięciem punktu zatwierdzenia, nie ma potrzeby odwoływania jakichkolwiek operacji, ponieważ transakcja nie miała wpływu na bazę danych zapisaną na dysku. Dlatego w dzienniku potrzebne są tylko **wpisy typu ponowienia**, obejmujące **nową wartość** (AFIM) elementu utworzoną przez operację zapisu. **Wpisy typu odwołania** nie są konieczne, ponieważ w trakcie odtwarzania nie będzie trzeba odwoływać operacji. Choć może to upraszczać proces odtwarzania, nie da się takiego rozwiązania stosować w praktyce, chyba że transakcje są krótkie i każda z nich zmienia niewiele elementów. W przypadku innych rodzajów transakcji istnieje potencjalne niebezpieczeństwo wyczerpania wolnej przestrzeni bufora, ponieważ zmiany wynikające z wykonywania transakcji muszą być przechowywane w buforach pamięci podręcznej do momentu osiągnięcia punktu zatwierdzenia. Dlatego wiele buforów pamięci podręcznej będzie *zajętych* i nie będzie można ich zastąpić.

Typowy protokół aktualizacji odroczonech można określić następująco:

- (1) Transakcja nie może zmienić stanu bazy danych na dysku, dopóki nie osiągnie swojego punktu zatwierdzenia. Dlatego wszystkie bufony zmodyfikowane przez transakcję muszą być zajęte do czasu zatwierdzenia transakcji (odpowiada to *strategii bez zabierania*).

---

<sup>4</sup> Stąd aktualizacje odroczone można, ogólnie rzecz biorąc, określić mianem *podejścia bez zabierania*.

- (2) Transakcja nie osiąga swojego punktu zatwierdzenia, dopóki wszystkie jej operacje aktualizacji nie zostaną zarejestrowane w dzienniku *oraz* bufor dziennika nie zostanie w wymuszony sposób zapisany na dysku.

Warto zauważyć, że punkt 2. takiego protokołu stanowi powtórzenie protokołu rejestrowania zapisów z wyprzedzeniem. Ze względu na fakt, że baza danych nigdy nie jest aktualizowana na dysku, dopóki transakcja nie zostanie zatwierdzona, nigdy nie ma potrzeby wykonywania operacji odwołania. Operacje ponawiania są konieczne w przypadku, gdy system ulegnie awarii po tym, jak transakcja zostanie zatwierdzona, ale zanim wszystkie zmiany zostaną zarejestrowane w bazie danych na dysku. W takim przypadku operacje transakcji są powtarzane na podstawie wpisów dziennika.

Zazwyczaj metoda odtwarzania po awarii ma bliski związek z metodą sterowania współbieżnego używaną w systemie wieloużytkownikowym. Wyobraź sobie system, w którym sterowanie współbieżne korzysta z blokowania dwufazowego, dlatego blokady na zapisywanych elementach są utrzymywane *do czasu dotarcia transakcji do punktu zatwierdzenia*. Następnie blokady można zwolnić. To gwarantuje powstawanie harmonogramów ścisłych i szeregowalnych. Jeśli w dzienniku zapisywane są wpisy [punkt kontrolny], można zastosować przedstawiony dalej algorytm odtwarzania RDU\_M (ang. *Recovery using Deferred Update in a Single-user environment* — odtwarzanie przy użyciu aktualizacji odroczonych w środowisku wieloużytkownikowym).

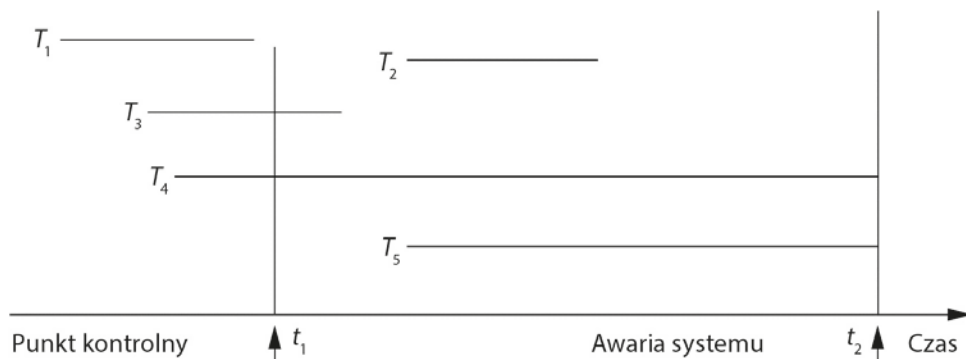
#### **PROCEDURA RDU\_M (algorytm NO-UNDO/REDO z punktami kontrolnymi).**

System przechowuje dwie listy transakcji: listę transakcji  $T$  zatwierdzonych od momentu utworzenia ostatniego punktu kontrolnego (**lista zatwierdzonych**) oraz listę aktywnych transakcji  $T'$  (**lista aktywnych**). Ponawiamy wszystkie operacje zapisu zatwierdzonych transakcji z dziennika w *kolejności, w jakiej zostały w nim zapisane*. Aktywne transakcje, które nie zostały zatwierdzone, są anulowane i trzeba ponownie je wykonać.

Oto definicja procedury PONÓW.

**Procedura** PONÓW(OP\_ZAPISU). Ponowienie operacji zapisz\_element o nazwie OP\_ZAPISU polega na zbadaniu jej wpisu w dzienniku [zapisz\_element,  $T$ ,  $X$ , nowa\_wartość] i ustawieniu elementu  $X$  w bazie danych na wartość nowa\_wartość, która jest obrazem końcowym (AFIM).

Na rysunku 22.2 przedstawiono oś czasu z możliwym harmonogramem wykonywanych transakcji. W momencie tworzenia punktu kontrolnego w czasie  $t_1$  transakcja  $T_1$  jest już zatwierdzona, natomiast transakcje  $T_3$  i  $T_4$  — jeszcze nie. Przed awarią systemu w czasie  $t_2$  zatwierdzone są transakcje  $T_3$  i  $T_2$ , ale nie  $T_4$  i  $T_5$ . Zgodnie z metodą RDU\_M nie trzeba ponawiać operacji zapisz\_element transakcji  $T_1$  ani innych transakcji zatwierdzonych przed wykonaniem ostatniego punktu kontrolnego  $t_1$ . Trzeba jednak ponowić operacje zapisz\_element transakcji  $T_2$  i  $T_3$ , ponieważ obie dotarły do punktu zatwierdzenia po zapisaniu ostatniego punktu kontrolnego. Warto przypomnieć, że przed zatwierdzeniem transakcji następuje wymuszony zapis dziennika. Transakcje  $T_4$  i  $T_5$  są ignorowane. Zostają anulowane (lub wycofane), ponieważ zgodnie z protokołem odraczanej aktualizacji (podejście bez zabierania) żadna z ich operacji zapisz\_element nie została zarejestrowana w bazie na dysku.



RYСУNEK 22.2. Przykładowa oś czasu odtwarzania ilustrująca wpływ tworzenia punktów kontrolnych

Aby *poprawić wydajność* algorytmu odtwarzania NO-UNDO/REDO, wystarczy zauważyć, że jeśli element danych  $X$  został od momentu utworzenia ostatniego punktu kontrolnego zaktualizowany przez zatwierdzone transakcje więcej niż raz (o czym informują wpisy z dziennika), w trakcie odtwarzania bazy wystarczy ponowić *ostatnią aktualizację*  $X$  z dziennika, ponieważ inne aktualizacje zostaną zastąpione tym ostatnim ponowieniem. W takim scenariuszu zaczynamy *od końca dziennika*, a gdy element zostanie ponownie zapisany, należy go dodać do listy takich elementów. Przed ponowieniem zapisu elementu należy sprawdzić listę. Jeśli element się na niej znajduje, jego zapis nie jest ponawiany, ponieważ odtworzono już najnowszą wartość.

Jeśli transakcja z jakiegoś powodu zostanie wycofana (np. przez metodę wykrywania zakleszczenia), system po prostu uruchomi ją ponownie, ponieważ nie zmodyfikowała bazy na dysku. Wadą opisaną tu metody jest ograniczenie współbieżnego wykonywania transakcji, ponieważ *wszystkie elementy z blokadami zapisu pozostają zablokowane do czasu dotarcia transakcji do punktu zatwierdzenia*. Ponadto technika ta może wymagać zbyt dużego bufora, aby przechowywać wszystkie zaktualizowane elementy do czasu zatwierdzenia transakcji. Główną zaletą tej metody jest to, że operacji transakcji *nigdy nie trzeba wycofywać*. Wynika to z dwóch powodów:

- (1) Transakcja nie wprowadza żadnych zmian w bazie na dysku do czasu dotarcia do punktu zatwierdzenia (czyli do momentu udanego zakończenia pracy). Dlatego transakcja nigdy nie jest wycofywana z powodu awarii w trakcie wykonywania.
- (2) Transakcja nigdy nie wczytuje wartości elementu zapisanej przez niezatwierdzoną transakcję, ponieważ elementy pozostają zablokowane do czasu dotarcia transakcji do punktu zatwierdzenia. Dlatego nie nastąpi wycofywanie kaskadowe.

Na rysunku 22.3 pokazano przykład odtwarzania bazy w systemie z wieloma użytkownikami, gdzie stosowana jest opisana tu metoda odtwarzania bazy i sterowania współbieżnego.

(a)	$T_1$	$T_2$	$T_3$	$T_4$
	odczytaj_element(A)	odczytaj_element(B)	odczytaj_element(A)	odczytaj_element(B)
	odczytaj_element(D)	zapisz_element(B)	zapisz_element(A)	zapisz_element(B)
	zapisz_element(D)	odczytaj_element(D)	odczytaj_element(C)	odczytaj_element(A)
		zapisz_element(D)	zapisz_element(C)	zapisz_element(A)

(b)	[rozpocznij_transakcję, $T_1$ ]
	[zapisz_element, $T_1$ , D, 20]
	[zatwierdź, $T_1$ ]
	[punkt kontrolny]
	[rozpocznij_transakcję, $T_4$ ]
	[zapisz_element, $T_4$ , B, 15]
	[zapisz_element, $T_4$ , A, 20]
	[zatwierdź, $T_4$ ]
	[rozpocznij_transakcję, $T_2$ ]
	[zapisz_element, $T_2$ , B, 12]
	[rozpocznij_transakcję, $T_3$ ]
	[zapisz_element, $T_3$ , A, 30]
	[zapisz_element, $T_2$ , D, 25]

← Awaria systemu

Transakcje  $T_2$  i  $T_3$  są ignorowane, ponieważ nie osiągnęły swojego punktu zatwierdzenia

Transakcja  $T_4$  jest ponawiana, ponieważ jej punkt zatwierdzenia znajduje się po ostatnim punkcie kontrolnym systemu

RYSUNEK 22.3. Przykład odtwarzania przy użyciu aktualizacji odroczonej z transakcjami współbieżnymi. (a) Operacje odczytu i zapisu czterech transakcji. (b) Dziennik systemu w momencie wystąpienia awarii

## 22.3. Techniki odtwarzania oparte na aktualizacjach natychmiastowych

W przypadku takich technik, kiedy transakcja wykonuje polecenie aktualizacji, baza danych może zostać zaktualizowana *natychmiast*, bez potrzeby oczekiwania na osiągnięcie przez transakcję punktu zatwierdzenia. Warto zauważyć, że *nie jest wymagane*, by każda aktualizacja była natychmiast zapisywana na dysku; jest jedynie możliwe, że niektóre aktualizacje zostaną wprowadzone na dysk *przed zatwierdzeniem transakcji*.

Należy przedsięwziąć pewne działania wstępne w celu umożliwienia *odwoływania* efektów operacji aktualizacji, które zastosowano do bazy danych w *transakcjach zakończonych niepowodzeniem*. Zapewnia się to, wycofując transakcję i odwołując efekty działania jej operacji zapisz\_element. Dlatego w dzienniku przechowywane muszą być **wpisy dziennika typu UNDO**, obejmujące **wartość pierwotną** (BFIM) elementu. Ponieważ w trakcie odtwarzania bazy konieczne może być odwołanie operacji, te metody działają zgodnie

z **podejściem z zabieraniem**, gdy decydują, kiedy zaktualizowane bufor y z pamięci głównej mogą zostać ponownie zapisane na dysku (patrz punkt 22.1.3).

Z teoretycznego punktu widzenia można wyróżnić dwie główne kategorie algorytmów aktualizacji natychmiastowych.

- (1) Jeżeli technika odtwarzania zapewnia, że wszystkie aktualizacje transakcji są rejestrowane w bazie danych na dysku *przed jej zatwierdzeniem*, nigdy nie ma potrzeby wykonywania operacji REDO dla zatwierdzonych transakcji. Jest to tak zwany **algorytm odtwarzania UNDO/NO-REDO** (ang. *UNDO/NO-REDO recovery algorithm*). W tej metodzie wszystkie aktualizacje transakcji muszą zostać zapisane na dysku *przed zatwierdzeniem transakcji*, dlatego operacja REDO nigdy nie jest potrzebna. Zatem trzeba tu zastosować **strategię z zabieraniem i wymuszaniem** w trakcie decydowania o tym, kiedy zaktualizowane bufor y z pamięci głównej ponownie zapisać na dysku (patrz punkt 22.1.3).
- (2) Jeżeli transakcja może zostać zatwierdzona zanim wszystkie jej aktualizacje zostaną zapisane w bazie danych, otrzymujemy najbardziej ogólny przypadek, znany jako **algorytm odtwarzania UNDO/REDO** (ang. *UNDO/REDO recovery algorithm*). W tej sytuacji stosowana jest strategia **z zabieraniem i bez wymuszania** (patrz punkt 22.1.3). Jest to najbardziej skomplikowana, ale najczęściej stosowana w praktyce technika. Poniżej zostaną omówione dwa przykłady algorytmów UNDO/REDO oraz jako ćwiczenie dla Czytelnika pozostawimy opracowanie odmiany algorytmu UNDO/NO-REDO. W podrozdziale 22.5 zostanie opisane bardziej praktyczne podejście, znane jako technika odtwarzania ARIES.

Kiedy dozwolone jest wykonywanie współbieżne, proces odtwarzania zależy od protokołów używanych w zakresie sterowania współbieżnego. Procedura RIU\_M (*Recovery using Immediate Update in a Multiuser environment*, odtwarzanie przy użyciu aktualizacji natychmiastowych w środowisku wieloużytkownikowym) stanowi zarys algorytmu odtwarzania dla transakcji współbieżnych z aktualizacjami natychmiastowymi (odtw arzanie UNDO/REDO). Zakładamy, że dziennik zawiera punkty kontrolne oraz że protokół sterowania współbieżnego daje *harmonogramy ścisłe* — na przykład tak, jak ma to miejsce w przypadku protokołu ścisłego blokowania dwufazowego. Należy przypomnieć, że harmonogram ścisły nie pozwala, aby transakcja odczytywała lub zapisywała element, jeżeli nie została zatwierdzona transakcja, która zmieniła jego wartość. Jednakże w przypadku ścisłego blokowania dwufazowego mogą występować zakleszczenia, co wymaga operacji anulowania i odwoływania transakcji. W przypadku harmonogramu ścisłego odwołanie operacji wymaga zmiany wartości elementu na pierwotną wartość (BFIM).

#### **Procedura RIU\_M (UNDO/REDO z punktami kontrolnymi)**

- (1) Używamy dwóch list transakcji przechowywanych przez system: transakcji zatwierdzonych od momentu utworzenia ostatniego punktu kontrolnego oraz transakcji aktywnych.
- (2) Odwołujemy wszelkie operacje zapisz\_element transakcji *aktywnych* (niezatwierdzonych), wykorzystując opisaną powyżej procedurę ODWOŁAJ. Operacje powinny zostać odwołane w kolejności odwrotnej do tej, w jakiej zostały zapisane w dzienniku.



- (3) Ponawiamy wszelkie operacje zapisz\_element transakcji *zatwierdzonych* na podstawie informacji z dziennika w kolejności, w jakiej zostały w nim zapisane. Stosujemy tu zdefiniowaną wcześniej procedurę ponawiania.

Procedura odwoływania jest zdefiniowana w następujący sposób:

**Procedura ODWOŁAJ(OP\_ZAPISU).** Odwołanie operacji zapisz\_element o nazwie OP\_ZAPISU polega na zbadaniu jej wpisu w dzienniku [zapisz\_element, T, X, stara\_wartość, nowa\_wartość] i ustawieniu elementu X w bazie danych na wartość stara\_wartość, która jest obrazem pierwotnym (BFIM). Odwołanie wielu operacji zapisz\_element związanych z jedną lub wieloma transakcjami na podstawie dziennika musi przebiegać w *odwrotnej kolejności* w stosunku do tej, w jakiej zapisywano je w dzienniku.

Jak stwierdzono w omówieniu procedury NO-UNDO/REDO, etap 3. można przeprowadzić w wydajniejszy sposób, rozpoczynając *od końca dziennika* i ponawiając *tylko ostate aktualizacje każdego elementu X*. Kiedy element podlega ponowieniu, jest dodawany do odpowiedniej listy i w przyszłości jest pomijany. Podobną procedurę można opracować w celu zwiększenia wydajności działania etapu 2. Dzięki temu odwołanie zmian elementu odbywa się w momencie odtwarzania bazy najwyżej raz. W tym scenariuszu najpierw wykonywana jest najwcześniejsza operacja odwołania, co wymaga przejrzania dziennika, zaczynając od jego początku. Po odwołaniu zmian elementu zostaje on dodany do listy przetworzonych elementów i dalsze zmiany nie są odwoływane.

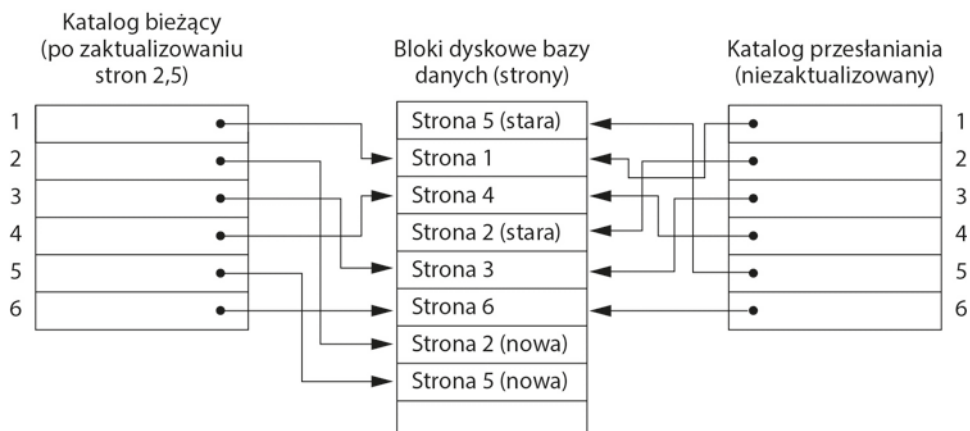
## 22.4. Stronicowanie z przesłanianiem

Taki schemat odtwarzania nie wymaga użycia dziennika w systemie jednoużytkownikowym. W przypadku środowiska wieloużytkownikowego dziennik może być potrzebny dla celów metody sterowania współbieżnego. Stronicowanie z przesłanianiem traktuje bazę danych jako zbiór stron dyskowych (bloków dyskowych) o stałym rozmiarze — powiedzmy, że takich stron jest  $n$ . Konstruowany jest **katalog**<sup>5</sup> o  $n$  wpisach, w którym  $i$ -ty wpis wskazuje na  $i$ -tą stronę dyskową. Katalog jest przechowywany w pamięci głównej (jeżeli nie jest zbyt duży). Trafiają do niego wszystkie referencje — dotyczące odczytów i zapisów — do stron bazy danych na dysku. Kiedy transakcja rozpoczyna wykonywanie, **katalog bieżący** (ang. *current directory*), którego wpisy wskazują na bieżące strony bazy danych na dysku, jest kopiowany do **katalogu przesłaniania** (ang. *shadow directory*). Katalog przesłaniania jest następnie zapisywany na dysku, zaś katalog bieżący jest używany przez transakcję.

W czasie wykonywania transakcji katalog przesłaniania *nigdy* nie podlega modyfikacjom. Kiedy jest wykonywana operacja zapisz\_element, tworzona jest nowa kopia zmodyfikowanej strony bazy danych, ale stara kopia tej strony *nie jest nadpisywana*. Zamiast tego nowa strona jest zapisywana w innym miejscu, zamiast pewnego nieużywanego bloku dyskowego. Wpis katalogu bieżącego zostaje zmodyfikowany tak, aby wskazywał na nowy blok dyskowy, natomiast katalog przesłaniania nie jest modyfikowany i wciąż wska-

<sup>5</sup> Katalog przypomina **tabelę stronicowania** przechowywaną przez system operacyjny dla każdego procesu.

zuje na stary, niezmodyfikowany blok dyskowy. Na rysunku 22.4 przedstawiono ilustrację katalogu bieżącego i przesłaniania. W przypadku stron aktualizowanych przez transakcje przechowywane są obie wersje. Do starej wersji odwołuje się katalog przesłaniania, zaś do nowej wersji — katalog bieżący.



RYSUNEK 22.4. Przykład stronicowania z przesłanianiem

W celu odtworzenia po awarii, która wystąpiła w czasie wykonywania transakcji, wystarczy zwolnić zmodyfikowane strony bazy danych i usunąć katalog bieżący. Stan bazy danych sprzed wykonania transakcji jest dostępny poprzez katalog przesłaniania i stan ten zostaje odtworzony dzięki przywróceniu tego katalogu. Baza danych powraca więc do swojego stanu sprzed rozpoczęcia transakcji, która była wykonywana w momencie wystąpienia awarii, i wszelkie zmodyfikowane strony zostają usunięte. Zatwierdzenie transakcji odpowiada usunięciu poprzedniego katalogu przesłaniania. Ze względu na fakt, że odtwarzanie nie uwzględnia ani odwoływania, ani ponawiania jakichkolwiek operacji na elementach, technikę tę można zaklasyfikować jako algorytm odtwarzania NO-UNDO/NO-REDO.

W środowisku wieloużytkownikowym z transakcjami współbieżnymi w ramach techniki stronicowania z przesłanianiem muszą zostać uwzględnione dzienniki i punkty kontrolne. Jedną z wad stronicowania z przesłanianiem jest to, że zaktualizowane strony bazy danych zmieniają swoją lokalizację na dysku. Utrudnia to zachowanie obok siebie na dysku powiązanych stron bazy danych bez wykorzystania skomplikowanych strategii zarządzania przestrzenią. Ponadto, jeżeli katalog jest obszerny, narzut związany z zapisywaniem katalogów przesłaniania na dysku w czasie zatwierdzenia transakcji może być znaczny. Kolejną komplikację stanowi kwestia **czyszczenia pamięci** (ang. *garbage collection*) po zatwierdzeniu transakcji. Stare strony, do których odwołuje się katalog przesłaniania, muszą zostać zwolnione i dodane do listy stron wolnych. Strony te nie są dłużej potrzebne po zatwierdzeniu transakcji. Kolejną problematyczną kwestią jest to, że operacja przełączenia między katalogiem bieżącym a katalogiem przesłaniania musi zostać zaimplementowana jako operacja niepodzielna.

## 22.5. Algorytm odtwarzania ARIES

Poniżej zostanie opisany algorytm ARIES jako przykład algorytmu odtwarzania wykorzystywanego w systemach bazodanowych. Jest on używany w wielu produktach firmy IBM z obszaru relacyjnych baz danych. System ARIES wykorzystuje podejście z zabieraniem i bez wymuszania w zakresie operacji zapisu i bazuje na trzech koncepcjach: rejestrowania zapisów z wyprzedzeniem, powtarzania historii w trakcie ponawiania oraz rejestrowania zmian w czasie odwoływania. Rejestrowanie zapisów z wyprzedzeniem omówiono już w podrozdziale 22.1.3. Drugie pojęcie, **powtarzania historii** (ang. *repeating history*), oznacza, że algorytm ARIES powtarza wszystkie akcje systemu bazodanowego sprzed wystąpienia awarii w celu zrekonstruowania jej stanu z *momentu, w którym nastąpiła awaria*. Transakcje, które były niezatwierdzone w czasie wystąpienia awarii (transakcje aktywne), są odwoływane. Trzecie pojęcie, **rejestrowania w czasie odwoływania** (ang. *logging during undo*), zapobiega powtarzaniu przez system ARIES zakończonych operacji odwołania, jeżeli awaria wystąpiła w czasie odtwarzania, co powoduje ponowienie całego procesu.

Procedura odtwarzania w algorytmie ARIES składa się z trzech głównych etapów: analizy, ponowienia i odwołania. **Etap analizy** (ang. *analysis step*) identyfikuje zmodyfikowane strony w buforze<sup>6</sup> oraz zbiór transakcji aktywnych w czasie wystąpienia awarii. Określany jest również odpowiedni punkt w dzienniku, od którego powinny się rozpocząć operacje ponowienia. **Faza ponowienia** (ang. *REDO phase*) polega na faktycznym zastosowaniu aktualizacji względem bazy danych na podstawie informacji z dziennika. Ogólnie rzecz biorąc, operacja REDO jest stosowana tylko względem transakcji zatwierdzonych, jednak w przypadku systemu ARIES jest inaczej. Określone informacje z dziennika systemu ARIES definiują punkt początkowy ponawiania, od którego operacje REDO są stosowane do momentu osiągnięcia końca dziennika. Ponadto informacje przechowywane przez system ARIES oraz na stronach danych pozwalają na określenie, czy operacje, które mają zostać ponowione, zostały faktycznie zastosowane względem bazy danych, a zatem nie muszą być powtarzane. Tak więc w czasie odtwarzania stosowane są *tylko wymagane operacje ponawiania*. Wreszcie, w czasie **fazy odwołania** (ang. *UNDO phase*), dziennik jest przeglądany w tył i operacje transakcji, które były aktywne w momencie awarii, są odwoływane w odwrotnej kolejności. Informacje wymagane przez system ARIES w celu przeprowadzenia takiej procedury odtwarzania to między innymi dziennik, tabela transakcji (ang. *Transaction Table*) oraz tabela zmodyfikowanych stron (ang. *Dirty Page Table*). Ponadto wykorzystywany jest mechanizm tworzenia punktów kontrolnych. Wymienione tabele są przechowywane przez menedżera transakcji i zapisywane w dzienniku w czasie tworzenia punktu kontrolnego.

W przypadku algorytmu ARIES każdy rekord dziennika posiada **numer sekwencyjny dziennika** (ang. *log sequence number, LSN*), który wzrasta monotonicznie i określa adres rekordu dziennika na dysku. Każdy numer LSN odpowiada *określonej zmianie* (akcji) pewnej transakcji. Ponadto każda strona danych przechowuje numer LSN *ostatniego rekordu dziennika odpowiadającego zmianie dla tej strony*. Rekord dziennika jest zapisywany dla każdej z następujących akcji: aktualizacja strony (ang. *write*), zatwierdzenie transakcji

---

<sup>6</sup> Faktyczne bufory mogą zostać utracone w czasie awarii, ponieważ znajdują się w pamięci. Dodatkowe tabele przechowywane w dzienniku w czasie tworzenia punktu kontrolnego (tabela zmodyfikowanych stron, tabela transakcji) pozwalają systemowi ARIES na zidentyfikowanie tych informacji.

(ang. *commit*), anulowanie transakcji (ang. *abort*), odwołanie aktualizacji (ang. *undo*) oraz zakończenie transakcji (ang. *end*). Potrzebę uwzględnienia w dzienniku pierwszych trzech wymienionych akcji już omówiliśmy, ale dwie pozostałe wymagają pewnych wyjaśnień. Kiedy aktualizacja jest odwoływana, w dzienniku jest zapisywany *rekord kompensacji* (ang. *compensation log record*). Kiedy transakcja jest kończona, czy to po zatwierdzeniu, czy anulowaniu, zapisywany jest *rekord zakończenia* (ang. *end log record*).

Do pól wspólnych dla wszystkich rekordów dziennika należą: poprzedni numer LSN danej transakcji, identyfikator transakcji oraz rodzaj rekordu dziennika. Poprzedni numer LSN jest istotny, ponieważ łączy on rekordy dziennika (w odwrotnej kolejności) związane z tą samą transakcją. W przypadku operacji aktualizacji (zapisu) do dodatkowych pól w rekordzie dziennika należą: identyfikator strony zawierającej element, długość aktualizowanego elementu, jego przesunięcie względem początku strony, obraz pierwotny elementu oraz jego obraz końcowy.

Oprócz dziennika, w celu zapewnienia wydajnego odtwarzania potrzebne są dwie tabele: **tabela transakcji** (ang. *Transaction Table*) oraz **tabela zmodyfikowanych stron** (ang. *Dirty Page Table*), które są zarządzane przez menedżera transakcji. W przypadku wystąpienia awarii tabele te są rekonstruowane w trakcie etapu analizy procesu odtwarzania. Tabela transakcji zawiera wpis dla *każdej transakcji aktywnej* z informacjami takimi jak identyfikator transakcji, jej status oraz numer LSN ostatniego rekordu dziennika dla danej transakcji. Tabela zmodyfikowanych stron zawiera wpis dla każdej zmodyfikowanej strony w pamięci podręcznej SZBD i uwzględnia identyfikator strony oraz numer LSN odpowiadający najwcześniejszej aktualizacji danej strony.

**Wykonywanie punktów kontrolnych** w systemie ARIES składa się z następujących etapów: zapisanie w dzienniku rekordu *rozpocznij\_punkt\_kontrolny*, zapisanie w dzienniku rekordu *zakończ\_punkt\_kontrolny* oraz zapisanie *numeru LSN* rekordu *zakończ\_punkt\_kontrolny* w specjalnym pliku. W czasie procesu odtwarzania do takiego pliku specjalnego uzyskiwany jest dostęp w celu zlokalizowania informacji o ostatnim punkcie kontrolnym. Wraz z rekordem *zakończ\_punkt\_kontrolny* zawartość tabeli transakcji oraz tabeli zmodyfikowanych stron jest dołączana na koniec dziennika. W celu zredukowania związanych z tym kosztów wykorzystywany jest mechanizm **przybliżonych punktów kontrolnych**, tak aby SZBD mógł kontynuować wykonywanie transakcji w czasie tworzenia punktu kontrolnego (patrz podrozdział 22.1.4). Ponadto zawartość pamięci podręcznej SZBD nie musi być zapisywana na dysku w czasie tworzenia punktu kontrolnego, gdyż tabela transakcji i tabela zmodyfikowanych stron — dołączane do dziennika na dysku — zawierają wszelkie informacje potrzebne dla celów odtwarzania. Należy zauważyć, że jeżeli awaria nastąpi w czasie tworzenia punktu kontrolnego, plik specjalny odwołuje się do poprzedniego punktu kontrolnego, używanego w celu odtwarzania.

Po wystąpieniu awarii kontrolę przejmuje menedżer odtwarzania systemu ARIES. Najpierw uzyskiwany jest dostęp do informacji o ostatnim punkcie kontrolnym poprzez plik specjalny. **Faza analizy** rozpoczyna się od rekordu *rozpocznij\_punkt\_kontrolny* i trwa aż do osiągnięcia końca dziennika. Kiedy zostanie napotkany rekord *zakończ\_punkt\_kontrolny*, uzyskiwany jest dostęp do tabeli transakcji oraz tabeli zmodyfikowanych stron (warto przypomnieć, że tabele te zostały zapisane w dzienniku w czasie operacji tworzenia punktu kontrolnego). W czasie analizy sprawdzane rekordy dziennika mogą powodować modyfikacje tych dwóch tabel. Przykładowo, jeżeli dla transakcji *T* z tabeli transakcji

zostanie napotkany rekord zakończenia, wówczas dotyczący jej wpis jest usuwany z tabeli. Jeżeli zostanie napotkany inny rodzaj rekordu dla transakcji  $T'$ , wówczas wpis dla tej transakcji jest wstawiany do tabeli transakcji, o ile jeszcze tam nie występuje, i zostaje zmodyfikowane pole z ostatnią wartością LSN. Jeżeli rekord dziennika odpowiada zmianie strony  $P$ , wówczas do tabeli wstawiany jest związany z nią wpis (o ile już tam nie występuje) i modyfikowana jest wartość odpowiedniego pola LSN. Po zakończeniu fazy analizy w tabelach znajdują się wszystkie informacje wymagane przez procesy ponawiania i odwoływania.

Następnie występuje **faza ponawiania**. W celu zredukowania ilości wymaganej pracy, system ARIES rozpoczyna operacje ponawiania od punktu w dzienniku, o którym wie (na pewno), że poprzednie zmiany zmodyfikowanych stron *zostały już zastosowane względem bazy danych na dysku*. Może to zostać określone poprzez znalezienie najmniejszej wartości LSN, czyli  $M$ , spośród wszystkich zmodyfikowanych stron w tabeli zmodyfikowanych stron, określającej pozycję w dzienniku, od której system ARIES musi rozpocząć fazę ponawiania. Wszelkie zmiany odpowiadające wartości  $LSN < M$ , w przypadku transakcji umożliwiających ponowienie, musiały już zostać zapisane na dysku lub zostały nadpisane w buforze. W przeciwnym razie zmodyfikowane strony z takim numerem LSN znajdowałyby się w buforze (oraz w tabeli zmodyfikowanych stron). Tak więc proces ponawiania rozpoczyna się od rekordu dziennika z wartością  $LSN = M$  i jest kontynuowany aż do końca dziennika.

Dla każdej zmiany zarejestrowanej w dzienniku algorytm ponawiania sprawdza, czy zmiana ma zostać ponowiona. Przykładowo, jeżeli zmiana zarejestrowana w dzienniku dotyczy strony  $S$ , która nie występuje w tabeli zmodyfikowanych stron, oznacza to, że taka zmiana została już zapisana na dysku i nie musi być ponawiana. Jeżeli jednak zmiana zarejestrowana w dzienniku (powiedzmy, dla  $LSN = N$ ) dotyczy strony  $S$ , a tabela zmodyfikowanych stron zawiera wpis dla  $S$  z wartością LSN większą od  $N$ , wówczas zmiana została już uwzględniona. Jeżeli nie jest spełniony żaden z tych dwóch warunków, z dysku wczytywana jest strona  $S$  i numer LSN przechowywany na tej stronie,  $LSN(S)$ , jest porównywany z  $N$ . Jeżeli  $N < LSN(S)$ , wówczas zmiana została zastosowana i strona nie musi być zapisywana z powrotem na dysku.

Kiedy faza ponawiania zostanie zakończona, baza danych znajduje się dokładnie w takim samym stanie, w jakim była w momencie wystąpienia awarii. Zbiór transakcji aktywnych — określany nazwą `undo_set` — został zidentyfikowany w tabeli transakcji na etapie analizy. Następnie zostaje wykonana **faza odwołania** polegająca na przeglądaniu dziennika w tył i odwoływaniu odpowiednich akcji. Dla każdej takiej akcji w dzienniku jest zapisywany rekord kompensacji. Dziennik jest odczytywany w tył do momentu, aż każda transakcja należąca do zbioru transakcji `undo_set` zostanie odwołana. Po wykonaniu tych działań proces odtwarzania jest zakończony i można rozpocząć normalne przetwarzanie.

Weźmy pod uwagę przykład odtwarzania z rysunku 22.5. Mamy tu do czynienia z trzema transakcjami:  $T_1$ ,  $T_2$  oraz  $T_3$ . Transakcja  $T_1$  aktualizuje stronę  $C$ , transakcja  $T_2$  aktualizuje strony  $B$  i  $C$ , zaś transakcja  $T_3$  — stronę  $A$ . Na rysunku 22.5(a) przedstawiono częściową zawartość dziennika, zaś na rysunku 22.5(b) — zawartość tabeli transakcji oraz tabeli zmodyfikowanych stron. Załóżmy, że w tym momencie następuje awaria. Ze względu na fakt, że został utworzony punkt kontrolny, pobierany jest adres odpowiedniego rekordu

rozpoczni\_j\_punkt\_kontrolny, którego lokalizację określa wartość 4. Faza analizy rozpoczyna się od lokalizacji 4. i jest kontynuowana aż do końca. Na etapie rekordu koniec\_punktu\_kontrolnego tabela transakcji i tabela zmodyfikowanych stron mają postać przedstawioną na rysunku 22.5(b) i faza analizy powoduje dalszą rekonstrukcję tych tabel. Kiedy przetwarzanie dochodzi do rekordu 6., do tabeli transakcji jest wstawiany nowy wpis dla transakcji  $T_3$ , zaś do tabeli zmodyfikowanych stron — wpis dla strony  $A$ . Po przeanalizowaniu rekordu 8. status transakcji  $T_2$  w tabeli transakcji jest zmieniany na zatwierdzony. Na rysunku 22.5(c) przedstawiono obie tabele po zakończeniu fazy analizy.

(a)

LSN	OSTATNI_LSN	ID_TRAN	RODZAJ	ID_STRONY	INNE INFORMACJE
1	0	$T_1$	Aktualizacja	$C$	...
2	0	$T_2$	Aktualizacja	$B$	...
3	1	$T_1$	Zatwierdzenie		...
4	Rozpoczni_j_punkt_kontrolny				
5	Zakończ_punkt_kontrolny				
6	0	$T_3$	Aktualizacja	$A$	...
7	2	$T_2$	Aktualizacja	$C$	...
8	7	$T_2$	Zatwierdzenie		...

TABELA TRANSAKCJI

(b)

ID_TRANSAKCJI	OSTATNI_LSN	STATUS
$T_1$	3	Zatwierdzenie
$T_2$	2	W toku

TABELA ZMODYFIKOWANYCH STRON

ID_STRONY	LSN
$C$	1
$B$	2

TABELA TRANSAKCJI

(c)

ID_TRANSAKCJI	OSTATNI_LSN	STATUS
$T_1$	3	Zatwierdzenie
$T_2$	8	Zatwierdzenie
$T_3$	6	W toku

TABELA ZMODYFIKOWANYCH STRON

ID_STRONY	LSN
$C$	7
$B$	2
$A$	6

RYСУNEK 22.5. Przykład odtwarzania w przypadku algorytmu ARIES. (a) Dziennik w momencie wystąpienia awarii. (b) Tabela transakcji i zmodyfikowanych stron w momencie wykonywania punktu kontrolnego. (c) Tabela transakcji i zmodyfikowanych stron po fazie analizy

W przypadku fazy ponawiania, numerem LSN o najmniejszej wartości w tabeli zmodyfikowanych stron jest 1. Stąd operacje rozpoczynają wykonywanie od rekordu 1. Numery LSN {1, 2, 6, 7}, odpowiadające aktualizacjom dla stron, odpowiednio,  $C$ ,  $B$ ,  $A$  i  $C$ , są nie mniejsze od numerów LSN tych stron (co pokazuje zawartość tabeli zmodyfikowanych stron). Tak więc owe strony danych są ponownie odczytywane i wykonywane są aktualizacje na podstawie informacji z dziennika (zakładając, że faktyczne numery LSN przechowywane w tych stronach danych są mniejsze od odpowiadających im wpisów dziennika). W tym momencie faza ponawiania zostaje zakończona i rozpoczyna się faza odwoływania.



Na podstawie tabeli transakcji (rysunek 22.5(c)) operacje odwołania są stosowane tylko względem aktywnej transakcji  $T_3$ . Faza odwołania rozpoczyna się od wpisu dziennika o numerze 6 (ostatniej aktualizacji transakcji  $T_3$ ) i jest przeprowadzana przy przeglądaniu dziennika wstecz. Zostaje prześledzony w odwrotnym kierunku łańcuch aktualizacji transakcji  $T_3$  (w omawianym przykładzie dotyczy to tylko rekordu 6.) i odwoływane są odpowiednie operacje.

## 22.6. Odtwarzanie w systemach wielu baz danych

Jak dotąd niejawnie zakładaliśmy, że transakcja uzyskuje dostęp do pojedynczej bazy danych. W pewnych przypadkach pojedyncza transakcja, określana mianem **transakcji wielu baz danych** (ang. *multidatabase transaction*), może wymagać uzyskania dostępu do wielu baz danych. Takie bazy mogą nawet być przechowywane w różnych SZBD. Przykładowo, niektóre SZBD mogą być relacyjne, zaś inne obiektowe, hierarchiczne lub sieciowe. W takim przypadku każdy SZBD związany z transakcją wielu baz danych może wykorzystywać własną technikę odtwarzania i własnego menedżera transakcji, niezależnego od innych SZBD. Sytuacja taka przypomina nieco przypadek systemu zarządzania rozproszoną bazą danych (patrz rozdział 23.), gdzie fragmenty bazy danych są umieszczane w odrębnych miejscach, połączonych siecią.

W celu zachowania niepodzielności transakcji wielu baz danych konieczne jest posiadanie dwupoziomowego mechanizmu odtwarzania. **Globalny menedżer odtwarzania** (ang. *global recovery manager*), czyli **koordynator**, służy do przechowywania informacji wymaganych do odtwarzania, zaś lokalne menedżery odtwarzania przechowują dzienniki i tablice. Koordynator zwykle wykorzystuje protokół znany jako **protokół zatwierdzania dwufazowego** (ang. *two-phase commit protocol*). Dwie fazy jego działania można opisać następująco:

- **Faza 1.** Kiedy wszystkie współdziałające bazy danych sygnalizują koordynatorowi, że część transakcji wielu baz danych dotycząca każdej z nich została wykonana, koordynator przesyła do każdej komunikat *przygotowania do zatwierdzenia*. Każda współdziałająca baza, otrzymując taki komunikat, przeprowadza wymuszony zapis na dysku wszystkich rekordów dziennika oraz informacji wymaganych przez lokalny mechanizm odtwarzania i przesyła do koordynatora komunikat *gotowości do zatwierdzenia* lub *OK*. Jeżeli wymuszony zapis na dysku zakończy się niepowodzeniem lub lokalna transakcja nie będzie mogła z pewnego powodu zostać zatwierdzona, taka baza danych przesyła do koordynatora komunikat o *niemożności zatwierdzenia* lub *nie OK*. Jeżeli koordynator nie otrzyma od bazy danych żadnej odpowiedzi w ramach określonego przedziału czasu, przyjmuje, że został przesłany komunikat *nie OK*.
- **Faza 2.** Jeżeli *wszystkie* współdziałające bazy prześlą komunikat *OK*, a decyzją koordynatora również jest *OK*, transakcja kończy się sukcesem i koordynator przesyła komunikat *zatwierdzenia* do współdziałających baz danych. Ze względu na fakt, że wszystkie lokalne efekty transakcji i informacje potrzebne w celu dokonania odtwarzania lokalnego zostały zarejestrowane w dziennikach współdziałających baz, możliwe staje się w tym momencie odtwarzanie po awariach. Każda współdziałająca



baza danych kończy zatwierdzenie transakcji, zapisując wpis [zatwierdź] dla danej transakcji w dzienniku i, o ile to konieczne, trwale aktualizując bazę danych. Z drugiej strony, jeżeli jedna lub większa liczba współdziałających baz danych lub koordynator prześle komunikat *nie OK*, transakcja kończy się niepowodzeniem i koordynator przesyła komunikat *wycofania* lub odwołania lokalnych efektów transakcji w każdej współdziałającej bazie danych. Jest to przeprowadzane poprzez odwołanie operacji zapisanych w dzienniku.

Wspólnym efektem obu faz działania protokołu jest albo zatwierdzenie przez wszystkie bazy transakcji, albo niezatwierdzenie jej przez żadną. W przypadku, gdy któraś z baz (lub koordynator) ulegnie awarii, zawsze istnieje możliwość wykonania operacji przywracania do stanu, w którym transakcja jest albo zatwierdzana, albo wycofywana. Awaria występująca w czasie działania fazy 1. zwykle wymaga wycofania transakcji, natomiast awaria występująca w fazie 2. oznacza, że udana transakcja może zostać odtworzona i zatwierdzona.

## 22.7. Tworzenie kopii bezpieczeństwa bazy danych i odtwarzanie po awariach katastroficznych

Jak dotąd wszystkie omawiane techniki dotyczyły awarii niekatastroficznych. Kluczowe znaczenie miało tu przyjęcie założenia, że dziennik systemowy był przechowywany na dysku i nie został utracony w wyniku awarii. Podobnie katalog przesłaniania musi być przechowywany na dysku w celu umożliwienia odtwarzania w przypadku użycia techniki stronicowania z przesłanianiem. Omówione techniki odtwarzania wykorzystują wpisy z dziennika systemowego lub z katalogu przesłaniania w celu wznowienia działania po awarii poprzez przywrócenie bazy danych do spójnego stanu.

Menedżer odtwarzania SZBD musi również być wyposażony w mechanizmy umożliwiające odtwarzanie po awariach katastroficznych, takich jak awaria dysku. Główną techniką używaną w takim przypadku jest tworzenie **kopii bezpieczeństwa bazy danych** (ang. *database backup*). Cała baza danych i dziennik są okresowo zapisywane na tanim nośniku danych, takim jak taśma magnetyczna lub inny pojemny nośnik pamięci masowej przechowywany w trybie offline. W przypadku wystąpienia katastroficznej awarii systemu ostatnia kopia bezpieczeństwa może zostać skopiowana z taśmy na dysk, a system zrestartowany.

Dane z aplikacji krytycznych, np. baz w obszarach bankowości, ubezpieczeń, giełdy, są okresowo archiwizowane w całości i przenoszone do fizycznie odrębnej bezpiecznej lokalizacji. W celu ochrony takich danych przed powodzią, sztormami, trzęsieniem ziemi lub ogniem stosuje się podziemne skarbcze danych. Wydarzenia takie jak atak terrorystyczny z 11 września w Nowym Jorku (2001 r.) lub huragan Katrina w Nowym Orleanie (2005 r.) zwiększyły świadomość potrzeby *odtworzenia krytycznych baz danych po katastrofach*.

W celu uniknięcia możliwości utraty efektów działania wszystkich transakcji, które zostały wykonane od ostatniej archiwizacji, kopie dziennika są często wykonywane w krótszych odstępach czasu niż pełna kopia bazy danych; w tym celu dziennik jest okresowo ko-

piowany na taśmę magnetyczną. Dziennik systemowy zazwyczaj posiada znacznie mniejszy rozmiar od samej bazy, co umożliwia jego częstszą archiwizację. Użytkownicy nie tracą wówczas żadnych transakcji wykonanych od czasu utworzenia ostatniej kopii bezpieczeństwa bazy danych. Wszystkie zatwierdzone transakcje, zarejestrowane w części dziennika, która została zarchiwizowana na taśmie, są wykorzystywane do ponowienia wykonanych operacji. Po każdorazowym wykonaniu kopii bezpieczeństwa bazy danych tworzony jest nowy dziennik. Stąd w celu odtworzenia bazy po awarii dysku najpierw odtwarzana jest baza danych na podstawie jej ostatniej kopii. Następnie rekonstruowane są efekty wszystkich zatwierdzonych transakcji, które zostały zarejestrowane w zarchiwizowanych kopiach dziennika systemowego.

## 22.8. Podsumowanie

W niniejszym rozdziale omówiono techniki odtwarzania transakcji po awariach. Głównym celem odtwarzania jest zapewnienie właściwości niepodzielności transakcji. Jeżeli transakcja ulegnie awarii przed swoim zakończeniem, mechanizm odtwarzania musi zapewniać, że nie będzie ona miała żadnego trwałego wpływu na stan bazy danych. Najpierw (w podrozdziale 22.1) przedstawiono nieformalne omówienie procesu odtwarzania, a następnie omówienie pojęć związanych z systemem odtwarzania. Uwzględniono między innymi kwestię pamięci podręcznej, aktualizowania bezpośredniego i przesłaniania, obrazy pierwotne i końcowe elementów danych, operacje odtwarzania UNDO i REDO, strategie z zabieraniem i bez zabierania oraz z wymuszaniem i bez wymuszania, tworzenie systemowych punktów kontrolnych oraz protokół rejestrowania zapisów z wyprzedzeniem.

Następnie omówiono dwa różne podejścia do problemu odtwarzania: aktualizacje odroczone (podrozdział 22.2) oraz aktualizacje natychmiastowe (podrozdział 22.3). Techniki aktualizacji odroczone odkładają faktyczne aktualizacje bazy danych na dysku do momentu, aż transakcja osiągnie swój punkt zatwierdzenia. Transakcja wymusza zapis dziennika na dysku przed zarejestrowaniem zmian w bazie danych. Podejście to, używane z określonymi metodami sterowania współbieżnego, zapewnia, że nigdy nie jest konieczne przeprowadzanie operacji wycofania transakcji i odtwarzanie polega jedynie na ponowieniu operacji transakcji zatwierdzonych po ostatnim punkcie kontrolnym na podstawie informacji z dziennika. Wadą tego rozwiązania jest fakt, że może wymagać zbyt dużej przestrzeni buforów, gdyż aktualizacje są przechowywane w buforach i nie są stosowane względem bazy danych aż do momentu zatwierdzenia transakcji. Aktualizacje odroczone prowadzą do wykorzystania algorytmu znanego jako NO-UNDO/REDO. Techniki aktualizacji natychmiastowych mogą stosować zmiany względem bazy danych na dysku zanim transakcja zostanie z powodzeniem zakończona. Wszelkie zmiany zastosowane względem bazy danych muszą najpierw być zarejestrowane w dzienniku i zapisane na dysku, tak aby w razie potrzeby można je było odwołać. Przedstawiono również omówienie algorytmu odtwarzania dla aktualizacji natychmiastowych, znanego jako algorytm UNDO/REDO. Można również opracować inny algorytm, znany jako UNDO/NO-REDO, dla aktualizacji natychmiastowych, jeżeli wszystkie działania transakcji są rejestrowane w bazie danych przed zatwierdzeniem.

W podrozdziale 22.4 omówiono technikę stronicowania z przesłanianiem dla celów odtwarzania, która przechowuje wszystkie stare strony bazy danych przy użyciu katalogu przesłaniania. Technika ta, klasyfikowana jako algorytm NO-UNDO/NO-REDO, nie wymaga użycia dziennika w systemie jednoużytkownikowym, ale w przypadku systemu wieloużytkownikowego dziennik może okazać się potrzebny. W podrozdziale 22.5 przedstawiono algorytm ARIES — konkretny schemat odtwarzania używany w przypadku niektórych produktów z obszaru relacyjnych baz danych firmy IBM. Następnie, w podrozdziale 22.6, omówiono protokół zatwierdzania dwufazowego, który jest używany w celu odtwarzania po awariach w przypadku transakcji pochodzących z wielu baz danych. Wreszcie w podrozdziale 22.7 omówiono problem odtwarzania po awariach katastroficznych, co zwykle zapewnia się, wykonując kopie bezpieczeństwa bazy danych i dziennika oraz rejestrując je na taśmach. Dziennik może być archiwizowany częściej niż baza danych i taki zarchiwizowany dziennik można wykorzystać w celu ponowienia wszystkich operacji od czasu wykonania ostatniej kopii bezpieczeństwa samej bazy.

## Pytania powtórkowe

- 22.1. Omów różne rodzaje awarii transakcji. Co rozumiemy przez *awarię katastroficzną*?
- 22.2. Omów działania podejmowane przez operacje `odczytaj_element` i `zapisz_element` w bazie danych.
- 22.3. W jakim celu jest używany dziennik systemowy? Jakie są typowe rodzaje wpisów zawartych w dzienniku? Czym są punkty kontrolne i dlaczego są istotne? Czym są punkty zatwierdzenia transakcji i dlaczego są istotne?
- 22.4. W jaki sposób są wykorzystywane techniki buforowania i pamięci podręcznej przez podsystemy odtwarzania?
- 22.5. Czym jest obraz pierwotny (BFIM) i obraz końcowy (AFIM) elementu danych? Jaka istnieje różnica między aktualizowaniem bezpośrednim a przesłanianiem pod względem obsługi przez nie obrazów BFIM i AFIM?
- 22.6. Czym są wpisy dziennika typu UNDO i REDO?
- 22.7. Opisz protokół rejestrowania zapisów z wyprzedzeniem.
- 22.8. Zidentyfikuj trzy typowe listy transakcji, które są przechowywane przez podsystem odtwarzania.
- 22.9. Co rozumiemy pod pojęciem *wycofania transakcji*? Co rozumiemy przez *wycofanie kaskadowe*? Dlaczego praktyczne metody odtwarzania wykorzystują protokoły, które nie pozwalają na występowanie wycofania kaskadowego? Które techniki odtwarzania w ogóle nie wymagają wycofywania?
- 22.10. Omów operacje UNDO i REDO oraz techniki odtwarzania, które je wykorzystują.
- 22.11. Omów technikę odtwarzania opartą na aktualizacjach odroczonej. Jakie są zalety i wady tej techniki? Dlaczego określa się ją mianem algorytmu NO-UNDO/REDO?
- 22.12. W jaki sposób proces odtwarzania może obsłużyć operacje transakcji, które nie mają wpływu na bazę danych, takie jak wyświetlanie przez transakcję raportów?

- 22.13. Omów techniki odtwarzania oparte na aktualizacjach natychmiastowych w środowiskach jedno- i wieloużytkownikowych. Jakie są zalety i wady aktualizacji natychmiastowych?
- 22.14. Jaka jest różnica między algorytmami UNDO/REDO a UNDO/NO-REDO w przypadku odtwarzania z aktualizacjami natychmiastowymi? Opracuj zarys algorytmu UNDO/NO-REDO.
- 22.15. Opisz technikę odtwarzania ze stronicowaniem z przesłanianiem. W jakiej sytuacji nie wymaga ona użycia dziennika?
- 22.16. Opisz trzy fazy metody odtwarzania algorytmu ARIES.
- 22.17. Czym są numery sekwencyjne dziennika (LSN) w systemie ARIES? W jaki sposób są wykorzystywane? Jakie informacje przechowuje tabela zmodyfikowanych stron i tabela transakcji? Opisz, w jaki sposób w systemie ARIES używa się techniki tworzenia przybliżonych punktów kontrolnych.
- 22.18. Co w kontekście zarządzania buforami w zakresie przetwarzania transakcji oznaczają pojęcia *zabierania* (*niezabierania*) oraz *wymuszania* (*niewymuszania*)?
- 22.19. Opisz protokół odtwarzania dwufazowego w przypadku transakcji wielu baz danych.
- 22.20. Omów, w jaki sposób obsługuje się odtwarzanie po awariach katastroficznych.

## Ćwiczenia

- 22.21. Załóżmy, że system uległ awarii przed zapisaniem wpisu [odczytaj\_element, T3, A] w dzienniku z rysunku 22.1(b). Czy będzie to stanowiło jakąś różnicę dla procesu odtwarzania?
- 22.22. Załóżmy, że system uległ awarii przed zapisaniem wpisu [zapisz\_element, ↪T2, D, 25, 26] w dzienniku z rysunku 22.1(b). Czy będzie to stanowiło jakąś różnicę dla procesu odtwarzania?
- 22.23. Na rysunku 22.6 przedstawiono dziennik odpowiadający określonego harmonogramowi w momencie wystąpienia awarii dla czterech transakcji  $T_1$ ,  $T_2$ ,  $T_3$  i  $T_4$ . Załóżmy, że używamy *protokołu aktualizacji natychmiastowych* z punktami kontrolnymi. Opisz proces odtwarzania po awarii systemu. Określ, które transakcje są wycofywane, które operacje z dziennika są ponawiane, a które (o ile takie istnieją) są odwoływane, oraz czy występuje wycofywanie kaskadowe.
- 22.24. Załóżmy, że używamy protokołu aktualizacji odroczonej dla przykładu z rysunku 22.6. Pokaż, czym różniłby się dziennik w przypadku aktualizacji odroczonej, usuwając niepotrzebne wpisy dziennika. Następnie opisz proces odtwarzania, używając takiego zmodyfikowanego dziennika. Załóż, że stosowane są wyłącznie operacje ponowienia; określ, które operacje z dziennika są ponawiane, a które ignorowane.
- 22.25. Czym różni się proces tworzenia punktu kontrolnego w systemie ARIES od tworzenia takich punktów zgodnie z opisem zawartym w punkcie 22.1.4?

[rozpocznij_transakcję, $T_1$ ]
[odczytaj_element, $T_1, A$ ]
[odczytaj_element, $T_1, D$ ]
[zapisz_element, $T_1, D, 20, 25$ ]
[zatwierdź, $T_1$ ]
[punkt kontrolny]
[rozpocznij_transakcję, $T_2$ ]
[odczytaj_element, $T_2, B$ ]
[zapisz_element, $T_2, B, 12, 18$ ]
[rozpocznij_transakcję, $T_4$ ]
[odczytaj_element, $T_4, D$ ]
[zapisz_element, $T_4, D, 25, 15$ ]
[rozpocznij_transakcję, $T_3$ ]
[zapisz_element, $T_3, C, 30, 40$ ]
[zapisz_element, $T_4, A$ ]
[zapisz_element, $T_4, A, 30, 20$ ]
[zatwierdź, $T_4$ ]
[odczytaj_element, $T_2, D$ ]
[zapisz_element, $T_2, D, 15, 25$ ]

← Awaria systemu

RYSUNEK 22.6. Przykład harmonogramu i odpowiadający mu dziennik

- 22.26. W jaki sposób system ARIES używa numerów sekwencyjnych dziennika w celu zredukowania ilości działań związanych z ponawianiem potrzebnych w ramach procesu odtwarzania? Zilustruj odpowiednim przykładem użycie informacji przedstawionych na rysunku 22.5. Możesz przyjmować własne założenia co do momentów, w których strony są zapisywane na dysku.
- 22.27. Jakie implikacje w kontekście punktów kontrolnych i odtwarzania niesłaby ze sobą strategia bez zabierania z wymuszaniem?

*Dla każdego z poniższych pytań wybierz prawidłową odpowiedź spośród dostępnych czterech możliwości.*

- 22.28. Rejestrowanie przyrostowe z aktualizacjami odroczonymi implikuje, że system odtwarzania musi
- przechowywać w dzienniku starą wartość zaktualizowanego elementu;
  - przechowywać w dzienniku nową wartość zaktualizowanego elementu;
  - przechowywać w dzienniku zarówno starą, jak i nową wartość zaktualizowanego elementu;
  - przechowywać w dzienniku tylko rekordy rozpoczęcia transakcji i zakończenia transakcji.

- 22.29. Protokół rejestrowania zapisów z wyprzedzeniem oznacza po prostu, że
- a) zapis elementu danych powinien odbywać się przed rejestrowaniem operacji;
  - b) rekord dziennika dla operacji powinien być zapisywany przed zapisem faktycznych danych;
  - c) wszystkie rekordy dziennika powinny być zapisywane przed rozpoczęciem wykonywania nowej transakcji;
  - d) dziennik nigdy nie musi być zapisywany na dysku.
- 22.30. W przypadku awarii transakcji w modelu aktualizacji odroczonej i rejestrowania przyrostowego, które z poniższych operacji są wymagane:
- a) operacje odwołania;
  - b) operacje ponowienia;
  - c) operacje odwołania i ponowienia;
  - d) żadne z powyższych.
- 22.31. W przypadku rejestrowania przyrostowego z aktualizacjami natychmiastowymi rekord dziennika dla transakcji zawiera:
- a) nazwę transakcji, nazwę elementu danych, starą wartość elementu, nową wartość elementu;
  - b) nazwę transakcji, nazwę elementu danych, starą wartość elementu;
  - c) nazwę transakcji, nazwę elementu danych, nową wartość elementu;
  - d) nazwę transakcji oraz nazwę elementu danych.
- 22.32. W celu zapewnienia poprawnego zachowania w czasie odtwarzania operacje odwołania i ponowienia muszą być
- a) przemienne;
  - b) łączne;
  - c) powtarzalne;
  - d) rozdzielne.
- 22.33. W razie wystąpienia awarii sprawdzany jest dziennik i każda operacja jest albo odwoływana, albo ponawiana. Stanowi to problem, gdyż
- a) przeszukiwanie całego dziennika jest czasochłonne;
  - b) wiele operacji ponowienia jest niepotrzebnych;
  - c) prawdziwe jest zarówno stwierdzenie a), jak i b);
  - d) żadne z powyższych.
- 22.34. Używając schematu odtwarzania opartego na dzienniku można zwiększyć wydajność, jak również zapewnić mechanizm odtwarzania, poprzez
- a) zapisywanie rekordów dziennika na dysku w momencie zatwierdzenia transakcji;
  - b) zapisywanie odpowiednich rekordów dziennika na dysku w czasie wykonywania transakcji;
  - c) czekanie z zapisaniem rekordów dziennika do momentu, aż zostanie zatwierdzonych kilka transakcji, i zapisanie ich razem;
  - d) rezygnację z zapisywania rekordów dziennika na dysku.

- 22.35. Możliwość wycofywania kaskadowego występuje wówczas, gdy
- a) transakcja zapisuje elementy, które zostały zapisane tylko przez transakcje zatwierdzone;
  - b) transakcja zapisuje element, który wcześniej został zapisany przez niezatwierdzoną transakcję;
  - c) transakcja odczytuje element, który wcześniej został zapisany przez niezatwierdzoną transakcję.
  - d) spełnione są warunki z punktów b) i c).
- 22.36. W celu obsłużenia awarii nośników (dysku) konieczne jest
- a) aby SZBD wykonywał transakcje tylko w środowisku jednoużytkownikowym;
  - b) przechowywanie nadmiarowej kopii bazy danych;
  - c) rezygnacja z anulowania transakcji;
  - d) konieczne są wszystkie powyższe działania.
- 22.37. Jeżeli w celu zapisywania elementu danych na dysku wykorzystywane jest podejście z przesłaniem, to
- a) element jest zapisywany na dysku po zatwierdzeniu transakcji;
  - b) element jest zapisywany w innej lokalizacji na dysku;
  - c) element jest zapisywany na dysku przed zatwierdzeniem transakcji;
  - d) element jest zapisywany w tej samej lokalizacji na dysku, z której został odczytany.

## Wybrane publikacje

Książki Bersteina i in. (1987) oraz Papadimitriou (1986) są poświęcone teorii i zasadom sterowania współbieżnego i odtwarzania. Książka Graya i Reutera (1993) to encyklopedyczna pozycja poświęcona sterowaniu współbieżnemu, odtwarzaniu i innym kwestiom związanym z przetwarzaniem transakcji.

Verhofstad (1978) prezentuje samouczek oraz analizę technik odtwarzania w systemach bazodanowych. Podział algorytmów w oparciu o ich charakterystyki UNDO/REDO omówiono w pozycjach Haerdera i Reutera (1983) oraz Bernsteina i in. (1983). Gray (1978) omawia odtwarzanie wraz z innymi aspektami implementacji systemów operacyjnych w kontekście baz danych. Technika stronicowania z przesłaniem została omówiona w pozycji Lorigo (1977), Verhofstada (1978) i Reutera (1980). Gray i in. (1981) omawiają mechanizm odtwarzania systemu SYSTEM R. Pozycje Lockemana i Knutsena (1968), Daviesa (1973) oraz Bjorka (1973) to jedne z pierwszych prac poświęconych odtwarzaniu. Chandy i in. (1975) omawiają wycofywanie transakcji. Lilien i Bhargava (1985) omawiają pojęcie bloków spójności oraz ich użycie w celu zwiększenia wydajności odtwarzania.

Odtwarzanie przy użyciu rejestrowania zapisów z wyprzedzeniem poddano analizie w pracy Jhingrana i Khedkara (1992) i jest ono wykorzystywane w systemie ARIES (Mohan i in. 1992). Nowsze badania poświęcone odtwarzaniu dotyczą między innymi kompensacji transakcji (Korth i in. 1990) oraz odtwarzania baz danych przechowywanych w pamięci głównej (Kumar 1991). Algorytm odtwarzania ARIES (Mohan i in. 1992) okazał



się w praktyce udany. Franklin i in. (1992) omawia odtwarzanie w systemie EXODUS. Dwie nowsze książki Kumara i Hsu (1998) oraz Kumara i Songa (1998) szczegółowo omawiają problematykę odtwarzania i zawierają opisy metod odtwarzania używanych w wielu istniejących produktach z obszaru relacyjnych baz danych. Przykładowe strategie zastępowania stron specyficzne dla baz danych omówiono w pracach Chou i DeWitta (1985) oraz Pazosa i in. (2006).





# Rozproszone bazy danych, systemy NOSQL i big data



## Zagadnienia z obszaru rozproszonych baz danych

W tym rozdziale przyjrzymy się bliżej rozproszonym bazom danych (*DDB* — ang. *Distributed Database*) i rozproszonym systemom zarządzania bazami danych (*DDBMS* — ang. *Distributed Database Management System*), oraz zastosowaniu architektury klient-serwer jako platformy operacyjnej dla systemów baz danych. Rozproszone bazy danych wprowadzają zalety przetwarzania rozproszonego do systemów baz danych. **Rozproszony system obliczeniowy** składa się z wielu elementów obliczeniowych, które są połączone siecią komputerową i mogą współpracować w celu zrealizowania wspólnego zadania. Ogólnie rzecz ujmując, systemy rozproszone dzielą duży, trudny do rozwiązania problem na mniejsze i rozwiązują je wydajnie w skoordynowany sposób. Do rozwiązania złożonego problemu wykorzystywana jest więc większa moc obliczeniowa, a każdy autonomiczny element systemu może być niezależnie zarządzany i współpracować z innymi, udostępniając funkcje potrzebne do wykonania zadania. Technologia rozproszonych baz danych powstała z połączenia dwóch dziedzin: systemów baz danych i systemów rozproszonych.

W latach 80. i 90. opracowano kilka prototypowych rozproszonych systemów baz danych, aby rozwiązać problemy z obszaru udostępniania danych, replikacji danych, przetwarzania zapytań i transakcji rozproszonych, zarządzania metadanymi rozproszonych baz danych itd. Później pojawiło się wiele nowych technologii łączących systemy rozproszone i technologie bazodanowe. Te technologie i systemy są rozwijane w celu obsługi składowania, analizowania oraz eksplorowania dużych ilości generowanych i zbieranych danych. Ogólnie takie rozwiązania są nazywane **technologiami big data**. Źródła tych technologii sięgają systemów rozproszonych i bazodanowych, a także eksplorowania danych i algorytmów uczenia maszynowego, które potrafią przetwarzać duże ilości danych w celu uzyskania potrzebnej wiedzy.

W tym rozdziale omawiamy podstawowe zagadnienia związane z danymi rozproszonymi i zarządzaniem nimi. W dwóch dalszych rozdziałach znajdziesz przegląd nowych technologii zarządzania i przetwarzania big data. Rozdział 24. zawiera omówienie nowej kategorii systemów baz danych — systemów NOSQL, które mają zapewniać rozproszone rozwiązania do zarządzania dużą ilością danych potrzebnych np.: w mediach społecznościowych, służbie zdrowia i aplikacjach z obszaru bezpieczeństwa. Rozdział 25. to wprowadzenie do zagadnień i systemów z dziedziny przetwarzania i analizy big data. Opisano tam np. model MapReduce i inne technologie przetwarzania rozproszonego. W rozdziale 25. przedstawimy też przetwarzanie w chmurze.

Podrozdział 23.1 wprowadza pojęcia związane z zarządzaniem rozproszonymi bazami danych. Szczegóły projektowania takich systemów, w tym kwestie podziału danych, rozpraszania na wiele miejsc i możliwej replikacji, są omawiane w podrozdziale 23.2. Podrozdział 23.3 zawiera opis sterowania współbieżnego i odtwarzania w bazach rozproszonych. Podrozdziały 23.4 i 23.5 wprowadzają techniki rozproszonego przetwarzania zapytań i transakcji. Podrozdziały 23.6 i 23.7 wprowadzają różne rodzaje rozproszonych baz danych (w tym systemy federacyjne i wielobazowe) oraz ich architektury, a także nakreślają problemy związane z heterogenicznością i potrzebą autonomii w systemach federacyjnych. W podrozdziale 23.8 przedstawiono modele zarządzania katalogiem w rozproszonych bazach danych. Podrozdział 23.9 obejmuje podsumowanie rozdziału.

Jeśli interesuje Cię krótkie wprowadzenie do rozproszonych baz danych, możesz zapoznać się z podrozdziałami od 23.1 do 23.5 i pominąć pozostałe podrozdziały.

## 23.1. Zagadnienia z obszaru rozproszonych baz danych

Zdefiniujemy **rozproszoną bazę danych** jako zbiór wielu wzajemnie logicznie powiązanych **baz danych rozproszonych** w sieci komputerowej, a **rozproszony system zarządzania bazą danych** jako oprogramowanie, które pozwala na **zarządzanie rozproszoną bazą danych** tak, że fakt rozproszenia danych nie jest widoczny dla użytkownika.

### 23.1.1. Co sprawia, że baza danych jest rozproszona?

Aby bazę można było nazwać rozproszoną, powinny być spełnione następujące warunki minimalne:

- **Połączenie węzłów bazy danych siecią komputerową.** Jest wiele komputerów nazywanych **jednostkami** lub **węzłami**. Muszą one być powiązane **siecią** przekazującą dane i polecenia między jednostkami.
- **Logiczne zależności między powiązаныmi bazami.** Konieczne jest, aby informacje z różnych węzłów bazy danych były logicznie ze sobą powiązane.
- **Możliwa różnorodność powiązanych węzłów.** Nie jest konieczne, aby wszystkie węzły były identyczne ze względu na dane, sprzęt i oprogramowanie.

Poszczególne węzły mogą być fizycznie położone niedaleko siebie — na przykład w tym samym budynku lub w grupie sąsiadujących budynków — i połączone **lokalną siecią komputerową**, lub mogą być rozproszone na dużych obszarach i połączone **siecią rozległą**. Lokalne sieci komputerowe zazwyczaj używają koncentratorów bezprzewodowych lub kabli sieciowych, a sieci rozległe — łączy satelitarnych, kablowych lub telefonicznych albo infrastruktury bezprzewodowej. Możliwe jest również łączenie różnych rodzajów sieci.

Sieci mogą mieć różne **topologie** określające połączenia pomiędzy wybranymi węzłami. Rodzaj i topologia sieci mogą mieć istotny wpływ na wydajność, a co za tym idzie — wybór strategii rozproszonego przetwarzania zapytań i projekt całego systemu bazodanowego. Nie ma to jednak znaczenia z punktu widzenia architektury wysokopoziomowej,

istotne jest jedynie, by każdy węzeł miał możliwość komunikacji (bezpośrednio lub pośrednio) z innymi węzłami systemu. W pozostałej części tego rozdziału przyjmiemy, że pomiędzy węzłami istnieje jakiś rodzaj komunikacji (używana topologia nie będzie istotna). Nie będziemy wdawać się w szczegóły sieciowe, choć należy cały czas pamiętać, że projekt sieci i kwestie związane z wydajnością mają bardzo istotny wpływ na sprawne działanie rozproszonego systemu bazy danych i są integralną częścią całego rozwiązania. Szczegóły dotyczące pracy sieci są niewidoczne dla użytkownika końcowego.

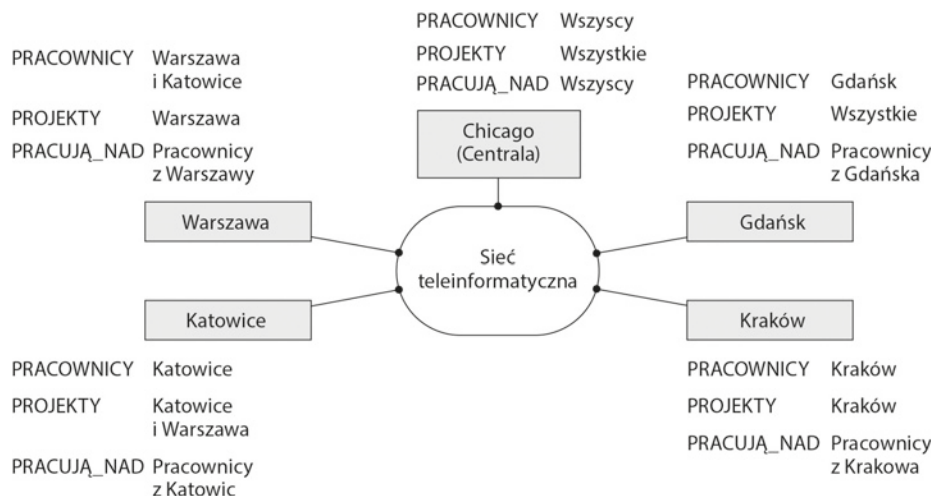
## 23.1.2. Przezroczystość

Zagadnienie przezroczystości to rozwinięcie ogólnej zasady ukrywania szczegółów implementacji przed użytkownikami końcowymi. Wysoce przezroczysty system zapewnia dużą swobodę użytkownikom końcowym i programistom aplikacji, ponieważ wymaga niewielkiej (lub zerowej) znajomości szczegółów. W tradycyjnych scentralizowanych bazach danych przezroczystość dotyczy logicznej i fizycznej niezależności danych z perspektywy programistów aplikacji. W bazach rozproszonych dane i oprogramowanie są rozdzielane między wiele węzłów połączonych siecią komputerową. Dlatego występują tu dodatkowe rodzaje przezroczystości.

Rozważ omawianą w książce bazę FIRMA z rysunku 5.5. Tabele PRACOWNIK, PROJEKT i PRACUJE\_↪NAD mogą być podzielone poziomo (czyli na zbiory wierszy, co omówimy w podrozdziale 23.2) i przechowywane z użyciem replikacji (patrz rysunek 23.1). Występują następujące rodzaje przezroczystości:

- **Przezroczystość organizacji danych (nazywana też przezroczystością sieci lub rozproszenia).** Odnosi się do całkowitego uniezależnienia użytkownika od szczegółów operacyjnych sieci i lokalizacji danych w systemie rozproszonym. Można ją dalej podzielić na przezroczystość miejsca i przezroczystość nazw. **Przezroczystość miejsca** polega na tym, że komenda wykonująca na danych określone czynności jest niezależna od miejsca przechowywania tych danych oraz węzła, za pomocą którego ją uruchomiono. **Przezroczystość nazw** polega na tym, że po powiązaniu nazwy z obiektem możliwy jest jednoznaczny dostęp do nazwanych obiektów bez konieczności dookreślania lokalizacji danych.
- **Przezroczystość replikacji.** Jak pokazano na rysunku 23.1, kopie danych mogą być przechowywane w wielu miejscach równocześnie, co zwiększa ich dostępność, wydajność i niezawodność. Przezroczystość replikacji polega na synchronizacji tych kopii bez wiedzy i udziału użytkownika.
- **Przezroczystość fragmentacji.** Istnieją dwa rodzaje fragmentacji. **Fragmentacja pozioma** rozprasza relacje (tabele) w kilka zestawów krotek (wierszy). W nowszych systemach przetwarzania big data i w rozwiązaniach opartych na chmurze ta technika jest nazywana **shardingiem**. **Fragmentacja pionowa** rozprasza relacje w kilka podrelacji, a każda z nich jest zdefiniowana jako podzbiór kolumn oryginalnej relacji. Ogólne zapytanie użytkownika musi być w tej sytuacji podzielone na kilka zapytań fragmentarycznych. Przy zachowaniu przezroczystości fragmentacji użytkownik nie jest świadomy istnienia tych fragmentów.
- Inne rodzaje przezroczystości to **przezroczystość projektu i wykonania**. Związane są one z tym, że nie trzeba wiedzieć, jak rozproszona baza danych jest zaprojektowana i gdzie transakcje są wykonywane.





RYSUNEK 23.1. Rozpraszanie i replikacja danych w rozproszonych bazach danych

### 23.1.3. Stabilność i dostępność

*Stabilność i dostępność* to dwie najczęściej wymieniane zalety rozproszonych baz danych. **Stabilność** jest ogólnie określana jako prawdopodobieństwo, że system w określonym momencie działa, a **dostępność** jest prawdopodobieństwem, że system jest stale dostępny w danym okresie czasu. Stabilność i dostępność bazy danych można bezpośrednio połączyć z powiązanymi z nią awariami, błędami i usterkami. **Awaria** (ang. *failure*) to odchylenie działania systemu od tego, które ma zapewniać poprawne wykonywanie operacji. **Błędy** (ang. *errors*) to podzbiór stanów systemu powodujących awarię. **Usterka** (ang. *fault*) to przyczyna błędu.

Aby zbudować stabilny system, można zastosować kilka podejść. W jednym z często wykorzystywanych podkreślana jest *odporność na usterki*. Uwzględnia się wtedy, że usterki będą występować, dlatego trzeba projektować mechanizmy, które potrafią wykrywać i eliminować usterki, zanim doprowadzą one do awarii systemu. Inne, bardziej rygorystyczne podejście to próba zapewnienia, że gotowy system nie będzie zawierał usterek. Wymaga to rozbudowanego procesu projektowania, po którym następują szczegółowa kontrola jakości i testy. Stabilny rozproszony SZBD jest odporny na awarie używanych komponentów i przetwarzania żądania użytkownika dopóty, dopóki spójność bazy nie jest naruszona. Menedżer odtwarzania rozproszonego SZBD musi sobie radzić z awariami w transakcjach, sprzęcie i sieciach komunikacyjnych. Awarie sprzętowe mogą skutkować utratą zawartości pamięci głównej lub utratą zawartości drugorzędnych mechanizmów składowania danych. Awarie sieci są powodowane błędami w obszarze komunikatów lub linii przesyłowych. Błędy dotyczące komunikatów mogą polegać na ich utracie, uszkodzeniu lub docieraniu do celu w niewłaściwej kolejności.

Opisane tu definicje są stosowane w kontekście systemów komputerowych, gdzie występuje rozróżnienie na stabilność i dostępność. W większości prac z obszaru rozproszonych baz danych zamiast tych dwóch pojęć używane jest zwykle zbiorcze określenie **dostępność**.

### 23.1.4. Skalowalność i odporność na podział

**Skalowalność** oznacza zakres, w jakim system może powiększać swoją przepustowość, wciąż działając bez zakłóceń. Są dwa rodzaje skalowalności:

- (1) **Skalowalność pozioma.** Dotyczy ona zwiększania liczby węzłów w systemie rozproszonym. Gdy węzły są dodawane do systemu, możliwe powinno być przekazywanie części danych i obciążenia roboczego z istniejących węzłów do nowych.
- (2) **Skalowalność pionowa.** Związana jest ze zwiększaniem przepustowości pojedynczych węzłów w systemie — np. powiększaniem pojemności pamięci masowej lub mocy obliczeniowej.

Gdy w systemie zwiększana jest liczba węzłów, możliwe, że w łączącej je sieci wystąpią błędy, co spowoduje podział węzłów na grupy. Węzły w każdej grupie są powiązane podsiecią, jednak komunikacja między grupami nie istnieje. **Odporność na podział** oznacza, że system powinien zachować możliwość pracy po podziale sieci.

### 23.1.5. Autonomia

**Autonomia** określa zakres, w jakim poszczególne węzły lub bazy w rozproszonej bazie danych mogą działać niezależnie od siebie. Wysoki poziom autonomii jest pożądanym ze względu na większą elastyczność i możliwość niestandardowej konserwacji poszczególnych węzłów. Autonomia może występować w obszarach projektowania, komunikacji i wykonania. **Autonomia projektowa** dotyczy niezależności używanych modeli danych i technik zarządzania transakcjami w poszczególnych węzłach. **Autonomia komunikacji** określa zakres, w jakim każdy węzeł może decydować o wymianie informacji z innymi węzłami. **Autonomia wykonania** polega na tym, że użytkownicy mogą wykonywać dowolne działania.

### 23.1.6. Zalety rozproszonych baz danych

Poniżej przedstawiono niektóre z głównych zalet tego typu rozwiązań:

- (1) **Ułatwienie i zwiększona elastyczność budowania aplikacji.** Rozwijanie i konserwacja aplikacji w geograficznie rozproszonych placówkach organizacji są łatwiejsze dzięki przezroczystości dystrybucji danych i kontroli nad nimi.
- (2) **Wyższa dostępność.** Osiąga się ją dzięki ograniczeniu usterek do jednostki będącej ich źródłem, bez wpływania na inne węzły bazodanowe podłączone do sieci. Gdy dane i oprogramowanie rozproszonego SZBD są rozdzielone między wiele węzłów, jeden węzeł może ulec awarii, a inne nadal będą działać. Niemożliwy będzie tylko dostęp do danych i oprogramowania z niesprawnego węzła. Dodatkową poprawę można osiągnąć, rozsądnie replikując dane i oprogramowanie w więcej niż jednym węźle. W scentralizowanym systemie awaria w jednym węźle sprawia, że system staje się niedostępny dla wszystkich użytkowników. W bazie rozproszonej niektóre dane mogą być niedostępne, jednak użytkownicy mają dostęp do innych fragmentów bazy. Jeśli dane z niesprawnego węzła zostały przed awarią zreplikowane do innego węzła, użytkownik w ogóle nie odczuje problemu. Na dostępność wpływa też to, czy system potrafi przetrwać podział sieci.

- (3) **Wzrost wydajności.** Rozproszony system baz danych dzieli bazę danych, przechowując dane bliżej miejsca, gdzie są potrzebne. **Używanie danych lokalnych** redukuje zapotrzebowanie na moc obliczeniową i operacje wejścia/wyjścia, zmniejszając równocześnie opóźnienia w dostępie do danych spowodowane ich przesyłaniem przez sieć rozległą. Kiedy duża baza danych jest rozpraszana między wiele oddziałów, w każdym z nich tworzona jest mała baza danych. W wyniku tej operacji lokalne zapytania i transakcje dotyczą mniejszych baz danych i są przez to obsługiwane szybciej. Ponadto, każdy oddział przetwarza w danym momencie mniej transakcji niż w przypadku wykonywania ich wszystkich w jednej centralnej bazie danych. Co więcej, równoległe przetwarzanie pojedynczych zapytań i ich grup poprzez wykonywanie zapytań w różnych miejscach lub równoczesną obsługę kilku podzapytań prowadzi do dalszego polepszenia wydajności.
- (4) **Łatwiejsze rozszerzanie dzięki skalowalności.** W środowisku rozproszonym rozwój systemu w sensie dodawania nowych danych, zwiększania objętości baz danych czy dodawania nowych procesorów jest znacznie ułatwiony w porównaniu z systemami scentralizowanymi (nierozproszonymi).

Przezroczystość przedstawiona w punkcie 23.1.2 wiąże się z koniecznością znalezienia kompromisu pomiędzy wygodą używania a kosztami związanymi z jej zachowaniem. Pełna przezroczystość daje globalnemu użytkownikowi możliwość korzystania z rozproszonego systemu bazy danych tak, jakby był to jeden scentralizowany system. Przezroczystość jest dopełnieniem **autonomii**, która pozwala użytkownikom na ściślejszą kontrolę nad ich lokalnymi bazami danych. Funkcje przezroczystości mogą być zaimplementowane jako część języka użytkownika, który może przełożyć wymagane usługi na określone operacje.

## 23.2. Techniki fragmentacji, replikacji i alokacji danych w projekcie rozproszonej bazy danych

W tym podrozdziale omówimy techniki używane do dzielenia bazy danych na logiczne jednostki zwane **fragmentami**, które mogą być przeznaczone do składowania danych w różnych węzłach. Omówimy także zastosowanie **replikacji danych**, która pozwala na przechowywanie określonych danych w więcej niż jednym miejscu naraz (w celu poprawy dostępności i stabilności), oraz proces **alokacji** fragmentów — lub replik fragmentów — w różnych węzłach. Techniki te są używane podczas procesu **projektowania rozproszonej bazy danych**. Informacja dotycząca fragmentacji, alokacji i replikacji danych jest przechowywana w **centralnym katalogu**, który jest używany w miarę potrzeby przez system zarządzający bazą danych.

### 23.2.1. Fragmentacja danych i sharding

Istotną decyzją w DDB jest określenie, w którym węzle mają być zapisane poszczególne fragmenty bazy danych. Na razie założymy, że *nie stosujemy replikacji*, czyli każda relacja (lub część relacji) jest przechowywana tylko w jednym węźle. Replikację i jej konsekwencje

omówimy później. Będziemy również używali terminologii relacyjnych systemów baz danych; podobne pojęcia mają zastosowanie również w innych modelach. Zaczniemy od relacyjnego schematu bazy danych i potrzeby rozproszenia relacji po różnych węzłach systemu. Do zilustrowania rozważań użyjemy dobrze znanego schematu z rysunku 5.5.

Zanim zdecydujemy, jak rozproszyć dane, musimy określić *logiczne jednostki* rozpraszanej bazy danych. Najprostsze jednostki to całe relacje; każda relacja będzie w tym przypadku w *całości* zapisana w określonym węźle. W naszym przykładzie trzeba zdecydować o przydziale do węzła każdej z relacji PRACOWNIK, DZIAŁ, PROJEKT, PRACUJE\_NAD i CZŁONEK\_RODZINY z rysunku 5.5. W wielu przypadkach relacja może być jednak podzielona na mniejsze jednostki. Dla przykładu rozważmy firmową bazę danych z rysunku 5.6 i przyjmijmy istnienie trzech węzłów informatycznych — po jednym dla każdego działu<sup>1</sup>.

Możemy przechowywać dane powiązane z każdym działem w odpowiadającym mu węźle. Do podzielenia danych z każdej relacji można wówczas użyć techniki nazywanej *fragmentacją poziomą* (inna jej nazwa to *sharding*).

**Fragmentacja pozioma (sharding).** Fragment poziomy relacji jest podzbiorem krotek tej relacji. Krotki należące do wybranego fragmentu poziomego są określane przez warunek dotyczący jednego lub kilku atrybutów relacji albo inny mechanizm. Często stosuje się wybór na podstawie tylko jednego atrybutu. Na przykład, możemy określić fragmenty poziome relacji PRACOWNIK z rysunku 5.6 na podstawie następujących warunków: (NRDZ=5), (NRDZ=4) i (NRDZ=1). Każdy fragment zawiera krotki odpowiadające pracownikom przydzielonym do odpowiedniego działu. W podobny sposób można określić fragmenty poziome dla relacji PROJEKT warunkami (NR\_DZ=5), (NR\_DZ=4) i (NR\_DZ=1); każdy fragment zawiera tutaj krotki dotyczące projektów nadzorowanych przez określony dział. **Fragmentacja pozioma** dzieli relację *poziomo*, grupując wiersze w podzbiory krotek i przypisując każdemu podzbiorowi pewne logiczne znaczenie. Każdy z tych fragmentów może być później przypisany innym węzłom w rozproszonym SZBD. **Pochodna fragmentacja pozioma** polega na zastosowaniu podziału pierwszej relacji (w naszym przykładzie — DZIAŁ) do podziału dalszych relacji (PRACOWNIK i PROJEKT), powiązanych z pierwszą poprzez klucz obcy. W ten sposób powiązane dane z pierwszej i drugiej relacji są fragmentowane w ten sam sposób.

**Fragmentacja pionowa.** Każdy z węzłów może potrzebować tylko niektórych atrybutów relacji, co prowadzi do potrzeby innego podzielenia danych. **Fragmentacja pionowa** dzieli relację „pionowo” według kolumn. **Fragment pionowy** relacji przechowuje tylko niektóre z jej atrybutów. Możemy na przykład chcieć podzielić relację PRACOWNIK na dwa fragmenty. Pierwszy z nich będzie zawierał dane osobowe (NAZWISKO, DATAUR, ADRES, PŁEĆ), a drugi — informacje związane z pracą (PESEL, PENSJA, PESELPRZEŁOŻ, NRDZ). Taka fragmentacja nie jest do końca prawidłowa, ponieważ jeżeli oba fragmenty będą przechowywane osobno, to nie ma możliwości złożenia krotek z powrotem w jedną całość, bowiem fragmenty te nie mają *wspólnego atrybutu*. W każdym fragmencie pionowym trzeba umieścić klucz główny lub atrybut klucza kandydującego, co pozwoli na zrekonstruowanie oryginalnej relacji. Musimy zatem do pierwszego przykładowego fragmentu dodać atrybut PESEL.

Zauważmy, że każdy fragment poziomy relacji  $R$  może być określony w algebrze relacyjnej jako operacja  $\sigma_{C_i}(R)$ . Zestaw fragmentów poziomych zgodnych z warunkami  $C_1, C_2, \dots, C_n$  zawiera wszystkie krotki w  $R$  — czyli każda krotka w  $R$  spełnia ( $C_1 \text{ OR } C_2 \text{ OR } \dots \text{ OR } C_n$ ).

<sup>1</sup> Oczywiście w rzeczywistości może istnieć znacznie więcej krotek, niż pokazano na rysunku 5.6.

... OR  $C_n$ ) — jest więc nazywany **całkowitą fragmentacją poziomą**  $R$ . W wielu przypadkach całkowita fragmentacja jest również **rozłączna**, czyli nie istnieje krotka w  $R$  spełniająca  $(C_i \text{ AND } C_j)$  dla dowolnych  $i \neq j$ . Wcześniejsze przykłady fragmentacji relacji PRACOWNIK i PROJEKT były zarówno całkowite, jak i rozłączne. Aby odtworzyć oryginalną relację  $R$  z *całkowitej* fragmentacji poziomej, należy zastosować do fragmentów operację sumowania (UNION).

Fragmentacja pionowa relacji  $R$  może być określona w algebrze relacyjnej jako operacja  $\pi_{L_i}(R)$ . Zbiór fragmentów pionowych, których projekcje  $L_1, L_2, \dots, L_n$  zawierają wszystkie atrybuty  $R$ , ale współdzielą jedynie atrybut klucza głównego, jest nazywany **całkowitą fragmentacją pionową**  $R$ . W tym przypadku lista projekcji spełnia następujące dwa warunki:

- $L_1 \cup L_2 \cup \dots \cup L_n = \text{ATTRS}(R)$
- $L_i \cap L_j = \text{PK}(R)$  dla każdego  $i \neq j$ , gdzie  $\text{ATTRS}(R)$  jest zbiorem wszystkich atrybutów  $R$  a  $\text{PK}(R)$  jest kluczem głównym  $R$ .

Aby zrekonstruować relację  $R$  z *całkowitej* fragmentacji pionowej, stosujemy do wszystkich fragmentów operację OUTER UNION (o ile nie była użyta równocześnie fragmentacja pozioma). Zauważmy, że można również zastosować operację FULL OUTER JOIN i uzyskać ten sam rezultat, nawet jeśli została równocześnie użyta fragmentacja pozioma. Dwa fragmenty pionowe relacji PRACOWNIK z projekcjami  $L_1 = \{\text{PESEL}, \text{NAZWISKO}, \text{DATAUR}, \text{ADRES}, \text{PŁEĆ}\}$  i  $L_2 = \{\text{PESEL}, \text{PENSJA}, \text{PESELPRZEŁOŻ}, \text{NRDZ}\}$  stanowią całkowitą fragmentację pionową relacji.

Dwa fragmenty poziome, które nie stanowią fragmentacji całkowitej ani nie są rozłączne, mogą być określone dla relacji PRACOWNIK z rysunku 5.5 warunkami  $(\text{PENSJA} > 5000)$  i  $(\text{NRDZ} = 4)$ . Fragmenty te mogą nie zawierać wszystkich krotek relacji i mogą mieć niektóre z nich wspólne. Dwa pionowe fragmenty, które nie stanowią fragmentacji całkowitej, mogą być określone przez listy atrybutów  $L_1 = \{\text{NAZWISKO}, \text{ADRES}\}$  i  $L_2 = \{\text{PESEL}, \text{NAZWISKO}, \text{PENSJA}\}$ ; listy te naruszają oba warunki całkowitej fragmentacji pionowej.

**Fragmentacja mieszana (hybrydowa).** Można mieszać oba rodzaje fragmentacji, tworząc **fragmentację mieszaną**. Na przykład, można połączyć podaną wcześniej przykładową fragmentację poziomą z pionową w relacji PRACOWNIK, tworząc sześć fragmentów. W tym przypadku oryginalna relacja jest rekonstruowana przez zastosowanie w odpowiedniej kolejności operacji UNION i OUTER UNION (lub OUTER JOIN). Ogólnie, **fragment** relacji  $R$  może być określony jako kombinacja typu SELECT-PROJECT operacji  $\pi_L(\sigma_C(R))$ . Jeśli  $C = \text{TRUE}$  (czyli wybrane są wszystkie krotki) i  $L \neq \text{ATTRS}(R)$ , otrzymujemy fragment pionowy; jeśli zaś  $C \neq \text{TRUE}$  i  $L = \text{ATTRS}(R)$ , to otrzymujemy fragment poziomy. Jeśli  $C \neq \text{TRUE}$  i  $L \neq \text{ATTRS}(R)$ , to otrzymujemy fragment mieszany. Zauważmy, że cała relacja może być również traktowana jako fragment o  $C = \text{TRUE}$  i  $L = \text{ATTRS}(R)$ . W dalszych rozważaniach termin *fragment* będzie używany w stosunku do całej relacji lub dowolnej jej części.

**Schemat fragmentacji** bazy danych stanowi definicję zbioru fragmentów zawierających *wszystkie* atrybuty i krotki bazy danych oraz spełnia warunek mówiący, że kompletna baza danych może być stworzona z tych fragmentów poprzez odpowiednie zastosowanie ciągu operacji OUTER UNION (lub OUTER JOIN) i UNION. W niektórych przypadkach użyteczną (ale nie niezbędną) własnością schematu fragmentacji jest rozłączność wszystkich frag-

mentów, z wyjątkiem klucza głównego powtórzonego we fragmentach pionowych (lub mieszanych). W przypadku fragmentacji hybrydowej, replikacja i rozproszenie fragmentów jest określone później, niezależnie od fragmentacji.

**Schemat alokacji** opisuje alokację fragmentów w węzłach rozproszonej bazy danych. Jest to zatem odwzorowanie określające dla każdego fragmentu węzeł (lub węzły), w którym ma on być przechowywany. Jeżeli fragment jest zapisany w więcej niż jednym węźle, to mówimy, że jest **replikowany**. Kwestie replikacji i alokacji są przedstawione w następnym punkcie.

### 23.2.2. Replikacja i alokacja danych

Replikacja zwiększa dostępność danych. Skrajnym przypadkiem replikacji jest przechowywanie kopii *całej bazy danych* w każdym węźle systemu rozproszonego, tworząc **w pełni replikowaną rozproszoną bazę danych**. Istotnie zwiększa to dostępność, ponieważ cały system pracuje poprawnie, dopóki choć jeden węzeł jest sprawny. Zwiększa to również wydajność znajdowania wyników zapytań globalnych (wydajność odczytu), ponieważ są one przechowywane w lokalnej replice, a zatem wszystkie zapytania mogą być obsługiwane przez lokalny węzeł (o ile ten węzeł zawiera moduł serwera). Wadą pełnej replikacji jest drastyczne spowolnienie przeprowadzania aktualizacji danych, ponieważ dla zachowania spójności jedna logiczna zmiana musi zostać zapisana we *wszystkich kopiach*. Spowolnienie jest szczególnie uciążliwe, jeśli istnieje wiele kopii bazy. Pełna replikacja zwiększa również koszt technik sterowania współbieżnego i odtwarzania danych, co zostanie pokazane w podrozdziale 23.5.

Drugą skrajnością, przeciwną do pełnej replikacji, jest **brak replikacji**, który oznacza, że każdy fragment jest zapisany dokładnie w jednym węźle. W tym przypadku wszystkie fragmenty *muszą* być rozłączne (za wyjątkiem kluczy głównych we fragmentach pionowych lub mieszanych). Taka sytuacja jest również nazywana **alokacją nieredundantną**.

Pomiędzy tymi dwiema skrajnościami istnieje szerokie spektrum **częściowej replikacji** danych, w przypadku której niektóre fragmenty bazy danych są replikowane, a inne nie. Liczba kopii każdego fragmentu może się wahać od jednej do całkowitej liczby wszystkich węzłów w systemie rozproszonym. Szczególny przypadek częściowej replikacji jest stosowany w przypadku pracowników mobilnych, takich jak handlowcy, planiści finansowi czy inspektorzy ubezpieczeniowi, którzy przenoszą replikowaną bazę danych ze sobą na urządzeniach przenośnych typu PDA (ang. *Personal Digital Assistant*) lub laptopach i synchronizują ją okresowo z serwerem bazy danych. Opis replikacji poszczególnych fragmentów jest czasami nazywany **schematem replikacji**.

Każdy fragment (lub każda kopia fragmentu) musi być przypisany do określonego węzła w systemie rozproszonym. Proces ten nazywa się **rozpraszaniem danych** lub **alokacją danych**. Wybór węzłów i stopień replikacji zależy od wymaganej wydajności i dostępności systemu oraz rodzaju i częstotliwości transakcji wykonywanych przez każdy z węzłów. Jeżeli na przykład wymagana jest wysoka dostępność systemu, transakcje mogą być wykonywane przez każdy węzeł i większość transakcji polega na odczytywaniu danych, to dobrym rozwiązaniem będzie pełna replikacja. Z drugiej strony, jeżeli transakcje używające określonych części bazy danych są głównie wykonywane przez jeden węzeł, to warto umieścić odpowiednie fragmenty tylko w tym węźle. Dane używane w kilku węzłach



dobrze jest umieścić właśnie w nich, pamiętając jednak, że w przypadku dokonywania wielu aktualizacji danych warto ograniczyć replikację. Znalezienie optymalnego, a przynajmniej dobrego rozwiązania problemu alokacji rozproszonych danych jest złożonym problemem optymalizacyjnym.

### 23.2.3. Przykłady fragmentacji, alokacji i replikacji danych

Rozważmy teraz przykład fragmentacji i rozproszenia firmowej bazy danych z rysunku 5.5 i 5.6. Przypuśćmy, że firma ma trzy ośrodki komputerowe — po jednym dla każdego działu. Węzły 2. i 3. odpowiadają działom 5. i 4. W każdym z tych węzłów oczekiwane są częste operacje odczytujące z relacji PRACOWNIK i PROJEKT dane pracowników *pracujących w działach* ulokowanych w określonych węzłach i informacje o projektach *nadzorowanych przez te działy*. Dalej, przyjmijmy, że odczytywanymi atrybutami pracowników są głównie NAZWISKO, PESEL, PENSJA i PESELPRZEŁOŻ. Węzeł 1. odpowiada centralnej siedzibie firmy i używa regularnie wszystkich informacji o pracownikach i projektach, a dodatkowo przechowuje informacje o członkach rodziny dla celów ubezpieczeniowych.

Stosując się do powyższych wymagań, cała baza danych z rysunku 5.6 może być przechowywana w węźle 1. Aby określić fragmenty, które powinny być replikowane w węzłach 2. i 3., posłużymy się najpierw fragmentacją poziomą relacji DZIAŁ według atrybutu NUMERDZ. Następnie zastosujemy fragmentację pochodną do relacji PRACOWNIK, PROJEKT i LOKALIZACJE\_DZIAŁÓW w oparciu o klucze obce wskazujące numer działu, czyli odpowiednio o atrybuty NRDZ, NR\_DZ i NUMERDZ. Możemy następnie przeprowadzić fragmentację pionową uzyskanych fragmentów relacji PRACOWNIK, pobierając jedynie atrybuty {NAZWISKO, PESEL, PENSJA, PESELPRZEŁOŻ, NRDZ}. Rysunek 23.2 pokazuje mieszane fragmenty PRACD5 i PRACD4, które zawierają krotki relacji PRACOWNIK spełniające odpowiednio warunki  $NRDZ=5$  i  $NRDZ=4$ . Poziome fragmenty relacji PROJEKT, DZIAŁ i LOKALIZACJE\_DZIAŁÓW są podzielone podobnie według numeru działu. Wszystkie te fragmenty zapisywane w węzłach 2. i 3. są fragmentami replikowanymi, ponieważ są również zapisane w centralnym węźle 1.

Należy teraz dokonać fragmentacji relacji PRACUJE\_NAD i zdecydować, które jej fragmenty mają być zapisane w węźle 2., a które w 3. Stajemy w tym kroku wobec problemu braku atrybutu relacji PRACUJE\_NAD wskazującego bezpośrednio przynależność krotki do określonego działu. Każda krotka PRACUJE\_NAD wiąże jedynie pracownika  $e$  z projektem  $p$ . Można podzielić PRACUJE\_NAD w oparciu o dział  $d$ , w którym  $e$  pracuje, lub w oparciu o dział  $d'$ , który nadzoruje  $p$ . Fragmentacja jest prosta, jeżeli przestrzegane jest ograniczenie  $d = d'$  dla wszystkich krotek PRACUJE\_NAD, czyli jeśli pracownicy pracują tylko nad projektami nadzorowanymi przez ich działy. Takiego ograniczenia nie ma jednak na rysunku 5.6. Na przykład, krotka PRACUJE\_NAD  $\langle 55120834598, 10, 10.0 \rangle$  wiąże pracownika pracującego w dziale nr 5 z projektem nadzorowanym przez dział nr 4. W tym przypadku można podzielić PRACUJE\_NAD według działu, w którym pracuje pracownik (co jest wyrażone warunkiem C), a następnie przeprowadzić dalszą fragmentację w oparciu o numer działu nadzorującego projekt. Fragmentację tej relacji przedstawiono na rysunku 23.3.

Na rysunku 23.3 suma fragmentów G1, G2 i G3 daje wszystkie krotki PRACUJE\_NAD dla pracowników pracujących w dziale 5. Podobnie, suma fragmentów G4, G5 i G6 daje wszystkie krotki dla pracowników działu 4. Z drugiej strony, suma fragmentów G1, G4 i G7 daje wszystkie krotki projektów nadzorowanych przez dział 5. Warunki dla wszystkich



(a) PRACD5

IMIĘ	INICJAŁ DRIMIENIA	NAZWISKO	PESEL	PENSJA	PESELPRZEŁOŻ	NRDZIAŁU
Jan	B	Szewczyk	65010912345	3000	55120834598	5
Fryderyk	T	Wieszczycki	55120834598	4000	37111045873	5
Robert	K	Napierski	62091502054	3800	55120834598	5
Joanna	A	Englert	72073110039	2500	55120834598	5

DZIAŁ5

NAZWADZ	NUMERDZ	PESELKIEROWNIKA	DATAPRZEJKIEROWNICTWA
Badania	5	55120834598	1988-05-22

LOK\_DZ5

NUMERDZ	LOKALIZACJA
5	Kielce
5	Kraków
5	Warszawa

PRACUJE\_NAD5

PESELPRAC	NR_PROJ	GODZINY
65010912345	1	32.5
65010912345	2	7.5
62091502054	3	40.0
72073110039	1	20.0
72073110039	2	20.0
55120834598	2	10.0
55120834598	3	10.0
55120834598	10	10.0
55120834598	20	10.0

PROJ5

NAZWAPROJ	NUMERPROJ	LOKALIZACJAPROJ	NR_DZ
Produkt X	1	Kielce	5
Produkt Y	2	Kraków	5
Produkt Z	3	Warszawa	5

Dane w węźle 2

(b) PRACD4

IMIĘ	INICJAŁ DRIMIENIA	NAZWISKO	PESEL	PENSJA	PESELPRZEŁOŻ	NRDZIAŁU
Alicja	J	Zielińska	68011932514	2500	41062013258	4
Maria	S	Owsiak	41062013258	4300	37111045873	4
Adam	K	Głębocki	62091502054	2500	41062013258	4

DZIAŁ4

NAZWADZ	NUMERDZ	PESELKIEROWNIKA	DATAPRZEJKIEROWNICTWA
Administracja	4	41062013258	1995-01-01

LOK\_DZ4

NUMERDZ	LOKALIZACJA
4	Gliwice

PRACUJE\_NAD4

PESELPRAC	NR_PROJ	GODZINY
55120834598	10	10.0
68011932514	30	30.0
68011932514	10	10.0
69032923149	10	35.0
69032923149	30	5.0
41062013258	30	20.0
41062013258	20	15.0

PROJ4

NAZWAPROJ	NUMERPROJ	LOKALIZACJAPROJ	NR_DZ
Komputeryzacja	10	Gliwice	4
Zwiększenie Zysków	30	Gliwice	4

Dane w węźle 3

Rysunek 23.2. Alokacja fragmentów w węzłach. (a) Fragmenty relacji w węźle 2. odpowiadają działowi 5. (b) Fragmenty relacji w węźle 3. odpowiadają działowi 4.

fragmentów G1 do G9 są pokazane na rysunku 23.3. Relacje reprezentujące związki typu *M:N* (takie jak PRACUJE\_NAD) często mają kilka możliwych schematów fragmentacji. W rozproszeniu z rysunku 23.2 wybrano zapisywanie w węzłach 2. i 3. wszystkich krotek, które mogą być powiązane z krotkami relacji PRACOWNIK bądź PROJEKT. Zatem umieszczono sumę fragmentów G1, G2, G3, G4 i G7 w węźle 2., a sumę fragmentów G4, G5, G6, G2 i G8 w węźle 3. Zauważmy, że fragmenty G2 i G4 są replikowane w obu węzłach. Taka strategia alokacji pozwala na złączenia pomiędzy fragmentami relacji PRACOWNIK, PROJEKT i PRACUJE\_NAD lokalnie w węzłach 2. i 3. Przytoczony przykład pokazuje jasno, jak złożonym problemem może być fragmentacja i alokacja w dużych bazach danych. Wybrane publikacje pod koniec rozdziału wymieniają niektóre prace dotyczące tej problematyki.

(a) Pracownicy działu 5

G1

PESELPRAC	NR_PROJ	GODZINY
65010912345	1	32.5
65010912345	2	7.5
62091502054	3	40.0
72073110039	1	20.0
72073110039	2	20.0
55120834598	2	10.0
55120834598	3	10.0

C1 = C AND (NR\_PROJ IN (SELECT  
NUMERPROJ FROM PROJEKT  
WHERE NR\_DZ = 5))

G2

PESELPRAC	NR_PROJ	GODZINY
55120834598	10	10.0

C2 = C AND (NR\_PROJ IN (SELECT  
NUMERPROJ FROM PROJEKT  
WHERE NR\_DZ = 4))

G3

PESELPRAC	NR_PROJ	GODZINY
55120834598	20	10.0

C3 = C AND (NR\_PROJ IN (SELECT  
NUMERPROJ FROM PROJEKT  
WHERE NR\_DZ = 1))

(b) Pracownicy działu 4

G4

PESELPRAC	NR_PROJ	GODZINY
-----------	---------	---------

C4 = C AND (NR\_PROJ IN (SELECT  
NUMERPROJ FROM PROJEKT  
WHERE NR\_DZ = 5))

G5

PESELPRAC	NR_PROJ	GODZINY
68011932514	30	30.0
68011932514	10	10.0
69032923149	10	35.0
69032923149	30	5.0
41062013258	30	20.0

C5 = C AND (NR\_PROJ IN (SELECT  
NUMERPROJ FROM PROJEKT  
WHERE NR\_DZ = 4))

G6

PESELPRAC	NR_PROJ	GODZINY
41062013258	20	15.0

C6 = C AND (NR\_PROJ IN (SELECT  
NUMERPROJ FROM PROJEKT  
WHERE NR\_DZ = 1))

(c) Pracownicy działu 1

G7

PESELPRAC	NR_PROJ	GODZINY
-----------	---------	---------

C7 = C AND (NR\_PROJ IN (SELECT  
NUMERPROJ FROM PROJEKT  
WHERE NR\_DZ = 5))

G8

PESELPRAC	NR_PROJ	GODZINY
-----------	---------	---------

C8 = C AND (NR\_PROJ IN (SELECT  
NUMERPROJ FROM PROJEKT  
WHERE NR\_DZ = 4))

G9

PESELPRAC	NR_PROJ	GODZINY
37111045873	20	Null

C9 = C AND (NR\_PROJ IN (SELECT  
NUMERPROJ FROM PROJEKT  
WHERE NR\_DZ = 1))

RYSUNEK 23.3. Pełna i rozłączna fragmentacja relacji PRACUJE\_NAD. (a) Fragmenty PRACUJE\_NAD odpowiadające pracownikom działu 5. (C = [PESELPRAC IN (SELECT PESEL FROM PRACOWNIK WHERE NR\_DZ = 5)]). (b) Fragmenty PRACUJE\_NAD odpowiadające pracownikom działu 4. (C = [PESELPRAC IN (SELECT PESEL FROM PRACOWNIK WHERE NR\_DZ = 4)]). (c) Fragmenty PRACUJE\_NAD odpowiadające pracownikom działu 1. (C = [PESELPRAC IN (SELECT PESEL FROM PRACOWNIK WHERE NR\_DZ = 1)])

## 23.3. Techniki sterowania współbieżnego i odtwarzania danych w rozproszonych bazach danych

W kontekście sterowania współbieżnego i odtwarzania danych w rozproszonej bazie danych pojawia się wiele problemów, które nie istnieją w przypadku środowiska scentralizowanego. Są to między innymi:

- **Praca z wieloma kopiami tych samych danych.** Sterowanie współbieżne odpowiada za utrzymanie spójności pomiędzy poszczególnymi kopiami. Metoda odtwarzania danych ma za zadanie przywrócić spójności w sytuacji, gdy jeden z węzłów zawierających zreplikowane dane uległ awarii, a następnie został przywrócony do pracy.
- **Awaryjne poszczególnych węzłów.** W przypadku awarii jednego lub kilku węzłów system rozproszony powinien nadal działać (używając sprawnych węzłów). Węzły przywracane do pracy muszą zostać zsynchronizowane z pozostałymi, zanim zostaną w pełni dołączone do systemu.
- **Awaryjne łączów komunikacyjnych.** System musi mieć możliwość funkcjonowania w przypadku awarii jednego lub kilku połączeń pomiędzy węzłami. Skrajnym przypadkiem tego typu awarii jest **podzielenie sieci**. W takiej sytuacji węzły mogą być podzielone na dwie lub więcej partycji, gdzie węzły mogą się komunikować tylko z innymi węzłami w tej samej partycji.
- **Rozproszone zatwierdzanie.** Awaria jednego z węzłów używanego do wykonania rozproszonej transakcji używającej baz z wielu węzłów może powodować dodatkowe problemy z zatwierdzeniem tej transakcji. Do ich rozwiązywania często używa się omawianego w podrozdziale 21.6 **protokołu zatwierdzania dwufazowego** (ang. *two-phase commit protocol*).
- **Rozproszone zakleszczenie.** Pomiedzy kilkoma węzłami może nastąpić zakleszczenie, rozproszony system baz danych powinien więc posiadać mechanizmy biorące taką możliwość pod uwagę.

Techniki rozproszonego sterowania współbieżnego i odtwarzania danych muszą być dostosowane do obsługi powyższych i innych problemów. W dalszych punktach przedstawimy kilka technik stosowanych w rozproszonych systemach baz danych.

### 23.3.1. Rozproszone sterowanie współbieżne oparte na wyróżnionej kopii danych

Powstało wiele metod sterowania współbieżnego do obsługi replikowanych danych w rozproszonych bazach danych, rozszerzających te opracowane na potrzeby systemów scentralizowanych. Omówimy je w kontekście rozszerzenia scentralizowanego pojęcia *blokad*. Podobne rozszerzenia dotyczą również technik sterowania współbieżnego. Idea polega na wybraniu jednej *konkretnej kopii* jako **wyróżnionej kopii** danych. Blokad określonych danych są powiązane z tą *wyróżnioną kopią*, a wszystkie żądania zablokowania i odblokowania są wysyłane do węzła, w którym jest zapisana.

W oparciu o tę koncepcję powstało wiele różnych metod, ale różnią się one głównie metodą wybierania wyróżnionej repliki. W **technice węzła podstawowego** wszystkie wyróżnione kopie są przechowywane w jednym wyróżnionym węźle. Odmianą tego podejścia jest podstawowy węzeł z **węzłem zapasowym**. Inną koncepcją jest metoda **podstawowej kopii**, w której kopie wyróżnione mogą być zapisywane w różnych węzłach. Węzeł przechowujący taką kopię jest **węzłem koordynującym** w ramach mechanizmu sterowania współbieżnego dotyczącego tych danych. Wszystkie te techniki są dokładniej przedstawione poniżej.

**Technika podstawowego węzła.** W tej metodzie pojedynczy **węzeł podstawowy** jest wybrany jako **węzeł koordynujący dla wszystkich obiektów bazy danych**. Wszystkie blokady są zatem przechowywane w tym właśnie węźle i do niego wysyłane są wszystkie żądania zablokowania i odblokowania danych. Metoda ta jest więc rozwinięciem koncepcji scentralizowanej blokady. Przykładowo, jeżeli wszystkie transakcje stosują dwufazowy protokół blokowania, to zapewniona jest szeregowałość. Zaletą tego podejścia jest fakt, że jest ono prostym rozwinięciem koncepcji scentralizowanej, a zatem nie jest zbyt złożone. Z drugiej strony, wiąże się to z pewnymi wadami. Jedną z nich jest możliwość przeciążenia pojedynczego węzła obsługującego wszystkie blokady całego systemu, co może stworzyć tzw. „wąskie gardło”. Drugą wadą jest możliwość unieruchomienia całego systemu podczas awarii jednego węzła, na którym przechowywane są wszystkie informacje o blokadach. Ogranicza to dostępność i stabilność rozproszonego SZBD.

Pomimo że wszystkie blokady są przechowywane w węźle podstawowym, same dane mogą być odczytywane z dowolnego węzła, w którym są przechowywane. Na przykład transakcja, która uzyskała z podstawowego węzła blokadę do odczytu danych, może odczytać dane z dowolnej repliki. Jeśli jednak transakcja uzyskała blokadę do zapisu danych i aktualizowała dane w jednym z węzłów, to zadaniem systemu rozproszonego jest synchronizacja danych we *wszystkich węzłach* przed zniesieniem blokady.

**Podstawowy węzeł z węzłem zapasowym.** Ta metoda niweluje drugą z wymienionych wad techniki podstawowego węzła poprzez wyznaczenie **węzła zapasowego**. Wszystkie informacje dotyczące blokad są przechowywane zarówno w węźle podstawowym, jak i w węźle zapasowym. W przypadku awarii węzła podstawowego jego rolę przejmuje dotychczasowy zapasowy i wybierany jest nowy węzeł zapasowy. Upraszcza to proces przywracania systemu do pracy po awarii podstawowego węzła, ponieważ działa zapasowy, a system może wrócić do przetwarzania zadań natychmiast po wybraniu nowego węzła zapasowego i skopiowaniu na niego danych o aktualnych blokadach. Spowalnia to jednak proces uzyskiwania blokady, ponieważ informacja o niej musi być *skopiowana z węzła podstawowego na zapasowy* przed wysłaniem potwierdzenia do transakcji. Pozostaje również nierozwiązany problem przeciążenia węzłów podstawowego i zapasowego przetwarzaniem wszystkich danych o blokadach.

**Technika podstawowej kopii.** Ta metoda polega na rozproszeniu obciążenia związanego z zarządzaniem blokadami poprzez wyznaczenie wyróżnionych kopii różnych elementów danych *przechowywanych w różnych węzłach*. Awaria danego węzła dotyczy jedynie transakcji korzystających z blokad obiektów, których podstawowe kopie są przechowywane w tym węźle, i nie ma wpływu na pozostałe. W tej metodzie również można stosować węzły zapasowe, co zwiększa dostępność i stabilność.

**Wybór nowego węzła koordynującego w przypadku awarii.** W sytuacji, gdy węzeł koordynujący ulega awarii, wszystkie przedstawione techniki wymagają wybrania nowego węzła koordynującego. W przypadku techniki podstawowego węzła *bez* węzła zapasowego wszystkie wykonywane aktualnie transakcje muszą zostać przerwane i uruchomione od początku, co wydłuża proces przywracania systemu do pracy. Częścią tego procesu jest również wybór nowego węzła podstawowego, uruchomienie na nim procesu kontrolującego blokady i stworzenie rejestru blokad. W przypadku metod używających węzłów zapasowych wykonywanie transakcji jest tylko chwilowo zawieszone na czas wyznaczenia nowego węzła zapasowego i przesłania pełnego rejestru aktualnych blokad.

Jeśli dotychczasowy węzeł zapasowy  $X$  ma się stać węzłem podstawowym, to  $X$  może wybrać nowy węzeł zapasowy spośród wszystkich pracujących węzłów. Jeśli jednak nie istniał węzeł zapasowy lub uległ on awarii razem z podstawowym, to do wyznaczenia nowego węzła koordynującego można użyć procesu zwanego **wyborami (elekcją)**. Każdy węzeł  $Y$ , który bezskutecznie próbuje połączyć się z węzłem koordynującym, może przyjąć, że uległ on awarii, a następnie uruchomić proces wyborów, wysyłając do wszystkich działających węzłów propozycję wyznaczenia siebie ( $Y$ ) jako nowego węzła koordynującego. Po uzyskaniu od pozostałych węzłów większości głosów  $Y$  może oświadczyć, że przejmuje zadania węzła koordynującego. Sam proces wyborów może być dość złożony, lecz jego idea pozostaje taka sama. Algorytmy tego typu rozwiązują również konflikty w przypadku, kiedy kilka węzłów równocześnie stara się zostać węzłem koordynującym. Wybrane publikacje wymienione na końcu rozdziału przedstawiają tę problematykę dokładniej.

### 23.3.2. Rozproszone sterowanie współbieżne oparte na głosowaniu

Przedstawione w poprzednim punkcie metody sterowania współbieżnego stosują do zarządzania blokadami koncepcję wyróżnionej kopii. W **metodzie z głosowaniem** nie ma wyróżnionych kopii. Żądanie blokady jest wysyłane do wszystkich węzłów przechowujących replikę określonych danych. Każdy z tych węzłów zarządza swoimi własnymi blokadami i może zatwierdzać lub odmawiać ich przyznania. Jeśli transakcja żądająca blokady uzyska zgodę *większości* węzłów, to informuje *wszystkie węzły* o jej przyznaniu. Jeśli transakcja nie otrzyma zgody większości węzłów w *określonym czasie*, to cofa żądanie blokady i informuje o tym wszystkie węzły.

Metoda głosowania jest uważana za rzeczywiście rozproszoną metodę sterowania współbieżnego, ponieważ wszystkie węzły są jednakowo odpowiedzialne za podjętą decyzję. Symulacje pokazują jednak, że głosowanie powoduje zwiększenie ruchu pomiędzy węzłami w stosunku do metod z wyróżnioną kopią. Ponadto algorytm uwzględniający możliwość awarii podczas głosowania staje się bardzo skomplikowany.

### 23.3.3. Rozproszone odtwarzanie danych

Proces przywracania systemu w przypadku rozproszonych baz danych jest dość skomplikowany. Dlatego przedstawione są tu tylko niektóre aspekty z nim związane. W niektórych przypadkach samo stwierdzenie, że węzeł uległ awarii, jest trudne i wymaga wymiany wielu

komunikatów z pozostałymi węzłami. Przypuśćmy na przykład, że węzeł *X* wysyła komunikat do węzła *Y* i nie otrzymuje oczekiwanej odpowiedzi. Istnieje kilka możliwych wyjaśnień takiej sytuacji:

- komunikat nie został dostarczony do *Y* z powodu błędu komunikacji;
- węzeł *Y* jest wyłączony i nie mógł odpowiedzieć;
- węzeł *Y* pracuje poprawnie i wysłał odpowiedź, która nie została dostarczona.

Bez dodatkowych informacji i wymiany dalszych komunikatów trudno jest określić, co w rzeczywistości się stało.

Innym problemem w rozproszonym przywracaniu jest rozproszone zatwierdzanie. Kiedy transakcja aktualizuje dane w kilku węzłach, to nie może zakończyć pracy, dopóki nie ma pewności, że wynik jej działania zostanie zapisany w *każdym* węźle. Oznacza to, że najpierw każdy węzeł musi zapisać efekt transakcji na dysku w lokalnym dzienniku zdarzeń. Do zapewnienia poprawności wykonania rozproszonego zatwierdzenia transakcji często używany jest protokół zatwierdzania dwufazowego omówiony w podrozdziale 21.6.

## 23.4. Przegląd zarządzania transakcjami w rozproszonych bazach danych

Globalne i lokalne moduły oprogramowania do zarządzania transakcjami wraz z menedżerem sterowania współbieżnego i odtwarzania danych w rozproszonym SZBD wspólnie gwarantują właściwości ACID transakcji (patrz rozdział 20.).

Na potrzeby obsługi transakcji rozproszonych wprowadzono dodatkowy komponent — **globalny menedżer transakcji**. Węzeł źródłowy transakcji może tymczasowo przyjąć rolę globalnego menedżera transakcji i koordynować wykonywanie transakcji razem z menedżerami z innych węzłów. Menedżer transakcji udostępnia swoje funkcje za pomocą interfejsu podobnego do tego opisanego w punkcie 20.2.1. Obejmuje on operacje: `BEGIN_TRANSACTION`, `READ` lub `WRITE`, `END_TRANSACTION`, `COMMIT_TRANSACTION` i `ROLLBACK` (lub `ABORT`). Menedżer przechowuje powiązane z każdą transakcją informacje administracyjne: unikatowy identyfikator, węzeł źródłowy, nazwę itd. W operacjach `READ` (odczytu) menedżer zwraca lokalną kopię danych, jeśli jest ona prawidłowa i dostępna. W operacjach `WRITE` (zapisu) menedżer zapewnia, że aktualizacje są wprowadzane we wszystkich węzłach obejmujących kopie (repliki) określonego elementu danych. W operacjach `ABORT` (anulowania) menedżer gwarantuje, że w żadnym węźle rozproszonej bazy danych nie pozostały żadne efekty wykonywania transakcji. W operacjach `COMMIT` (zatwierdzania) menedżer powoduje, że efekty zapisu są trwale rejestrowane we wszystkich bazach obejmujących kopie określonego elementu danych. Atomowe kończenie transakcji rozproszonych (`COMMIT` i `ABORT`) jest zwykle implementowane za pomocą protokołu zatwierdzania dwufazowego (patrz podrozdział 22.6).

Menedżer transakcji przekazuje do modułu kontrolera sterowania współbieżnego operacje na bazie danych i powiązane informacje. Kontroler odpowiada za zajmowanie i zwalnianie odpowiednich blokad. Jeśli transakcja wymaga dostępu do zablokowanych zasobów,



jej wykonywanie jest wstrzymywane do momentu uzyskania blokady. Po uzyskaniu blokady operacja jest przekazywana do procesora wykonawczego, który odpowiada za wykonanie operacji na bazie danych. Po zakończeniu operacji blokady są zwalniane, a menedżer transakcji otrzymuje wynik operacji.

### 23.4.1. Protokół zatwierdzania dwufazowego

W podrozdziale 22.6 opisano *protokół zatwierdzania dwufazowego*, wymagający **globalnego menedżera odtwarzania** lub **koordynatora** do przechowywania informacji potrzebnych w trakcie odtwarzania. Potrzebne są też lokalne menedżery odtwarzania i aktualizowane przez nie informacje (dzienniki, tabele). Protokół zatwierdzania dwufazowego ma pewne wady, dlatego opracowano opisany dalej protokół zatwierdzania trójfazowego.

### 23.4.2. Protokół zatwierdzania trójfazowego

Największą wadą protokołu zatwierdzania dwufazowego jest to, że może skutkować zablokowaniem pracy. Awaria koordynatora blokuje wszystkie powiązane węzły i wymaga, aby oczekiwały one na wznowienie pracy przez koordynatora. Może to prowadzić do spadku wydajności — zwłaszcza w sytuacji, gdy powiązane węzły utrzymują blokady współużytkowanych zasobów. Mogą też wystąpić problemy innego rodzaju, przez co nie da się jednoznacznie ocenić skutków transakcji.

Rozwiązaniem tych problemów jest protokół zatwierdzania trójfazowego, dzielący drugą fazę zatwierdzania na dwie podfazy: **przygotowanie do zatwierdzenia** i **zatwierdzenie**. Faza przygotowania do zatwierdzenia służy do zakomunikowania wyniku fazy głosowania wszystkim węzłom. Jeśli wszystkie węzły głosują pozytywnie, koordynator nakazuje im przejście w stan przygotowania do zatwierdzenia. Faza zatwierdzania przebiega identycznie jak w procesie dwufazowym. Jeżeli w tej podfazie nastąpi awaria koordynatora, inny węzeł może nadzorować transakcję do czasu jej zakończenia. Ten inny węzeł może „zapytać” uszkodzony węzeł o to, czy ten ostatni otrzymał komunikat przygotowania do zatwierdzenia. Jeśli nie otrzymał, można bezpiecznie anulować transakcję. W tym modelu stan protokołu można odtworzyć niezależnie od tego, który węzeł uległ awarii. Ponadto dzięki skróceniu czasu potrzebnego na zatwierdzenie i anulowanie transakcji maksymalnie do limitu czasu oczekiwania protokół gwarantuje, że transakcja próbująca zatwierdzić zmiany za pomocą procesu trójfazowego zwolni blokady po upływie tego limitu.

Podstawowa idea polega na tym, by ograniczyć czas oczekiwania w węzłach przygotowanych do zatwierdzenia i czekających na globalne polecenie zatwierdzenia lub anulowania transakcji od koordynatora. Gdy węzeł otrzyma komunikat z żądaniem przygotowania do zatwierdzenia, wie, że pozostałe węzły głosowały za zatwierdzeniem. Jeśli taki komunikat nie został otrzymany, węzeł anuluje operację i zwalnia wszystkie blokady.



### 23.4.3. Obsługa zarządzania transakcjami w systemie operacyjnym

Oto główne korzyści zarządzania transakcjami przez system operacyjny:

- SZBD zwykle korzysta z semaforów<sup>2</sup> do gwarantowania wzajemnie wykluczającego się dostępu do zasobów współużytkowanych. Ponieważ semafony są implementowane w przestrzeni użytkownika na poziomie oprogramowania SZBD, system operacyjny nie ma informacji o nich. Dlatego jeśli system operacyjny zdezaktywuje proces SZBD utrzymujący blokadę, inne procesy SZBD potrzebujące zablokowanego zasobu zostaną wstrzymane. Taka sytuacja może prowadzić do znacznego spadku wydajności. Dostęp systemu operacyjnego do informacji o semaforach pomaga wyeliminować podobne przypadki.
- Aby obniżyć koszty przetwarzania, można wykorzystać wyspecjalizowaną obsługę blokad z poziomu sprzętowego. Może to być bardzo istotne, ponieważ blokowanie to jedna z najczęściej wykonywanych operacji w SZBD.
- Udostępnienie w jądrze zestawu standardowych operacji do obsługi transakcji pozwala programistom aplikacji skoncentrować się na dodawaniu nowych funkcji do produktów zamiast na ponownym implementowaniu typowych funkcji w każdej aplikacji. Przykładowo, jeśli w jednym komputerze współdzielą różne SZBD korzystające z protokołu zatwierdzania dwuetapowego, korzystniejsze jest zaimplementowanie tego protokołu w jądrze, dzięki czemu programiści rozproszonego SZBD mogą się skupić na dodawaniu mechanizmów do rozwijanych produktów.

## 23.5. Przetwarzanie zapytań i optymalizacja w rozproszonych bazach danych

Omówimy teraz, jak rozproszony system baz danych przetwarza i optymalizuje zapytania. Najpierw zostaną omówione etapy przetwarzania zapytań, dalej koszty związane z komunikacją przy wykonywaniu rozproszonych zapytań, a następnie specjalna operacja nazywana złączeniem częściowym (*semijoin*) używana przy optymalizacji niektórych rodzajów zapytań w systemach rozproszonych. Szczegółowe omawianie algorytmów optymalizacji wykracza poza zakres tej książki. Staramy się tu przedstawić zasady optymalizacji za pomocą odpowiednich przykładów<sup>3</sup>.

---

<sup>2</sup> Semafony to struktury danych używane do synchronizowanego i wyłącznego dostępu do zasobów współużytkowanych. Pozwalają uniknąć sytuacji wyścigu w systemach z przetwarzaniem równoległym.

<sup>3</sup> Szczegółowe omówienie algorytmów optymalizacji znajdziesz w pracy Ozszy i Valdureza (1999).

### 23.5.1. Przetwarzanie zapytań rozproszonych

Zapytanie w bazach rozproszonych jest przetwarzane w następujących etapach:

- (1) **Odwzorowywanie zapytania.** Wejściowe zapytanie dotyczące rozproszonych danych jest zapisywane formalnie w języku zapytań. Następnie jest przekształcane na zapytanie algebraiczne z użyciem globalnych relacji. Przekształcanie odbywa się za pomocą odwołań do globalnego schematu koncepcyjnego bez uwzględniania rzeczywistego rozproszenia i replikacji danych. Dlatego przekształcenia odbywają się prawie identycznie jak w scentralizowanych SZBD. Najpierw zapytanie jest normalizowane, analizowane pod kątem błędów semantycznych, upraszczane, a ostatecznie przekształcane w zapytanie algebraiczne.
- (2) **Lokalizacja.** W rozproszonych bazach wyniki podziału relacji są zapisywane w odrębnych węzłach. Niektóre fragmenty mogą być replikowane. Na tym etapie zapytanie rozproszone dotyczące globalnego schematu jest odwzorowywane na odrębne zapytania związane z poszczególnymi fragmentami. Odbywa się to na podstawie informacji o rozproszeniu i replikacji danych.
- (3) **Optymalizacja zapytania globalnego.** Optymalizacja obejmuje wybór z listy kandydatów strategii najbliższej optymalnemu rozwiązaniu. Listę zapytań kandydujących można uzyskać na podstawie permutacji kolejności operacji w wygenerowanym na poprzednim etapie zapytaniu dotyczącym fragmentu danych. Preferowaną jednostką pomiaru kosztów jest czas. Łączny koszt jest wyznaczany na podstawie kombinacji kosztów obliczeniowych, operacji wejścia-wyjścia i komunikacyjnych (poszczególnym składowym przypisywane są wagi). Ponieważ rozproszone bazy danych są powiązane siecią, najistotniejsze są często koszty komunikacji. Jest to prawdą zwłaszcza w sytuacji, gdy węzły są połączone siecią rozległą.
- (4) **Optymalizacja zapytania lokalnego.** Ten etap jest wykonywany we wszystkich węzłach rozproszonej bazy danych. Stosowane są tu techniki podobne jak w systemach scentralizowanych.

Pierwsze trzy etapy są wykonywane w centralnym węźle sterującym. Ostatni etap jest przetwarzany lokalnie.

### 23.5.2. Koszty przesyłu danych w przetwarzaniu zapytań rozproszonych

W rozdziale 19. omówione zostało przetwarzanie i optymalizacja zapytań w scentralizowanych systemach baz danych. W przypadku systemu rozproszonego przetwarzanie zapytań wiąże się z kilkoma dodatkowymi problemami. Pierwszy z nich to koszt przesyłania danych przez sieć. Takie dane zawierają pośrednie pliki przesyłane do innych węzłów do dalszego przetwarzania, a także ostateczne wyniki przesyłane do węzła, w którego są one potrzebne. Co prawda koszty mogą nie być wysokie w przypadku, kiedy węzły są połączone szybką siecią lokalną, ale stają się dość istotne przy innych typach sieci. Algorytmy optymalizacji zapytań w rozproszonych systemach baz danych mają zatem na celu zmniejszanie *ilości przesyłanych danych*. Jest to jedno z kryteriów optymalizacji na etapie wyboru strategii wykonywania zapytania rozproszonego.

Zilustrujemy to dwoma przykładowymi zapytaniem. Przyjmijmy, że relacje PRACOWNIK i DZIAŁ z rysunku 3.5 są rozproszone tak, jak to zostało pokazane na rysunku 23.4. Żadna z relacji nie jest więc podzielona. Zgodnie z rysunkiem 23.4, rozmiar relacji PRACOWNIK wynosi  $100 \cdot 10\,000 = 10^6$  bajtów, a rozmiar relacji DZIAŁ to  $35 \cdot 100 = 3500$  bajtów. Rozważmy następujące zapytanie Z: *Dla każdego pracownika odczytać jego imię i nazwisko oraz nazwę działu, w którym pracuje*. Za pomocą algebry relacyjnej można je wyrazić następująco:

Z:  $\pi_{IMIE, NAZWISKO, NAZWADZ} (PRACOWNIK \bowtie_{NRDZ = NUMERDZ} DZIAŁ)$

Węzeł 1:

PRACOWNIK

IMIE	INICJALDRIM	NAZWISKO	PESEL	DATAUR	ADRES	PLEC	PENSJA	PESELPRZEŁOŻ	NRDZ
------	-------------	----------	-------	--------	-------	------	--------	--------------	------

10 000 rekordów

każdy rekord ma 100 bajtów

PESEL ma 9 bajtów

NRDZ ma 4 bajty

IMIE ma 15 bajtów

NAZWISKO ma 15 bajtów

Węzeł 2:

DZIAŁ

NAZWADZ	NUMERDZ	PESELKIEROWNIKA	DATAPRZEJKIEROWNICTWA
---------	---------	-----------------	-----------------------

100 rekordów

każdy rekord ma 35 bajtów

NUMERDZ ma 4 bajty

PESELKIEROWNIKA ma 9 bajtów

NAZWADZ ma 10 bajtów

RYСУNEK 23.4. Przykład ilustrujący ilość przesyłanych danych

Wynik tego zapytania będzie zawierała 10 000 rekordów przy założeniu, że każdy pracownik jest przypisany do jakiegoś działu. Założmy, że każdy rekord wyniku zajmuje 40 bajtów. Zapytanie zostało uruchomione w zdalnym węźle 3., który nazywamy **węzłem wynikowym**, ponieważ tam właśnie trzeba dostarczyć wynik zapytania. Ani PRACOWNIK, ani DZIAŁ nie są przechowywane w tym węźle. Istnieją trzy różne strategie wykonania zapytania:

- (1) Przesłać zarówno relację PRACOWNIK, jak i DZIAŁ do węzła 3. i tam wykonać złączenie. W tym przypadku należy przesłać w sumie  $1\,000\,000 + 3500 = 1\,003\,500$  bajtów.
- (2) Przesłać relację PRACOWNIK do węzła 2., tam wykonać złączenie i przesłać wynik do węzła 3. Rozmiar wyniku zapytania to  $40 \cdot 10\,000 = 400\,000$  bajtów, a więc w sumie trzeba przesłać 1 400 000 bajtów.
- (3) Przesłać relację DZIAŁ do węzła 1., tam wykonać złączenie, po czym wynik przesłać do węzła 3. W tym przypadku rozmiar przesyłanych danych wynosi  $400\,000 + 3500 = 403\,500$  bajtów.

Jeśli jako kryterium optymalizacyjne przyjmujemy minimalizację przesyłanych danych, to najlepszym rozwiązaniem jest strategia 3. Rozważmy teraz inne zapytanie Z': *Dla każdego działu wybrać nazwę działu oraz imię i nazwisko jego kierownika*. Można je wyrazić za pomocą algebry relacyjnej w następujący sposób:

Z':  $\pi_{IMIE, NAZWISKO, NAZWADZ} (DZIAŁ \bowtie_{PESELKIEROWNIKA = PESEL} PRACOWNIK)$

Ponownie przyjmijmy, że węzłem wynikowym jest węzeł 3. W przypadku zapytania  $Z'$  można zastosować te same trzy strategie co w przypadku zapytania  $Z$ , przy czym wynik  $Z'$  będzie zawierał tylko 100 rekordów (przy założeniu, że każdy dział ma kierownika):

- (1) Przesłać zarówno relację PRACOWNIK, jak i DZIAŁ do węzła wynikowego i tam wykonać złączenie. W tym przypadku należy przesłać w sumie  $1\,000\,000 + 3500 = 1\,003\,500$  bajtów.
- (2) Przesłać relację PRACOWNIK do węzła 2., tam wykonać złączenie i przesłać wynik do węzła 3. Rozmiar wyniku zapytania wynosi  $40 \cdot 100 = 4000$  bajtów, więc w sumie trzeba przesłać  $1\,000\,000 + 4000 = 1\,004\,000$  bajtów.
- (3) Przesłać relację DZIAŁ do węzła 1., tam wykonać złączenie i wysłać wynik do węzła 3. W tym przypadku rozmiar przesyłanych danych wynosi  $4000 + 3500 = 7500$  bajtów.

Ponownie należy wybrać strategię 3., w tym przypadku dającą bardzo wyraźną różnicę w ilości przesyłanych danych w porównaniu ze strategiami 1. i 2. Przedstawione wyżej trzy strategie są najbardziej oczywistymi rozwiązaniami w przypadku, gdy węzeł wynikowy (węzeł 3.) jest inny od wszystkich przechowujących wymagane dane (węzeł 1. i 2.). Jeżeli jednak węzłem wynikowym byłby 2., to rozważane byłyby dwie proste strategie:

- (1) Przesłać relację PRACOWNIK do węzła 2., wykonać zapytanie i przedstawić wynik użytkownikowi w lokalnym węźle. Dla obu zapytań ( $Z$  i  $Z'$ ) trzeba wtedy przesłać  $1\,000\,000$  bajtów.
- (2) Przesłać relację DZIAŁ do węzła 1., tam wykonać zapytanie, a następnie przesłać wynik z powrotem do węzła 2. W takim przypadku trzeba przesłać  $400\,000 + 3500 = 403\,500$  bajtów dla zapytania  $Z$  i  $4000 + 3500 = 7500$  bajtów dla zapytania  $Z'$ .

Bardziej złożone strategie, które czasami są bardziej wydajne od zaprezentowanych powyżej prostych strategii, używają operacji **złączenia częściowego**. Wprowadzimy samą operację i jej zastosowanie w przetwarzaniu rozproszonych zapytań w następnym punkcie.

### 23.5.3. Rozproszone przetwarzanie zapytań z użyciem złączeń częściowych

Koncepcja stojąca za rozproszonym przetwarzaniem zapytań z użyciem *operacji złączenia częściowego* polega na zredukowaniu liczby krotek w relacji jeszcze przed ich przesłaniem do innego węzła. Mechanizm opiera się na przesłaniu *kolumny łączącej* jednej relacji  $R$  do węzła, gdzie jest przechowywana inna relacja  $S$ . Przesłana kolumna będzie potem łączona z  $S$ . Następnie atrybuty łączące razem z atrybutami wymaganymi w wyniku zapytania są przesyłane z powrotem do oryginalnego węzła i łączone z  $R$ . Zatem w jedną stronę przesyłana jest tylko łącząca kolumna  $R$ , a w drugą podzbiór  $S$  bez zbędnych krotek i atrybutów. Jeśli złączenie dotyczy tylko niedużej części spośród krotek  $S$ , to jest to wydajne rozwiązanie zmniejszające ilość przesyłanych danych.

Rozważmy następującą strategię wykonania  $Z$  i  $Z'$  ilustrującą prezentowaną koncepcję:

- (1) Dokonać projekcji atrybutów łączących relacji DZIAŁ w węźle 2. i przesłać je do węzła 1. Dla  $Z$  przesyłamy  $F = \pi_{\text{NUMERDZ}}(\text{DZIAŁ})$ , o rozmiarze równym  $4 \cdot 100 = 400$  bajtów; dla  $Z'$  przesyłamy  $F' = \pi_{\text{PESELKIEROWNIKA}}(\text{DZIAŁ})$ , o rozmiarze równym  $9 \cdot 100 = 900$  bajtów.

- (2) Połączyć przesłany plik z relacją `PRACOWNIK` w węźle 1. i przesłać wymagane atrybuty do węzła 2. Dla  $Z$  przesyłamy  $R = \pi_{NRDZ, IMIE, NAZWISKO}(F \bowtie_{NUMERDZ = NRDZ} PRACOWNIK)$ , o rozmiarze  $34 \cdot 10\,000 = 340\,000$  bajtów; dla  $Z'$  przesyłamy  $R' = \pi_{PESELKIEROWNIKA, IMIE, NAZWISKO}(F' \bowtie_{PESELKIEROWNIKA = PESEL} PRACOWNIK)$  o rozmiarze  $39 \cdot 100 = 3900$  bajtów.
- (3) Wykonać zapytanie łączące przesłany plik  $R$  lub  $R'$  z relacją `DZIAŁ` i dostarczyć wynik użytkownikowi w węźle 2.

Używając tej strategii, przesyłamy 340 400 bajtów dla zapytania  $Z$  i 4800 dla zapytania  $Z'$ . Ograniczyliśmy atrybuty i krotki relacji `PRACOWNIK` przesyłane do węzła 2. w kroku 2. do tych, które *rzeczywiście będą łączone* z krotkami relacji `DZIAŁ` w kroku 3. W zapytaniu  $Z$  okazały się to być wszystkie krotki, co spowodowało nieduży zysk. Z drugiej strony w zapytaniu  $Z'$  potrzebne było tylko 100 krotek spośród 10 000.

Operacja złączenia częściowego jest wynikiem sformalizowanej przedstawionej strategii. **Operacja złączenia częściowego**  $R \bowtie A = B S$ , gdzie  $A$  i  $B$  są atrybutami z dziedziny odpowiednio  $R$  i  $S$ , daje wynik taki sam jak wyrażenie z algebry relacyjnej  $\pi_R(R \bowtie_{A=B} S)$ . W środowisku rozproszonym, gdzie  $R$  i  $S$  są zapisane w różnych miejscach, złączenie częściowe jest zazwyczaj realizowane poprzez przesłanie  $F = \pi_B(S)$  do węzła, w którym zapisana jest relacja  $R$ , a następnie złączenie  $F$  z  $R$ , co prowadzi do zrealizowania przedstawionej wcześniej strategii.

Zauważmy, że operacja złączenia częściowego nie jest przemienne, czyli

$$R \bowtie S \neq S \bowtie R$$

## 23.5.4. Dekompozycja zapytań i aktualizacji

W systemie rozproszonym *bez przezroczystości rozproszenia* użytkownik adresuje zapytanie bezpośrednio do określonego fragmentu danych. Na przykład, rozważmy inne zapytanie  $Z$ : *Odczytać imiona i nazwiska pracowników oraz liczbę godzin poświęconych przez nich projektom nadzorowanym przez dział 5.* wykonane w rozproszonej bazie danych, której relacje w węzłach 2. i 3. są pokazane na rysunku 23.2, a dla węzła 1. na rysunku 5.6. Użytkownik uruchamiający takie zapytanie musi określić, czy odnosi się ono do fragmentów `PROJ5` i `PRACUJE_NAD5` z węzła 2. (rysunek 23.2), czy do relacji `PROJEKT` i `PRACUJE_NAD` z węzła 1. (rysunek 5.6). Użytkownik musi również zadbać o zachowanie spójności replikowanych danych podczas aktualizacji bazy danych, która *nie obsługuje przezroczystości replikacji*.

Z drugiej strony, w rozproszonym systemie baz danych obsługującym *pełną przezroczystość rozproszenia, fragmentacji i replikacji* użytkownik może wykonywać zapytania i aktualizacje w schemacie z rysunku 5.5 tak, jakby była to jedna scentralizowana baza danych. W przypadku aktualizacji danych system rozproszony musi dbać o *spójność replikowanych elementów* — stosowane są do tego celu algorytmy rozproszonego sterowania współbieżnego omawiane w podrozdziale 23.3. W przypadku zapytań moduł **dekompozycji zapytań** musi **podzielić** oryginalne zapytanie na kilka **podzapytań**, które mogą być wykonane przez poszczególne węzły. Należy również stworzyć strategię łączenia wyników podzapytań w celu uzyskania ostatecznego rezultatu całego zapytania. Jeżeli system stwierdzi, że dane, których dotyczy jedno z podzapytań, są replikowane, to musi on wybrać (**zmaterializować**) określoną replikę, która będzie używana podczas wykonywania zapytania.

W celu określenia, która replika zawiera dane potrzebne do wykonania zapytania, rozproszony system bazy danych odnosi się do zapisanych w jego katalogu informacji o fragmentacji, replikacji i rozproszeniu danych. We fragmentacji pionowej w katalogu zapisywane są listy atrybutów poszczególnych fragmentów. W przypadku fragmentacji poziomej stosuje się narzędzie nazywane **strażnikiem** (ang. *guard*). Jest to warunek określający, jakie krotki są elementami danego fragmentu, a nazywany jest strażnikiem, ponieważ dopuszcza do zapisania w danym fragmencie *tylko te krotki, które spełniają jego warunki*. W przypadku stosowania fragmentacji hybrydowej, w katalogu przechowywana jest zarówno lista atrybutów, jak i strażnik.

We wcześniejszym przykładzie strażnikami dla fragmentów w węźle 1. (rysunek 5.6) są wartości TRUE (czyli warunek jest spełniony przez każdą krotkę), a listami atrybutów \* (wszystkie atrybuty). Listy atrybutów i strażnicy dla fragmentów z rysunku 23.2 są przedstawione na rysunku 23.5. Kiedy system rozproszony dokonuje dekompozycji zapytania aktualizacji danych, może stwierdzić, które fragmenty mają być zaktualizowane, używając przypisanych im warunków-strażników. Na przykład, komenda dodająca nową krotkę <'Aleksander', 'F', 'Nowakowski', '64042215533', '22.04.1964', 'ul. Bażantów 35, Katowice', M, 3500, '70050502233', 4> do relacji PRACOWNIK będzie rozłożona na dwie komendy: pierwsza doda powyższą krotkę do bazy danych w węźle 1., a druga doda krotkę po odpowiedniej projekcji <'Aleksander', 'F', 'Nowakowski', '64042215533', 3500, '70050502233', 4> do fragmentu PRACD4 w węźle 3.

(a) PRACD5

lista atrybutów: IMIĘ, INICJAŁDRIMIENIA, NAZWISKO, PESEL, PENSJA, PESELPRZEŁÓŻ, NRDZ

warunek: NRDZ=5

DZIAŁ5

lista atrybutów: \* (wszystkie atrybuty NUMERDZ, NAZWADZ, PESELKIEROWNIKA, DATAPRZEJKIEROWNICTWA)

warunek: NUMERDZ=5

LOK\_DZ5

lista atrybutów: \* (wszystkie atrybuty NUMERDZ, LOKALIZACJADZ)

warunek: NUMERDZ=5

PROJ5

lista atrybutów: \* (wszystkie atrybuty NAZWAPROJ, NUMERPROJ, LOKALIZACJAPROJ, NR\_DZ)

warunek: NR\_DZ=5

PRACUJE\_NAD5

lista atrybutów: \* (wszystkie atrybuty PESELPRAC, NR\_PROJ, GODZINY)

warunek: PESELPRAC IN ( $\pi_{PESEL}$  (PRACD5)) OR NR\_PROJ IN ( $\pi_{NUMERPROJ}$  (PROJ5))

(b) PRACD4

lista atrybutów: IMIĘ, INICJAŁDRIMIENIA, NAZWISKO, PESEL, PENSJA, PESELPRZEŁÓŻ, NRDZ

warunek: NRDZ=4

DZIAŁ4

lista atrybutów: \* (wszystkie atrybuty NUMERDZ, NAZWADZ, PESELKIEROWNIKA, DATAPRZEJKIEROWNICTWA)

warunek: NUMERDZ=4

```

LOK_DZ4
  lista atrybutów: * (wszystkie atrybuty NUMERDZ, LOKALIZACJADZ)
warunek: NUMERDZ=4
PROJ4
  lista atrybutów: * (wszystkie atrybuty NAZWAPROJ, NUMERPROJ, LOKALIZACJAPROJ,
  NR_DZ)
warunek: NR_DZ=4
PRACUJE_NAD4
  lista atrybutów: * (wszystkie atrybuty PESELPRAC, NR_PROJ, GODZINY)
warunek: PESELPRAC IN ( $\pi$ PESEL (PRACD4)) OR NR_PROJ IN ( $\pi$ NUMERPROJ (PROJ4))

```

RYSUNEK 23.5. Listy atrybutów i strażnicy dla przykładowych fragmentów. (a) Fragmenty w węźle 2. (b) Fragmenty w węźle 3.

Przy dekompozycji zapytania rozproszony system bazy danych może określić, które fragmenty mogą zawierać potrzebne krotki, porównując warunek zapytania z warunkiem strażnika. Rozważmy zapytanie Z: *Dla każdego pracownika pracującego nad projektem nadzorowanym przez dział 5. odczytać imię i nazwisko oraz liczbę godzin poświęcanych tygodniowo projektowi.* Takie zapytanie w schemacie określonym na rysunku 5.5 można wyrazić w SQL następująco:

```

Z: SELECT IMIĘ, NAZWISKO, GODZINY
   FROM PRACOWNIK, PROJEKT, PRACUJE_NAD
   WHERE NR_DZ=5 AND NUMERPROJ=NR_PROJ AND PESELPRAC=PESEL;

```

Przypuśćmy, że zapytanie zostało uruchomione w węźle 2., więc tam należy przesłać wynik zapytania. Rozproszony system SZBD może na podstawie warunków dla PROJ5 i PRACUJE\_NAD5 określić, że wszystkie krotki spełniające warunek zapytania ( $NR\_DZ=5$  AND  $NUMERPROJ=NR\_PROJ$ ) są zapisane w węźle 2. Można zatem rozbić zapytanie na następujące podzapytania w algebrze relacyjnej:

```

T1 ←  $\pi$ PESELPRAC (PROJ5 ⋈NUMERPROJ=NR_PROJ PRACUJE_NAD5)
T2 ←  $\pi$ PESELPRAC, IMIĘ, NAZWISKO (T1 ⋈PESELPRAC=PESEL PRACOWNIK)
WYNIK ←  $\pi$ IMIĘ, NAZWISKO, GODZINY (T2 * PRACUJE_NAD5)

```

Taki podział może być zastosowany przy wykonywaniu zapytania z wykorzystaniem strategii złączenia częściowego. System rozproszony może na podstawie strażnika określić, że PROJ5 zawiera dokładnie te krotki, które spełniają ( $NR\_DZ=5$ ), a PRACUJE\_NAD5 zawiera wszystkie krotki, które mają być złączone z PROJ5. Zatem podzapytanie T1 może być wykonane w węźle 2., a projektowana kolumna PESELPRAC może być przesłana do węzła 1. Podzapytanie T2 może być następnie wykonane w węźle 1., a jego wynik przesłany z powrotem do węzła 2., gdzie ostateczny wynik jest wyliczany i prezentowany zainteresowanemu użytkownikowi. Alternatywną strategią byłoby przesłanie całego zapytania Z do węzła 1., który zawiera wszystkie krotki bazy danych, wykonanie go i przesłanie ostatecznego wyniku z powrotem do węzła 2. Moduł optymalizujący zapytania powinien porównać koszty obu strategii i wybrać tę z nich, która powoduje mniejsze koszty.

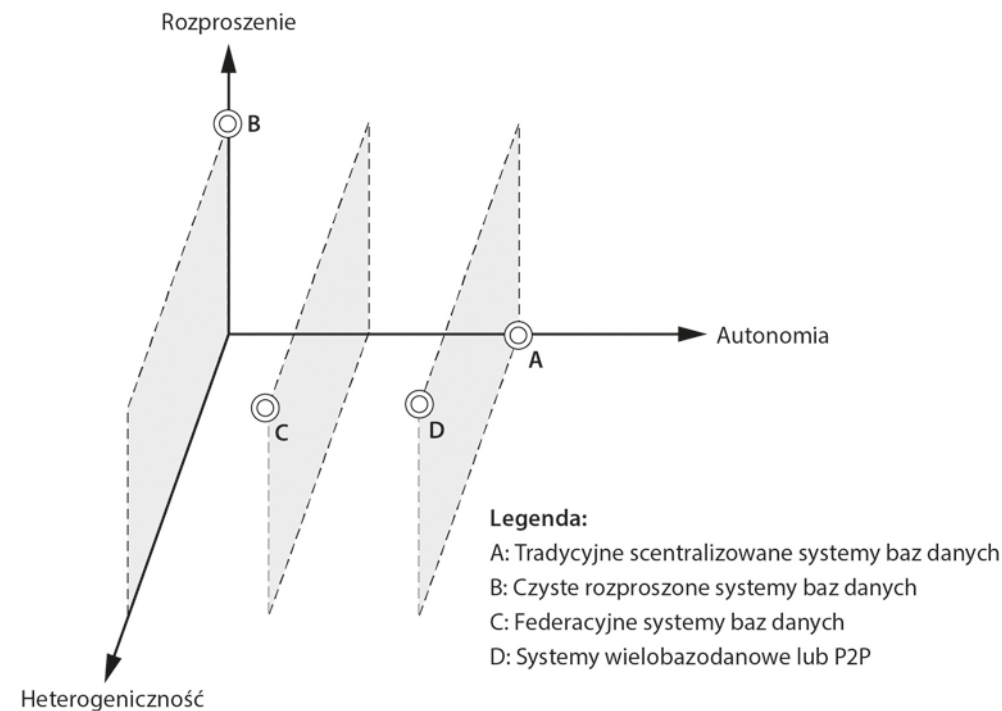


## 23.6. Rodzaje rozproszonych systemów baz danych

Termin *rozproszony system baz danych* może dotyczyć systemów różniących się pod wieloma względami. Główną cechą wspólną dla nich wszystkich jest rozproszenie danych i oprogramowania po kilku miejscach połączonych jakimś rodzajem sieci telekomunikacyjnej. Niniejszy podrozdział dotyczy różnych typów rozproszonych systemów baz danych i czynników, które je różnią.

Pierwszym czynnikiem jest **stopień homogeniczności** oprogramowania rozproszonego SZBD. Jeśli wszystkie serwery (lub pojedyncze lokalne systemy baz danych) i wszyscy użytkownicy (klienci) tych serwerów używają identycznego oprogramowania, to taki system rozproszony nazywamy **homogenicznym**. W przeciwnym przypadku mamy do czynienia z systemem **heterogenicznym**. Innym czynnikiem związanym ze stopniem homogeniczności jest **stopień lokalnej autonomii**. Jeśli lokalny węzeł nie ma możliwości pracy jako samodzielny system bazy danych, to mówimy, że **lokalna autonomia nie istnieje**. Z drugiej strony — jeśli możliwy jest *bezpośredni dostęp* do serwera i wykonywanie na nim lokalnych transakcji, to system ma pewien stopień lokalnej autonomii.

Na rysunku 23.6 pokazana jest klasyfikacja rozproszonych SZBD według trzech osi: poziomu rozproszenia, autonomii i heterogeniczności. Scentralizowane bazy danych charakteryzują się pełną autonomią, ale całkowitym brakiem rozproszenia i heterogeniczności (punkt A na rysunku). Widać tu, że poziom lokalnej autonomii umożliwia podział na systemy federacyjne i wielobazodanowe. Po jednej stronie mamy skrajny przypadek, gdzie z punktu widzenia użytkownika rozproszony system bazy danych *wygląda*, jakby był scentralizowany, ale ma zerową autonomię (punkt B). Istnieje jeden spójny schemat danych, a dostęp do całego systemu odbywa się poprzez węzeł należący do rozproszonego SZBD, co oznacza brak lokalnej autonomii. Na osi autonomii wyróżnione są dwa rodzaje rozproszonych SZBD: *federacyjny system baz danych* (punkt C) i *system wielobazodanowy* (punkt D). W takich systemach każdy serwer jest niezależnym i samodzielnym scentralizowanym systemem bazy danych, ma swoich lokalnych użytkowników, lokalne transakcje i własne konto administratora, co skutkuje bardzo wysokim stopniem *lokalnej autonomii*. Termin **federacyjny system baz danych (FSBD)** jest stosowany w przypadku, kiedy istnieje pewna globalna perspektywa czy schemat baz danych współdzielonych przez aplikacje (punkt C). Z drugiej strony, **system wielobazodanowy** ma pełną lokalną autonomię w tym sensie, że nie posiada wspólnego schematu, a aplikacje tworzą go interaktywnie w miarę aktualnych potrzeb (punkt D). Oba systemy są hybrydą rozproszonych i scentralizowanych systemów baz danych, a rozróżnienie pomiędzy nimi nie jest ściśle przestrzegane. Będziemy się dalej odnosić do nich w ogólnym znaczeniu jako do federacyjnego systemu baz danych. Punkt D na rysunku może też oznaczać system o pełnej lokalnej autonomii i pełnej heterogeniczności. Może to być system baz danych typu P2P. W heterogenicznym FSBD jeden serwer może być relacyjnym, inny sieciowym (np. IDMS firmy Computer Associates lub IMAGE/3000 firmy HP), a jeszcze inny obiektowym (takim jak ObjectStore firmy Object Design) lub hierarchicznym systemem baz danych (np. IMS firmy IBM). W takim przypadku istnieje potrzeba użycia wspólnego systemowego języka oraz modułów tłumaczących podzapytania z języka systemowego na języki każdego z serwerów.



RYSUNEK 23.6. Klasyfikacja rozproszonych baz danych

Poniżej pokrótce omówimy kwestie związane z projektowaniem federacyjnych systemów baz danych.

### 23.6.1. Zarządzanie federacyjnymi systemami baz danych

Rodzaj heterogeniczności w FSBD może dotyczyć różnych obszarów. Omówimy je po kolei, a następnie przedstawimy różne typy autonomii, które mogą pomóc w rozwiązywaniu problemów w heterogenicznych FSBD.

- Różnice w modelach danych.** Bazy danych w organizacji przybierają różne postacie, od starszych typów systemów (sieciowych lub hierarchicznych), przez modele relacyjny i obiektowy, po zwykłe pliki. Możliwości modelowania danych w poszczególnych typach systemów mogą być różne, a zatem obsługa ich wszystkich w postaci jednego globalnego schematu i przetwarzanie danych w jednym systemowym języku bywa trudne. Nawet jeśli dwa systemy są relacyjnymi bazami danych, ta sama informacja może być w nich reprezentowana jako nazwa atrybutu, nazwa relacji lub wartości w różnych bazach danych. Taka sytuacja wymaga inteligentnego mechanizmu przetwarzania zapytań, który potrafi powiązać różne informacje bazując na metadanych.
- Różnice w ograniczeniach.** Różne systemy mają różne ograniczenia definicyjne i implementacyjne. Pewne porównywalne funkcje muszą być wzięte pod uwagę podczas tworzenia globalnego schematu systemu. Na przykład relacje w modelu ER

odpowiadają ograniczeniom integralności odwołań w modelu relacyjnym.

Do zaimplementowania niektórych ograniczeń w modelu relacyjnym niezbędne mogą być wyzwalacze. Globalny schemat musi również brać pod uwagę potencjalne konflikty pomiędzy różnymi ograniczeniami.

- **Różnice w językach zapytań.** Nawet w przypadku takich samych modeli danych, języki i ich wersje różnią się między sobą. Na przykład SQL ma wiele wersji, takich jak SQL-89, SQL-92, SQL-99 i SQL:2008, a każdy system wprowadza swoje własne typy danych, operatory porównań, funkcje znakowe itp.

**Heterogeniczność semantyczna.** Heterogeniczność semantyczna dotyczy sytuacji, w której pojawiają się różnice w znaczeniu, interpretacji i zastosowaniu tych samych lub podobnych danych. Różnice semantyczne pomiędzy komponentami systemu rozproszonego są największym problemem przy projektowaniu globalnych schematów heterogenicznych baz danych. **Autonomia projektowa** składowych baz danych odnosi się do swobody wyboru parametrów projektowych, co z kolei przekłada się na złożoność federacyjnego systemu baz danych.

- **Rozbieżności przestrzeni atrybutów, z których pochodzą dane.** Na przykład dwa rachunki klientów — w systemie federacyjnym z bazami danych z Europy i Japonii — mogą zawierać zupełnie różne zestawy atrybutów związane z różnymi praktykami księgowymi. Problemy mogą również powodować wahania kursu walut. Dlatego dwie relacje o podobnych nazwach i przeznaczeniu (np. KLIENT lub RACHUNEK) mogą zawierać część wspólnych, a część zupełnie różnych danych.
- **Reprezentacje i nazewnictwo.** Reprezentacja i nazewnictwo poszczególnych elementów struktury danych mogą być z góry określone dla każdej lokalnej bazy danych.
- **Zrozumienie, znaczenie i subiektywność interpretacji danych.** Te elementy stanowią podstawę heterogeniczności semantycznej.
- **Ograniczenia transakcji i polityk.** Dotyczą one szeregowalności kryteriów, kompensacji transakcji i innych kwestii związanych z transakcjami.
- **Obliczanie podsumowań.** Systemy mogą w różnym stopniu obsługiwać agregację, podsumowania i inne funkcje i operacje związane z przetwarzaniem statystycznym danych.

Z wymienionymi problemami związanymi z heterogenicznością semantyczną zmagają się wszystkie duże organizacje międzynarodowe i rządowe działające w różnych dziedzinach. W dzisiejszym środowisku komercyjnym większość przedsiębiorstw ucieka się do heterogenicznych systemów federacyjnych z powodu poniesienia w ostatnich 20 – 30 latach wysokich nakładów na opracowanie wielu systemów baz danych używających różnych modeli danych i platform operacyjnych. Przedsiębiorstwa sięgają po różnego rodzaju oprogramowanie (zwykle nazywane **warstwą pośrednią**; ang. *middleware*), pakiety internetowe (**serwery aplikacji** takie jak WebLogic lub WebSphere), a nawet generyczne systemy (**systemy ERP** — ang. *enterprise resource planning*; przykłady to SAP i system ERP firmy J.D. Edwards). Te narzędzia służą do przekazywania zapytań i transakcji z aplikacji globalnych do poszczególnych baz danych (czasem z dodatkowym przetwarzaniem z użyciem reguł biznesowych) oraz danych z heterogenicznych serwerów bazodanowych do aplikacji globalnych. Szczegółowe omawianie tego rodzaju systemów oprogramowania wykracza poza zakres tego tekstu.

Podobnie jak w trakcie tworzenia architektury każdej rozproszonej bazy danych celem jest zapewnienie pełnej przezroczystości, tak w składowych lokalnych bazach danych dąży się do zachowania autonomii. **Autonomia komunikacji** składowych baz danych polega na ich zdolności do podejmowania decyzji co do komunikowania się z innymi węzłami systemu. **Autonomia wykonania** odnosi się do możliwości samodzielnego wykonywania lokalnych operacji bez zakłóceń ze strony innych mechanizmów i decydowania o kolejności ich wykonywania. **Autonomia powiązań** składowej bazy danych polega na decydowaniu, jaki obszar lokalnej funkcjonalności (obsługiwanych operacji) i zasobów (zarządzanych danych) będzie dostępny dla pozostałych baz składowych. Podstawowym wyzwaniem przy projektowaniu systemów federacyjnych jest zapewnienie współpracy poszczególnych systemów składowych przy jednoczesnym zachowaniu wymienionych wyżej rodzajów autonomii.

## 23.7. Architektury rozproszonych baz danych

W tym podrozdziale najpierw pokrótce omówimy różnice między architekturami równoległych i rozproszonych baz danych. Choć oba te rozwiązania są obecnie powszechnie stosowane, w dużych firmach występują różne, stale modyfikowane odmiany architektur rozproszonych. Architektura równoległa częściej jest używana w obliczeniach o wysokiej wydajności, gdzie architektury wieloprocessorowe są potrzebne do radzenia sobie z dużą ilością danych w aplikacjach do przetwarzania transakcji i obsługi hurtowni danych. Dalej przedstawimy generyczną architekturę rozproszonych baz danych, a następnie architekturę trójwarstwową typu klient-serwer i architekturę federacyjnych systemów baz danych.

### 23.7.1. Architektura równoległa a rozproszona

W powszechnym użyciu są dwa główne rodzaje architektur systemów wieloprocessorowych:

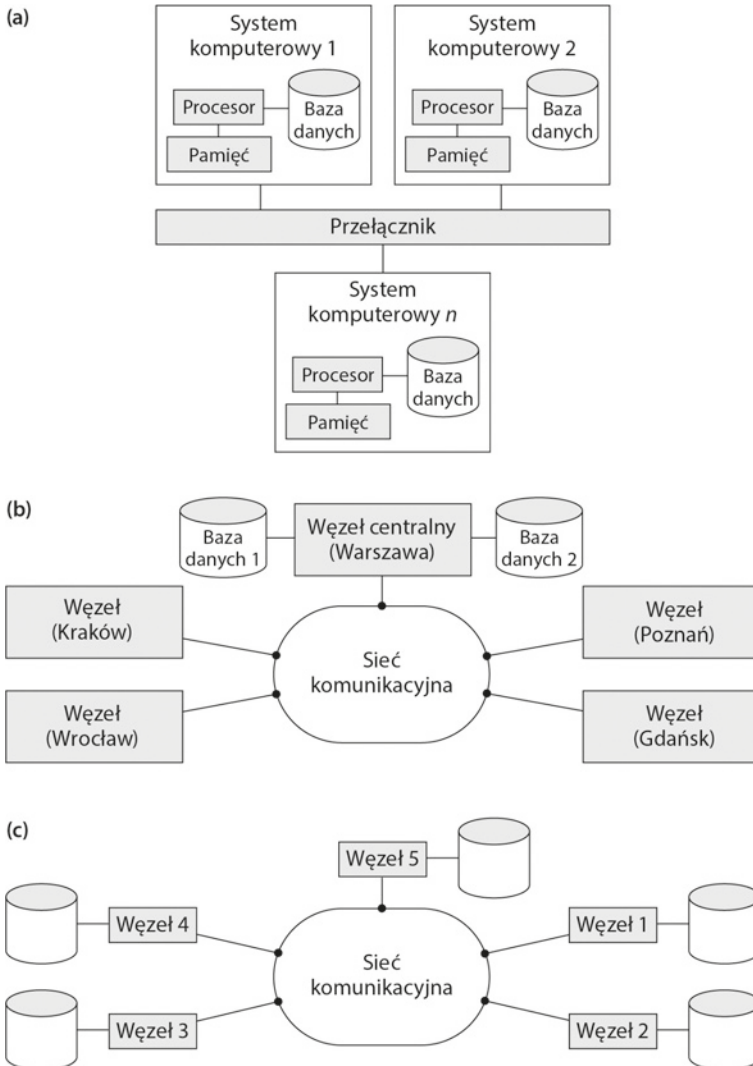
- **Systemy o dzielonej pamięci (ściśle powiązane).** Wiele procesorów dzieli drugorzędny nośnik danych (dysk) oraz pamięć główną.
- **Systemy o dzielonym dysku (luźno powiązane).** Wiele procesorów dzieli drugorzędny nośnik danych (dysk), ale każdy korzysta z własnego obszaru pamięci głównej.

Architektury te pozwalają procesorom na komunikowanie się bez narzutu powodowanego wymianą danych za pośrednictwem sieci<sup>4</sup>. Systemy baz danych opracowane w oparciu o powyższe architektury są określane raczej jako **równoległe systemy zarządzania bazami danych**, a nie jako systemy rozproszone, ponieważ używają technologii wieloprocessorowej. Inny typ systemów wieloprocessorowych jest nazywany **architekturą bez zasobów współdzielonych**. W tym przypadku każdy z procesorów dysponuje własną pamięcią główną i drugorzędną (dyskową), nie istnieją współdzielone fragmenty pamięci, a cała komunikacja odbywa się za pośrednictwem szybkiego połączenia sieciowego (magistrali lub przełącznika). Co prawda architektura bez zasobów współdzielonych przypo-

---

<sup>4</sup> Jeżeli współdzielona jest zarówno pamięć główna, jak i drugorzędna (pamięć główna i dyskowa), to systemy takie są nazywane również **systemami w pełni dzielonymi**.

mina środowisko rozproszone, ale istnieją pewne wyraźne różnice w pracy takich systemów. W architekturze bez zasobów współdzielonych zachowana jest symetria i homogeniczność poszczególnych węzłów. Inaczej w przypadku architektury rozproszonej — tutaj heterogeniczność sprzętu i systemów operacyjnych jest bardzo częsta. Architektura bez zasobów współdzielonych jest więc zaliczana do systemów równoległych. Rysunek 23.7(a) przedstawia bazę równoległą (bez zasobów współdzielonych), na rysunku 23.7(b) widoczna jest scentralizowana baza z dostępem rozproszonym, a rysunek 23.7(c) ilustruje zwykłą bazę rozproszoną. W tym miejscu nie omawiamy dokładnie architektur równoległych ani powiązanych zagadnień z obszaru zarządzania danymi.



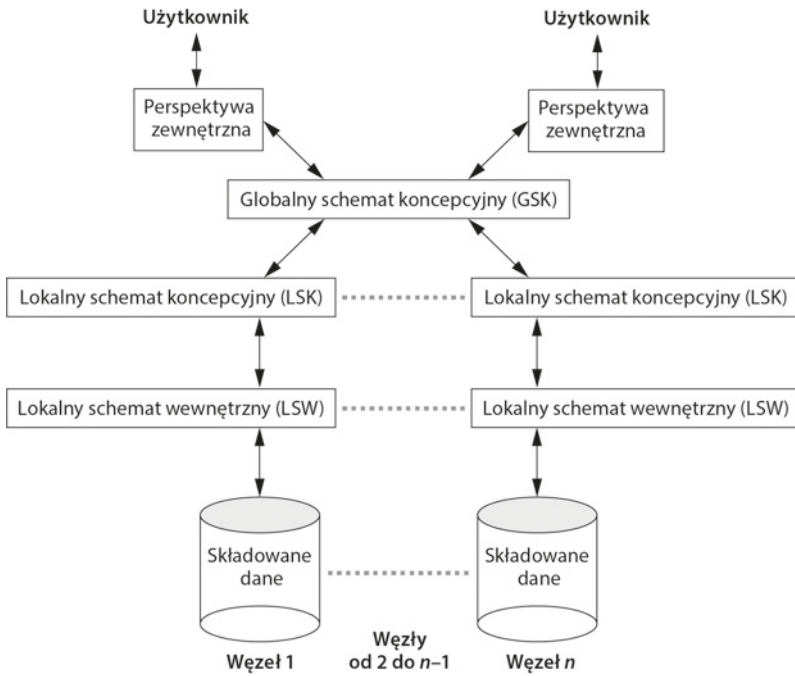
RYSUNEK 23.7. Różne architektury systemów baz danych. (a) Architektura bez zasobów współdzielonych. (b) Architektura sieciowa ze scentralizowaną bazą w jednym z węzłów. (c) Architektura w pełni rozproszonej bazy danych

### 23.7.2. Ogólna architektura czystych baz rozproszonych

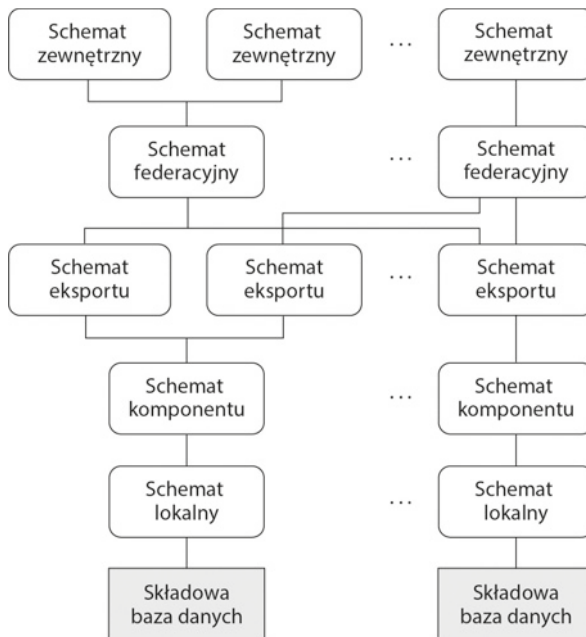
W tym punkcie omawiamy modele logiczny i komponentowy architektury rozproszonych baz danych. Na rysunku 23.8, gdzie przedstawiony jest generyczny schemat architektury rozproszonej bazy danych, widoczny jest system przedsiębiorstwa ze spójną, jednolitą logiczną strukturą danych we wszystkich węzłach. Ta struktura jest reprezentowana przez globalny schemat koncepcyjny (GSK), który zapewnia przezroczystość sieci (patrz punkt 23.1.2). Aby uwzględnić możliwą heterogeniczność rozproszonej bazy danych, dla każdego węzła pokazany jest lokalny schemat wewnętrzny oparty na szczegółach fizycznej organizacji danych w konkretnym węźle. Logiczna organizacja danych w poszczególnych węzłach jest oparta na lokalnym schemacie koncepcyjnym (LSK). GSK i LSK oraz powiązane z nimi odwzorowania zapewniają opisaną w punkcie 23.1.2 przezroczystość fragmentacji i replikacji. Na rysunku 23.8 pokazana jest architektura rozproszonej bazy danych. Jest ona rozwinięciem jej scentralizowanego odpowiednika z rysunku 2.3 z rozdziału 2. Dla uproszczenia pominięto tu elementy wspólne w obu architekturach. Kompilator zapytań globalnych używa globalnego schematu koncepcyjnego z globalnego katalogu systemowego, aby sprawdzić i wymusić zdefiniowane ograniczenia. Optymalizator zapytań globalnych używa obu schematów koncepcyjnych (globalnego i lokalnego), aby na podstawie zapytań globalnych wygenerować zoptymalizowane zapytania lokalne. Optymalizator ocenia wszystkie strategie kandydujące za pomocą funkcji kosztu, która szacuje koszt według czasu odpowiedzi (zależnego od procesora, operacji wejścia-wyjścia i opóźnienia sieciowego) i przewidywanej wielkości wyników pośrednich. Te ostatnie są ważne zwłaszcza w zapytaniach obejmujących złączenia. Po obliczeniu kosztu dla każdej strategii kandydującej optymalizator wybiera tę z nich, dla której koszt wykonania jest najniższy. Na każdy lokalny SZBD składają się: optymalizator zapytań lokalnych, menedżer transakcji i silnik wykonawczy, a także lokalny katalog systemowy, gdzie znajdują się schematy lokalne. Globalny menedżer transakcji odpowiada za koordynację wykonania w wielu węzłach i współdziała z lokalnymi menedżerami transakcji z tych węzłów.

### 23.7.3. Architektura federacyjnych baz danych

Typowa pięciowarstwowa architektura charakterystyczna dla globalnych systemów federacyjnych jest pokazana na rysunku 23.9. W takiej architekturze **schemat lokalny** jest schematem koncepcyjnym (pełną definicją bazy danych) dla składowej bazy danych, a **schemat komponentu** jest uzyskiwany poprzez zapis lokalnego schematu w kanonicznym modelu danych lub wspólnym modelu danych (*CDM* — ang. *common data model*) systemu federacyjnego. Tłumaczeniu schematu lokalnego na schemat komponentu towarzyszy tworzenie odwzorowań ogólnych komend schematu komponentu na komendy lokalne. **Schemat eksportu** przedstawia podzbiór schematu komponentu dostępny dla pozostałych węzłów FSBD. **Schemat federacyjny** jest globalnym schematem systemu, będącym wynikiem połączenia wszystkich współdzielonych schematów eksportu. **Schematy zewnętrzne** określają schematy dla grup użytkowników lub aplikacji, podobnie jak w architekturze trójwarstwowej.



RYSUNEK 23.8. Zarys architektury rozproszonych baz danych



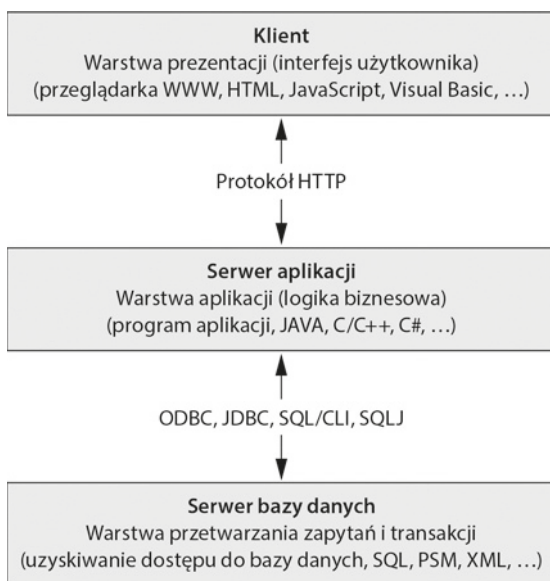
RYSUNEK 23.9. Pięciowarstwowa architektura w federacyjnym systemie baz danych. Źródło: podane za Sheth i Larson: Federated Database Systems for Managing Distributed Heterogeneous Autonomous Databases. ACM Computing Surveys (Tom 22., Nr 3., wrzesień 1990)



Wszystkie problemy związane z przetwarzaniem zapytań i transakcji, a także z zarządzaniem katalogiem i metadanymi oraz odtwarzaniem danych, mają zastosowanie również do federacyjnych systemów baz danych. Omówienie ich wszystkich szczegółowo wykracza poza ramy niniejszej książki.

### 23.7.4. Przegląd trójwarstwowej architektury klient-serwer

Jak zostało już wspomniane we wprowadzeniu do rozdziału, nie stworzono w pełni rozproszonego systemu baz danych wyposażonego w pełną funkcjonalność opisaną wcześniej. W zamian dostępne są rozproszone aplikacje bazodanowe pracujące w architekturze klient-serwer. Dwuwarstwowa architektura tego typu została już wprowadzona w podrozdziale 2.5. Obecnie częściej spotyka się architekturę trójwarstwową, szczególnie w aplikacjach dostępnych przez WWW. Architektura taka została przedstawiona na rysunku 23.10.



RYSUNEK 23.10. Trójwarstwowa architektura klient-serwer

W trójwarstwowej architekturze klient-serwer wyróżniamy następujące warstwy:

- (1) **Warstwa prezentacji (klient).** Odpowiada ona za skonstruowanie interfejsu i za interakcję z użytkownikiem. Programy tej warstwy oferują klientowi dostęp do aplikacji za pośrednictwem interfejsów i formularzy WWW. Często wykorzystywane są tutaj przeglądarki WWW, a stosowane języki to HTML, XHTML, CSS, Flash, MathML, Scalable Vector Graphics (SVG), Java, JavaScript, Adobe Flex itp. Ta warstwa akceptuje dane wejściowe, dane wyjściowe i nawigację. Przyjmuje polecenia użytkownika i wyświetla żądane informacje zazwyczaj w formie statycznych lub dynamicznych (w przypadku interakcji obejmującej bazę danych) stron WWW. Jeżeli aplikacja korzysta z interfejsu przez WWW, to warstwa prezentacji komunikuje się z warstwą aplikacji za pośrednictwem protokołu HTTP.

- (2) **Warstwa aplikacji (logika biznesowa).** Ta warstwa odpowiada za logikę aplikacji. Są tu na przykład formułowane, na podstawie danych wejściowych z klienta, zapytania do bazy danych, a także formatowane wyniki zapytań przed wysłaniem do klienta w celu zaprezentowania użytkownikowi. Warstwa aplikacji może też realizować dodatkowe funkcje programowe takie jak kontrola zabezpieczeń, weryfikacja tożsamości użytkownika i inne. Może współpracować z jednym lub kilkoma bazami danych lub źródłami danych za pośrednictwem różnych technologii dostępu do baz danych, np. ODBC, JDBC, SQL/CLI itd.
- (3) **Serwer bazy danych.** Ta warstwa obsługuje zapytania i żądania aktualizacji danych z warstwy aplikacji, przetwarza zapytania i wysyła ich wyniki. Jeżeli używany jest relacyjny lub relacyjno-obiektowy system baz danych, to warstwa ta wykorzystuje zwykle język SQL; ponadto może też używać składowanych procedur bazy danych. Wyniki zapytań (i same zapytania) mogą być podczas przesyłania pomiędzy warstwą aplikacji a serwerem baz danych sformułowane w języku XML (patrz rozdział 13.).

Szczegóły podziału pomiędzy klienta, serwer aplikacji i serwer baz danych mogą być różne. Popularnym rozwiązaniem jest zawarcie funkcjonalności scentralizowanego systemu bazy danych na poziomie serwera bazy danych. Taką taktykę przyjęło wielu producentów relacyjnych SZBD, oferując **serwer SQL**. Serwer aplikacji musi w takiej sytuacji sformułować odpowiednie zapytanie SQL i w miarę potrzeby połączyć się z serwerem bazy danych. Klient jest odpowiedzialny za interakcję z użytkownikiem. Ponieważ SQL jest standardem dla relacyjnych baz danych, to różne serwery baz danych akceptują polecenia SQL przesyłane za pośrednictwem różnych standardów komunikacyjnych, takich jak ODBC, JDBC, SQL/CLI (patrz rozdział 10.).

W przedstawionej architekturze serwer aplikacji może korzystać ze słownika danych zawierającego informacje o rozproszeniu danych na różnych serwerach SQL, a także z modułów rozkładających globalne zapytania na szereg podzapytań, z których każde może być wykonane w innym węźle. Interakcja pomiędzy serwerem aplikacji a serwerem baz danych podczas przetwarzania zapytania SQL może przebiegać następująco:

- (1) Serwer aplikacji formułuje zapytanie w oparciu o dane wejściowe dostarczone przez warstwę klienta, a następnie rozkłada je na kilka niezależnych podzapytań. Każde z nich jest wysyłane do odpowiedniego węzła serwera bazy danych.
- (2) Każdy serwer bazy danych przetwarza lokalne zapytanie i wysyła wyniki do serwera aplikacji. Coraz częściej jako standardowy mechanizm wymiany danych wykorzystywany jest tu język XML (patrz rozdział 13.), więc serwer bazy danych może przed przesłaniem wyniku zapytania sformatować go jako dokument XML.
- (3) Serwer aplikacji łączy wyniki podzapytań w wynik oryginalnego zapytania, formatuje go jako dokument HTML lub w innej formie akceptowanej przez klienta i wysyła do warstwy prezentacji w celu przedstawienia użytkownikowi.

Serwer aplikacji jest odpowiedzialny za stworzenie planu rozproszonego wykonania zapytania lub transakcji i za nadzorowanie wykonania komend przez poszczególne serwery. Komendy te zawierają lokalne zapytania i transakcje, a także polecenia przesyłające dane do innych klientów lub serwerów. Inną funkcją kontrolowaną przez serwer aplikacji (lub koordynatora) jest zapewnienie spójności replikowanych kopii danych poprzez wdraża-

nie rozproszonych (lub globalnych) technik sterowania współbieżnego. Serwer aplikacji musi również zapewniać niepodzielność globalnych transakcji i podejmować odpowiednie akcje w przypadku awarii poszczególnych węzłów.

Jeżeli rozproszony system bazy danych ma możliwość *ukrywania* szczegółów rozproszenia danych przed serwerem aplikacji, to ten ostatni może wykonywać globalne zapytania i transakcje tak, jakby dotyczyły jednej scentralizowanej bazy danych, czyli bez określania konkretnych węzłów przechowujących dane. Taka własność jest nazywana **przezroczystością rozproszenia**. Niektóre rozproszone systemy baz danych nie obsługują przezroczystości rozproszenia i wymagają, aby aplikacje znały szczegóły rozmieszczenia danych.

## 23.8. Zarządzanie rozproszonym katalogiem

Wydajne zarządzanie katalogiem w rozproszonych bazach danych jest niezbędne do osiągnięcia zadowalającej wydajności w zakresie autonomii węzłów, zarządzania perspektywami oraz rozproszenia i replikacji danych. Katalogi same są bazami danych i zawierają metadane na temat rozproszonych systemów baz danych.

Trzy popularne modele katalogów rozproszonych to katalogi *scentralizowane*, katalogi *z pełną replikacją* i katalogi *z replikacją częściową*. Wybór modelu zależy od bazy danych i od występujących w aplikacji wzorców dostępu do danych.

**Katalogi scentralizowane.** W tym modelu cały katalog jest przechowywany w jednym węźle. Z powodu scentralizowanego charakteru rozwiązanie to łatwo jest zaimplementować. Negatywnie wpływa ono jednak na stabilność, dostępność, autonomię i podział obciążenia roboczego. Na potrzeby operacji odczytu z węzłów niecentralnych żądane dane z katalogu są blokowane w węźle centralnym, a następnie przesyłane do żądającego ich węzła. Po zakończeniu operacji odczytu do węzła centralnego przesyłane jest potwierdzenie, co powoduje odblokowanie danych. Wszystkie operacje aktualizacji muszą być przetwarzane z użyciem węzła centralnego. Może on szybko stać się wąskim gardłem w aplikacjach z wieloma operacjami zapisu.

**Katalogi z pełną replikacją.** W tym modelu w każdym węźle dostępne są identyczne kopie całego katalogu. Ten model przyspiesza odczyty, ponieważ można je obsługiwać lokalnie. Jednak wszystkie aktualizacje muszą być rozsyłane do każdego węzła. Aktualizacje są traktowane jak transakcje, a w celu zapewnienia spójności katalogu używany jest scentralizowany proces zatwierdzania dwuetapowego. Podobnie jak w modelu ze scentralizowanym katalogiem, tak i tu aplikacje z wieloma operacjami zapisu mogą powodować wzrost ruchu w sieci dotyczący rozsyłania danych powiązanych z tymi operacjami.

**Katalogi z replikacją częściową.** Modele scentralizowany i z pełną replikacją ograniczają autonomię węzłów, ponieważ muszą gwarantować spójny globalny obraz katalogu. W modelu z replikacją częściową każdy węzeł przechowuje kompletny katalog z informacjami na temat lokalnie składowanych danych. Każdy węzeł może też zapisywać w pamięci podręcznej wpisy ze zdalnych węzłów.

Nie ma jednak gwarancji, że kopie w pamięci podręcznej to najnowsze i zaktualizowane dane. System śledzi wpisy w katalogach węzłów, gdzie dany obiekt został utworzony, i w węzłach obejmujących kopie danego obiektu. Wszystkie modyfikacje kopii są natychmiast przesyłane do pierwotnego (źródłowego) węzła. Pobieranie zaktualizowanych kopii w celu zastąpienia nieaktualnych danych może być odraczane do momentu dostępu do określonych danych. Zwykle możliwy powinien być jednoznaczny dostęp do fragmentów relacji z różnych węzłów. Ponadto aby zapewnić przezroczystość rozproszenia danych, możliwe powinno być tworzenie przez użytkowników synonimów zdalnych obiektów i korzystanie z tych synonimów w późniejszych odwołaniach.

## 23.9. Podsumowanie

Niniejszy rozdział stanowił wprowadzenie do tematyki rozproszonych baz danych. Jest to bardzo szeroki temat i omówione zostały tylko podstawowe techniki używane przez tego typu systemy. Najpierw, w podrozdziale 23.1, omówiliśmy przyczyny podziału baz i zagadnienia z obszaru rozproszonych baz danych (punkt 23.1.1). Następnie przedstawione zostało pojęcie przezroczystości rozproszenia i powiązane z nim przezroczystość fragmentacji i replikacji (punkt 23.1.2). W punkcie 23.1.3 opisano dostępność i niezawodność w środowisku rozproszonym, a w punkcie 23.1.4 — skalowalność i odporność na podział. Autonomia węzłów w systemach rozproszonych została przedstawiona w punkcie 23.1.5, a zalety baz rozproszonych w porównaniu z systemami scentralizowanymi zaprezentowano w punkcie 23.1.6.

W podrozdziale 23.2 omówiono kwestie projektowe związane z fragmentacją, replikacją i rozpraszaniem danych. W punkcie 23.2.1 wyróżniono poziomą (sharding) i pionową fragmentację relacji. Następnie, w punkcie 23.2.2, objaśniono stosowanie replikacji danych do poprawy stabilności i dostępności systemów. W podrozdziale 23.3 pokrótce opisano techniki sterowania współbieżnego i odtwarzania danych stosowane w rozproszonych SZBD. Dalej przedstawiono przegląd wybranych dodatkowych problemów, z którymi trzeba sobie radzić w środowisku rozproszonym, a które nie występują w środowisku scentralizowanym. W podrozdziale 23.4 opisano zarządzanie transakcjami, w tym różne protokoły zatwierdzania (dwufazowy i trójfazowy), a także obsługę zarządzania transakcjami w systemie operacyjnym.

Kolejno przedstawiono wybrane techniki przetwarzania zapytań rozproszonych (podrozdział 23.5). Omówiono też koszty komunikacji między węzłami. Koszty te uważa się za istotny czynnik w optymalizacji zapytań rozproszonych. Porównano też różne techniki wykonywania złączeń, a następnie zaprezentowano metodę złączeń częściowych relacji znajdujących się w różnych węzłach (punkt 23.5.3).

Następnie, w podrozdziale 23.6, podzielono rozproszone SZBD na podstawie takich kryteriów jak stopień homogeniczności modułów oprogramowania i poziom lokalnej autonomii. W podrozdziale 23.7 dokonano podziału na architektury systemów równoległych i rozproszonych, a potem przedstawiono ogólną architekturę rozproszonych baz danych na podstawie komponentów i schematycznej architektury. W punkcie 23.7.3 omówiono zarządzanie federacyjnymi bazami danych. Skoncentrowano się tam na konieczności

zapewniania autonomii różnego rodzaju i radzeniu sobie z heterogenicznością semantyczną. W punkcie 23.7.4 znalazł się przegląd zagadnień z obszaru architektury klient-serwer, które powiązano z bazami rozproszonymi. Podrozdział 23.8 to omówienie zarządzania katalogiem w rozproszonych bazach danych oraz podsumowanie względnych zalet i wad takich baz.

W rozdziałach 24. i 25. opiszemy najnowsze osiągnięcia w zakresie rozproszonych baz danych i przetwarzania rozproszonego w kontekście big data. Rozdział 24. to omówienie tzw. systemów NOSQL. Są to wysoce skalowalne rozproszone systemy baz danych obsługujące duże zbiory danych. W rozdziale 25. przedstawimy technologie przetwarzania w chmurze i przetwarzania rozproszonego potrzebne do radzenia sobie z big data.

## Pytania powtórkowe

- 23.1. Jakie są główne powody stosowania i zalety rozproszonych baz danych?
- 23.2. Jakie dodatkowe funkcje posiadają rozproszone SZBD w porównaniu ze scentralizowanymi?
- 23.3. Omów znaczenie następujących terminów: *stopień homogeniczności rozproszonego SZBD, stopień lokalnej autonomii rozproszonego SZBD, federacyjny system baz danych, przezroczystość rozproszenia, przezroczystość fragmentacji, przezroczystość replikacji, systemy wielobazodanowe.*
- 23.4. Omów architekturę rozproszonego SZBD. W kontekście scentralizowanego SZBD pokrótce wyjaśnij nowe komponenty potrzebne z powodu rozproszenia danych.
- 23.5. Jakie są główne moduły programowe rozproszonych systemów baz danych? Opisz funkcje każdego z nich w kontekście architektury klient-serwer?
- 23.6. Porównaj architektury dwu- i trójwarstwowe.
- 23.7. Co nazywamy fragmentem relacji? Jakie są podstawowe rodzaje fragmentów? Dlaczego fragmentacja jest przydatna przy projektowaniu rozproszonych baz danych?
- 23.8. Do czego służy replikacja w rozproszonych bazach danych? Jakie jednostki danych są replikowane?
- 23.9. Co rozumiemy pod pojęciem *alokacji danych* w systemach rozproszonych? Jakie jednostki danych są poddawane alokacji?
- 23.10. Jak jest definiowana fragmentacja pozioma relacji? Jak można odtworzyć oryginalną relację po takiej fragmentacji?
- 23.11. Jak jest definiowana fragmentacja pionowa relacji? Jak można odtworzyć oryginalną relację po takiej fragmentacji?
- 23.12. Omów problem nazw w rozproszonych bazach danych.
- 23.13. Podaj etapy przetwarzania zapytań w rozproszonych SZBD.
- 23.14. Omów różne techniki równozłączeń dwóch plików umieszczonych w różnych węzłach. Jakie główne czynniki wpływają na koszt przesyłu danych?
- 23.15. Omów metodę złączenia częściowego przy wykonywaniu równozłączenia dwóch plików umieszczonych w różnych węzłach. W jakich warunkach strategia równozłączenia jest wydajna?

- 23.16. Określ czynniki wpływające na dekompozycję zapytania. Jak podczas procesu dekompozycji zapytania używane są listy atrybutów i strażnicy fragmentów relacji?
- 23.17. Czym różni się dekompozycja transakcji aktualizującej dane od dekompozycji zapytania? Jakie warunki i listy atrybutów fragmentów są używane podczas przetwarzania żądania aktualizacji danych?
- 23.18. Omów funkcje udostępniane przez system operacyjny z rozproszonym SZBD i płynące z tego korzyści.
- 23.19. Omów czynniki niewystępujące w scentralizowanych bazach danych, które mają wpływ na mechanizmy sterowania współbieżnego i przywracania danych w systemach rozproszonych.
- 23.20. Omów protokół zatwierdzania dwufazowego używany do zarządzania transakcjami w rozproszonych SZBD. Wymień jego ograniczenia i wyjaśnij, w jaki sposób zostały one przezwyciężone w protokole zatwierdzania trójfazowego.
- 23.21. Porównaj metodę podstawowego węzła z metodą podstawowej kopii w przypadku rozproszonego sterowania współbieżnego. Jaki wpływ na każdą z tych metod ma zastosowanie węzłów zapasowych?
- 23.22. Kiedy w rozproszonych bazach danych używane są głosowanie i wybory?
- 23.23. Omów zarządzanie katalogiem w rozproszonych bazach danych.
- 23.24. Opisz główne wyzwania stojące przed tradycyjnymi rozproszonymi SZBD w kontekście dzisiejszych aplikacji internetowych. W jaki sposób próbuje się uwzględnić te wyzwania za pomocą chmury obliczeniowej?
- 23.25. Omów pokrótce obsługę architektur homogenicznych, heterogenicznych i typu klient-serwer w bazach Oracle.
- 23.26. Opisz pokrótce katalogi, zarządzanie nimi i ich rolę w rozproszonych bazach danych.

## Ćwiczenia

- 23.27. Rozważmy rozproszenie danych w bazie danych FIRMA, której fragmenty w węzłach 2. i 3. są pokazane na rysunku 23.3, a fragmenty z węzła 1. są pokazane na rysunku 3.6. Dla każdego z niżej zamieszczonych zapytań przedstaw co najmniej dwie strategie dekompozycji i wykonania zapytania. W jakich warunkach te strategie będą działały wydajnie?
  - a) Dla każdego pracownika działu 5. odczytaj nazwisko pracownika oraz członków jego rodziny.
  - b) Wyświetl nazwiska wszystkich pracowników, którzy pracują w dziale nr 5 i biorą udział w projekcie *nienadzorowanym* przez swój dział.
- 23.28. Rozważ następujące relacje:

KSIĄŻKI (Książka#, Pierwszy\_autor, Temat, Na\_stanie, Cena)

KSIĘGARNIA (Sklep#, Miasto, Województwo, Kod\_pocztowy, Wartość\_inwentarza)

MAGAZYN (Sklep#, Książka#, Liczba)

NA\_STANIE jest liczbą wszystkich książek w magazynie, a WARTOŚĆ\_INWENTARZA jest sumaryczną wartością wszystkich książek w sklepie.

- a) Podaj przykład dwóch prostych predykatów, których można użyć do fragmentacji poziomej relacji KSIĘGARNIA.
- b) Jak można określić fragmentację poziomą pochodną relacji MAGAZYN na podstawie podziału relacji KSIĘGARNIA?
- c) Przedstaw predykaty, według których można podzielić poziomo relację KSIĄŻKI według tematu.
- d) Opisz, jak można dalej podzielić fragmenty relacji MAGAZYN wprowadzone w podpunkcie (b), dodając predykaty (c).

23.29. Rozważmy rozproszoną bazę danych dla sieci księgarń Polskie Książki z trzema oddziałami PÓŁNOC, CENTRUM, POŁUDNIE. Schematy relacji zostały podane w ćwiczeniu 23.28. Przyjmijmy, że KSIĄŻKI są podzielone według atrybutu Cena na:

B1: KSIĄŻKI1: do 20 zł

B2: KSIĄŻKI2: od 20,01 zł do 50 zł

B3: KSIĄŻKI3: od 50,01 zł do 100 zł

B4: KSIĄŻKI4: od 100,01 zł wzwyż

Przyjmijmy, że KSIĘGARNIE mogą być podzielone według kodów pocztowych:

S1: PÓŁNOC: kody od 70-000.

S2: CENTRUM: kody pocztowe do 26-999.

S3: POŁUDNIE: kody pocztowe od 27-000 do 69-999

Przypuśćmy, że MAGAZYN jest dzielony według fragmentacji pochodnej opartej na fragmentacji relacji KSIĘGARNIA.

- a) Rozważ następujące zapytanie:

```
SELECT Książka#, Na_stanie
FROM Książki
WHERE cena>15 AND cena<55;
```

Przyjmijmy, że fragmenty relacji KSIĘGARNIA nie są replikowane i alokacja następuje według regionu. Przyjmijmy też, że KSIĄŻKI są alokowane następująco:

PÓŁNOC: B1, B4

CENTRUM: B1, B2

POŁUDNIE: B1, B2, B3, B4

Zakładając, że zapytanie zostało wysłane na północy, jakie zdalne zapytania spowoduje? Napisz je w SQL.

- b) Jeśli cena książki o atrybucie Książka# = 1234 jest zmieniona z 45 zł na 55 zł w węźle CENTRUM, to jakie to spowoduje aktualizacje? Opisz je własnymi słowami i w SQL.
- c) Podaj przykładowe zapytanie, które uruchomione w węźle POŁUDNIE spowoduje wykonanie podzapytania w węźle CENTRUM.



- d) Napisz zapytanie stosujące operacje selekcji i projekcji dotyczące przytoczonych powyżej przykładowych relacji i narysuj dwa możliwe drzewa zapytań ilustrujące różne sposoby jego wykonania.
- 23.30. Przypuśćmy, że poproszono Cię o zaproponowanie architektury bazy danych w dużej firmie (na przykład General Motors). Projektowana baza danych ma łączyć wszystkie dane ze starszych systemów (w modelach hierarchicznym i sieciowym; nie jest wymagana znajomość tych modeli) oraz z systemów relacyjnych rozproszonych w oddziałach firmy. Baza ma być używana w globalnych aplikacjach. Załóżmy, że pierwsza możliwość polega na pozostawieniu dotychczasowych baz danych bez zmian, a druga na zmianie ich na systemy relacyjne i połączeniu w rozproszony system bazy danych.
- a) Narysuj dwa schematyczne diagramy dla powyższych możliwości, pokazujące połączenia pomiędzy poszczególnymi węzłami. Dla pierwszej możliwości wybierz sposób opracowania schematów eksportu danych dla każdej bazy danych oraz stworzenia zunifikowanego schematu dla każdej aplikacji.
- b) Wymień kroki, które trzeba podjąć dla każdej z dwóch przedstawionych możliwości, aby przejść od aktualnej sytuacji do dostępności globalnych aplikacji.
- c) Porównaj je z problemami związanymi z:
- (i) fazą projektowania systemu,
  - (ii) pracą systemu.

## Wybrane publikacje

Rozproszonym bazom danych poświęcone są książki Ceriego i Pelagattiego (1984a) oraz Ozsu i Valduriez (1999). Publikacje Petersona i Davie'go (2008), Tannenbauma (2003) i Stallinsa (2007) traktują o sieciach komputerowych i wymianie danych. Comer (2008) omawia kwestie sieci i internetu. Ozsu i in. (1994) napisali szereg prac na temat zarządzania obiektami w środowisku rozproszonym.

Większość badań nad projektowaniem baz danych, przetwarzaniem zapytań i ich optymalizacją w środowisku rozproszonym przeprowadzono w latach 80. i 90. Tu pokrótce wymienimy ważne prace. Projektowanie rozproszonych baz danych jest opracowane jako kwestie związane z poziomą i pionową fragmentacją danych, alokacją i replikacją. Ceri i in. (1982) definiuje pojęcie fragmentów poziomych. Ceri i in. (1983) opracował model optymalizacji alokacji i fragmentacji poziomej oparty na programowaniu całkowitoliczbowym. Navathe i in. (1984) przedstawia algorytmy fragmentacji pionowej oparte na podobieństwie atrybutów i pokazuje różnorodne konteksty ilustrujące alokacje pionowych fragmentów relacji. Wilson i Navathe (1986) prezentują model analityczny optymalnej alokacji fragmentów. Książka Elmasriego i in. (1987) omawia fragmentację w modelu ECR, a Karlapalem i in. (1996) — kwestie projektowania rozproszonych obiektowych baz danych. Navathe i in. (1996) przedstawia fragmentację mieszaną łączącą podział poziomy i pionowy; Karlapalem i in. (1996) prezentuje model przeprojektowania rozproszonych baz danych.

Rozproszone przetwarzanie zapytań, optymalizacja i dekompozycja są opisywane w pracach Hevnera i Yao (1979), Kerschberga i in. (1982), Aversa i in. (1983), Ceriego i Pelagattiego (1984) oraz Bodoricka i in. (1992). Bernstein i Goodman (1981) omawiają teorię leżącą u podstaw przetwarzania złączeń częściowych. Wong (1983) opisuje zastosowanie relacji pomiędzy tabelami przy ich fragmentacji. Sterowanie współbieżne i schematy odtwarzania danych są opisane w publikacji Bernsteina i Goodmana (1981a). Kumar i Hsu (1998) opublikowali kilka artykułów związanych z odtwarzaniem danych w rozproszonych bazach danych. Wybory w systemach rozproszonych są opisywane w pracy Garcii-Moliny (1982). Lamport (1978) omawia problemy z generowaniem unikatowych znaczników czasowych w systemach rozproszonych. Rahimi i Haug (2007) opisali bardziej elastyczny sposób tworzenia krytycznych metadanych dotyczących zapytań w bazach typu P2P. Ouzzani i Bouguettaya (2004) przedstawili podstawowe problemy z zakresu rozproszonego przetwarzania zapytań dotyczących internetowych źródeł danych.

Techniki sterowania współbieżnego w zastosowaniu do replikacji danych w oparciu o głosowanie są omówione w pracy Thomasa (1979). Gifford (1979) proponuje zastosowanie głosowania z wagami, a Paris (1986) opisuje metodę nazwaną *głosowaniem ze świadkami*. Jajodia i Mutchler (1990) omawiają głosowania dynamiczne. W pracy Bernsteina i Goodmana (1984) zaproponowana jest technika *dostępnej kopii*, a zastosowanie grup opisano w publikacji ElAbbadiego i Touega (1988). Dalsze prace dotyczące replikacji danych opublikowali Gladney (1989), Agrawal i ElAbbadiego (1990), ElAbbadiego i Toueg (1989), Kumar i Segev (1993), Mukkamala (1989) oraz Wolfson i Milo (1991). Bassiouni (1988) omawia protokoły optymistyczne sterowania współbieżnego w rozproszonych bazach danych. Książki Garcii-Moliny (1983) oraz Kumara i Stonebrakera (1987) przedstawiają techniki używające semantyki transakcji. Rozproszone sterowanie współbieżne oparte na blokadach i rozróżnianiu kopii jest zaprezentowane w publikacjach Menascego i in. (1980) oraz Minoury i Wiederholda (1982). Obermark (1982) przedstawia algorytmy do wykrywania zakleszczeń w środowisku rozproszonym. W nowszej pracy Vadivelu i in. (2008) opisują stosowanie kopii zapasowych i wielopoziomowych zabezpieczeń w algorytmach zwiększających współbieżność. Madria i in. (2007) proponują mechanizm oparty na blokowaniu dwufazowym z użyciem wielu wersji i znaczników czasu, aby rozwiązać problemy ze współbieżnością charakterystyczne dla mobilnych systemów baz danych. Boukerche i Tuck (2001) opisują technikę dopuszczającą drobne odstępstwa w kolejności wykonywania transakcji. Próbuje w ten sposób ułatwić pracę programistom aplikacji, wykorzystując środowisko sieciowe i generując harmonogram równoważny czasowo uporządkowanemu harmonogramowi szeregowemu. Han i in. (2004) proponują wolny od zakleszczeń i szeregowalny rozszerzony model sieciowy Petriego dla internetowych rozproszonych baz danych działających w czasie rzeczywistym.

Opracowanie dotyczące technik odtwarzania danych zostało opublikowane przez Kohlera (1981). Reed (1983) omawia pewne atomowe operacje na rozproszonych danych. Książka pod redakcją Bhargavy (1987) przedstawia różne metody i techniki związane ze współbieżnością i stabilnością w rozproszonych systemach baz danych.

Federacyjne systemy baz danych definiują po raz pierwszy McLeod i Heimbinger (1985). Techniki integracji schematów w federacyjnych bazach danych są przedstawione w książkach Elmasriego i in. (1986), Battiniego i in. (1986), Haynego i Rama (1990) oraz Motro (1987). Elmagarmid i Helal (1988) oraz Gamal-Eldin i in. (1988) omawiają problem ak-

tualizacji w heterogenicznych rozproszonych systemach baz danych. Problemy związane z takimi systemami rozważają również Hsiao i Kamel (1989). Sheth i Larson (1990) przedstawiają kompletne opracowanie o zarządzaniu federacyjnymi bazami danych.

Od lat 80. istotnymi tematami stały się systemy wielobazodanowe oraz interoperacyjność. Techniki rozwiązywania problemów niekompatybilności pomiędzy wieloma bazami danych przedstawiają DeMichiel (1989), Siegel i Madnick (1991), Krishnamurthy i in. (1991) oraz Wang i Madnick (1989). Castano i in. (1998) opublikował wyśmienite opracowanie technik analizowania schematów. Pitoura i in. (1995) opisują obiektowość w systemach wielobazodanowych. Xiao i in. (2003) proponują oparty na formacie XML wspólny model danych dla systemów wielobazodanowych i prezentują nowe, bazujące na tym modelu podejście do odwzorowywania schematów. Lakshmanan i in. (2001) opisują rozszerzenie języka SQL pod kątem interoperacyjności oraz omawiają architekturę i algorytmy pozwalające osiągnąć ten cel.

Przetwarzanie transakcji w systemach wielobazodanowych omawiają w swoich pracach między innymi Mehrotra i in. (1992), Georgakopoulos i in. (1991), Elmagarmid i in. (1990) oraz Brietbart i in. (1990). Elmagarmid i in. (1992) skupia się na przetwarzaniu transakcji w zaawansowanych aplikacjach takich jak aplikacje inżynierskie, omawiane również w pracy Heilera i in. (1992).

Coraz popularniejsze w firmach systemy obiegu danych używają wielopoziomowych i zagnieżdżonych transakcji w połączeniu z rozproszonymi bazami danych. Weikum (1991) omawia zarządzanie transakcjami wielopoziomowymi, a Alonso i in. (1997) — ograniczenia obecnych systemów tego typu. Lopes i in. (2009) proponują, aby użytkownicy definiowali i wykonywali własne obiegi danych za pomocą przeglądarek. Autorzy starają się wykorzystać trendy z nurtu Web 2.0, aby uprościć zarządzanie obiegiem danych przez użytkowników. Jung i Yeom (2008) wykorzystują obieg danych do opracowania usprawnionego systemu zarządzania transakcjami, zapewniającego jednoczesny płynny dostęp do heterogenicznych magazynów danych tworzących system HVEM DataGrid. Deelman i Chervanak (2008) opisują trudności z obszaru obiegu pracy w aplikacjach naukowych intensywnie przetwarzających dane. Koncentrują się na zautomatyzowanym zarządzaniu danymi, wydajnych technikach odwzorowywania i informacjach zwrotnych od użytkowników w ramach odwzorowywania obiegu danych. Autorzy ci twierdzą, że ponowne wykorzystanie danych jest wydajnym sposobem na zarządzanie nimi, i przedstawiają wyzwania z tego obszaru.

Powstało kilka eksperymentalnych rozproszonych systemów baz danych. Są to między innymi INGRES (Epstein i in., 1978), DDTS (Devor i Weeldreyer, 1980), SDD-1 (Rothnie i in., 1980), System R\* (Lindsay i in., 1984), SIRIUS-DELTA (Ferrier i Stangret, 1982) oraz MULTIBASE (Smith i in., 1981). Przykładami federacyjnych systemów rozproszonych są OMNIBASE (Rusinkiewicz i in., 1988) oraz Federated Information Base opracowany w oparciu o model danych Candide (Navathe i in., 1994). Pitoura i in. (1995) przedstawia opracowanie porównawcze prototypów systemów federacyjnych. Większość producentów komercyjnych systemów baz danych oferuje programy pracujące w architekturze klient-serwer i wersje rozproszone swoich produktów. Niektóre kwestie systemowe dotyczące architektury klient-serwer systemów SZBD omówiono w pozycji Careya i in. (1991), DeWitta i in. (1990) oraz Wanga i Rowe'a (1991). Khoshafian i in. (1992) omawia kwestie projektowe dotyczące relacyjnych SZBD w środowisku klient-serwer. Kwestie

związane z zarządzaniem systemami klasy klient-serwer omówiono w wielu książkach, między innymi Zantingego i Adriaansa (1996). Di Stefano (2005) omawia kwestie związane z rozproszeniem danych specyficzne dla przetwarzania siatkowego (ang. *grid computing*). Duża część tej pracy dotyczy także przetwarzania w chmurze.

## Bazy danych NOSQL i systemy składowania big data

Teraz skupimy się na klasie systemów opracowanych na potrzeby zarządzania dużymi zbiorami danych w firmach takich jak Google, Amazon, Facebook i Twitter oraz w aplikacjach takich jak sieci społecznościowe lub systemy zarządzania odsyłaczami, profilami użytkownika, danymi marketingowymi i sprzedażowymi, postami i tweetami, mapami drogowymi, danymi przestrzennymi i e-mailami. Nazwa **NOSQL** jest zwykle interpretowana jako *Not Only SQL* (a nie *NO to SQL*) i ma oznaczać, że wiele aplikacji do wspomagania zarządzania danymi wymaga systemów innych niż tradycyjne relacyjne systemy oparte na języku SQL. Większość systemów NOSQL to rozproszone bazy danych lub systemy składowania danych, opracowane głównie z myślą o składowaniu częściowo ustrukturyzowanych danych, wysokiej wydajności, dostępności, replikacji danych i skalowalności, a nie o natychmiastowej spójności danych, rozbudowanych językach zapytań i składowaniu danych ustrukturyzowanych.

Zacniemy od przedstawienia w podrozdziale 24.1 wprowadzenia do systemów NOSQL, ich scharakteryzowania i pokazania, pod jakimi względami różnią się od systemów opartych na języku SQL. Scharakteryzujemy też cztery ogólne kategorie systemów NOSQL: dokumentowe, z parami klucz-wartość, kolumnowe i grafowe. W podrozdziale 24.2 opiszemy, jak w systemach NOSQL za pomocą modelu **spójności ostatecznej** poradcono sobie z kwestią spójności między wieloma replikami (kopiami). Omówimy też twierdzenie **CAP**, które pomaga zrozumieć nacisk kładziony w systemach NOSQL na dostępność. W podrozdziałach od 24.3 do 24.6 przedstawimy wszystkie kategorie systemów NOSQL. Zacniemy od systemów dokumentowych, a następnie przejdziemy do systemów z parami klucz-wartość, kolumnowych i grafowych. Niektóre systemy nie pasują precyzyjnie do jednej kategorii i wykorzystują techniki z przynajmniej dwóch klas systemów NOSQL. Podrozdział 24.7 zawiera podsumowanie rozdziału.

## 24.1. Wprowadzenie do systemów NOSQL

### 24.1.1. Powstanie systemów NOSQL

Wiele firm i organizacji korzysta z aplikacji przechowujących duże ilości danych. Pomyśl o bezpłatnych aplikacjach do obsługi poczty elektronicznej (takich jak Google Mail lub Yahoo! Mail) lub podobnych usługach. Takie aplikacje mogą mieć miliony użytkowników,

z których każdy przechowuje tysiące e-maili. Potrzebny jest system składowania, który potrafi zarządzać tymi e-mailami. Ustrukturyzowany relacyjny system SQL-owy nie będzie odpowiedni, ponieważ: (1) takie systemy udostępniają zbyt wiele usług (rozbudowany język zapytań, sterowanie współbieżne itd.), których dana aplikacja może nie potrzebować, i (2) ustrukturyzowany model danych, taki jak tradycyjny model relacyjny, może być zbyt restrykcyjny. Choć nowsze systemy relacyjne udostępniają bardziej złożone modele obiektowo-relacyjne (patrz rozdział 12.), nadal wymagają schematów, które w wielu systemach NOSQL nie są potrzebne.

Oto następny przykład: rozważ aplikację taką jak Facebook, w której miliony użytkowników przesyłają posty, często ze zdjęciami i filmami. Takie posty muszą być wyświetlane na stronach innych użytkowników na podstawie zapisanych w sieci społecznościowej związków między tymi osobami. Profile użytkowników, związki pomiędzy osobami i posty muszą być przechowywane w wielkich magazynach danych, a odpowiednie posty muszą być udostępniane grupom użytkowników, którzy przejawili zainteresowanie danymi informacjami. Niektóre dane w aplikacjach tego typu nie nadają się do przechowywania w tradycyjnych systemach relacyjnych i zwykle wymagają kilku rodzajów baz oraz systemów składowania danych.

Pewne organizacje korzystające z takich aplikacji do zarządzania danymi i składowania ich zdecydowały się opracować własne systemy:

- Firma Google opracowała niestandardowy system NOSQL o nazwie **BigTable**. Jest on stosowany w wielu aplikacjach tej firmy (takich jak Gmail, Google Maps i indeksowanie witryn), które wymagają dużo miejsca na dane. Apache HBase to otwarty system NOSQL o podobnych zasadach działania. Innowacje firmy Google doprowadziły do powstania **kolumnowych** (ang. *column-based*, *wide column* lub *column family*) systemów NOSQL.
- Firma Amazon zbudowała system NOSQL o nazwie **DynamoDB**. Jest on dostępny w usługach działających w chmurze tej firmy. Te innowacje doprowadziły do powstania magazynów danych z **parami klucz-wartość** (ang. *key-value*, *key-tuple* lub *key-object*).
- Facebook opracował system NOSQL o nazwie **Cassandra**. Obecnie jest on dostępny jako otwarty system Apache Cassandra. Ten system NOSQL wykorzystuje mechanizmy z systemów kolumnowych i z parami klucz-wartość.
- Inni producenci oprogramowania zaczęli rozwijać własne rozwiązania i udostępniać je użytkownikom potrzebującym określonych funkcji. Te rozwiązania to np. bazy **MongoDB** i **CouchDB** należące do kategorii **dokumentowych** systemów NOSQL (ang. *document-based* lub *document stores*).
- Inną kategorią systemów NOSQL są systemy **grafowe** (ang. *graph-based* lub *graph databases*). Należą do nich m.in. systemy **Neo4J** i **GraphBase**.
- Niektóre systemy NOSQL, np. **OrientDB**, łączą rozwiązania z wielu opisanych wyżej kategorii.
- Oprócz wymienionych wcześniej nowych typów systemów NOSQL do tej grupy można też przypisać systemy baz danych oparte na modelu obiekowym (patrz rozdział 12.) lub natywnym modelu XML (patrz rozdział 13.), choć nie zawsze cechują się one wysoką wydajnością i możliwościami replikacji typowymi dla innych rodzajów systemów NOSQL.

To tylko kilka przykładów opracowanych systemów NOSQL. Istnieje wiele systemów tego typu, a wymienianie ich wszystkich wykracza poza zakres tej książki.

## 24.1.2. Cechy systemów NOSQL

Teraz omówimy cechy wielu systemów NOSQL. Pokażemy też, jak te systemy różnią się od tradycyjnych systemów SQL-owych. Cechy podzielimy na dwie grupy: związane z rozproszonymi bazami danych i systemami oraz dotyczące modeli danych i języków zapytań.

### Cechy systemów NOSQL związane z rozproszonymi bazami danych i systemami.

W systemach NOSQL nacisk położony jest na wysoką dostępność, dlatego nieodłącznym aspektem wielu takich systemów jest replikacja. Inną ważną cechą jest skalowalność, ponieważ liczne aplikacje używające systemów NOSQL przechowują dane, których ilość wciąż rośnie. Inną pożądaną cechą jest wysoka wydajność, natomiast spójność sekwencyjna (związana z szeregowalnością) może być w niektórych aplikacjach z obszaru NOSQL mało istotna. Oto niektóre z ważniejszych cech:

- (1) **Skalowalność.** W punkcie 23.1.4 wspomniano, że w systemach rozproszonych występują dwa rodzaje skalowalności: pozioma i pionowa. W systemach NOSQL zwykle stosowana jest **skalowalność pozioma**, gdzie system rozproszony wraz ze wzrostem ilości danych jest powiększany w wyniku dodawania węzłów służących do przechowywania i przetwarzania danych. Z kolei **skalowalność pionowa** dotyczy rozbudowywania pamięci i mocy obliczeniowej istniejących węzłów. W systemach NOSQL skalowalność pozioma jest stosowana w czasie ich pracy, dlatego niezbędne są techniki przesyłania istniejących danych do nowych węzłów bez zakłócenia działania systemu. Niektóre z tych technik opiszemy w podrozdziałach od 24.3 do 24.6 w trakcie omawiania konkretnych systemów.
- (2) **Dostępność, replikacja i spójność ostateczna.** W wielu aplikacjach używających systemów NOSQL niezbędna jest ciągła dostępność systemu. Aby ją uzyskać, dane są automatycznie replikowane w przynajmniej dwóch węzłach, dlatego gdy jeden z nich zawiedzie, dane będą dostępne w pozostałych. Replikacja zwiększa dostępność danych, a ponadto czasem pozwala poprawić wydajność odczytu, ponieważ żądania odczytu często można obsługiwać za pomocą dowolnego węzła ze zreplikowanymi danymi. Jednak wydajność zapisu może spaść, ponieważ aktualizacje trzeba wprowadzać we wszystkich kopiach zreplikowanych elementów danych. Jeśli wymagana jest spójność sekwencyjna (patrz podrozdział 23.3), może to skutkować spadkiem wydajności zapisu. W wielu aplikacjach NOSQL spójność sekwencyjna nie jest wymagana, dlatego stosowana jest mniej restrykcyjna **spójność ostateczna**. Omawiamy ją szczegółowo w podrozdziale 24.2.
- (3) **Modele replikacji.** Dwa główne modele replikacji stosowane w systemach NOSQL to: nadrzędna-podrzędna i nadrzędna-nadrzędna. W **replikacji nadrzędna-podrzędna** jedna kopia jest nadrzędna. Wszystkie operacje zapisu muszą być wprowadzane w kopii nadrzędnej, a następnie przesyłane do kopii podrzędnych — zwykle w modelu spójności ostatecznej (kopie podrzędne *ostatecznie* będą identyczne z kopią nadrzędną). Zależnie od potrzeb odczytu replikację nadrzędna-podrzędna można skonfigurować na kilka sposobów. Jedna z konfiguracji wymaga, by wszystkie odczyty odbywały się z użyciem kopii nadrzędnej. Przypomina to metody sterowania



współbieżnego w środowisku rozproszonym z użyciem węzła podstawowego lub kopii podstawowej (patrz punkt 23.3.1). Podobne są też wady i zalety obu tych rozwiązań. Inna konfiguracja dopuszcza odczyty z użyciem kopii podrzędnych, jednak bez gwarancji tego, że pobierane wartości są zgodne z najnowszymi zapisami. Jest tak, ponieważ zapisy w węzłach podrzędnych mogą mieć miejsce po ich wprowadzeniu w kopii nadrzędnej. **Replikacja nadrzędna-nadrzędna** umożliwia odczyty i zapisy z użyciem dowolnych replik, jednak nie gwarantuje, że odczyty w węzłach przechowujących różne kopie dadzą te same wyniki. Różni użytkownicy mogą jednocześnie zapisać ten sam element danych w różnych węzłach systemu, dlatego wartość tego elementu może być tymczasowo niespójna. W ramach replikacji nadrzędna-nadrzędna trzeba zaimplementować metody eliminowania sprzecznych operacji zapisu tego samego elementu danych w różnych węzłach.

- (4) **Sharding plików.** W wielu aplikacjach NOSQL pliki (lub zbiory obiektów z danymi) mogą obejmować miliony rekordów (lub dokumentów albo obiektów). Dostęp do tych rekordów mogą uzyskiwać tysiące użytkowników jednocześnie. Dlatego niepraktyczne jest przechowywanie całego pliku w jednym węźle. W systemach NOSQL często stosowany jest **sharding** (inna nazwa to **fragmentacja pozioma**; patrz podrozdział 23.2) rekordów pliku. Dzięki temu można rozdzielić obciążenie związane z dostępem do rekordów pliku między wiele węzłów. Połączenie shardingu rekordów pliku z replikacją fragmentów pozwala poprawić równoważenie obciążenia i zwiększyć dostępność danych. Wybrane techniki shardingu opiszemy w podrozdziałach od 24.3 do 24.6 w ramach omawiania konkretnych systemów.
- (5) **Wysoce wydajny dostęp do danych.** W wielu aplikacjach NOSQL konieczne jest wyszukiwanie poszczególnych rekordów lub obiektów (elementów danych) wśród milionów rekordów danych lub obiektów z pliku. W tym celu w większości systemów stosowana jest jedna z dwóch technik: mieszanie lub podział na zakresy według kluczy obiektów. Większośćostępów do obiektów odbywa się na podstawie wartości klucza, a nie według złożonych warunków z zapytań. Klucz obiektu przypomina identyfikator obiektu (patrz podrozdział 12.1). W **mieszanu** funkcja mieszająca  $h(K)$  jest stosowana do klucza  $K$ , a lokalizacja obiektu z kluczem  $K$  jest ustalana na podstawie wartości  $h(K)$ . W **podziale na zakresy** lokalizacja jest określana według zakresów wartości kluczy. Przykładowo, lokalizacja  $i$  zawiera obiekty, których wartości  $K$  należą do przedziału  $K_{i_{\min}} \leq K \leq K_{i_{\max}}$ . W aplikacjach wymagających zapytań zakresowych, gdzie pobieranych jest wiele obiektów o wartościach klucza z danego zakresu, preferowany jest podział na zakresy. Można też posłużyć się innymi indeksami do lokalizowania obiektów na podstawie warunków dotyczących atrybutów innych niż klucz  $K$ . Wybrane techniki mieszania, podziału i indeksowania omówimy w podrozdziałach od 24.3 do 24.6 w kontekście opisu konkretnych systemów.

**Cechy systemów NOSQL związane z modelami danych i językami zapytań.** W systemach NOSQL nacisk położony jest na wydajność i elastyczność, a nie na możliwości w obszarze modelowania i obsługę złożonych zapytań. Tu omówimy niektóre z tych cech.

- (1) **Schemat nie jest wymagany.** W wielu systemach NOSQL elastyczność wynikająca z tego, że schemat nie jest niezbędny, uzyskuje się dzięki obsłudze częściowo ustrukturyzowanych i samoopisowych danych (patrz podrozdział 13.1). W niektórych

systemach użytkownicy mogą utworzyć częściowy schemat, aby zwiększyć wydajność składowania danych, jednak w większości systemów NOSQL *schemat nie jest wymagany*. Ponieważ schemat określający ograniczenia może być nieobecny, wszelkie ograniczenia danych trzeba zaprogramować w aplikacjach korzystających z elementów danych. Istnieją różne języki do opisu częściowo ustrukturyzowanych danych. Niektóre z nich to JSON (ang. *JavaScript Object Notation*) i XML (ang. *Extensible Markup Language*; patrz rozdział 13.). JSON jest używany w różnych systemach NOSQL, ale można też stosować inne metody opisu częściowo ustrukturyzowanych danych. Język JSON opisujemy w podrozdziale 24.3, w trakcie prezentowania dokumentowych systemów NOSQL.

- (2) **Mniej rozbudowane języki zapytań.** Wiele aplikacji używających systemów NOSQL nie wymaga rozbudowanego języka zapytań takiego jak SQL, ponieważ w zapytaniach z wyszukiwaniem (w odczytach) w takich systemach często można zlokalizować pojedyncze obiekty w pojedynczych plikach na podstawie kluczy. Systemy NOSQL zwykle udostępniają zestaw funkcji i operacji jako interfejs API, dlatego odczyt i zapis obiektów z danymi odbywa się za pomocą wywołań odpowiednich operacji przez programistę. W wielu sytuacjach są to **operacje CRUD** (od ang. *create, read, update i delete*, czyli utwórz, wczytaj, aktualizuj i usuń). Niektóre systemy NOSQL udostępniają wysokopoziomowy język zapytań, który jednak nie oferuje pełnych możliwości języka SQL. Przede wszystkim wiele systemów NOSQL nie zapewnia operacji złączania w ramach samego języka zapytań. Złączenia trzeba implementować w aplikacjach.
- (3) **Wersjonowanie.** Niektóre systemy NOSQL umożliwiają przechowywanie wielu wersji elementów danych ze znacznikami czasu utworzenia poszczególnych wersji tych elementów. Ten aspekt omówimy w podrozdziale 24.5, w kontekście prezentowania kolumnowych systemów NOSQL.

W następnym punkcie znajdziesz przegląd różnych kategorii systemów NOSQL.

### 24.1.3. Kategorie systemów NOSQL

Systemy NOSQL zostały podzielone na cztery podstawowe kategorie (dodatkowe klasy łączą cechy z różnych kategorii). Najczęstszy podział obejmuje cztery następujące główne kategorie:

- (1) **Dokumentowe systemy NOSQL.** Systemy tego rodzaju przechowują dane w postaci dokumentów za pomocą znanych formatów takich jak JSON. Dokumenty są dostępne na podstawie identyfikatorów, ale czasem można też uzyskać szybki dostęp do nich za pomocą innych indeksów.
- (2) **Systemy NOSQL z parami klucz-wartość.** Takie systemy mają prosty model danych oparty na szybkim dostępie na podstawie klucza do powiązanej z nim wartości. Wartością może być rekord, obiekt, dokument, a nawet bardziej złożona struktura danych.
- (3) **Kolumnowe systemy NOSQL.** Takie systemy dzielą tabele na rodziny kolumn (jest to forma podziału pionowego; patrz podrozdział 23.2), gdzie każda rodzina jest przechowywana w odrębnych plikach. Możliwe jest tu też wersjonowanie wartości danych.

- (4) **Grafowe systemy NOSQL.** Dane są tu reprezentowane jako grafy, a powiązane ze sobą węzły można znaleźć, poruszając się krawędziami za pomocą wyrażeń ścieżkowych.

Można też dodać inne kategorie, aby uwzględnić systemy, które nie wpasowują się dobrze w cztery wyżej wymienione, a także inne systemy, dostępne jeszcze przed wprowadzeniem nazwy NOSQL.

- (5) **Hybrydowe systemy NOSQL.** Takie systemy mają cechy przynajmniej dwóch z czterech podanych wcześniej kategorii.
- (6) **Obiektowe bazy danych.** Te systemy zostały opisane w rozdziale 12.
- (7) **Bazy danych wykorzystujące XML.** Język XML omówiono w rozdziale 13.

Nawet wyszukiwarki oparte na słowach kluczowych przechowują duże ilości danych i wymagają szybkiego dostępu w trakcie wyszukiwania, dlatego składowane dane można traktować jak duże magazyny NOSQL zawierające big data.

Oto struktura zawartości dalszej części rozdziału: w podrozdziałach od 24.3 do 24.6 omówimy po jednej z czterech głównych kategorii systemów NOSQL i szczegółowo objaśnimy cechy każdej z nich. Wcześniej, w podrozdziale 24.2, szczegółowo przedstawimy spójność ostateczną oraz powiązane z tym twierdzenie CAP.

## 24.2. Twierdzenie CAP

W trakcie omawiania sterowania współbieżnego w rozproszonych bazach danych (w podrozdziale 23.3) przyjmowaliśmy, że rozproszony SZBD musiał wymuszać właściwości ACID (niepodzielność, spójność, izolację i trwałość) wykonywanych współbieżnie transakcji (patrz podrozdział 20.3). W systemie z replikacją danych sterowanie współbieżne jest bardziej skomplikowane, ponieważ może istnieć wiele kopii każdego elementu danych. Dlatego jeśli zaktualizowana zostanie jedna kopia elementu, trzeba też w spójny sposób zmodyfikować wszystkie pozostałe kopie. Możliwe jest, że jedna kopia elementu  $X$  zostanie zaktualizowana przez transakcję  $T_1$ , natomiast inna kopia — przez transakcję  $T_2$ . Dlatego w dwóch różnych węzłach systemu rozproszonego będą się znajdować dwie niespójne kopie tego samego elementu. Jeżeli dwie inne transakcje,  $T_3$  i  $T_4$ , zechcą wczytać  $X$ , każda z nich może odczytać inną kopię tego elementu.

W podrozdziale 23.3 pokazano, że istnieją metody sterowania współbieżnego w środowisku rozproszonym, które nie dopuszczają niespójności między kopiami tego samego elementu danych. Wymuszają w ten sposób szeregowalność, a tym samym izolację, gdy stosowana jest replikacja. Jednak techniki te generują wysokie koszty, co jest sprzeczne z tworzeniem wielu kopii w celu poprawy wydajności i dostępności w rozproszonych systemach baz danych takich jak NOSQL. W obszarze systemów rozproszonych występują różne poziomy spójności między replikowanymi elementami danych — od słabej do mocnej. Wymuszanie szeregowalności to najsilniejsza postać spójności, powodująca jednak wysokie koszty, co może obniżać wydajność operacji odczytu i zapisu oraz negatywnie wpływać na wydajność systemu.

Twierdzenie CAP, pierwotnie zaproponowane jako zasada CAP, pozwala wyjaśnić pewne sprzeczne wymogi występujące w systemie rozproszonym z replikacją. Trzy litery z akronimu CAP oznaczają pożądane cechy systemów rozproszonych z replikowanymi danymi: **spójność** (między replikowanymi kopiami; ang. *consistency*), **dostępność** (systemu na potrzeby operacji odczytu i zapisu; ang. *availability*) i **odporność na podział** (w sytuacji, gdy węzły w systemie zostają podzielone z powodu awarii sieci; ang. *partition tolerance*). *Dostępność* oznacza, że każde żądanie odczytu lub zapisu elementu danych albo zostanie z powodzeniem przetworzone, albo pojawi się komunikat o niemożności ukończenia operacji. *Odporność na podział* oznacza, że system może kontynuować pracę, jeśli w sieci łączącej węzły wystąpi problem powodujący jej podział na przynajmniej dwie części. Węzły z każdej z tych części mogą się wtedy komunikować tylko między sobą. *Spójność* oznacza, że węzły będą udostępniać te same kopie replikowanych elementów danych dla różnych transakcji.

Warto zauważyć, że słowo *spójność* w twierdzeniu CAP i właściwościach ACID *nie oznacza tego samego*. W twierdzeniu CAP określenie *spójność* dotyczy spójności wartości z różnych kopii tego samego elementu danych w systemie rozproszonym z replikacją. We właściwościach ACID zachowanie spójności polega na tym, że transakcja nie będzie naruszała ograniczeń integralności określonych w schemacie bazy danych. Jeśli jednak uznamy, że spójność replikowanych kopii jest *określonym ograniczeniem*, to oba zastosowania nazwy *spójność* będą ze sobą powiązane.

Zgodnie z **twierdzeniem CAP** w rozproszonym systemie z replikacją danych *nie da się zapewnić* wszystkich trzech pożądanych cech (spójności, dostępności i odporności na podział) jednocześnie. Projektant systemu rozproszonego musi wybrać dwie z trzech gwarantowanych właściwości. W wielu tradycyjnych aplikacjach (używających języka SQL) zwykle zakłada się, że ważne jest gwarantowanie spójności za pomocą właściwości ACID. W rozproszonych magazynach danych NOSQL często akceptowalny jest niższy poziom spójności, a istotne jest dostarczenie dwóch pozostałych cech (dostępności i odporności na podział). Dlatego w systemach NOSQL zamiast gwarantowania szeregowości często stosowane są niższe poziomy spójności. Najczęściej używana w takich systemach jest **spójność ostateczna**. W podrozdziałach od 24.3 do 24.6 opiszemy niektóre modele spójności stosowane w konkretnych systemach NOSQL.

Cztery następne podrozdziały zawierają omówienie cech czterech głównych kategorii systemów NOSQL. W podrozdziale 24.3 na podstawie systemu MongoDB opiszemy dokumentowe systemy NOSQL. W podrozdziale 24.4 omówimy systemy NOSQL nazywane magazynami z parami klucz-wartość. W podrozdziale 24.5 na przykładzie systemu HBase przedstawimy kolumnowe systemy NOSQL. W podrozdziale 24.6 znajdziesz wprowadzenie do grafowych systemów NOSQL.

## 24.3. Dokumentowe systemy NOSQL i baza MongoDB

Dokumentowe systemy NOSQL zwykle przechowują dane jako **kolekcje** podobnych **dokumentów**. Systemy tego rodzaju można też nazywać **magazynami dokumentów**. Poszczególne dokumenty przypominają nieco *obiekty złożone* (patrz podrozdział 12.3) lub dokumenty XML (patrz rozdział 13.). Główna różnica między systemami dokumentowymi

a systemami obiektowymi, obiektowo-relacyjnymi i bazami XML polega na tym, że w tych pierwszych nie trzeba tworzyć schematu. Zamiast tego dokumenty są tworzone jako **samoopisowe dane** (patrz podrozdział 13.1). Choć dokumenty w kolekcji mogą być *podobne* do siebie, mogą obejmować różne elementy danych (atrybuty). Ponadto nowe dokumenty mogą zawierać nowe elementy danych, nieistniejące w obecnych dokumentach z kolekcji. System pobiera nazwy elementów danych z samoopisowych dokumentów z kolekcji, a użytkownik może zażądać, by system utworzył indeksy na podstawie określonych elementów danych. Dokumenty można tworzyć w różnych formatach, np. w XML-u (patrz rozdział 13.). Popularnym językiem tworzenia dokumentów w systemach NOSQL jest **JSON**.

Istnieje wiele dokumentowych systemów NOSQL, w tym MongoDB i CouchDB. W tym podrozdziale przedstawimy system MongoDB. Należy zauważyć, że w różnych systemach można stosować różne modele, języki i metody implementacji. Jednak przedstawianie kompletnego przeglądu wszystkich dokumentowych systemów NOSQL wykracza poza zakres tej książki.

### 24.3.1. Model danych z systemu MongoDB

Dokumenty w systemie MongoDB są przechowywane w formacie BSON (ang. *Binary JSON*). Jest to odmiana formatu JSON z dodatkowymi typami danych i zajmująca mniej miejsca niż JSON. Poszczególne **dokumenty** są przechowywane w **kolekcji**. Posłużymy się tu prostym przykładem opartym na używanej w książce bazie FIRMA. Do tworzenia każdej kolekcji używana jest operacja `createCollection`. Przykładowo, pokazana dalej instrukcja pozwala utworzyć kolekcję o nazwie `projekt` przechowującą obiekty `PROJEKT` z bazy FIRMA (patrz rysunki 5.5 i 5.6).

```
db.createCollection("projekt", { capped : true, size : 1310720, max : 500 } )
```

Pierwszy parametr, "projekt", to **nazwa** kolekcji, po której następuje opcjonalny dokument określający **opcje kolekcji**. W przykładzie używana jest opcja `capped`. Oznacza to, że stosowane jest górne ograniczenie zajmowanej przestrzeni (`size`) i liczby dokumentów (`max`). Parametry dotyczące ograniczeń pomagają systemowi dobrać dla każdej kolekcji opcje składowania danych. Dostępne są też inne opcje kolekcji, ale nie będziemy ich tu omawiać.

W przykładzie utworzymy też inną kolekcję dokumentów, `pracownik`, przechowującą informacje o osobach pracujących nad każdym projektem:

```
db.createCollection("pracownik", { capped : true, size : 5242880, max : 2000 } ) )
```

Każdy dokument w kolekcji ma pole `_id` typu `ObjectId`. Zawiera ono unikatową wartość i jest automatycznie indeksowane w kolekcji, chyba że użytkownik bezpośrednio zażąda, by nie tworzono indeksu na polu `_id`. Wartość typu `ObjectId` może zostać *podana przez użytkownika* lub *wygenerowana przez system*, jeśli użytkownik nie poda pola `_id` dla konkretnego dokumentu. Wartości typu `ObjectId` *generowane przez system* mają specyficzny format i łączą znacznik czasu określający moment utworzenia obiektu (cztery bajty w wewnętrznym formacie systemu MongoDB), identyfikator węzła (trzy bajty), identyfikator procesu (dwa bajty) i licznik (trzy bajty), co daje 16-bajtową wartość. Wartości typu `ObjectId` *generowane przez użytkownika* mogą mieć dowolną wartość, przy czym muszą w unikatowy sposób identyfikować dokument, dlatego zwykle używa się tu kluczy głównych z systemów relacyjnych.

Kolekcja nie ma schematu. Struktura pól danych z dokumentów jest tworzona na podstawie sposobu dostępu do dokumentów i ich użytkowania. Użytkownik może wybrać projekt znormalizowany (podobny do znormalizowanych krotek z systemów relacyjnych) lub zdenormalizowany (podobny jak w dokumentach XML lub obiektach złożonych). Odwołania do innych dokumentów można tworzyć, zapisując w jednym dokumencie identyfikatory `ObjectId` innych, powiązanych dokumentów. Na rysunku 24.1(a) pokazano uproszczony dokument z systemu MongoDB z wybranymi danymi z rysunku 5.6 z używanej w tej książce przykładowej bazy FIRMA. W przykładzie wartości `_id` są definiowane przez użytkownika. Dokumenty z polem `_id` rozpoczynającym się od liter PRO (od słowa „projekt”) są zapisywane w kolekcji „projekt”, natomiast dokumenty z polem `_id` z literami PRA (od słowa „pracownik”) są umieszczane w kolekcji „pracownik”.

(a) Dokument dotyczący projektów, obejmujący zagnieżdżoną tablicę pracowników

```
{
  _ID: "PR01",
  NAZWAPROJ: "ProduktX",
  LOKALIZACJAPROJ: "Gliwice",
  PRACOWNICY: [
    { Ename: "Jan Szewczyk",
      Hours: 32.5
    },
    { Ename: "Joanna Englert",
      Hours: 20.0
    }
  ]
};
```

(b) Dokument dotyczący projektów, obejmujący zagnieżdżoną tablicę identyfikatorów pracowników

```
{
  _ID: "PR01",
  NAZWAPROJ: "ProduktX",
  LOKALIZACJAPROJ: "Gliwice",
  IDPRACOWNIKÓW: [ "W1", "W2" ]
}
{ _ID: "PRA1",
  IMIĘNAZWISKO: "Jan Szewczyk",
  GODZINY: 32.5
}
{ _ID: "PRA2",
  IMIĘNAZWISKO: "Joanna Englert",
  GODZINY: 20.0
}
```

(c) Znormalizowane dokumenty dotyczące projektu i pracowników (nie jest to jednak w pełni znormalizowany projekt relacji M:N)

```
{
  _ID: "PR01",
  NAZWAPROJ: "ProduktX",
  LOKALIZACJAPROJ: "Gliwice"
```



```

    }
    {
      _ID: "PRA1",
      IMIĘNAZWISKO: "Jan Szewczyk",
      IDPROJEKTU: "PRO1",
      GODZINY: 32.5
    }
    {
      _ID: "PRA2",
      IMIĘNAZWISKO: "Joanna Englert",
      IDPROJEKTU: "PRO1",
      GODZINY: 20.0
    }
  }

```

(d) Wstawianie dokumentów z punktu (c) do kolekcji PROJEKT i PRACOWNIK

```

db.projekt.insert({_ID: "PRO1", NAZWAPROJ: "ProduktX", LOKALIZACJAPROJ: "Gliwice"})
db.pracownik.insert([{_ID: "PRA1", IMIĘNAZWISKO: "Jan Szewczyk", IDPROJEKTU: "PRO1",
  GODZINY: 32.5},
  {_ID: "PRA2", IMIĘNAZWISKO: "Joanna Englert", IDPROJEKTU: "PRO1",
  GODZINY: 20.0}])

```

RYSUNEK 24.1. Przykład prostych dokumentów w MongoDB. (a) Zdenormalizowany projekt dokumentu z zagnieżdżonymi poddokumentami. (b) Zagnieżdżona tablica referencji do dokumentów. (c) Dokumenty znormalizowane. (d) Wstawianie dokumentów z rysunku 24.1(c) do odpowiednich kolekcji

Na rysunku 24.1(a) informacje o pracownikach są *osadzone w dokumencie projekt*. Nie trzeba więc tworzyć kolekcji „pracownik”. Jest to tzw. *wzorzec zdenormalizowany*, podobny do tworzenia złożonych obiektów (patrz rozdział 12.) lub dokumentów XML (patrz rozdział 13.). Lista wartości w *nawiasach kwadratowych* [] reprezentuje w dokumencie pole, którego wartością jest **tablica**.

Inna możliwość to zastosowanie projektu z rysunku 24.1(b), gdzie w dokumencie projektu osadzone są *referencje do pracowników*, a dokumenty pracowników są przechowywane w odrębnej kolekcji „pracownik”. Trzecie rozwiązanie, z rysunku 24.1(c), to projekt znormalizowany przypominający relacje w pierwszej postaci normalnej (patrz punkt 14.3.4). Wybór opcji projektowej zależy od sposobu dostępu do danych.

Należy zauważyć, że prosty projekt z rysunku 24.1(c) *nie jest ogólnym projektem znormalizowanym* dla związków wiele do wielu (np. między pracownikami i projektami). Potrzebne byłyby tu trzy kolekcje: „projekt”, „pracownik” i „pracuje\_nad”, co opisano szczegółowo w podrozdziale 9.1. Wiele kompromisów projektowych omówionych w rozdziałach 9. i 14. (relacje w pierwszej postaci normalnej i sposoby odwzorowywania z modelu ER na model relacyjny) oraz 12. i 13. (obiekty złożone i format XML) dotyczy też tworzenia odpowiedniego projektu struktur i kolekcji dokumentów, dlatego nie będziemy ponownie opisywać tych zagadnień. W projekcie z rysunku 24.1(c) pracownik zajmujący się kilkoma projektami jest reprezentowany za pomocą *wielu dokumentów pracownika* z różnymi wartościami `_id`. Każdy dokument reprezentuje wtedy daną osobę *jako pracownika zajmującego się danym projektem*. Przypomina to rozwiązania z projektu schematu XML (patrz podrozdział 13.6). Jednak ponownie należy zauważyć, że w typowym systemie dokumentowym *nie występują schematy*. Dlatego reguły projektowe należy stosować na etapie dodawania poszczególnych dokumentów do kolekcji.



## 24.3.2. Operacje CRUD w systemie MongoDB

W systemie MongoDB dostępne są różne **operacje CRUD**. Dokumenty można *tworzyć* i wstawiać do kolekcji za pomocą operacji `insert` o składni:

```
db.<nazwa_kolekcji>.insert(<dokumenty>)
```

Parametry operacji `insert` mogą obejmować jeden dokument lub tablicę dokumentów (patrz rysunek 24.1(d)). Operacja *usuwania* nosi nazwę `remove`. Oto jej składnia:

```
db.<nazwa_kolekcji>.remove(<warunek>)
```

Dokumenty usuwane z kolekcji są podawane za pomocą warunku logicznego dotyczącego pól dokumentów z kolekcji. Dostępna jest też operacja `update` z warunkiem wybierania pewnych dokumentów i klauzulą `$set` określającą sposób aktualizacji. Można też zastosować operację aktualizacji, aby zastąpić istniejący dokument innym, ale zachować ten sam identyfikator `ObjectId`.

W zapytaniach z *odczytem* podstawowe polecenie to `find`. Oto jego składnia:

```
db.<nazwa_kolekcji>.find(<warunek>)
```

Jako `<warunek>` można podać ogólne warunki logiczne, a w wyniku zapytania zwrócone zostaną dokumenty z kolekcji o wartości `true` dla określonych warunków. Pełne omówienie operacji CRUD z systemu MongoDB znajdziesz w internetowej dokumentacji tego systemu podanej w literaturze na końcu rozdziału.

## 24.3.3. Cechy systemu rozproszonego MongoDB

Większość aktualizacji w systemie MongoDB jest niepodzielna (jeśli dotyczą pojedynczego dokumentu). Jednak MongoDB umożliwia też tworzenie transakcji dotyczących wielu dokumentów. Ponieważ MongoDB to system rozproszony, stosuje się w nim **zatwierdzenie dwufazowe** do zapewnienia niepodzielności i spójności w transakcjach z użyciem wielu dokumentów. Niepodzielność i spójność transakcji omówiono w podrozdziale 20.3, a protokół zatwierdzania dwufazowego w podrozdziale 22.6.

**Replikacja w systemie MongoDB.** W MongoDB do tworzenia wielu kopii tego samego zbioru danych w różnych węzłach systemu rozproszonego stosowany jest **zbiór replik**. Replikacja odbywa się zgodnie z odmianą modelu **nadrzędna-podrzędna**. Załóżmy np., że chcemy zreplikować kolekcję dokumentów C. Zbiór replik będzie obejmował jedną **kopię nadrzędną** kolekcji C w węźle N1 i przynajmniej jedną **kopię podrzędną** (replikę) kolekcji C w innym węźle N2. W razie potrzeby można zapisać dodatkowe kopie w węzłach N3, N4 itd., jednak koszt składowania i aktualizowania (zapisu) danych rośnie wraz z liczbą replik. Zbiór replik musi obejmować przynajmniej trzy elementy, dlatego jeśli potrzebna jest tylko jedna kopia podrzędna, w trzecim węźle, N3, trzeba uruchomić element zbioru replik nazywany **arbiterem**. Arbiter nie obejmuje repliki kolekcji, ale uczestniczy w **wyborach** nowej kopii nadrzędnej, jeśli węzeł przechowujący obecną kopię nadrzędną zawiedzie. Jeżeli liczba elementów w zbiorze replik wynosi  $n$  (jedna kopia nadrzędna plus  $i$  kopii podrzędnych, co daje  $n = i + 1$ ), to  $n$  musi być liczbą nieparzystą. W przeciwnym razie dodawany jest *arbiter*, aby zagwarantować poprawny przebieg procesu wyborów po awarii kopii nadrzędnej. Wybory w systemach rozproszonych omówiono w punkcie 23.3.1.

W replikacji w systemie MongoDB wszystkie operacje zapisu trzeba stosować do kopii nadrzędnej, a następnie przekazywać do kopii podrzędnych. W operacjach odczytu użytkownik może określić w aplikacji **preferencje odczytu**. Domyślne preferencje odczytu powodują, że wszystkie odczyty odbywają się z użyciem kopii nadrzędnej. Dlatego wszystkie operacje odczytu i zapisu mają miejsce w węźle nadrzędnym. W takiej sytuacji kopie podrzędne służą głównie do gwarantowania, że system będzie działał po awarii węzła nadrzędnego i że MongoDB potrafi dla każdego żądania odczytu zwrócić najnowszą wartość dokumentu. Aby zwiększyć wydajność odczytów, można ustawić preferencje tak, by *żądania odczytu mogły być przetwarzane w dowolnej replice* (nadrzędnej lub podrzędnej). Jednak odczyt z repliki podrzędnej nie gwarantuje otrzymania najnowszej wersji dokumentu, ponieważ przenoszenie zapisów z repliki nadrzędnej do replik podrzędnych może się odbywać z opóźnieniem.

**Sharding w systemie MongoDB.** Gdy kolekcja przechowuje bardzo dużą liczbę dokumentów lub wymaga dużej ilości miejsca, składowanie wszystkich dokumentów w jednym węźle może skutkować problemami z wydajnością — zwłaszcza w sytuacji, gdy wiele operacji użytkowników jednocześnie uzyskuje dostęp do dokumentów za pomocą wielu operacji CRUD. **Sharding** dokumentów z kolekcji (nazywany też *podziałem poziomym*) polega na podziale dokumentów na odrębne partycje nazywane **shardami**. Dzięki temu system może dodać w razie potrzeby nowe węzły (jest to proces **skalowania poziomego** systemu rozproszonego; patrz punkt 23.1.4) i przechowywać shardy kolekcji w różnych węzłach w celu równoważenia obciążenia. Każdy węzeł przetwarza wtedy tylko operacje dotyczące dokumentów z shardu składowanego w tym węźle. Ponadto każdy shard obejmuje mniej dokumentów niż w sytuacji, gdy cała kolekcja jest przechowywana w jednym węźle. Skutkuje to dodatkowym wzrostem wydajności.

W systemie MongoDB istnieją dwa sposoby podziału kolekcji na shardy — **podział na zakresy** i **podział z użyciem mieszania**. Oba podejścia wymagają, aby użytkownik wskazał pole dokumentu, na podstawie którego odbywać się będzie podział dokumentów na shardy. *Pole podziału* (nazywane w systemie MongoDB **kluczem shardów** — ang. *shard key*) musi mieć dwie cechy: musi występować w *każdym dokumencie* kolekcji i wymagany jest *indeks* na nim. Można wykorzystać pole typu `ObjectId`, jednak na potrzeby shardingu można zastosować także dowolne inne pole o dwóch wymienionych cechach. Wartości pola podziału są dzielone na **grupy** (albo według zakresów, albo za pomocą mieszania), a dokumenty są dzielone na podstawie grup wartości tego pola.

*Podział na zakresy* polega na tworzeniu grup przez podanie zakresów wartości klucza shardów. Przykładowo, jeśli klucz przyjmuje wartości od 1 do 10 000 000, można utworzyć dziesięć zakresów: od 1 do 1 000 000, od 1 000 001 do 2 000 000, ..., od 9 000 001 do 10 000 000. Każda grupa zawiera wtedy wartości klucza z jednego zakresu. *Podział z użyciem mieszania* polega na zastosowaniu funkcji mieszającej  $h(K)$  do każdego klucza shardów  $K$ , a podział kluczy na grupy odbywa się według wartości tej funkcji (mieszanie oraz jego wady i zalety opisano w podrozdziale 16.8). Jeśli często wykonywane są **zapytania zakresowe** dotyczące kolekcji (np. w celu pobrania wszystkich dokumentów o kluczu shardów z wartością z przedziału od 200 do 400), preferowany jest podział na zakresy, ponieważ każde zapytanie zakresowe zwykle będzie kierowane do jednego węzła zawierającego wszystkie potrzebne dokumenty z jednego shardu. Jeżeli większość operacji wyszukiwania pobiera pojedyncze dokumenty, preferowany może być podział z użyciem mieszania, ponieważ powoduje on losowy rozkład wartości kluczy shardów między grupy.

Gdy stosowany jest sharding, zapytania w systemie MongoDB są kierowane do modułu **routera zapytań**. Ten moduł śledzi, które węzły zawierają konkretne shardy. Robi to na podstawie metody podziału zastosowanej do kluczy shardów. Zapytanie (operacja CRUD) jest kierowane do węzłów zawierających shardy z dokumentami żadanymi w tym zapytaniu. Jeśli system nie potrafi określić, które shardy przechowują potrzebne dokumenty, zapytanie trafi do wszystkich węzłów przechowujących shardy z danej kolekcji. Sharding i replikacja są stosowane razem. Sharding ma przede wszystkim poprawiać wydajność za pomocą równoważenia obciążenia i skalowalności poziomej. Replikacja ma gwarantować dostępność systemu po awarii niektórych węzłów systemu rozproszonego.

Z architekturą systemu rozproszonego i komponentami MongoDB związanych jest też wiele innych szczegółów, jednak opisywanie ich wszystkich wykracza poza zakres tego omówienia. MongoDB udostępnia także wiele innych usług w obszarach takich jak zarządzanie systemem, indeksowanie, zabezpieczenia i agregowanie danych. Nie będziemy jednak omawiać ich w tym miejscu. Kompletną dokumentację systemu MongoDB znajdziesz w internecie (patrz wybrane publikacje).

## 24.4. Magazyny NOSQL z parami klucz-wartość

**Magazyny z parami klucz-wartość** mają zapewniać wysoką wydajność, dostępność i skalowalność dzięki przechowywaniu danych w rozproszonym systemie składowania. Model danych używanych w magazynach z parami klucz-wartość jest stosunkowo prosty. W wielu takich systemach nie występuje język zapytań. Dostępny jest tylko zbiór operacji, z których mogą korzystać programiści aplikacji. **Klucz** to unikatowy identyfikator powiązany z elementem danych i używany do szybkiego zlokalizowania określonego elementu danych. **Wartością** jest sam element danych. W różnych systemach z parami klucz-wartość może on mieć zupełnie inne formaty. W niektórych scenariuszach wartością jest *ciąg bajtów* lub *tablica bajtów*, a aplikacja używająca magazynu z parami klucz-wartość musi zinterpretować strukturę tej wartości. W innych sytuacjach dozwolone są dane o standardowym formacie — np. ustrukturyzowane wiersze danych (krotki) podobne jak w danych relacyjnych albo częściowo ustrukturyzowane dane w formacie JSON lub innym samoopisowym formacie danych. Różne magazyny z parami klucz-wartość mogą więc przechowywać nieustrukturyzowane, częściowo ustrukturyzowane lub ustrukturyzowane elementy danych (patrz podrozdział 13.1). Podstawową cechą takich magazynów jest to, że każda wartość (każdy element danych) musi być powiązana z unikatowym kluczem, a pobieranie danej wartości na podstawie podanego klucza musi się odbywać bardzo szybko.

Istnieje wiele systemów zaliczanych do magazynów z parami klucz-wartość. Dlatego zamiast opisywać szczegółowo jeden konkretny system, przedstawimy krótki wprowadzający przegląd wybranych systemów i ich cech.

### 24.4.1. Przegląd systemu DynamoDB

DynamoDB to produkt Amazona dostępny w ramach platform **AWS/SDK** (ang. *Amazon Web Services* i *Software Development Kit*) tej firmy. Można z niego korzystać jako z komponentu składowania danych w ramach usług chmury obliczeniowej Amazona.

**Model danych z DynamoDB.** Podstawowy model danych w systemie DynamoDB jest oparty na tabelach, elementach i atrybutach. **Tabela** w DynamoDB *nie ma schematu*. Przechowuje ona kolekcję *samoopisowych elementów*. Każdy **element** składa się z zestawu par (atrybut, wartość), a atrybuty mogą być jedno- lub wielowartościowe. Tak więc tabela przechowuje kolekcję elementów, a każdy element to samoopisowy rekord (lub obiekt). System DynamoDB umożliwia też użytkownikom podawanie elementów w formacie JSON, a system przekształca je na wewnętrzny format składowania danych systemu DynamoDB.

W trakcie tworzenia tabeli trzeba określić jej **nazwę** i **klucz główny**. Klucz główny jest używany do szybkiego znajdowania elementów w tabeli. Tak więc w systemie DynamoDB z parami klucz-wartość klucz główny jest **kluczem**, a element **wartością**. Atrybut klucza głównego musi występować w każdym elemencie tabeli. Oto dwa możliwe rodzaje kluczy głównych:

- **Pojedynczy atrybut.** System DynamoDB używa takiego atrybutu do zbudowania indeksu z mieszaniem na elementach tabeli. Jest to *klucz główny oparty na mieszaniu*. Elementy w magazynie nie są porządkowane według wartości atrybutu mieszania.
- **Para atrybutów.** Jest to *klucz główny oparty na haszowaniu i zakresach*. Kluczem głównym jest wtedy para atrybutów (A, B), gdzie atrybut A jest używany do mieszania, a ponieważ istnieje wiele elementów o tej samej wartości A, wartości B są stosowane do uporządkowania rekordów o tej samej wartości A. Tabela z kluczami tego rodzaju może mieć dodatkowe indeksy drugorzędne na jej atrybutach. Przykładowo, jeśli chcesz przechowywać w tabeli wiele wersji elementów tego samego typu, możesz zastosować w kluczu głównym opartym na haszowaniu i zakresach pole `ID`em do haszowania i pole `Data` lub `Znacznik_czasu` (określające czas utworzenia danej wersji) jako zakres.

**Cechy rozproszonego systemu DynamoDB.** Ponieważ DynamoDB jest zastrzeżonym systemem, w następnym punkcie opiszemy mechanizmy używane do replikacji, shardingu i innych technik z systemów rozproszonych stosowane w otwartym systemie z parami klucz-wartość — w narzędziu Voldemort. System Voldemort jest oparty na wielu technikach zaproponowanych w systemie DynamoDB.

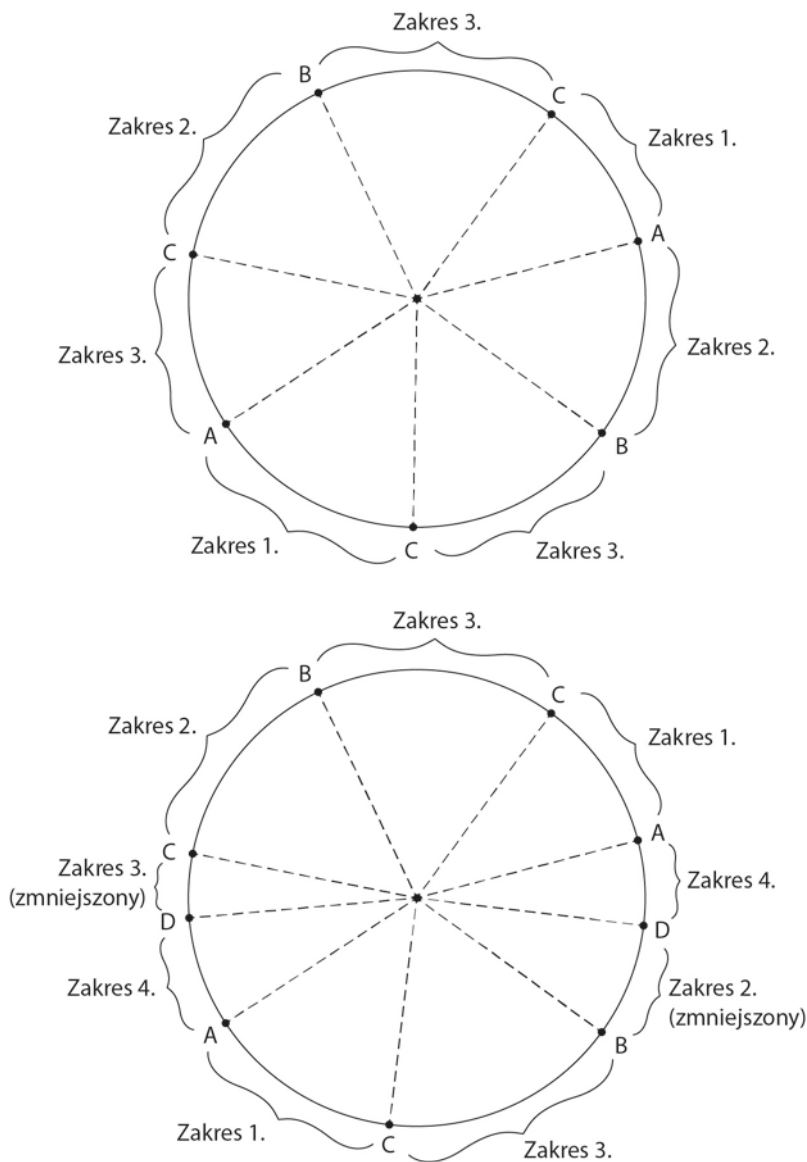
## 24.4.2. Rozproszony magazyn danych z parami klucz-wartość — Voldemort

Voldemort to otwarty system dostępny na licencji Apache 2.0. Jest on oparty na systemie DynamoDB Amazona. Nacisk położono w nim na wysoką wydajność i skalowalność poziomą, a także na zapewnienie replikacji pod kątem wysokiej dostępności i shardingu w celu skrócenia opóźnienia (czasu odpowiedzi) dla żądań odczytu i zapisu. Wszystkie trzy wymienione mechanizmy (replikacja, sharding i skalowalność pozioma) są realizowane w wyniku podziału par klucz-wartość między rozproszone węzły klastra. Rozkład danych odbywa się za pomocą **haszowania stabilnego**. System Voldemort był używany do składowania danych w serwisie LinkedIn. Oto wybrane funkcje tego systemu:

- **Proste podstawowe operacje.** Kolekcja par (klucz, wartość) jest przechowywana w **magazynie** systemu Voldemort. Tu przyjmijmy, że nazwa tego magazynu to `s`. Podstawowy interfejs do zapisywania i pobierania danych jest bardzo prosty

i obejmuje trzy operacje: `get`, `put` i `delete`. Operacja `s.put(k, v)` wstawia element jako parę klucz-wartość z kluczem `k` i wartością `v`. Operacja `s.delete(k)` usuwa z magazynu element o kluczu `k`, a operacja `v = s.get(k)` pobiera wartość `v` powiązaną z kluczem `k`. W aplikacji można na bazie tych podstawowych operacji określać wymagania. Na podstawowym poziomie klucze i wartości to tablice (ciągi) bajtów.

- **Wysokopoziomowe, sformatowane wartości danych.** Wartości `v` z elementów `(k, v)` mogą być podawane w formacie JSON, a system dokonuje przekształceń między formatem JSON a wewnętrznym formatem składowania danych. Można też stosować inne formaty obiektów z danymi, jeśli aplikacja obsługuje za pomocą klasy *Serializer* konwersję (**serializację**) między formatem użytkownika a formatem składowania. Klasa *Serializer* musi zostać podana przez użytkownika i obejmować operacje przekształcające dane z formatu użytkownika na ciąg bajtów składowany jako wartość i z ciągu bajtów pobieranego za pomocą operacji `s.get(k)` na format użytkownika. Voldemort udostępnia wbudowane klasy tego typu dla formatów innych niż JSON.
- **Mieszanie stabilne używane do podziału par (klucz, wartość).** W systemie Voldemort do podziału danych między rozproszone węzły klastra używana jest odmiana algorytmu rozdzielania danych nazywana **mieszaniami stabilnymi**. Funkcja mieszająca  $h(k)$  jest stosowana do klucza `k` każdej pary `(k, v)` i określa, gdzie dany element zostanie zapisany. W tej metodzie zakładamy, że wynik funkcji  $h(k)$  to liczba całkowita; zwykle pochodzi ona z przedziału od 0 do  $H_{max} = 2^{n-1}$ , gdzie  $n$  jest ustalane na podstawie pożądanego zakresu wartości funkcji mieszającej. Tę metodę najlepiej jest zwizualizować, przyjmując, że wszystkie możliwe całkowitoliczbowe wartości funkcji mieszającej od 0 do  $H_{max}$  są równomiernie rozłożone po okręgu. Węzły w systemie rozproszonym są też rozmieszczone na tym okręgu. Zwykle każdemu węzłowi odpowiada kilka lokalizacji (patrz rysunek 24.2). Rozmieszczenie na okręgu punktów reprezentujących węzły odbywa się w pseudolosowy sposób. Element `(k, v)` jest zapisywany w węźle, którego pozycja na okręgu znajduje się za pozycją wartości  $h(k)$  zgodnie z *ruchem wskazówek zegara*. Na rysunku 24.2(a) zakładamy, że w rozproszonym węźle występują trzy węzły: A, B i C, przy czym pojemność węzła C jest większa niż węzłów A i B. W typowym systemie węzłów jest znacznie więcej. Na okręgu rozmieszczone są po dwie instancje węzłów A i B oraz trzy instancje węzła C (ponieważ ma on większą pojemność). Są one rozmieszczone pseudolosowo i pokrywają cały okrąg. Na rysunku 24.2(a) pokazano, które elementy `(k, v)` są umieszczone na podstawie wartości  $h(k)$  w poszczególnych węzłach.
- Jeśli wartość  $h(k)$  odpowiada fragmentom okręgu oznaczonym na rysunku 24.2(a) jako zakres 1., element `(k, v)` jest zapisywany w węźle A, ponieważ to jego nazwa występuje po takiej wartości  $h(k)$ , patrząc zgodnie z ruchem wskazówek zegara. Elementy o wartościach  $h(k)$  z zakresu 2. trafiają do węzła B, a elementy o wartościach  $h(k)$  z zakresu 3. są umieszczane w węźle C. To rozwiązanie umożliwia *skalowalność poziomą*, ponieważ gdy do systemu rozproszonego dodawany jest nowy węzeł, można go (w zależności od jego pojemności) powiązać z jedną lub kilkoma lokalizacjami na okręgu. Tylko ograniczony procent elementów `(k, v)` zostanie przeniesiony do nowego węzła z istniejących węzłów. Odbędzie się to na podstawie algorytmu rozmieszczania z mieszaniami stabilnymi. Elementy przypisywane do nowego węzła



RYSUNEK 24.2. Przykład mieszania stabilnego. (a) Okrąg z trzema węzłami: A, B i C, przy czym C ma największą pojemność. Elementy  $(k, v)$  o wartościach  $h(k)$  odpowiadających punktom z zakresu 1. trafiają do węzła B, elementy z zakresu 2. są zapisywane w węzle C, a elementy z zakresu 3. trafiają do węzła A. (b) Po dodaniu do okręgu węzła D elementy z zakresu 4. są przenoszone do węzła D z węzłów B (zmniejszenie zakresu 3.) i C (zmniejszenie zakresu 3.)

mogą pochodzić z różnych istniejących węzłów, ponieważ nowy węzeł można powiązać z wieloma miejscami na okręgu. Przykładowo, jeśli dodany zostanie węzeł D, któremu odpowiadają dwie lokalizacje na okręgu (rysunek 24.2(b)), niektóre elementy z węzłów B i C zostaną przeniesione do węzła D. Elementy o kluczach odpowiadających *zakresowi* 4. na okręgu (patrz rysunek 24.2(b))



zostaną przeniesione do węzła D. Ten schemat umożliwia też *replikację* w wyniku rozmieszczenia określonej liczby replik elementu w kolejnych węzłach z okręgu (patrzac zgodnie z ruchem wskazówek zegara). *Sharding* jest wbudowany w tę metodę, a różne elementy z magazynu (pliku) są rozmieszczane w różnych węzłach rozproszonego klastra. Oznacza to, że elementy są poziomo dzielone (*sharding*) między węzły rozproszonego systemu. Gdy węzeł ulegnie awarii, obciążenie związane z jego elementami danych można rozdzielić między inne istniejące węzły, których nazwy znajdują się na okręgu po nazwach uszkodzonego węzła. Węzłom o większej pojemności może odpowiadać większa liczba lokalizacji, co ilustruje węzeł C na rysunku 24.2(a) — powoduje to zapisanie w nich większej liczby elementów niż w węzłach o mniejszej pojemności.

- **Spójność i wersjonowanie.** W systemie Voldemort na potrzeby zachowania spójności replik stosowana jest metoda podobna do tej z systemu DynamoDB. Dozwolone są jednoczesne zapisy w różnych procesach, dlatego jeśli element jest replikowany, z tym samym kluczem mogą być powiązane inne wartości w różnych węzłach. Spójność jest osiągana, gdy element jest wczytywany; stosuje się tu technikę *wersjonowania i naprawy przy odczycie*. Jednoczesne zapisy są dopuszczalne, ale każdy z nich jest powiązany z wartością *zegara wektorowego*. W momencie odczytu możliwe jest wczytanie z różnych węzłów różnych wersji tej samej wartości (powiązanych z tym samym kluczem). Jeśli system potrafi uzgodnić jedną ostateczną wartość, może ją zwrócić. W przeciwnym razie system przekazuje kilka wersji do aplikacji, a ta ustala na podstawie określonej semantyki jedną wersję i zwraca ją do węzłów.

### 24.4.3. Przykładowe inne magazyny z parami klucz-wartość

W tym punkcie pokrótce omówimy trzy inne magazyny z parami klucz-wartość. Należy zastrzec, że do tej kategorii można zaliczyć także wiele innych systemów, a tu przedstawiamy tylko kilka z nich.

**Magazyn z parami klucz-wartość firmy Oracle.** Oracle udostępnia jeden ze znanych relacyjnych systemów baz danych opartych na języku SQL, ale oferuje też system z parami klucz-wartość. Jego nazwa to **Oracle NoSQL Database**.

**Redis — pamięć podręczna i magazyn z parami klucz-wartość.** Redis różni się od innych omawianych tu systemów, ponieważ zapisuje dane w pamięci podręcznej utrzymywanej w pamięci głównej, aby zapewnić dodatkowy wzrost wydajności. Obsługiwana jest w nim replikacja typu nadrzędna-podrzędna, a system zapewnia wysoką dostępność i utrwalanie danych za pomocą zapisu danych z pamięci podręcznej na dysku.

**Apache Cassandra.** Cassandra to system NOSQL, który trudno jest zaliczyć do jednej kategorii. Czasem uznaje się go za system kolumnowy (patrz podrozdział 24.5), a czasem za system z parami klucz-wartość. Udostępnia funkcje z kilku kategorii systemów NOSQL i jest używany w serwisie Facebook oraz przez wiele innych organizacji.



## 24.5. Kolumnowe systemy NOSQL

Inną kategorią systemów NOSQL są systemy **kolumnowe**. **BigTable**, rozproszony system składowania big data w firmie Google, to dobrze znany przykład z tej kategorii. Jest on używany w wielu aplikacjach firmy Google wymagających składowania dużych ilości danych, np. w aplikacji Gmail. BigTable używa systemu **Google File System (GFS)** do składowania i udostępniania danych. Otwarty system **Apache HBase** jest nieco podobny do systemu BigTable Google'a, ale do składowania danych używa systemu **HDFS** (ang. *Hadoop Distributed File System*). System HDFS działa w wielu aplikacjach pracujących w chmurze, o których przeczytasz w rozdziale 25. W zakresie składowania danych HBase może też korzystać z systemu **Simple Storage System (S3)** Amazona. Innym znanym kolumnowym systemem NOSQL jest Cassandra, omówiony pokrótce w punkcie 24.4.3, ponieważ można go traktować także jako magazyn danych z parami klucz-wartość. W tym podrozdziale jako przykładowy kolumnowy system NOSQL omówimy system HBase.

System BigTable (i HBase) jest czasem nazywany *rzadkim wielowymiarowym rozproszonym trwałym posortowanym odwzorowaniem*, gdzie słowo *odwzorowanie* oznacza *kolekcję par (klucz, wartość)* (klucz jest *odwzorowywany* na wartość). Jedną z podstawowych różnic między systemami kolumnowymi a systemami z parami klucz-wartość (patrz podrozdział 24.4) jest *charakter klucza*. W systemach kolumnowych, takich jak HBase, klucz jest *wielowymiarowy* i ma kilka składowych. Zwykle jest to połączenie nazwy tabeli, klucza wiersza, nazwy kolumny i znacznika czasu. Zobaczysz, że nazwa kolumny obejmuje przeważnie dwa elementy: rodzinę kolumn i kwalifikator. Dalej bardziej szczegółowo omówimy, jak te zagadnienia są realizowane w systemie Apache HBase.

### 24.5.1. Model danych i wersjonowanie w systemie HBase

**Model danych w systemie HBase.** W modelu danych w systemie HBase dane są uporządkowane na podstawie *przestrzeni nazw, tabel, rodzin kolumn, kwalifikatorów kolumn, kolumn, wierszy i komórek danych*. Kolumny są identyfikowane za pomocą kombinacji (nazwa rodziny:kwalifikator kolumny). Dane są przechowywane w samoopisowej formie dzięki połączeniu kolumn z wartościami danych (te wartości to ciągi znaków). System HBase przechowuje *wiele wersji* elementu danych ze *znacznikiem czasu* powiązanym z każdą wersją. Dlatego wersje i znaczniki czasu też są częścią modelu danych w systemie HBase (to rozwiązanie jest podobne do wersjonowania atrybutów w czasowych bazach danych; patrz podrozdział 26.2). Podobnie jak w innych systemach NOSQL, ze składowanymi elementami danych powiązane są unikatowe klucze, co zapewnia szybki dostęp do tych elementów. Klucze identyfikują tu *komórki* w systemie składowania danych. Ponieważ nacisk położony jest na wysoką wydajność przy składowaniu dużych ilości danych, model danych obejmuje rozwiązania połączone ze składowaniem. Dalej opiszemy zagadnienia z obszaru modelowania danych w systemie HBase i zdefiniujemy terminologię. Należy zauważyć, że słowa *tabela*, *wiersz* i *kolumna* nie są tu stosowane w dokładnie tym samym znaczeniu co w relacyjnych bazach danych.

- **Tabele i wiersze.** Dane w systemie HBase są przechowywane w **tabelach**, a każda tabela ma nazwę. Dane w tabeli są składowane jako samoopisowe **wiersze**. Każdy wiersz ma unikatowy **klucz wiersza**. Tymi kluczami są ciągi znaków umożliwiające leksykograficzne uporządkowanie. Dlatego w kluczach wierszy nie mogą być używane znaki, których nie da się uporządkować leksykograficznie.

- **Rodziny kolumn, kwalifikatory kolumn i kolumny.** Tabela jest powiązana z jedną lub kilkoma **rodzinami kolumn**. Każda rodzina ma nazwę, a rodziny kolumn powiązane z tabelą *trzeba określić* w momencie jej tworzenia i nie można ich później zmienić. Na rysunku 24.3(a) pokazano, w jaki sposób można utworzyć tabelę. Po nazwie tabeli podawane są nazwy powiązanych z nią rodzin. Gdy dane są wczytywane do tabeli, każdą rodzinę można powiązać z wieloma **kwalifikatorami kolumn**. Jednak te kwalifikatory *nie są określane* w ramach tworzenia tabeli. Tak więc kwalifikatory sprawiają, że model danych jest samoopisowy (ponieważ można je podawać dynamicznie w trakcie tworzenia nowych wierszy i wstawiania ich do tabeli). **Kolumna** jest określana za pomocą połączenia RodzinaKolumn:KwalifikatoryKolumn. Rodziny kolumn są sposobem grupowania powiązanych kolumn (atrybutów w terminologii relacyjnej) na potrzeby ich składowania, a kwalifikatory nie są podawane w momencie tworzenia tabeli. Podaje się je podczas tworzenia danych i zapisywania ich w wierszach. Dane są więc *samoopisowe*, ponieważ dla każdego nowego wiersza danych można zastosować nowy kwalifikator kolumn (patrz rysunek 24.3(b)). Ważne jest jednak to, by programiści aplikacji wiedzieli, które kwalifikatory kolumn należą do poszczególnych rodzin (przy czym można tworzyć nowe kwalifikatory kolumn na bieżąco w momencie dodawania wierszy danych). Rodziny kolumn przypominają nieco *podział pionowy* (patrz podrozdział 23.2), ponieważ kolumny (atrybuty) używane razem z powodu przynależności do tej samej rodziny są zapisywane w tych samych plikach. Każda rodzina kolumn z tabeli jest zapisywana we własnych plikach w systemie plików HDFS.

#### (a) Tworzenie tabeli

```
create 'PRACOWNIK', 'ImięNazwisko', 'Adres', 'Szczegóły'
```

#### (b) Wstawianie wierszy do tabeli PRACOWNIK

```
put 'PRACOWNIK', 'wiersz1', 'ImięNazwisko:Imię', 'Jan'
put 'PRACOWNIK', 'wiersz1', 'ImięNazwisko:Nazwisko', 'Szewczyk'
put 'PRACOWNIK', 'wiersz1', 'ImięNazwisko:Pseudonim', 'Jasio'
put 'PRACOWNIK', 'wiersz1', 'Szczegóły:Stanowisko', 'Inżynier'
put 'PRACOWNIK', 'wiersz1', 'Szczegóły:Ocena', 'Dobra'
put 'PRACOWNIK', 'wiersz2', 'ImięNazwisko:Imię', 'Alicja'
put 'PRACOWNIK', 'wiersz2', 'ImięNazwisko:Nazwisko', 'Zalewska'
put 'PRACOWNIK', 'wiersz2', 'ImięNazwisko:DrugieImię', 'Joanna'
put 'PRACOWNIK', 'wiersz2', 'Szczegóły:Stanowisko', 'Administrator baz danych'
put 'PRACOWNIK', 'wiersz2', 'Szczegóły:Przełożony', 'Józef Bąk'
put 'PRACOWNIK', 'wiersz3', 'ImięNazwisko:Imię', 'Józef'
put 'PRACOWNIK', 'wiersz3', 'ImięNazwisko:Nazwisko', 'Bąk'
put 'PRACOWNIK', 'wiersz3', 'ImięNazwisko:InicjałDrImienia', 'E.'
put 'PRACOWNIK', 'wiersz3', 'ImięNazwisko:Przyrostek', 'Mł.'
put 'PRACOWNIK', 'wiersz3', 'Szczegóły:Stanowisko', 'CEO'
put 'PRACOWNIK', 'wiersz3', 'Szczegóły:Pensja', '100 000'
```

RYСУNEK 24.3. Przykłady z systemu HBase. (a) Tworzenie tabeli PRACOWNIK z trzema rodzinami kolumn: ImięNazwisko, Adres i Szczegóły. (b) Wstawianie danych do tabeli PRACOWNIK; w różnych wierszach można stosować różne samoopisowe kwalifikatory kolumn (Imię, Nazwisko, Pseudonim, DrugieImię, InicjałDrImienia, Przyrostek itd. w rodzinie kolumn ImięNazwisko; Stanowisko, Ocena, Przełożony, Pensja itd. w rodzinie kolumn Szczegóły). (c) Wybrane operacje CRUD w systemie HBase

- **Wersje i znaczniki czasu.** HBase potrafi przechowywać kilka **wersji** elementu danych. Z każdą z nich powiązany jest **znacznik czasu**. Znacznik czasu to długa liczba całkowita reprezentująca czas systemowy z momentu utworzenia wersji (nowsze wersje mają więc większe wartości znacznika czasu). W systemie HBase jako zerowy znacznik czasu używana jest północ 1 stycznia 1970 czasu UTC. Wartością systemowego znacznika czasu jest długa liczba całkowita określająca liczbę milisekund od zerowego znacznika czasu (podobna wartość jest zwracana przez funkcję `java.util.Date.getTime()` Javy oraz używana w systemie MongoDB). Ponadto użytkownik może definiować znaczniki czasu bezpośrednio w formacie `Date`, zamiast stosować znaczniki czasu wygenerowane przez system.
- **Komórki.** Komórka przechowuje podstawowy element danych systemu HBase. Klucz (adres) komórki jest określany za pomocą połączenia (tabela, idwiersza, rodzinakolumn, kwalifikatorkolumny, znacznikczasu). Jeśli znacznik czasu został pominięty, pobierana jest najnowsza wersja elementu — chyba że ustalona jest domyślna liczba wersji (np. trzy najnowsze wersje). Domyślną liczbę pobieranych wersji, a także domyślną liczbę wersji zachowywanych przez system można określić za pomocą parametrów w momencie tworzenia tabeli.
- **Przestrzenie nazw.** Przestrzeń nazw to kolekcja tabel. Przestrzeń nazw określa kolekcję tabel, które są zwykle używane razem w aplikacjach użytkowników. W terminologii relacyjnej jej odpowiednikiem jest baza danych zawierająca kolekcję tabel.

## 24.5.2. Operacje CRUD w systemie HBase

System HBase udostępnia niskopoziomowe operacje CRUD podobne do operacji z wielu innych systemów NOSQL. Składnię niektórych podstawowych operacji CRUD w systemie HBase pokazano na rysunku 24.3(c).

HBase udostępnia tylko niskopoziomowe operacje CRUD. To w aplikacji trzeba zaimplementować bardziej skomplikowane operacje, np. złączenia między wierszami z różnych tabel. Operacja *create* tworzy nową tabelę i pozwala określić powiązane z nią rodziny kolumn. Nie podaje się w niej jednak (co omówiono wcześniej) kwalifikatorów kolumn. Operacja *put* służy do wstawiania nowych danych lub nowych wierszy istniejących elementów danych. Operacja *get* pozwala pobrać dane powiązane z jednym wierszem tabeli, a operacja *scan* pobiera wszystkie wiersze.

## 24.5.3. Zagadnienia związane ze składowaniem danych i systemem rozproszonym w HBase

Każda tabela w systemie HBase jest podzielona na szereg **obszarów** (ang. *regions*). Każdy obszar przechowuje *zakres* kluczy wierszy tabeli. To dlatego klucze wierszy muszą być uporządkowane leksykograficznie. Każdy obszar obejmuje zestaw **magazynów** (ang. *stores*). Każda rodzina kolumn jest przypisywana do jednego magazynu w danym obszarze. Obszary są przypisywane do **serwerów obszarów** (węzłów składowania danych), gdzie są przechowywane. **Serwer nadrzędny** (węzeł nadrzędny) odpowiada za monitorowanie serwerów obszaru oraz za podział tabeli na obszary i przypisywanie ich do serwerów obszarów.

HBase korzysta z otwartego systemu **Apache ZooKeeper** w zakresie usług zarządzania nazwami, rozdzielania danych, synchronizacji danych między rozproszonymi serwerami systemu HBase, a także koordynacji i replikacji danych. Ponadto HBase używa systemu Apache HDFS do obsługi plików w środowisku rozproszonym. HBase jest więc zbudowany na podstawie narzędzi HDFS i ZooKeeper. ZooKeeper może utworzyć kilka replik w różnych węzłach, aby zapewnić dostępność, i przechowuje potrzebne dane w pamięci głównej w celu przyspieszenia dostępu do serwerów nadrzędnych i serwerów obszarów.

Nie przedstawiamy tu wielu innych szczegółów dotyczących architektury systemu rozproszonego i komponentów systemu HBase. Kompletnie omówienie tego systemu wykracza poza zakres niniejszej książki. Pełna dokumentacja systemu HBase jest dostępna w internecie (patrz wybrane publikacje).

## 24.6. Grafowe bazy NOSQL i system Neo4j

Inną kategorią systemów NOSQL są **grafowe bazy danych** lub **grafowe systemy NOSQL**. Dane są w nich reprezentowane za pomocą grafu, czyli zbioru wierzchołków (węzłów) i krawędzi. Węzły i krawędzie mogą być opisane, aby określały typy reprezentowanych encji i relacji. Zwykle możliwe jest przechowywanie danych powiązanych z pojedynczymi węzłami i krawędziami. Do kategorii grafowych baz danych można zaliczyć wiele systemów. Tu skupimy się na jednym z nich, Neo4j, wykorzystywanym w wielu zastosowaniach. Neo4j to otwarty system zaimplementowany w Javie. W punkcie 24.6.1 omówimy model danych z systemu Neo4j, a w punkcie 24.6.2 przedstawimy wprowadzenie do zapytań z tego systemu. Punkt 24.6.3 to przegląd środowiska rozproszonego i wybranych cech systemu Neo4j.

### 24.6.1. Model danych w systemie Neo4j

W modelu danych w systemie Neo4j dane są uporządkowane za pomocą **węzłów** i **związków**. Węzły i związki mogą mieć **właściwości**. Właściwości przechowują elementy danych powiązane z węzłami i związkami. Węzły mogą mieć **etykiety**. Węzły o *tej samej etykiecie* są grupowane w kolekcje określające podzbiory węzłów grafu bazy danych na potrzeby obsługi zapytań. Węzeł może mieć dowolną liczbę etykiet (w tym zero). Związki są skierowane. Każdy związek ma *węzeł początkowy* i *węzeł końcowy*, a także **typ związku**. Typ ma funkcję podobną do etykiety węzła — określa podobne związki tego samego typu. Właściwości można podawać za pomocą **wzorca odwzorowań**, który obejmuje jedną lub kilka par „nazwa:wartość” zapisanych w nawiasie klamrowym. Oto przykład: {Nazwisko: 'Szewczyk', Imię: 'Jan', InicjałDrImienia: 'B.'}.

W teorii grafów węzły i związki są zwykle nazywane *wierzchołkami* i *krawędziami*. Model danych w grafach systemu Neo4j przypomina nieco sposób reprezentowania danych w modelach ER i EER (patrz rozdziały 3. i 4.), przy czym występują ważne uwagi różnice. Jeśli porównać model grafów z Neo4j z elementami z modeli ER i EER, to węzły odpowiadają *encjom*, etykiety węzłów odpowiadają *typom encji* i *podklasom*, związki to *instancje związków*, a właściwości to *atrybuty*. Istotną różnicą jest to, że w Neo4j związki są *skierowane*, natomiast w modelach ER i EER — nieskierowane. Inna różnica dotyczy tego, że w Neo4j węzły mogą nie mieć etykiet, co w modelach ER i EER jest niedopuszczalne,

ponieważ każda encja musi być określonego typu. Trzecia ważna różnica polega na tym, że model grafowy w Neo4j jest podstawą wysoce wydajnego rozproszonego systemu danych, natomiast modele ER i EER są stosowane głównie do projektowania baz danych.

Na rysunku 24.4(a) pokazano, jak utworzyć kilka węzłów w systemie Neo4j. Węzły i związki można tworzyć na różne sposoby. Możesz np. wywołać odpowiednie operacje z interfejsów API systemu Neo4j. Tu pokażemy tylko wysokopoziomową składnię tworzenia węzłów i związków. Wykorzystamy polecenie `CREATE` z systemu Neo4j; pochodzi ono z wysokopoziomowego deklaratywnego języka zapytań **Cypher**. Neo4j udostępnia wiele opcji i możliwości tworzenia węzłów i związków za pomocą różnych interfejsów skryptowych, jednak ich kompletne omawianie wykracza poza zakres tej książki.

- **Etykiety i właściwości.** W momencie tworzenia węzła można podać jego etykietę. Można też tworzyć węzły bez etykiet. Na rysunku 24.4(a) etykiety węzłów to `PRACOWNIK`, `DZIAŁ`, `PROJEKT` i `LOKALIZACJA`. Tworzone węzły odpowiadają danym z bazy `FIRMA` z rysunku 5.6, choć z kilkoma modyfikacjami. Przykładowo, używamy atrybutu `IDPRAC` zamiast `PESEL` i uwzględniamy tylko mały podzbiór danych. Właściwości są umieszczane w nawiasach klamrowych `{}`. Niektóre węzły mogą mieć wiele etykiet. Przykładowo, ten sam węzeł może mieć etykiety `OSOBA`, `PRACOWNIK` i `KIEROWNIK`. Należy je rozdzielić dwukropkami: `OSOBA:PRACOWNIK:KIEROWNIK`. Użycie kilku etykiet przypomina sytuację, gdy encja należy do typu encji (`OSOBA`) oraz do podklas tego typu (`PRACOWNIK` i `KIEROWNIK`) w modelu EER (patrz rozdział 4.). To rozwiązanie można też stosować w innych celach.
- **Związki i typy związków.** Na rysunku 24.4(b) pokazano kilka przykładowych związków w systemie Neo4j. Związki te pochodzą z bazy `FIRMA` z rysunku 5.6. Strzałka (`→`) określa kierunek związku, jednak poruszać się można w obu kierunkach. Typy związków (etykiety) na rysunku 24.4(b) to `PracujeW`, `Kierownik`, `ZlokalizowanyW` i `PracujeNad`. Na rysunku właściwości (tu: `Godziny`) mają tylko związki typu `PracujeNad`.
- **Ścieżki.** Ścieżka określa drogę w części grafu. Zwykle jest używana jako fragment zapytania określający wzorzec; zapytanie pobiera wtedy z grafu dane pasujące do tego wzorca. Ścieżka jest zwykle określana za pomocą węzła początkowego, po którym następuje jeden lub kilka związków prowadzących do jednego lub kilku węzłów zgodnych ze wzorcem. Przypomina to nieco wyrażenia ścieżkowe opisane w rozdziałach 12. i 13. w kontekście języków zapytań dla obiektowych baz danych (OQL) i formatu XML (XPath i XQuery).
- **Opcjonalny schemat.** W Neo4j **schemat** jest opcjonalny. Grafy można tworzyć i stosować bez schematu, jednak w Neo4j w wersji 2.0 dodano kilka funkcji dotyczących schematów. Podstawowy mechanizm związany z tworzeniem schematu to budowanie indeksów i ograniczeń na podstawie etykiet i właściwości. Można np. utworzyć dla właściwości etykiety odpowiednik ograniczenia klucza, aby wszystkie węzły powiązane z daną etykietą musiały mieć unikatowe wartości danej właściwości.
- **Indeksowanie i identyfikatory węzła.** W momencie tworzenia węzła system Neo4j tworzy dla każdego węzła wewnętrzny unikatowy identyfikator definiowany przez system. Aby wydajnie pobierać poszczególne węzły za pomocą innych właściwości, użytkownik może utworzyć dla zbioru węzłów o określonej etykietcie

**indeksy.** Zwykle indeksy są tworzone na jednej lub kilku właściwościach węzłów ze zbioru. Można np. wykorzystać właściwość `IDPRAC` do indeksowania węzłów o etykiecie `PRACOWNIK`, właściwość `NRDZ` do indeksowania węzłów o etykiecie `DZIAŁ` i właściwość `NRPROJ` do indeksowania węzłów o etykiecie `PROJEKT`.

## 24.6.2. Język zapytań Cypher w systemie Neo4j

W Neo4j dostępny jest wysokopoziomowy język zapytań Cypher. Istnieją w nim deklaratywne polecenia do tworzenia węzłów i związków (patrz rysunki 24.4(a) i (b)), a także do wyszukiwania węzłów i związków na podstawie wzorców. W języku Cypher możliwe jest też usuwanie i modyfikowanie danych. We wcześniejszym punkcie wprowadziliśmy polecenie `CREATE`. Teraz przedstawimy krótki przegląd innych mechanizmów tego języka.

Zapytanie w języku Cypher składa się z *klauzul*. Gdy zapytanie obejmuje kilka klauzul, wynik z jednej klauzuli można wykorzystać jako dane wejściowe w następnej. Przedstawimy tu podstawy tego języka, omawiając wybrane klauzule za pomocą przykładów. Nie ma to być szczegółowa prezentacja języka Cypher, a jedynie wprowadzenie do niektórych jego funkcji. Rysunek 24.4(c) to podsumowanie niektórych podstawowych klauzul, jakie mogą się składać na zapytanie w języku Cypher. Język ten pozwala tworzyć złożone zapytania i aktualizacje dotyczące grafowej bazy danych. Na rysunku 24.4(d) przedstawiamy kilka przykładów ilustrujących proste zapytania w języku Cypher.

### (a) Tworzenie węzłów z danymi dotyczącymi firmy (z rysunku 5.6)

```
CREATE (e1: PRACOWNIK, {IdPrac: '1', Nazwisko: 'Szewczyk', Imię: 'Jan',
    InicjałDrImienia: 'B.'})
CREATE (e2: PRACOWNIK, {IdPrac: '2', Nazwisko: 'Wieszczycki', Imię:
    'Franciszek'})
CREATE (e3: PRACOWNIK, {IdPrac: '3', Nazwisko: 'Zalewska', Imię: 'Alicja'})
CREATE (e4: PRACOWNIK, {IdPrac: '4', Nazwisko: 'Wojtczak', Imię: 'Janina',
    InicjałDrImienia: 'S.'})
...
CREATE (d1: DZIAŁ, {NumerDz: '5', NazwaDz: 'Badania'})
CREATE (d2: DZIAŁ, {NumerDz: '4', NazwaDz: 'Administracja'})
...
CREATE (p1: PROJEKT, {NumerProj: '1', NazwaProj: 'ProduktX'})
CREATE (p2: PROJEKT, {NumerProj: '2', NazwaProj: 'ProduktY'})
CREATE (p3: PROJEKT, {NumerProj: '10', NazwaProj: 'Komputeryzacja'})
CREATE (p4: PROJEKT, {NumerProj: '20', NazwaProj: 'Reorganizacja'})
...
CREATE (l1: LOKALIZACJEDZIAŁÓW, {LokalizacjaDz: 'Szczecin'})
CREATE (l2: LOKALIZACJEDZIAŁÓW, {LokalizacjaDz: 'Bydgoszcz'})
CREATE (l3: LOKALIZACJEDZIAŁÓW, {LokalizacjaDz: 'Poznań'})
CREATE (l4: LOKALIZACJEDZIAŁÓW, {LokalizacjaDz: 'Świebodzin'})
...
```



**(b) Tworzenie związków na podstawie danych z relacji FIRMA (z rysunku 5.6)**

```

CREATE (e1) - [ : PracujeDla ] -> (d1)
CREATE (e3) - [ : PracujeDla ] -> (d2)
...
CREATE (d1) - [ : Kierownik ] -> (e2)
CREATE (d2) - [ : Kierownik ] -> (e4)
...
CREATE (d1) - [ : ZlokalizowanyW ] -> (l1)
CREATE (d1) - [ : ZlokalizowanyW ] -> (l3)
CREATE (d1) - [ : ZlokalizowanyW ] -> (l4)
CREATE (d2) - [ : ZlokalizowanyW ] -> (l2)
...
CREATE (e1) - [ : PracujeNad, {Godziny: '32.5'} ] -> (p1)
CREATE (e1) - [ : PracujeNad, {Godziny: '7.5'} ] -> (p2)
CREATE (e2) - [ : PracujeNad, {Godziny: '10.0'} ] -> (p1)
CREATE (e2) - [ : PracujeNad, {Godziny: '10.0'} ] -> (p2)
CREATE (e2) - [ : PracujeNad, {Godziny: '10.0'} ] -> (p3)
CREATE (e2) - [ : PracujeNad, {Godziny: '10.0'} ] -> (p4)
...

```

**(c) Podstawowa uproszczona składnia niektórych często stosowanych klauzul z języka Cypher**

Wyszukiwanie węzłów i związków pasujących do wzorca: MATCH <wzorzec>

Określanie agregacji i innych zmiennych w zapytaniach: WITH <specyfikacje>

Określanie warunków dotyczących pobieranych danych: WHERE <warunek>

Określanie zwracanych danych: RETURN <dane>

Sortowanie zwracanych danych: ORDER BY <dane>

Ograniczanie liczby zwracanych elementów danych: LIMIT <maks. liczba>

Tworzenie węzłów: CREATE <węzeł, opcjonalnie etykiety i właściwości>

Tworzenie związków: CREATE <związek, typ związku i opcjonalnie właściwości>

Usuwanie: DELETE <węzły lub związki>

Określanie wartości i etykiet właściwości: SET <wartości i etykiety właściwości>

Usuwanie wartości i etykiet właściwości: REMOVE <wartości i etykiety właściwości>

**(d) Przykładowe proste zapytania w języku Cypher**

1. MATCH (d : DZIAŁ {NumerDz: '5'}) - [ : ZlokalizowanyW ] → (loc)  
RETURN d.NazwaDz, loc.LokalizacjaDz
2. MATCH (e: PRACOWNIK {IdPrac: '2'}) - [ w: PracujeNad ] → (p)  
RETURN e.Nazwisko, w.Godziny, p.NazwaProj
3. MATCH (e) - [ w: PracujeNad ] → (p: PROJEKT {NumerProj: 2})  
RETURN p.NazwaProj, e.Nazwisko, w.Godziny
4. MATCH (e) - [ w: PracujeNad ] → (p)  
RETURN e.Nazwisko, w.Godziny, p.NazwaProj  
ORDER BY e.Nazwisko
5. MATCH (e) - [ w: PracujeNad ] → (p)  
RETURN e.Nazwisko, w.Godziny, p.NazwaProj  
ORDER BY e.Nazwisko  
LIMIT 10



```

6. MATCH (e) - [ w: PracujeNad ] → (p)
   WITH e, COUNT(p) AS liczbaProj
   WHERE liczbaProj > 2
   RETURN e.Nazwisko, liczbaProj
   ORDER BY liczbaProj
7. MATCH (e) - [ w: PracujeNad ] → (p)
   RETURN e, w, p
   ORDER BY e.Nazwisko
   LIMIT 10
8. MATCH (e: PRACOWNIK {IdPrac: '2'})
   SET e.Stanowisko = 'Inżynier'

```

RYSUNEK 24.4. Przykłady zastosowania języka Cypher w systemie Neo4j. (a) Tworzenie węzłów. (b) Tworzenie związków. (c) Podstawowa składnia zapytań w języku Cypher. (d) Przykładowe zapytania w języku Cypher

W zapytaniu 1. z rysunku 24.4(d) pokazano, jak używać w zapytaniu klauzul `MATCH` i `RETURN`. To zapytanie pobiera lokalizacje działu nr 5. Klauzula `MATCH` obejmuje *wzorzec* i *zmienne zapytania* (*d* i *loc*), a `RETURN` określa pobierany wynik zapytania za pomocą zmiennych. W zapytaniu 2. występują trzy zmienne (*e*, *w* i *p*). To zapytanie zwraca projekty i liczbę godzin tygodniowo, jaką poświęcają im pracownicy o `IDPRAC = 2`. Z kolei zapytanie 3. zwraca osoby pracujące nad projektem o `NRPROJ = 2` i liczbę godzin przeznaczanych przez tych pracowników na ten projekt tygodniowo. Zapytanie 4. to przykład zastosowania klauzuli `ORDER BY`. Zwraca ono wszystkich pracowników i projekty, w których uczestniczą. Dane są sortowane według nazwisk pracowników. Można też ograniczyć liczbę zwracanych wyników, stosując klauzulę `LIMIT` tak jak w zapytaniu 5., które zwraca tylko pierwszych 10 odpowiedzi.

Zapytanie 6. ilustruje zastosowanie klauzuli `WITH` i agregacji. `WITH` można stosować do rozdzielania klauzul w zapytaniu nawet wtedy, gdy agregacja nie jest stosowana. Zapytanie 6. ilustruje klauzulę `WHERE`, gdzie zdefiniowane są dodatkowe warunki. To zapytanie zwraca osoby pracujące nad więcej niż dwoma projektami oraz liczbę projektów, którymi zajmują się te osoby. W wyniku zapytania często zwracane są też węzły i związki, a nie (jak w opisanych wcześniej zapytaniach) wartości właściwości węzłów. Zapytanie 7. jest podobne do zapytania 5., ale zwraca tylko węzły i związki. Dlatego wynik zapytania można wyświetlić jako graf za pomocą narzędzia wizualizacyjnego z systemu Neo4j. Można też dodawać oraz usuwać etykiety i właściwości węzłów. W zapytaniu 8. pokazano, jak dodać do węzła dodatkowe właściwości. Tu do węzła pracownika dodawana jest właściwość `STANOWISKO`.

Te informacje stanowią krótkie wprowadzenie do języka zapytań Cypher z systemu Neo4j. Kompletny podręcznik dotyczący tego języka jest dostępny w internecie (patrz wybrane publikacje).

### 24.6.3. Cechy interfejsów i systemu rozproszonego w Neo4j

Neo4j udostępnia też inne interfejsy, które można wykorzystać do tworzenia, pobierania i aktualizowania węzłów oraz związków w grafowych bazach danych. Dostępne są dwie podstawowe wersje tego systemu: Enterprise (obejmująca dodatkowe możliwości) i Community. W tym punkcie omawiamy wybrane dodatkowe funkcje systemu Neo4j.

- **Wersje Enterprise i Community.** Obie wersje obsługują grafowy model danych i system składowania danych, zgodny z grafami język zapytań Cypher, a także kilka innych interfejsów, w tym wysoce wydajny natywny interfejs API oraz sterowniki dla kilku popularnych języków programowania, takich jak Java, Python, PHP i REST (ang. *Representational State Transfer*). Ponadto obie wersje zapewniają właściwości ACID. Wersja Enterprise udostępnia dodatkowe funkcje zwiększające wydajność, np.: pamięć podręczną, klastrowanie danych i blokady.
- **Interfejs do wizualizowania grafów.** Neo4j udostępnia interfejs do wizualizowania grafów, dzięki czemu podzbiór węzłów i krawędzi z grafowej bazy danych można wyświetlić w formie grafu. To narzędzie pozwala też wizualizować wyniki zapytań w postaci grafu.
- **Replikacja nadrzędna-podrzędna.** Neo4j można skonfigurować w klastrze rozproszonych węzłów (komputerów), gdzie jeden z węzłów jest konfigurowany jako nadrzędny. Dane i indeksy są w pełni replikowane w każdym węźle klastra. W rozproszonym klastrze można skonfigurować różne sposoby synchronizacji danych między węzłami nadrzędnymi i podrzędnymi.
- **Pamięć podręczna.** Można skonfigurować pamięć podręczną w pamięci głównej, aby przechowywać w niej dane grafu w celu zwiększenia wydajności.
- **Dzienniki logiczne.** Można przechowywać dzienniki na potrzeby odtwarzania danych po awariach.

Kompletne omówienie wszystkich funkcji i interfejsów systemu Neo4j wykracza poza zakres tej książki. Pełna dokumentacja tego systemu jest dostępna w internecie (patrz wybrane publikacje).

## 24.7. Podsumowanie

W tym rozdziale omówiono kategorię systemów baz danych nazywanych systemami NOSQL. Najważniejsze są w nich wydajne składowanie i pobieranie dużych ilości danych (big data). Aplikacje korzystające z systemów tego typu to: serwisy społecznościowe, systemy zarządzania odsyłaczami internetowymi, profilami użytkowników, marketingiem i sprzedażą, postami i tweetami, mapami drogowymi, danymi przestrzennymi i e-mailami. Nazwa *NOSQL* jest zwykle interpretowana jako *Not Only SQL* (nie tylko SQL), a nie jako *No to SQL* (nie dla SQL-a), i ma oznaczać, że w wielu aplikacjach jako uzupełnienie potrzeb z zakresu zarządzania danymi niezbędne są rozwiązania inne niż tradycyjne relacyjne systemy oparte na SQL-u. Systemami NOSQL są rozproszone bazy danych i rozproszone systemy składowania danych. Nacisk jest w nich położony na składowanie częściowo ustrukturyzowanych danych, wysoką wydajność, dostępność, replikację danych i skalowalność, a nie na natychmiastową spójność danych, rozbudowane języki zapytań i przechowywanie danych strukturalnych.

Zaczęliśmy w podrozdziale 24.1 od wprowadzenia do systemów NOSQL oraz omówienia ich cech i tego, czym różnią się od systemów opartych na języku SQL. Cztery ogólne kategorie systemów NOSQL to systemy dokumentowe, z parami klucz-wartość, kolumnowe i grafowe. W podrozdziale 24.2 opisano, w jaki sposób w systemach NOSQL trakto-

wana jest spójność między wieloma replikami (kopiami). Używany jest w nich model spójności ostatecznej. Omówiono też twierdzenie CAP, które pomaga zrozumieć nacisk kładziony w systemach NOSQL na dostępność. W poszczególnych rozdziałach od 24.3 do 24.6 zaprezentowano przegląd każdej z czterech podstawowych kategorii systemów NOSQL. Zaczęliśmy od systemów dokumentowych (podrozdział 24.3), potem przeszliśmy do systemów z parami klucz-wartość (podrozdział 24.4), systemów kolumnowych (podrozdział 24.5) i systemów grafowych (podrozdział 24.6). Zwróciliśmy też uwagę na to, że niektóre systemy NOSQL nie wpasowują się w jedną kategorię, a wykorzystują techniki z dwóch lub więcej kategorii.

## Pytania powtórkowe

- 24.1. Na potrzeby jakiego rodzaju aplikacji opracowano systemy NOSQL?
- 24.2. Wymień główne kategorie systemów NOSQL. Podaj kilka systemów NOSQL z każdej z tych kategorii.
- 24.3. Podaj podstawowe cechy systemów NOSQL związane z modelami danych i językami zapytań.
- 24.4. Podaj podstawowe cechy systemów NOSQL dotyczące systemów rozproszonych i rozproszonych baz danych.
- 24.5. Czym jest twierdzenie CAP? Jakie trzy właściwości (spójność, dostępność, odporność na podział) są najważniejsze w systemach NOSQL?
- 24.6. Wymień podobieństwa i różnice między spójnością w twierdzeniu CAP a spójnością we właściwościach ACID.
- 24.7. Jakie mechanizmy modelowania danych są używane w systemie MongoDB? Podaj podstawowe operacje CRUD dostępne w tym systemie.
- 24.8. Omów replikację i sharding w systemie MongoDB.
- 24.9. Opisz mechanizmy modelowania danych z systemu DynamoDB.
- 24.10. Opisz schemat mieszania stabilnego używany na potrzeby rozdzielania danych, replikacji i shardingu. Jak przebiega zapewnianie spójności i wersjonowanie w systemie Voldemort?
- 24.11. Jakie mechanizmy modelowania danych są używane w kolumnowych systemach NOSQL i w systemie HBase?
- 24.12. Wymień podstawowe operacje CRUD w systemie HBase.
- 24.13. Omów techniki składowania danych i metody typowe dla systemów rozproszonych używane w systemie HBase.
- 24.14. Podaj mechanizmy modelowania danych stosowane w grafowym systemie NOSQL Neo4j.
- 24.15. Jak się nazywa język zapytań w systemie Neo4j?
- 24.16. Omów interfejsy i cechy systemu rozproszonego z Neo4j.

## Wybrane publikacje

Rozproszony system składowania danych BigTable firmy Google po raz pierwszy opisano w pracy Changa i in. (2006). System składowania par klucz-wartość Dynamo firmy Amazon po raz pierwszy przedstawiono w pracy DeCandii i in. (2007). Istnieje wiele tekstów porównujących różne systemy NOSQL z relacyjnymi systemami opartymi na języku SQL (np. Parker i in., 2013). W innych pracach systemy NOSQL są porównywane z innymi systemami tego rodzaju (np.: Cattell, 2010, Hecht i Jablonski, 2011, Abramova i Bernardino, 2013).

Dokumentację, podręczniki użytkownika i samouczki dotyczące wielu systemów NOSQL znajdziesz w internecie. Oto kilka przykładów:

Samouczki dotyczące systemu MongoDB: <http://docs.mongodb.org/manual/tutorial/>.

Podręcznik systemu MongoDB: <http://docs.mongodb.org/manual/>.

Dokumentacja systemu Voldemort: <http://docs.project-voldemort.com/voldemort/>.

Witryna systemu Cassandra: <http://cassandra.apache.org>.

Witryna systemu HBase: <http://HBase.apache.org>.

Dokumentacja systemu Neo4j: <http://neo4j.com/docs/>.

Ponadto w wielu witrynach systemy NOSQL są podzielone na dodatkowe kategorie na podstawie ich przeznaczenia. Przykładową witryną tego rodzaju jest <http://nosql-database.org>.

## Technologie z obszaru big data oparte na modelu MapReduce i systemie Hadoop<sup>1</sup>

Ilość danych na świecie rośnie od czasu powstania sieci WWW mniej więcej w 1994 r. Nie wiele potem pojawiły się pierwsze wyszukiwarki — AltaVista (przejęta przez Yahoo! w 2003 r. i przekształcona później w wyszukiwarkę Yahoo!) i Lycos (wyszukiwarka i portal internetowy; Lycos utworzono wkrótce po powstaniu internetu). Później wyszukiwarki te znalazły się w cieniu takich narzędzi jak Google i Bing. Następnie pojawiły się różne sieci społecznościowe, takie jak Facebook (2004 r.) i Twitter (2006 r.). LinkedIn, sieć dla pracowników udostępniona w 2003 r., chwali się 250 milionami użytkowników na całym świecie. Facebook ma obecnie ponad 1,3 miliarda użytkowników. Spośród nich ok. 800 milionów korzysta z Facebooka każdego dnia. Na początku 2014 r. szacowano, że Twitter miał 980 milionów użytkowników. W październiku 2012 r. ogłoszono, że liczba tweetów dziennie sięgnęła miliarda. Te statystyki są stale aktualizowane i można je łatwo znaleźć w internecie.

Ważnym skutkiem powstania i wykładniczego wzrostu popularności internetu, dzięki czemu laicy z całego świata zetknęli się z informatyką, jest to, że zwykli ludzie zaczęli tworzyć różnego rodzaju transakcje i treści, co doprowadziło do generowania nowych danych. Użytkownicy i konsumenci danych multimedialnych potrzebują systemów, które natychmiastowo dostarczają dostosowane do użytkownika dane z olbrzymich magazynów, a jednocześnie same wytwarzają olbrzymie ilości danych. W efekcie nastąpił wykładniczy wzrost ilości danych generowanych i przekazywanych sieciami na całym świecie. Ponadto firmy i instytucje rządowe elektronicznie rejestrują wszystkie transakcje każdego klienta, sprzedawcy i dostawcy. W ten sposób dane są zbierane w tzw. hurtowniach danych (opiszemy je w rozdziale 29.). Do tej góry danych możemy dodać dane generowane przez czujniki wbudowane w urządzenia takie jak smartfony, inteligentne liczniki energii, samochody i różnego rodzaju gadżety oraz maszyny, które wykrywają, tworzą i przesyłają dane w internecie rzeczy (ang. *internet of things*). Oczywiście trzeba też uwzględnić dane rejestrowane codziennie przez satelity i sieci komunikacyjne.

Zdumiewający wzrost szybkości generowania danych powoduje, że ilość danych w jednym repozytorium może być mierzona w petabajtach ( $10^{15}$  bajtów, czyli ok.  $2^{50}$  bajtów) lub terabajtach (np. 1000 terabajtów). Nazwa *big data* trafiła do języka codziennego i oznacza takie właśnie olbrzymie ilości danych. W raporcie firmy McKinsey<sup>2</sup> zdefinio-

---

<sup>1</sup> Za wkład w powstanie tego rozdziału dziękujemy Harishowi Butanemu, członkowi grupy Hive Program Management Committee, i Balajiemu Palanisamy'emu z University of Pittsburgh.

<sup>2</sup> Niniejsze wprowadzenie jest w dużej części oparte na raporcie na temat big data (McKinsey, 2012) opracowanym przez McKinsey Global Institute.

wano określenie *big data* jako zbiory danych na tyle duże, że nie jest możliwe ich zapisywanie, składowanie, analizowanie i zarządzanie nimi w typowych SZBD. Znaczenie i skutki tej powodzi danych zostały odzwierciedlone w pewnych faktach przytoczonych w raporcie firmy McKinsey:

- Dysk kosztujący ok. 2000 złotych pozwala zapisać całą znaną na świecie muzykę.
- Każdego miesiąca na Facebooku zapisywanych jest 30 miliardów materiałów.
- W 15 z 17 sektorów ekonomii Stanów Zjednoczonych zapisywanych jest więcej danych niż w Bibliotece Kongresu (która w 2011 r. obejmowała 235 terabajtów danych).
- Obecnie w Stanach Zjednoczonych potrzebnych jest ponad 140 tysięcy analityków *deep data* i ponad 1,5 miliona menedżerów obeznanych z danymi. Analizy *deep data* obejmują analizy nastawione na odkrywanie wiedzy.

Big data są wszechobecne, dlatego w każdym sektorze ekonomii trwają starania o wykorzystanie ich z użyciem technologii, które pomogą użytkownikom danych i menedżerom podejmować lepsze decyzje na podstawie dowodów historycznych. Oto informacje z raportu firmy McKinsey:

Szacujemy, że gdyby [system] opieki zdrowotnej w Stanach Zjednoczonych potrafił w kreatywny i skuteczny sposób wykorzystać big data do wzrostu wydajności i jakości usług, dane z tego sektora mogłyby przynieść wartość ponad 300 miliardów dolarów każdego roku.

Big data dają nieskończone możliwości w zakresie szybkiego przekazywania konsumentom informacji, które będą przydatne w podejmowaniu decyzji, wykrywaniu potrzeb, poprawie wydajności, dostosowywaniu produktów i usług, zapewnianiu decydom skuteczniejszych narzędzi algorytmicznych i generowaniu wartości dzięki innowacjom w obszarach nowych produktów, usług i modeli biznesowych. Firma IBM potwierdza te spostrzeżenia w wydanej niedawno książce<sup>3</sup>, w której wyjaśniono, dlaczego IBM podjął się misji analizowania big data na poziomie przedsiębiorstw. W książce firmy IBM opisano różnego rodzaju zastosowania analiz:

- **Analizy opisowe i prognostyczne.** Analizy opisowe są związane z raportami o tym, co się stało, z analizowaniem danych, które przyczyniły się do powstania obecnej sytuacji, i z monitorowaniem nowych danych w celu ustalenia, co dzieje się obecnie. W analizach prognostycznych techniki statystyczne i techniki drążenia danych (patrz rozdział 28.) są wykorzystywane do prognozowania tego, co wydarzy się w przyszłości.
- **Analizy zakończone sugestiami.** Są to analizy prowadzące do rekomendowania działań.
- **Analizy mediów społecznościowych.** Polegają na analizowaniu sentymentu w celu oceny opinii publicznej na temat określonych kwestii lub wydarzeń. Umożliwiają też użytkownikom wykrywanie wzorców zachowań i preferencji, co pomaga firmom oferować produkty i usługi dostosowane do odbiorców.

---

<sup>3</sup> Patrz IBM (2014), *Analytics Across the Enterprise: How IBM Realizes Business Value from Big Data and Analytics*.

- **Analizy encji.** To stosunkowo nowa dziedzina, polegająca na grupowaniu danych na temat interesujących encji i dowiadywaniu się o nich nowych rzeczy.
- **Obliczenia kognitywne.** Są związane z rozwijaniem systemów obliczeniowych, które komunikują się z ludźmi, aby zapewnić im porady i lepsze zrozumienie pewnych kwestii.

W innej książce Bill Franks z firmy Teradata<sup>4</sup> przedstawia podobne wnioski. Jego zdaniem wykorzystanie big data do usprawnienia analiz jest konieczne, aby uzyskać przewagę konkurencyjną w każdej branży. Franks pokazuje, jak w dowolnej organizacji opracować „zaawansowane środowisko analiz big data” w celu odkrywania nowych możliwości biznesowych.

Z branżowych publikacji ekspertów wynika, że korzystanie z big data wkracza w nowy obszar, w którym te dane posłużą do budowania aplikacji analitycznych pozwalających zwiększyć produktywność, poprawić jakość i osiągnąć wzrost w firmach wszelkiego rodzaju. W tym rozdziale omówiono opracowane w ostatnim dziesięcioleciu technologie korzystania z big data. Skoncentrujemy się na technologiach związanych z narzędziami MapReduce i Hadoop. Stosuje się je w większości otwartych projektów z obszaru big data. Nie będzie możliwe omówienie tu zastosowań big data w analityce. Jest to rozległy obszar sam w sobie. Niektóre podstawowe zagadnienia dotyczące drażenia danych są opisane w rozdziale 28. Jednak dostępne obecnie możliwości analityczne znacznie wykraczają poza omówione tu podstawowe kwestie.

W podrozdziale 25.1 przedstawimy najważniejsze aspekty big data. W podrozdziale 25.2 opiszemy historię technologii MapReduce i Hadoop oraz różne wersje Hadoopa. Podrozdział 25.3 zawiera omówienie systemu plików używanego w Hadoopie — **HDFS** (ang. *Hadoop Distributed File System*). Opiszemy jego architekturę, obsługiwane operacje wejścia-wyjścia i skalowalność. W podrozdziale 25.4 zaprezentujemy szczegółowe informacje o modelu MapReduce, w tym środowisko uruchomieniowe oraz wysokopoziomowe interfejsy Pig i Hive. Zademonstrujemy też możliwości modelu MapReduce na przykładzie złączeń relacyjnych implementowanych na różne sposoby. Podrozdział 25.5 jest poświęcony nowszym wersjom omawianych technologii: Hadoop 2, MapReduce 2 i YARN, gdzie zarządzanie zasobami jest oddzielone od zarządzania pracami. Najpierw wspomnimy o powodach stosowania tych rozwiązań, a następnie zaprezentujemy ich architekturę i inne platformy opracowane z użyciem platformy YARN. W podrozdziale 25.6 omówimy ogólne kwestie związane z technologiami MapReduce i Hadoop. Najpierw opiszemy te transakcje w porównaniu z równoległymi SZBD. Dalej przedstawimy je w kontekście chmur obliczeniowych. Objaśnimy też poprawę wydajności na podstawie lokalności danych. Z kolei opiszemy YARN jako platformę usług z obszaru danych, a także ogólne wyzwania związane z big data. Rozdział zakończymy podrozdziałem 25.7, w którym opiszemy rozwijane obecnie projekty, oraz podsumowaniem.

---

<sup>4</sup> Patrz Franks (2013), *Taming The Big Data Tidal Wave*.



## 25.1. Czym jest big data?

*Big data* staje się popularnym, a nawet modnym określeniem. Ludzie posługują się nim, gdy w analizach uwzględniane są duże ilości danych. Zdaniem takich osób użycie tej nazwy sprawi, że analizy wydadzą się bardziej zaawansowane. Jednak pojęcie *big data* dotyczy zbiorów danych o takiej wielkości, że typowe narzędzia bazodanowe nie radzą sobie z ich zapisywaniem, składowaniem i analizowaniem oraz zarządzaniem nimi. W dzisiejszych środowiskach wielkość zbiorów danych, jakie można uznać za big data, sięga od terabajtów ( $10^{12}$  bajtów) lub petabajtów ( $10^{15}$  bajtów) do eksabajtów ( $10^{18}$  bajtów). To, jakie dane zostaną uznane za big data, zależy od branży, sposobu korzystania z danych, ilości uwzględnianych danych historycznych i wielu innych cech. Znane przedsiębiorstwo Gartner Group, którego działania śledzą firmy chcące poznać najnowsze trendy, w 2011 r. scharakteryzowało big data za pomocą trzech liter V: *volume* (ilość), *velocity* (szybkość generowania) i *variety* (różnorodność). Inni badacze dodali do tych cech inne, np. *veracity* (wiarygodność) i *value* (wartość). Zobaczmy pokrótce, co oznaczają te cechy.

**Ilość.** Ilość danych oczywiście dotyczy wielkości danych zarządzanych przez system. Dane generowane automatycznie zwykle zajmują dużo miejsca. Mogą to być dane z czujników (np. w fabrykach lub zakładach przetwórczych), z urządzeń skanujących (np. z czytników kart procesorowych i kredytowych) lub z urządzeń pomiarowych (takich jak inteligentne liczniki i urządzenia do pomiarów środowiskowych).

Oczekuje się, że **przemysłowy internet rzeczy** (ang. *industrial internet of things* — IIOT lub IOT) wywoła rewolucję, która usprawni wydajność operacyjną przedsiębiorstw i otworzy nowe obszary zastosowań dla inteligentnych technologii. IOT spowoduje podłączenie do internetu miliardów urządzeń stale generujących dane. Przykładowo, technologia NGS w obszarze sekwencjonowania DNA spowoduje, że ilość danych z sekwencjami genów wzrośnie wykładniczo.

Rozwijanych jest wiele dodatkowych rozwiązań, które powoli trafiają do użytku. Te rozwiązania to np.: zdalne wykrywanie podziemnych źródeł energii, monitorowanie środowiskowe, monitorowanie i regulacja ruchu za pomocą automatycznych czujników zamontowanych na samochodach i w drogach, zdalne monitorowanie stanu pacjentów za pomocą specjalnych skanerów i sprzętu, a także dokładniejsze kontrolowanie stanu magazynu i uzupełnianie braków za pomocą technologii **RFID** (ang. *radio-frequency identification*) i innych mechanizmów. Wszystkie te nowinki są powiązane z dużą ilością danych. Sieci społecznościowe, takie jak Twitter i Facebook, mają na całym świecie setki milionów użytkowników, którzy generują nowe dane wraz z każdą wysłaną wiadomością i zamieszczanym postem. W październiku 2012 r. liczba tweetów dziennie na Twitterze osiągnęła 500 milionów<sup>5</sup>. Ilość danych potrzebnych do zapisania jednej sekundy filmu w jakości HD może być równa 2000 stron tekstu. Dlatego dane multimedialne przesyłane do serwisu YouTube i podobnych systemów składowania wideo zajmują znacznie więcej miejsca niż proste dane liczbowe lub tekstowe. W 2010 r. firmy przechowywały ponad 13 eksabajtów ( $10^{18}$  bajtów) danych, czyli przeszło 50 000 więcej niż w Bibliotece Kongresu<sup>6</sup>.

<sup>5</sup> Patrz Terdiman (2012), <http://www.cnet.com/news/report-twitter-hits-half-a-billion-tweets-a-day/>.

<sup>6</sup> Za: Jagadish i in. (2014).

**Szybkość generowania.** Definicja *big data* nie ogranicza się do samej ilości danych. Ważne są też rodzaje i szybkość generowania danych, przez co tradycyjne narzędzia bazodanowe nie potrafią sobie poradzić z takimi danymi. W poświęconym big data raporcie firmy McKinsey<sup>7</sup> szybkość generowania opisano jako tempo, w jakim dane są tworzone, akumulowane, przyjmowane i przetwarzane. Rozważ np. typową szybkość przeprowadzania transakcji na giełdach akcji. W niektóre dni sięga ona miliardów transakcji. Jeśli trzeba przetwarzać takie transakcje w celu wykrywania możliwych oszustw lub gdy konieczne jest analizowanie miliardów rozmów przez telefony komórkowe, aby odkryć jakieś szkodliwe poczynania, mamy do czynienia z wymiarem szybkości generowania danych. Dane generowane w czasie rzeczywistym i dane z transmisji strumieniowych są akumulowane przez firmy takie jak Twitter i Facebook z bardzo dużą szybkością. Szybkość generowania pomaga w wykrywaniu trendów wśród osób, które sumarycznie co trzy minuty publikują miliony tweetów. Jest także istotna w trakcie przetwarzania danych strumieniowych na potrzeby analiz.

**Różnorodność.** W tradycyjnych aplikacjach źródłami danych były przede wszystkim transakcje związane z branżami: finansową, ubezpieczeniową, opieki zdrowotnej, handlową, biurami podróży, a także z przetwarzaniem danych rządowych i sądowych. Obecnie zakres typów źródeł danych bardzo się rozszerzył. Źródłami danych są: internet (np. strumienie kliknięć lub serwisy społecznościowe), badania (np. ankiety i raporty branżowe), usługi lokalizacyjne (np. z urządzeń mobilnych lub dane geoprzestrzenne), obrazy (np. z systemów obserwacji, satelitów i obrazowania medycznego), e-maile, łańcuchy dostaw (np. elektroniczna wymiana danych i katalogi sprzedawców), sygnały (np. z czujników i urządzeń RFID) i filmy (w serwisie YouTube każdej minuty dodawane są setki minut filmów). Big data obejmują dane strukturalne, częściowo strukturalne i niestukturalne (patrz omówienie w rozdziale 26.) w różnych proporcjach i zależnie od kontekstu.

Dane strukturalne są zgodne z formalnie ustrukturyzowanym modelem danych, np. z modelem relacyjnym, w którym dane mają postać tabel zawierających wiersze i kolumny, lub z bazami hierarchicznymi w systemie IMS, gdzie występują typy rekordowe w segmentach i pola w rekordach.

Dane niestukturalne nie mają formalnej struktury. W rozdziale 24. omówiliśmy systemy takie jak MongoDB, który przechowuje niestukturalne dane dokumentowe, i Neo4j, gdzie dane są zapisywane w formie grafów. Inne rodzaje danych niestukturalnych to e-maile, blogi, pliki PDF, nagrania audio i wideo, zdjęcia, strumienie kliknięć i materiały internetowe. Pojawienie się sieci WWW w latach 1993 – 1994 doprowadziło do znacznego wzrostu ilości danych niestukturalnych. Niektóre rodzaje danych niestukturalnych mogą mieć format ze zdefiniowanymi znacznikami rozdzielającymi elementy semantyczne. Taki format czasem umożliwia określanie hierarchii danych. Mechanizmy opisowe w formacie XML pozwalają tworzyć hierarchie, a ponadto pojawiło się wiele odmian formatu XML z różnych dziedzin, np.: w biologii (bioML — ang. *biopolymer markup language*), w systemach informacji geograficznej (gML — ang. *geography markup language*) i w browarnictwie (BeerXML — język do wymiany danych związanych z warzeniem piwa). Dane niestukturalne są największym wyzwaniem w dzisiejszych systemach big data.

---

<sup>7</sup> Patrz McKinsey (2013).

**Wiarygodność.** Wymiar wiarygodności big data jest aspektem nowszym od powstania internetu. Z wiarygodnością nieodłącznie związane są dwie kwestie: rzetelność źródła i dostosowanie danych do docelowych odbiorców. Wymiar ten jest ściśle związany z zaufaniem. Podawanie wiarygodności jako jednego z wymiarów big data oznacza przyznanie, że dane napływające do aplikacji z obszaru big data mają zróżnicowaną wiarygodność. Dlatego przed zaakceptowaniem danych na potrzeby analiz lub innych zastosowań trzeba poddać je testom jakości i badaniom rzetelności. Wiele źródeł generuje niepewne, niekompletne i nieprecyzyjne dane, przez co ich wiarygodność jest wątpliwa.

Teraz skupimy się na technologiach uznawanych za podstawy w obszarze big data. Przewiduje się, że do 2016 r. ponad połowa danych na świecie może być przetwarzana przez technologie związane z Hadoopem. Dlatego ważne jest, aby prześledzić rewolucję związaną z technologiami MapReduce i Hadoop oraz zrozumieć obecne znaczenie obu technologii. Historycznie rozwój tych mechanizmów rozpoczął się od powstania modelu programowania nazywanego MapReduce.

## 25.2. Wprowadzenie do technologii MapReduce i Hadoop

W tym podrozdziale przedstawimy technologię analiz big data i przetwarzania danych o nazwie Hadoop. Jest to otwarta implementacja modelu programowania MapReduce. Dwa podstawowe komponenty Hadoopa to model programowania MapReduce i system HDFS (ang. *Hadoop Distributed File System*). Pokrótkę omówimy tu historię Hadoopa, a następnie modelu MapReduce. Dalej krótko opiszemy środowisko Hadoopa i wersje tego narzędzia.

### 25.2.1. Tło historyczne

Hadoop powstał w wyniku prób zbudowania otwartej wyszukiwarki. Pierwsze kroki zostały poczynione przez ówczesnego dyrektora archiwum internetowego Douga Cuttinga i doktora University of Washington Mike'a Carafellę. Cutting i Carafella opracowali system Nutch, który potrafił przeglądać i indeksować setki milionów stron internetowych. Jest to otwarty projekt organizacji Apache<sup>8</sup>. Po opublikowaniu przez firmę Google artykułu „The Google File System”<sup>9</sup> (październik 2003 r.) i pracy o modelu programowania MapReduce<sup>10</sup> (grudzień 2004 r.) Cutting i Carafella zdali sobie sprawę, że na podstawie pomysłów z tych dwóch tekstów mogą usprawnić szereg elementów. Zbudowali więc system plików i platformę obliczeniową tworzące narzędzie znane później jako Hadoop (używano w nim języka Java zamiast stosowanego w modelu MapReduce C++), po czym dostosowali do niego system Nutch. W 2006 r. Cutting został zatrudniony w Yahoo!, gdzie trwały prace nad otwartymi technologiami opartymi na technikach z systemu Google File System i modelu programowania MapReduce. Firma Yahoo! chciała rozbudować swoją wyszukiwarkę i wypracować

---

<sup>8</sup> Dokumentację systemu Nutch znajdziesz na stronie <http://nutch.apache.org>.

<sup>9</sup> Ghemawat, Gbioff i Leung (2003).

<sup>10</sup> Dean i Ghemawat (2004).

otwartą infrastrukturę na bazie systemu Google File System i modelu MapReduce. Wydzieliła mechanizmy składowania i przetwarzania danych z systemu Nutch, tworząc narzędzie **Hadoop** (nazwane tak po pluszowym słoniu syna Cuttinga). Początkowo Hadoop miał w wysoce skalowalny sposób wykonywać zadania wsadowe. Jednak ok. 2006 r. potrafił działać w tylko niewielkiej liczbie węzłów. Później Yahoo! założyła forum badawcze dla zatrudnionych w niej badaczy danych (ang. *data scientists*). Doprowadziło to do zwiększenia adekwatności wyników wyszukiwania i dochodów z reklam w wyszukiwarce, a jednocześnie pomogło dojrzeć technologii Hadoop. W 2011 r. Yahoo! wydzieliła jednostkę odpowiedzialną za Hadoopa — Hortonworks. Na tym etapie infrastruktura firmy obejmowała setki petabajtów pamięci i 42 000 węzłów w klastrze. Odkąd Hadoop stał się otwartym projektem rozwijanym przez organizację Apache, tysiące programistów z całego świata wniosło wkład w jego rozwój. We wspólnym projekcie firm Google i IBM oraz organizacji NSF używano klastra z Hadoopem obejmującego 2000 węzłów. Klaster ten działał w centrum danych w Seattle i pomógł w badaniach uniwersyteckich nad Hadoopem. Narzędzie to zostało znacznie rozwinięte od czasu powstania w 2008 r. pierwszej komercyjnej firmy związanej z Hadoopem, Cloudera, i późniejszego wysypu wielu podobnych start-upów. IDC, firma analityczna badająca rynek oprogramowania, prognozowała, że w 2016 r. wartość rynku związanego z Hadoopem przekroczy 800 milionów dolarów. Według prognoz tej samej firmy wartość rynku big data w 2016 r. miała przekroczyć 23 miliardy dolarów. Więcej szczegółów na temat historii Hadoopa znajdziesz w czteroczęściowym artykule Harris<sup>11</sup>.

Integralną częścią Hadoopa jest model programowania MapReduce. Zanim przejdziemy dalej, warto spróbować zrozumieć, na czym polega ten model. Szczegółowe omawianie systemu plików HDFS odkładamy do podrozdziału 25.3.

### 25.2.2. Model MapReduce

Model programowania i środowisko uruchomieniowe MapReduce zostało opisane po raz pierwszy przez Jeffreya Deana i Sanjaya Ghemawata (2004) na podstawie ich pracy w firmie Google. Użytkownicy piszą programy w stylu funkcyjnym z zadaniami **mapowania** (ang. *map*) i **redukcji** (ang. *reduce*), po czym programy te są automatycznie przekształcane na postać równoległą i wykonywane w dużych klastrach ze standardowym sprzętem. Ten model programowania istnieje od czasów języka LISP zaprojektowanego przez Johna McCarthy'ego pod koniec lat 50. Jednak „nowe wcielenie” tego podejścia do programowania równoległego i sposób implementacji go w firmie Google były początkiem nowoczesnego myślenia, które doprowadziło do późniejszego powstania technologii takich jak Hadoop. Ten system uruchomieniowy obsługuje wiele skomplikowanych inżynierskich aspektów równoległego wykonywania zadań, odporności na błędy, rozdzielania danych, równoważenia obciążenia i zarządzania komunikacją między zadaniami. Dopóki użytkownicy przestrzegają **kontraktów** określonych w systemie MapReduce, mogą się skupić na logicznych aspektach programu. Pozwala to programistom bez doświadczenia w korzystaniu z systemów rozproszonych przeprowadzać analizy na bardzo dużych zbiorach danych.

---

<sup>11</sup> Derreck Harris, „The history of Hadoop: from 4 nodes to the future of data”, <https://gigaom.com/2013/03/04/the-history-of-hadoop-from-4-nodes-to-the-future-of-data/>.

Motywacją do powstania systemu MapReduce były lata poświęcone przez jego twórców i inne osoby z Google'a na implementowanie setek specjalnych obliczeń na dużych zbiorach danych. Te obliczenia dotyczyły np. tworzenia indeksów odwróconych na podstawie materiałów internetowych uzyskanych w wyniku crawlingu, budowania grafów stron internetowych i generowania na podstawie dzienników internetowych statystyk takich jak rozkład częstotliwości wyszukiwań według tematów, regionów, typów użytkowników itd. Konceptyjnie zdefiniowanie tych zadań nie jest trudne, jednak z powodu skali na poziomie miliardów stron internetowych i rozproszenia danych na tysiącach maszyn wykonywanie tych zadań nie było proste. Niezwykle istotne stały się kwestie sterowania programem, zarządzania danymi, rozdzielania danych, przekształcania obliczeń na postać równoległą i obsługi awarii.

Model programowania i środowisko uruchomieniowe MapReduce zaprojektowano po to, aby radzić sobie z opisaną złożonością. Inspiracją do powstania tego abstrakcyjnego rozwiązania były proste mechanizmy mapowania i redukcji dostępne w LISP-ie i wielu innych językach funkcyjnych. Zakłada się tu istnienie określonego modelu danych, w którym obiekt jest traktowany jak unikatowy klucz powiązany z treścią lub wartością (mamy więc parę klucz-wartość). Zaskakujące jest to, że wiele obliczeń można wyrazić jako zastosowanie operacji mapowania do każdego logicznego „rekordu”, co daje zbiór pośrednich par klucz-wartość, a następnie użycie operacji redukcji do wszystkich wartości o tym samym kluczu (wspólne klucze pozwalają łączyć uzyskane dane). Ten model umożliwia infrastrukturze łatwe przekształcanie na postać równoległą rozbudowanych obliczeń i stosowanie ponownego wykonywania jako podstawowego mechanizmu zapewniania odporności na błędy. Pomysł tworzenia ograniczonego modelu programowania, aby środowisko uruchomieniowe mogło automatycznie przekształcać obliczenia na postać równoległą, nie jest nowy. MapReduce to rozszerzenie istniejących pomysłów. W dzisiejszej postaci MapReduce jest implementacją odporną na błędy i środowiskiem uruchomieniowym skalującym się do poziomu tysięcy procesorów. Programista nie musi się martwić obsługą awarii. W dalszych punktach często będziemy skracać nazwę Map ↪ Reduce do postaci **MR**.

**Model programowania MapReduce.** W dalszym opisie posłużymy się formalnymi wywodami i definicjami z pierwotnej pracy Deana i Ghemawata (2010)<sup>12</sup>. Funkcje mapowania (`map`) i redukcji (`reduce`) mają następującą ogólną postać:

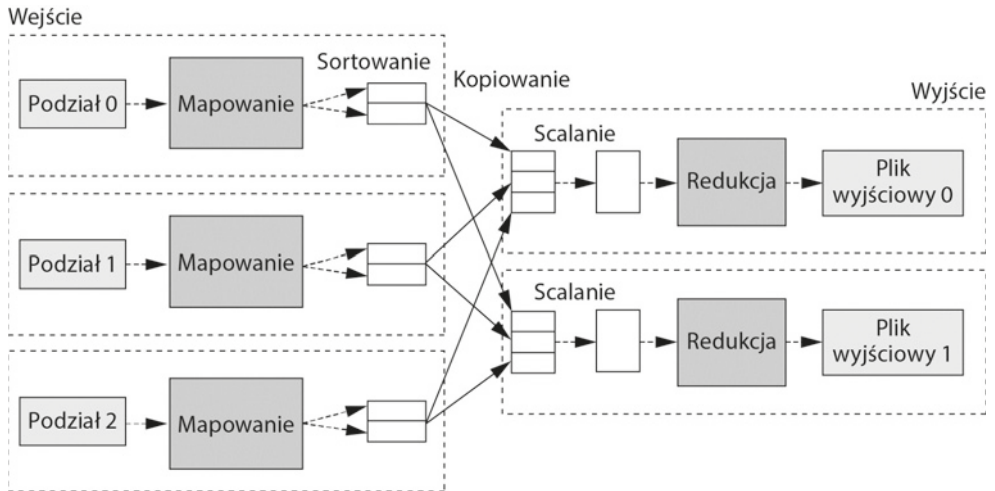
`map[K1,V1]`, czyli (klucz, wartość) : `List[K2,V2]` oraz  
`reduce(K2, List[V2])` : `List[K3,V3]`

`map` to funkcja generyczna przyjmująca klucz typu `K1` i wartość typu `V1` oraz zwracająca listę par klucz-wartość typu `K2` i `V2`. `reduce` to funkcja generyczna przyjmująca klucz typu `K2` i listę wartości typu `V2` oraz zwracająca parę typu `(K3, V3)`. Zwykle typy `K1`, `K2` i `K3` są różne. Jedyń wymóg dotyczy tego, by typy wyjściowe funkcji `map` pasowały do typów wejściowych funkcji `reduce`.

Podstawowy przepływ pracy w modelu MapReduce jest pokazany na rysunku 25.1.

---

<sup>12</sup> Jeffrey Dean i Sanjay Ghemawat, „MapReduce: Simplified Data Processing on Large Clusters”, w: OSDI (2004).



RYSUNEK 25.1. Przegląd wykonywania zadań w MapReduce (zaadaptowane z: T. White, 2012)

Żałujemy, że chcemy uzyskać listę słów z dokumentu wraz z częstotliwością ich występowania. Wszelchobecnny przykład *zliczania słów*, zaczerpnięty bezpośrednio z publikacji Deana i Ghemawata (2004), zapisany w pseudokodzie wygląda tak:

```
map (String key, String value):
    dla każdego słowa w z value wygeneruj dane pośrednie (w, "1");
```

Tu key to nazwa dokumentu, a value to jego tekstowa zawartość.

Następnie listy par (słowo, 1) są dodawane w celu zwrócenia łącznych liczb wszystkich słów znalezionych w dokumencie. Odbywa się to tak:

```
reduce (String key, Iterator values) :
    // key to słowo, a values to lista liczb wystąpień //
    Int result = 0;
    For each v in values :
        result += Parse int (v);
    Emit (key, As string (result));
```

Ten przykład napisany w modelu MapReduce wygląda tak:

```
map[LongWritable,Text](key, value) : List[Text, LongWritable] = {
    String[] words = split(value)
    for(word : words) {
        context.out(Text(word), LongWritable(1))
    }
}
reduce[Text, Iterable[LongWritable]](key, values) : List[Text, LongWritable] =
{
    LongWritable c = 0
    for( v : values) {
        c += v
    }
    context.out(key,c)
}
```



Typy danych użyte w tym przykładzie to `LongWritable` i `Text`. Dla każdej pracy w modelu MapReduce trzeba zarejestrować funkcje `map` i `reduce`. Funkcja `map` przyjmuje każdą parę klucz-wartość i dla każdego wywołania może zwrócić 0 lub więcej par klucz-wartość. W sygnaturze funkcji `map` określone są typy danych wejściowych i wyjściowych par klucz-wartość. Funkcja `reduce` przyjmuje klucz i iterator wartości powiązanych z tym kluczem. Każde wywołanie tej funkcji może zwrócić jedną lub więcej par klucz-wartość. Sygnatura funkcji `reduce` określa typy danych wejściowych i wyjściowych. Typ danych wyjściowych funkcji `map` musi pasować do typu danych wejściowych funkcji `reduce`. W przykładzie ze zliczaniem słów funkcja `map` pobiera jako wartość każdy wiersz, dzieli go na słowa i generuje (za pomocą funkcji `context.out`) dla każdego słowa wiersz z liczbą wystąpień równą 1. Każde wywołanie funkcji `reduce` otrzymuje dla danego słowa listę liczb wystąpień obliczoną na etapie mapowania. Następnie funkcja dodaje te liczby i zwraca jako dane wyjściowe każde słowo oraz liczbę jego wystąpień. Te funkcje komunikują się z *kontekstem*. Kontekst służy do interakcji z modelem. Jest używany przez klienty do przesyłania do zadań informacji o konfiguracji. Zadania mogą wykorzystać kontekst do uzyskania dostępu do systemu HDFS i wczytywania danych bezpośrednio z niego, do zwracania par klucz-wartość oraz do przesyłania informacji o statusie (np. liczników z zadań) z powrotem do klienta.

Oto oparty na modelu MapReduce sposób implementacji innych funkcji (na podstawie publikacji Deana i Ghemawata, 2004):

### Narzędzie `grep` w środowisku rozproszonym

Narzędzie `grep` wyszukuje wzorzec w pliku. Funkcja `map` zwraca wiersz, jeśli pasuje on do podanego wzorca. Funkcja `reduce` jest tu funkcją tożsamościową, która kopiuje otrzymane dane pośrednie do wyjścia. Jest to przykład *zadania obejmującego tylko mapowanie*. Nie trzeba tu ponosić kosztów **przestawiania** (ang. *shuffle*). Więcej informacji na ten temat przekażemy w omówieniu środowiska uruchomieniowego MapReduce.

### Odwrócony graf odsyłaczy internetowych

Celem jest zwrócenie par (docelowy adres URL, źródłowy adres URL) dla każdego odsyłacza do strony docelowej znalezionej na stronie źródłowej. Funkcja `reduce` łączy listę wszystkich źródłowych adresów URL powiązanych z danym docelowym adresem URL i zwraca parę <adres docelowy, lista(adresy źródłowe)>.

### Indeks odwrócony

Celem jest zbudowanie indeksu odwróconego opartego na wszystkich słowach znajdujących się w repozytorium dokumentów. Funkcja `map` przetwarza każdy dokument i zwraca sekwencję par (słowo, identyfikator dokumentu). Funkcja `reduce` przyjmuje wszystkie pary z danym słowem, sortuje je według identyfikatorów dokumentów i zwraca parę (słowo, lista(identyfikatory dokumentów)). Zbiór wszystkich takich par tworzy indeks odwrócony.

Te przykładowe techniki pozwalają dostrzec bogaty zestaw zastosowań modelu programowania MapReduce i łatwość wyrażania logiki aplikacji za pomocą etapów mapowania i redukcji.



**Praca** (ang. *job*) w modelu MapReduce obejmuje kod etapów mapowania i redukcji (zwykle w pakiecie *.jar*), zestaw artefaktów potrzebnych do wykonania zadania (np.: plików, innych pakietów *.jar* i archiwów) i, co najważniejsze, zbiór właściwości określonych w konfiguracji. Można podać setki właściwości. Oto podstawowe z nich:

- zadanie mapowania,
- zadanie redukcji,
- dane wejściowe, na których ma operować praca (zwykle podaje się je jako ścieżki do systemu HDFS),
- format (struktura) danych wejściowych,
- ścieżka danych wyjściowych,
- struktura danych wyjściowych,
- poziom równoległości na etapie redukcji.

Praca jest przesyłana do narzędzia **JobTracker**, które szereguje pracę i zarządza jej wykonywaniem. Wspomniane narzędzie udostępnia zestaw interfejsów do monitorowania działających prac. Więcej szczegółów na temat funkcjonowania narzędzia JobTracker znajdziesz w serwisie Hadoop Wiki<sup>13</sup>.

### 25.2.3. Wersje Hadoopa

Od czasu powstania Hadoopa jako nowej rozproszonej platformy do uruchamiania programów w modelu MapReduce powstały różne wersje tego narzędzia.

Wersje 1.x Hadoopa są kontynuacją pierwotnego kodu bazowego z wersji 0.20. W podwersjach z tej rodziny wprowadzono zabezpieczenia, dodatkowe usprawnienia systemu HDFS i modelu MapReduce w celu obsługi systemu HBase, lepszy model programowania MR, a także inne ulepszenia.

W wersjach 2.x znalazły się następujące ważne funkcje:

- YARN (ang. *Yet Another Resource Negotiator*), czyli ogólny menedżer zasobów wyodrębniony z narzędzia JobTracker z pierwszej wersji MR.
- Nowe środowisko uruchomieniowe dla MR oparte na platformie YARN.
- Usprawniony system HDFS obsługujący federacje i zapewniający wyższą dostępność.

W czasie gdy powstaje ta książka, Hadoop 2.0 jest dostępny od ok. roku. Liczba instalacji tej wersji szybko rośnie, jednak znaczny odsetek użytkowników nadal korzysta z wersji 1.

## 25.3. System HDFS

Jak wcześniej wspomniano, innym podstawowym komponentem Hadoopa (obok modelu MapReduce) jest system plików HDFS. W tym podrozdziale najpierw objaśnimy architekturę tego systemu, następnie opiszemy operacje wejścia-wyjścia na plikach obsługiwane przez ten system, a na końcu omówimy jego skalowalność.

---

<sup>13</sup> Adres serwisu Hadoop Wiki to <http://hadoop.apache.org/>.

### 25.3.1. Wymagania wstępne związane z systemem HDFS

HDFS to używany w Hadoopie system plików, zaprojektowany pod kątem pracy w klastrze standardowego sprzętu. Jest wzorowany na systemie plików UNIX-a, jednak rozluźniono w nim kilka wymagań interfejsu POSIX (ang. *Portable Operating System Interface for UNIX*), aby umożliwić strumieniowy dostęp do danych. HDFS zapewnia wysoką przepustowość dostępu do dużych zbiorów danych i przechowuje osobno metadane systemu plików oraz dane aplikacji. Metadane są przechowywane na specjalnym serwerze, nazywanym Name Node, a dane aplikacji umieszcza się na innych serwerach (węzłach DataNode). Wszystkie serwery są ze sobą połączone i komunikują się za pomocą protokołów opartych na TCP. Aby dane były trwałe, zawartość plików jest replikowana w wielu serwerach DataNode — podobnie jak w systemie Google File System. Nie tylko zwiększa to stabilność, ale też zwielokrotnia przepustowość na potrzeby transferu danych i umożliwia przeprowadzanie obliczeń tam, gdzie znajdują się dane. HDFS został zaprojektowany z myślą o następujących założeniach i celach:

**Awarie sprzętowe.** Gdy używany jest standardowy sprzęt, awarie są normą, a nie wyjątkiem. Dlatego gdy używane są tysiące węzłów, automatyczne wykrywanie awarii i odtwarzanie stanu to konieczność.

**Przetwarzanie wsadowe.** HDFS został zaprojektowany przede wszystkim na potrzeby przetwarzania wsadowego, a nie użytkowania interaktywnego. Nacisk jest położony na wysoką przepustowość, a nie na niskie opóźnienie dostępu do danych. Typowe jest pełne skanowanie plików.

**Duże zbiory danych.** HDFS zaprojektowano pod kątem obsługi bardzo dużych plików, zajmujących od setek gigabajtów do terabajtów miejsca.

**Prosty model zachowywania spójności.** W aplikacjach używających HDFS dostęp do plików z modelu może uzyskać jeden proces zapisujący i wiele procesów odczytujących. Zawartości plików nie można aktualizować — dane można tylko dodawać do plików. Ten model łagodzi problemy z zachowaniem spójności między kopiami danych.

### 25.3.2. Architektura systemu HDFS

W systemie HDFS używana jest architektura nadrzędny-podrzędny. Serwer nadrzędny, **NameNode**, zarządza obszarem składowania danych systemu plików (przestrzenią nazw). Klienci używają przestrzeni nazw za pośrednictwem serwera NameNode. Serwery podrzędne, nazywane **DataNode**, działają w klastrze standardowych maszyn (zwykle po jednym na maszynę). Zarządzają magazynem danych podłączonym do węzła, w którym pracują. Przestrzeń nazw obejmuje pliki i katalogi. Serwer NameNode przechowuje węzły indeksowe (i-węzły; ang. *i-node*) z informacjami o plikach i katalogach, z którymi powiązane są atrybuty takie jak właściciel, uprawnienia, czas utworzenia, czas dostępu i ilość miejsca na dysku. Za pomocą i-węzłów bloki plików są odwzorowywane na serwery DataNode. Te ostatnie odpowiadają za obsługę żądań odczytu i zapisu otrzymywanych od klientów. Serwery DataNode tworzą, usuwają i replikują bloki zgodnie z instrukcjami od serwera Name Node. Kłaster może obejmować tysiące serwerów DataNode i dziesiątki tysięcy jednocześnie podłączonych klientów systemu HDFS.

W celu wczytania pliku klient najpierw łączy się z serwerem NameNode i pobiera lokalizacje bloków danych z pliku, do którego chce uzyskać dostęp. Następnie łączy się bezpośrednio z serwerami DataNode, które zawierają te bloki, i wczytuje dane.

Oto ważne cechy architektury systemu HDFS:

- (1) HDFS umożliwia oddzielenie operacji na metadanych od operacji na danych. Operacje na metadanych są wykonywane szybko, natomiast transfer danych jest dużo wolniejszy. Jeśli lokalizacja metadanych i transfer danych nie są od siebie oddzielone, szybkość w środowisku rozproszonym spada, ponieważ czas odpowiedzi zależy od transferu danych i wydłuża się.
- (2) Stosowana jest replikacja w celu zapewnienia stabilności i wysokiej dostępności. Każdy blok jest replikowany do kilku węzłów klastra (domyślnie używane są trzy kopie). Pliki używane przez wiele jednostek, np. biblioteki prac w modelu MapReduce, mają więcej replik, co pozwala ograniczyć ruch w sieci.
- (3) Ruch w sieci jest ograniczany do minimum. Przy odczycie klienty są kierowane do najbliższego serwera DataNode. O ile to możliwe, dane są wczytywane z lokalnego systemu plików, bez transferu danych w sieci. Następnym wyborem jest użycie kopii z węzła z tej samej szafy serwerowej. Dopiero potem system sprawdza inne szafy serwerowe. Przy zapisie pierwsza kopia jest umieszczana w tym samym węźle, w którym działa klient; celem jest tu ograniczenie transferu w sieci. Jeśli chodzi o inne kopie, minimalizowana jest ilość danych przesyłanych między szafami serwerowymi.

**NameNode.** NameNode przechowuje **obraz** systemu plików składający się z i-węzłów i powiązanych z nimi lokalizacji bloków. Zmiany w systemie plików są zapisywane w dzienniku rejestrowania zapisów z wyprzedzeniem (patrz omówienie rejestrowania zapisów z wyprzedzeniem w rozdziale 22.) — w **dzienniku zmian** (ang. *journal*). Na potrzeby odtwarzania danych tworzone są punkty kontrolne. Reprezentują one trwały zapis obrazu bez dynamicznych informacji związanych z lokalizacją bloków. Te ostatnie informacje są okresowo pobierane z serwerów DataNode w opisany dalej sposób. W trakcie wznowiania pracy obraz jest przywracany do ostatniego punktu kontrolnego, po czym stosowane są do niego wpisy z dziennika zmian. Następnie tworzone są nowy punkt kontrolny i pusty dziennik zmian, aby serwer NameNode mógł rozpocząć przyjmowanie nowych żądań od klientów. Czas rozruchu serwera NameNode jest proporcjonalny do wielkości pliku dziennika Journal. Okresowe scalanie punktu kontrolnego z dziennikiem zmian skraca czas wznowiania pracy.

Zauważ, że w tej architekturze uszkodzenie punktu kontrolnego lub dziennika zmian oznacza katastrofę. Aby zabezpieczyć się przed uszkodzeniem takich danych, zapisuje się je w różnych katalogach na różnych woluminach.

**Pomocnicze serwery NameNode.** Są to dodatkowe serwery NameNode, które można utworzyć na potrzeby zapisywania punktów kontrolnych lub jako serwer rezerwowy. Serwer odpowiedzialny za punkt kontrolny okresowo łączy istniejące punkty kontrolne i pliki dziennika zmian. Serwer w trybie archiwizacji działa jak dodatkowe miejsce zapisu dziennika zmian z głównego serwera NameNode. Serwer rezerwowy jest aktualny względem systemu plików i może przejąć rolę serwera głównego po awarii tego ostatniego. W Hadoopie 1 zmiana serwera głównego odbywa się ręcznie.

**Serwery DataNode.** Bloki są przechowywane w serwerach DataNode w natywnym systemie plików tych serwerów. Serwer NameNode kieruje klienty do serwerów DataNode zawierających kopię bloku, którą dany klient chce wczytać. Każdy blok jest reprezentowany w natywnym systemie plików za pomocą dwóch plików. Jeden zawiera dane, a drugi meta-dane, w tym sumy kontrolne danych z bloku i czas wygenerowania bloku. Serwery DataNode i NameNode nie komunikują się bezpośrednio, tylko za pomocą **sygnałów kontrolnych** (ang. *heartbeat mechanism*), co polega na okresowym informowaniu serwera NameNode o stanie serwera DataNode. Te informacje mają postać raportu o blokach. Ten raport obejmuje identyfikator, znacznik czasu wygenerowania i długość każdego bloku. Obraz przestrzeni nazw nie obejmuje lokalizacji bloków. Lokalizacje trzeba pobierać z raportów o blokach. Lokalizacje zmieniają się wraz z przenoszeniem bloków. Narzędzie JobTracker z modelu MapReduce wraz z serwerem NameNode używają informacji z najnowszego raportu o blokach do szeregowania zadań. W odpowiedzi na sygnał kontrolny z serwera DataNode serwer NameNode przesyła do serwera DataNode polecenia następujących typów:

- replikacji bloku na innym serwerze;
- usunięcia repliki bloku;
- ponownego zarejestrowania lub zamknięcia serwera;
- natychmiastowego przesłania raportu o bloku.

### 25.3.3. Operacje wejścia-wyjścia na plikach i zarządzanie replikami w systemie HDFS

System HDFS obsługuje model z jednym procesem zapisującym i wieloma procesami odczytującymi. Plików nie można aktualizować — dozwolone jest tylko dodawanie do nich danych. Plik składa się z bloków. Dane są zapisywane w 64-kilobajtowych pakietach w **potoku zapisu**, co ma minimalizować obciążenie sieci. Dane zapisane w ostatnim bloku stają się dostępne dopiero po bezpośrednim wykonaniu operacji `hflush`. W trakcie zapisu danych możliwy jest jednoczesny ich odczyt przez klienty. Dla każdego bloku generowana i zapisywana jest suma kontrolna. Klient sprawdza ją, aby wykryć uszkodzenie danych. Po wykryciu uszkodzonego bloku powiadamiany jest o tym serwer NameNode, który inicjuje proces replikacji bloku i nakazuje serwerowi DataNode usunięcie niepoprawnego bloku. W trakcie wykonywania operacji odczytu następuje próba pobrania repliki z możliwie bliskiego serwera. Serwery są w tym celu porządkowane rosnąco według odległości od klienta. Odczyt kończy się niepowodzeniem, gdy serwer DataNode jest niedostępny, gdy suma kontrolna jest nieprawidłowa lub gdy replika nie znajduje się już na danym serwerze DataNode. System HDFS został zoptymalizowany pod kątem przetwarzania wsadowego w sposób podobny jak model MapReduce.

**Rozmieszczanie bloków.** Serwery w klastrze Hadoopa są zwykle rozproszone po wielu szafach serwerowych. Przeważnie są uporządkowane w taki sposób, że serwery z jednej szafy są podłączone do tego samego przełącznika, a przełączniki z szaf są podłączone do szybkiego przełącznika wyższego poziomu. Przykładowo, na poziomie szafy używany może być przełącznik o szybkości 1 Gb, a na wyższym poziomie — przełącznik o szybkości 10 Gb. System HDFS szacuje przepustowość łącza między serwerami DataNode na podstawie odle-

głości między nimi. Odległość między serwerami DataNode z tego samego fizycznego węzła wynosi 0, serwery z tej samej szafy znajdują się w odległości 2, a serwery z różnych szaf — w odległości 4. Domyślna strategia rozmieszczania bloków w systemie HDFS równoważy minimalizowanie kosztów zapisu oraz maksymalizowanie stabilności i dostępności danych, a także dostępną przepustowość przy odczytach. Wykorzystana przepustowość sieci jest szacowana na podstawie odległości między serwerami DataNode. Ostatecznym celem rozmieszczenia bloków jest minimalizacja kosztów zapisu przy maksymalizacji dostępności i stabilności danych oraz dostępnej przepustowości na potrzeby odczytów. Replikami zarządza się w ten sposób, aby przynajmniej jedna znajdowała się w węźle, gdzie działa klient, który utworzył dane. Pozostałe repliki są rozproszone w innych szafach serwerowych. Preferowane jest wykonywanie zadań na serwerach, w których znajdują się dane. Trzy repliki zapewniają programowi szeregującemu wystarczającą swobodę, aby mógł wykonywać zadania tam, gdzie znajdują się dane.

**Zarządzanie replikami.** Na podstawie raportów o blokach z serwera DataNode serwer NameNode śledzi liczbę replik i lokalizację każdego bloku. Kolejka replikacji (uwzględniająca priorytety) obejmuje bloki wymagające replikacji. Działający w tle wątek obserwuje tę kolejkę i nakazuje serwerom DataNode tworzenie replik i rozsyłanie ich między szafy serwerowe. Serwer NameNode preferuje przechowywanie replik bloku w możliwie dużej liczbie różnych szaf serwerowych. Nadmierna liczba replik bloku powoduje, że niektóre z nich są usuwane. Proces ten odbywa się na podstawie wykorzystania przestrzeni w serwerach DataNode.

### 25.3.4. Skalowalność systemu HDFS

Ponieważ w tym rozdziale omawiamy technologie z obszaru big data, uzasadnione jest omówienie granic skalowalności w systemie HDFS. Shvachko, członek grupy zarządzania programem Hadoop, napisał, że w klastrze z systemem HDFS w firmie Yahoo! osiągnięto podane dalej poziomy, inne od docelowych wartości (Shvachko, 2010). Liczby w nawiasach to docelowe wartości podane przez Shvachkę. Pojemność: 14 petabajtów (docelowe 10 petabajtów); liczba węzłów: 4000 (docelowe 10 000); klienci: 15 000 (docelowe 100 000); pliki: 60 milionów (docelowe 100 milionów). Tak więc w 2010 r. firma zbliżyła się do docelowych wartości, choć jej klastr był mniejszy (4000 węzłów), podobnie jak liczba klientów. Yahoo! przekroczyła jednak docelowy poziom łącznej ilości przechowywanych danych.

Warto wspomnieć o niektórych spostrzeżeniach Shvachki (2010). Są one oparte na konfiguracji systemu HDFS używanej w Yahoo! w 2010 r. Poniżej prezentujemy rzeczywiste i szacunkowe wartości, aby pomóc czytelnikom zrozumieć aspekty związane z tak ogromnymi środowiskami przetwarzania danych.

- Wielkość bloku wynosiła 128 KB, a pliki średnio zajmowały 1,5 bloku. Serwer NameNode używał ok. 200 bajtów na blok i dodatkowych 200 bajtów na i-węzeł. 100 milionów plików przekładające się na referencje do 200 milionów bloków wymagało ponad 60 GB pamięci RAM.
- 100 milionów plików w 200 milionach bloków przy współczynniku replikacji równym 3 wymagało 60 PB pamięci dyskowej. Zaproponowano więc ogólną regułę, zgodnie z którą 1 GB pamięci RAM z serwera NameNode odpowiadał ok. 1 PB pamięci dyskowej (przy założeniu, że wielkość bloku wynosi 128 KB, a pliki zajmują 1,5 bloku).

- Aby zmieścić 60 PB danych w klastrze o 10 000 węzłów, każdy węzeł musiał mieć pojemność 6 TB. Można ją uzyskać za pomocą 8 dysków o pojemności 0,75 TB.
- Wewnętrzne obciążenie serwera NameNode było związane z obsługą raportów o blokach. Serwer NameNode otrzymywał na sekundę ok. 3 raportów zawierających informacje o 60 000 bloków na raport.
- Zewnętrzne obciążenie serwera NameNode było związane z zewnętrznymi połączeniami i zadaniami z prac z modelu MapReduce. Wymagało to dziesiątek tysięcy jednoczesnych połączeń.
- Odczyt danych przez klienta polegał na sprawdzaniu bloku w celu uzyskania jego lokalizacji od serwera NameNode, po czym klient używał najbliższej repliki bloku. Typowy klient (mapowanie w zadaniu w modelu MapReduce) wczytywał dane z 1000 plików, średnio pobierając połowę każdego z nich, co w sumie dawało 96 MB danych. Szacunkowo zajmowało to 1,45 sekundy. Przy tej szybkości 100 000 klientów przesyłało na serwer NameNode 68 750 żądań podania lokalizacji bloku na sekundę. Leżało to w zasięgu możliwości serwera NameNode, które szacowano na 126 000 żądań na sekundę.
- Obciążenie robocze związane z zapisem: przy przepustowości zapisu 40 MB/sekundę klient średnio zapisywał 96 MB w 2,4 sekundy. To daje ponad 41 000 żądań „utwórz blok” do serwera NameNode (przy 100 000 węzłów). Znacznie przekraczało to możliwości serwera NameNode.

W tych analizach zakładano, że w każdym węźle działa tylko jedno zadanie. W praktyce w rzeczywistym systemie w każdym węźle mogło działać wiele zadań; w Yahoo! w węzłach wykonywane były po cztery zadania w modelu MapReduce. Spowodowało to powstanie wąskiego gardła na poziomie serwera NameNode. Problemy tego rodzaju zostały rozwiązane w Hadoopie 2, którą to wersję opiszemy w następnym podrozdziale.

## 25.3.5. Ekosystem Hadoopa

Hadoop jest najbardziej znany z modelu programowania MapReduce, infrastruktury uruchomieniowej i systemu HDFS. Jednak ekosystem Hadoopa obejmuje zestaw powiązanych projektów, które zapewniają dodatkowe funkcje obok wymienionych podstawowych mechanizmów. Wiele tych rozwiązań to głównie otwarte projekty rozwijane przez organizację Apache i bardzo liczne grupy kontrybutorów. Oto kilka ważnych projektów tego rodzaju:

**Pig i Hive.** Zapewniają wysokopoziomowy interfejs do pracy z platformą Hadoop.

- Pig dostarcza język przepływu danych. Skrypt napisany w języku PigScript jest przekształcany na skierowany graf acykliczny z pracami z modelu MapReduce.
- Hive udostępnia interfejs języka SQL bazujący na modelu MapReduce. SQL obsługiwany w narzędziu Hive obejmuje większość funkcji ze standardu SQL-92 i wiele zaawansowanych mechanizmów analitycznych z nowszych standardów tego języka. W narzędziu Hive zdefiniowany jest też abstrakcyjny mechanizm SerDe (ang. *serialization/deserialization*), określający sposób modelowania struktury rekordów w zbiorach danych w systemie HDFS na poziomie wykraczającym poza pary klucz-wartość. Oba te narzędzia opiszemy szczegółowo w punkcie 25.4.4.



**Oozie.** Jest to usługa szeregująca i wykonująca przepływy prac (poszczególnymi krokami mogą być prace w modelu MapReduce, zapytania narzędzia Hive, skrypty narzędzia Pig itd.).

**Sqoop.** Jest to biblioteka i środowisko uruchomieniowe służące do wydajnego przenoszenia danych między bazami relacyjnymi a systemem HDFS.

**HBase.** Jest to kolumnowy magazyn danych z parami klucz-wartość używający systemu HDFS. Szczegółowe omówienie systemu HBase znajdziesz w rozdziale 24. HBase obsługuje przetwarzanie wsadowe w modelu MapReduce i wyszukiwanie oparte na kluczach. Dzięki odpowiedniemu zaprojektowaniu schematu klucz-wartość można wykorzystać HBase w różnorodnych aplikacjach — np.: do analizy szeregów czasowych, w hurtowniach danych, do generowania kostek, przeszukiwania danych wielowymiarowych lub strumieniowania danych.

## 25.4. Model MapReduce: dodatkowe szczegóły

W punkcie 25.2.2 wprowadziliśmy model MapReduce. Teraz omówimy go dokładniej w kontekście środowiska uruchomieniowego. Wyjaśnimy, jak wykonywać w nim relacyjne złączenia. Przyjrzyjmy się też wysokopoziomowym interfejsom Pig i Hive. Na zakończenie opiszemy zalety połączenia modelu MapReduce z Hadoopem.

### 25.4.1. Środowisko uruchomieniowe MapReduce

W tym punkcie chcemy ogólnie przedstawić środowisko uruchomieniowe MapReduce. Szczegółowy opis czytelnicy znajdą w pracy White'a (2012). MapReduce to system typu nadrzędny-podrzędny, zwykle działający w tym samym klastrze co HDFS. Średnie i duże klastry Hadoopa mają zazwyczaj architekturę dwu- lub trzywarstwową i są zbudowane z serwerów montowanych w szafach serwerowych.

**JobTracker.** Proces nadrzędny to *JobTracker*. Zarządza on cyklem życia prac i szeregowaniem zadań w klastrze. Jest odpowiedzialny za:

- Przesyłanie prac, inicjowanie prac, przekazywanie statusu i stanu prac do klientów i procesów podrzędnych (*TaskTrackers*) oraz za kończenie prac.
- Szeregowanie zadań mapowania i redukcji w klastrze. Odbywa się to za pomocą konfigurowalnego programu szeregującego.

**TaskTracker.** Proces podrzędny to *TaskTracker*. Ten proces działa w każdym **węźle roboczym** klastra. W takich węzłach uruchamiane są zadania mapowania i redukcji. Demony procesu TaskTracker działające w węzłach rejestrują się w momencie rozruchu w procesie JobTracker, a następnie wykonują zadania przypisane im przez proces JobTracker. Zadania są wykonywane w odrębnym procesie węzła. Cyklem życia tego procesu zarządza właśnie TaskTracker. Tworzy on proces zadania, śledzi jego wykonywanie, przesyła okresowo sygnały kontrolne do procesu JobTracker, a po awarii może zamknąć proces zadania na żądanie procesu JobTracker. TaskTracker świadczy usługi zadaniom. Najważniejszą z tych usług jest **przestawianie**, opisane w podpunkcie dalej.



## A. Ogólny przebieg prac w modelu MapReduce.

Praca w modelu MapReduce przechodzi przez kroki przesyłania pracy, inicjowania pracy, przypisywania zadań, wykonywania zadań i kończenia pracy. Uczestniczą w tym opisane wcześniej procesy JobTracker i TaskTracker. Dalej pokrótce omawiamy te kroki.

**Przesyłanie pracy.** Klient przesyła pracę do procesu **JobTracker**. Pakiet z pracą obejmuje pliki wykonywalne (w archiwum *.jar*) i inne komponenty (pliki, archiwa *.jar*) potrzebne do wykonania pracy, a także porcje danych wejściowych dla określonej pracy.

**Inicjowanie pracy.** Proces JobTracker przyjmuje pracę i umieszcza ją w kolejce prac. Na podstawie porcji danych wejściowych tworzy zadania mapowania dla każdej z tych porcji. Zgodnie z konfiguracją pracy tworzona jest określona liczba zadań redukcji.

**Przypisywanie zadań.** Program szeregujący w procesie JobTracker przypisuje do procesu TaskTracker zadanie z jednej z wykonywanych prac. W Hadoopie 1 proces TaskTracker obsługiwał stałą liczbę slotów dla zadań mapowania i redukcji. Program szeregujący w trakcie szeregowania zadań w węzłach klastra uwzględnia informacje o lokalizacji plików wejściowych.

**Wykonywanie zadań.** Po zaszeregowaniu zadania w slotcie TaskTracker zarządza jego wykonywaniem — udostępnia wszystkie artefakty procesowi zadania, uruchamia maszynę JVM zadania, śledzi proces zadania i koordynuje działanie z procesem JobTracker w kontekście operacji administracyjnych (np. porządkowania zasobów po zakończeniu zadania i zamykania zadań po wystąpieniu błędu). TaskTracker udostępnia też *usługę przestawiania* zadań. Opiszemy ją w ramach omawiania procedury przestawiania.

**Kończenie pracy.** Po zakończeniu ostatniego zadania w pracy JobTracker porządkuje zasoby pracy (pozwala to usunąć pliki pośrednie w systemie HDFS i w lokalnych systemach plików procesów TaskTracker).

## B. Odporność na usterki w modelu MapReduce.

Są trzy rodzaje awarii: awaria zadania, awaria procesu TaskTracker i awaria procesu JobTracker.

**Awaria zadania.** Może wystąpić, jeśli kod zadania zgłosi wyjątek czasu wykonania lub gdy nastąpi nieoczekiwana awaria maszyny JVM. Inny problem występuje, gdy TaskTracker nie otrzymuje przez pewien (konfigurowalny) czas żadnych aktualizacji od procesu zadania. We wszystkich tych sytuacjach TaskTracker powiadamia JobTrackera o niepowodzeniu zadania. Gdy JobTracker otrzyma informację o niepowodzeniu, ponownie szereguje wykonanie zadania.

**Awaria TaskTrackera.** Proces TaskTracker może ulec awarii lub utracić połączenie z JobTrackerem. Gdy JobTracker oznaczy danego TaskTrackera jako niesprawnego, zadania mapowania ukończone przez tego TaskTrackera jeszcze raz trafiają do kolejki w oczekiwaniu na ich ponowne zaszeregowanie. Ponownie szeregowane są również wszystkie zadania mapowania i redukcji wykonywane w danym momencie w niesprawnym TaskTrackerze.

**Awaria JobTrackera.** W Hadoopie 1 awaria JobTrackera nie umożliwiała odtworzenia stanu. JobTracker był pojedynczym punktem podatności na awarię i wymagał ręcznego ponownego uruchomienia. Po wznowieniu jego działania wszystkie wykonywane prace trzeba było ponownie przestać. Była to jedna z wad Hadoopa 1, rozwiązana w nowej generacji modelu MapReduce w Hadoopie — na platformie YARN.

**Obsługa awarii.** Jeśli podane przez użytkownika operatory mapowania i redukcji są funkcjami deterministycznymi zależnymi od wartości wejściowych, model MapReduce daje te same dane wyjściowe co przy sekwencyjnym bezproblemowym wykonaniu całego programu. Każde zadanie zapisuje dane wyjściowe do prywatnego katalogu zadania. Jeśli JobTracker otrzyma wyniki wielu wykonań tego samego zadania, ignoruje je wszystkie oprócz pierwszego. Po zakończeniu pracy dane wyjściowe z zadań są przenoszone do katalogu z danymi wyjściowymi pracy.

### C. Procedura przestawiania.

Najważniejszą cechą modelu MapReduce jest to, że reducery łączą wszystkie wiersze o tym samym kluczu. Jest to zapewniane dzięki procedurze **przestawiania** (ang. *shuffle*) w modelu MapReduce. Przestawianie obejmuje etapy mapowania, kopiowania i redukcji.

**Etap mapowania.** Gdy wiersze są przetwarzane w zadaniach mapowania, początkowo znajdują się w utrzymywanym w pamięci buforze o konfigurowalnym rozmiarze (domyślnie jest to 100 MB). Działający w tle wątek dzieli wiersze z bufora na podstawie liczby reducerów w pracy i *mechanizmu podziału na partycje* (ang. *partitioner*). *Mechanizm podziału na partycje* to konfigurowalny interfejs, który wybiera reducer dla każdej wartości klucza i określa liczbę reducerów w pracy. Podzielone wiersze są sortowane według wartości kluczy. Można je dodatkowo posortować za pomocą podanego *mechanizmu porównywania* (ang. *comparator*), aby wiersze o tym samym kluczu były sortowane w niezmienny sposób. To podejście jest używane na potrzeby złączeń, aby zagwarantować, że wiersze o tej samej wartości klucza z danej tabeli zostaną zgrupowane wspólnie. Można też podać interfejs *mechanizmu łączenia* (ang. *combiner*). Służy on do zmniejszania liczby wierszy zwracanych przez mapper dla danego klucza, a stosuje się go, wykonując w każdym mapperze operację redukcji na wszystkich wierszach o tym samym kluczu. Na etapie mapowania można wykonać kilka serii podziału na partycje, sortowania i łączenia. Wynik końcowy to jeden posortowany według klucza plik lokalny dla każdego reducera.

**Etap kopiowania.** Reducery pobierają z wszystkich mapperów pliki, gdy te ostatnie staną się dostępne. Pliki są podawane przez JobTrackera w odpowiedziach na sygnały kontrolne. Każdy mapper jest powiązany ze zbiorem wątków nasłuchujących, które obsługują żądania plików nadsyłane przez reducery.

**Etap redukcji.** Reducer wczytuje wszystkie pliki z mapperów. Pliki te są scalane przed strumieniowym przekazaniem ich do funkcji redukującej. Scalanie może się odbywać w wielu etapach (zależy to od tego, w jaki sposób udostępniane są pliki z mapperów). Reducer unika niepotrzebnego scalania. Przykładowo, ostatnich  $N$  plików jest scalanych w trakcie strumieniowego przesyłania wierszy do funkcji redukującej.

### D. Szeregowanie zadań.

**JobTracker** w modelu MapReduce 1.0 odpowiada za szeregowanie pracy w węzłach klastra. Prace nadsyłane przez klienty są dodawane do kolejki prac w JobTrackerze. W początkowych wersjach Hadoopa używano programu szeregującego FIFO, który szeregował prace sekwencyjnie wraz z ich nadsyłaniem. W dowolnym momencie klastr wykonywał zadania z jednej pracy. To powodowało niepotrzebne opóźnienia w wykonywaniu krótkich prac, np. jednorazowych zapytań z narzędzia Hive, ponieważ musiały one czekać

na długie prace związane np. z uczeniem maszynowym. Czas oczekiwania mógł być dłuższy od czasu wykonywania prac, a przepustowość klastra spadała. Ponadto klaster nie wykorzystywał pełni swoich możliwości. Dalej pokrótce opiszemy dwa inne rodzaje programów szeregujących: sprawiedliwy program szeregujący (ang. *fair scheduler*) i program szeregujący zależny od zasobów (ang. *capacity scheduler*), które łągodzą opisany problem.

**Sprawiedliwy program szeregujący.** Jego celem jest zapewnienie krótkiego czasu odpowiedzi krótkim pracom we współużytkowanym klastrze z Hadoopem. Prace są tu grupowane w pule, a zasoby klastra są równomiernie przydzielane do pul. W każdym momencie zasoby klastra są po równo rozdzielone między pule, dzięki czemu możliwości klastra są wykorzystywane na jednolitym poziomie. Typowy sposób tworzenia pul polega na przypisaniu puli każdemu użytkownikowi i przydzieleniu jej minimalnej liczby slotów.

**Program szeregujący zależny od zasobów.** Ten program szeregujący ma spełniać potrzeby dużych przedsiębiorstw. Jest zaprojektowany tak, aby umożliwiać wielu użytkownikom współużytkowanie zasobów w dużym klastrze z Hadoopem dzięki szybkiemu przydziałowi zasobów na podstawie zestawu dotyczących ich ograniczeń. W dużych przedsiębiorstwach poszczególne działy nie chcą korzystać z jednego scentralizowanego klastra z Hadoopem w obawie przed tym, że nie zdołają zrealizować umów o poziomie świadczonych usług (ang. *Service Level Agreements* — SLAs) związanych z ich aplikacjami. Program szeregujący zależny od zasobów ma gwarantować każdemu użytkownikowi zasoby klastra zgodnie z następującymi założeniami:

- Używanych jest wiele kolejek z twardymi i miękkimi limitami określającymi procent wykorzystywanych zasobów.
- Używane są listy kontroli dostępu (ang. *access control lists* — ACLs) określające, kto może przysyłać, wyświetlać i modyfikować prace w kolejce.
- Nadmiarowe zasoby są równomiernie rozdzielane między aktywne kolejki.
- Dla użytkowników określone są limity chroniące przed zmonopolizowaniem klastra przez określoną jednostkę.

## 25.4.2. Przykład: złączenia w modelu MapReduce

Aby zrozumieć możliwości i przydatność modelu MapReduce, pouczające będzie przeanalizowanie najważniejszej operacji algebry relacyjnej — przedstawionych w rozdziale 6. złączeń. Ich działanie omówiliśmy w kontekście zapytań w języku SQL (rozdziały 7. i 8.) oraz optymalizacji (rozdziały 18. i 19.). Przyjrzyjmy się teraz problemowi złączania dwóch relacji:  $R(A, B)$  i  $S(B, C)$  na podstawie warunku  $R.A = S.B$ . Załóżmy, że obie tabele znajdują się w systemie HDFS. Opisujemy tu kilka strategii wykonywania złączeń równościowych w środowisku MapReduce.

**Złączanie sortująco-scalające.** Najbardziej ogólna strategia złączania polega na zastosowaniu przedstawiania do podziału i posortowania danych oraz na scalaniu i generowaniu danych wyjściowych za pomocą reducerów. Można skonfigurować pracę w modelu Map ↪ Reduce, która wczytuje bloki z obu tabel na etapie mapowania. Można też utworzyć *mechanizm podziału na partycje*, aby podzielić na partycje za pomocą mieszania wiersze z relacji  $R$  i  $S$  na podstawie wartości kolumny  $B$ . Klucz zwracany z etapu mapowania obejmuje

*znacznik* tabeli. Tak więc klucz ma postać (znacznik, (klucz)). W modelu MapReduce można skonfigurować niestandardowe sortowanie na potrzeby etapu przestawiania w pracy. Niestandardowe sortowanie porządkuje wiersze mające ten sam klucz. W takiej sytuacji wiersze o tej samej wartości  $B$  można posortować według znaczników. Reducer otrzyma więc wszystkie wiersze o tej samej wartości  $B$  w kolejności: wiersze z mniejszej tabeli — wiersze z większej tabeli. Reducer może umieścić w buforze wiersze z mniejszej tabeli. Gdy zacznie otrzymywać wiersze z większej tabeli, może wyznaczyć w pamięci iloczyn wektorowy za pomocą wierszy przechowywanej w buforze mniejszej tabeli, aby wygenerować wynik złączenia. Największy wpływ na koszt tej strategii ma przestawianie, kiedy to każdy wiersz jest wielokrotnie zapisywany i wczytywany.

**Złączanie z użyciem mieszania po stronie mappera.** Gdy  $R$  lub  $S$  to mała tabela, którą można dla każdego zadania wczytać do pamięci, na etapie mapowania wystarczy podzielić na porcje dużą tabelę. Każde zadanie mapowania może wczytać całą małą tabelę i wykorzystując  $B$  jako klucz mieszania, wykonać mieszanie w pamięci, a następnie przeprowadzić złączanie z użyciem mieszania. Działa to podobnie jak złączanie z użyciem mieszania w bazach danych. Koszty wykonania tego zadania są w przybliżeniu równe kosztowi wczytywania dużej tabeli.

**Złączanie z użyciem partycji.** Załóżmy, że  $R$  i  $S$  są przechowywane po podziale ich na partycje według kluczy złączania. Wtedy wszystkie wiersze z każdej porcji należą do określonego przedziału z domeny wartości pola złączania (w omawianym przykładzie jest nim  $B$ ). Przyjmijmy, że  $R$  i  $S$  są przechowywane w  $p$  plikach. Przyjmijmy, że plik ( $i$ ) zawiera takie wiersze, że (wartość  $B$ ) modulo  $p = i$ . Wtedy wystarczy złączyć  $i$ -ty plik relacji  $R$  z odpowiadającym mu  $i$ -tym plikiem relacji  $S$ . Jednym ze sposobów na uzyskanie tego efektu jest zastosowanie odmiany opisanego wcześniej złączenia po stronie mappera. Mapper obsługujący  $i$ -tą partycję większej tabeli powinien wczytywać  $i$ -tą partycję mniejszej tabeli. Tę strategię można rozbudować, aby działała nawet w sytuacji, gdy liczba partycji w obu tabelach jest różna. Wystarczy, że liczba partycji jednej tabeli będzie wielokrotnością liczby partycji drugiej. Przykładowo, jeśli tabela  $A$  jest podzielona na dwie partycje, a tabela  $B$  na cztery, partycję 1. z tabeli  $A$  należy złączyć z partycjami 1. i 3. tabeli  $B$ , a partycję 2. tabeli  $A$  z partycjami 2. i 4. tabeli  $B$ . Często można też zastosować złączenia oparte na pakietach (patrz dalej). Załóżmy np., że  $R$  i  $S$  to dane wyjściowe z wcześniejszego złączenia sortująco-skalającego. Dane wyjściowe z tego złączenia są dzielone na partycje w wyrażeniach złączania. Późniejsze złączenie tego zbioru danych pozwala uniknąć przestawiania.

**Złączanie oparte na pakietach.** Jest to połączenie złączania po stronie mappera ze złączaniem z użyciem partycji. W tym podejściu na partycje dzielona jest tylko jedna relacja (np. prawostronna). Następnie można uruchomić mappery dla relacji lewostronnej i przeprowadzić złączenie po stronie mappera z każdą partycją prawostronną.

**n-stronne złączenia po stronie mappera.** Można przeprowadzić złączenie tabel  $R(A, B, C, D)$ ,  $S(B, E)$  i  $T(C, F)$  w jednej pracy modelu MapReduce, pod warunkiem jednak, że klucze wszystkich małych tabel mieszczą się w buforach w pamięci. Takie złączenia są typowe w hurtowniach danych (patrz rozdział 29.), gdzie  $R$  to tabela faktów, a  $S$  i  $T$  to tabele wymiarów z kluczami  $B$  i  $C$ . W zapytaniach w hurtowni danych zwykle stosowane są filtry oparte na atrybutach wymiarów. Dlatego każde zadanie mapowania ma wystarczającą ilość pamięci, aby pomieścić wyniki mieszania kilku małych tabel wymiarów. Gdy w zadaniu mapowania wczytywane są wiersze z tabeli faktów, można je złączyć za pomocą mieszania z wszystkimi tabelami wymiarów, które to zadanie wczytało do pamięci.

**Proste złączenia n-stronne.** Złączenie tabel  $R(A, B)$ ,  $S(B, C)$  i  $T(B, D)$  można przeprowadzić w jednej pracy w modelu MapReduce, pod warunkiem jednak, że wiersze z kluczem wszystkich małych tabel mieszczą się w buforach w pamięci. Załóżmy, że  $R$  to duża tabela, a  $S$  i  $T$  to stosunkowo małe tabele. Wtedy zwykle dla dowolnej wartości klucza  $B$  w pamięci zadania można zmieścić określoną liczbę wierszy w tabeli  $S$  lub  $T$ . Przypisując dużej tabeli największy znacznik, łatwo jest uogólnić złączanie sortująco-scalające do postaci złączania n-stronnego, w którym wyrażenia złączania są takie same. Reducer uwzględniający wartości klucza  $B$  najpierw otrzymuje wiersze tabeli  $S$ , następnie wiersze tabeli  $T$ , a na koniec wiersze tabeli  $R$ . Ponieważ zgodnie z założeniem liczba wierszy w tabelach  $S$  i  $T$  nie jest duża, reducer może umieścić je w pamięci podręcznej. Gdy reducer otrzymuje wiersze tabeli  $R$ , może obliczyć iloczyn wektorowy z użyciem zapisanych w pamięci podręcznej wierszy tabel  $S$  i  $T$  oraz zwrócić wynik złączenia.

Obok opisanych strategii wykonywania złączeń za pomocą modelu MapReduce zaproponowano też algorytmy dla złączeń innego rodzaju (wykazano np., że ogólne n-stronne złączenie naturalne ze specjalnymi przypadkami złączeń łańcuchowych lub złączeń w schemacie gwiazdy można wykonać za pomocą jednej pracy w modelu MapReduce)<sup>14</sup>. Zaproponowano też algorytmy radzące sobie ze skośnym rozkładem wartości atrybutów złączania (np. w tabeli faktów dotyczącej sprzedaży w niektóre dni liczba transakcji może być nieproporcjonalnie duża). Na potrzeby złączania na podstawie atrybutów ze skośnym rozkładem wartości zmodyfikowany algorytm pozwala *mechanizmowi podziału na partycje* przypisywać unikatowe wartości do danych o dużej liczbie elementów, dzięki czemu można te dane przetwarzać w zadaniach redukcji. Pozostałe wartości można dzielić na partycje za pomocą mieszania w standardowy sposób.

To omówienie powinno dać Czytelnikom wyobrażenie o wielu możliwościach implementowania strategii złączania za pomocą modelu MapReduce. Istnieją też inne czynniki wpływające na wydajność, np. używanie baz wierszowych lub kolumnowych oraz przenoszenie predykatów do mechanizmów obsługi składowania danych. Omawianie tych kwestii wykracza poza zakres tej książki. Zainteresowani Czytelnicy znajdą aktualne prace badawcze z tej dziedziny podobne do tekstu Afratiego i Ullmana (2010).

Naszym celem w tym podrozdziale jest zwrócenie uwagi na dwa ważne narzędzia, które wywarły istotny wpływ na społeczność związaną z big data, ponieważ zapewniły wysokopoziomowe interfejsy dla podstawowych technologii: Hadoopa i modelu MapReduce. Dalej omówimy pokrótce język Pig Latin i system Hive.

**Apache Pig.** Pig<sup>15</sup> to system zaprojektowany w Yahoo! Research w celu zapełnienia luki między interfejsami deklaratywnymi (takimi jak SQL), które omówiliśmy w kontekście modelu relacyjnego, a bardziej restrykcyjnym, niskopoziomowym programowaniem w stylu proceduralnym, jakiego wymaga model MapReduce (patrz punkt 25.2.2). Choć w modelu MapReduce możliwy jest zapis bardzo złożonych analiz, użytkownik musi przedstawiać programy w formie dwuetapowego procesu (mapowanie i redukcja) z jednym wejściem. Ponadto MapReduce nie udostępnia metod opisu złożonych przepływów danych obejmujących sekwencję transformacji danych wejściowych. Nie istnieje standardowy sposób wykonywania typowych transformacji, takich jak projekcje, filtrowanie, grupowanie i złączanie.

---

<sup>14</sup> Patrz Afrati i Ullman (2010).

<sup>15</sup> Patrz Olston i in. (2008).

W rozdziałach 7. i 8. zobaczyłeś, jak zapisywać wszystkie te operacje deklaratywnie w języku SQL. Istnieje jednak społeczność użytkowników i programistów myślących w sposób proceduralny. Dlatego twórcy systemu Pig wymyślili język Pig Latin, aby znaleźć złoty środek między językiem SQL a modelem MapReduce. Dalej przedstawimy przykładowe proste zapytanie Group By zapisane w języku Pig Latin w pracy Olstona i in. (2008):

```
There is a table of urls: (url,category,pagerank).
```

Celem jest znalezienie dla kategorii z dużą liczbą adresów URL średniego rankingu stron dla adresów URL z danej kategorii mających wysoki ranking. Wymaga to pogrupowania adresów URL według kategorii. Zapytanie w języku SQL odpowiadające opisanemu wymogowi może wyglądać tak:

```
SELECT category, AVG(pagerank)
FROM urls WHERE pagerank > 0.2
GROUP BY category HAVING COUNT(*) > 10**6
```

To samo zapytanie w języku Pig Latin wygląda tak:

```
good_urls = FILTER urls BY pagerank > 0.2;
groups = GROUP good_urls BY category;
big_groups = FILTER groups BY COUNT(good_urls)> 10**6;
output = FOREACH big_groups GENERATE
category, AVG(good_urls.pagerank);
```

Jak widać w tym przykładzie, skrypt napisany za pomocą języka skryptowego Pig Latin to sekwencja kroków transformacji danych. W każdym kroku zapisywana jest prosta transformacja, taka jak filtrowanie, grupowanie lub projekcja. Taki skrypt przypomina plan zapytania w języku SQL, podobny do planów opisanych w rozdziale 19. Omawiany język obsługuje operacje na zagnieżdżonych strukturach danych, np. w formatach JSON i XML. Udostępnia też rozbudowaną i rozszerzalną bibliotekę funkcji i umożliwia wiązanie schematu z danymi na bardzo późnym etapie (można też w ogóle zrezygnować z tego kroku).

System Pig został zaprojektowany w celu rozwiązania problemów z zadaniami takimi jak jednorazowe analizy dzienników internetowych i strumieni kliknięć. Dzienniki i strumienie kliknięć wymagają zwykle niestandardowego przetwarzania na poziomie wierszy, a także na poziomie zagregowanym. Pig dobrze obsługuje funkcje definiowane przez użytkownika, a także zagnieżdżony model danych z czterema następującymi typami:

**Atomy:** proste wartości atomowe, np. liczby lub ciągi znaków.

**Krotki:** sekwencje pól, z których każde może być dowolnego dozwolonego typu.

**Wielozbiory:** kolekcje krotek (krotki mogą się w nich powtarzać).

**Odwzorowania:** kolekcje elementów danych, w których każdy element ma klucz umożliwiający bezpośredni dostęp do danych.

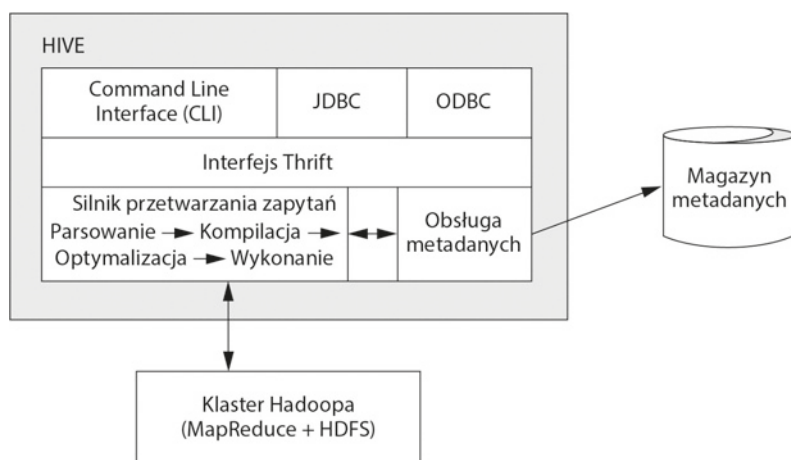
Olston i in. (2008) zademonstrowali ciekawe zastosowania dzienników z wykorzystaniem systemu Pig. Przykładem są analizy dzienników aktywności w wyszukiwarkach w dowolnym okresie (dnia, tygodnia, miesiąca itd.) w celu obliczenia częstotliwości wyszukiwań pojęć według lokalizacji geograficznej użytkowników. Potrzebne tu funkcje to odwzorowywanie adresów IP na lokalizacje geograficzne i pobieranie  $n$ -gramów. Inne zastosowanie to grupowanie zapytań wyszukiwania z jednego okresu i z innego okresu z przeszłości na podstawie szukanych pojęć.



Pig został zaprojektowany tak, aby mógł działać w różnych środowiskach uruchomieniowych. W systemie Pig język Pig Latin jest kompilowany do postaci fizycznych planów przekształcanych na serie prac w modelu MapReduce uruchamianych w Hadoopie. Pig okazał się przydatnym narzędziem do zwiększania produktywności programistów w środowisku Hadoopa.

### 25.4.3. Apache Hive

Hive został opracowany w firmie Facebook<sup>16</sup> w podobnym celu — aby zapewnić wyższego poziomu interfejs dla Hadoopa, oparty na zapytaniach w stylu języka SQL i obsługujący przetwarzanie zagregowanych zapytań analitycznych typowych w hurtowniach danych (patrz rozdział 29.). Hive pozostaje podstawowym interfejsem dostępu do danych w Hadoopie w firmie Facebook. Zyskał dużą popularność w społeczności związanej z otwartym oprogramowaniem i jest stale usprawniany. Twórcy systemu Hive wyszli poza język Pig Latin, ponieważ udostępnili nie tylko interfejs wysokopoziomowego języka dla Hadoopa, ale też warstwę, dzięki której Hadoop wygląda jak SZBD z obsługą języka DDL, repozytorium metadanych, dostępem do sterowników JDBC i ODBC, a także kompilatorem języka SQL. Architektura i komponenty systemu Hive są przedstawione na rysunku 25.2.



RYSUNEK 25.2. Architektura i komponenty systemu Hive

Rysunek 25.2 przedstawia interfejs Apache Thrift dla systemu Hive. Apache Thrift definiuje język IDL (ang. *Interface Definition Language*) i protokół komunikacyjny używane do rozwijania usług zdalnych. Thrift obejmuje środowisko uruchomieniowe i system generowania kodu, które można wykorzystać do rozwijania usług zdalnych w wielu językach, takich jak Java, C++, Python i Ruby. Thrift obsługuje protokoły binarne i oparte na formacie JSON, a także transfer danych z użyciem protokołu HTTP, gniazd i plików.

Język zapytań HiveQL z systemu Hive obejmuje podzbiór języka SQL z wszystkimi typami złączeń, operacjami grupowania, a także przydatnymi funkcjami powiązаныmi z prostymi i złożonymi typami danych. Dalej omawiamy wybrane ważne aspekty systemu Hive.

<sup>16</sup> Patrz Thusoo i in. (2010).



### Interfejs dla systemu HDFS:

- Tabele w systemie Hive są powiązane z katalogami systemu HDFS. Użytkownicy mogą definiować partycje w tabelach. Przykładowo, tabelę dziennika internetowego można dzielić na partycje według dni i — w ramach dni — według godzin. Każdy poziom partycji odpowiada poziomowi katalogów w systemie HDFS. Tabelę można też przechowywać w formie *pakietów* obejmujących zbiory kolumn. To oznacza, że składowane dane są fizycznie dzielone na podstawie kolumn. Przykładowo, w katalogu dla każdej godziny dane mogą być dzielone na *pakiety* według identyfikatorów użytkowników. Wtedy dane z każdej godziny są zapisywane w zbiorach plików, z których każdy reprezentuje pakiet użytkowników, a pakiety są wyznaczane na podstawie wyniku mieszania wartości z kolumny z identyfikatorem użytkownika. Użytkownicy mogą określać, na ile pakietów należy podzielić dane.
- Architektura wtyczek SerDe (Serialization/Deserialization) pozwala użytkownikom określić, w jaki sposób dane w natywnych formatach plików będą udostępniane jako wiersze operatorom języka SQL z systemu Hive. Hive udostępnia rozbudowany zestaw funkcji SerDe i obsługiwanych formatów plików (np.: CSV, JSON, SequenceFile), formatów kolumnowych (np.: RCFile, ORCFile, Parquet) i Avro (jest to inny system serializacji danych). Różne *mechanizmy składowania danych* (ang. *StorageHandlers*) są oparte na mechanizmie SerDe i umożliwiają konfigurowanie sposobu odczytu oraz zapisu danych, a także przekazywanie *predykatów* na potrzeby szybkiego przetwarzania danych. Przykładowo, *mechanizm składowania danych JDBC* umożliwia użytkownikom systemu Hive zdefiniowanie tabeli, która w rzeczywistości jest składowana w jakimś relacyjnym SZBD i używana za pomocą protokołu JDBC (patrz rozdział 10.) w trakcie wykonywania zapytań.

**Obsługa języka SQL i optymalizacje w systemie Hive.** W systemie Hive używane są optymalizacje logiczne i fizyczne podobne do tych stosowanych w zapytaniach SQL (patrz rozdziały 18. i 19.). W systemie tym od dawna obsługiwane były optymalizacje fizyczne takie jak pomijanie niepotrzebnych kolumn i przenoszenie predykatów selekcji w dół drzewa zapytania. Wprowadzono też optymalizacje fizyczne związane z przekształcaniem złączeń sortująco-scalających na złączenia po stronie mapowania na podstawie wskazówek od użytkowników i wielkości plików z danymi. W systemie Hive początkowo obsługiwany był podzbiór języka SQL-92 obejmujący instrukcje SELECT, JOIN i GROUP BY oraz filtry oparte na warunkach z klauzuli WHERE. Użytkownicy systemu Hive mogą zapisywać w nim złożone polecenia języka SQL. Już we wczesnych wersjach Hive pozwalał wykonywać 22 zapytania testowe z pakietu TPC organizacji TPC (ang. *Transaction Processing Performance Council*), choć wymagało to istotnych ręcznych modyfikacji.

Później wprowadzono znaczne usprawnienia w obsłudze języka oraz technikach pracy optymalizatora i środowiska uruchomieniowego. Oto wybrane z tych usprawnień:

- Do języka SQL w systemie Hive dodano wiele funkcji analitycznych, np.: predykaty w podzapytaniach, wyrażenia CTE (ang. *common table expression*; odpowiadają one klauzuli WITH języka SQL, która umożliwia użytkownikom nazywanie powtarzających się bloków podzapytań i wielokrotne wskazywanie ich w zapytaniu; te wyrażenia można uznać za perspektywy z poziomu zapytania), agregacje dotyczące określonych fragmentów danych, instrukcję ROLLUP (do agregowania danych na wyższych poziomach) i zbiory grupowania (pozwalają

zastosować wiele poziomów agregacji na jednym poziomie grupowania). Rozważ np. wyrażenie `Group By Grouping Sets ((rok, miesiąc), (dzień_tygodnia))`. Reprezentuje ono agregacje zarówno na poziomie `(rok, miesiąc)`, jak i na poziomie `dzień_tygodnia`. Obecnie obsługiwany jest też kompletny zestaw typów danych języka SQL, w tym typy `varchar`, typy liczbowe i daty. Hive obsługuje też — za pomocą instrukcji `Insert` i `Update` — standardowe przepływy ETL oparte na procesie `Change Data Capture (CDC)`. W magazynie danych proces udostępniania rzadko zmieniających się wymiarów (np. listy klientów w magazynie danych sprzedażowych) wymaga złożonego przepływu danych obejmującego identyfikowanie nowych i zaktualizowanych rekordów z danego wymiaru. Jest to właśnie proces `CDC`. Dzięki dodaniu do języka SQL w systemie Hive instrukcji `Insert` i `Update` można w nim zapisywać i wykonywać procesy `CDC`.

- Hive obejmuje obecnie znacznie rozbudowany zestaw instrukcji `DDL` służących do przyznawania uprawnień w kategoriach swobodnie konfigurowanej kontroli dostępu (patrz podrozdział 30.2).
- Wprowadzono kilka standardowych optymalizacji baz danych, w tym pomijanie partycji, zmianę kolejności złączeń, modyfikowanie indeksów i ograniczanie liczby prac w modelu `MapReduce`. Bardzo duże tabele, np. tabele faktów w hurtowniach danych, są zwykle dzielone na partycje. Prawdopodobnie najczęściej odbywa się to na podstawie atrybutu czasu. Gdy warstwą składowania danych jest system `HDFS`, użytkownicy zwykle przechowują dane przez długi czas. Jednak typowa hurtownia obejmuje tylko dane z ostatnich okresów (np. z ostatniego kwartału lub bieżącego roku). Te okresy są podawane jako filtry w zapytaniu. Pomijanie partycji to technika pobierania odpowiednich predykatów z filtrów zapytania i przekształcania ich w listę partycji tabeli, które należy wczytać. Oczywiście ma to istotny wpływ na wydajność i wykorzystanie zasobów klastra. Zamiast skanować wszystkie partycje zapisane w ostatnich  $N$  latach, można uwzględnić tylko partycje z kilku ostatnich tygodni lub miesięcy. Trwają prace nad rejestrowaniem statystyk z poziomu kolumn i tabel oraz generowaniem planów na podstawie modelu kosztów wykorzystującego te statystyki (podobnie jak w relacyjnych SZBD, co opisano w rozdziale 19.).
- Obecnie Hive obsługuje środowisko uruchomieniowe `Tez`, które ma istotne zalety w porównaniu z modelem `MapReduce` (m.in. nie trzeba zapisywać danych na dysku między pracami i nie trzeba ograniczać się do dwuetapowych procesów z jednym wejściem). Trwają też prace nad dodaniem do systemu Hive obsługi `Sparka` — nowej technologii, którą pokrótce opiszemy w podrozdziale 25.6.

## 25.4.4. Zalety technologii Hadoop i MapReduce

Hadoop 1 był zoptymalizowany pod kątem przetwarzania wsadowego bardzo dużych zbiorów danych. Do jego sukcesu przyczyniły się różne czynniki:

- (1) Szybkość przeszukiwania dysku jest czynnikiem ograniczającym wydajność przy pracy z obciążeniem roboczym na poziomie petabajtów. Szybkość ta jest ograniczona strukturą dysku mechanicznego, natomiast prędkość transferu jest zależna od elektroniki i stale rośnie (omówienie dysków twardych znajdziesz w podrozdziale 16.2). Rozwiązaniem problemu jest model `MapReduce`, gdzie

zbiory danych są skanowane równolegle. Przykładowo, sekwencyjne skanowanie z szybkością 50 MB/sekundę na jednej maszynie zbioru danych zajmującego 100 TB wymaga ok. 24 dni, a skanowanie równoległe tych samych danych za pomocą 1000 maszyn zajmuje tylko 35 minut. Twórcy Hadoopa zalecają stosowanie bloków o bardzo dużej wielkości — 64 MB lub większej. Dlatego w trakcie skanowania zbiorów danych procent czasu przeznaczanego na przeszukiwanie dysków jest pomijalny. Nieograniczona szybkość przeszukiwania dysków w połączeniu z równoległym przetwarzaniem dużych zbiorów danych w porcjach wpływa na skalowalność i szybkość modelu MapReduce.

- (2) Model MapReduce umożliwia łatwiejszą (w porównaniu z tradycyjnymi relacyjnymi SZBD, wymagającymi zdefiniowanych schematów) obsługę częściowo strukturalnych danych i zbiorów danych z parami klucz-wartość. W relacyjnych SZBD poważnym problemem są pliki takie jak bardzo duże pliki dziennika, ponieważ przed analizowaniem trzeba je przetwarzać na wiele sposobów.
- (3) Model MapReduce skaluje się liniowo, ponieważ można dodawać zasoby, aby liniowo skracać czas wykonywania prac i zwiększać przepustowość. Model obsługi awarii jest prosty, a pojedyncze zakończone awarią zadania można ponownie uruchomić bez wpływu na całą pracę.

## 25.5. Hadoop 2 (nazywany też YARN)

W poprzednich podrozdziałach szczegółowo omówiliśmy rozwój Hadoopa. Opisaliśmy podstawowe aspekty modelu programowania MapReduce i system HDFS będący podstawą infrastruktury składowania danych. Omówiliśmy też wysokopoziomowe interfejsy takie jak Pig i Hive, dzięki którym na platformie Hadoop możliwe jest wysokopoziomowe przetwarzanie danych w sposób podobny jak w języku SQL. Teraz skupimy się na nowszych rozwiązaniach, nazywanych ogólnie Hadoopem 2, modelem MapReduce 2 lub YARN (ang. *Yet Another Resource Negotiator*). Najpierw wymienimy braki platformy Hadoop 1 i uzasadnienie powstania platformy YARN.

### 25.5.1. Uzasadnienie powstania platformy YARN

Mimo sukcesu Hadoopa 1 doświadczenia jego użytkowników w aplikacjach dla przedsiębiorstw wykazały pewne braki tej platformy. Wynikało z nich, że konieczne mogą być usprawnienia Hadoopa 1:

- Wraz ze wzrostem wielkości klastrów i liczby użytkowników JobTracker stał się wąskim gardłem. Od zawsze było wiadomo, że stanowi on pojedynczy punkt podatności na awarie.
- Z uwagi na statyczny przydział zasobów dla funkcji mapujących i redukujących wykorzystanie możliwości klastra węzłów było niezadowolające.
- HDFS uznawano za jedyny system składowania danych w przedsiębiorstwie. Jednak użytkownicy chcieli uruchamiać różne aplikacje, które niełatwo było dostosować do modelu MapReduce. Użytkownicy zwykle radzili sobie z tym ograniczeniem, uruchamiając prace obejmujące wyłącznie etap mapowania. To jednak tylko nasilało problemy z szeregowaniem prac i wykorzystaniem zasobów.

- W dużych klastrach problemem było nadążanie z nowymi otwartymi wersjami Hadoopa, udostępnianymi co kilka miesięcy.

Te przyczyny uzasadniają powstanie drugiej wersji Hadoopa. Niektóre kwestie z tej listy zasługują na przedstawione dalej szczegółowe omówienie.

**Wielodostępność.** Wielodostępność dotyczy jednoczesnej obsługi wielu jednostek lub użytkowników, tak aby umożliwić im współużytkowanie zasobów. Wraz ze wzrostem wielkości klastra i liczby użytkowników z klastra Hadoopa mogły zacząć korzystać różne grupy osób. W firmie Yahoo! pierwszym rozwiązaniem tego problemu był model **Hadoop on Demand** (czyli Hadoop na żądanie), oparty na menedżerze zasobów Torque i programie szeregującym Maui. Użytkownicy mogli dla każdej pracy (lub grupy prac) skonfigurować odrębny klaster. To podejście miało kilka zalet:

- W każdym klastrze mogła działać inna wersja Hadoopa.
- Awarie JobTrackera były ograniczone do jednego klastra.
- Każdy użytkownik (lub każda organizacja) mógł podejmować niezależne decyzje co do wielkości i konfiguracji klastra na podstawie oczekiwanego obciążenia roboczego.

Jednak Yahoo! zrezygnowała z modelu Hadoop on Demand. Wynikało to z następujących powodów:

- Przydział zasobów nie odbywał się na podstawie lokalności danych. Dlatego większość odczytów i zapisów w systemie HDFS wymagała dostępu do zdalnych węzłów, co niweczyło jedną z ważnych zalet modelu MapReduce związaną z głównie lokalnym dostępem do danych.
- Przydział klastra odbywał się statycznie. To oznaczało, że duże fragmenty klastra pozostawały bezczynne:
  - W pracy w modelu MR w fazie mapowania nie były używane sloty redukcji, a na etapie redukcji nie były wykorzystywane sloty mapowania. Gdy używane były języki wyższego poziomu, np. Pig i Hive, wszystkie skrypty lub zapytania generowały wiele prac. Ponieważ przydział klastra odbywał się statycznie, trzeba było z góry uzyskać maksymalną liczbę węzłów potrzebnych w jakiejkolwiek pracy.
  - Nawet przy szeregowaniu sprawiedliwym lub zależnym od zasobów (patrz omówienie w punkcie 25.4.2) podział klastra na stałą liczbę slotów mapowania i redukcji oznaczał, że możliwości klastra nie były w pełni wykorzystane.
- Opóźnienie związane z pozyskaniem klastra było wysokie. Klaster przyznawano dopiero wtedy, gdy dostępna stawała się wystarczająca liczba węzłów. Użytkownicy zaczęli więc przedłużać czas życia klastrów i utrzymywać je dłużej, niż było to konieczne. Negatywnie wpływało to na poziom wykorzystania możliwości klastra.

**Skalowalność JobTrackera.** Gdy wielkość klastra rosła powyżej 4000 węzłów, problemy z zarządzaniem pamięcią i blokadami sprawiły, że trudno było rozbudować JobTrackera tak, by obsługiwał tak wysokie obciążenie robocze. Rozważano wiele możliwości, np.: przechowywanie danych o pracach w pamięci, ograniczenie liczby zadań na jedną pracę, ograniczenie liczby prac zgłaszanych przez użytkownika i ograniczenie liczby jednocześnie wykonywanych prac. Jednak żadne z tych rozwiązań nie zadowalało wszystkich użytkowników, a JobTrackerowi często brakowało pamięci.

Powiązany problem dotyczył ukończonych prac. Takie prace były przechowywane w JobTrackerze i zajmowały pamięć. Opracowano wiele schematów ograniczania liczby ukończonych prac i miejsca zajmowanego przez nie w pamięci. Ostatecznie skutecznym rozwiązaniem okazało się przeniesienie obsługi ukończonych prac do odrębnego demona odpowiedzialnego za historię prac.

Wraz ze wzrostem liczby TaskTrackerów opóźnienie sygnałów kontrolnych (z TaskTrackera do JobTrackera) wzrosło do prawie 200 milisekund. To oznaczało, że przy 200 TaskTrackerach w klastrze przerwy między sygnałami kontrolnymi mogły wynosić 40 sekund lub więcej. Próbowano rozwiązać ten problem, jednak ostatecznie zarzucono wysiłki w tym obszarze.

**JobTracker: pojedynczy punkt podatności na awarie.** Model odtwarzania danych w Hadoopie 1 był bardzo niedoskonały. Awaria JobTrackera powodowała, że cały klastr przestawał działać. W takiej sytuacji tracono stan działających prac, trzeba było ponownie przestać wszystkie prace i wznowić działanie JobTrackera. Próby utrwalania informacji o ukończonych pracach zakończyły się niepowodzeniem. Powiązany problem dotyczył instalowania nowych wersji oprogramowania. Wymagało to zaplanowania przestoju klastra, co skutkowało zaległymi pracami i późniejszym obciążeniem JobTrackera po wznowieniu funkcjonowania klastra.

**Niewłaściwe korzystanie z modelu MapReduce.** Środowisko uruchomieniowe MR nie było dobrze dostosowane do przetwarzania iteracyjnego. Było to prawdą zwłaszcza w kontekście algorytmów uczenia maszynowego w obciążeniu roboczym z obszaru analiz. Każda iteracja jest traktowana jak praca w modelu MR. Algorytmny grafowe lepiej jest zapisywać za pomocą modelu BSP (ang. *bulk synchronous parallel*), w którym używane jest przekazywanie komunikatów zamiast prostych mechanizmów mapowania i redukcji. Użytkownicy radzili sobie z ograniczeniami, stosując niewydajne rozwiązania alternatywne, takie jak implementowanie algorytmów uczenia maszynowego za pomocą długich prac obejmujących sam etap mapowania. Tego typu prace najpierw wczytywały dane z systemu HDFS, a następnie równolegle wykonywały pierwszy przebieg obliczeń, jednak potem wymieniały między sobą dane poza kontrolą platformy. Utracono też odporność na błędy. JobTracker nie wiedział, jak wykonane zostały prace, co skutkowało niskim wykorzystaniem zasobów i niestabilnością klastra.

**Problemy z modelem zasobów.** W Hadoopie 1 węzeł był dzielony na stałą liczbę slotów mapowania i redukcji. To prowadziło do niepełnego użycia zasobów, ponieważ nie można było wykorzystać nieaktywnych slotów. Ponadto w węzłach nie można było wykonywać prac spoza modelu MR, ponieważ obciążenie węzła było nieprzewidywalne.

Opisane problemy ilustrują, dlaczego Hadoop 1 wymagał aktualizacji. Choć podejmowano próby wyeliminowania w Hadoopie 1 wielu powyższych wad, stało się jasne, że platforma wymaga przeprojektowania. Oto cele, jakie wyznaczono nowemu projektowi:

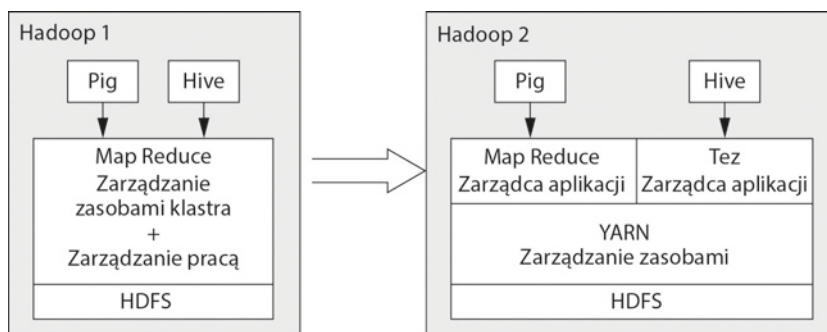
- Zwiększenie skalowalności i poziomu uwzględniania lokalności danych w porównaniu z Hadoopem 1.
- Zapewnienie wielodostępności i wysokiego wykorzystania zasobów klastra.
- Brak pojedynczego punktu podatności na awarię i wysoka dostępność.

- Obsługa mechanizmów innych niż tylko prace modelu MapReduce. Zasoby klastra nie powinny być reprezentowane jako statyczne sloty mapowania i redukcji.
- Zgodność wstecz, tak aby istniejące prace można było wykonywać w ich obecnej postaci i najlepiej bez konieczności ponownego kompilowania.

W efekcie powstała platforma YARN, czyli Hadoop 2, którą opiszemy w następnym punkcie.

## 25.5.2. Architektura platformy YARN

**Przegląd.** Po przedstawieniu uzasadnienia aktualizacji Hadoopa 1 teraz omówimy szczegółowo architekturę następnej wersji tej platformy, nazywanej MR 2, MapReduce 2.0, Hadoop 2 lub YARN<sup>17</sup>. Główną ideą twórców platformy YARN było rozdzielenie zarządzania zasobami klastra i zarządzania pracami. Ponadto na platformie YARN wprowadzono *zarządcę aplikacji* (ang. *ApplicationMaster*), który obecnie odpowiada za zarządzanie wykonywaniem zadań (przepływami danych w zadaniach, cyklem życia zadań, przełączaniem awaryjnym zadań itd.). Model MapReduce jest teraz dostępny jako usługa (lub aplikacja) oferowana przez *zarządcę aplikacji MapReduce*. Te dwa rozwiązania mają daleko idące implikacje i są podstawą systemu operacyjnego z obsługą danych. Na rysunku 25.3 pokazano ogólny schemat platform Hadoop 1 i Hadoop 2.



RYСУNEK 25.3. Schemat Hadoopa 1 i Hadoopa 2

*Menedżer zasobów* i używany dla węzłów roboczych *menedżer węzłów* wspólnie tworzą platformę, na której YARN może obsługiwać dowolne aplikacje. Menedżer zasobów zarządza klastrem, udostępniając zasoby na podstawie konfigurowalnej strategii szeregowania (np. szeregowania sprawiedliwego lub strategii optymalizacji wykorzystania zasobów klastra). Ten menedżer odpowiada też za cykl życia węzłów klastra, ponieważ śledzi, kiedy kończą pracę, stają się niedostępne lub są dodawane do klastra. Awarie węzłów są zgłaszane zarządcom aplikacji, którzy utrzymywali kontenery w uszkodzonych węzłach. Nowe węzły są udostępniane zarządcom aplikacji.

Zarządcy aplikacji przesyłają żądania zasobów do menedżera zasobów, który odpowiada przyznaniem dzierżawy kontenera w klastrze. **Kontener** jest dzierżawiony przez menedżera zasobów zarządcy aplikacji, który może wtedy korzystać z określonej części zasobów

<sup>17</sup> Aktualną dokumentację platformy YARN znajdziesz w witrynie organizacji Apache: <http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html>.



węzła z klastra. Zarządca aplikacji przekazuje kontekst uruchomienia kontenera menedżerowi węzła, którego dotyczy dana dzierżawa. Kontekst uruchomienia, obok zapewnienia dzierżawy, określa też, jak uruchomić proces powiązany z zadaniem i jak uzyskiwać zasoby takie jak archiwa *.jar*, biblioteki potrzebne procesowi, zmienne środowiskowe i tokeny bezpieczeństwa. Węzeł ma określoną moc obliczeniową zależną od liczby rdzeni, ilości pamięci, przepustowości sieci itd. Obecnie YARN uwzględnia tylko pamięć. Na podstawie mocy obliczeniowej węzeł można podzielić na zbiór używanych zamiennie kontenerów. Gdy zarządca aplikacji otrzyma dzierżawę kontenera, może swobodnie szeregować w nim prace. Zarządcy aplikacji, na podstawie obciążenia roboczego, mogą stale zmieniać wymogi dotyczące zasobów. Menedżer zasobów podejmuje decyzje o szeregowaniu wyłącznie na podstawie żądań, stanu klastra i używanej w klastrze strategii szeregowania. Nie wie, jakie zadania są aktualnie wykonywane w węzłach. Odpowiedzialność za zarządzanie pracami i analizowanie ich ponoszą zarządcy aplikacji.

*Menedżer węzła* odpowiada za zarządzanie kontenerami w danym węźle. Kontenery mają za zadanie informować o stanie węzła. Obsługują też procedurę dołączania węzłów do klastra. Kontenery świadczą zarządcom aplikacji usługę uruchamiania kontenera. Inne dostępne usługi to lokalna pamięć podręczna, dostępna na poziomie użytkownika, na poziomie aplikacji i na poziomie kontenera. Można też tak skonfigurować kontenery, aby świadczyły inne usługi na rzecz działających w nich zadań. Przykładowo, obecnie dla zadań MR świadczoną usługą z poziomu węzła jest przestawianie.

*Zarządca aplikacji* odpowiada za uruchamianie prac w klastrze. Na podstawie prac klastry negocjują z menedżerem zasobów przydział tych ostatnich. Sam *zarządca aplikacji* też działa w klastrze. W czasie rozruchu klient przesyła aplikację do menedżera zasobów, który następnie przydziela kontener zarządcy aplikacji i uruchamia zarządcę w danym kontenerze. W modelu MR zarządca aplikacji wykonuje większość obowiązków JobTrackera: uruchamia zadania mapowania i redukcji, podejmuje decyzje co do ich lokalizacji, zarządza przełączaniem awaryjnym zadań, utrzymuje liczniki podobne do liczników stanu prac i zapewnia interfejs monitorowania wykonywanych prac. Zarządzanie ukończonymi pracami i związany z nimi interfejs zostały przeniesione na odrębny serwer historii prac.

Oddzielenie zarządzania zasobami od zarządzania aplikacją w architekturze YARN ma następujące zalety:

- Można stosować bardzo różnorodne usługi dotyczące danych, aby wykorzystać zasoby klastra. W każdej z tych usług można stosować inny model programowania.
- Zarządcy aplikacji mogą negocjować zasoby w sposób zoptymalizowany pod kątem działania aplikacji. Przykładowo, aplikacje z obszaru uczenia maszynowego mogą zajmować kontenery na długi czas.
- Model zasobów i kontenerów umożliwia wykorzystanie możliwości węzłów w dynamiczny sposób, co zwiększa ogólny poziom wykorzystania zasobów klastra.
- Menedżer zasobów odpowiada za tylko jedną rzecz — zarządzanie zasobami. Dlatego jest wysoce skalowalny i może obsługiwać dziesiątki tysięcy węzłów.
- Ponieważ to zarządcy aplikacji zarządzają pracami, w klastrze można uruchamiać wiele wersji danej aplikacji. Nie jest konieczne globalne aktualizowanie klastra, co wymagałoby zatrzymania wszystkich prac.



Awaria zarządcy aplikacji wpływa tylko na zarządzane przez niego prace. Menedżer zasobów w pewnym stopniu steruje zarządcami aplikacji. Omówimy teraz pokrótce wszystkie komponenty środowiska YARN.

**Menedżer zasobów (MZ).** Menedżer zasobów odpowiada za przydział zasobów aplikacjom, a nie za optymalizowanie przetwarzania w ramach aplikacji. Strategie przydziału zasobów można zmieniać. Zarządcy aplikacji mają żądać zasobów w sposób optymalnie dostosowany do obciążenia roboczego.

Menedżer zasobów udostępnia następujące interfejsy:

- (1) Interfejs API dla klientów, aby umożliwić uruchamianie zarządców aplikacji.
- (2) Protokół dla zarządców aplikacji pozwalający negocjować przydział zasobów klastra.
- (3) Protokół dla menedżerów węzłów pozwalający przysyłać raporty o zasobach węzłów i umożliwiający zarządzanie zasobami przez menedżera zasobów.

Program szeregujący w menedżerze zasobów dopasowuje przesłane przez aplikacje żądania zasobów do globalnego stanu zasobów w klastrze. Przydział odbywa się na podstawie strategii z konfigurowalnego programu szeregującego (np. szeregowanie sprawiedliwe lub zależne od zasobów). Zarządcy aplikacji domagają się zasobów za pomocą **żądań zasobów**. Żądanie zasobów określa:

- Liczbę potrzebnych kontenerów.
- Zasoby fizyczne (procesor, pamięć) potrzebne w kontenerze.
- Preferencje wynikające z lokalności kontenerów (w danym węźle fizycznym lub w danej szafie serwerowej).
- Priorytet żądania powiązany z daną aplikacją.

Program szeregujący realizuje żądania na podstawie stanu klastra podawanego za pomocą sygnałów kontrolnych przez menedżer węzła. Lokalność i priorytety pomagają programowi szeregującemu wybierać rozwiązania. Przykładowo, jeśli żądany węzeł jest zajęty, następnym najlepszym wyborem jest inny węzeł z tej samej szafy serwerowej.

Program szeregujący potrafi też żądać od aplikacji zwrócenia zasobów, jeśli są one potrzebne, a nawet wymusić odzyskanie zasobów. Gdy aplikacja zwraca kontener, może albo przenieść swoje zadania do innego kontenera, albo zapisać stan w punkcie kontrolnym i odtworzyć go w innym kontenerze. Należy zwrócić uwagę na to, za co menedżer zasobów *nie odpowiada*: za obsługę wykonywania zadań aplikacji, za udostępnianie informacji o stanie aplikacji, za udostępnianie historii ukończonych prac i za odtwarzanie zadań zakończonych niepowodzeniem.

**Zarządca aplikacji (ZA).** Zarządca aplikacji odpowiada za koordynowanie wykonywania aplikacji w klastrze. Aplikacją może być zestaw procesów (np. praca modelu MR) lub długo działająca usługa (np. klastrer Hadoop on Demand obsługujący wiele prac modelu MR). Rodzaj aplikacji określa jej autor.

Zarządca aplikacji za pomocą sygnałów kontrolnych okresowo powiadamia menedżera zasobów o swoich żądaniach. Zasoby są udostępniane zarządcy aplikacji w formie dzierżaw kontenerów. Zasoby wykorzystywane przez aplikację są określone dynamicznie na podstawie postępów prac aplikacji i stanu klastra. Rozważmy przykład: zarządca apli-

kacji opartej na modelu MR uruchamiający pracę modelu MR żąda kontenera z każdego z  $m$  węzłów, w których znajdują się porcje danych wejściowych. Jeśli uzyska kontener w jednym z tych węzłów, albo wycofuje żądanie kontenerów z pozostałych  $m - 1$  węzłów, albo przynajmniej zmniejsza ich priorytet. Z kolei jeśli zadanie mapowania zakończy się niepowodzeniem, ZA wykrywa to i żąda kontenerów w innych węzłach zawierających replikę danej porcji danych wejściowych.

**Menedżer węzła.** W każdym węźle roboczym klastra działa menedżer węzła. Zarządza on kontenerami i udostępnia konfigurowalne usługi z nimi związane. Na podstawie szczegółowej specyfikacji *kontekstu uruchomienia kontenera* menedżer węzła może uruchomić w swoim węźle proces ze skonfigurowanym środowiskiem i lokalnymi katalogami. Menedżer monitoruje także, czy poziom wykorzystania zasobów nie jest wyższy niż w specyfikacji. Okresowo informuje też o stanie kontenerów i węzła. Menedżer węzła udostępnia lokalne usługi wszystkim kontenerom działającym w danym węźle. Usługa *agregacji dziennika* pozwala przysyłać informacje ze standardowego wyjścia i standardowego wyjścia błędów (stdout i stderr) do systemu HDFS. Menedżer węzła można ustawić tak, aby uruchamiał zestaw konfigurowalnych *usług pomocniczych*. Przykładowo, faza przestawiania z modelu MR jest udostępniana jako usługa menedżera węzła. Kontener, w którym działa zadanie mapowania, generuje dane wyjściowe i zapisuje je na dysku lokalnym. Te dane wyjściowe są dostępne dla reducerów z danej pracy dzięki usłudze przestawiania działającej w określonym węźle.

**Odporność na błędy i dostępność.** Menedżer zasobów (MZ) na platformie YARN jest pojedynczym punktem podatności na awarie. Po restarcie MZ może odtworzyć swój stan na podstawie trwałego magazynu danych. Zamyka wtedy wszystkie kontenery w klastrze i ponownie uruchamia każdego zarządcę aplikacji. Obecnie trwają prace nad udostępnieniem aktywnego i pasywnego trybu MZ. Niepowodzenie zarządcy aplikacji nie jest katastrofą, ponieważ wpływa tylko na jedną aplikację. Zarządca odpowiada za odtworzenie stanu danej aplikacji. Przykładowo, zarządca aplikacji w modelu MR odtwarza ukończone zadanie i ponownie uruchamia wykonywane zadania.

Niepowodzenie kontenera z powodu problemów z węzłem lub kodu aplikacji jest wykrywane przez platformę i zgłaszane zarządcy aplikacji. To zarządca aplikacji odpowiada za odtworzenie stanu po awarii.

### 25.5.3. Inne platformy w YARN

Opisana wcześniej architektura YARN umożliwiła rozwijanie innych platform aplikacji, a także obsługę innych modeli programowania, zapewniających dodatkowe usługi we współużytkowanym klastrze z Hadoopem. Oto lista wybranych platform dostępnych w YARN w czasie, gdy powstaje ta książka.

**Apache Tez.** Tez to rozszerzalna platforma rozwijana w firmie Hortonworks i przeznaczona do budowania wysoce wydajnych aplikacji działających w YARN. Te aplikacje mają obsługiwać duże zbiory danych (mierzone w petabajtach). Tez umożliwia użytkownikom zapisywanie przepływów pracy jako skierowanych grafów acyklicznych (ang. *directed acyclic graph* — DAG) obejmujących zadania. Prace są modelowane jako grafy DAG, w których wierzchołki są zadaniami lub operacjami, a krawędzie reprezentują zależności między operacjami lub przepływ danych. Tez obsługuje standardowe wzorce przepływu danych: potoki, roz-

dziel-zbierz i rozgłaszanie. Użytkownicy mogą w grafie DAG określić poziom współbieżności, a także aspekty przełączania awaryjnego (np. to, czy zapisywać dane wyjściowe z zadania w trwałej pamięci, czy ponownie je obliczać). Graf DAG może być modyfikowany w czasie wykonywania programu na podstawie stanu pracy i klastra. Model oparty na grafach DAG jest bardziej naturalnie dostosowany (w porównaniu z wykonywaniem prac modelu MapReduce) do skryptów w języku Pig i planów fizycznych zapytań w języku SQL. Hive i Pig obsługują obecnie tryb używający platformy Tez. W obu przypadkach korzyściami są prostsze plany wykonania zapytań i znaczny wzrost wydajności. Często przytaczaną optymalizacją wydajności jest wzorzec mapowanie-redukcja-redukcja. Zapytanie w języku SQL obejmujące złączenie, po którym następuje grupowanie, tradycyjnie było przekształcane na dwie prace modelu MR: jedną dla złączenia i jedną dla grupowania. W pierwszej pracy modelu MR dane wyjściowe ze złączenia są zapisywane w systemie HDFS, a następnie ponownie wczytywane w fazie mapowania w drugiej pracy modelu MR na potrzeby pracy wykonującej grupowanie. Na platformie Tez można uniknąć dodatkowego zapisu i odczytu w systemie HDFS; wystarczy utworzyć w grafie DAG wierzchołek reprezentujący złączenie, który strumieniowo przesyła wynikowe wiersze do wierzchołka reprezentującego grupowanie.

**Apache Giraph.** Apache Giraph to otwarta implementacja systemu Pregel firmy Google<sup>18</sup>. Jest to działający na dużą skalę system przetwarzania grafów służący do wykonywania algorytmu PageRank (definicję algorytmu PageRank znajdziesz w punkcie 27.7.3). Pregel jest oparty na masowym przetwarzaniu synchronicznym (ang. *bulk synchronous processing* — BSP)<sup>19</sup>. W systemie Giraph wzbogacono Pregel o kilka funkcji, w tym o agregatory z obsługą shardingu (sharding to odmiana podziału na partycje; patrz rozdział 24.) i dane wejściowe oparte na krawędziach. System Giraph dla Hadoopa 1 działał jako praca modelu MR, co nie było najlepszym rozwiązaniem. Wymagało to uruchamiania długich prac z samym etapem mapowania. W architekturze YARN system Giraph działa w modelu przetwarzania iteracyjnego. Obecnie Giraph jest używany w firmie Facebook do analizowania grafu użytkowników w sieci społecznościowej. Wierzchołkami są tu użytkownicy, a krawędziami — powiązania między nimi. Obecnie liczba użytkowników wynosi ok. 1,3 miliarda.

**Hoya: HBase w YARN.** Projekt Hoya (ang. *HBase on YARN*) firmy Hortonworks pozwala tworzyć elastyczne klastry z systemem HBase działające w YARN. Ma on zapewniać większą elastyczność i lepsze wykorzystanie zasobów klastra. System HBase opisaliśmy w podrödziale 24.5. Jest to rozproszony, otwarty, nierelacyjny system baz danych zarządzający tabelami z miliardami wierszy i milionami kolumn. HBase jest wzorowany na systemie BigTable firmy Google<sup>20</sup>, ale został zaimplementowany z użyciem Hadoopa i systemu HDFS. System Hoya jest rozwijany, aby móc tworzyć na żądanie klastry z systemem HBase, w których można używać różnych wersji HBase. Każdą instancję systemu HBase można skonfigurować niezależnie od innych. Zarządca aplikacji dla systemu Hoya uruchamia lokalnie zarządcę aplikacji dla systemu HBase, a także żąda od menedżera zasobów na platformie YARN zestawu kontenerów do uruchomienia w klastrze serwerów regionów systemu HBase. Serwery regionów systemu HBase to jego procesy robocze. Każda rodzina kolumn (przypominająca zbiór kolumn z tabeli relacyjnej) jest rozproszona w zbiorze serwerów regionów. Dzięki tym serwerom w klastrze można na żądanie uruchomić jedną lub kilka instancji systemu HBase. Klastry są elastyczne i można je na żądanie powiększać lub zmniejszać.

<sup>18</sup> System Pregel jest opisany w pracy Malewicz i in. (2010).

<sup>19</sup> BSP to model projektowania algorytmów równoległych zaproponowany w pracy Valianta (1990).

<sup>20</sup> System BigTable jest opisany w pracy Changa i in. (2006).

Trzy przedstawione przykłady aplikacji na platformę YARN powinny dać Czytelnikom obraz możliwości, jakie pojawiły się dzięki oddzieleniu zarządzania zasobami od zarządzania aplikacją w ogólnej architekturze YARN obejmującej Hadoopa i model MapReduce.

## 25.6. Ogólne omówienie

Do tej pory opisywaliśmy rozwój technologii big data, jaki nastąpił w latach 2004 – 2014. Skupiliśmy się na Hadoopie 1 i platformie YARN (nazywanej też Hadoop 2 lub MapReduce 2). W tym podrozdziale musimy zacząć od pewnego zastrzeżenia: rozwijanych jest wiele projektów — czy to jako otwarte rozwiązania Apache, czy to w firmach nastawionych na rozwój produktów z tego obszaru (np.: Hortonworks, Cloudera, MapR), czy to w licznych prywatnych start-upach. Także AMPLab z Uniwersytetu Kalifornijskiego i inne instytucje akademickie wnoszą duży wkład w rozwój technologii, których nie mogliśmy obszernie opisać. Ponadto jest wiele kwestii związanych z chmurą (np. używanie modelu MapReduce i hurtowni danych w chmurze), których nie omówiliśmy. Dlatego w tym miejscu przedstawiamy kilka ogólnych zagadnień, o których warto wspomnieć w kontekście szczegółowych opisów z wcześniejszych fragmentów tego rozdziału. Prezentujemy kwestie związane z rywalizacją między tradycyjnym sposobem tworzenia wysoce wydajnych aplikacji za pomocą równoległych relacyjnych SZBD a technologiami opartymi na platformach Hadoop i YARN. Dalej omawiamy kilka zagadnień dotyczących uzupełniania się technologii z obszarów big data i chmury. Pokrótkę opisujemy tematy lokalności danych i optymalizacji ściśle powiązane z chmurami danych i chmurami obliczeniowymi. Przedstawiamy też YARN jako platformę usług z obszaru danych oraz trwające próby okiełznania big data na potrzeby analiz. Na zakończenie opisujemy wybrane trudności, z jakimi zмага się cała społeczność zainteresowana big data.

### 25.6.1. Hadoop i MapReduce a równoległe relacyjne SZBD

Zespół ekspertów od danych, w tym Abadi, DeWitt, Madden i Stonebracker, przeprowadził staranne badania porównawcze na kilku równoległych systemach baz danych oraz otwartej wersji Hadoopa i modelu MapReduce (patrz np. Pavlo i in., 2009). Ci eksperci zmierzili wydajność obu podejść za pomocą tych samych testów porównawczych, używając klastra ze 100 węzłami. Przyznali, że wczytywanie i dostrajanie równoległej bazy danych trwało dłużej w porównaniu do uruchamiania modelu MapReduce, jednak wydajność równoległego SZBD okazała się „uderzająco wyższa”. Wymienimy tu obszary porównywane przez ekspertów w tym badaniu i spróbujemy pokazać postępy dokonane w SZBD i Hadoopie od czasu testów.

**Wydajność.** W swojej pracy Pavlo i in. stwierdzili, że równoległe SZBD były od trzech do sześciu razy szybsze od modelu MapReduce. W tej pracy podano wiele powodów, dla których SZBD zapewniał wyższą wydajność. Oto niektóre z tych przyczyn: (1) indeksowanie z użyciem B<sup>+</sup>-drzew, co przyspiesza selekcję i filtrowanie; (2) nowatorskie metody składowania danych (np. model kolumnowy ma określone zalety); (3) techniki umożliwiające bezpośrednią pracę na skompresowanych danych; (4) techniki optymalizacji zapytań równoległych często stosowane w równoległych SZBD.

Od czasu porównania opisanego w Pavlo i in., gdzie używano Hadoopa 0.19, w ekosystemie Hadoopa dokonano znacznych postępów w środowisku uruchomieniowym MapReduce, formatach składowania danych i mechanizmach planowania w obszarze szeregowania prac i optymalizowania złożonych przepływów danych. *ORC* i *Parquet* to zaawansowane kolumnowe formaty plików, gdzie stosowane są agresywne techniki kompresji, możliwe jest przenoszenie predykatów do warstwy składowania danych i można przetwarzać zagregowane zapytania bez skanowania danych. Pokróćce omówimy tu ulepszenia w systemie HDFS i modelu MapReduce. W systemie Apache Hive wprowadzono znaczne usprawnienia w środowisku uruchomieniowym i w optymalizacji kosztowej złożonych zapytań w języku SQL. W wyniku prób zmodyfikowania trybu pracy Hadoopa ze wsadowego na interaktywne przetwarzanie zapytań w czasie rzeczywistym firma Hortonworks (2014) poinformowała o wzroście wydajności obsługi zapytań o rzędy wielkości w ramach testów porównawczych TPC-DS (związanych ze wspomaganie podejmowania decyzji). Produkt Impala firmy Cloudera używa otwartego kolumnowego formatu danych Parquet i, zdaniem producenta, działa porównywalnie z tradycyjnymi relacyjnymi SZBD (Cloudera, 2014).

**Niższe koszty początkowe.** Hadoop wciąż jest tańszy. Z kilkoma wyjątkami jest przede wszystkim otwartą platformą. YARN, Hive i Spark są rozwijane jako projekty organizacji Apache i dostępne w formie bezpłatnych pakietów do pobrania.

**Obsługa danych niestrukuralnych i częściowo strukturalnych.** Model MR wczytuje dane, stosując do nich definicję schematu. Pozwala to na obsługę częściowo strukturalnych zbiorów danych, takich jak dokumenty CSV, JSON i XML. Proces wczytywania danych w systemach Hadoop i MapReduce jest stosunkowo mało kosztowny, jednak obsługa danych niestrukuralnych w relacyjnych SZBD zdecydowanie się poprawia. System PostgreSQL obsługuje obecnie pary klucz-wartość i format JSON, a większość relacyjnych SZBD obsługuje format XML. Z drugiej strony, jednym z powodów wzrostu wydajności Hadoopa było zastosowanie specjalnych formatów danych, takich jak *ORC* (ang. *Optimized Row Columnar*) i *Parquet* (inny otwarty format kolumnowy). Ta ostatnia cecha w niedalekiej przyszłości zapewne przestanie odróżniać relacyjne SZBD od systemów opartych na Hadoopie, ponieważ także w relacyjnych SZBD może zostać dodana obsługa specjalnych formatów danych.

**Obsługa języków wyższego poziomu.** Obsługa języka SQL była cechą zachęcającą do wyboru relacyjnych SZBD, gdy potrzebne były złożone zapytania analityczne. Jednak wcześniej wspomniano już, że w języku HiveQL z systemu Hive wprowadzono liczne elementy języka SQL, w tym grupowanie i agregację, a także podzapytania zagnieżdżone i wiele funkcji przydatnych w hurtowniach danych. System Hive 0.13 potrafi wykonać ok. 50 zapytań z testów porównawczych TPC-DS bez żadnych ręcznych modyfikacji. Pojawiają się też nowe biblioteki funkcji z obszaru uczenia maszynowego (np. biblioteka funkcji z serwisu <http://madlib.net> działa w tradycyjnych relacyjnych SZBD, takich jak PostgreSQL, a także w PHD — dystrybucji bazy Hadoopa opracowanej przez firmę Pivotal). Zdaniem firmy Pivotal jej produkt HAWQ jest najnowszym i oferującym największe możliwości równoległym silnikiem języka SQL oraz łączy zalety języka SQL i Hadoopa. Ponadto opisana wcześniej architektura YARN z obsługą wtyczek upraszcza rozszerzanie platform o nowe komponenty i funkcje. Pig i Hive mogą być rozszerzane za pomocą funkcji definiowanych przez użytkownika. W architekturze YARN dostępnych jest kilka usług z obszaru danych, np. Revolution R i Apache Mahout (uczenie maszynowe) oraz Giraph (przetwarzanie grafów). Wiele tradycyjnych SZBD działa

obecnie na platformie YARN. Są to np. platforma analityczna Vortex firmy Actian<sup>21</sup> i interfejs Big SQL 3.0 firmy IBM<sup>22</sup>.

**Odporność na usterki.** Jeśli chodzi o odporność na usterki, nadal zdecydowaną przewagę mają systemy oparte na modelu MR. Grupa autorów pracy Pavlo i in. (2009) także przyznaje, że „model MR lepiej sobie radzi z minimalizowaniem ilości pracy traczonej w sytuacji awarii sprzętu”. Autorzy piszą, że dzieje się to kosztem materializowania plików pośrednich między etapami mapowania i redukcji. Jednak ponieważ Hadoop zaczyna obsługiwać bardzo złożone przepływy danych (np. na platformie Apache Tez) i niskie opóźnienie nie jest wymagane, użytkownicy mogą zrezygnować z wydajności na rzecz odporności na usterki. Przykładowo, narzędzie Apache Spark można skonfigurować tak, aby zbiory danych RDD (ang. *Resilient Distributed Dataset*)<sup>23</sup> były materializowane albo na dysku, albo w pamięci, albo nawet ponownie obliczane na podstawie danych wejściowych.

Z tej dyskusji wynika, że choć model MR początkowo miał służyć do przetwarzania wsadowego, posiadał wydajność niższą od tradycyjnych równoległych relacyjnych SZBD, jeśli chodzi o obsługę zapytań interaktywnych. Wykazali to Pavlo i in. (2009). Jednak możliwości obu typów rozwiązań stały się dużo bardziej podobne. Siły rynkowe, np. inwestorzy finansujący nowe start-upy, wymagają, by silnik języka SQL w nowych aplikacjach radził sobie z bardzo dużymi częściowo strukturalnymi zbiorami danych. Z drugiej strony, zainteresowanie i zaangażowanie społeczności naukowej doprowadziło do znacznej poprawy możliwości Hadoopa w zakresie obsługi tradycyjnego, analitycznego obciążenia roboczego. Nadal pozostaje jednak dużo do zrobienia w obszarach wymienionych w pracy Pavlo i in. (2009) — w środowisku uruchomieniowym, w planowaniu i optymalizacji oraz w funkcjach analitycznych.

## 25.6.2. Big data w chmurach obliczeniowych

Rozwój chmur obliczeniowych i big data przez ponad dekadę odbywały się równolegle. W tym miejscu nie da się szczegółowo opisać chmur obliczeniowych. Podamy jednak przekonujące powody określające, dlaczego technologia big data jest w pewnym sensie zależna od chmury, jeśli ma się dalej rozwijać, a nawet przetrwać.

- Chmura zapewnia dużą swobodę w obszarze zarządzania zasobami. Umożliwia łatwe i niemal natychmiastowe skalowanie poziome, czyli dodawanie nowych węzłów lub zasobów, oraz skalowanie pionowe, czyli dodawanie zasobów w węzłach systemu, a nawet zmniejszanie ilości zasobów.
- Zasoby są wymienialne. Ta cecha w połączeniu z rozproszonym oprogramowaniem zapewnia sprawny ekosystem, w którym można łatwo radzić sobie z awariami i gdzie wirtualne instancje obliczeniowe mogą działać bez przeszkód. Kosztem kilkuset dolarów można przeprowadzać operacje eksplorowania danych obejmujące pełne skanowanie terabajtowych baz danych, a także analizować wielkie witryny internetowe obejmujące miliony stron.

---

<sup>21</sup> Aktualny opis znajdziesz na stronie <http://www.actian.com/about-us/blog/sql-hadoop-real-deal/>.

<sup>22</sup> Aktualne omówienie zawiera prezentacja na stronie [http://www.slideshare.net/Hadoop\\_Summit/w-325p230-azubirgrayatv4](http://www.slideshare.net/Hadoop_Summit/w-325p230-azubirgrayatv4).

<sup>23</sup> Patrz Zaharia i in. (2012).



- Nierzadko się zdarza, że projekty z obszaru big data działają nieprzewidywalnie lub mają wysokie szczytowe zapotrzebowanie na moc obliczeniową i pamięć. W takich projektach problemem jest zapewnienie realizacji zapotrzebowania szczytowego w momencie, gdy ono występuje, a nie stale. Jednocześnie jednostki biznesowe oczekują szybkich, tanich i godnych zaufania produktów oraz wyników prac nad projektami. Chmury są idealnym rozwiązaniem pozwalającym zrealizować te sprzeczne wymagania.
- Oto częsta sytuacja, w jakiej chmury i big data są ze sobą powiązane. Dane są przesyłane do systemu składowania danych w chmurze (lub zbierane w takim systemie), np. w S3 Amazona, na potrzeby zapisu plików dziennika lub eksportowania danych tekstowych. Można też używać adapterów bazodanowych, aby uzyskać dostęp do danych z bazy w chmurze. Do analizowania surowych danych (których źródłem może być chmura) służą platformy przetwarzania danych, np.: Pig, Hive i MapReduce, które opisano w podrozdziale 25.4.
- W projektach i start-upach z dziedziny big data bardzo korzystne jest stosowanie usług składowania danych w chmurze, ponieważ środki można przeznaczyć na działalność operacyjną zamiast na inwestycje. To doskonałe podejście, ponieważ nie wymaga nakładów kapitałowych i ponoszenia ryzyka. Składowanie danych w chmurze to stabilne i skalowalne rozwiązanie o jakości, jaką bardzo trudno jest uzyskać w inny sposób.
- Usługi i zasoby w chmurach są rozproszone globalnie. Zapewnia to wysoką dostępność i trwałość osiągalną w innych warunkach wyłącznie dla największych firm.

**Połączenie chmury i big data w firmie Netflix<sup>24</sup>.** Netflix to duża organizacja o bardzo zyskowym modelu biznesowym oraz niezwykle ekonomicznych i niezawodnych usługach świadczonych klientom. Przesyła strumieniowo filmy milionom odbiorców dzięki wysoce wydajnemu systemowi informatycznemu i hurtowni danych. Netflix jako platformy przetwarzania i analiz danych używa S3 Amazona, a nie systemu HDFS. Wynika to z kilku powodów. Obecnie Netflix korzysta z opracowanej przez Amazon dystrybucji EMR (ang. *Elastic MapReduce*) Hadoopa. Netflix podaje, że głównym powodem tych wyborów jest to, iż S3 ma zapewniać trwałość obiektów na poziomie 99,999999999% i ich dostępność na poziomie 99,99% w skali roku, a także jest odporny na jednoczesną utratę danych w dwóch kompleksach. S3 udostępnia wersjonowanie komórek (ang. *buckets*), dzięki czemu Netflix może odzyskać przypadkowo usunięte dane. Elastyczność systemu S3 zapewnia firmie Netflix praktycznie nieograniczoną pojemność magazynu danych, co pozwoliło na wzrost ilości danych z kilkuset terabajtów do petabajtów bez żadnych trudności lub wcześniejszego planowania. Używanie S3 jako hurtowni danych pozwala firmie Netflix uruchomić wiele klastrów Hadoopa, które są odporne na usterki i radzą sobie z przeciążeniem. Menedżerowie z firmy Netflix mówią, że nie martwią się o przenoszenie lub utratę danych w trakcie powiększania lub zmniejszania hurtowni danych. Choć należące do Netflixu klastry produkcyjne i obsługujące zapytania są klastrami działającymi długookresowo w chmurze, można je traktować tak, jakby były tymczasowe. Gdy klastr przestaje funkcjonować, Netflix może go w prosty sposób zastąpić innym klastrem o identycznej pojemności, który może działać w innej strefie geograficznej. Zajmuje to kilka minut i nie powoduje żadnej utraty danych.

---

<sup>24</sup> Na podstawie tekstu <http://techblog.netflix.com/2013/01/hadoop-platform-as-service-in-cloud.html>.



### 25.6.3. Problemy z lokalnością danych i optymalizacja zasobów w aplikacjach z obszaru big data działających w chmurze

Rosnące zainteresowanie przetwarzaniem w chmurze w połączeniu z dużymi wymogami technologii big data oznacza, że centra danych muszą być coraz bardziej ekonomiczne i dostosowane do klientów. Ponadto infrastruktura chmur często nie jest zaprojektowana pod kątem obsługi danych w skali niezbędnej do przeprowadzanych obecnie analiz. Dostawcy chmur zmagają się z trudnymi wyzwaniami w obszarze zarządzania zasobami i planowania przepustowości, aby dostosować się do aplikacji z dziedziny big data.

Obciążenie generowane w sieci przez liczne aplikacje z obszaru big data, w tym przez platformę Hadoop i model MapReduce, w centrach danych zasługuje na specjalną uwagę z powodu dużych ilości danych, jakie mogą być generowane w czasie wykonywania prac. Przykładowo, w pracy w modelu MR każde zadanie redukcji musi wczytywać dane wyjściowe wszystkich zadań mapowania, a nagły skok ilości ruchu w sieci może znacznie obniżyć wydajność chmury. Ponadto gdy dane są zlokalizowane w jednej infrastrukturze (np. w chmurze składowania danych, takiej jak S3 Amazona), a przetwarzane w chmurze obliczeniowej (np. w EC2 Amazona), wydajność prac spada z uwagi na znaczne opóźnienia spowodowane wczytywaniem danych.

W projektach badawczych zaproponowano<sup>25</sup> automatycznie konfigurowaną i opartą na lokalności platformę zarządzania danymi i maszynami wirtualnymi działającą w modelu składowania i obliczeń. Ta platforma umożliwia pracom modelu MapReduce dostęp do większości danych lokalnie lub w pobliskich węzłach. Dotyczy to wszystkich danych wejściowych, wyjściowych i pośrednich generowanych następnie na etapach mapowania i redukcji takich prac. Tego typu platformy kategoryzują prace za pomocą klasyfikatora uwzględniającego wielkość danych na cztery klasy. Następnie z uwzględnieniem lokalności udostępniane są wirtualne klastry z obsługą modelu MapReduce, co pozwala na wydajne wiązanie węzłów ze sobą i przydział zasobów w maszynach wirtualnych obsługujących model MR, aby zmniejszyć w sieci odległość między węzłami składowania danych a węzłami obliczeniowymi na etapach mapowania i redukcji.

Ostatnio pokazano, że pamięć podręczna pozwala poprawić wydajność prac modelu MapReduce przy różnym obciążeniu roboczym<sup>26</sup>. Platforma PACMAN zapewnia obsługę pamięci podręcznej przechowywanej w pamięci głównej, a system MixApart obsługuje dyskową pamięć podręczną, gdy dane są zapisywane na serwerze danych firmy w tej samej lokalizacji. Techniki wykorzystujące pamięć podręczną zapewniają swobodę, ponieważ dane są zapisywane w odrębnej infrastrukturze składowania danych, umożliwiając wstępne pobieranie najważniejszych danych i umieszczanie ich w pamięci podręcznej. W opublikowanej niedawno pracy<sup>27</sup> uwzględniono problem z zapisem w pamięci podręcznej big data w sytuacji, gdy trzeba zachować prywatność; dane przechowywane w zaszyfrowanej formie w publicznej chmurze trzeba wtedy przetwarzać w odrębnej, bezpiecznej lokalizacji należącej do firmy.

---

<sup>25</sup> Patrz Palanisamy i in. (2011).

<sup>26</sup> Platformę PACMAN opisali Ananthanarayanan i in. (2012), a system MixApart — Mihailescu i in. (2013).

<sup>27</sup> Patrz Palanisamy i in. (2014a).

Obok problemu lokalności danych jednym z najtrudniejszych wyzwań dla dostawców chmury jest zapewnianie dla prac optymalnej ilości zasobów w wirtualnych klastrach przy jednoczesnej minimalizacji ogólnego kosztu użytkowania centrum danych działającego w chmurze.

Ważnym aspektem optymalizacji zasobów w chmurze jest analizowanie sytuacji globalnie, na poziomie wszystkich prac z chmury, zamiast optymalizowania zasobów dla konkretnych prac. Dobrym przykładem globalnie optymalizowanego systemu zarządzanego przez chmurę jest nowy system BigQuery Google'a<sup>28</sup>. Umożliwia on firmie uruchamianie na bardzo dużych zbiorach danych (obejmujących nawet miliardy wierszy) za pomocą interfejsu podobnego do Excela zapytań podobnych jak w języku SQL. W usłudze BigQuery klienci przesyłają tylko zapytania dotyczące bardzo dużych zbiorów danych, a system chmury w inteligentny sposób zarządza zasobami na potrzeby zapytań podobnych do tych z języka SQL. Podobnie model optymalizacji zasobów Cura<sup>29</sup> zaproponowany dla modelu MapReduce w chmurze optymalizuje zasoby globalnie (zamiast optymalizować zasoby dla prac lub klientów), minimalizując ogólne zużycie zasobów w chmurze.

## 25.6.4. YARN jako platforma usług z obszaru danych

Oddzielenie zarządzania zasobami od zarządzania aplikacją pozwoliło przenieść platformę Hadoop na nowy poziom. W Hadoopie 1 najważniejszy był model MapReduce. W Hadoopie 2 MapReduce to jedna z wielu platform aplikacji, jakie mogą działać w klastrze. W podrozdziale 25.5 napisano, że to podejście umożliwiło udostępnianie na platformie YARN wielu usług (z własnymi modelami programowania). Dzięki temu nie trzeba przekształcać wszystkich technik przetwarzania danych i algorytmów na zestaw prac modelu MapReduce. Obecnie MapReduce jest używany tylko do przetwarzania wsadowego, np. w procesach ETL w hurtowniach danych (patrz rozdział 29.). Nowym trendem jest traktowanie Hadoopa jako **jeziora danych** (ang. *data lake*), w którym znajduje się znaczna część danych firmy i gdzie są one przetwarzane. Tradycyjnie w systemie HDFS znajdowały się dane historyczne firmy, ponieważ system ten radzi sobie z taką ilością danych. Większość nowych źródeł danych, które w dzisiejszych wyszukiwarkach i sieciach społecznościowych pochodzą z dzienników internetowych, dzienników maszyn, strumieni kliknięć, wiadomości (np. na Twitterze) i czujników, także w dużej części jest zapisywana w systemie HDFS.

W Hadoopie 1 stosowany był model **federacyjny**. Choć warstwą składowania był system HDFS, przetwarzanie odbywało się za pomocą modelu MapReduce i innych rozwiązań. Jedną z możliwości było pobieranie danych z HDFS do odrębnych systemów działających poza klastrzem. Takie dane przekazywano do systemów grafowych, analitycznych aplikacji do uczenia maszynowego itd. Maszyny używane w klastrze Hadoopa wykorzystywano też do zupełnie innych zastosowań, np. do przetwarzania strumieni poza Hadoopem. Był to scenariusz daleki od ideału, ponieważ zasoby fizyczne trzeba było rozdzielać statycznie, a trudno było przeprowadzać migracje i aktualizować narzędzia do nowych wersji, gdy na tych samych maszynach działały różne platformy. W architekturze YARN te problemy zostały rozwiązane. Tradycyjne usługi korzystają z menedżera zasobów platformy YARN i świadczą usługi w tym samym klastrze Hadoopa, w którym znajdują się dane.

---

<sup>28</sup> System BigQuery firmy Google opisano na stronie <https://developers.google.com/bigquery/>.

<sup>29</sup> Palanisamy i in. (2014b).

Choć wielu producentów deklarowało dodanie obsługi języka SQL w Hadoopie, pozostawia ona nieco do życzenia. Niektórzy dostawcy chcą przeniesienia danych z systemu HDFS do innej bazy danych, aby móc uruchamiać zapytania w języku SQL. Inni wymagają nakładek do wczytywania danych z systemu HDFS przed wykonaniem zapytania w tym języku. Nowym trendem w relacyjnych i tradycyjnych SZBD jest traktowanie klastra YARN jako akceptowalnej platformy. Jednym z przykładów jest platforma analityczna Actian, która udostępnia SQL w Hadoopie<sup>30</sup> i według deklaracji jest kompletną i stabilną implementacją języka SQL używającą kolumnowej bazy danych Actian Vectorwise (działającej jako aplikacja w architekturze YARN). Big SQL 3.0 firmy IBM<sup>31</sup> to projekt, który umożliwia działanie istniejącego SZBD (bez zasobów współużytkowanych) tej firmy w klastrze YARN.

Apache Storm to rozproszony skalowalny system strumieniowania danych, umożliwiający użytkownikom przetwarzanie napływających w czasie rzeczywistym danych z kanałów informacyjnych. Był on używany w serwisie Twitter. Storm on YARN (<https://hortonworks.com/apache/storm/>) i SAS on YARN (<http://hortonworks.com/partner/sas/>) pozwalają używać narzędzi Storm (rozproszonej aplikacji do przetwarzania strumieniowego) i SAS (oprogramowania do analiz statystycznych) jako aplikacji dla platformy YARN. Wcześniej opisano, że Giraph i HBase Hoya to obecnie rozwijane projekty szybko integrowane z platformą YARN. Wiele systemów używa klastrów z Hadoopem do składowania danych. Są to np. usługi z obszaru strumieniowania, uczenia maszynowego, statystyki, przetwarzania grafów, OLAP i magazynów danych z parami klucz-wartość. Te usługi znacznie wykraczają poza model MapReduce. Twórcy architektury YARN stawiają sobie za cel (i obiecują) umożliwienie współistnienia takich usług w jednym klastrze i wykorzystywania lokalności danych w systemie HDFS, przy czym YARN ma zarządzać użytkowaniem zasobów klastra.

### 25.6.5. Wyzwania związane z technologiami z obszaru big data

W opublikowanym niedawno artykule<sup>32</sup> kilku ekspertów od baz danych wyraziło obawy o wyzwania związane z technologiami z dziedziny big data, gdy technologie te są używane głównie do analiz. Oto niektóre z tych obaw:

- **Niejednorodność informacji.** Niejednorodność źródeł danych ze względu na typy danych, formaty danych, reprezentację danych i semantykę jest nie do uniknięcia. Jeden z etapów w cyklu życia big data obejmuje integrowanie danych. Koszt precyzyjnej integracji powodującej umieszczenie wszystkich danych w jednej strukturze jest nieakceptowalny w większości aplikacji — np. w obszarze służby zdrowia, energetyki, transportu, planowania przestrzeni i modelowania środowiskowego. Większość algorytmów uczenia maszynowego oczekuje danych o jednorodnej strukturze. Pochodzenie danych (czyli informacje o ich źródle i właścicielach) w większości aplikacji analitycznych nie jest zachowywane. Właściwa interpretacja wyników analizy danych wymaga dużych ilości metadanych.

---

<sup>30</sup> Aktualna dokumentacja jest dostępna na stronie <http://www.actian.com/about-us/blog/sql-hadoop-real-deal/>.

<sup>31</sup> Aktualne informacje znajdziesz na stronie [http://www.slideshare.net/Hadoop\\_Summit/w-325p230-azubirigraytv4](http://www.slideshare.net/Hadoop_Summit/w-325p230-azubirigraytv4).

<sup>32</sup> Patrz Jagadish i in. (2014).

- **Prywatność i poufność.** Regulacje i prawa dotyczące ochrony poufnych informacji nie zawsze obowiązują, dlatego czasem nie są ściśle przestrzegane w trakcie analiz big data. Wymuszanie przestrzegania regulacji HIPAA w służbie zdrowia to jeden z nielicznych obszarów, w których prywatność i poufność są ściśle zapewniane. Aplikacje wykorzystujące lokalizację (np. w smartfonach i innych urządzeniach wyposażonych w GPS), dzienniki transakcji użytkowników i strumienie kliknięć rejestrujące zachowania klientów ujawniają poufne informacje. Można śledzić wzorce poruszania się i modele zakupowe użytkowników, aby ujawnić tożsamość konkretnych osób. Ponieważ opisane w tym rozdziale technologie umożliwiają obecnie zbieranie i analizowanie miliardów rekordów z danymi użytkowników, powszechne są obawy o ujawnienie danych osobowych (możliwy jest np. wyciek danych o użytkownikach z sieci społecznościowej powiązanej z innymi sieciami danych). Organizacje przechowują dane na temat klientów, użytkowników kart i pracowników, co grozi naruszeniem prywatności. Jagadish i in. (2014) pisali o potrzebie ściślejszej kontroli nad zarządzaniem prawami do danych cyfrowych, podobnej do kontroli obowiązującej w branży muzycznej.
- **Potrzeba wizualizacji i lepszych interfejsów dla ludzi.** Systemy z obszaru big data przetwarzają olbrzymie ilości danych, a wyniki analiz muszą być interpretowane przez ludzi i zrozumiałe dla nich. Trzeba uwzględnić preferencje ludzi i prezentować dane w odpowiednio przystępnej formie. Ludzie są ekspertami od wykrywania wzorców i mają świetną intuicję co do danych, które znają. Maszyny nie potrafią dorównać im pod tym względem. Możliwe powinno być zebranie kilku ludzkich ekspertów, aby mogli dzielić się wynikami analiz i je interpretować, co pozwoli lepiej zrozumieć te wyniki. Obsługiwane muszą być różne tryby wizualnej eksploracji, aby móc optymalnie wykorzystywać dane i poprawnie interpretować wyniki spoza typowego zakresu, sklasyfikowane jako wartości odstające.
- **Niespójne i niekompletne informacje.** Jest to odwieczny problem w obszarze zbierania danych i zarządzania nimi. Przyszłe systemy używające big data będą umożliwiały obsługę wielu źródeł danych przez współistniejące aplikacje, dlatego problemy spowodowane brakującymi, błędnymi i niepewnymi danymi będą się nasilać. Większa ilość i wbudowana redundancja danych w systemach odpornych na usterki mogą w pewnym zakresie kompensować brakujące wartości, sprzeczne wartości, ukryte relacje itd. Gdy dane są zbierane od zwykłych użytkowników za pomocą standardowych urządzeń i docierają w różnej postaci (jako zdjęcia, z odmienną szybkością, z różnych kierunków), trudno jest być pewnym ich jakości. Nadal musimy się dużo nauczyć o tym, jak wykorzystać dane z publicznych źródeł do skutecznego podejmowania decyzji.

Wspomniane problemy nie są niczym nowym w systemach informatycznych. Jednak duża ilość i różnorodność informacji nieodłączna od systemów big data nasila opisane trudności.

## 25.6.6. Przyszłość

YARN umożliwia firmom uruchamianie wielu usług w jednym klastrze i zarządzanie nimi. Jednak budowanie rozwiązań z obszaru danych za pomocą Hadoopa nadal stanowi poważne wyzwanie. Takie rozwiązania mogą wymagać przetwarzania ETL, uczenia maszynowego,

przetwarzania grafów i/lub generowania raportów. Choć systemy o różnych funkcjach działają w tym samym klastrze, ich modele programowania i metadane nie są jednolite. Programiści aplikacji analitycznych muszą próbować integrować takie usługi w spójne rozwiązania.

W nowoczesnym sprzęcie każdy węzeł ma dużą ilość pamięci głównej i pamięci flash. Klaster obejmuje więc obszerne zasoby w postaci pamięci głównej i pamięci flash. W ważnych innowacyjnych rozwiązaniach zademonstrowano zyski wydajności zapewniane przez **systemy danych działające w pamięci**. Przykładowo, SAP HANA to działający w pamięci kolumnowy relacyjny SZBD ze skalowaniem poziomym, który zyskuje znaczną popularność<sup>33</sup>.

Rozwijana przez firmę Databricks (<https://databricks.com/>) platforma Spark, wyodrębniona z pakietu Berkeley Data Analytics Stack jednostki AMPLab z Berkeley<sup>34</sup>, uwzględnia obie wymienione nowinki: umożliwia uruchamianie różnorodnych aplikacji w jednym klastrze i wykorzystywanie dużych ilości pamięci głównej w celu przyspieszenia generowania odpowiedzi. Matei Zaharia w ramach pracy doktorskiej pisanej na Uniwersytecie Kalifornijskim w Berkeley opracował mechanizm RDD<sup>35</sup>, który doprowadził do powstania systemu Spark. Mechanizm ten jest na tyle uniwersalny, że można go używać we wszystkich silnikach Sparka: Spark Core (przepływ danych), Spark-SQL, GraphX (przetwarzanie grafów), MLlib (uczenie maszynowe) i Spark-Streaming (przetwarzanie strumieniowe). W Sparku można np. napisać skrypt reprezentujący przepływ danych, który obejmuje wczytanie danych z systemu HDFS, przekształcenie ich za pomocą zapytania w języku Spark-SQL, przekazanie informacji do biblioteki MLlib na potrzeby analiz z obszaru uczenia maszynowego i zapisanie wyniku z powrotem w systemie HDFS<sup>36</sup>.

Zbiory danych RDD są budowane na podstawie kolekcji z języka Scala<sup>37</sup>, które potrafią się odtwarzać na podstawie danych wejściowych. Zbiory RDD można skonfigurować zależnie od tego, jak dane są rozproszone i reprezentowane. Takie zbiory można zawsze odtwarzać na bazie danych wyjściowych lub zapisywać w pamięci podręcznej na dysku lub w pamięci głównej. Reprezentacje używane w pamięci głównej przyjmują różną postać: od serializowanych obiektów Javy po wysoce zoptymalizowane formaty kolumnowe oferujące wszystkie zalety baz kolumnowych (szybkość, ilość zajmowanego miejsca, działanie w zserializowanej postaci).

Możliwości zapewniane przez ujednolicony model programowania i zbiory danych przechowywane w pamięci zapewne zostaną dodane do ekosystemu Hadoopa. System Spark już jest dostępny jako usługa w architekturze YARN (<http://spark.apache.org/docs/1.0.0/running-on-yarn.html>). Szczegółowe omawianie Sparka i powiązanych technologii z pakietu

---

<sup>33</sup> Rozmaita dokumentację systemu HANA firmy SAP znajdziesz na stronie <http://www.saphana.com/welcome>.

<sup>34</sup> Projekty jednostki AMPLab z Uniwersytetu Kalifornijskiego w Berkeley są opisane na stronie <https://amplab.cs.berkeley.edu/software/>.

<sup>35</sup> Mechanizm RDD został zaproponowany po raz pierwszy w pracy Zaharii i in. (2012).

<sup>36</sup> Przykład zastosowania Sparka przedstawiono na stronie <https://databricks.com/blog/2014/03/26/spark-sql-manipulating-structured-data-using-spark-2.html>.

<sup>37</sup> Więcej informacji na temat kolekcji w języku Scala znajdziesz na stronie <http://docs.scala-lang.org/overviews/core/architecture-of-scala-collections.html>.

Berkeley Data Analysis Stack wykracza poza zakres tej książki. Agneeswaran (2014) omawia potencjał Sparka i powiązanych produktów. Zainteresowani Czytelnicy powinni zapoznać się z tą pracą.

## 25.7. Podsumowanie

W tym rozdziale omówiliśmy technologie z obszaru big data. W raportach firm IBM i McKinsey oraz badacza Billa Franka z firmy Teradata prognozowana jest ekscytująca przyszłość takich technologii. Mają one być podstawą przyszłych aplikacji z dziedziny analityki danych i uczenia maszynowego. Szacuje się, że w nadchodzących latach takie rozwiązania pozwolą zaoszczędzić firmom miliardy dolarów.

Omówienie rozpoczęliśmy od skoncentrowania się na osiągnięciach firmy Google: systemie plików Google'a i modelu MapReduce (jest to model programowania przetwarzania rozproszonego, który skaluje się do poziomu bardzo dużych ilości danych liczonych w petabajtach). Po przedstawieniu historii rozwoju omawianych technologii i opisanu ekosystemu Hadoopa, który obejmuje dużą liczbę aktywnie opracowywanych projektów Apache'a, przeszliśmy do systemu HDFS. Zaprezentowaliśmy jego architekturę i sposób obsługi operacji na plikach. Wspomnieliśmy też o badaniach nad skalowalnością tego systemu. Dalej dokładnie opisaliśmy środowisko uruchomieniowe MapReduce. Przedstawiliśmy przykład stosowania modelu MapReduce w różnych kontekstach. Zaprezentowaliśmy też szczegółowo przykład zastosowania tego modelu do optymalizacji różnych relacyjnych algorytmów złączania. Dalej pokrótce omówiliśmy systemy Pig i Hive, udostępniające podobny do języka SQL interfejs w postaci języków Pig Latin i HiveQL w warstwie nad niskopoziomowym programowaniem w modelu MapReduce. Wymieniliśmy też zalety połączenia technologii Hadoop i MapReduce.

Hadoop i MapReduce są stale rozwijane, a w wersji 2 (nazywanej MapReduce 2 lub YARN) zmieniono założenia ich działania. W wersji 2 zarządzanie zasobami jest oddzielone od zarządzania zadaniami i pracami. Omówiliśmy powody powstania platformy YARN, jej architekturę, a także inne rozwijane platformy oparte na YARN. Te inne platformy to np.: Apache Tez (środowisko modelowania przepływu pracy), Apache Giraph (działający na dużą skalę system przetwarzania grafów oparty na systemie Pregel firmy Google) i Hoya (opracowana w firmie Hortonworks odmiana elastycznych klastrów z systemem HBase przeznaczona na platformę YARN).

Na koniec zarysowaliśmy pewne problemy związane z technologiami MapReduce i Hadoop. Pokrótce omówiliśmy badania porównawcze przeprowadzone nad tą architekturą i równoległymi SZBD. W niektórych sytuacjach jedno z tych rozwiązań jest lepsze od drugiego, a twierdzenia o wyższości równoległych SZBD w zakresie prac wsadowych stają się nieprawdziwe z powodu architektonicznych nowinek w postaci rozwiązań stosowanych na platformie YARN. Opisaliśmy też powiązania big data i chmury oraz próby wyeliminowania problemów z lokalnością danych w kontekście składowania danych w chmurze i analiz big data. Stwierdziliśmy, że YARN jest uznawana za uniwersalną platformę świadczenia usług z obszaru danych. Wymieniliśmy też wyzwania związane z tą technologią opisane w pracy grupy ekspertów od baz danych. Rozdział zakończyliśmy podsumowaniem rozwijanych obecnie projektów z dziedziny big data.



## Pytania powtórkowe

- 25.1. Czym jest analityka danych? Jaka jest jej rola w nauce i przemyśle?
- 25.2. W jaki sposób ruch związany z big data wspomaga analitykę danych?
- 25.3. Jakie ważne zagadnienia zostały poruszone w raporcie McKinsey Global Institute z 2012 r.?
- 25.4. Jak zdefiniujesz big data?
- 25.5. Jakie różne rodzaje analityki są wymienione w książce firmy IBM z 2014 r.?
- 25.6. Jakie są cztery najważniejsze cechy big data? Podaj praktyczne przykłady dotyczące każdej z nich.
- 25.7. Czym jest *wiarygodność danych*?
- 25.8. Omów chronologicznie historię rozwoju technologii MapReduce i Hadoop.
- 25.9. Opisz przepływ pracy w środowisku programowania MapReduce.
- 25.10. Podaj przykłady zastosowań modelu MapReduce.
- 25.11. Wymień podstawowe cechy prac w modelu MapReduce.
- 25.12. Jakie funkcje pełni JobTracker?
- 25.13. Jakie wersje Hadoopa istnieją?
- 25.14. Opisz własnymi słowami architekturę Hadoopa.
- 25.15. Jaką funkcję pełnią serwery główne NameNode i pomocnicze serwery NameNode w systemie HDFS?
- 25.16. Czym jest dziennik zmian w systemie HDFS? Jakie dane są w nim przechowywane?
- 25.17. Opisz mechanizm sygnałów kontrolnych w systemie HDFS.
- 25.18. Jak wygląda zarządzanie kopiami danych (replikami) w systemie HDFS?
- 25.19. Shvachko (2012) opisał wydajność systemu HDFS. Co odkrył? Czy potrafisz podać niektóre z jego wyników?
- 25.20. Jakie inne projekty są realizowane w otwartym ekosystemie Hadoopa?
- 25.21. Opisz działanie JobTrackera i TaskTrackera w modelu MapReduce.
- 25.22. Opisz ogólny przebieg pracy w modelu MapReduce.
- 25.23. Jakie są sposoby zapewniania odporności na usterki w modelu MapReduce?
- 25.24. Czym jest procedura przestawiania w modelu MapReduce?
- 25.25. Opisz działanie różnych programów szeregujących prace w modelu MapReduce.
- 25.26. Jakie różne rodzaje złączeń można zoptymalizować za pomocą modelu MapReduce?



- 25.27. Opisz, jak w modelu MapReduce wyglądają procedury złączania sortująco-scalającego, złączania z użyciem partycji, złączania n-stronnego po stronie mappera i prostego złączania n-stronnego.
- 25.28. Czym są Apache Pig i Pig Latin? Podaj przykładowe zapytanie w języku Pig Latin.
- 25.29. Wymień główne funkcje systemu Apache Hive. Jak się nazywa używany w nim wysokopoziomowy język zapytań?
- 25.30. Czym jest architektura SerDe z systemu Hive?
- 25.31. Wymień wybrane optymalizacje z systemu Hive. W jaki sposób są obsługiwane w języku SQL?
- 25.32. Wymień zalety technologii MapReduce i Hadoop.
- 25.33. Podaj uzasadnienie przejścia od wersji Hadoop 1 do Hadoop 2 (YARN).
- 25.34. Opisz na ogólnym poziomie architekturę YARN.
- 25.35. Jak działa menedżer zasobów w architekturze YARN?
- 25.36. Czym są Apache Tez, Apache Giraph i Hoya?
- 25.37. Porównaj równoległe relacyjne SZBD z systemami opartymi na modelu MapReduce i Hadoopie.
- 25.38. W jaki sposób big data i chmura się uzupełniają?
- 25.39. Jakie problemy z lokalnością danych są związane z aplikacjami z obszaru big data przechowującymi dane w chmurze?
- 25.40. Jakie usługi (obok modelu MapReduce) potrafi udostępniać YARN?
- 25.41. Wymień wybrane wyzwania, z jakimi mierzą się obecnie twórcy technologii z obszaru big data.
- 25.42. Omów zbiory danych RDD.
- 25.43. Dowiedz się więcej o rozwijanych obecnie projektach takich jak Spark, Mesos, Shark i BlinkDB. Jak są one powiązane z pakietem Berkeley Data Analysis Stack?

## Wybrane publikacje

Omówione w tym rozdziale technologie z obszaru big data powstały w większości w ciągu ostatnich 10 lat. Początkiem tego trendu były przełomowe prace w firmie Google, w tym nad systemem plików Google'a (Ghemawat, Gobioff i Leung, 2003) i modelem programowania MapReduce (Dean i Ghemawat, 2004). System Nutch (rozwijany później w Yahoo!) był prekursorem technologii Hadoop, a obecnie jest otwartym projektem Apache'a (<http://nutch.apache.org>). System BigTable Google'a (Fay Chang i in., 2006) to rozproszony skalowalny system składowania danych służący do zarządzania danymi strukturalnymi o wielkości petabajtów i zapisanymi w tysiącach standardowych serwerów.

Nie da się podać jednej konkretnej publikacji jako „tej najważniejszej” pracy dotyczącej Hadoopa. W ostatnim dziesięcioleciu opublikowano wiele badań nad modelem MapReduce i Hadoopem. Tu wymienimy tylko najważniejsze dokonania z tej dziedziny. Schvachko

(2012) opisał ograniczenia systemu HDFS. Praca Afratiego i Ullmana (2010) jest dobrym przykładem wykorzystania modelu MapReduce w różnych kontekstach i zastosowaniach. Autorzy radzą, jak zoptymalizować relacyjne złączenia w tym modelu. Olston i in. (2008) opisują system Pig i wprowadzają wysokopoziomowy język programowania Pig Latin. Thusoo i in. (2010) omawiają system Hive jako działającą na skalę petabajtów hurtownię danych opartą na Hadoopie. System Pregel z Google'a, przeznaczony do przetwarzania grafów na dużą skalę, jest opisany w pracy Malewicz i in. (2010). Wykorzystywany jest w nim model przetwarzania równoległego BSP, zaproponowany po raz pierwszy przez Valianta (1990). W Pavlo i in. (2009) grupa ekspertów od baz danych porównuje dwa równoległe relacyjne SZBD z Hadoopem i modelem MapReduce. Wykazano, że w niektórych warunkach równoległe SZBD mogą działać lepiej. Wyniki tych badań nie są jednak w pełni miarodajne, ponieważ w Hadoopie 2 (YARN) wprowadzono znaczne usprawnienia w zakresie wydajności. Zbiory danych RDD używane do obliczeń w pamięci głównej w klastrach są istotą systemu Spark z pakietu Berkeley (Zaharia i in., 2013). W nowszej pracy Jagadisha i in. (2014) przedstawiona jest opinia grupy ekspertów od baz danych dotycząca wyzwań stojących przed twórcami technologii z obszaru big data.

Podstawowym źródłem wiedzy dla programistów aplikacji dla systemu Hadoop jest książka *Hadoop: The Definitive Guide* Toma White'a (2012), obecnie dostępna w trzecim wydaniu. W książce założyciela projektu YARN, Aruna Murthy'ego, i Vavilapalliego (2014) opisano, w jaki sposób YARN zwiększa skalowalność i poziom wykorzystania zasobów klastra, umożliwia stosowanie nowych modeli programowania i usług, a także rozszerza zakres zastosowań poza aplikacje wsadowe i Javę. Agneeswaran (2014) opisuje wyjście poza Hadoopa oraz przedstawia zastosowanie pakietu Berkeley Data Analysis Stack do analiz w czasie rzeczywistym i uczenia maszynowego. Ten pakiet obejmuje narzędzia Spark, Mesos i Shark. Autor opisuje też narzędzie Storm — złożony silnik przetwarzania zdarzeń opracowany w firmie Twitter i powszechnie używany w branży do przetwarzania i analizowania danych w czasie rzeczywistym.

Serwis wiki dotyczący Hadoopa jest dostępny na stronie <http://hadoop.apache.org>. W ramach organizacji Apache rozwijanych jest wiele otwartych projektów z obszaru big data, np.: Hive, Pig, Oozie, Sqoop, Storm i HBase. Aktualne informacje na ich temat znajdziesz w dokumentacji w witrynach i serwisach wiki projektów Apache'a. Firmy Cloudera, MapR i Hortonworks udostępniają w swoich witrynach dokumentację na temat własnych dystrybucji technologii związanych z modelem MapReduce i Hadoopem. Jednostka Berkeley AMPLab (<https://amplab.cs.berkeley.edu/>) udostępnia dokumentację na temat pakietu Berkeley Data Analysis Stack, dotyczącą m.in. rozwijanych obecnie projektów, takich jak GraphX, MLbase i BlinkDB.

Dostępne są dobre źródła wiedzy opisujące korzyści, jakie mogą przynieść technologie z obszaru big data i zarządzania danymi na dużą skalę. Bill Franks (2012) pisze, jak wykorzystać takie technologie do zaawansowanych analiz. Przedstawia też przemyślenia, które pomogą praktykom podejmować lepsze decyzje. Schmarzo (2013) wyjaśnia, w jaki sposób analizy big data mogą pomóc firmom. Dietrich i in. (2014) opisują, jak firma IBM wykorzystwała możliwości analiz big data do różnych zastosowań na całym świecie. W książce opublikowanej przez McKinsey Global Institute (2012) przedstawiono strategiczne spojrzenie na technologie z obszaru big data z naciskiem na produktywność, konkurencyjność i rozwój.

Jednocześnie dokonywane są postępy w technologiach związanych z chmurą, których nie mogliśmy szczegółowo opisać w tym rozdziale. Odsyłamy Czytelników do nowych książek na temat przetwarzania w chmurze. Erl i in. (2013) przedstawiają modele, architektury i praktyki biznesowe oraz opisują na praktycznych przykładach, jak dojrzewały technologie z tego obszaru. Kavis (2014) pokazuje różne modele usług, w tym SaaS (ang. *software as a service*), Paas (ang. *platform as a service*) i Iaas (ang. *infrastructure as a service*). Bahga i Madisetti (2013) przedstawiają praktyczne wprowadzenie do przetwarzania w chmurze. Wyjaśniają, jak pisać aplikacje działające na różnych platformach udostępniających chmurę, takich jak Amazon Web Services (AWS), Google Cloud i Microsoft Windows Azure.

# XI

---

## Zaawansowane modele, systemy i zastosowania baz danych



## Rozszerzone modele danych: wprowadzenie do aktywnych, czasowych, przestrzennych, multimedialnych i dedukcyjnych baz danych

W miarę jak systemy baz danych się rozwijały, użytkownicy wymagali od nich dodatkowych funkcji, które pozwoliłyby na łatwiejsze tworzenie bardziej zaawansowanych i skomplikowanych aplikacji. Obiektowe oraz obiektowo-relacyjne systemy baz danych posiadają mechanizmy pozwalające użytkownikom na rozszerzanie swoich systemów poprzez określanie dodatkowych abstrakcyjnych typów danych odpowiednich dla ich zastosowań. Użyteczne jednakże byłoby znalezienie pewnych wspólnych cech dla niektórych aplikacji i stworzenie modeli, które dostarczałyby gotowych rozwiązań odpowiadających tym cechom. Dodatkowo, mogą być też zaimplementowane specjalne struktury zapisu i metody indeksowania, co poprawiłoby wydajność baz danych korzystających ze znalezionych wspólnych cech. Cechy te mogłyby być zaimplementowane jako biblioteki klas lub abstrakcyjne typy danych, i sprzedawane niezależnie od podstawowego pakietu SZBD. Takie opcjonalne moduły są dostępne jako **DataBlades** (dla serwerów Informix) i **Cartridges** (dla systemów Oracle). Użytkownicy mogą bezpośrednio korzystać z tych dodatkowych modułów (o ile są odpowiednie dla ich zastosowań) bez potrzeby projektowania, implementowania i programowania wspólnych funkcji zawartych w dodatkowych modułach.

Niniejszy rozdział wprowadza pojęcia bazodanowe związane z pewnymi wspólnymi cechami potrzebnymi zaawansowanym aplikacjom, które stają się coraz szerzej używane. Do cech, które omówimy, zaliczają się *aktywne reguły* używane w aktywnych bazach danych, *pojęcia czasowe* związane z czasowymi aplikacjami bazodanowymi oraz niektóre kwestie związane z *przestrzennymi i multimedialnymi bazami danych*. Omówimy również *dedukcyjne bazy danych*. Należy zauważyć, że każdy z wymienionych tematów jest bardzo rozległy i tutaj przedstawione zostaną jedynie podstawowe wprowadzenia. W zasadzie, każdy z tych tematów mógłby wystarczyć jako samodzielny temat na kompletną książkę.

W podrozdziale 26.1 wprowadzimy temat aktywnych baz danych, które oferują dodatkową funkcjonalność poprzez tzw. **aktywne reguły**. Reguły te mogą być automatycznie wyzwalane przez różne zdarzenia, takie jak aktualizacja wpisów w bazie danych lub określona godzina, a po uruchomieniu mogą warunkowo podejmować akcje określone

w ich definicji. Wiele komercyjnych pakietów oferuje pewną funkcjonalność aktywnych baz danych pod postacią **wyzwalaczy** (ang. *triggers*). Wyzwalacze są częścią standardu SQL-99 i jego nowszych wersji.

W podrozdziale 26.2 wprowadzimy pojęcie **czasowych baz danych**, które przechowują historię zmian i pozwalają użytkownikom na odczytywanie zarówno danych aktualnych, jak i historycznych. Niektóre modele czasowych baz danych pozwalają również na przechowywanie spodziewanych przyszłych wartości, takich jak planowane harmonogramy. Należy zauważyć, że wiele aplikacji bazodanowych już jest aplikacjami czasowymi, ale zazwyczaj tego typu funkcjonalność jest zaimplementowana przez twórców aplikacji, a nie oferowana przez system bazy danych. Możliwość tworzenia czasowych baz danych i kierowania do nich zapytań została dodana w standardzie SQL:2011, a ponadto jest dostępna w systemie DB2. Nie omawiamy jej jednak w tym miejscu. Zainteresowanych użytkowników odsyłamy do wybranych publikacji podanych na końcu rozdziału.

Podrozdział 26.3 przedstawia krótko problematykę **przestrzennych baz danych**. Opiszemy tam typy danych przestrzennych, różne rodzaje analiz przestrzennych, operacje na danych przestrzennych, typy zapytań przestrzennych, indeksowanie danych przestrzennych, eksplorację takich danych i zastosowania przestrzennych baz danych. Większość komercyjnych i otwartych systemów relacyjnych zapewnia obsługę danych przestrzennych (za pomocą typów danych i języków zapytań), a także umożliwia indeksowanie i wydajne przetwarzanie zapytań w zakresie typowych operacji przestrzennych.

Podrozdział 26.4 jest poświęcony multimedialnym bazom danych. **Multimedialne bazy danych** oferują użytkownikom możliwość przechowywania i odczytywania różnych typów danych multimedialnych, takich jak **obrazy** (zdjęcia i rysunki), **wideo** (filmy, relacje telewizyjne i prywatne nagrania), **dźwięki** (piosenki, telefoniczne wiadomości głosowe, nagrane wypowiedzi) oraz **dokumenty** (takie jak książki czy artykuły). Opiszemy tam automatyczne analizowanie obrazów, wykrywanie obiektów na obrazach i semantyczne opisywanie obrazów.

W podrozdziale 26.5 omówimy dedukcyjne bazy danych<sup>1</sup> — obszar będący przecięciem zagadnień związanych z bazami danych, logiką i sztuczną inteligencją lub bazami wiedzy. **Dedukcyjny system baz danych** jest systemem mającym możliwość definiowania **reguł (wnioskowania)**, które pozwalają na uzyskanie dodatkowej informacji na podstawie faktów zapisanych w bazie danych. Ponieważ część teoretycznych podstaw tego typu systemów stanowi logika matematyczna, są one też nazywane **logicznymi bazami danych**. Inne rodzaje systemów, takie jak **systemy ekspertowe** i **bazy wiedzy**, również zawierają elementy wnioskowania. Takie systemy wykorzystują osiągnięcia z zakresu sztucznej inteligencji, wliczając w to takie elementy jak sieci semantyczne, ramy, systemy produkcyjne i reguły odnajdujące wiedzę związaną z określonym tematem. Podrozdział 26.6 zawiera podsumowanie rozdziału.

Czytelnik może wybrać tylko podrozdziały, które go interesują, ponieważ fragmenty tego rozdziału nie są ze sobą ściśle powiązane.

---

<sup>1</sup> Podrozdział 26.5 jest streszczeniem na temat dedukcyjnych baz danych. Pełny rozdział z trzeciego wydania, zawierający bardziej rozbudowane wprowadzenie, jest dostępny na stronie WWW niniejszej książki.



## 26.1. Wyzwalacze i inne pojęcia związane z aktywnymi bazami danych

Reguły wyzwalające automatycznie poszczególne akcje po zaistnieniu określonego zdarzenia niegdyś uważano za istotne nowe rozszerzenie funkcjonalności systemów baz danych. W rzeczywistości pojęcie **wyzwalaczy** — techniki definiowania pewnych rodzajów aktywnych reguł — istniało już we wczesnych wersjach specyfikacji SQL dla relacyjnych baz danych i stanowi obecnie część standardu SQL-99. Komercyjnie dostępne relacyjne systemy baz danych — takie jak Oracle, DB2 i SQL Server Microsoftu — oferują różne wersje wyzwalaczy. Od czasu pierwszej propozycji koncepcji wyzwalaczy przeprowadzono wiele badań rozwijających ogólny model aktywnych baz danych. W punkcie 26.1.1 omówione zostały ogólne wytyczne proponowane dla reguł w aktywnych bazach danych. W przykładach używana będzie składnia komercyjnego systemu Oracle, ponieważ implementacja wyzwalaczy w tym systemie jest bliska ich koncepcji w standardzie SQL. Punkt 26.1.2 przedstawia ogólne kwestie związane z projektowaniem i implementacją aktywnych baz danych. Następnie przedstawione są przykłady implementacji aktywnych baz danych w eksperymentalnym systemie STARBURST (w punkcie 26.1.3), który realizuje wiele koncepcji proponowanego rozwoju tego typu systemów. Punkt 26.1.4 omawia możliwe zastosowania aktywnych baz danych, a 26.1.5 opisuje, jak wyzwalacze są deklarowane w języku SQL-99.

### 26.1.1. Uogólniony model aktywnych baz danych i wyzwalacze w Oracle

Model używany przy opisywaniu reguł w aktywnych bazach danych jest modelem typu **Zdarzenie-Warunek-Akcja (ZWA)**. Reguły w modelu ZWA składają się z trzech elementów:

- (1) **Zdarzenie** (lub zdarzenia) uruchamiające regułę — takimi zdarzeniami zazwyczaj są aktualizacje wpisów w bazie danych. Ogólnie jednak może to być również zdarzenie związane z czasem<sup>2</sup> lub dowolny inny typ zewnętrznego zdarzenia.
- (2) **Warunek** określający, czy akcja reguły ma być wykonana — po zaistnieniu zdarzenia sprawdzany jest *opcjonalny* warunek. Jeżeli warunek *nie jest zdefiniowany*, to akcja zostanie wykonana bezwarunkowo. Jeżeli warunek jest zdefiniowany, to akcja zostanie wykonana tylko wtedy, gdy *jest on spełniony*.
- (3) **Akcja** — akcja jest zazwyczaj ciągiem wyrażeń języka SQL, ale może to być również transakcja bazy danych lub wykonanie zewnętrznego programu, które ma nastąpić automatycznie po zaistnieniu określonego zdarzenia.

Rozważmy kilka przykładów ilustrujących przytoczone rozwiązanie. Opierają się one na uproszczonej wersji bazy danych FIRMA z rysunku 5.5, przedstawionej na rysunku 26.1. Każdy pracownik ma przypisane nazwisko (NAZWISKO), numer PESEL (PESEL), wynagrodzenie (PENSJA), dział, dla którego pracuje (NRDZ, klucz obcy tabeli DZIAŁ) oraz bezpośredniego przełożonego (PESEL\_PRZEŁOŻ — rekurencyjny klucz obcy do tabeli PRACOWNIK). W tym przykładzie przyjmiemy, że atrybut NRDZ może być pusty, co oznacza, że pracownik tym

---

<sup>2</sup> Przykładem zdarzenia związanego z czasem (czasowego) jest zdarzenie okresowe, takie jak uruchomienie reguły codziennie o godzinie 5.30.

## PRACOWNIK

IMIĘNAZWISKO	PESEL	PENSJA	NRDZ	PESEL_PRZEŁOŻ
--------------	-------	--------	------	---------------

## DZIAŁ

NAZWADZ	NRDZ	SUMA_PENSJI	PESEL_KIEROWNIKA
---------	------	-------------	------------------

Rysunek 26.1. Uproszczona baza danych FIRMA używana w przykładach aktywnych reguł

czasowo nie jest przydzielony do żadnego działu. Każdy dział ma swoją nazwę (NAZWADZ), numer (NRDZ), sumę pensji wszystkich pracowników przydzielonych do pracy w dziale (SUMA\_PENSJI) oraz kierownika (PESEL\_KIEROWNIKA — klucz obcy tabeli PRACOWNIK).

Zauważmy, że SUMA\_PENSJI jest w rzeczywistości atrybutem pochodnym, którego wartość powinna być sumą pensji wszystkich pracowników przydzielonych do pracy w danym dziale. Utrzymywanie prawidłowej wartości takiego atrybutu można zrealizować za pomocą aktywnej reguły. Najpierw trzeba określić **zdarzenia**, które *mogą spowodować* zmianę wartości atrybutu SUMA\_PENSJI, a są to:

- (1) dodanie (jednej lub więcej) nowej krotki do tabeli pracowników,
- (2) zmiana wynagrodzenia (jednego lub więcej) istniejących pracowników,
- (3) zmiana przypisania do działu istniejących pracowników,
- (4) skasowanie (jednej lub więcej) krotek pracowników.

W przypadku zdarzenia 1. musimy wyliczyć nową wartość atrybutu SUMA\_PENSJI tylko wówczas, gdy nowy pracownik jest od razu przypisany do działu, czyli jeśli wartość NRDZ dla nowego pracownika nie jest pusta (przyjmując, że wartość pusta jest w ogóle dozwolona). Byłby to więc **warunek**, który należy sprawdzić. Podobny warunek będzie również istniał dla zdarzenia 2. (oraz 4.) i będzie polegał na sprawdzeniu, czy pracownik, którego pensja uległa zmianie (lub który jest usuwany) jest obecnie przypisany do jakiegoś działu. Dla zdarzenia 3. akcja zawsze powinna być wykonana, więc nie będzie do niego przypisany żaden warunek.

**Akcją** dla zdarzeń 1., 2. i 4. będzie automatyczne uaktualnienie wartości atrybutu SUMA\_PENSJI dla działu pracownika, który jest dodawany, zmieniany lub usuwany. W przypadku zdarzenia 3. potrzebna jest dwustopniowa akcja; najpierw trzeba zmodyfikować atrybut SUMA\_PENSJI w starym dziale pracownika, a następnie w nowym.

Na rysunku 26.2(a) przedstawiono (w notacji stosowanej w systemie Oracle) cztery aktywne reguły (wyzwalacze) R1, R2, R3 i R4 odpowiadające przedstawionemu przykładowi. W celu objaśnienia składni tworzenia wyzwalaczy w Oracle rozważmy regułę R1. Wyrażenie CREATE TRIGGER określa nazwę nowego wyzwalacza (aktywnej reguły); dla R1 jest to SUMAPENSJI1. Słowo kluczowe AFTER mówi, że reguła ma być uruchomiona *po* danym zdarzeniu. Zdarzenia wyzwalające regułę (w przykładzie jest to dodanie nowego pracownika) są wymieniane po słowie AFTER<sup>3</sup>.

<sup>3</sup> Jak zobaczymy później, można również użyć słowa kluczowego BEFORE, które oznacza, że reguła ma być uruchomiona *przed* zdarzeniem, które ją wywołuje.

Wyrażenie **ON** wskazuje relacje, dla której reguła jest definiowana (w przykładzie jest to **PRACOWNIK**). *Opcjonalne* wyrażenie **FOR EACH ROW** określa, że akcja ma być wykonana *po jednym razie dla każdej krotki* związanej ze zdarzeniem<sup>4</sup>.

*Opcjonalne* wyrażenie **WHEN** jest używane do wskazania warunków, które muszą być sprawdzone po uruchomieniu reguły, ale przed wykonaniem akcji. W końcu akcja, która ma zostać wykonana, jest określona jako blok instrukcji w języku PL/SQL, który zazwyczaj zawiera jedno lub kilka zapytań SQL albo wywołań zewnętrznych procedur.

```
(a) R1: CREATE TRIGGER SUMAPENSJI1
      AFTER INSERT ON PRACOWNIK
      FOR EACH ROW
      WHEN (NEW.NRDZ IS NOT NULL)
      UPDATE DZIAŁ
      SET SUMA_PENSJI=SUMA_PENSJI + NEW.PENSJA
      WHERE NRDZ=NEW.NRDZ;

R2: CREATE TRIGGER SUMAPENSJI2
      AFTER UPDATE OF PENSJA ON PRACOWNIK
      FOR EACH ROW
      WHEN (NEW.NRDZ IS NOT NULL)
      UPDATE DZIAŁ
      SET SUMA_PENSJI=SUMA_PENSJI + NEW.PENSJA - OLD.PENSJA
      WHERE NRDZ=NEW.NRDZ;

R3: CREATE TRIGGER SUMAPENSJI3
      AFTER UPDATE OF NRDZ ON PRACOWNIK
      FOR EACH ROW
      BEGIN
      UPDATE DZIAŁ
      SET SUMA_PENSJI=SUMA_PENSJI + NEW.PENSJA
      WHERE NRDZ=NEW.NRDZ;
      UPDATE DZIAŁ
      SET SUMA_PENSJI=SUMA_PENSJI-OLD.PENSJA
      WHERE NRDZ=OLD.NRDZ;
      END;

R4: CREATE TRIGGER SUMA_PENSJI4
      AFTER DELETE ON PRACOWNIK
      FOR EACH ROW
      WHEN (OLD.NRDZ IS NOT NULL)
      UPDATE DZIAŁ
      SET SUMA_PENSJI=SUMA_PENSJI-OLD.PENSJA
      WHERE NRDZ=OLD.NRDZ;
```

---

<sup>4</sup> Ponownie, jak zobaczymy później, możliwe jest również uruchomienie wyzwalacza *tylko raz* nawet, jeśli zdarzenie wyzwalające dotyczy większej liczby krotek.

```
(b) R5: CREATE TRIGGER POWIADOM_PRZEŁOŻ1
BEFORE INSERT OR UPDATE OF PENSJA, PESEL_PRZEŁOŻ ON PRACOWNIK
FOR EACH ROW
WHEN
  (NEW.PENSJA > (SELECT PENSJA FROM PRACOWNIK
    WHERE PESEL=NEW.PESEL_PRZEŁOŻ))
  POWIADOM_PRZEŁOŻ(NEW.PESEL_PRZEŁOŻ, NEW.PESEL);
```

RYSUNEK 26.2. Określanie aktywnych reguł jako wyzwalaczy w notacji Oracle. (a) Wyzwalacze do automatycznej aktualizacji wartości atrybutu SUMA\_PENSJI tabeli DZIAŁ. (b) Wyzwalacz porównujący pensję pracownika z pensją jego przełożonego (przełożonej)

Cztery przykładowe wyzwalacze (aktywne reguły) R1, R2, R3 i R4 ilustrują kilka istotnych cech aktywnych reguł. Po pierwsze, podstawowymi **zdarzeniami**, które mogą uruchomić regułę, są polecenia aktualizujące z języka SQL (INSERT, UPDATE i DELETE). W notacji Oracle są one określane za pomocą słów kluczowych odpowiednio: INSERT, UPDATE i DELETE. W przypadku słowa UPDATE można wyszczególnić atrybuty, których zmiana ma uaktywnić regułę (na przykład pisząc UPDATE OF PENSJA, NRDZ). Po drugie, projektant reguł musi mieć możliwość odwoływania się do krotek, które są dodawane, kasowane i zmieniane. W notacji Oracle do tego celu służą słowa kluczowe NEW i OLD. NEW odnosi się do nowo dodanej lub zaktualizowanej krotki, a OLD — do krotki, która została usunięta lub krotki przed modyfikacją.

Reguła R1 jest więc aktywowana podczas operacji INSERT zastosowanej do relacji PRACOWNIK. W R1 sprawdzany jest warunek (NEW.NRDZ IS NOT NULL) i jeżeli jest on spełniony, czyli nowy pracownik jest związany z jakimś działem, to wykonywana jest akcja. Akcja w R1 aktualizuje krotki relacji DZIAŁ związane z nowo dodanymi pracownikami, dodając ich wynagrodzenie (NEW.PENSJA) do atrybutu SUMA\_PENSJI działu, w którym pracują.

Reguła R2 jest podobna do R1, ale jest uruchamiana przez operację UPDATE, która zmienia atrybut PENSJA, a nie dodaje nowych krotek, jak robi to polecenie INSERT. Reguła R3 jest wywoływana przez aktualizację atrybutu NRDZ relacji PRACOWNIK, co oznacza przeniesienie pracownika z jednego działu do drugiego. R3 nie ma określonego dodatkowego warunku, czyli akcja jest wykonywana zawsze, kiedy tylko zaistnieje opisane zdarzenie. Akcja aktualizuje zarówno *nowy*, jak i *stary* dział przenoszonych pracowników, dodając ich pensje do SUMY\_PENSJI w *nowym* dziale i odejmując je od wartości SUMA\_PENSJI w *starym*. Zauważmy, że ta reguła będzie poprawnie działała również w sytuacji, gdy nowy dział nie jest zdefiniowany (ma wartość null), ponieważ wtedy żaden dział nie będzie wybrany przez akcję reguły<sup>5</sup>.

Istotne jest zwrócenie uwagi na znaczenie opcjonalnej frazy FOR EACH ROW, która zaznacza, że akcja reguły ma być wykonana osobno *dla każdej krotki*. Jest to tak zwany **wyzwalacz na poziomie wiersza**. Jeżeli ta fraza zostanie pominięta, to stworzymy **wyzwalacz na poziomie wyrażenia**, który będzie uruchamiany tylko raz dla każdego wyrażenia. Aby unaocznić różnicę pomiędzy tymi dwoma typami wyzwalaczy, rozważmy przykład, w którym pracownicy działu 5. otrzymują 10% podwyżki. Operacja ta spowoduje uruchomienie reguły R2:

<sup>5</sup> R1, R2 i R4 mogą być również zapisane bez dodatkowych warunków. Przytoczone definicje będą jednak znacznie bardziej wydajne dzięki warunkowi, który sprawia, że akcja jest wywoływana tylko wtedy, gdy jest to naprawdę konieczne.

```
UPDATE PRACOWNIK
SET PENSJA=1.1 * PENSJA
WHERE NRDZ = 5;
```

Ponieważ powyższe wyrażenie może zaktualizować wiele rekordów, reguła działająca na poziomie wiersza (jak przykładowa reguła R2 z rysunku 26.2) będzie wywoływana *raz dla każdego wiersza*, podczas gdy reguła na poziomie wyrażenia będzie uruchomiona *tylko raz*. System Oracle pozwala użytkownikom na określenie, której z tych dwóch opcji chcą użyć dla każdej reguły. Dodanie opcjonalnego wyrażenia `FOR EACH ROW` tworzy wyzwalacz poziomu wiersza, a brak takiego wyrażenia — wyzwalacz poziomu wyrażenia. Zauważmy, że słowa kluczowe `NEW` i `OLD` mogą być używane tylko w przypadku wyzwalaczy poziomu wiersza.

Jako drugi przykład założmy, że chcemy sprawdzić, czy któryś pracownik zarabia więcej niż jego bezpośredni przełożony. Kilka sytuacji może uruchamiać tego typu regułę: dodanie nowego pracownika, zmiana wynagrodzenia pracownika oraz zmiana przełożonego. Przypuśćmy, że akcją będzie uruchomienie zewnętrznej procedury `POWIADOM_PRZEŁOŻ`<sup>6</sup>, która wyśle powiadomienie do przełożonego. Reguła powinna być napisana tak jak R5 (patrz rysunek 26.2(b)).

Rysunek 26.3 pokazuje składnię określającą niektóre z głównych opcji dostępnych dla wyzwalaczy w systemie Oracle. Standard SQL-99 dla wyzwalaczy opiszemy w punkcie 26.1.5.

```
<wyzwalacz> ::= CREATE TRIGGER <nazwa wyzwalacza>
(AFTER|BEFORE) <zdarzenia aktywujące> ON <nazwa tabeli>
[FOR EACH ROW]
[WHEN <warunek>]
<akcje wyzwalacza>;
<zdarzenia aktywujące> ::= <zdarzenie wyzwalacza> {OR <zdarzenie wyzwalacza>}
<zdarzenie wyzwalacza> ::= INSERT | DELETE | UPDATE [OF <nazwa kolumny> {, <nazwa
kolumny>}]
<akcja wyzwalacza> ::= <blok PL/SQL>
```

RYСУNEK 26.3. Ogólna składnia definiowania wyzwalaczy w systemie Oracle (tylko najważniejsze opcje)

## 26.1.2. Projektowanie i implementacja aktywnych baz danych

W poprzednim punkcie zostały przedstawione podstawowe pojęcia związane z definiowaniem aktywnych reguł. W tym punkcie omówimy bardziej szczegółowe kwestie związane z projektowaniem i implementacją takich reguł. Pierwsza z nich dotyczy aktywacji, deaktywacji i grupowania reguł. Poza tworzeniem reguł aktywny system baz danych powinien umożliwiać użytkownikom *aktywowanie*, *deaktywowanie* i *kasowanie* reguł przy użyciu ich nazw. **Reguła nieaktywna** nie będzie uruchamiana przez jej zdarzenie aktywujące. Ta funkcja pozwala użytkownikom na wyłączanie okresowe reguł, kiedy nie są potrzebne. **Komenda aktywująca** z powrotem włącza nieaktywną regułę. **Komenda kasująca** usuwa regułę z systemu. Inną opcją jest tworzenie **zestawów reguł**, co umożliwi równoczesną deaktywację, aktywację lub kasowanie wszystkich reguł zgrupowanych w zestawie. Uży-

<sup>6</sup> Przyjmujemy, że została zdefiniowana odpowiednia zewnętrzna funkcja. Ten mechanizm jest dostępny w standardzie SQL-99 i nowszych wersjach.

teczna jest również możliwość ręcznego uruchomienia reguł przez użytkownika komendą PROCESS RULES.

Druga kwestia dotyczy tego, czy akcja ma być uruchamiana *przed*, *po* czy *równocześnie* ze zdarzeniem wyzwalającym. **Wyzwalacze** BEFORE są uruchamiane przed wykonaniem zdarzenia aktywującego wyzwalacz. Można je zastosować np. do sprawdzania naruszenia ograniczeń. **Wyzwalacze** AFTER są uruchamiane po wykonaniu zdarzenia. Można je wykorzystać np. do aktualizowania danych pochodnych i monitorowania systemu pod kątem konkretnych zdarzeń i warunków. **Wyzwalacze** INSTEAD OF są wykonywane zamiast zdarzenia. Pozwalają np. wykonywać powiązane aktualizacje na relacjach bazy w reakcji na zdarzenie aktualizacji perspektywy.

Powiązane z tym jest również pytanie, czy uruchamiana akcja powinna być uruchamiana jako *oddzielna transakcja*, czy ma być częścią transakcji, która uruchomiła regułę. Spróbujmy najpierw uporządkować różne opcje. Należy pamiętać, że nie wszystkie opcje mogą być dostępne w każdym systemie baz danych. Najczęściej większość z nich *ogranicza się zaledwie do jednej lub dwóch* spośród omawianych tutaj funkcji.

Przyjmijmy, że zdarzenie wyzwalające jest częścią wykonywanej transakcji. Wówczas należy najpierw rozważyć wpływ tego zdarzenia na warunek zawarty w regule. *Sprawdzanie warunku* reguły jest również nazywane **rozważaniem reguły**, ponieważ akcja będzie podjęta dopiero po rozważeniu, czy warunek jest spełniony, czy nie. Istnieją trzy główne możliwości rozważania reguły:

- (1) **Natychmiastowe rozważanie:** warunek jest sprawdzany w ramach tej samej transakcji co zdarzenie wyzwalające i *natychmiast*. Ten przypadek może być dalej podzielony na trzy kategorie:
  - sprawdzanie warunku *przed* wykonaniem zdarzenia;
  - sprawdzanie warunku *po* wykonaniu zdarzenia;
  - sprawdzanie warunku *zamiast* wykonywania zdarzenia.
- (2) **Opóźnione rozważanie:** warunek jest sprawdzany pod koniec transakcji związanej ze zdarzeniem wyzwalającym. W tym przypadku istnieje możliwość nagromadzenia większej liczby reguł oczekujących na rozważenie pod koniec transakcji.
- (3) **Niezależne rozważanie:** warunek jest sprawdzany jako oddzielna transakcja uruchomiona przez transakcję, która zawiera zdarzenie wyzwalające.

Następny zestaw opcji dotyczy związku pomiędzy sprawdzaniem warunku reguły a *wykonaniem* jej akcji. Tutaj znów mamy do czynienia z trzema możliwościami: **natychmiastowego**, **opóźnionego** i **niezależnego** wykonania. Większość aktywnych systemów baz danych stosuje pierwszą opcję, czyli akcja jest wykonywana *natychmiast* po rozważeniu reguły (o ile warunek jest spełniony).

System Oracle (patrz punkt 26.1.1) stosuje model *natychmiastowego rozważania*, ale pozwala użytkownikowi na określenie, czy warunek reguły ma być sprawdzany *przed*, czy *po* zaistnieniu zdarzenia wyzwalającego. Wykonanie akcji następuje również *natychmiast* po sprawdzeniu warunku. System STARBURST (patrz punkt 26.1.3) używa *opóźnionego rozważania*, co oznacza, że wszystkie reguły uruchomione przez zdarzenia danej transakcji oczekują na jej zakończenie i dopiero po wykonaniu przez nią komendy COMMIT WORK ich warunki są sprawdzane<sup>7</sup>.

<sup>7</sup> STARBURST pozwala również na ręczne uruchomienie rozważania reguł poprzez komendę PROCESS RULES.

Kolejną kwestią związaną z aktywnymi regułami baz danych jest rozróżnienie pomiędzy *regułami poziomu wiersza* i *poziomu wyrażenia*. Ponieważ wyrażenia SQL zmieniające dane (które są określane jako zdarzenia wyzwalające reguły) mogą dotyczyć wielu krotek, należy określić, czy reguła powinna być wykonana tylko raz dla *całego wyrażenia*, czy też osobno dla *każdego zmienianego wiersza* (czyli krotki). Standard SQL-99 (patrz punkt 26.1.5) i system Oracle (patrz punkt 26.1.1) pozwalają użytkownikowi na wybór jednej z powyższych opcji podczas definiowania reguły, a system STARBURST powala jedynie na reguły wyzwalane dla całego wyrażenia. W punkcie 26.1.3 przedstawione zostaną przykłady definiowania wyzwalaczy tego typu.

Jednym z utrudnień, które mogło wpłynąć na ograniczenie użycia aktywnych reguł w bazach danych, pomimo związanego z ich stosowaniem uproszczenia struktury baz danych oraz implementacji oprogramowania, jest brak łatwych w użyciu technik projektowania, pisania i weryfikacji reguł. Na przykład sprawdzenie, czy zestaw aktywnych reguł jest **spójny**, co oznacza, że poszczególne reguły nie znoszą wzajemnie swojego działania, jest dość trudne do zrealizowania. Trudne jest również zagwarantowanie, że zestaw reguł zawsze po wykonaniu **zakończy** swoje działanie. Aby krótko zilustrować problem zakończenia działania reguł, rozważmy reguły przedstawione na rysunku 26.4. Reguła R1 jest uruchamiana przez operację INSERT przeprowadzaną na tabeli TABELA1, a jej akcja polega na zmianie atrybutu ATRYBUT1 w TABELA2. Z drugiej strony, zdarzeniem wyzwalającym regułę R2 jest aktualizacja atrybutu ATRYBUT1 w TABELA2, a jej akcją jest dodanie krotki do tabeli TABELA1 (operacja INSERT). W przytoczonym przykładzie łatwo zauważyć, że te dwie reguły będą się stale wzajemnie uruchamiały, co prowadzi do zapętlenia. W przypadku, gdy reguł w systemie są dziesiątki, nie sposób zagwarantować, że taka sytuacja nigdy się nie zdarzy.

```
R1: CREATE TRIGGER T1
AFTER INSERT ON TABELA1
FOR EACH ROW
UPDATE TABELA2
SET ATRYBUT1=...;

R2: CREATE TRIGGER T2
AFTER UPDATE OF ATRYBUT1 ON TABELA2
FOR EACH ROW
INSERT INTO TABELA1 VALUES (...);
```

RYSUNEK 26.4. Przykład ilustrujący problem stopu dla aktywnych reguł

Aby aktywne reguły mogły rozwinąć swoje możliwości, niezbędne jest opracowanie narzędzi na potrzeby projektowania, debugowania i kontroli, które pozwolą użytkownikom na projektowanie reguł, a później ich debugowania.



## 26.1.3. Przykładowe aktywne reguły poziomu wyrażenia w systemie STARBURST

Niniejszy punkt przedstawia kilka przykładów ilustrujących określanie reguł w eksperymentalnym systemie baz danych STARBURST. Pozwoli to na zademonstrowanie użycia reguł poziomu wyrażenia (jest to jedyny rodzaj reguł, który obsługuje STARBURST).

Trzy aktywne reguły R1S, R2S i R3S z rysunku 26.5 odpowiadają trzem pierwszym regułom z rysunku 26.2, ale używają notacji STARBURST i modelu poziomu wyrażenia. Objasnimy strukturę reguł na przykładzie R1S. Wyrażenie CREATE RULE określa nazwę reguły — dla R1S jest to SUMAPENSJI1. Klauzula ON określa relację, dla której definiowana jest reguła — dla R1S jest to PRACOWNIK. Warunek WHEN określa **zdarzenia** uruchamiające regułę<sup>8</sup>. *Opcjonalne* wyrażenie IF określa **warunki** wykonania akcji reguły. Kończące definicję wyrażenie THEN określa **akcję** (lub akcje), która zazwyczaj jest jednym (lub kilkoma) zapytaniem SQL.

```
R1S: CREATE RULE SUMAPENSJI1 ON PRACOWNIK
WHEN INSERTED
IF EXISTS (SELECT * FROM INSERTED WHERE NRDZ IS NOT NULL)
THEN UPDATE DZIAŁ AS D
    SET D.SUMA_PENSJI=D.SUMA_PENSJI +
        (SELECT SUM(I.PENSJA) FROM INSERTED AS I WHERE D.NRDZ=I.NRDZ)
    WHERE D.NRDZ IN (SELECT NRDZ FROM INSERTED);

R2S: CREATE RULE SUMAPENSJI2 ON PRACOWNIK
WHEN UPDATED (PENSJA)
IF EXISTS (SELECT * FROM NEW-UPDATED WHERE NRDZ IS NOT NULL)
    OR EXISTS (SELECT * FROM OLD-UPDATED WHERE NRDZ IS NOT NULL)
THEN UPDATE DZIAŁ AS D
    SET D.SUMA_PENSJI=D.SUMA_PENSJI +
        (SELECT SUM(N.PENSJA) FROM NEW-UPDATED AS N WHERE D.NRDZ=N.NRDZ) -
        (SELECT SUM(O.PENSJA) FROM OLD-UPDATED AS O WHERE D.NRDZ=O.NRDZ)
    WHERE D.NRDZ IN (SELECT NRDZ FROM NEW-UPDATED) OR
        D.NRDZ IN (SELECT NRDZ FROM OLD-UPDATED);

R3S: CREATE RULE SUMAPENSJI3 ON PRACOWNIK
WHEN UPDATED (NRDZ)
THEN UPDATE DZIAŁ AS D
    SET D.SUMA_PENSJI=D.SUMA_PENSJI +
        (SELECT SUM(N.PENSJA) FROM NEW-UPDATED AS N WHERE D.NRDZ=N.NRDZ)
    WHERE D.NRDZ IN (SELECT NRDZ FROM NEW-UPDATED);

    UPDATE DZIAŁ AS D
    SET D.SUMA_PENSJI=D.SUMA_PENSJI -
        (SELECT SUM(O.PENSJA) FROM OLD-UPDATED AS O WHERE D.NRDZ=O.NRDZ)
    WHERE D.NRDZ IN (SELECT NRDZ FROM OLD-UPDATED);
```

RYSunek 26.5. Aktywne reguły używające semantyki poziomu wyrażenia w notacji STARBURST

<sup>8</sup> Należy zwrócić uwagę, że słowo kluczowe WHEN w systemie STARBURST określa **zdarzenie** wyzwalające, natomiast w SQL i systemie Oracle jest używane do definiowania **warunku** reguły.

W STARBURST podstawowymi zdarzeniami, które mogą być określone jako zdarzenia wyzwajające, są podstawowe komendy SQL: INSERT, UPDATE i DELETE. W notacji STARBURST można się do nich odwoływać poprzez słowa kluczowe INSERTED, UPDATED i DELETED. Użytkownik tworzący regułę musi też mieć możliwość odwoływania się do modyfikowanych krotek. Aby odwołać się do **tabel tymczasowych** (ang. *transition tables*) obejmujących krotki, które zostały dodane, usunięte, krotek *przed* modyfikacją i *po* modyfikacji, w notacji STARBURST należy użyć odpowiednio słów kluczowych INSERTED, DELETED, OLD-UPDATED i NEW-UPDATED. Oczywiście, w zależności od typu zdarzenia wyzwajającego, dostępne będą tylko niektóre z tych tymczasowych tabel. Autor reguły może się posługiwać tymi tabelami, pisząc warunek i akcję. Tymczasowe tabele zawierają krotki tego samego typu co tabela wskazana w warunku ON (dla R1S, R2S i R3S jest to relacja PRACOWNIK).

W semantyce poziomu wyrażenia projektant reguły może odwoływać się jedynie do całych tymczasowych tabel i reguła wykonywana jest tylko raz. Musi więc być pisana inaczej niż w przypadku semantyki poziomu wiersza. Ponieważ jednym wyrażeniem SQL można dodać wiele krotek, należy sprawdzić czy *przynajmniej jedna* z nich jest związana z konkretnym działem. W R1S sprawdzany jest warunek:

```
EXISTS (SELECT * FROM INSERTED WHERE NRZ IS NOT NULL)
```

i jeżeli okaże się prawdziwy, to uruchamiana jest akcja. Akcja aktualizuje pojedynczym wyrażeniem krotki DZIAŁ związane z nowo dodanymi pracownikami, dodając ich pensje do atrybutu SUMA\_PENSJI każdego z działów. Ponieważ do każdego z nich może należeć więcej niż jeden nowy pracownik, zastosowano funkcję agregującą SUM do uwzględnienia przy dodawaniu wszystkich pensji.

Reguła R2S jest podobna do R1S, ale jest uruchamiana nie przez operację INSERT, a przez operację UPDATE, która zmienia pensję jednego lub więcej pracowników. Reguła R3S jest uruchamiana przez zmianę atrybutu NRZ tabeli PRACOWNIK, co oznacza przeniesienie jednego lub więcej pracowników z jednego działu do innego. W R3S nie ma żadnego warunku, więc akcja będzie wykonana zawsze, kiedy zaistnieje zdarzenie wyzwajające<sup>9</sup>. Akcja aktualizuje zarówno stary, jak i nowy dział (lub działy) przeniesionych pracowników poprzez dodanie ich wynagrodzenia do wartości SUMA\_PENSJI dla *nowych* działów i odjęcie dla każdego *starego* działu.

W przytoczonym przykładzie reguły poziomu wyrażenia są znacznie bardziej skomplikowane niż te stosujące semantykę poziomu wiersza (co łatwo zauważyć, porównując rysunki 26.2 i 26.5). Nie jest to jednak ogólna zasada i w innych przypadkach aktywnych reguł reguły poziomu wyrażenia mogą być znacznie łatwiejsze do określenia niż reguły poziomu wiersza.

Model przyjęty w systemie STARBURST używa **opóźnionego rozważania reguł**. Oznacza to, że wszystkie uruchomione podczas transakcji reguły są umieszczane w zbiorze nazywanym **zbiorem niezgodności** (ang. *conflict set*) i wykonywane dopiero po całkowitym zakończeniu transakcji (czyli po wykonaniu komendy COMMIT WORK). STARBURST pozwala również na ręczne uruchomienie rozważania reguł w środku transakcji poprzez wydanie komendy PROCESS RULES. Ponieważ równocześnie powinno być wykonane wiele

---

<sup>9</sup> Podobnie jak w przykładzie w Oracle, reguły R1S i R2S nie muszą mieć warunków. Będą jednak bardziej wydajne, jeśli akcja nie będzie wykonywana za każdym razem, a tylko w przypadku kiedy to rzeczywiście jest potrzebne.

reguł, konieczne jest określenie *kolejności* ich wykonywania. STARBURST pozwala na jej samodzielne określenie przez użytkownika podczas definiowania reguły<sup>10</sup>. Poza tym, tabele tymczasowe (INSERTED, DELETED, NEW-UPDATED i OLD-UPDATED) zawierają *sumę rezultatów* wszystkich operacji w transakcji, które dotyczyły każdej z tabel (ponieważ do każdej tabeli w trakcie transakcji zastosowanych mogło zostać wiele operacji).

## 26.1.4. Możliwe zastosowania aktywnych baz danych

Omówimy teraz krótko niektóre potencjalne obszary zastosowań reguł aktywnych. Oczywiście, pierwszym z nich jest **powiadamanie** o zaistnieniu określonych warunków. Aktywna baza danych może być używana na przykład do monitorowania temperatury w piecu przemysłowym. Aplikacja może okresowo zapisywać do bazy danych odczyt temperatury z czujników, a aktywne reguły mogą być tak określone, żeby ich uruchomienie następowało w przypadku dodania nowej informacji o temperaturze, z warunkiem sprawdzającym, czy temperatura nie przekracza bezpiecznego poziomu, i akcją uruchamiającą alarm.

Aktywne reguły mogą być również używane do **wymuszania więzów integralności** poprzez określanie rodzaju zdarzeń, które mogą te zasady naruszać, i warunków, które sprawdzają, czy zasady rzeczywistości zostały naruszone, czy nie. W ten sposób mogą być również wymuszane złożone ograniczenia związane z konkretną aplikacją, nazywane często **regułami biznesowymi**. W przykładowej bazie danych UNIWERSYTET można wprowadzić regułę monitorującą średnią ocen studentów za każdym razem, kiedy dodawana jest nowa ocena, i powiadamiającą promotora o przekroczeniu pewnego minimalnego pułapu. Inna reguła może sprawdzać, czy spełnione są warunki wstępne przy zapisywaniu studenta na określony kurs.

Kolejnym zastosowaniem może być automatyczne **kontrolowanie danych pochodnych**, tak jak to miało miejsce w przykładowych regułach R1 do R4 aktualizujących atrybut SUMA\_ → PENSJI za każdym razem, gdy zmieniały się krotki odpowiadające poszczególnym pracownikom. Podobnym zastosowaniem jest używanie aktywnych reguł do aktualizacji **zmaterializowanych perspektyw** (patrz podrozdział 5.3) za każdym razem, gdy zmieniają się relacje w bazie danych. Innym zdarzeniem wyzwalającym może być aktualizacja perspektywy, którą to operację za pomocą wyzwalacza INSTEAD OF można przekształcić w aktualizację relacji bazowych. Takie zastosowania są istotne także w nowych technologiach z hurtowni danych (patrz rozdział 29.). Powiązanym zastosowaniem jest aktualizowanie **replikowanych tabel** za każdym razem, gdy modyfikowana jest zawartość tabeli głównej.

## 26.1.5. Wyzwalacze w SQL-99

Wyzwalacze w standardzie SQL-99 są dość podobne do przykładów omawianych w punkcie 26.1.1., z pewnymi drobnymi zmianami składni. Podstawowe **zdarzenia**, które mogą być wykorzystywane do uruchamiania reguł, to standardowe komendy aktualizujące SQL: INSERT, DELETE i UPDATE. W przypadku UPDATE można określić, aktualizacja których atrybu-

<sup>10</sup> Jeżeli nie zostanie zdefiniowany żaden porządek w zbiorze niezgodności, to system przyjmuje, że pierwsze mają być wykonane te reguły, które były wcześniej zadeklarowane.

tów będzie powodować uruchomienie. Obsługiwane są zarówno wyzwalacze poziomu wyrażenia, jak i wiersza, określane wewnątrz definicji za pomocą wyrażeń odpowiednio `FOR EACH STATEMENT` lub `FOR EACH ROW`. Jedyną różnicą składniową jest to, że wyzwalacz może używać nazw zmiennych konkretnej krotki dla nowych i starych krotek, zamiast używania słów kluczowych `NEW` i `OLD` (jak na rysunku 26.1). Wyzwalacz `W1` z rysunku 26.6 pokazuje, jak można zdefiniować regułę poziomu wiersza `R2` z rysunku 26.1(a) w języku SQL-99. Wewnątrz wyrażenia `REFERENCING` nazwano zmienne krotek `S` i `N` w odniesieniu do starej krotki (`OLD` — przed modyfikacją) i nowej (`NEW` — po modyfikacji). Wyzwalacz `W2` z rysunku 26.6 przedstawia wyzwalacz poziomu wyrażenia `R2S` z rysunku 26.5 w języku SQL-99. W wyzwalaczu poziomu wyrażenia, klauzula `REFERENCING` odwołuje się do tabeli wszystkich nowych krotek (nowo dodanych lub nowo zmienionych) jako `N`, podczas gdy tabela wszystkich starych krotek (usuniętych, lub krotek przed zmianą ich zawartości) jest oznaczona zmienną `S`.

```

W1: CREATE TRIGGER SUMAPENSJI1
AFTER UPDATE OF PENSJA ON PRACOWNIK
REFERENCING OLD ROW AS S, NEW ROW AS N
FOR EACH ROW
WHEN (N.NRDZ IS NOT NULL)
  UPDATE DZIAŁ
  SET SUMA_PENSJI = SUMA_PENSJI + N.PENSJA - S.PENSJA
  WHERE NRDZ = N.NRDZ;
W2: CREATE TRIGGER SUMA_PENSJI2
AFTER UPDATE OF PENSJA ON PRACOWNIK
REFERENCING OLD TABLE AS S, NEW TABLE AS N
FOR EACH STATEMENT
WHEN EXISTS (SELECT * FROM N WHERE N.NRDZ IS NOT NULL) OR
  EXISTS (SELECT * FROM S WHERE S.NRDZ IS NOT NULL)
  UPDATE DZIAŁ AS D
  SET D.SUMA_PENSJI = D.SUMA_PENSJI
    + (SELECT SUM(N.PENSJA) FROM N WHERE D.NRDZ=N.NRDZ)
    - (SELECT SUM(S.PENSJA) FROM S WHERE D.NRDZ=S.NRDZ)
  WHERE NRDZ IN ((SELECT NRDZ FROM N) UNION (SELECT NRDZ FROM S));

```

RYSunek 26.6. Wyzwalacz `W1` ilustrujący składnię wyzwalaczy w SQL-99

## 26.2. Koncepcja czasowych baz danych

Czasowe bazy danych, w szerokim rozumieniu, obejmują wszystkie aplikacje bazodanowe, które do uporządkowywania informacji wykorzystują aspekt czasu. Ilustrują one zatem potrzebę opracowania zestawu standardowych pojęć dla osób zajmujących się tworzeniem tego typu aplikacji. Czasowe aplikacje bazodanowe powstawały od samego początku istnienia baz danych. Kwestie związane z wyodrębnianiem, projektowaniem i implementacją upływu czasu były w tych aplikacjach pozostawione głównie projektantom systemów. Istnieje wiele przykładów aplikacji, które przechowują dane powiązane z upływem czasu. Należą do nich systemy *opieki zdrowotnej*, gdzie przechowywane są historie chorób pacjentów; *ubezpieczeniowe*, przechowujące dane o roszczeniach i historiach wypadków, a także informacje o czasie trwania polis; wszelkiego typu *systemy rezerwacyjne* (hotelowe, kole-

jowe, lotnicze itp.) przechowujące dane o okresach rezerwacji; *naukowe*, gromadzące wyniki badań przeprowadzonych w różnych okresach czasu, i inne. Nawet dwa przykłady z tej książki można łatwo rozszerzyć o funkcjonalność czasową. W bazie danych FIRMA możemy chcieć przechowywać dla każdego pracownika historie atrybutów PENSJA, STANOWISKO i PROJEKT. W bazie danych UNIWERSYTET czas jest już uwzględniony w postaci atrybutów SEMESTR i ROK relacji KURS i PRZEDMIOT, historii ocen w relacji STUDENT oraz informacji o historii grantów badawczych. W zasadzie można przyjąć, że większość aplikacji bazodanowych ma jakiś związek z czasem. Użytkownicy często starają się uprościć lub wręcz zignorować w swoich aplikacjach aspekt czasu, ponieważ wprowadza on liczne komplikacje.

W niniejszym podrozdziale wprowadzimy pewne pojęcia opracowane na potrzeby złożonych czasowych aplikacji bazodanowych. Punkt 26.2.1 przedstawia przegląd sposobów prezentowania czasu w systemach baz danych, różnych typów informacji o czasie, a także różnych wymiarów czasu, które mogą mieć zastosowanie. W punkcie 26.2.2 opisano, jak uwzględniać czas w relacyjnych bazach danych. Punkt 26.2.3 omawia pewne dodatkowe opcje przedstawiania czasu w modelach baz danych pozwalających na używanie obiektów o złożonej strukturze, takich jak obiektowe bazy danych. Punkt 26.2.4 wprowadza zapytania w czasowych bazach danych i pokrótce przedstawia język TSQL2 rozszerzający funkcjonalność SQL o kwestie związane z czasem. Punkt 26.2.5 skupia się na szeregu czasowych, które są niezwykle istotnym typem danych czasowych znajdującym zastosowanie w praktyce.

## 26.2.1. Reprezentacja czasu, kalendarze i wymiary czasu

W czasowych bazach danych czas jest traktowany jako *uporządkowany ciąg punktów* o pewnej określonej przez aplikację **rozdzielczości**. Przypuśćmy na przykład, że aplikacja nigdy nie wymaga, aby jednostki czasu były mniejsze od jednej sekundy. W takim przypadku każdy punkt czasu odpowiada jednej sekundzie. W rzeczywistości każda sekunda jest (krótkim) *okresem czasu*, a nie punktem, ponieważ może być podzielona na milisekundy, mikrosekundy itd. Osoby zajmujące się badaniami związanymi z czasowymi bazami danych używają do określania najmniejszej rozdzielczości czasu potrzebnej w danej aplikacji terminu **chronon**, zamiast określenia *punkt*. Głównym następstwem wyboru minimalnej wymaganej rozdzielczości (np. jedna sekunda) jest fakt, że zdarzenia mające miejsce w tej samej sekundzie będą traktowane jak *zdarzenia równoczesne*, mimo że nie musiały takimi być w rzeczywistości.

Ponieważ nie istnieje żaden znany początek ani koniec czasu, trzeba określić punkt odniesienia, od którego czas będzie liczony. Różne kalendarze opracowane w różnych kulturach (takie jak gregoriański, chiński, islamski, indyjski, żydowski, koptyjski itd.) używają różnych punktów odniesienia. **Kalendarze** służą do uporządkowania czasu w różne jednostki dla wygody stosujących je ludzi. Większość kalendarzy grupuje 60 sekund w minutę, 60 minut w godzinę, 24 godziny w dzień (odpowiadający okresowi obrotu Ziemi wokół własnej osi), a 7 dni w tydzień. Dalsze grupowanie dni w miesiące i miesięcy w lata odpowiada zjawiskom naturalnym związanym z ruchem słońca lub księżyca i jest zazwyczaj nieregularne. W kalendarzu gregoriańskim, stosowanym w większości krajów zachodnich, dni są pogrupowane w miesiące trwające 28, 29, 30 lub 31 dni, a miesiące są pogrupowane w 12-miesięczne lata. Do odwzorowań pomiędzy poszczególnymi jednostkami czasu stosowane są skomplikowane równania.

W SQL2 czasowe typy danych (patrz rozdział 4.) zawierają typ DATE (określający rok, miesiąc i dzień jako RRRR-MM-DD), TIME (określający godzinę, minutę i sekundę jako gg:mm:ss), TIMESTAMP (określający połączenie daty i godziny, posiada opcję dzielenia na jednostki krótsze niż sekunda), INTERVAL (okres czasu taki jak 10 dni lub 250 minut) oraz PERIOD (*osadzony* okres czasu liczony od konkretnego punktu, na przykład 10 dni od 1 stycznia 1999 do 10 stycznia 1999, włącznie z tym punktem)<sup>11</sup>.

**Informacje o zdarzeniach a informacje o stanie (trwającym).** Czasowe bazy danych przechowują informacje związane z pewnymi zdarzeniami lub faktami, które zaistniały. Istnieje kilka różnych typów informacji związanych z czasem. **Fakty**, lub **zdarzenia punktowe**, są zazwyczaj związane w bazie danych z **pojedynczym punktem w czasie** (o pewnej rozdzielczości). Na przykład złożenie depozytu bankowego będzie związane z czasem złożenia tego depozytu, a miesięczne wyniki sprzedaży (fakt) będą związane z konkretnym miesiącem (np. lutym 2010). Zauważmy, że pomimo iż różne typy faktów będą powiązane z czasem o różnej rozdzielczości, zawsze dotyczą *pojedynczej wartości czasu* w bazie danych. Ten typ danych często jest reprezentowany jako **szereg czasowy**, który będzie omówiony w punkcie 26.2.5. Z drugiej strony, **fakty** lub **zdarzenia trwające** są związane z pewnym **okresem czasu**<sup>12</sup>. Przykładem takiego faktu jest stwierdzenie, że pracownik pracował w firmie od 15 sierpnia 2003 do 20 listopada 2008.

**Okres czasu** jest definiowany przez jego moment (punkt) **rozpoczęcia** i **zakończenia** [CZAS-ROZPOCZĘCIA, CZAS-ZAKOŃCZENIA]. Na przykład, przytoczony powyżej okres jest przedstawiany jako [2003-08-15, 2008-11-20]. Taki okres jest najczęściej rozumiany jako *zbiór wszystkich punktów czasu* o pewnej rozdzielczości od czasu rozpoczęcia do czasu zakończenia. Zatem, przyjmując rozdzielczość jednego dnia, przykładowy okres oznacza wszystkie dni od 15 sierpnia 2003 do 20 listopada 2008 włącznie<sup>13</sup>.

**Wymiary czasu rzeczywistego i czasu transakcji.** Dla danego zdarzenia lub faktu powiązanego w bazie danych z określonym punktem lub okresem czasu, powiązanie to można rozumieć w różny sposób. Najbardziej naturalną interpretacją jest ta, że powiązana wartość jest momentem, w którym zdarzenie miało miejsce, lub okresem, w którym fakt był zgodny z prawdą w *świecie rzeczywistym*. Jeśli przyjmiemy taką interpretację, to wartość czasu nazywamy **rzeczywistym czasem** (ang. *valid time*), a czasową bazę danych używającą takiej interpretacji — **bazą danych czasu rzeczywistego**.

Jednakże można użyć też innej interpretacji czasu, gdzie czas przypisany zdarzeniom oznacza czas, kiedy informacja została zapisana w bazie danych, czyli jest to czas systemowy, w którym informacja została potwierdzona w *systemie*<sup>14</sup>. W tym przypadku czas

---

<sup>11</sup> Niestety terminologia nie jest używana konsekwentnie. Na przykład termin okres (ang. *interval*) jest często używany do określania osadzonego okresu (ang. *period*). Będziemy używali konsekwentnie przedstawionej terminologii SQL.

<sup>12</sup> Jest to to samo co *osadzony okres czasu* (ang. *anchored duration*). Często używa się też określenia *interwał czasowy* (ang. *time interval*), ale aby uniknąć zamieszania, będziemy używać określenia zgodnego z terminologią standardu SQL — *okres* (ang. *period*).

<sup>13</sup> Przedstawienie [2003-08-15, 2008-11-21] jest nazywane *przedstawieniem domkniętym*. Można również używać *przedstawienia otwartego* [2003-08-15, 2008-11-21), które oznacza zbiór wszystkich punktów czasu bez ostatniego punktu. Drugi sposób jest czasami wygodniejszy w użyciu, ale dla uniknięcia nieporozumień będziemy używać notacji domkniętej.

<sup>14</sup> Bardziej szczegółowe wyjaśnienie jest przedstawione w punkcie 26.2.3.



w bazie danych nazywamy **czasem transakcji**, a bazę danych — **bazą danych czasu transakcji**.

Można też stosować jeszcze inne interpretacje czasu, ale te dwie są najpopularniejsze i nazywa się je **wymiarami czasu**. W niektórych aplikacjach wystarczy tylko jeden z wymiarów czasu, w innych wymagane są obydwa. W drugim przypadku baza danych jest nazywana **dwuczasową** (ang. *bitemporal*). Jeżeli używane są inne interpretacje czasu określone przez użytkownika i odpowiednio zaprogramowane w aplikacji, to taki czas nazywamy **czasem użytkownika**.

Następny punkt przedstawia przykłady zastosowania powyższych pojęć w relacyjnych bazach danych, a punkt 26.2.3 wprowadza je do obiektowych baz danych.

## 26.2.2. Wprowadzenie czasu do relacyjnych baz danych z obsługą wersji krotek

**Relacje czasu rzeczywistego.** Zobaczmy teraz, jak różne typy czasowych baz danych mogą być zrealizowane w modelu relacyjnym. Na początek przypuśćmy, że chcemy zapisywać historię zmian mających miejsce w rzeczywistym świecie. Rozważmy ponownie bazę danych z rysunku 26.1 i przyjmijmy, że rozdzielczość czasu wynosi jeden dzień. Następnie przekonwertujmy istniejące relacje PRACOWNIK i DZIAŁ w **relacje czasu rzeczywistego**, dodając do nich atrybuty RCR (rzeczywisty czas rozpoczęcia) i RCZ (rzeczywisty czas zakończenia) o typie DATE, co odzwierciedla rozdzielczość jednego dnia. Taka baza danych jest pokazana na rysunku 26.7(a) (relacjom zostały nadane nowe nazwy PRAC\_RC i DZIAŁ\_RC).

Rozważmy, jak relacja PRAC\_RC różni się od niezwiązanej z czasem relacji PRACOWNIK (rysunek 26.1)<sup>15</sup>. W PRAC\_RC każda krotka *V* reprezentuje **wersję** informacji o pracowniku, która jest ważna (w świecie rzeczywistym) tylko w okresie czasu [*V*.RCR, *V*.RCZ], podczas gdy w oryginalnej tabeli PRACOWNIK każda krotka reprezentuje tylko obecny stan (obecną wersję) każdego pracownika. W PRAC\_RC **obecna wersja** pracownika ma zazwyczaj jako czas zakończenia specjalną wartość — *now*. Ta specjalna wartość jest **zmienną czasową**, która niejawnie przedstawia obecny moment upływającego czasu. Nieczasowa relacja PRACOWNIK zawierała tylko te krotki z relacji PRAC\_RC, dla których wartość RCZ jest równa *now*.

Rysunek 26.8 pokazuje kilka krotek wersji z rzeczywistym czasem relacji PRAC\_RC i DZIAŁ\_RC. Istnieją dwie wersje Szewczyka, trzy wersje Wieszczyckiego, jedna wersja Bąka i jedna wersja Napierskiego. Widać teraz, jak relacja czasu rzeczywistego powinna reagować na zmiany informacji. Za każdym razem, gdy jeden lub kilka atrybutów pracownika się **zmienia**, zamiast nadpisywać stare wartości (jakby to miało miejsce w systemie nieczasowym), system powinien stworzyć nową wersję i **zamknąć** dotychczasową, ustawiając odpowiednio jej czas zakończenia (RCZ). Zatem gdy użytkownik wyda komendę aktualizującą pensję Szewczyka na 3000 zł od 1 czerwca 2003, to tworzona jest druga wersja Szewczyka (patrz rysunek 26.8). W chwili tej modyfikacji pierwsza wersja Szewczyka była wersją aktualną z wartością specjalną *now* jako atrybutem RCZ, ale po aktualizacji danych *now* zostało zmienione na 31 maja 2003 (czyli 1 czerwca 2003 minus jeden, w rozdzielczości jednodniowej). Dzięki temu można zaznaczyć, że ta wersja stała się **wersją zamkniętą i historyczną**, a nowa (druga) wersja Szewczyka jest wersją obecną.

<sup>15</sup> Relacje niezwiązane z czasem są też nazywane **relacjami migawkowymi**, ponieważ obrazują one tylko *aktualną migawkę*, czyli *obecną stan* bazy danych.



(a) PRAC\_RC

NAZWISKO	PESEL	PENSJA	NRDZ	PESEL_PRZEŁOŻ	RCR	RCZ
----------	-------	--------	------	---------------	-----	-----

DZIAŁ\_RC

NAZWADZ	NRDZ	SUMA_PENSJI	PESEL_KIEROWNIKA	RCR	RCZ
---------	------	-------------	------------------	-----	-----

(b) PRAC\_CT

NAZWISKO	PESEL	PENSJA	NRDZ	PESEL_PRZEŁOŻ	TCR	TCZ
----------	-------	--------	------	---------------	-----	-----

DZIAŁ\_CT

NAZWADZ	NRDZ	SUMA_PENSJI	PESEL_KIEROWNIKA	TCR	TCZ
---------	------	-------------	------------------	-----	-----

(c) PRAC\_DC

NAZWISKO	PESEL	PENSJA	NRDZ	PESEL_PRZEŁOŻ	RCR	RCZ	TCR	TCZ
----------	-------	--------	------	---------------	-----	-----	-----	-----

DZIAŁ\_DC

NAZWADZ	NRDZ	SUMA_PENSJI	PESEL_KIEROWNIKA	RCR	RCZ	TCR	TCZ
---------	------	-------------	------------------	-----	-----	-----	-----

RYSUNEK 26.7. Różne typy relacyjnych czasowych baz danych. (a) Schemat bazy danych czasu rzeczywistego. (b) Schemat bazy danych czasu transakcji. (c) Dwuczasowa baza danych

PRAC\_RC

NAZWISKO	PESEL	PENSJA	NRDZ	PESEL_PRZEŁOŻ	RCR	RCZ
Szewczyk	65010912345	2500	5	333445555	2002-06-15	2003-05-31
Szewczyk	65010912345	3000	5	333445555	2003-06-01	Now
Wieszczycki	55120834598	2500	4	999887777	1999-08-20	2001-01-31
Wieszczycki	55120834598	3000	5	999887777	2001-02-01	2002-03-31
Wieszczycki	55120834598	4000	5	888665555	2002-04-01	Now
Nowak	74040107777	2800	4	999887777	2001-05-01	2002-08-10
Napierski	62091502054	3800	5	333445555	2003-08-01	Now

...

DZIAŁ\_RC

NAZWADZ	NRDZ	PESEL_KIEROWNIKA	RCR	RCZ
Badania	5	70070705555	2001-09-20	2002-03-31
Badania	5	69060606060	2002-04-01	Now

...

RYSUNEK 26.8. Przykładowe wersje krotek w relacjach czasu rzeczywistego PRAC\_RC i DZIAŁ\_RC

Istotne jest, aby zauważyć, że w relacji czasu rzeczywistego użytkownik musi zwykle dostarczyć informację o czasie aktualizacji. Na przykład zmiana pensji Nowaka mogła być wprowadzona do bazy danych 15 maja 2003 o godzinie 8:52:12, mimo że w rzeczywistości pensja zmieniła się dopiero 1 czerwca 2003. Taka aktualizacja jest nazywana **proaktywną**, ponieważ jest zapisana do bazy danych *zanim* wydarzy się w rzeczywistości. Jeżeli aktualizacja bazy danych ma miejsce *po* momencie, w którym stała się faktem w rze-

czywistym świecie, to mówimy o **aktualizacji retroaktywnej**. Aktualizacja wprowadzana do bazy danych w tym samym czasie, w którym ma miejsce w rzeczywistości, jest **aktualizacją równoczesną**.

Akcji **usunięcia** pracownika w nieczasowej bazie danych będzie odpowiadało *zamknięcie obecnej wersji* usuwanego pracownika w bazie danych czasu rzeczywistego. Na przykład, jeśli Szewczyk zwolni się z pracy w firmie 19 stycznia 2004, to w bazie danych będzie to odzwierciedlone poprzez zmianę wartości atrybutu RCZ obecnej wersji odpowiadającej mu krotki z *now* na 2004-01-19. Na rysunku 26.8 nie ma aktualnej wersji Nowaka, ponieważ opuścił on firmę 10 sierpnia 2002 i dane zostały *logicznie usunięte*. Ponieważ jednak baza danych jest czasowa, to informacja o przeszłym zatrudnieniu nadal jest w niej przechowywana.

Operacji **dodawania** nowego pracownika odpowiada *stworzenie pierwszej wersji krotki* dla tego pracownika (która stanie się wersją obecną) i przypisaniu atrybutowi RCR wartości odpowiadającej rzeczywistej dacie zatrudnienia. Na rysunku 26.7 jest to zilustrowane krotką odpowiadającą Głębockiemu, której pierwsza wersja nie została jeszcze zmieniona.

Zauważmy, że w relacji czasu rzeczywistego *klucze nieczasowe*, takie jak PESEL z tabeli PRACOWNIK, nie są unikatowe dla każdej krotki (wersji). Nowym kluczem w PRAC\_RC jest połączenie klucza nieczasowego i atrybutu czasowego RCR<sup>16</sup>. Jako klucza podstawowego będziemy więc używać pary (PESEL, RCR). Możemy użyć właśnie takiej pary dzięki temu, że w dowolnym momencie powinna istnieć *najwyżej jedna obecna wersja* każdego obiektu. W relacjach czasu rzeczywistego należy więc przestrzegać ograniczenia, które mówi, że dowolne dwie wersje krotki reprezentujące ten sam rzeczywisty obiekt muszą mieć przypisane *nieprzecinające się okresy czasu*. Zauważmy, że jeżeli dozwolona jest modyfikacja nieczasowego klucza głównego, to istotne jest, aby stworzyć **atrybut klucza sztucznego**, którego wartość w rzeczywistości nigdy się nie zmienia i który może posłużyć do wzajemnego powiązania krotek odpowiadających różnym wersjom tego samego obiektu.

Relacje czasu rzeczywistego odzwierciedlają historie zmian, które mają miejsce w *rzeczywistości*. Stany zapisane w bazie danych odpowiadają zatem historii *rzeczywistych zmian* opisywanych obiektów. Ponieważ jednak zmiany mogą być nanoszone w bazie danych proaktywnie lub retroaktywnie, nie są przechowywane informacje o *stanie bazy danych* w wybranym punkcie czasu. Jeżeli dla danej aplikacji istotniejszy jest bieżący stan bazy danych niż historia zmian stanów rzeczywistych, to należy użyć *relacji czasu transakcji*.

**Relacje czasu transakcji.** W bazie danych czasu transakcji każda zmiana odnotowana w bazie danych ma przypisany **znacznik czasu** wykonania danej operacji (modyfikacji, dodania lub usunięcia krotki) przez system bazy danych. Tego typu bazy danych są odpowiednie dla zastosowań, w których zmiany są zazwyczaj zapisywane przy użyciu *aktualizacji równoczesnej* — na przykład wskaźniki giełdowe lub transakcje bankowe. Jeżeli przekonwertujemy bazę danych z rysunku 26.1 na bazę danych czasu transakcji, to relacje PRACOWNIK i DZIAŁ zmienimy w **relacje czasu transakcji**, dodając do nich atrybuty TCR (transakcyjny czas rozpoczęcia) i TCZ (transakcyjny czas zakończenia) typu TIMESTAMP. Zmiany są pokazane na rysunku 26.7(b) w postaci relacji PRAC\_CT i DZIAŁ\_CT.

---

<sup>16</sup> Można również użyć połączenia klucza nieczasowego z atrybutem RCZ.

W PRAC\_CT każda krotka *V* reprezentuje *wersję* informacji o pracowniku, która została zapisana w momencie *V.TCR* i (logicznie) usunięta w momencie *V.TCZ* (ponieważ informacja nie była już aktualna). W PRAC\_CT obecna wersja każdego pracownika ma zazwyczaj przypisaną wartość specjalną *uc* (ang. *Until Changed* — przed zmianą) jako wartość transakcyjnego czasu zakończenia, co oznacza, że krotka zawiera aktualne informacje *przed dokonaniem zmiany* przez inną transakcję<sup>17</sup>. Baza danych czasu transakcji jest często nazywana **bazą danych z wycofaniem** (ang. *rollback dabatase*<sup>18</sup>), ponieważ użytkownik może logicznie cofnąć stan bazy danych do dowolnego momentu w czasie *T* poprzez odczytanie wersji *V* o okresie transakcji [*V.TCR*, *V.TCZ*] zawierającym punkt *T*.

**Relacje dwuczасowe.** Niektóre zastosowania wymagają zapisywania zarówno rzeczywistego czasu zdarzenia, jak i czasu transakcji, co prowadzi do powstania **relacji dwuczасowych**. W przytaczanym tu przykładzie rysunek 26.7(c) przedstawia, jak nieczasowe relacje PRACOWNIK i DZIAŁ z rysunku 26.1 wyglądałyby po przerobieniu ich na relacje dwuczасowe PRAC\_DC i DZIAŁ\_DC. Na rysunku 26.9 przedstawiono kilka przykładowych krotek tych relacji. W tych relacjach krotki z transakcyjnym czasem zakończenia równym *uc* są krotkami reprezentującymi aktualną informację, podczas gdy krotki z wartością *TCZ* będącą konkretnym punktem czasu są krotkami, które były aktualne przed tym punktem. Zatem krotki z wartością *uc* z rysunku 26.9 odpowiadają krotkom czasu rzeczywistego z rysunku 26.7. Atrybut *TCR* dla każdej krotki jest czasem jej zapisania w bazie danych.

PRAC\_RC

NAZWISKO	PESEL	PENSJA	NRDZ	PESEL_PRZEŁOŻ	RCR	RCZ	TCR	TCZ
Szewczyk	65010912345	2500	5	333445555	2002-06-15	Now	2002-06-08, 13:05:58	2003-06-04,08:56:12
Szewczyk	65010912345	2500	5	333445555	2002-06-15	2003-05-31	2003-06-04, 08:56:12	uc
Szewczyk	65010912345	3000	5	333445555	2003-06-01	Now	2003-06-04, 08:56:12	uc
Wieszczycki	55120834598	2500	4	999887777	1999-08-20	Now	1999-08-20, 11:18:23	2001-01-07,14:33:02
Wieszczycki	55120834598	2500	4	999887777	1999-08-20	2001-01-31	2001-01-07, 14:33:02	uc
Wieszczycki	55120834598	3000	5	999887777	2001-02-01	Now	2001-01-07, 14:33:02	2002-03-28,09:23:57
Wieszczycki	55120834598	3000	5	999887777	2001-02-01	2002-03-31	2002-03-28, 09:23:57	uc
Wieszczycki	55120834598	4000	5	888667777	2002-04-01	Now	2002-03-28, 09:23:57	uc
Nowak	74040107777	2800	4	999887777	2001-05-01	Now	2001-04-27, 16:22:05	2002-08-12,10:11:07
Nowak	74040107777	2800	4	999887777	2001-05-01	2002-08-10	2002-08-12, 10:11:07	uc
Napierski	62091502054	3800	5	333445555	2003-08-01	Now	2003-07-28, 09:25:37	uc

...

DZIAŁ\_RC

NAZWADZ	NRDZ	PESEL_KIEROWNIKA	RCR	RCZ	TCR	TCZ
Badania	5	37111045873	2001-09-20	Now	2001-09-15,14:52:12	2001-03-28,09:23:57
Badania	5	37111045873	2001-09-20	1997-03-31	2002-03-28,09:23:57	uc
Badania	5	55120834598	2002-04-01	Now	2002-03-28,09:23:57	uc

RYSUNEK 26.9. Przykładowe wersje krotek w dwuczасowych relacjach PRAC\_DC i DZIAŁ\_DC

<sup>17</sup> Zmienna *uc* w relacjach czasu transakcji odpowiada zmiennej *now* w relacjach czasu rzeczywistego. Znaczenie tych zmiennych jest jednak trochę inne.

<sup>18</sup> Użyty tutaj termin *wycofanie* nie ma tego samego znaczenia co *wycofanie transakcji* (patrz rozdział 23.) podczas odzyskiwania danych, kiedy to elementy transakcji są *fizycznie anulowane*. Tutaj operacje są anulowane jedynie *logicznie*, co pozwala na sprawdzenie jaki był stan bazy danych w określonym momencie w przeszłości.

Rozważmy teraz **operację aktualizacji** w relacji dwuczasewej. W tym modelu dwuczasewych baz danych<sup>19</sup> *fizycznej zmianie nie ulegają żadne atrybuty* krotek (z wyjątkiem atrybutu TCZ z wartością uc)<sup>20</sup>. Aby zobrazować tworzenie krotek, rozważmy relację PRAC\_DC. *Aktualna wersja* V pracownika ma przypisaną atrybutowi TCZ wartość uc, a atrybutowi RCZ wartość now. Jeżeli zmianie ulega jeden z atrybutów (na przykład PENSJA), to transakcja T zapisująca tę zmianę w bazie danych powinna mieć dwa parametry: nową wartość atrybutu PENSJA i moment rozpoczęcia MR, od którego zaczyna (w rzeczywistości) obowiązywać nowe wynagrodzenie. Przyjmijmy, że MR- jest punktem przed rzeczywistym MR (w granicach określonych przyjętą rozdzielczością czasu), a transakcja ma znacznik czasu ZC(T). W tej sytuacji do tabeli PRAC\_DC należy wprowadzić następujące fizyczne zmiany:

- (1) Utworzyć kopię V2 aktualnej wersji V; ustawić V2.RCZ na MR-, V2.TCR na ZC(T), V2.TCZ na uc oraz dodać V2 do tabeli PRAC\_DC; V2 jest kopią wcześniej aktualnej wersji V *po tym, jak została ona zamknięta w czasie rzeczywistym MR-*.
- (2) Utworzyć kopię V3 aktualnej wersji V; ustawić V3.RCZ na MR, V3.RCZ na now, V3.PENSJA na nową wartość pensji, V3.TCR na ZC(T), V3.TCZ na uc oraz dodać V3 do PRAC\_DC; V3 przedstawia nową aktualną wersję.
- (3) Ustawić V.TCZ na ZC(T), ponieważ wersja V nie zawiera już aktualnych informacji.

Jako ilustrację przykładu rozważmy pierwsze trzy krotki V1, V2 i V3 w tabeli PRAC\_DC z rysunku 26.9. Przed aktualizacją pensji Nowaka z 2500 na 3000 w tabeli istniała tylko krotka V1 i przedstawiała ona aktualną wersję z TCZ równym uc. Następnie transakcja T o znaczniku czasu ZC(T) równym 2003-06-04 08:56:12 zaktualizowała informacje o pensji na 3000 z rzeczywistym czasem równym 2003-06-01. Została stworzona krotka V2, która jest kopią V1 z RCZ ustawionym na 2003-05-31 (czyli o jeden dzień mniej niż rzeczywisty czas rozpoczęcia dla nowej pensji) i TCR równym znacznikowi ZC(T). Stworzona została również krotka V3, która przechowuje wartość nowej pensji, ma RCR ustawiony na 2003-06-01 a TCR na znacznik czasu transakcji aktualizującej. Ostatecznie wartość TCZ dla V1 jest również ustawiana na znacznik czasu transakcji, czyli 2003-06-04 08:45:12. Zauważmy, że jest to *aktualizacja retroaktywna*, gdyż transakcja ma miejsce 4 czerwca 2003, ale dotyczy zmiany, która w rzeczywistości odbyła się 1 czerwca 2003.

Podobnie, podczas zmiany wynagrodzenia i działu Kowalskiego (w tym samym czasie) na 3000 i 5., znacznik czasu transakcji aktualizującej jest równy 2001-01-07 14:33:02, a rzeczywisty czas rozpoczęcia 2001-02-01. Jest to zatem *aktualizacja proaktywna*, ponieważ transakcja ma miejsce 7 stycznia 2001, a rzeczywisty czas zdarzenia to 1 lutego 2001. W tym przypadku krotka V4 jest logicznie zastąpiona krotkami V5 i V6.

Przyjrzyjmy się teraz na przykładzie krotek V9 i V10 z relacji PRAC\_DC z rysunku 26.9, jak zrealizowana jest **operacja kasowania** w relacji dwuczasewej. Pracownik Jankowski opuścił firmę 10 sierpnia 2002, a logiczne usunięcie z bazy danych zostało zrealizowane przez transakcję T o ZC(T) równym 2002-08-12 10:11:07. Przed tym momentem V9 było aktualną wersją Jankowskiego z TCZ równym uc. Logiczne usunięcie jest zrealizowane poprzez ustawienie V9.TCZ na 2002-08-12 10:11:07 w celu unieważnienia wersji oraz

<sup>19</sup> Istnieje wiele zaproponowanych modeli dwuczasewych baz danych. Tutaj opisywane są tylko wybrane z nich w celu zilustrowania samego pojęcia.

<sup>20</sup> Niektóre modele dopuszczają również modyfikację atrybutu RCZ, ale interpretacja zapisanych danych jest w nich inna od tutaj przedstawianej.

stworzenia dla Jankowskiego *wersji ostatecznej* V10 z RCZ=2002-08-10 (patrz rysunek 26.9). **Operacja dodawania** jest realizowana poprzez stworzenie *pierwszej wersji*, jak zostało to pokazane w krotce V11 tabeli PRAC\_DC.

**Rozważania implementacyjne.** Istnieją różne możliwości przechowywania krotek w relacji czasowej. Jedną z nich jest zapisywanie wszystkich krotek w jednej tabeli, tak jak to zostało pokazane na rysunkach 26.8 i 26.9. Inną możliwością jest stworzenie dwóch tabel: jednej z aktualnymi informacjami i drugiej z danymi historycznymi. Na przykład w dwuczasowej relacji PRAC\_DC krotki z TCZ równym uc i RCZ równym now byłyby w jednej relacji (*tabeli aktualnej*), ponieważ przechowują informacje aktualne (odzwierciedlają bieżący moment czasu), a wszystkie pozostałe krotki byłyby zapisane w drugiej tabeli. Pozwala to administratorowi bazy danych określać różne ścieżki dostępu, takie jak indeksy, a także ułatwia zarządzanie rozmiarem tabel. Innym rozwiązaniem jest stworzenie trzeciej tabeli z poprawionymi krotkami, których TCZ jest różny od uc.

Kolejną możliwością jest *pionowy podział* atrybutów w relacji czasowej na osobne relacje. Uzasadnieniem takiego podziału jest fakt, że w przypadku gdy relacja ma wiele atrybutów, za każdym razem, gdy choć jeden z nich się zmienia, trzeba tworzyć całą nową krotkę. Jeśli atrybuty są modyfikowane asynchronicznie, to powoduje to niepotrzebne nagromadzenie krotek różniących się tylko jedną wartością. Jeżeli określimy osobną relację zawierającą atrybuty, które *zawsze zmieniają się równocześnie*, oraz powtórzone atrybuty klucza podstawowego, to mówimy, że baza danych jest w **czasowej postaci normalnej**. Aby jednak połączyć z powrotem pełne informacje, musi zostać wykonana odmiana złączenia nazywana **czasowym złączeniem przecięć** (ang. *temporal intersection join*), która jest dość kosztowna w implementacji.

Należy zapamiętać, że w dwuczasowych bazach danych zapisywane są wszystkie zmiany. Możliwe jest nawet zapisywanie historii poprawek. Można na przykład przechowywać dwie wersje krotek odpowiadające temu samemu pracownikowi z tym samym rzeczywistym czasem, ale innymi atrybutami, o ile zachowana jest różnica w wartości czasu transakcji. W takim przypadku wersja z późniejszym czasem transakcji jest **poprawką** pozostających wersji. Można w ten sposób wprowadzać korekty dotyczące również czasu rzeczywistego. Błędne zapisy pozostaną dostępne dla zapytań jako przeszłe stany bazy danych. System pozwalający na przechowywanie pełnej historii zmian i poprawek nazywany jest **systemem tylko do dołączania** (ang. *append only database*).

### 26.2.3. Czas w obiektowych bazach danych z obsługą wersji atrybutów

Poprzedni punkt omawiał **zastosowanie wersji krotek** do realizacji czasowych baz danych. W takim modelu zmiana pojedynczego atrybutu pociąga za sobą konieczność dodania nowej krotki, mimo że wszystkie pozostałe atrybuty pozostają takie same. Odmienne podejście do problemu można zastosować w systemach obsługujących **obiekty o złożonej strukturze**, takie jak obiektowe (patrz rozdział 11.) lub obiektowo-relacyjne systemy baz danych. Taki model nazywamy **pracą z wersjami atrybutów** (ang. *attribute versioning*).

W modelu pracy z wersjami atrybutów wszystkie zmiany dotyczące atrybutów danego obiektu są przechowywane w jego pojedynczej instancji. Atrybuty, które zmieniają się wraz z upływem czasu, nazywamy **atrybutami zmiennymi czasowo**, a ich wartość jest

podzielona na wersje, którym przypisane są okresy czasu. Okresy mogą reprezentować rzeczywisty czas, czas transakcji, lub mogą być dwuczasowe (w zależności od potrzeb aplikacji). Atrybuty, które się nie zmieniają, nazywamy **niezmiennymi czasowo**; nie są one powiązane z informacjami o czasie. Aby to zilustrować, rozważmy przykład z rysunku 26.10, który przedstawia obiekt PRACOWNIK w języku ODL (patrz rozdział 11.) z atrybutami uwzględniającymi zmiany według czasu rzeczywistego. Przyjęto tutaj, że nazwisko oraz numer PESEL są atrybutami niezmiennymi czasowo, a pensja, dział i przełożony są zmienne czasowo (mogą się zmieniać wraz z upływem czasu). Każdy atrybut zmienny czasowo jest reprezentowany przez listę krotek <RZECZ\_CZAS\_ROZP, RZECZ\_CZAS\_ZAK, WARTOŚĆ> uporządkowaną według czasu rozpoczęcia.

```
class Czasowa_Pensja
{
    attribute Date rzecz_czas_rozp;
    attribute Date rzecz_czas_zak;
    attribute float pensja;
};

class Czasowy_Dział
{
    attribute Date rzecz_czas_rozp;
    attribute Date rzecz_czas_zak;
    attribute Dział_RC dział;
};

class Czasowy_Przełożony
{
    attribute Date rzecz_czas_rozp;
    attribute Date rzecz_czas_zak;
    attribute Pracownik_RC przełożony;
};

class Czasowy_Okres_Życia {
    attribute Date rzecz_czas_rozp;
    attribute Date rzecz_czas_zak;
};

class Pracownik_RC
( extent pracownicy )
{
    attribute list<Czasowy_Okres_Życia> okres_życia;
    attribute string nazwisko;
    attribute string pesel;
    attribute list<Czasowa_Pensja> hist_pensja;
    attribute list<Czasowy_Dział> hist_dział;
    attribute list<Czasowy_Przełożony> hist_przełożony;
};
```

RYСУNEK 26.10. Przykładowy schemat ODL dla klasy Pracownik\_RC używającej czasowych wersji atrybutów



W tym modelu każdorazowa modyfikacja atrybutu powoduje *zamknięcie* aktualnej wersji oraz dodanie **nowej wersji atrybutu** do listy. Pozwala to na asynchroniczną modyfikację poszczególnych atrybutów. Aktualna wartość ma przypisaną wartość specjalną *now* jako RZECZ\_CZAS\_ZAK. Przy pracy z wersjami atrybutów użyteczne jest określenie **atrybutu czasu życia** powiązanego z całym obiektem i przechowującego jeden lub kilka okresów istnienia całego obiektu. Logiczne usunięcie obiektu jest realizowane za pomocą zamknięcia czasu życia. W tym podejściu należy wymusić zawieranie się wszystkich okresów przypisywanych poszczególnym atrybutom w okresie czasu życia całego obiektu.

W dwuczasowych bazach danych każda wersja atrybutu byłaby krotką z pięcioma elementami:

```
<rzecz_czas_rozp, rzecz_czas_zak, trans_czas_rozp, trans_czas_zak, wartość>
```

Okres życia obiektu również powinien zawierać w takim przypadku obydwa stosowane wymiary czasu. Pełne możliwości dwuczasowych baz danych mogą być zatem osiągnięte przy użyciu wersji atrybutów. Do aktualizacji ich wersji można zastosować mechanizmy analogiczne do omawianych wcześniej mechanizmów aktualizacji wersji krotek.

## 26.2.4. Konstruowanie zapytań czasowych i język TSQL2

Dotychczas omówione zostały możliwości rozszerzenia różnych modeli danych o konstrukcje związane z czasem. Przedstawimy teraz, jak muszą być rozszerzone mechanizmy zapytań w celu obsługi informacji czasowych zapisanych w bazie danych. Następnie zostanie krótko omówiony język TSQL2 rozszerzający funkcjonalność SQL o zapytania związane z rzeczywistym czasem, czasem transakcji oraz o zapytania dwuczasowe.

W nieczasowych relacyjnych bazach danych typowe warunki wyboru ograniczają się do warunków narzuconych atrybutom, a wynik zawiera zestaw *aktualnych krotek* o atrybutach spełniających te warunki. Co za tym idzie, atrybut, którego dotyczy zapytanie, jest *efektem projekcji* (patrz rozdział 6.). Przykładowo, w zapytaniu o nazwiska wszystkich pracowników działu 5., których pensja jest wyższa niż 3000, warunkiem wyboru byłoby:

```
((PENSJA > 3000) AND (NRDZ=5))
```

Atrybutem projekcji będzie NAZWISKO. W czasowej bazie danych warunki wyboru poza atrybutami mogą dotyczyć również czasu. **Czysty warunek czasowy** dotyczy wyłącznie czasu — na przykład wybranie wszystkich pracowników w danym *momencie czasu*  $t$ , lub w *okresie*  $[t1, t2]$ . W tym przypadku podany okres czasu jest porównywany z okresem zapisanym w każdej krotce wersji  $[t.RCR, t.RCZ]$  i rezultatem są tylko te krotki, które spełniają zadany warunek. W tego typu kryteriach przyjmuje się, że okres jest zbiorem wszystkich punktów czasu od  $t1$  do  $t2$  włącznie, dzięki czemu można stosować standardowe operatory porównujące zbiory. Dodatkowo potrzebne są również operatory sprawdzające, czy jeden okres kończy się *zanim* inny się zaczyna<sup>21</sup>.

Najpopularniejsze operatory używane w zapytaniach czasowych to:

```
[t.RCR, t.RCZ] INCLUDES [t1, t2] odpowiada  $t1 \geq t.RCR$  AND  $t2 \leq t.RCZ$ 
```

```
[t.RCR, t.RCZ] INCLUDED_IN [t1, t2] odpowiada  $t1 \leq t.RCR$  AND  $t2 \geq t.RCZ$ 
```

<sup>21</sup> Pełny zbiór operatorów porównujących okresy czasu jest zdefiniowany jako **algebra Allena**.



$[t.RCR, t.RCZ]$  OVERLAPS  $[t1, t2]$  odpowiada  $(t1 \leq t.RCZ \text{ AND } t2 \geq t.RCR)^{22}$   
 $[t.RCR, t.RCZ]$  BEFORE  $[t1, t2]$  odpowiada  $t1 \geq t.RCZ$   
 $[t.RCR, t.RCZ]$  AFTER  $[t1, t2]$  odpowiada  $t2 \leq t.RCR$   
 $[t.RCR, t.RCZ]$  MEETS\_BEFORE  $[t1, t2]$  odpowiada  $t1 = t.RCZ + 1^{23}$   
 $[t.RCR, t.RCZ]$  MEETS\_AFTER  $[t1, t2]$  odpowiada  $t2 + 1 = t.RCR$

Ponadto, potrzebne są operacje do manipulowania okresami czasu, takie jak obliczanie złączenia lub przecięcia dwóch okresów. Wyniki takich działań nie muszą być okresami, ale raczej **elementami czasu** — kolekcjami jednego lub więcej *rozłącznych* okresów, z których żadne dwa bezpośrednio do siebie nie przylegają, czyli dla dowolnych dwóch okresów czasu  $[t1, t2]$  i  $[t3, t4]$  w elemencie czasu muszą zachodzić następujące warunki:

- **przecięcie**  $[t1, t2]$  i  $[t3, t4]$  jest puste;
- $t3$  nie jest punktem czasu następnym po  $t2$  (w danej rozdzielczości czasu);
- $t1$  nie jest punktem czasu następnym po  $t4$  (w danej rozdzielczości czasu).

Ostatnie dwa warunki są potrzebne do zapewnienia jednoznaczności reprezentacji elementów czasowych. Jeżeli dwa okresy  $[t1, t2]$  i  $[t3, t4]$  są do siebie przyległe, to są łączone w jeden okres  $[t1, t4]$ . Taką operację nazywamy **łączeniem** okresów i dotyczy ona również przecinających się okresów czasu.

Aby zilustrować użycie czystych warunków czasowych, przypuśćmy, że użytkownik chce odczytać wszystkie wersje krotek pracowników, które były ważne w 2002 roku. Prawidłowy warunek zastosowany do relacji z rysunku 26.8 to:

$[T.RCR, T.RCZ]$  OVERLAPS  $[2002-01-01, 2002-12-31]$

Większość zapytań związanych z czasem odnosi się do czasu rzeczywistego. W dwuczasiowych bazach danych odczytuje się zazwyczaj aktualne wpisy, które jako końcowy czas transakcji mają wartość *uc*. Jednakże, jeśli zapytanie ma dotyczyć wcześniejszego stanu bazy danych, do zapytania należy dodać wyrażenie *AS OF T*, które oznacza, że zapytanie ma być zastosowane do krotek aktualnych w bazie danych z momentu *T*.

Poza zapytaniami czysto czasowymi możliwe są zapytania z **warunkami dotyczącymi wartości atrybutów i czasu**. Przykładowo, przypuśćmy, że chcemy odczytać wszystkie wersje *T* krotek *PRAC\_RC* odpowiadające pracownikom zatrudnionym w dziale 5. w roku 2002. W tym przypadku warunek zapytania będzie następujący:

$([T.RCR, T.RCZ]$  OVERLAPS  $[2002-01-01, 202-12-31])$  AND  $(T.NRD=5)$

Przedstawimy teraz podstawy języka TSQL2, który rozszerza SQL o konstrukcje właściwe czasowym bazom danych. Istotą TSQL2 jest umożliwienie użytkownikom określenia, czy dana relacja jest czasowa, czy nie (czyli jest standardową relacją SQL). Wyrażenie *CREATE TABLE* jest rozszerzone o *opcjonalną* klauzulę *AS*, która pozwala na zdefiniowanie następujących związanych z czasem opcji:

<sup>22</sup> Ten operator zwraca prawdę, jeśli *przecięcie* dwóch okresów jest zbiorem niepustym; jest równoważny z operatorem *INTERSECTS\_WITH*.

<sup>23</sup> 1 (jeden) oznacza tutaj jeden punkt czasu w określonej rozdzielczości. Operatory z rodziny *MEET* określają, czy jeden okres zaczyna się bezpośrednio po zakończeniu drugiego.

- AS VALID STATE <ROZDZIELCZOŚĆ> (relacja czasu rzeczywistego z okresami czasu rzeczywistego),
- AS VALID EVENT <ROZDZIELCZOŚĆ> (relacja czasu rzeczywistego z punktami czasu rzeczywistego),
- AS TRANSACTION <ROZDZIELCZOŚĆ> (relacja czasu transakcji z okresami czasu transakcji),
- AS VALID STATE <ROZDZIELCZOŚĆ> AND TRANSACTION (relacja dwuczасowa z okresami czasu rzeczywistego),
- AS VALID EVENT <ROZDZIELCZOŚĆ> AND TRANSACTION (relacja dwuczасowa z punktami czasu rzeczywistego).

Słowa kluczowe STATE i EVENT służą do określania, czy z danym wymiarem czasu są powiązane *okresy*, czy *punkty* czasu. W TSQL2 użytkownik nie zna szczegółów implementacji relacji czasowych (tak jak to zostało przedstawione we wcześniejszych punktach); posiada jedynie dostęp do konstrukcji języka, które pozwalają na korzystanie z różnych typów wyborów zależnych od czasu, projekcji, agregacji, konwersji pomiędzy rozdzielczościami czasu i wielu innych. Język ten jest dokładniej opisany w książce autorstwa Snodgrassa i in. (1995).

### 26.2.5. Szeregi czasowe

Szeregi czasowe są bardzo często używane w aplikacjach finansowych, handlowych i ekonomicznych. Dotyczą wartości, które są zapisywane według z góry zdefiniowanego ciągu punktów w czasie. Jest to zatem specjalny typ danych związanych z **punktami czasu rzeczywistego**, przy czym punkty te są ułożone w harmonogram znany z góry. Rozważmy przykład dziennych cen zamknięcia określonej spółki na giełdzie papierów wartościowych. Rozdzielczością będzie tu dzień, ale dni otwarcia giełdy są znane (dni robocze z wyjątkiem świąt). Opracowano więc procedury obliczające **kalendarze** powiązane z szeregami czasowymi. Typowe zapytania w takiej bazie danych wiążą się z **agregacją** wartości w dłuższych okresach czasu, na przykład znajdowanie średniej lub maksymalnej *tygodniowej* ceny zamknięcia, czy *miesięcznego* minimum i maksimum na podstawie *codziennych* notowań.

Jako inny przykład rozważmy informacje o wysokości sprzedaży w sieci sklepów wybranej firmy. Ponownie, typowymi zapytaniami agregującymi będzie odczytywanie tygodniowych, miesięcznych i rocznych obrotów na podstawie codziennych informacji o sprzedaży (za pomocą agregującej funkcji sumowania) lub porównywanie wyników sprzedaży pomiędzy różnymi miesiącami.

W związku ze szczególną naturą szeregów czasowych i brakiem obsługi takiego typu danych w starszych SZBD często zamiast systemów baz danych ogólnego zastosowania używa się wyspecjalizowanych **systemów zarządzania szeregami czasowymi**. W takich systemach najczęściej dane są zapisywane kolejno do pliku, a do ich analizowania służą specjalnie opracowane do tego celu procedury. Wadą tego typu systemów jest brak pełni możliwości języków zapytań wysokiego poziomu takich jak SQL.

Obecnie niektóre z dostępnych na rynku pakietów SZBD zawierają rozszerzenia obsługujące szeregi czasowe (na przykład Time Series Cartridge firmy Oracle i TimeSeries DataBlade dla Informix Universal Server). Ponadto, obsługa tabel zdarzeń w TSQL2 oferuje pewną obsługę szeregów czasowych.

## 26.3. Zagadnienia z obszaru przestrzennych baz danych<sup>24</sup>

### 26.3.1. Wprowadzenie do przestrzennych baz danych

**Przestrzenne bazy danych** oferują funkcjonalność pozwalającą na pracę z obiektami w przestrzeni wielowymiarowej. Na przykład, systemy kartograficzne przechowujące mapy zawierają dwuwymiarowe opisy obiektów geograficznych od krajów i stanów po rzeki, miasta, drogi, morza itp. Tego typu aplikacje są nazywane aplikacjami typu **GIS** (ang. *Geographical Information Systems* — systemy informacji geograficznej) i są stosowane w ochronie środowiska, transporcie, zarządzaniu kryzysowym oraz na polach bitew. Inne bazy danych, takie jak systemy meteorologiczne przechowujące dane pogodowe, są trójwymiarowe, ponieważ temperatury i inne dane meteorologiczne są związane z punktami w przestrzeni trójwymiarowej. Ogólnie, przestrzenne systemy baz danych przechowują obiekty, które są opisywane przez cechy związane z położeniem w przestrzeni. Relacje pomiędzy obiektami w przestrzeni są szczególnie istotne, ponieważ w tego typu aplikacjach są bardzo często wykorzystywane w zapytaniach. Co prawda, ogólnie przestrzenne bazy danych mogą dotyczyć  $n$ -wymiarowej przestrzeni dla dowolnego  $n$ , ale dla ilustracji naszych rozważań ograniczymy się do przestrzeni dwuwymiarowej.

Przestrzenne bazy danych są optymalizowane pod kątem przechowywania i pobierania danych dotyczących obiektów w przestrzeni, w tym punktów, linii i wielokątów. Znanym przykładem są tu obrazy satelitarne danych przestrzennych. Zapytania o dane przestrzenne, gdzie predykatami selekcji są parametry przestrzenne, są nazywane **zapytaniami przestrzennymi**. Oto przykładowe zapytanie przestrzenne: „Podaj nazwy wszystkich księgarń w odległości do 5 kilometrów od budynku Collegium Maius Uniwersytetu Jagiellońskiego”. Ponieważ typowe bazy danych przetwarzają dane liczbowe i tekstowe, na potrzeby przetwarzania przestrzennych typów danych należy udostępnić dodatkowe mechanizmy. Zapytanie typu: „Podaj wszystkich klientów zamieszkałych w odległości do 30 kilometrów od siedziby firmy” wymaga przetwarzania przestrzennych typów danych, co zwykle wykracza poza zakres standardowej algebry relacyjnej i może obejmować korzystanie z zewnętrznej geograficznej bazy danych, która odwzorowuje lokalizację siedziby firmy i klientów na dwuwymiarową mapę na podstawie adresów. Każdy klient jest wtedy łączony z lokalizacją <szerokość geograficzna, długość geograficzna>. Tradycyjny indeks w postaci B<sup>+</sup>-drzewa oparty na kodach pocztowych klientów i innych atrybutach nieprzestrzennych nie jest przydatny do przetwarzania takiego zapytania, ponieważ tradycyjne indeksy nie potrafią porządkować wielowymiarowych danych ze współrzędnymi. Dlatego niezbędne są bazy danych dostosowane do obsługi danych i zapytań przestrzennych.

W tabeli 26.1 wymieniono typowe operacje analityczne związane z przetwarzaniem danych geograficznych lub przestrzennych<sup>25</sup>. **Operacje pomiaru** służą do pomiaru globalnych właściwości pojedynczych obiektów (np.: obszaru, odległości między częściami obiektu,

<sup>24</sup> Dziękujemy Praneshowi Parimali Ranganathanowi za wkład w powstanie tego podrozdziału.

<sup>25</sup> Lista operacji analiz GIS jest oparta na pracy Albrechta (1996).

zwięzłości lub symetrii) oraz względnej pozycji różnych obiektów w kategoriach odległości i kierunku. Operacje **analizy przestrzennej**, w których często używane są techniki statystyczne, służą do wykrywania *relacji przestrzennych* w ramach odwzorowanych warstw danych i między takimi warstwami. Przykładem jest utworzenie *mapy prognozytycznej*, która określa miejsce zamieszkania potencjalnych nabywców konkretnych produktów na podstawie historycznych danych sprzedażowych i demograficznych. Operacje **analizy przepływu** pomagają ustalić najkrótszą ścieżkę między dwoma punktami oraz występowanie połączeń między węzłami lub obszarami grafu. Celem **analiz lokalizacji** jest stwierdzenie, czy dany zbiór punktów i linii znajduje się w obrębie określonego wielokąta (danej lokalizacji). Ten proces obejmuje generowanie bufora wokół istniejących obiektów geograficznych i identyfikowanie lub pobieranie obiektów na podstawie tego, czy znajdują się wewnątrz tego bufora, czy poza nim. **Cyfrowe analizy terenu** służą do generowania trójwymiarowych modeli, w których topografię lokalizacji geograficznej można reprezentować za pomocą modelu danych  $x, y, z$  znanego jako DTM/DEM (ang. *Digital Terrain Model* lub *Digital Elevation Model*). W DTM wymiary  $x$  i  $y$  reprezentują płaszczyznę poziomą, a  $z$  określa wysokość punktu o współrzędnych  $x$  i  $y$ . Takie modele służą do analiz danych środowiskowych lub tworzenia projektów inżynierskich wymagających informacji o terenie. Wyszukiwanie przestrzenne umożliwia użytkownikom szukanie obiektów w określonym obszarze. Przykładowo, **wyszukiwanie tematyczne** pozwala szukać obiektów powiązanych z danym tematem lub klasą — np. „Znajdź wszystkie zbiorniki wodne w odległości 40 kilometrów od Krakowa”, gdzie klasą jest *woda*.

TABELA 26.1. Standardowe typy analiz danych przestrzennych

Rodzaje analiz	Rodzaje operacji i pomiarów
Pomiary	Odległość, obwód, kształt, przystawanie, kierunek
Analizy i statystyki przestrzenne	Wzorec, autokorelacja, indeksy podobieństwa oraz topologia dla danych przestrzennych i nieprzestrzennych
Analizy przepływu	Łączność i najkrótsze ścieżki
Analizy lokalizacji	Analizy punktów i linii w wielokątach
Analizy terenu	Nachylenie, zlewisko, sieć rzeczna
Wyszukiwanie	Wyszukiwanie tematyczne, wyszukiwanie w obszarach

Występują też **relacje topologiczne** między obiektami przestrzennymi. Często wykorzystuje się je w predykcjach logicznych do selekcji obiektów na podstawie relacji przestrzennych między nimi. Przykładowo, jeśli granice miasta są reprezentowane jako wielokąty, a autostrady jako linie, warunek „Znajdź wszystkie autostrady przechodzące przez Warszawę” polega na wyznaczeniu części *wspólnej* w celu ustalenia, które autostrady (linie) przecinają się z granicami miasta (wielokątem).

## 26.3.2. Typy i modele danych przestrzennych

W tym punkcie pokrótce opisujemy standardowe typy danych i modele używane do składowania danych przestrzennych. Takie dane mają trzy podstawowe postacie. Stały się one niepisanym standardem z powodu ich powszechnego stosowania w systemach komercyjnych.

- **Dane dotyczące map**<sup>26</sup> obejmują różne geograficzne lub przestrzenne cechy obiektów z mapy, np. kształt i lokalizację określonego obiektu na mapie. Trzy podstawowe cechy to punkty, linie i wielokąty (obszary). **Punkty** służą do reprezentowania przestrzennych cech obiektów, których lokalizację określają współrzędne dwuwymiarowe (x,y lub długość i szerokość geograficzna) w skali odpowiedniej dla danej aplikacji. W zależności od skali punkty mogą reprezentować np.: budynki, wieże telefonii komórkowej lub nieruchome pojazdy. Poruszające się pojazdy i inne obiekty można przedstawiać za pomocą sekwencji zmieniających się w czasie lokalizacji punktów. **Linie** są reprezentacją obiektów o określonej długości (np. drogi lub rzeki), których cechy przestrzenne można w przybliżeniu przedstawić za pomocą sekwencji połączonych linii. **Wielokąty** służą do przedstawiania przestrzennych cech obiektów mających granicę, np.: państw, województw, jezior lub miast. Warto zauważyć, że niektóre obiekty (np. budynki lub miasta) mogą być reprezentowane albo jako punkty, albo jako wielokąty, zależnie od skali.
- **Dane o atrybutach** to dane opisowe, które systemy GIS łączą z **obiektami z map**. Załóżmy, że mapa obejmuje obiekty reprezentujące gminy w województwie w Polsce (np. małopolskim lub opolskim). Atrybutami każdego takiego obiektu mogą być: liczba mieszkańców, największe miasto, powierzchnia w kilometrach kwadratowych itd. Dla innych obiektów z mapy, takich jak województwa, miasta, okręgi wyborcze czy obszary spisu powszechnego, można uwzględnić inne atrybuty.
- **Dane obrazowe** to dane takie jak obrazy satelitarne lub zdjęcia powietrzne, zwykle wykonywane za pomocą aparatów fotograficznych. Na takie obrazy można nakładać zidentyfikowane ważne obiekty, np. budynki i drogi. Zdjęcia mogą być też atrybutami obiektów z map. Możesz dodawać obrazy do obiektów, aby kliknięcie danego elementu powodowało wyświetlenie zdjęcia. Obrazy powietrzne i satelitarne to typowe przykłady danych rastrowych.

**Modele informacji przestrzennych** są czasem dzielone na dwie ogólne kategorie: *modele pól* i *modele obiektowe*. Aplikacje przestrzenne (np. do obsługi zdalnych czujników lub sterowania ruchem na drogach) są modelowane na podstawie jednej z tych kategorii w zależności od wymagań i modeli tradycyjnie stosowanych w aplikacjach określonego rodzaju. **Modele pól** często służą do modelowania danych przestrzennych mających charakter ciągły, np.: wysokości terenu, danych o temperaturze i charakterystyki gleby. **Modele obiektowe** są zwykle stosowane do sieci transportowych, działek, budynków i innych obiektów posiadających atrybuty zarówno przestrzenne, jak i nieprzestrzenne.

### 26.3.3. Operatory i zapytania przestrzenne

Operatory przestrzenne służą do rejestrowania wszystkich istotnych geometrycznych cech obiektów z przestrzeni fizycznej i relacji między tymi obiektami, a także do przeprowadzania analiz przestrzennych. Te operatory należą do trzech ogólnych kategorii.

---

<sup>26</sup> Podane rodzaje danych geograficznych są oparte na wskazówkach organizacji ESRI dotyczących systemów GIS (patrz [http://www.gis.com/implementing\\_gis/data/data\\_types.html](http://www.gis.com/implementing_gis/data/data_types.html)).

- **Operatory topologiczne.** Właściwości topologiczne są niezmiennikami w trakcie wykonywania transformacji topologicznych. Te właściwości nie zmieniają się w wyniku przekształceń takich jak rotacja, translacja lub skalowanie. Operatory topologiczne są hierarchicznie ustrukturyzowane na wielu poziomach. Na podstawowym poziomie operatory umożliwiają sprawdzanie szczegółowych relacji topologicznych między obszarami z szerokimi granicami, a na wyższych poziomach znajdują się bardziej abstrakcyjne operatory, pozwalające użytkownikom zgłaszać zapytania o niepewne dane przestrzenne niezależnie od używanego geometrycznego modelu danych. Przykładami są operacje otwórz(obszar), zamknij(obszar) lub wewnątrz(punkt, obrys).
- **Operatory projektywne.** Operatory projektywne, np. *wyznaczanie powłoki wypukłej*, służą do zapisywania predykatów związanych z wklęsłością lub wypukłością obiektów, a także z innymi relacjami przestrzennymi (np. znajdowania się na wklęsłej powierzchni danego obiektu).
- **Operatory metryczne.** Te operatory służą do dokładnego opisywania geometrii obiektów. Pozwalają mierzyć globalne cechy pojedynczych obiektów (np.: obszar, względną wielkość części obiektu, zwartość i symetrię), a także względną lokalizację różnych obiektów (z uwzględnieniem odległości i kierunku). Przykładowe operacje tego typu to długość(łuk) i odległość(punkt, punkt).

**Dynamiczne operatory przestrzenne.** Operacje wykonywane za pomocą wymienionych wcześniej operatorów są statyczne (przeprowadzenie operacji nie wpływa na stan operandów). Przykładowo, obliczenie długości krzywej nie zmienia jej. **Operacje dynamiczne** modyfikują obiekty, których dotyczą. Trzy podstawowe operacje dynamiczne to *utwórz*, *usuń* i *zaktualizuj*. Dobrym przykładem operacji dynamicznej jest aktualizacja obiektu przestrzennego, która może obejmować translację (zmianę lokalizacji), rotację (zmianę orientacji), powiększenie, pomniejszenie, odbicie lustrzane i odkształcenie (deformację).

**Zapytania przestrzenne.** Zapytania przestrzenne to żądania danych przestrzennych wymagające użycia operacji przestrzennych. Wymienione niżej kategorie ilustrują trzy typowe rodzaje zapytań przestrzennych:

- **Zapytania zakresowe.** Znajdź wszystkie obiekty określonego rodzaju na danym obszarze. Przykładowo, znajdź wszystkie szpitale we Wrocławiu. Odmianą zapytań tego typu jest wyszukiwanie wszystkich obiektów w określonej odległości od danej lokalizacji — np. znajdowanie wszystkich karetek w odległości do 8 kilometrów od miejsca wypadku.
- **Zapytania o najbliższego sąsiada.** Znajdź obiekt określonego typu najbliższy podanej lokalizacji — np. znajdź samochód policyjny znajdujący się najbliżej miejsca przestępstwa. Te zapytania można uogólnić do wyszukiwania  $n$  najbliższych sąsiadów — np. 5 karetek zlokalizowanych najbliżej miejsca wypadku.
- **Złączenia przestrzenne (nakładanie).** Zwykle polegają na złączaniu obiektów dwóch rodzajów na podstawie jakiegoś warunku przestrzennego, np. obiektów przecinających się lub pokrywających w przestrzeni, lub znajdujących się w określonej odległości od siebie. Oto przykłady: znajdź wszystkie miasta zlokalizowane przy drodze łączącej dwa miasta lub znajdź wszystkie domy położone do 3 kilometrów od jeziora. Pierwszy przykład to złączenie przestrzenne obiektu *miasto* z obiektem *autostrada*. W drugim przykładzie przestrzennie złączane są obiekty *jezioro* i *dom*.



## 26.3.4. Indeksowanie danych przestrzennych

Indeks przestrzenny służy do uporządkowania obiektów w grupy (odpowiadające stronom w pamięci drugorzędnej), tak aby możliwe było łatwe zlokalizowanie obiektów z określonego obszaru. Dla każdej grupy wyznaczony jest obszar, czyli fragment przestrzeni obejmujący wszystkie obiekty z danej grupy. Obszarami są zwykle prostokąty. Gdy danymi są punkty, obszary są rozłączne i dzielą przestrzeń w taki sposób, że każdy punkt należy do dokładnie jednej grupy. Używane są tu dwa podstawowe podejścia:

- (1) Wyspecjalizowane struktury indeksujące, umożliwiające wydajne wyszukiwanie obiektów na podstawie operacji wyszukiwania przestrzennego, zapisane w systemie baz danych. Takie struktury odgrywają podobną rolę jak indeksy w postaci  $B^+$ -drzew w tradycyjnych systemach baz danych. Przykładowe struktury tego rodzaju to *pliki matrycowe* i *R-drzewa*. Do przyspieszania złączania przestrzennego można wykorzystać indeksy przestrzenne specjalnego rodzaju — *indeksy złączy przestrzennych*.
- (2) Zamiast tworzyć nowe struktury indeksujące, można przekształcić przestrzenne dane dwuwymiarowe (2D) w dane jednowymiarowe (1D), co pozwala zastosować tradycyjne techniki indeksowania ( $B^+$ -drzewa). Algorytmy przekształcania danych 2D na dane 1D są nazywane *krzywymi wypełniającymi płaszczyznę* (ang. *space filling curves*). Nie będziemy ich tu omawiać (proponowane źródła znajdziesz w wybranych publikacjach).

Teraz przedstawimy przegląd wybranych technik indeksowania przestrzennego.

**Pliki matrycowe.** Pliki matrycowe w kontekście indeksowania danych na podstawie wielu atrybutów przedstawiliśmy w rozdziale 17. Takie pliki można też wykorzystać do indeksowania danych dwuwymiarowych i  $n$ -wymiarowych danych przestrzennych. Metoda **stałej matrycy** polega na podziale  $n$ -wymiarowej hiperprzestrzeni na grupy o równej wielkości. Do implementacji stałej matrycy służą tablice  $n$ -wymiarowe. Obiekty znajdujące się w danej komórce (w całości lub częściowo) można zapisać w dynamicznej strukturze, aby uwzględnić przepełnienie. To podejście jest przydatne dla danych o rozkładzie jednorodnym, np. dla obrazów satelitarnych. Jednak struktura stałej matrycy jest sztywna, a matryca może się okazać rzadka i duża.

**R-drzewa.** **R-drzewo** to drzewo zrównoważone ze względu na wysokość. Jest ono odmianą  $B^+$ -drzewa działającą dla  $k$  wymiarów, gdzie  $k > 1$ . Dla dwóch wymiarów (2D) lokalizacja obiektów przestrzennych jest w przybliżeniu określana w R-drzewie za pomocą **minimalnego zawierającego prostokąta** (ang. *minimum bounding rectangle* — MBR), czyli najmniejszego prostokąta z bokami równoległymi do osi systemu współrzędnych ( $x$  i  $y$ ) zawierającego dany obiekt. Poniżej opisano cechy R-drzew. Cechy te są podobne jak w  $B^+$ -drzewach (patrz podrozdział 18.3), ale dostosowane do dwuwymiarowych obiektów przestrzennych. Podobnie jak w podrozdziale 18.3 używamy litery  $M$  do oznaczenia maksymalnej liczby elementów mieszczących się w węźle R-drzewa.

- (1) Struktura każdego wpisu (rekordu) w liściu indeksu to  $(I, \text{identyfikator-obiektu})$ , gdzie  $I$  to MBR obiektu przestrzennego o identyfikatorze *identyfikator-obiektu*.
- (2) Każdy węzeł oprócz korzenia musi być zapełniony przynajmniej w połowie. Dlatego liść niebędący korzeniem powinien zawierać  $m$  wpisów  $(I, \text{identyfikator-obiektu})$ , gdzie  $M/2 \leq m \leq M$ . Podobnie węzeł niebędący liściem lub korzeniem powinien



obejmować  $m$  wpisów ( $I$ , *wskaźnik-do-dziecka*), gdzie  $M/2 \leq m \leq M$ , a  $I$  to MBR obejmujący sumę wszystkich prostokątów z węzła wskazywanego przez *wskaźnik-do-dziecka*.

- (3) Wszystkie liście znajdują się na tym samym poziomie, a korzeń powinien obejmować przynajmniej dwa wskaźniki (chyba że jest liściem).
- (4) Wszystkie obszary MBR mają boki równoległe do osi globalnego systemu współrzędnych.

Inne struktury składowania danych przestrzennych to m.in. drzewa czwórkowe i ich odmiany. **Drzewa czwórkowe** dzielą przestrzeń lub podprzestrzeń na obszary o równej wielkości, a następnie dzielą te obszary itd., aby określić pozycje różnych obiektów. Ostatnio zaproponowano wiele nowych struktur dostępu do danych przestrzennych. Jest to obszar, w którym prowadzone są aktywne badania.

**Indeksy złączeń przestrzennych.** Indeks złączeń przestrzennych wymaga wstępnego przetworzenia operacji złączenia przestrzennego i zapisania wskaźników do powiązanego obiektu w strukturze indeksu. Indeksy złączeń zwiększają wydajność powtarzających się zapytań obejmujących złączenia rzadko aktualizowanych tabel. Warunki złączeń przestrzennych służą do wykonywania zapytań takich jak: „Utwórz listę przecinających się autostrad i rzek”. To złączenie przestrzenne służy do identyfikowania i pobierania par obiektów spełniających związek przestrzenny *przecinają się*. Ponieważ wyznaczanie wyników związków przestrzennych jest zwykle czasochłonne, wynik można obliczyć raz i zapisać w tabeli z parami identyfikatorów obiektów (lub identyfikatorów krotek) spełniających dany warunek. W ten sposób powstaje indeks złączenia.

Indeks złączenia można opisać za pomocą grafu dwudzielnego  $G = (V_1, V_2, E)$ , gdzie  $V_1$  obejmuje identyfikatory krotek z relacji  $R$ , a  $V_2$  zawiera identyfikatory krotek z relacji  $S$ . Zbiór krawędzi zawiera krawędź  $(v_r, v_s)$  dla  $v_r$  z relacji  $R$  i  $v_s$  z relacji  $S$ , jeśli w indeksie złączania znajduje się krotka odpowiadająca krawędzi  $(v_r, v_s)$ . Graf dwudzielny tworzy model wszystkich powiązanych krotek reprezentowanych jako połączone wierzchołki grafu. Indeksy złączeń przestrzennych są używane w operacjach (patrz punkt 26.3.3) obejmujących wyznaczanie związków między obiektami przestrzennymi.

## 26.3.5. Eksploracja danych przestrzennych

Dane przestrzenne są zwykle wysoce skorelowane. Przykładowo, ludzie o podobnych cechach, zawodach i doświadczeniu często mieszkają w tej samej okolicy. Trzy główne techniki eksploracji danych przestrzennych to klasyfikacja przestrzenna, asocjacja przestrzenna i klastrowanie przestrzenne.

- **Klasyfikacja przestrzenna.** Celem klasyfikacji jest oszacowanie wartości atrybutu relacji na podstawie wartości innych atrybutów relacji. Przykładem problemu z zakresu klasyfikacji przestrzennej jest określanie lokalizacji gniazd na mokradłach na podstawie wartości innych atrybutów (np. trwałości roślinności i głębokości wody). Takie zadania są też nazywane *problemem prognozowania lokalizacji*. Problemem tego rodzaju jest również określanie miejsc nasilonej aktywności przestępczej.

- **Asocjacja przestrzenna. Reguły asocjacji przestrzennej** są definiowane w kategoriach predykatów przestrzennych, a nie obiektów. Taka reguła ma postać:

$$P_1 \wedge P_2 \wedge \dots \wedge P_n \Rightarrow Q_1 \wedge Q_2 \wedge \dots \wedge Q_m$$

gdzie przynajmniej jeden z elementów  $P_i$  lub  $Q_j$  to predykat przestrzenny. Poniższa reguła:

$\text{jest}(x, \text{państwo}) \wedge \text{styczne\_z}(x, \text{Morze Śródziemne}) \Rightarrow \text{jest}(x, \text{eksporter\_wina})$   
(reguła ta mówi, że państwo leżące nad Morzem Śródziemnym jest zwykle eksporterem wina) to przykładowa reguła asocjacyjna o określonym poziomie wsparcia  $s$  (ang. *support*) i ufności  $c$  (ang. *confidence*)<sup>27</sup>.

- **Reguły kolokacji przestrzennej** to próba uogólnienia reguł asocjacji w taki sposób, aby uwzględnić w nich zbiory danych z indeksami przestrzennymi. Asocjacje przestrzenne i nieprzestrzenne różnią się pod kilkoma ważnymi względami. Oto niektóre z nich:
  - (1) W kontekście przestrzennym nie występują transakcje, ponieważ dane znajdują się w przestrzeni ciągłej. Podział przestrzeni na transakcje prowadziłby do przeszacowania lub niedoszacowania ważnych miar, np. wsparcia i ufności.
  - (2) Wielkość zbioru elementów w przestrzennych bazach danych jest mała. Oznacza to, że zbiory danych przestrzennych obejmują dużo mniej elementów niż zbiory danych nieprzestrzennych.

W większości sytuacji elementy przestrzenne to nieciągłe odpowiedniki zmiennych ciągłych. Przykładowo, w Polsce obszary o określonych dochodach mogą zostać zdefiniowane jako miejsca o średnich miesięcznych zarobkach z określonych przedziałów, np.: poniżej 3000 zł, od 3000 do 5000 zł i powyżej 5000 zł.

**Klastrowanie przestrzenne** ma służyć do grupowania obiektów baz danych w taki sposób, aby najbardziej podobne do siebie obiekty znajdowały się w tym samym klastrze, a obiekty z różnych klastrow były jak najmniej do siebie podobne. Jednym z zastosowań klastrowania przestrzennego jest grupowanie wstrząsów sejsmicznych w celu wykrywania uskoków tektonicznych. Przykładowy algorytm klastrowania przestrzennego to **klastrowanie na podstawie gęstości**, próbujące wyznaczyć klastry na podstawie gęstości punktów danych w określonym obszarze. W takich algorytmach klastry to obszary z gęstym rozmieszczeniem obiektów w przestrzeni danych. Dwie wersje takich algorytmów to klastrowanie przestrzenne na podstawie gęstości w zastosowaniach, gdzie występuje szum (DBSCAN)<sup>28</sup>, oraz klastrowanie na podstawie gęstości (DENCLUE)<sup>29</sup>. DBSCAN jest algorytmem klastrowania opartym na gęstości, ponieważ wyszukuje klastry według szacowanego rozkładu gęstości dla uwzględnianych węzłów.

<sup>27</sup> Kwestie wsparcia i ufności dla reguł asocjacji są opisane w ramach eksploracji danych w podrozdziale 28.2.

<sup>28</sup> Algorytm DBSCAN został zaproponowany w pracy Estera, Kriegela, Sandera i Xu (1996).

<sup>29</sup> Algorytm DENCLUE zaproponowano w pracy Hinnenberga i Gabriela (2007).

### 26.3.6. Zastosowania danych przestrzennych

Zarządzanie danymi przestrzennymi jest przydatne w wielu dziedzinach, w tym w geografii, korzystaniu ze zdalnych czujników, planowaniu przestrzennym i zarządzaniu zasobami naturalnymi. Zarządzanie bazami przestrzennymi odgrywa istotną rolę w rozwiązywaniu trudnych problemów naukowych dotyczących np. globalnych zmian klimatu i genomiki. Z powodu przestrzennego charakteru danych o genomie systemy GIS i przestrzenne bazy danych mają duże znaczenie w dziedzinie bioinformatyki. Niektóre z typowych zastosowań to wykrywanie wzorców (np. w celu sprawdzenia, czy topologia konkretnego genu z genomu występuje w jakiejś innej mapie cech w bazie danych), budowanie przeglądarek genomu i tworzenie map z wizualizacjami. Innym ważnym obszarem zastosowań eksploracji danych przestrzennych jest wykrywanie przestrzennych wartości odstających. **Przestrzenna wartość odstająca** to obiekt w przestrzeni, którego wartości atrybutów nieprzestrzennych są wyraźnie inne niż w obiektach występujących w pobliżu danego. Przykładowo, jeśli w sąsiedztwie starszych domów pojawi się tylko jeden nowy, będzie on wartością odstającą według atrybutu nieprzestrzennego „wiek\_domu”. Wykrywanie przestrzennych wartości odstających jest przydatne w wielu zastosowaniach systemów informacji geograficznej i przestrzennych baz danych. Obszary zastosowań to np.: transport, ekologia, bezpieczeństwo publiczne, służba zdrowia, nauki o klimacie i usługi oparte na lokalizacji.

## 26.4. Zagadnienia z obszaru multimedialnych baz danych

**Multimedialne bazy danych** zapewniają funkcje pozwalające użytkownikom na przechowywanie i odczytywanie różnych typów informacji multimedialnych, takich jak *obrazy* (zdjęcia i rysunki), *nagrania wideo* (filmy, relacje telewizyjne, nagrania domowe), *dźwięki* (piosenki, telefoniczne wiadomości głosowe, przemówienia) czy *dokumenty* (książki i artykuły). Główne rodzaje zapytań w tego typu bazach danych dotyczą znajdowania źródła zawierającego poszukiwany obiekt. Przykładowo, użytkownik może szukać wszystkich nagrań wideo z określoną osobą, np. z Michaeliem Jacksonem. Można również szukać nagrań zawierających określone działanie, na przykład nagrań pokazujących zdobywanie bramki przez danego zawodnika lub określoną drużynę.

Powyższe typy zapytań są nazywane **wyszukiwaniem według treści**, ponieważ poszukiwane są obiekty zawierające dane dotyczące określonej osoby lub czynności. Multimedialna baza danych musi zatem stosować pewien model organizowania i indeksowania źródeł informacji multimedialnej na podstawie jej treści. *Identyfikowanie zawartości* źródeł multimedialnych jest trudnym i czasochłonnym zadaniem. Istnieją dwa główne rozwiązania tego problemu. Pierwsze jest oparte o **automatyczną analizę danych**, która znajduje w nich pewne określone matematycznie cechy. Ta technika stosuje różne metody analizy w zależności od rodzaju danych (obraz, tekst, wideo lub dźwięk). Drugie rozwiązanie zależy od **ręcznej identyfikacji** obiektów i czynności w każdym ze źródeł i od zastosowania uzyskanej w ten sposób informacji w indeksowaniu zasobów. Takie podejście można zastosować do wszystkich rodzajów informacji multimedialnej, ale wymaga ono fazy ręcznego przetwarzania danych, w której człowiek przegląda dane, identyfikuje i kataloguje obiekty oraz działania, które mogą być później użyte do tworzenia indeksu.

W dalszej części tego punktu przedstawimy charakterystykę każdego rodzaju danych multimedialnych — obrazów, wideo, dźwięku i dokumentów tekstowych. Dalej omówimy techniki automatycznej analizy obrazów oraz problem rozpoznawania obiektów na obrazach. Podrozdział zakończymy uwagami na temat analizowania materiałów dźwiękowych.

**Obraz** jest zazwyczaj zapisywany w formie „surowej” — jako zestaw wartości pikseli lub komórek — albo w postaci skompresowanej w celu zaoszczędzenia miejsca. *Wskaźnik kształtu* obrazu opisuje jego oryginalny kształt, który zazwyczaj jest prostokątem **komórek** określonej wysokości i szerokości. Każdy obraz może być zatem przedstawiony jako siatka  $n$  na  $m$  komórek. Każda komórka ma przypisaną wartość piksela opisującą jej zawartość. W obrazach czarno-białych piksel może być jednobitowy. W szarych lub kolorowych obrazach piksel zawiera wiele bitów. Ponieważ obrazy często zajmują duże obszary pamięci, są zazwyczaj przechowywane w formie skompresowanej. Standardy kompresji obrazu takie jak GIF, JPEG czy MPEG używają różnych transformacji matematycznych w celu zmniejszenia liczby przechowywanych komórek przy zachowaniu cech oryginalnego obrazu. Wspomniane transformacje matematyczne wykorzystują między innymi dyskretną transformatę Fouriera (DFT), dyskretną transformatę kosinusową (DCT) i transformaty falkowe.

Do identyfikowania poszukiwanych obiektów w obrazie jest on dzielony na jednorodne fragmenty przy użyciu *predykatu homogeniczności*. W przypadku obrazu kolorowego w jeden fragment będą pogrupowane przylegające do siebie komórki o podobnej zawartości. Predykat homogeniczności precyzuje warunki automatycznego grupowania takich komórek. Segmentacja i kompresja mogą zatem określać podstawowe cechy obrazu.

Typowe zapytanie w bazie danych tego typu polega na odnalezieniu wszystkich obrazów podobnych do danego obrazu wzorcowego. Tym obrazem może być wybrany fragment zawierający poszukiwany wzór, a zadaniem systemu bazy danych jest odnalezienie innych obrazów zawierających ten sam wzór. Istnieją dwie podstawowe techniki dla tego typu wyszukiwań. Pierwsza z nich stosuje do porównywania przechowywanych obrazów z zadaniem wzorem **funkcję odległości**. Jeżeli wartość odległości jest mała, to prawdopodobieństwo dopasowania jest duże. Do ograniczenia przestrzeni poszukiwań tworzy się indeksy grupujące bliskie obrazy (w sensie metryki dopasowującej). Druga technika, nazywana **podejściem transformacyjnym**, mierzy podobieństwo dwóch obrazów, określając liczbę przekształceń potrzebnych do transformacji jednego obrazu w drugi. Transformacje to między innymi obroty, przesunięcia i skalowanie. Technika transformacyjna jest bardziej ogólna, ale również trudniejsza w implementacji i bardziej czasochłonna.

Dane **wideo** są zazwyczaj prezentowane jako seria ramek, gdzie każda ramka jest nieruchomym obrazem. Zamiast identyfikowania obiektów zawartych w każdej z ramek osobno nagranie wideo dzielone jest na **segmenty** sąsiednich ramek przedstawiających te same obiekty lub czynności. Segmenty są identyfikowane poprzez ich pierwsze i ostatnie ramki. Indeksy są budowane w oparciu o zawarte w segmentach obiekty i wykorzystują specjalne struktury nazywane *drzewami segmentów ramek*. Tego typu indeksy zawierają zarówno informacje o obiektach, takich jak ludzie, domy, samochody, jak i czynnościach, takich jak *wygłaszanie* przemówienia lub *rozmowa* dwóch osób. Podobnie jak obrazy, nagrania wideo również są często poddawane kompresji (np. MPEG).

**Dane audio** to nagrania dźwiękowe, takie jak przemówienia, prezentacje czy zapisy wiadomości lub rozmów telefonicznych podsłuchanych przez organy ścigania. W tym przypadku do identyfikacji podstawowych cech głosu określonej osoby (w celu utworzenia indeksu opartego na podobieństwie i wyszukiwania danych) można użyć przekształceń dyskretnych. Krótkie omówienie analizy dźwięku zawiera punkt 26.4.4.

Multimedialne **dane tekstowe (dokumenty)** są zazwyczaj pełnymi tekstami artykułów, książek lub gazet. Tego typu dane są indeksowane poprzez indeksowanie występujących w nich słów kluczowych i częstotliwości ich pojawiania się. W tym procesie eliminowane są „wypełniacze” i często występujące słowa (**słowa pomijane**; ang. *stopwords*). Ponieważ w trakcie indeksowania zbioru dokumentów liczba słów kluczowych może być duża, opracowano specjalne techniki redukujące zbiór słów do najbardziej istotnych w danej kolekcji dokumentów. Jedną z nich jest oparta na transformacjach macierzy technika *rozkładu według wartości szczególnych* (SVD — ang. *Singular Value Decomposition*). Do grupowania podobnych dokumentów używa się struktur *drzew wektorów teleskopowych* (ang. *telescoping vector trees*, *TV-trees*). Przetwarzanie dokumentów jest szczegółowo opisane w rozdziale 27.

### 26.4.1. Automatyczna analiza obrazów

Analizy materiałów multimedialnych są niezbędne, aby zapewnić obsługę różnych zapytań i interfejsów wyszukiwania danych. Dane multimedialne (takie jak obrazy) trzeba reprezentować za pomocą cech pozwalających definiować podobieństwo. W pracach prowadzonych do tej pory w tym obszarze używano niskopoziomowych cech wizualnych, takich jak kolor, tekstura i kształt, które są bezpośrednio powiązane z percepcyjnymi aspektami zawartości obrazu. Te cechy są łatwe do wyodrębnienia i reprezentowania. Ponadto wygodnie jest projektować miary podobieństwa na podstawie statystycznych właściwości tych cech.

**Kolor** jest jedną z najczęściej używanych cech wizualnych w wyszukiwaniu obrazu na podstawie zawartości, ponieważ nie zależy od wielkości obrazu ani jego orientacji. Wyszukiwanie według podobieństwa kolorów odbywa się głównie na podstawie analizy histogramu koloru w każdym obrazie. Pozwala to określić procent pikseli w trzech kanałach kolorów (czerwonym, zielonym i niebieskim; ang. *red, green, blue* — **RGB**). Jednak na reprezentację w formacie RGB wpływają orientacja obiektu względem oświetlenia i ukierunkowanie aparatu. Dlatego w nowych technikach wyszukiwania obrazów używane są konkurencyjne niezmiennicze reprezentacje, takie jak **HSV** (ang. *hue, saturation, value*). Format HSV opisuje kolory jako punkty w cylindrze, w którym środkowa oś zmienia się od czarnego koloru na dole do białego u góry; między nimi występują neutralne kolory. Kąt względem osi odpowiada odcieniowi, odległość od osi określa saturację, a odległość na osi wyznacza jasność.

**Tekstura** dotyczy powtarzających się na obrazie wzorców, które nie wynikają z występowania jednego koloru lub barw o tej samej jaskrawości. Przykładami rodzajów tekstury są chropowaty i jedwabisty. Przykładowe tekstury, jakie można identyfikować, to: skóra cieleca, mata słomiana, płótno bawełniane itd. Podobnie jak obrazy są reprezentowane za pomocą tablic pikseli (elementów obrazu), tak tekstury są reprezentowane za pomocą **tablic tekseli** (elementów tekstury). Tekstury są przypisywane do zbiorów, które zależą od tego, ile tekstur zidentyfikowano na obrazie. Te zbiory obejmują nie tylko definicję

tekstury, ale też określają, gdzie na obrazie znajduje się dana tekstura. Identyfikowanie tekstur odbywa się w wyniku modelowania ich za pomocą dwuwymiarowych map odcieni szarości. Względna jasność pary pikseli jest obliczana w celu oszacowania poziomu kontrastu, regularności, szorstkości i kierunku.

**Kształt** dotyczy obszaru na obrazie. Zwykle określa się go za pomocą segmentacji lub wykrywania krawędzi. **Segmentacja** to oparta na obszarach technika, w której uwzględniany jest cały obszar (zbiór pikseli). **Wykrywanie krawędzi** to podejście oparte na granicach, w którym uwzględniane są tylko cechy zewnętrznych ograniczeń obiektów. Na potrzeby translacji, rotowania i skalowania reprezentacja kształtu zwykle musi być niezmienna. Znanymi metodami reprezentowania kształtów są deskryptory Fouriera i niezmienniki momentowe.

## 26.4.2. Wykrywanie obiektów na obrazach

**Wykrywanie obiektów** polega na identyfikowaniu obiektów ze świata rzeczywistego na obrazie lub w nagraniu video. System musi radzić sobie z identyfikowaniem obiektów nawet w sytuacji, gdy obrazy obiektu różnią się pod względem punktu widzenia, wielkości, skali, a nawet gdy stosowana jest rotacja lub translacja. Opracowano techniki polegające na podziale pierwotnego obrazu na obszary na podstawie podobieństwa przyległych pikseli. Tak więc na obrazie przedstawiającym tygrysa w dżungli można wykryć fragment z tygrysem na tle dżungli, a następnie, po porównaniu tego fragmentu ze zbiorem obrazów treninowych, oznaczyć obraz jako reprezentujący tygrysa.

Niezwykle istotna jest reprezentacja obiektów multimedialnych w modelu obiektowym. Jedną z technik polega na podziale obrazu na jednorodne segmenty na podstawie predykatu homogenicznego (ang. *homogeneous predicate*). Przykładowo, w kolorowym obrazie przyległe komórki o podobnych wartościach pikseli są grupowane w segmenty. Predykat homogeniczności definiuje warunki automatycznego grupowania komórek. Można więc zidentyfikować podstawowe cechy obrazu za pomocą segmentacji i kompresji. Inne podejście polega na wyszukiwaniu pomiarów obiektu niezmiennych w kontekście transformacji. Nie da się przechowywać bazy danych z przykładami wszystkich możliwych transformacji obrazu. Aby poradzić sobie z tym problemem, w technikach wykrywania obiektów na obrazie wyszukiwane są istotne punkty (lub cechy) niezmiennie w kontekście transformacji.

Ważny wkład w tę dziedzinę wniósł Lowe<sup>30</sup>, który do niezawodnego wykrywania obiektów zastosował cechy niezmiennie w kontekście skalowania. To podejście to **SIFT** (ang. *scale-invariant feature transform*). Cechy SIFT są niezmiennie w kontekście skalowania i rotacji obrazu oraz częściowo niezmiennie w kontekście zmian oświetlenia i punktu widzenia w przestrzeni trójwymiarowej. Są też dobrze zlokalizowane w przestrzeni i występują odpowiednio często, co zmniejsza prawdopodobieństwo błędów z powodu przesłonięcia, zakłóceń lub szumu. Ponadto są to cechy wysoce dystynktywne, dzięki czemu jedną cechę można z wysokim prawdopodobieństwem poprawnie dopasować do dużej bazy cech, co stanowi podstawę do wykrywania obiektów i scen.

---

<sup>30</sup> Patrz Lowe (2004), „Distinctive Image Features from Scale-Invariant Keypoints”.



W trakcie dopasowywania i wykrywania obrazów cechy SIFT (inna nazwa: *cechy z punktami charakterystycznymi*; ang. *keypoint features*) są wyodrębniane ze zbioru obrazów wzorcowych i zapisywane w bazie. Następnie ma miejsce rozpoznawanie obiektów w wyniku porównywania każdej cechy z nowego obrazu z cechami zapisanymi w bazie i wyszukiwania kandydujących pasujących cech na podstawie odległości euklidesowej wektorów cech. Ponieważ cechy z punktami charakterystycznymi są wysoce dystynktywne, w dużej bazie danych cech można z wysokim prawdopodobieństwem poprawnie dopasować pojedynczą cechę.

Obok techniki SIFT istnieje też wiele innych metod rozpoznawania obiektów w skomplikowanym otoczeniu lub przy częściowym zasłonięciu. Przykładowo, **RIFT**, uogólniona wersja metody SIFT działająca także dla zrotowanych obiektów, identyfikuje grupy lokalnych obszarów afinicznych (cech obrazu mających charakterystyczny wygląd i eliptyczny kształt). Zależności między takimi obszarami pozostają mniej więcej stałe dla obiektu widzianego z różnych perspektyw i dla wielu egzemplarzy obiektów z danej klasy.

### 26.4.3. Semantyczne opisywanie obrazów

Pośrednie opisywanie obrazów to ważne zagadnienie w zakresie rozpoznawania i porównywania obrazów. Do obrazu lub jego fragmentu można przypisać wiele znaczników. We wcześniej wspomnianym przykładzie z obrazem można powiązać znaczniki takie jak „tygrys”, „dżungla”, „zielony” i „paski”. Większość technik wyszukiwania obrazów pobiera obrazy na podstawie znaczników podanych przez użytkowników, które nie zawsze są precyzyjne lub kompletne. Aby poprawić jakość wyszukiwania, w wielu nowych systemach dąży się do zautomatyzowanego generowania znaczników. W danych multimedialnych większość semantyki jest zawarta w ich treści. Systemy wykorzystują przetwarzanie obrazu i techniki modelowania statystycznego do analizowania treści obrazu w celu generowania precyzyjnych znaczników opisowych, które można następnie wykorzystać do wyszukiwania obrazów na podstawie treści. Ponieważ w różnych schematach do opisu obrazów używane mogą być inne słowniki, poziom wyszukiwania obrazów będzie niski. Aby rozwiązać ten problem, w nowych technikach wyszukiwania zaproponowano zastosowanie hierarchii, taksonomii lub ontologii z użyciem języka **OWL** (ang. *Web Ontology Language*). W tym języku pojęcia i relacje między nimi są ściśle zdefiniowane. Można je wykorzystać do wnioskowania na temat elementów wyższego poziomu na podstawie znaczników. W takiej taksonomii pojęcia typu „niebo” lub „trawa” można podzielić na „czyste niebo” i „pochmurne niebo” lub „wysuszone trawa” i „zielona trawa”. Te techniki zwykle zalicza się do dziedziny opisywania semantycznego i można z nich korzystać w połączeniu z opisanymi wcześniej strategiami analizy cech i identyfikowania obiektów.

### 26.4.4. Analizy danych audio

Dane audio można na ogólnym poziomie podzielić na mowę, muzykę i inne. Każda z tych kategorii znacząco różni się od pozostałych. Dlatego różne rodzaje danych audio są traktowane w inny sposób. Dane audio trzeba przekształcić na postać cyfrową, zanim będzie można je przetwarzać i składować. Indeksowanie i wyszukiwanie danych audio są zwykle trudniejsze niż innych danych multimedialnych, ponieważ dźwięk, podobnie jak wideo, jest ciągły i (w odróżnieniu od tekstu) nie ma łatwo mierzalnych cech. Czystość nagrań dźwiękowych



jest łatwa do oceny dla człowieka, ale trudna do ilościowego ujęcia w uczeniu maszynowym. Co ciekawe, do danych z mową często stosowane są techniki rozpoznawania mowy (w celu wzbogacenia samych treści audio), ponieważ może to znacznie ułatwić indeksowanie danych i zwiększyć precyzję tego zadania. To podejście jest czasem nazywane *tekstowym indeksowaniem danych audio*. Metadane związane z mową są zwykle zależne od nagrania, ponieważ są generowane na jego podstawie. Uwzględniane są długość wypowiedzi, liczba mówiących osób itd. Niektóre dane mogą być niezależne od treści. Dotyczy to np. długości całego nagrania i formatu składowania danych. Z kolei indeksowanie muzyki odbywa się na podstawie statystycznych analiz sygnału audio (*indeksowanie oparte na treści*). W tym podejściu często uwzględniane są najważniejsze cechy dźwięku: natężenie, wysokość, barwa i rytm. Można porównać różne fragmenty danych audio i pobrać informacje o nich na podstawie obliczeń różnych cech, a także w wyniku zastosowania określonych transformacji.

## 26.5. Wprowadzenie do dedukcyjnych baz danych

### 26.5.1. Przegląd dedukcyjnych baz danych

W dedukcyjnym systemie baz danych reguły są zazwyczaj definiowane przy użyciu **języka deklaratywnego**, czyli języka programowania, w którym określa się raczej rezultat działania, a nie sposób jego osiągnięcia. **Moduł wnioskujący (mechanizm dedukcyjny)** w systemie może wywodzić nowe fakty z bazy danych za pomocą zadanych reguł. Model danych używany w tego typu systemach jest ściśle powiązany z modelem relacyjnym, a w szczególności z rachunkiem relacyjnym domen (patrz podrozdział 6.6). Jest również związany z tematem **programowania logicznego** i językiem **Prolog**. Początki prac nad dedukcyjnymi bazami danych łączą się właśnie z tym językiem. Jego odmianą jest **Datalog**, który jest używany do deklaratywnego określania reguł w połączeniu z istniejącym zbiorem relacji, które są w nim traktowane jak literały. Co prawda struktury języka Datalog przypominają te znane z języka Prolog, jednak jego znaczenie operacyjne — czyli sposób wykonywania programu — jest nieco inne.

Dedukcyjna baza danych składa się z dwóch głównych rodzajów specyfikacji: faktów i reguł. **Fakty** są określane w sposób podobny do definiowania relacji, z tą różnicą, że nie jest wymagane nazywanie atrybutów. Przypomnijmy, że krotka w relacji opisuje pewien fakt, a jej znaczenie jest częściowo określane przez nazwy atrybutów. W dedukcyjnych bazach danych znaczenie atrybutu w krotce jest określane wyłącznie przez jego *pozycję* wewnątrz krotki. **Reguły** są w pewnym sensie podobne do perspektyw (widoków) relacyjnych. Określają one virtualne relacje, które nie są przechowywane w bazie danych, ale mogą być tworzone na podstawie faktów poprzez zastosowanie mechanizmów wnioskowania określonych przez reguły. Główną różnicą pomiędzy perspektywami a regułami jest możliwość stosowania reguł rekurencyjnie, co może prowadzić do stworzenia wirtualnych relacji niemożliwych do określenia za pomocą mechanizmu perspektyw.

Wykonywanie programu w Prologu opiera się na technice nazywanej *nawrotami*, która realizuje zaprogramowane cele w kolejności zstępującej. W dedukcyjnych bazach danych używających języka Datalog szczególną uwagę poświęcono wykorzystaniu dużych

ilości danych zgromadzonych w relacyjnej bazie danych. Zatem techniki wykonywania programu zostały przystosowane do wykonywania w przeciwnym kierunku. Ograniczeniem języka Prolog jest fakt, że kolejność specyfikowania faktów i reguł ma wpływ na wykonywanie programu; co więcej — podobny wpływ ma również kolejność deklarowania literałów (omówionych w punkcie 26.5.3) wewnątrz reguły. Techniki wykonywania programu w języku Datalog zostały opracowane z myślą o obejściu tych problemów.

## 26.5.2. Notacja języków Prolog i Datalog

Notacja używana w Prologu (i Datalogu) opiera się na deklarowaniu predykatów o unikatowych nazwach. **Predykat** ma określone znaczenie, które jego nazwa powinna odzwierciedlać, oraz stałą liczbę **argumentów**. Jeżeli argumenty są wartościami stałymi, to predykat po prostu stwierdza istnienie pewnego faktu. Jeśli natomiast argumenty są zmiennymi, to predykat jest zapytaniem lub częścią reguły. W niniejszym podrozdziale będziemy stosować konwencję z języka Prolog, zgodnie z którą wszystkie **stałe wartości** w predykacie są *wartościami liczbowymi lub ciągami znaków*, a ich identyfikatory (nazwy) zaczynają się od *małej litery*, podczas gdy **nazwy zmiennych** zawsze zaczynają się od *wielkiej litery*.

Rozważmy przykład z rysunku 26.11, który jest oparty na uproszczonej wersji relacyjnej bazy danych z rysunku 5.6. Są na nim przedstawione trzy nazwy predykatów *nadzór*, *przełożony* i *podwładny*. Predykat *nadzór* jest określany poprzez zestaw faktów, z których każdy ma dwa argumenty: nazwisko przełożonego i nazwisko jego *bezpośredniego* podwładnego. Fakty te odpowiadają danym zapisanym w bazie danych i można je traktować jako zestaw krotek z relacji *NADZÓR* z dwoma atrybutami. Odpowiadający im schemat to

NADZÓR (Przełożony, Podwładny)

### (a) Fakty

nadzór(joanna, jan).  
 nadzór(joanna, agnieszka).  
 nadzór(joanna, kuba).  
 nadzór(marta, alicja).  
 nadzór(marta, piotr).  
 nadzór(jerzy, joanna).  
 nadzór(jerzy, marta).  
 ...

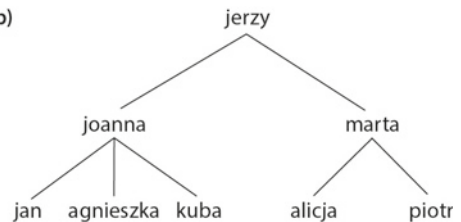
### Reguły

przełożony(X, Y) :- nadzór(X, Y).  
 przełożony(X, Y) :- nadzór(X, Z), przełożony(Z, Y).  
 podwładny(X, Y) :- przełożony(Y, X).

### Zapytania

przełożony(jerzy, Y)?  
 przełożony(jerzy, kuba)?

### (b)



RYSUNEK 26.11. (a) Notacja języka Prolog. (b) Drzewo zwierzchnictwa

Zatem  $\text{nadzór}(X, Y)$  określa fakt, że „X nadzoruje Y”. Należy zwrócić uwagę na brak nazw atrybutów w notacji języka Prolog. Nazwy atrybutów są reprezentowane jedynie poprzez ich pozycję w predykacie — pierwszy argument oznacza przełożonego, a drugi podwładnego.

Pozostałe dwa predykaty są definiowane przez reguły. Istotną innowacją w dedukcyjnych systemach baz danych jest możliwość wprowadzania reguł rekurencyjnych i tworzenia podstaw do wnioskowania nowych informacji w oparciu o dany zestaw reguł. Reguły są zapisywane w formie **nagłówek :- ciało**, gdzie symbol **:** - oznacza *wtedy i tylko wtedy*. Lewa strona reguły, nazywana też **nagłówkiem** lub **konkluzją**, składa się zazwyczaj z jednego **predykatu**, a prawa strona (inaczej **ciało** lub **przesłanka** reguły) — z *jednego lub więcej predykatów*. Predykat o stałych argumentach nazywany jest **pewnikiem** albo **predykatem skonkretyzowanym** (ang. *instantiated predicate*). Argumenty predykatów używanych w definicji reguły zazwyczaj stanowią listę symboli zmiennych, aczkolwiek mogą to być również wartości stałe. Reguła mówi, że jeśli **powiązanie** (przypisanie) stałych wartości do zmiennych w ciele powoduje, że *wszystkie* predykaty po prawej stronie reguły są **prawdziwe**, to dla takiego samego zestawu wartości prawdziwa jest również lewa strona reguły. Reguły pozwalają zatem na tworzenie nowych faktów będących instancjami ich nagłówków. Te nowe informacje są oparte na faktach, które istniały wcześniej w odniesieniu do instancji (lub powiązań) predykatów z ciała reguły. Zauważmy, że wymieniając w ciele reguły kilka predykatów, użytkownik łączy je logicznym operatorem **koniunkcji (I)**. Przecinki pomiędzy predykatami w prawej stronie reguły mogą być interpretowane jako „i”.

Rozważmy definicję predykatu przełożony z rysunku 26.11, którego pierwszym argumentem jest imię pracownika, a drugim pracownik, który jest jego *pośrednim* bądź *bezpośrednim* podwładnym. Przez wyrażenie *pośredni podwładny* należy tu rozumieć podwładnego o dowolnym stopniu zwierzchnictwa. Wyrażenie  $\text{przełożony}(X, Y)$  oznacza więc, że *X jest przełożonym Y* poprzez bezpośrednią lub pośrednią relację zwierzchnictwa. Możemy teraz napisać dwie reguły, które wspólnie stworzą znaczenie dla nowego predykatu. Pierwsza reguła mówi, że dla dowolnych wartości  $X$  i  $Y$  jeżeli  $\text{nadzór}(X, Y)$  — ciało reguły — jest prawdą, to  $\text{przełożony}(X, Y)$  — nagłówek reguły — również jest prawdą, ponieważ  $Y$  jest bezpośrednim podwładnym  $X$  (na następnym poziomie). Tej reguły można użyć do tworzenia wszystkich relacji bezpośredniego zwierzchnictwa i podlegania z faktów określających predykat nadzór. Druga (rekurencyjna) reguła mówi, że jeśli *obydwie* reguły  $\text{nadzór}(X, Z)$  i  $\text{przełożony}(Z, Y)$  są prawdziwe, to  $\text{przełożony}(X, Y)$  jest również prawdziwe. Jest to przykład **reguły rekurencyjnej**, gdzie jeden z predykatów prawej strony reguły jest taki sam jak predykat lewej strony reguły. Ogólnie, ciało reguły określa kilka przesłanek, takich że spełnienie wszystkich pozwala wnioskować, że konkluzja również jest spełniona. Zauważmy, że jeśli mamy dwie (lub więcej) reguły z tym samym nagłówkiem (lewą stroną reguły), to predykat nagłówka jest prawdziwy, jeżeli ciało *choć jednej* z reguł jest prawdziwe, czyli tego typu konstrukcja odpowiada **logicznej alternatywie (LUB)**. Na przykład, zdefiniowanie dwóch reguł  $X:-Y$  i  $X:-Z$  jest równoznaczne z wyrażeniem  $X:-Y \text{ or } Z$ . Ostatnia forma nie jest używana w systemach dedukcyjnych, ponieważ nie spełnia warunków standardowej formy reguły określanej mianem *klauzuli Horna*, która jest omówiona w punkcie 26.5.4.

System Prolog zawiera kilka **wbudowanych** predykatów interpretowanych przez system w sposób bezpośredni. Zawierają one operator równości  $=(X, Y)$ , który zwraca prawdę, jeżeli  $X$  i  $Y$  są identyczne, i może być zapisywany przy użyciu standardowej notacji wrostkowej jako  $X=Y$ .<sup>31</sup> Inne operatory porównania dla liczb, takie jak  $<$ ,  $<=$ ,  $>$  i  $>=$ ,

<sup>31</sup> System Prolog zawiera kilka różnych predykatów równości o różnych interpretacjach.

mogą być traktowane jako predykaty binarne. Funkcje arytmetyczne takie jak  $+$ ,  $-$ ,  $*$  i  $/$  mogą być w Prologu argumentami predykatów. Dla odmiany, Datalog (w swojej podstawowej formie) *nie* obsługuje funkcji arytmetycznych jako argumentów. Jest to jedna z głównych różnic pomiędzy Prologiem a Datalogiem. Najnowsze rozszerzenia proponują jednak włączyć tę funkcjonalność również do języka Datalog.

**Zapytanie** zazwyczaj wykorzystuje symbol predykatu z pewnymi zmiennymi jako argumentami, a jego znaczeniem (czyli *odpowiedzią*) jest wywnioskowanie wszystkich kombinacji stałych takich, że ich **przypisanie** (powiązanie) do zmiennych powoduje spełnienie predykatu. Przykładowo, pierwsze zapytanie z rysunku 26.11 odczytuje listę wszystkich podwładnych „jrzego”. Inny rodzaj zapytań, który przyjmuje jako argumenty jedynie symbole stałych, zwraca jako wynik prawdę lub fałsz w zależności od tego, czy dostarczone argumenty mogą być wywnioskowane ze znanych faktów i reguł. Na przykład, drugie zapytanie z rysunku 26.11 zwraca prawdę, ponieważ można wywnioskować predykat `przełożony(józef, joanna)`.

### 26.5.3. Notacja języka Datalog

W Datalogu, inaczej niż w pozostałych językach deklaratywnych (logicznych), program jest zbudowany z podstawowych obiektów nazywanych **formułami atomowymi**. Istnieje możliwość definiowania składni języków logicznych poprzez określanie składni formuł atomowych oraz sposobu ich łączenia w program. W Datalogu formułami atomowymi są **literały** w formie  $p(a_1, a_2, \dots, a_n)$ , gdzie  $p$  jest nazwą predykatu, a  $n$  jest liczbą argumentów predykatu  $p$ . Różne symbole predykatów mogą mieć różną liczbę argumentów, nazywaną **arnością** lub **stopniem** predykatu. Argumentami mogą być wartości stałe bądź nazwy zmiennych. Jak już wspomniano wcześniej, używamy tutaj konwencji rozpoczynania nazw wartości stałych od *małej* litery, a nazw zmiennych od *wielkiej* litery.

Datalog zawiera pewne **predykaty wbudowane**, które również mogą być użyte do tworzenia formuł atomowych. Predykaty wbudowane dzielą się na dwa główne rodzaje: binarne predykaty porównania w zbiorach uporządkowanych (`<(less)`, `<=(less_or_equal)`, `>(greater)` i `>=(greater_or_equal)` oraz predykaty porównania w zbiorach uporządkowanych lub nieuporządkowanych (`=(equal)` i `!=(not_equal)`). Mogą one być używane jako predykaty binarne dokładnie w ten sam sposób co inne predykaty (na przykład `less(X, 3)`) lub w standardowej notacji wrostkowej  $X < 3$ . Ponieważ dziedziny tych predykatów są potencjalnie nieskończone, należy ich używać w definicji reguł ze szczególną uwagą. Na przykład predykat `greater(X, 3)` użyty niezależnie generuje nieskończony zbiór wartości  $X$  spełniających określony warunek (wszystkie liczby większe od 3).

**Literałami** są formuły atomowe opisane wcześniej (nazywane **literałami pozytywnymi**) lub formuły atomowe poprzedzone słowem kluczowym `not`. Druga forma jest zaprzeczeniem oryginalnej formuły atomowej i jest nazywana **literałem negatywnym**. Programy w języku Datalog są *podzbiorem* formuł **rachunku predykatów**, które są podobne do formuł relacyjnego rachunku domen (patrz podrozdział 6.7). W Datalogu jednak formuły te muszą być konwertowane do **formy klauzulowej** przed wprowadzeniem do systemu. W języku Datalog dopuszczalne jest używanie tylko formuł w ograniczonej formie klauzulowej, nazywanej *klauzulami Horna*<sup>32</sup>.

<sup>32</sup> Nazwa pochodzi od nazwiska matematyka Alfreda Horna.

## 26.5.4. Forma klauzulowa i klauzule Horna

Przypomnijmy z podrozdziału 6.6, że formuła w rachunku relacyjnym jest warunkiem zawierającym predykaty nazywane *atomami* (oparte na nazwach relacji). Ponadto, formuła może zawierać *kwantyfikatory*: *uniwersalny* (dla każdego) i *egzystencjalny* (istnieje). W formie klauzulowej formuła musi być przetworzona w inną formułę o następujących cechach:

- Wszystkie zmienne w formule są objęte kwantyfikatorami uniwersalnymi. Nie jest konieczne jawne dodawanie takich kwantyfikatorów, więc są one usuwane. Wszystkie zmienne są jednak nadal *niejawnie* objęte ich działaniem.
- W formie klauzulowej formuła składa się z kilku klauzul, a każda **klauzula** z literałów połączonych operatorami logicznymi OR. Każda klauzula jest zatem *alternatywą* literałów.
- Same *klauzule* tworzące formułę są łączone tylko operatorem logicznym AND. **Formuła w formie klauzulowej** jest zatem *koniunkcją* klauzul.

Zostało udowodnione, że każda *formuła może być przedstawiona w formie klauzulowej*. Dla naszych zastosowań interesująca jest głównie forma poszczególnych klauzul będących alternatywą literałów. Przypomnijmy, że literały mogą być pozytywne i negatywne. Rozważmy następującą klauzulę:

$$\text{not}(P_1) \text{ OR } \text{not}(P_2) \text{ OR } \dots \text{ OR } \text{not}(P_n) \text{ OR } Q_1 \text{ OR } Q_2 \text{ OR } \dots \text{ OR } Q_m \quad (1)$$

Ta klauzula ma  $n$  negatywnych literałów i  $m$  pozytywnych. Można ją przekształcić w następującą formułę logiczną:

$$P_1 \text{ AND } P_2 \text{ AND } \dots \text{ AND } P_n \Rightarrow Q_1 \text{ OR } Q_2 \text{ OR } \dots \text{ OR } Q_m \quad (2)$$

gdzie  $\Rightarrow$  jest symbolem **wynikania**. Formuły (1) i (2) są równoważne, czyli ich wartość jest zawsze ta sama. Oto uzasadnienie: znaczenie symbolu wynikania mówi, że jeżeli wszystkie literały  $P_i$  ( $i=1, 2, \dots, n$ ) są prawdziwe, to cała formuła (2) jest prawdziwa tylko wtedy, gdy choć jeden z literałów  $Q_i$  jest prawdziwy; takie jest znaczenie symbolu wynikania  $\Rightarrow$ . Dla formuły (1) jeśli wszystkie literały  $P_i$  ( $i=1, 2, \dots, n$ ) są prawdziwe, to ich zaprzeczenie jest fałszywe; cała formuła (1) jest więc prawdziwa tylko wtedy, gdy choć jeden z  $Q_i$  jest prawdziwy. W języku Datalog reguły są wyrażane w ograniczonej formie klauzul nazywanej **klauzulami Horna**, w których każda klauzula może zawierać *najwyżej jeden* literał pozytywny. Klauzula Horna jest zatem przedstawiana w postaci:

$$\text{not}(P_1) \text{ OR } \text{not}(P_2) \text{ OR } \dots \text{ OR } \text{not}(P_n) \text{ OR } Q \quad (3)$$

lub w postaci

$$\text{not}(P_1) \text{ OR } \text{not}(P_2) \text{ OR } \dots \text{ OR } (P_n) \quad (4)$$

Klauzula Horna (3) może zostać przekształcona do postaci

$$P_1 \text{ AND } P_2 \text{ AND } \dots \text{ AND } P_n \Rightarrow Q \quad (5)$$

co w Datalogu będzie zapisane jako reguła

$$Q :- P_1, P_2, \dots, P_n \quad (6)$$

Klauzula Horna (4) może być przekształcona do postaci

$$P_1 \text{ AND } P_2 \text{ AND } \dots \text{ AND } P_n \Rightarrow (7)$$

co w języku Datalog można zapisać jako

$$P_1, P_2, \dots, P_n \text{ (8)}$$

**Reguła języka Datalog**, taka jak (6), jest zatem klauzulą Horna, a jej znaczenie oparte na formule (5) można przeczytać następująco: Jeżeli wszystkie predykaty  $P_1, P_2, \dots, P_n$  są prawdziwe dla pewnego przypisania zmiennych argumentów, to  $Q$  również jest prawdziwe i może być wywnioskowane. Wyrażenie (8) języka Datalog można traktować jako ograniczenie integralności, w którym wszystkie predykaty muszą być prawdziwe w celu spełnienia warunków zapytania.

Ogólnie **zapytanie w języku Datalog** składa się z dwóch części:

- programu w Datalogu, który jest skończonym zestawem reguł;
- literału  $P(X_1, X_2, \dots, X_n)$ , gdzie  $X_i$  są zmiennymi lub stałymi.

System oparty na językach Prolog lub Datalog jest wyposażony w wewnętrzny **moduł wnioskowania**, który może być użyty do przetwarzania i obliczania wyników tego typu zapytań. Moduły wnioskujące oparte na języku Prolog zazwyczaj zwracają po jednym wyniku zapytania (czyli po jednym zestawie wartości zmiennych określonych w zapytaniu) i następne zestawy muszą być w nich jawnie pobierane. Dla odmiany, systemy oparte na języku Datalog zwracają wszystkie wyniki naraz.

### 26.5.5. Interpretacja reguł

Istnieją dwie główne możliwości interpretacji teoretycznego znaczenia reguł: *dowodowo-teoretyczna* i *modelowo-teoretyczna*. W praktycznych zastosowaniach mechanizm wnioskowania w systemie definiuje dokładną interpretację, która może nie być w zbieżna z żadnym z mechanizmów teoretycznych. Mechanizm ten jest procedurą obliczeniową i jako taki operuje obliczeniową interpretacją znaczenia reguł. W tym punkcie omówimy dwie interpretacje teoretyczne, a następnie przedstawimy mechanizm wnioskowania jako sposób definiowania znaczenia reguł.

W teoretycznej interpretacji **dowodowo-teoretycznej** reguł przyjmuje się, że fakty i reguły są wyrażeniami prawdziwymi, czyli **aksjomatami**. **Aksjomat podstawowy** nie zawiera żadnych zmiennych. Fakty są aksjomatami podstawowymi przyjmowanymi za prawdziwe. Reguły są nazywane **aksjomatami dedukcyjnymi**, ponieważ służą do wnioskowania nowych faktów. Na przykład, rysunek 26.12 pokazuje, jak udowodnić fakt przełożony  $\hookrightarrow$ (józef, albert) za pomocą faktów i reguł z rysunku 26.11. Interpretacja dowodowo-teoretyczna pozwala na proceduralną lub obliczeniową metodę znajdowania odpowiedzi na zapytanie w języku Datalog. Proces dowodzenia, czy dany fakt (twierdzenie) jest prawdziwy, nazywamy **dowodzeniem twierdzeń**.

- |  |                                   |
|--|-----------------------------------|
| 1. przełożony(X, Y) :- nadzór(X, Y).                   | (reguła 1.)                       |
| 2. przełożony(X, Y) :- nadzór(X, Z), przełożony(Z, Y). | (reguła 2.)                       |
| 3. nadzór(janina, albert).                             | (dany aksjomat podstawowy)        |
| 4. nadzór(józef, janina).                              | (dany aksjomat podstawowy)        |
| 5. przełożony(janina, albert).                         | (zastosowanie reguły 1. do 3)     |
| 6. przełożony(józef, albert).                          | (zastosowanie reguły 2. do 4 i 5) |

RYSunEK 26.12. Udowadnianie nowego faktu



Drugi rodzaj interpretacji jest nazywany **modelowo-teoretycznym**. W jego przypadku, mając daną skończoną lub nieskończoną dziedzinę stałych wartości<sup>33</sup>, przypisujemy argumentom predykatu każdą ich możliwą kombinację. Należy następnie określić, czy predykat jest prawdziwy, czy fałszywy. Ogólnie, wystarczy określić kombinację argumentów, które pokazują, że predykat jest prawdziwy, i przyjąć, że dla wszystkich pozostałych zestawów argumentów jest on fałszywy. Jeśli taką czynność wykonuje się dla każdego predykatu z pewnego zbioru, to czynność tę nazywamy **interpretacją** zbioru predykatów. Na przykład, rozważmy interpretację pokazaną na rysunku 26.13 dla predykatów nadzór i przełożony. Interpretacja przypisuje do dwóch predykatów wartość prawdy (prawdę lub fałsz) w każdej możliwej kombinacji wartości argumentów (ze skończonej dziedziny).

#### Reguły

przełożony(X, Y) :- nadzór(X, Y).

przełożony(X, Y) :- nadzór(X, Z), przełożony(Z, Y).

#### Interpretacja

*Znane Fakty:*

nadzór(fryderyk, jan) jest **prawdą**.

nadzór(fryderyk, robert) jest **prawdą**.

nadzór(fryderyk, joanna) jest **prawdą**.

nadzór(janina, alicja) jest **prawdą**.

nadzór(janina, albert) jest **prawdą**.

nadzór(józef, fryderyk) jest **prawdą**.

nadzór(józef, janina) jest **prawdą**.

nadzór(X, Y) jest **fałszywy** dla wszystkich pozostałych kombinacji wartości X i Y.

*Fakty wywnioskowane:*

przełożony(fryderyk, jan) jest **prawdą**.

przełożony(fryderyk, robert) jest **prawdą**.

przełożony(fryderyk, joanna) jest **prawdą**.

przełożony(janina, alicja) jest **prawdą**.

przełożony(janina, albert) jest **prawdą**.

przełożony(józef, fryderyk) jest **prawdą**.

przełożony(józef, janina) jest **prawdą**.

przełożony(józef, jan) jest **prawdą**.

przełożony(józef, robert) jest **prawdą**.

przełożony(józef, joanna) jest **prawdą**.

przełożony(józef, alicja) jest **prawdą**.

przełożony(józef, albert) jest **prawdą**.

przełożony(X, Y) jest **fałszem** dla wszystkich możliwych pozostałych kombinacji wartości X i Y.

RYСУNEK 26.13. Interpretacja będąca modelem minimalnym

Interpretacja jest nazywana **modelem** dla *określonego zestawu reguł*, jeżeli te reguły są *zawsze prawdziwe* dla tej interpretacji, co oznacza, że dla dowolnych wartości przypisanych zmiennym reguł nagłówek reguły jest prawdziwy, kiedy tą interpretacją zastąpimy

<sup>33</sup> Najczęściej wybierana dziedzina jest skończona i jest nazywana *przestrzenią Herbranda*.



wartości prawdy przypisane predykatom w ciele reguły. Zatem dla określonego podstawienia wartości zmiennym w regule, jeśli wszystkie predykaty w ciele reguły są przy tej interpretacji prawdziwe, to predykat w nagłówku reguły również musi być prawdziwy. Interpretacja pokazana na rysunku 26.13 jest modelem dla dwóch reguł, ponieważ w żadnym przypadku nie powoduje złamania reguł. Zauważmy, że reguła jest złamana, jeżeli określone przypisanie stałych wartości do zmiennych powoduje, że wszystkie predykaty w ciele są prawdziwe, ale predykat w nagłówku jest fałszywy. Na przykład, jeśli  $\text{nadzór}(a, b)$  i  $\text{przełożony}(b, c)$  są dla pewnej interpretacji prawdziwe, ale  $\text{przełożony}(a, c)$  jest fałszywe, to interpretacja nie może być modelem dla reguły rekurencyjnej

$$\text{przełożony}(X, Y) :- \text{nadzór}(X, Z), \text{przełożony}(Z, Y)$$

W podejściu modelowo-teoretycznym znaczenie reguł jest określane przez opracowanie dla nich modelu. Model nazywamy **minimalnym** dla pewnego zestawu reguł, jeśli nie można zmienić żadnego faktu z prawdy na fałsz, tak, aby nadal można było znaleźć dla nich model. Na przykład, rozważmy interpretację z rysunku 26.13 i przyjmijmy, że predykat  $\text{nadzór}$  jest zdefiniowany poprzez zbiór znanych faktów, a predykat  $\text{przełożony}$  jest określony jako interpretacja (model) dla reguł. Przypuśćmy, że dodamy do prawdziwych predykatów  $\text{nadzór}(\text{józef}, \text{bartek})$ . Pokazany model pozostaje niezmienny, ale nie jest on już modelem minimalnym, ponieważ zmiana wartości prawdy dla  $\text{nadzór}(\text{józef}, \text{bartek})$  z prawdy na fałsz nadal pozwala na korzystanie z tego samego modelu. Model pokazany na rysunku 26.13 jest modelem minimalnym dla zestawu faktów zdefiniowanych przez predykat  $\text{nadzór}$ .

Ogólnie, model minimalny odpowiadający zadanemu zbiorowi faktów w interpretacji modelowo-teoretycznej powinien być taki sam jak fakty generowane w interpretacji dowodowo-teoretycznej dla tego samego zestawu aksjomatów podstawowych i dedukcyjnych. Jednak powyższa własność jest prawdziwa tylko w przypadku reguł o prostej strukturze. Jeżeli pozwolimy na stosowanie negacji w definicjach reguł, to odpowiedniość pomiędzy interpretacjami *nie* zachodzi. Co więcej, w takim przypadku dla danego zbioru faktów może istnieć wiele różnych modeli minimalnych.

Trzecia możliwość interpretowania znaczenia reguł wiąże się z mechanizmem wnioskującym, który jest używany przez system do wywodzenia faktów z reguł. Mechanizm ten definiuje dla znaczenia reguł **interpretację obliczeniową**. Logiczny język programowania Prolog używa swojego mechanizmu wnioskowania do definiowania znaczenia reguł i faktów w programach pisanych w tym języku. Nie wszystkie programy w Prologu odpowiadają interpretacji dowodowo-teoretycznej lub modelowo-teoretycznej (zależy to od rodzaju reguł w programie). Dla wielu prostych programów w Prologu mechanizm dedukcyjny wnioskuje fakty odpowiadające interpretacji dowodowo-teoretycznej lub modelowi minimalnemu interpretacji modelowo-teoretycznej.

## 26.5.6. Programy w języku Datalog i ich bezpieczeństwo

Istnieją dwie główne metody definiowania wartości prawdy w rzeczywistych programach w języku Datalog. **Predykaty definiowane przez fakty** (lub **relacje**) są określane poprzez wymienienie wszystkich kombinacji wartości (krotek), które spełniają predykat. Odpowiada to podstawowym relacjom, których zawartość jest przechowywana w systemie bazy danych. Rysunek 26.14 pokazuje definiowane faktami predykaty  $\text{pracownik}$ ,  $\text{mężczyzna}$ ,

kobieta, dział, nadzór, projekt i pracujenad, które odpowiadają częściom relacyjnej bazy danych z rysunku 5.6. **Predykaty definiowane przez reguły (perspektywy)** są definiowane jako nagłówki (umieszczone po lewej stronie) jednej lub kilku reguł w Datalogu. Odpowiadają one *wirtualnym relacjom*, których zawartość jest wywnioskowana przez mechanizm wnioskowania. Rysunek 26.15 prezentuje wybrane predykaty definiowane przez reguły.

```
pracownik(jan).
pracownik(fryderyk).
pracownik(alicja).
pracownik(janina).
pracownik(robert).
pracownik(joanna).
pracownik(albert).
pracownik(józef).

pensja(jan, 3000).
pensja(fryderyk, 4000).
pensja(alicja, 2500).
pensja(janina, 4300).
pensja(robert, 3800).
pensja(joanna, 2500).
pensja(albert, 2500).
pensja(józef, 5500).

dział(jan, badania).
dział(fryderyk, badania).
dział(alicja, administracja).
dział(janina, administracja).
dział(robert, badania).
dział(joanna, badania).
dział(albert, administracja).
dział(józef, centrala).

nadzór(fryderyk, jan).
nadzór(fryderyk, robert).
nadzór(fryderyk, joanna).
nadzór(janina, alicja).
nadzór(janina, albert).
nadzór(józef, fryderyk).
nadzór(józef, janina).

mężczyzna(jan).
mężczyzna(fryderyk).
mężczyzna(robert).
mężczyzna(albert).
mężczyzna(józef).

kobieta(alicja).
kobieta(janina).
mężczyzna(joanna).
```

```

projekt(produktx).
projekt(produkty).
projekt(produktz).
projekt(komputeryzacja).
projekt(reorganizacja).
projekt(zwiekszeniezyskow).

pracujenad(jan, produktx, 32).
pracujenad(jan, produkty, 8).
pracujenad(robert, produktz, 40).
pracujenad(joanna, produktx, 20).
pracujenad(joanna, produkty, 20).
pracujenad(fryderyk, produkty, 10).
pracujenad(fryderyk, produktz, 10).
pracujenad(fryderyk, komputeryzacja, 10).
pracujenad(fryderyk, reorganizacja, 10).
pracujenad(alicja, zwiekszeniezyskow, 30).
pracujenad(alicja, komputeryzacja, 35).
pracujenad(albert, komputeryzacja, 10).
pracujenad(albert, zwiekszeniezyskow, 5).
pracujenad(janina, zwiekszeniezyskow, 20).
pracujenad(janina, reorganizacja, 15).
pracujenad(jozef, reorganizacja, 10).

```

RYSUNEK 26.14. Fakty odpowiadające fragmentowi schematu bazy danych z rysunku 5.6

```

przełożony(X, Y) :- nadzór(X, Y).
przełożony(X, Y) :- nadzór(X, Z), przełożony(Z, Y).

podwładny(X, Y) :- przełożony(Y, X).

kierownik(X) :- pracownik(X), nadzór(X, Y).
prac_ponad_4000(X) :- pracownik(X), pensja(X, Y), Y >= 4000.
kierownik_poniżej_4000(X) :- kierownik(X), not(prac_ponad_4000(X)).
pracownik_głównie_produktx(X) :- pracownik(X), pracujenad(X, produktx, Y), Y >= 20.
prezes(X) :- pracownik(X), not(nadzór(Y, X)).

```

RYSUNEK 26.15. Predykaty definiowane przez reguły

Mówimy, że program lub reguła są **bezpieczne**, jeżeli generują *skończony* zbiór faktów. Ogólnie, teoretyczny problem określania, czy dany zbiór reguł jest bezpieczny, jest nierozstrzygalny. Można jednak określić bezpieczeństwo reguł w ograniczonej formie. Na przykład reguły pokazane na rysunku 26.16 są bezpieczne. Jedną z sytuacji, które powodują powstanie niebezpiecznych reguł generujących nieskończone zbiory faktów, jest taka, gdy jedna ze zmiennych reguł należy do nieskończonej dziedziny wartości i nie jest ograniczona do skończonej relacji. Na przykład, rozważmy regułę

```

duża_pensja(Y) :- Y > 6000

rel_jeden(A, B, C).
rel_dwa(D, E, F).
rel_trzy(G, H, I, J).

```

```

wybierz_jeden_A_równe_c(X, Y, Z) :- rel_jeden(c, Y, Z).
wybierz_jeden_B_mniejsze_5(X, Y, Z) :- rel_jeden(X, Y, Z), Y<5.
wybierz_jeden_A_równe_c_i_B_mniejsze_5(X, Y, Z) :- rel_jeden(c, Y, Z), Y<5.

wybierz_jeden_A_równe_c_lub_B_mniejsze_5(X, Y, Z) :- rel_jeden(c, Y, Z).
wybierz_jeden_A_równe_c_lub_B_mniejsze_5(X, Y, Z) :- rel_jeden(X, Y, Z), Y<5.

proj_trzy_na_G_H(W, X) :- rel_trzy(W, X, Y, Z).

suma_jeden_dwa(X, Y, Z) :- rel_jeden(X, Y, Z).
suma_jeden_dwa(X, Y, Z) :- rel_dwa(X, Y, Z).

przecięcie_jeden_dwa(X, Y, Z) :- rel_jeden(X, Y, Z), rel_dwa(X, Y, Z).

różnica_dwa_jeden(X, Y, Z) :- rel_dwa(X, Y, Z) not(rel_jeden(X, Y, Z)).

iloczyn_kart_jeden_trzy(T, U, V, W, X, Y, Z) :- rel_jeden(T, U, V),
    rel_trzy(W, X, Y, Z).

złączenie_nat_jeden_trzy_C_równe_G(U, V, W, X, Y, Z) :- rel_jeden(U, V, W),
    rel_trzy(W, X, Y, Z).

```

RSUNEEK 26.16. Predykaty ilustrujące operacje relacyjne

Jeżeli  $Y$  obejmuje wszystkie możliwe liczby, to możemy uzyskać nieskończony wynik. Jeżeli jednak przekształcimy regułę do postaci:

```
duża_pensja(Y) :- pracownik(X), pensja(X, Y), Y>6000
```

w drugiej regule wynik nie jest nieskończony, ponieważ wartości  $Y$  są ograniczone do liczb będących pensją pracownika z bazy danych, czyli do zbioru skończonego. Możemy również zapisać regułę następująco:

```
duża_pensja(Y) :- Y>6000, pracownik(X), pensja(X, Y).
```

W tym przypadku reguła nadal jest teoretycznie bezpieczna. Jednakże w Prologu, czy dowolnym innym systemie używającym mechanizmu wnioskowania zstępującego i w głąb, taka reguła stworzy nieskończoną pętlę, ponieważ najpierw wyszukiwane będą możliwe wartości  $Y$ , a dopiero później będzie sprawdzany warunek, czy dana liczba jest pensją któregoś z pracowników. Wynikiem jest wygenerowanie nieskończonej liczby wartości  $Y$ , pomimo że od pewnego punktu nie będą one prowadziły do zbioru prawdziwych predykatów ciała reguł. W definicji języka Datalog obydwie reguły będą bezpieczne, ponieważ język ten nie zależy od żadnego konkretnego mechanizmu wnioskowania. Zaleca się jednak, aby pisać tego typu reguły w ich bezpieczniejszej formie, z predykatami ograniczającymi możliwe podstawienia na początku ciała reguły. Innym przykładem niebezpiecznej reguły jest:

```
ma_coś(X, Y) :- pracownik(X).
```

Tutaj znów istnieje możliwość wygenerowania nieskończonej liczby wartości  $Y$ , ponieważ  $Y$  jest wymieniona tylko w nagłówku reguły, a zatem nie jest ograniczona do skończonego zbioru wartości. Aby formalnie zdefiniować bezpieczne reguły, posłużymy się pojęciem zmiennej ograniczonej. Zmienna  $X$  jest **ograniczona** w regule, jeżeli (1) jest

wymieniona w (niewbudowanym) predykanie w ciele reguły; (2) jest użyta w predykanie postaci  $X=c$  lub  $c=X$ , lub  $(c1 \leq X \text{ and } X \leq c2)$  w ciele reguły, gdzie  $c, c1, c2$  są wartościami stałymi; lub (3) jest użyta w predykanie postaci  $X=Y$  lub  $Y=X$  w ciele reguły, gdzie  $Y$  jest zmienną ograniczoną. Mówimy, że reguła jest **bezpieczna**, jeśli wszystkie jej zmienne są ograniczone.

### 26.5.7. Zastosowanie operacji relacyjnych

Wiele operacji algebry relacyjnej można łatwo zapisać w postaci reguł języka Datalog, których wyniki są określane poprzez stosowanie tych operacji do zawartości bazy danych (fakty). Oznacza to łatwość definiowania zapytań relacyjnych i perspektyw. Dodatkową możliwością oferowaną przez Datalog jest możliwość specyfikacji zapytań rekurencyjnych oraz opartych na nich perspektyw. W tym punkcie pokażemy, jak zdefiniować kilka typowych operacji relacyjnych w postaci reguł języka Datalog. W przykładach użyte będą podstawowe relacje (predykaty definiowane przez fakty) `rel_jeden`, `rel_dwa` i `rel_trzy`, których schematy są przedstawione na rysunku 26.16. W Datalogu nie trzeba określać nazw atrybutów jak na rysunku 26.16, istotna jest za to arność (stopień) każdego z predykatów. W rzeczywistym systemie ważna jest również dziedzina (typ danych) każdego atrybutu, szczególnie dla operacji takich jak UNION (suma), INTERSECTION (przecięcie) i JOIN (złączenie). Przyjmujemy, że poszczególne typy atrybutów są w odpowiednich operacjach kompatybilne (patrz rozdział 3.).

Rysunek 26.16 ilustruje kilka podstawowych operacji relacyjnych. Zauważmy, że jeśli model języka Datalog jest oparty na modelu relacyjnym, a zatem predykaty (fakty i wyniki zapytań) odpowiadają zbiorom krotek, to duplikaty krotek w predykatkach są automatycznie eliminowane. W zależności od przyjętego przez system modułu wnioskującego dla języka Datalog, powyższa zależność może, ale nie musi być prawdziwa. Na pewno jednak *nie* jest prawdziwa w Prologu, więc wszelkie reguły z rysunku 26.16 związane z eliminacją duplikatów nie są w tym przypadku prawidłowe. Na przykład, chcąc określić w Prologu reguły dla operacji UNION z eliminacją duplikatów, musimy je przepisać następująco:

```
suma_jeden_dwa(X, Y, Z) :- rel_jeden(X, Y, Z).
suma_jeden_dwa(X, Y, Z) :- rel_dwa(X, Y, Z), not(rel_jeden(X, Y, Z)).
```

Reguły z rysunku 26.16 powinny jednak działać w Datalogu, jeżeli duplikaty są eliminowane automatycznie. Podobnie, reguły dla operacji projekcji pokazane na rysunku 26.16 będą działały w przypadku programu w języku Datalog, choć nie są prawidłowe w Prologu i powodują w nim powstawanie duplikatów.

### 26.5.8. Wykonywanie zapytań nierekurencyjnych

Aby używać Datalogu jako dedukcyjnego systemu baz danych, należy zdefiniować mechanizm wnioskowania oparty o koncepcję przetwarzania zapytań w relacyjnej bazie danych. Strategia wnioskowania opiera się na analizie wstępującej, poczynając od relacji podstawowych; kolejność wykonywania operacji może się zmieniać w zależności od optymalizacji zapytania. W tym punkcie omówimy **mechanizm wnioskowania** oparty na operacjach relacyjnych, które mogą być zastosowane w zapytaniach **nierekurencyjnych** w Datalogu. Do

ilustracji naszych rozważań posłużymy się faktami i regułami pokazanymi na rysunkach 26.14 i 26.15.

Jeżeli zapytanie dotyczy tylko predykatów definiowanych przez fakty, to wnioskowanie ogranicza się do wyszukiwania wyniku w danym zbiorze faktów. Przykładowe zapytanie

`dział(X, badawczy)?`

polega na znalezieniu wszystkich imion  $X$  pracowników działu badawczego. W algebrze relacyjnej można to zapytanie przedstawić następująco:

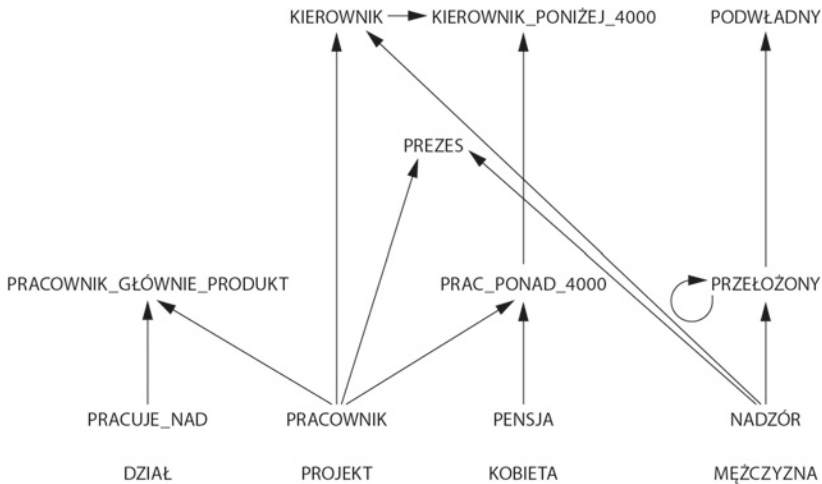
$\pi_{\$1} (\sigma_{\$2 = \text{"Badawczy"}} (\text{dział}))$

a jego wynik znaleźć, przeszukując listę predykatów definiowanych przez fakty `dział(X, Y)`. Zapytanie korzysta z operacji relacyjnych `SELECT` i `PROJECT` zastosowanych do podstawowej relacji, a takie działanie może być obsłużone za pomocą technik przetwarzania i optymalizacji zapytań do baz danych omówionych w rozdziale 19.

Kiedy zapytanie dotyczy predykatów definiowanych przez reguły, to mechanizm wnioskujący musi obliczyć wynik w oparciu o definicje reguł. Jeśli zapytanie nie jest rekurencyjne i dotyczy predykatu  $p$ , który jest nagłówkiem reguły  $p :- p_1, p_2, \dots, p_n$ , to najpierw oblicza się relacje odpowiadające symbolom  $p_1, p_2, \dots, p_n$ , a następnie relację odpowiadającą  $p$ . Zależności pomiędzy predykatami w dedukcyjnej bazie danych można wygodnie zapisywać w postaci **grafu zależności predykatów**. Rysunek 26.17 przedstawia taki graf dla faktów i reguł z rysunków 26.14 i 26.15. Każdemu predykatowi odpowiada w nim **węzeł**. Kiedy predykat  $A$  jest określony w ciele reguły, a jej nagłówkiem jest predykat, to mówimy, że  $B$  **zależy od**  $A$ , co w grafie zależności odpowiada krawędzi skierowanej od  $A$  do  $B$ . Oznacza to, że w celu obliczenia faktów dla predykatu  $B$  (nagłówka reguły) należy najpierw obliczyć fakty dla wszystkich predykatów  $A$  z ciała reguły. Jeśli graf zależności nie posiada cykli, to mówimy, że zestaw reguł jest **nierekurencyjny**, a jeśli w grafie istnieje choć jeden cykl, to zestaw jest **rekurencyjny**. Na rysunku 26.17 jest przedstawiony tylko jeden predykat rekurencyjny (przełożony), który posiada krawędź rekurencji skierowaną do siebie samego. Ponadto, ponieważ predykat podwładny zależy od przełożony, to dla jego obliczenia również wymagane jest użycie rekurencji.

Zapytanie, które zawiera jedynie nierekurencyjne predykaty, jest nazywane **zapytaniem nierekurencyjnym**. W tym punkcie omówimy mechanizmy wnioskowania tylko dla tego typu zapytań. Na rysunku 26.17 każde zapytanie, które nie dotyczy predykatów podwładny lub przełożony, jest nierekurencyjne. W grafie zależności predykatów węzły odpowiadające predykatom definiowanym faktami nie mają żadnych krawędzi przychodzących, ponieważ odpowiadające im fakty są zapisane w bazie danych. Zawartość tych predykatów może być obliczona przez bezpośrednie odczytanie krotek odpowiadających im relacji.

Głównym zadaniem mechanizmu wnioskującego jest obliczanie faktów odpowiadających predykatom zawartym w zapytaniu. Można to osiągnąć, generując **wyrażenie relacyjne** używające operatorów relacyjnych takich jak `SELECT`, `PROJECT`, `JOIN`, `UNION` i `SET DIFFERENCE` (przy zachowaniu odpowiednich zabezpieczeń), które po wykonaniu zwracają wynik zapytania. Zapytanie relacyjne może być następnie wykonane przy użyciu wewnętrznego optymalizatora i procesora zapytań systemu bazy danych. Za każdym razem, gdy mechanizm wnioskujący ma obliczyć zestaw faktów odpowiadający nierekurencyjnemu predykatowi  $p$ , najpierw znajduje wszystkie reguły mające  $p$  w swoim nagłówku. Ogólna



RYСУNEK 26.17. Graf zależności predykatów dla rysunków 26.15 i 26.16

koncepcja polega na znalezieniu dla każdej z takich reguł zbioru faktów, a następnie zastosowaniu do wyników operacji UNION, która odpowiada logicznej alternatywie OR. Graf zależności pokazuje wszystkie predykaty  $q$ , od których zależy  $p$ ; o ile  $p$  nie jest rekurencyjny, można dzięki temu określić częściowy porządek w zbiorze takich predykatów. Przed obliczeniem zbioru faktów dla  $p$  najpierw trzeba, opierając się na tym porządku, obliczyć zbiory faktów dla każdego  $q$ , od którego  $p$  zależy. Na przykład, jeżeli zapytanie dotyczy predykatu `kierownik_poniżej_4000`, to najpierw trzeba obliczyć predykaty `kierownik` i `prac_powyżej_4000`, a te zależą tylko od predykatów definiowanych faktami `pracownik`, `pensja` i `nadzór`, więc mogą być wyliczone bezpośrednio z zawartości relacji w bazie danych.

Na tym kończymy rozważania dotyczące dedukcyjnych baz danych. Dodatkowe materiały można znaleźć na stronie WWW poświęconej książce, na której dostępny jest pełny rozdział 25. trzeciej edycji. Prezentowane tam są również informacje na temat algorytmów przetwarzania zapytań rekurencyjnych. W końcowej części niniejszego rozdziału przedstawiona jest obszerna bibliografia z zakresu dedukcyjnych baz danych, przetwarzania zapytań rekurencyjnych, „zbiorów magicznych” (ang. *magic sets*), stosowania reguł dedukcyjnych w relacyjnych bazach danych i systemu GLUE-NAIL!.

## 26.6. Podsumowanie

W tym rozdziale wprowadzone zostały rozwiązania związane z pewnymi funkcjami baz danych wymaganymi w zaawansowanych aplikacjach: aktywne, czasowe, przestrzenne, multimedialne i dedukcyjne bazy danych. Istotną obserwacją jest fakt, że każdy z tych tematów jest bardzo szeroki i zasługuje na samodzielne opracowanie książkowe.

Na początku (w podrozdziale 26.1) wprowadzony został temat aktywnych baz danych, które zapewniają dodatkową funkcjonalność pozwalającą na definiowanie aktywnych reguł. Przedstawiony został również model zdarzenie-warunek-akcja (ECA — ang. *event-condition-action*) stosowany w tego typu bazach danych. Aktywne reguły mogą być auto-



matycznie uruchamiane przez zdarzenia takie jak aktualizacja bazy danych i mogą powodować określone akcje zdefiniowane w deklaracji reguły, o ile spełnione są określone warunki. Wiele komercyjnie dostępnych pakietów oferuje już pewne funkcje aktywnych baz danych w formie wyzwalaczy. W punkcie 26.1.1 wymieniliśmy przykładowe nisko-poziomowe wyzwalacze z komercyjnego systemu Oracle. W punkcie 26.1.2 zostały omówione różne opcje specyfikujące reguły, takie jak reguły poziomu wiersza i wyrażenia, przetwarzanie przed i po zdarzeniem oraz wykonanie natychmiastowe i opóźnione. W punkcie 26.1.3 przedstawiono przykłady reguły poziomu wyrażenia w eksperymentalnym systemie STARBURST. W punkcie 26.1.4 opisano wybrane zagadnienia projektowe i możliwe zastosowania aktywnych baz danych. W punkcie 26.1.5 omówiono składnię wyzwalaczy w standardzie SQL-99.

Następnie, w podrozdziale 26.2, wprowadzono koncepcję czasowych baz danych, które pozwalają na przechowywanie historii zmian i przeszukiwanie zarówno obecnych, jak i przeszłych stanów bazy danych. W punkcie 26.2.1 omówiono, jak reprezentowany jest czas i jakie są różnice pomiędzy wymiarami czasu rzeczywistego a czasu transakcji. W punkcie 26.2.2 opisano, jak rzeczywisty czas, czas transakcji i relacje dwuczasowe mogą być zaimplementowane w bazie danych przy użyciu wersji krotek w modelu relacyjnym wraz z przykładami ilustrującymi operacje dodawania, kasowania i zmieniania danych. Pokazano również zastosowanie złożonych obiektów w implementacji czasowych baz danych używających wersji atrybutów (punkt 26.2.3). W punkcie 26.2.4 przedstawiono pewne rodzaje zapytań w czasowych relacyjnych bazach danych oraz przegląd języka TSQL2.

W podrozdziale 26.3 przedstawiono przestrzenne bazy danych. Przestrzenne bazy danych pozwalają na pracę z obiektami o cechach przestrzennych. W punkcie 26.3.1 przedstawiono wprowadzenie do przestrzennych baz danych. W punkcie 26.3.2 omówiono rodzaje i modele danych przestrzennych, a w punkcie 26.3.3 — typy operatorów służących do przetwarzania danych przestrzennych i odmiany zapytań przestrzennych. W punkcie 26.3.4 zaprezentowano przegląd technik indeksowania przestrzennego, w tym często używanych R-drzew. Dalej, w punkcie 26.3.5, przedstawiono wybrane techniki eksploatacji danych przestrzennych, a w punkcie 26.3.6 — wybrane rozwiązania wymagające baz przestrzennych.

W podrozdziale 26.4 opisano podstawowe rodzaje multimedialnych baz danych i ich ważne cechy. Multimedialne bazy danych oferują użytkownikom możliwości przechowywania i odczytywania różnych typów danych multimedialnych, takich jak obrazy (zdjęcia i rysunki), wideo (filmy, relacje telewizyjne i prywatne nagrania), dźwięki (piosenki, telefoniczne wiadomości głosowe, nagrane wypowiedzi) oraz dokumenty (takie jak książki czy artykuły). Przedstawiono rodzaje źródeł danych multimedialnych i sposoby ich indeksowania. Dane w postaci obrazów są obecnie bardzo często zapisywane w bazach i nieraz stanowią dużą część przechowywanych danych. Dlatego szczegółowo omówiono zagadnienia związane z obrazami: ich automatyczną analizę (punkt 26.4.1), rozpoznawanie obiektów na obrazach (punkt 26.4.2) i opisywanie semantyczne (punkt 26.4.3). Mechanizmy z tych obszarów pomagają tworzyć lepsze systemy pobierania obrazów na podstawie ich treści, co wciąż stanowi duże wyzwanie. Ponadto w punkcie 26.4.4 omówiono analizę danych audio.

Na zakończenie rozdziału zaprezentowano wprowadzenie do dedukcyjnych baz danych (podrozdział 26.5). Bazy te przedstawiono w punkcie 26.5.1, a w punktach 26.5.2 i 26.5.3 opisano notacje używane w Prologu i Datalogu. W punkcie 26.5.4 omówiono formuły w postaci klauzulowej. W regułach w Datalogu można stosować tylko klauzule Horna, zawierające najwyżej jeden pozytywny literał. W punkcie 26.5.5 opisano dowodowo-teoretyczną i modelowo-teoretyczną interpretację reguł. W punkcie 26.5.6 pokrótce przedstawiono bezpieczeństwo reguł Datalogu, a w punkcie 26.5.7 — sposoby zapisu operatorów relacyjnych za pomocą reguł Datalogu. Na zakończenie, w punkcie 26.5.8, omówiono mechanizmy wnioskowania oparte na operacjach relacyjnych, które można stosować do przetwarzania nierekurencyjnych zapytań Datalogu za pomocą technik optymalizacji zapytań relacyjnych. Choć Datalog jest popularnym językiem używanym w niektórych obszarach, implementacje dedukcyjnych systemów baz danych (takie jak LDL lub VALIDITY) nie trafiły do powszechnego użytku komercyjnego.

## Pytania powtórkowe

- 26.1. Jakie są różnice pomiędzy aktywnymi regułami dla wierszy i wyrażeń?
- 26.2. Jakie są różnice pomiędzy natychmiastowym, opóźnionym i równoczesnym *rozważaniem* warunków aktywnych reguł?
- 26.3. Jakie są różnice pomiędzy natychmiastowym, opóźnionym i równoczesnym *wykonaniem* aktywnej reguły?
- 26.4. Omów problemy spójności i zatrzymania programu, które należy wziąć pod uwagę, projektując zestaw aktywnych reguł.
- 26.5. Przedstaw przykłady zastosowań aktywnych baz danych.
- 26.6. Omów reprezentacje czasu w czasowych bazach danych i porównaj różne wymiary czasu.
- 26.7. Jakie są różnice pomiędzy relacjami czasu rzeczywistego, czasu transakcji i relacjami dwuczasowymi?
- 26.8. Opisz, jak powinny być realizowane operacje dodawania, kasowania i zmieniania krotek w relacji czasu rzeczywistego.
- 26.9. Opisz, jak powinny być realizowane operacje dodawania, kasowania i zmieniania krotek w relacji dwuczasowej.
- 26.10. Opisz, jak powinny być realizowane operacje dodawania, kasowania i zmieniania krotek w czasie transakcji.
- 26.11. Jakie są podstawowe różnice pomiędzy wersjami krotek a wersjami atrybutów?
- 26.12. Czym różnią się przestrzenne bazy danych od zwykłych?
- 26.13. Podaj różne typy danych przestrzennych.
- 26.14. Wymień główne typy operatorów przestrzennych i kategorie zapytań przestrzennych.
- 26.15. Opisz cechy R-drzew używanych w indeksach danych przestrzennych.
- 26.16. Wyjaśnij, jak utworzyć indeks złączeń przestrzennych dla obiektów przestrzennych.
- 26.17. Wymień różne typy eksploracji danych przestrzennych.

- 26.18. Przedstaw ogólną postać reguły asocjacji przestrzennych. Podaj przykładowe reguły tego rodzaju.
- 26.19. Jakie są różne typy danych multimedialnych?
- 26.20. Jak są indeksowane różne dane multimedialne w bazach danych?
- 26.21. Wymień ważne cechy używane do porównywania obrazów.
- 26.22. Opisz różne sposoby rozpoznawania obiektów na obrazach.
- 26.23. Jak wykorzystywane jest semantyczne opisywanie obrazów?
- 26.24. Opisz trudności związane z analizowaniem danych audio.
- 26.25. Czym są dedukcyjne bazy danych?
- 26.26. Napisz proste reguły w Prologu, aby zdefiniować, że kursy o numerach powyżej INF5000 są kursami dla magistrantów, a MAGBD to magistranci zapisani na kursy INF6400 i CS8803.
- 26.27. Zdefiniuj, czym jest forma klauzulowa formuł i czym są klauzule Horna.
- 26.28. Czym jest dowodzenie twierdzeń? Czym jest dowodowo-teoretyczna interpretacja reguł?
- 26.29. Czym jest interpretacja modelowo-teoretyczna? W jaki sposób różni się od interpretacji dowodowo-teoretycznej?
- 26.30. Czym są predykaty zdefiniowane poprzez zbiór faktów i predykaty zdefiniowane jako interpretacja dla reguł?
- 26.31. Czym jest bezpieczna reguła?
- 26.32. Podaj przykładowe reguły, które mogą definiować operacje relacyjne SELECT, PROJECT, JOIN i SET.
- 26.33. Opisz mechanizm wnioskowania oparty na operacjach relacyjnych, który można zastosować do przetwarzania nierekurencyjnych zapytań w Datalogu.

## Ćwiczenia

- 26.34. Rozważ bazę danych FIRMA przedstawioną na rysunku 5.6. Używając składni wyzwalaczy Oracle, napisz aktywne reguły realizujące następujące zadania:
  - a) Kiedy zmieniany jest przydział pracownika do projektu, sprawdź, czy suma godzin spędzonych przez pracownika przy tym projekcie jest mniejsza niż 30 lub większa niż 40; jeżeli przydział spełnia te warunki — powiadom bezpośredniego przełożonego.
  - b) Za każdym razem, gdy usuwana jest krotka PRACOWNIK, usuń również powiązane z nią krotki RODZINA i PROJEKT, a jeżeli pracownik zarządzał działem lub miał podwładnych, ustaw atrybut PESELKIEROWNIKA dla działu oraz PESELPRZEŁOŻ dla podwładnych na wartość *null*.
- 26.35. Powtórz zadanie 26.34, używając do tworzenia aktywnych reguł składni STARBURST.
- 26.36. Dla schematu relacyjnego przedstawionego na rysunku 26.18 napisz aktywne reguły dostosowujące atrybut SUMA\_PROWIZJI relacji HANDLOWIEC do wartości sumy atrybutów PROWIZJA odpowiednich krotek z relacji SPRZEDAŻ. Reguły powinny

również sprawdzać, czy SUMA\_PROWIZJI przekracza 100 000 i (w przypadku przekroczenia) uruchamiać procedurę POWIADOM\_KIEROWNIKA(H\_ID). Opracuj zarówno reguły poziomu wyrażenia w notacji STARBURST, jak i reguły poziomu wiersza w notacji Oracle.

#### SPRZEDAŻ

H_ID	P_ID	PROWIZJA
------	------	----------

#### HANDLOWIEC

HANDLOWIEC_ID	IMIĘNAZWISKO	TYTUŁ	TELEFON	SUMA_PROWIZJI
---------------	--------------	-------	---------	---------------

RYSUNEK 26.18. Schemat bazy danych dotyczącej prowizji z ćwiczenia 26.36

- 26.37. Rozważając schemat EER UNIWERSYTET z rysunku 4.10, napisz zasady (po polsku), które wymuszają więzy integralności istotne dla tej aplikacji i mogą być zrealizowane za pomocą aktywnych reguł.
- 26.38. Omów, które aktualizacje krotek z rysunku 26.9 są zastosowane proaktywnie, a które retroaktywnie.
- 26.39. Pokaż, jak poniższe zmiany wprowadzane kolejno do relacji PRAC\_DC z rysunku 26.9 zmieniają jej zawartość. Dla każdej zmiany określ, czy jest to zmiana proaktywna, czy retroaktywna.
- 2004-03-10 o godzinie 17.30 pensja Głębockiego została zmieniona na 3500 i zaczęła obowiązywać od 2004-03-01.
  - 2003-07-30 o godzinie 8.31 pensja Nowaka została poprawiona w celu odzwierciedlenia, że jej wartość powinna być równa 2600 (a nie 2500). Nowa wartość obowiązuje od 2003-06-01.
  - 2004-03-18 o godzinie 8.31 zawartość bazy danych została zmieniona tak, by zaznaczyć odejście z firmy (logiczne usunięcie) Głębockiego dnia 2004-03-31.
  - 2004-04-20 o godzinie 14.07.33 dodano do bazy danych krotkę <"Malinowski", "75070512233", 1, null> odpowiadającą nowemu pracownikowi zatrudnionemu od 2004-04-20.
  - 2004-04-28 o godzinie 12.54.02 wprowadzono do bazy danych informację o odejściu z firmy Kowalskiego z dniem 2004-06-01.
  - 2004-05-05 o godzinie 13.07.33 zaznaczono ponowne przyjęcie Jankowskiego do pracy od 2004-05-01 na tym samym stanowisku i w tym samym dziale co poprzednio, ale z pensją wynoszącą 4000.
- 26.40. Pokaż, jak kolejne modyfikacje z ćwiczenia 26.39 wpłynęłyby na zawartość tabeli PRAC\_RC z rysunku 26.8.
- 26.41. Dodaj do przykładowej bazy danych z rysunku 26.11 następujące fakty:  
 nadzór(albert, bartek), nadzór(fryderyk, patrycja).
- Najpierw zmodyfikuj drzewo zwierzchnictwa z rysunku 26.11(b), odzwierciedlając tę zmianę. Następnie utwórz rysunek ilustrujący zstępujące przetwarzanie zapytania przełożony(józef, Y) z użyciem reguł 1. i 2. z rysunku 26.12.

- 26.42. Przyjmijmy następujący zbiór faktów dla relacji  $\text{rodzic}(X, Y)$ , gdzie  $Y$  jest rodzicem  $X$ :

$\text{rodzic}(a, aa), \text{rodzic}(a, ab), \text{rodzic}(aa, aaa), \text{rodzic}(aa, aab),$   
 $\text{rodzic}(aaa, aaaa), \text{rodzic}(aaa, aaab).$

oraz reguły

$r1: \text{przodek}(X, Y) :- \text{rodzic}(X, Y)$   
 $r2: \text{przodek}(X, Y) :- \text{rodzic}(X, Z), \text{przodek}(Z, Y)$

definiujące  $Y$  jako przodka  $X$ .

- a) Pokaż, jak obliczyć zapytanie Dataloga:

$\text{przodek}(aa, X)?$

Pokaż każdy kolejny krok rozumowania.

- b) Pokaż to samo zapytanie, obliczając tylko zmiany w relacji  $\text{przodek}$ , i używając za każdym razem reguły 2.

*(To pytanie zostało przeniesione z pracy Bancilhona i Ramakrishnana z 1986 r.)*

- 26.43. Rozważmy dedukcyjną bazę danych z następującymi regułami:

$\text{przodek}(X, Y) :- \text{ojciec}(X, Y)$   
 $\text{przodek}(X, Y) :- \text{ojciec}(X, Z), \text{przodek}(Z, Y)$

Zauważmy, że  $\text{ojciec}(X, Y)$  oznacza, iż  $Y$  jest ojcem  $X$ , a  $\text{przodek}(X, Y)$  oznacza, że  $Y$  jest przodkiem  $X$ .

Przyjmijmy następujące podstawowe fakty:

$\text{ojciec}(\text{Henryk}, \text{Zbigniew}), \text{ojciec}(\text{Zbigniew}, \text{Jan}), \text{ojciec}(\text{Jan}, \text{Adam}).$

- a) Stwórz modelowo-teoretycznej interpretacji powyższych reguł, używając podanych faktów.
- b) Rozważ bazę danych zawierającą powyższe relacje  $\text{ojciec}(X, Y)$  oraz relacje  $\text{brat}(X, Y)$  i  $\text{urodz}(X, B)$ , gdzie  $B$  jest datą urodzenia  $X$ . Określ regułę, która oblicza kuzynów pierwszego stopnia (ojcowie kuzynów muszą być braćmi).
- c) Pokaż kompletny program w języku Datalog z literałami opartymi na faktach i na regułach, który oblicza następującą relację: wymień pary kuzynów, gdzie pierwsza osoba jest urodzona po roku 1960, a druga po 1970. Przy definiowaniu literałów można użyć wbudowanego predykatu *większy niż*. *(Uwaga: należy również pokazać przykładowe fakty dla braci, daty urodzenia i osoby).*

- 26.44. Rozważ następujące reguły:

$\text{dostępny}(X, Y) :- \text{lot}(X, Y)$   
 $\text{dostępny}(X, Y) :- \text{lot}(X, Z), \text{dostępny}(Z, Y)$

gdzie  $\text{dostępny}(X, Y)$  oznacza, że miasto  $Y$  jest dostępne z miasta  $X$ , a  $\text{lot}(X, Y)$  oznacza, że istnieje lot z miasta  $X$  do miasta  $Y$ .

- a) Stwórz fakty opisujące następujące informacje:

Los Angeles, Nowy Jork, Chicago, Atlanta, Frankfurt, Paryż, Singapur i Sydney są miastami.

Istnieją następujące loty: z Los Angeles do Nowego Jorku, z Nowego Jorku do Atlanty, z Atlanty do Frankfurtu, z Frankfurtu do Atlanty, z Frankfurtu do Singapuru, z Singapuru do Sydney (*uwaga*: nie można przyjmować istnienia lotu powrotnego na tej samej trasie).

- b) Czy przedstawione dane są cykliczne? Jeśli tak, to w jakim sensie?
- c) Stwórz model teoretycznej interpretacji powyższych faktów i reguł (czyli interpretacji podobnej do tej z rysunku 26.13).
- d) Rozważ zapytanie:

dostępny(Atlanta, Sydney)?

Jak takie zapytanie będzie wykonywane? Wymień wszystkie kolejne kroki przetwarzania.

- e) Rozważ następujące predykaty definiowane przez reguły:

```
dostępny_z_powrotem(X, Y) :- dostępny(X, Y), dostępny(Y, X),
    czas_lotu(X, Y, Z)
```

Narysuj dla nich graf zależności (*uwaga*: `czas_lotu(X, Y, Z)` oznacza, że lot z  $X$  do  $Y$  zajmuje  $Z$  godzin).

- f) Napisz w Datalogu następujące zapytanie: jakie miasta są osiągalne w ciągu 12 godzin z Atlanty? Przyjmij istnienie wbudowanych predykatów takich jak `wiekszy-niz(X, Y)`. Czy to zapytanie może być łatwo przedstawione jako wyrażenie algebry relacyjnej? Dlaczego tak lub dlaczego nie?
- g) Przyjmując, że predykat `populacja(X, Y)` oznacza populację  $Y$  miasta  $X$ , rozważ następujące zapytanie: wymień wszystkie możliwe przypisania dla predykatu `para(X, Y)`, gdzie  $Y$  jest miastem osiągalnym z jedną przesiadką z  $X$ , które ma ponad milion mieszkańców. Zaprezentuj to zapytanie w Datalogu. Narysuj drzewo zapytania w znaczeniu terminów algebry relacyjnej.

## Wybrane publikacje

Książka autorstwa Zaniolo i in. (1997) zawiera kilka części, z których każda dotyczy zaawansowanych pojęć bazodanowych takich jak aktywne, czasowe, przestrzenne, tekstowe i multimedialne bazy danych. Widom i Ceri (1996) oraz Ceri i Fraternali (1997) skupiają się na systemach aktywnych baz danych. Snodgrass (1995) opisuje język TSQL2 oraz czasowy model danych. Khoshafian i Baker (1996), Faloutsos (1996) i Subrahmanian (1998) opisują pojęcia związane z multimedialnymi bazami danych, a praca Tansela i in. (1992) zawiera kilka rozdziałów o czasowych bazach danych. Czasowe rozszerzenia standardu SQL:2011 są opisane w pracy Kulkarni i Michelsa (2012).

Reguły systemu STARBURST są opisane przez Widoma i Finkelsteina (1990). Częścią wczesnych prac nad aktywnymi bazami danych był projekt HiPAC opisany przez Chakravarthy'ego i in. (1989) oraz Chakravarthy'ego (1990). Słownik pojęć związanych z czasowymi bazami danych został podany przez Jensena i in. (1994). Snodgrass (1987) opisuje TQuel — wczesny język zapytań dla czasowych baz danych.

Normalizacja w systemach czasowych jest zdefiniowana przez Navathe i Ahmeda (1989). Prace Patona (1999) oraz Patona i Diaza (1999) dotyczą aktywnych baz danych. Chakravarthy i in. (1994) opisują SENTINEL i obiektowe aktywne systemy baz danych. Lee i in. (1998) omawiają zarządzanie szeregami czasowymi.

W książce Shekhara i Chawla (2003) opisano różne aspekty przestrzennych baz danych, w tym modele, składowanie, indeksowanie i eksplorację danych przestrzennych. Praca Scholla i in. (2001) to następny podręcznik poświęcony zarządzaniu danymi przestrzennymi. Albrecht (1996) opisuje szczegółowo różne operacje analityczne w systemach GIS. Clementini i Di Felice (1993) szczegółowo opisują operatory przestrzenne. Güting (1994) omawia struktury danych przestrzennych i języki zapytań dla systemów przestrzennych baz danych. Guttman (1984) proponuje R-drzewa do indeksowania danych przestrzennych. Książka Manolopoulou i in. (2005) jest poświęcona teorii i zastosowaniom R-drzew. Papadias i in. (2003) opisują przetwarzanie za pomocą R-drzew zapytań dotyczących sieci przestrzennych. Ester i in. (2001) przedstawiają kompletne omówienie algorytmów i zastosowań eksploracji danych przestrzennych. Koperski i Han (1995) omawiają wykrywanie reguł asocjacji na podstawie geograficznych baz danych. Brinkhoff i in. (1993) przedstawiają wyczerpujące omówienie używania R-drzew do wydajnego przetwarzania złączeń przestrzennych. Rotem (1991) dokładnie opisuje indeksy złączeń przestrzennych. Praca Shekhara i Xiong (2008) to kompilacja różnych materiałów dotyczących rozmaitych aspektów systemów zarządzania przestrzennymi bazami danych i systemów GIS. Algorytmy klastrowania na podstawie gęstości, DBSCAN i DENCLUE, zostały zaproponowane przez Estera i in. (1996) oraz Hinnenberga i Gabriela (2007).

Modelowaniu danych multimedialnych poświęcona jest obszerna literatura. Trudno wymienić tu wszystkie ważne prace. System QBIC (ang. *Query By Image Content*) firmy IBM, opisany w pracy Niblacka i in. (1998), był jednym z pierwszych kompletnych systemów wyszukiwania obrazów na podstawie treści. Obecnie jest on dostępny jako rozszerzenie Image dla bazy DB2 firmy IBM. Zhao i Grosky (2002) omawiają wyszukiwanie obrazów na podstawie treści. Carneiro i Vasconcelos (2005) prezentują skupione na bazach danych spojrzenie na semantyczne opisywanie i wyszukiwanie obrazów. Wyszukiwanie fragmentów obrazów na podstawie treści opisano w Luo i Nascimento (2004). Tuceryan i Jain (1998) omówili różne aspekty analizy tekstur. Lowe (2004) opisuje rozpoznawanie obiektów za pomocą metody SIFT. Lazebnik i in. (2004) omawiają wykorzystanie lokalnych obszarów afinicznych do modelowania obiektów trójwymiarowych (technika RIFT). Jeśli chodzi o inne techniki rozpoznawania obiektów, to Kim i in. (2006) opisują podejście G-RIF, Bay i in. (2006) przedstawiają metodę SURF, Ke i Sukthankar (2004) — technikę PCA-SIFT, a Mikołajczyk i Schmid (2005) — podejście GLOH. Fan i in. (2004) prezentują technikę automatycznego opisywania obrazów za pomocą kluczowych obiektów. Praca Fotouhi i in. (2007) pochodzi z pierwszych międzynarodowych warsztatów poświęconych wielu aspektom semantyki w danych multimedialnych. Konferencja ta odbywa się corocznie. Thuraishingham (2001) dzieli dane audio na różne kategorie i traktując każdą z nich w odmienny sposób, omawia wykorzystanie metadanych do danych dźwiękowych. Prabhakaran (1996) opisuje, jak za pomocą technik przetwarzania mowy dodać cenne metadane do nagrań audio.

Wczesne osiągnięcia w dziedzinie łączenia logiki i baz danych są opisywane przez Gallairego i in. (1984). Reiter (1984) przedstawia rekonstrukcję teorii relacyjnych baz danych, a Levesque (1984) rozważa kwestię częściowej wiedzy w świetle logiki. Gallaire i Minker (1978) wydali jedną z pierwszych książek na ten temat. Szczegółowe opracowanie powiązań logiki i baz danych pojawia się w pracy Ullmana (1989, cz. 2.), a związany z tym tematem rozdział znajduje się również w części 1. (1988). Ceri, Gottlob i Tanca (1990)



prezentują kompletne, ale i zwięzłe opracowanie o logice i bazach danych. Książka autorstwa Dasa (1992) jest obszerną publikacją dotyczącą dedukcyjnych baz danych i programowania logicznego. Początki języka Datalog są przedstawione przez Maiera i Warrena (1988). Clocksin i Mellish (2003) przedstawiają wyśmienite omówienie języka Prolog.

Aho i Ullman (1979) podają wczesne algorytmy obsługujące zapytania rekurencyjne wykorzystujące operator punktu stałego. Bancilhon i Ramakrishnan (1986) przedstawiają bardzo dobry i dokładny opis technik przetwarzania zapytań rekurencyjnych ze szczegółowymi przykładami zastosowania metody naiwnej i półnaiwnej. Świetne artykuły o dedukcyjnych bazach danych i przetwarzaniu zapytań rekurencyjnych opublikowali również Warren (1992) oraz Ramakrishnan i Ullman (1993). Pełny opis metody półnaiwnej w oparciu o algebrę relacyjną jest podany przez Bancilhona (1985). Inne podejścia do przetwarzania zapytań rekurencyjnych zawierają między innymi strategię Vieille'a (1986), która jest strategią interpretowaną zstępująco, a Henschen-Naqvi (1984) opisuje strategię iteracyjną interpretowaną wstępująco. Balbin and Ramamohanrao (1987) omawiają rozszerzenie półnaiwnej metody różnicowej dla wielu predykatów.

Oryginalna praca o magicznych zbiorach powstała pod redakcją Bancilhona (1986), a Beeri i Ramakrishnan (1987) ją rozszerzyli. Mumick i in. (1990a) pokazują zastosowanie magicznych zbiorów do nierekurencyjnych zagnieżdżonych zapytań SQL. Inne metody optymalizacji reguł bez ich przepisowywania przedstawia Vieille (1986, 1987). Kifer i Lozinskii (1986) proponują jeszcze inną technikę. Bry (1990) omawia, jak można pogodzić przetwarzanie zstępujące i wstępujące. Whang i Navathe (1992) opisują rozszerzoną technikę rozłącznej postaci normalnej w zastosowaniu do rekurencji w wyrażeniach algebry relacyjnej, dla zapewnienia interfejsu pomiędzy systemem ekspertowym a relacyjną bazą danych.

Chang (1981) opisuje wczesny system łączący reguły dedukcyjne z relacyjnymi bazami danych. Prototyp systemu LDL jest opisany w pracy Chimenti i in. (1990). Krishnamurthy i Naqvi (1989) wprowadzają do LDL pojęcie *wyboru*. Zaniolo (1988) omawia kwestie języka dla systemu LDL. Przegląd języka CORAL prezentuje praca Ramakrishnana i in. (1992), a implementacja jest opisana przez Ramakrishnana i in. (1993). Rozszerzenie funkcji obiektowych nazwane CORAL++ jest opisane w pracy Srivastavy i in. (1993). Ullman (1985) przedstawia podstawy systemu NAIL!, który jest opisany przez Morrisa i in. (1987). Phipps i in. (1991) opisują dedukcyjny system baz danych GLUE-NAIL!.

Zaniolo (1990) prezentuje przegląd teoretycznych podstaw i praktycznego zastosowania dedukcyjnych baz danych. Nicolas (1997) opisuje niezwykłą historię osiągnięć prowadzących do powstania systemów typu DOOD (ang. *deductive object-oriented database*), a Falcone i in. (1997) przedstawia obecny stan w tej dziedzinie. O systemie VALIDITY piszą między innymi Friesen i in. (1995), Vieille (1997) i Dietrich i in. (1999).



# Wprowadzenie do wyszukiwania informacji i danych w internecie

W większości wcześniejszych rozdziałów z tej książki omawialiśmy techniki modelowania, projektowania, pobierania, przetwarzania transakcji i zarządzania dotyczące *danych strukturalnych*. W podrozdziale 13.1 opisaliśmy różnice między danymi strukturalnymi, półstrukturalnymi i niestukturalnymi. Wyszukiwanie informacji (ang. *information retrieval*) dotyczy głównie *danych niestukturalnych* oraz technik indeksowania, wyszukiwania i pobierania informacji z dużych niestukturalnych dokumentów. W rozdziale 24. (poświęconym technologiom NOSQL) opisaliśmy systemy takie jak MongoDB, dostosowane do obsługi danych w postaci dokumentów. W tym rozdziale<sup>1</sup> przedstawimy wprowadzenie do wyszukiwania informacji. Jest to bardzo obszerne zagadnienie, dlatego skupimy się na podobieństwach i różnicach między wyszukiwaniem informacji a technologiami bazodanowymi, a także na technikach indeksowania, które stanowią podstawę wielu systemów wyszukiwania informacji.

Oto struktura tego rozdziału: w podrozdziale 27.1 przedstawimy zagadnienia z obszaru wyszukiwania informacji (WI) i powiemy, jak systemy WI różnią się od tradycyjnych baz danych. Podrozdział 27.2 jest poświęcony omówieniu modeli wyszukiwania, stanowiących podstawę WI. W podrozdziale 27.3 omówimy różne typy zapytań stosowane w systemach WI. Podrozdział 27.4 zawiera omówienie wstępnego przetwarzania tekstu, a w podrozdziale 27.5 opiszemy indeksowanie, będące istotą każdego systemu WI. W podrozdziale 27.6 pokażemy różne miary stosowane do oceny wydajności systemów WI. W podrozdziale 27.7 szczegółowo przedstawimy analizy danych w internecie i ich związki z WI. Podrozdział 27.8 to krótkie wprowadzenie do obecnych trendów w WI. Podrozdział 27.9 zawiera podsumowanie rozdziału. Jeśli Czytelnik jest zainteresowany skrótowym przeglądem WI, zalecamy lekturę podrozdziałów od 27.1 do 27.6.

## 27.1. Zagadnienia z obszaru wyszukiwania informacji (WI)

**Wyszukiwanie informacji** to proces pobierania dokumentów z kolekcji w odpowiedzi na zapytanie (lub żądanie wyszukiwania) zgłoszone przez użytkownika. W tym podrozdziale przedstawimy przegląd zagadnień z tej dziedziny. W punkcie 27.1.1 przedstawimy wyszukiwanie informacji na ogólnym poziomie, a następnie opiszemy różne rodzaje i poziomy wyszukiwania. W punkcie 27.1.2 porównamy WI z technologiami bazodanowymi. W punkcie

---

<sup>1</sup> Współautorem tego rozdziału jest Saurav Sahay z Intel Labs.

27.1.3 znajdziesz krótką historię WI. Dalej (w punkcie 27.1.4) przedstawimy różne tryby interakcji użytkowników z systemami WI. W punkcie 27.1.5 najpierw opisemy typowy proces WI ze szczegółowym zestawem zadań, a następnie przedstawimy uproszczoną wersję procesu. Podrozdział zakończymy krótkim objaśnieniem bibliotek cyfrowych i internetu.

### 27.1.1. Wprowadzenie do wyszukiwania informacji

Najpierw omówimy różnice między danymi strukturalnymi i niestukturalnymi (patrz podrozdział 13.1), aby pokazać, jak wyszukiwanie informacji różni się od zarządzania danymi strukturalnymi. Przyjrzyj się relacji (lub tabeli) DOMY o następujących atrybutach:

DOMY(Nieruchomość#, Adres, Powierzchnia, Cena\_ofertowa)

Jest to przykład *danych strukturalnych*. Można porównać tę relację z dokumentami zakupu nieruchomości, stanowiącymi *dane niestukturalne*. Dokumenty tego rodzaju mogą być różne w poszczególnych miastach, a nawet hrabstwach danego stanu w Stanach Zjednoczonych. Dokument zakupu w danym stanie zawiera standardową listę klauzul opisanych w punktach podzielonych na sekcje, a także wstępnie określony (stały) tekst i zmienne fragmenty, których treść jest uzupełniana przez konkretnych nabywców i sprzedawców. Inne zmienne informacje mogą obejmować stopę oprocentowania kredytu, kwotę płatną z góry, datę zamknięcia transakcji itd. Dokumenty mogą też zawierać zdjęcia wykonane w trakcie oględzin domu. Informacje z takich dokumentów to *dane niestukturalne*. Można je przechowywać w różnych układach i formatach. Jako **informacje niestukturalne** uznajemy zwykle informacje bez dobrze zdefiniowanego formalnego modelu i języka formalnego używanego do reprezentowania danych oraz wnioskowania na ich temat. Takie informacje są oparte na zrozumieniu języka naturalnego.

Wraz z pojawieniem się sieci World Wide Web (WWW) ilość informacji niestukturalnych przechowywanych w wiadomościach i dokumentach zawierających dane tekstowe oraz multimedialne gwałtownie wzrosła. Takie dokumenty są przechowywane w różnych standardowych formatach, jak HTML, XML (patrz rozdział 13.) oraz liczne standardy formatowania danych audio i wideo. Wyszukiwanie informacji dotyczy składowania, indeksowania i wyszukiwania takich informacji zgodnie z potrzebami użytkowników. Problemy, z jakimi muszą radzić sobie systemy WI, są nasilane przez to, że liczba stron internetowych i interakcji już jest mierzona w miliardach i ciągle błyskawicznie rośnie. Wszystkie opisane wcześniej rodzaje danych niestukturalnych są dodawane z szybkością milionów dziennie, przez co przeszukiwana przestrzeń internetu szybko się powiększa.

Historycznie **wyszukiwanie informacji** to „dziedzina dotycząca struktury, analizy, organizacji, składowania, wyszukiwania i pobierania informacji”, jak zdefiniował to Gerald Salton, pionier WI<sup>2</sup>. Możemy nieco rozbudować tę definicję i stwierdzić, że WI jest związane z dokumentami niestukturalnymi i ma zaspokajać potrzeby użytkowników. Ta dziedzina istnieje jeszcze dłużej niż bazy danych, a pierwotnie dotyczyła wyszukiwania skatalogowanych informacji w bibliotekach na podstawie tytułów, autorów, tematów i słów kluczowych. Na uczelniach dziedzina WI od dawna znajdowała się w programach studiów z zakresu bibliotekoznawstwa i informatologii. Informacje w kontekście WI (inaczej niż w systemach relacyjnych baz danych) nie wymagają struktur zrozumiałych

---

<sup>2</sup> Patrz książka Saltona z 1968 r., *Automatic Information Organization and Retrieval*.

dla maszyn. Przykładami takich informacji są pisany tekst, streszczenia, dokumenty, książki, strony internetowe, e-maile, wiadomości w komunikatorach czy zbiory z bibliotek cyfrowych. Tak więc wszystkie informacje o swobodnej reprezentacji (niestrukturalne) lub półstrukturalne są związane z dziedziną WI.

W rozdziale 13. wprowadziliśmy modelowanie i wyszukiwanie informacji w formacie XML. W rozdziale 26. opisaliśmy zaawansowane typy danych, w tym dane przestrzenne, czasowe i multimedialne. Producenci relacyjnych SZBD w nowszych wersjach swoich produktów zapewniają moduły do obsługi wielu takich typów danych, a także danych w formacie XML. Te nowsze wersje są czasem nazywane *rozszerzonymi relacyjnymi SZBD* lub *obiektowo-relacyjnymi SZBD* (patrz rozdział 12.). Problem radzenia sobie z danymi niestrukturalnymi to w dużym stopniu problem wyszukiwania informacji, przy czym naukowcy z dziedziny baz danych stosują do niektórych problemów tego typu indeksowanie i techniki wyszukiwania.

Systemy WI wykraczają poza systemy baz danych, ponieważ nie ograniczają użytkowników do stosowania konkretnego języka zapytań ani nie wymagają znajomości struktury (schematu) lub zawartości konkretnej bazy. Systemy WI uwzględniają potrzeby użytkowników zapisane w formie **swobodnych żądań wyszukiwania** (czasem nazywanych **zapytaniami wyszukiwania słów kluczowych** lub po prostu **zapytaniami**) interpretowanych przez system. Choć historycznie WI przez dekady dotyczyło katalogowania, przetwarzania i dostępu do tekstu w postaci dokumentów, współcześnie dominującym sposobem wyszukiwania informacji stały się wyszukiwarki internetowe. Tradycyjne problemy indeksowania tekstu i umożliwiania przeszukiwania kolekcji dokumentów zostały zastąpione przekształcaniem internetu na szybko dostępne repozytorium ludzkiej wiedzy lub wirtualną bibliotekę cyfrową.

System WI można opisać na różnych poziomach: za pomocą typów *użytkowników*, typów *danych* i rodzajów *potrzeb związanych z informacjami*. Ważne są też wielkość i skala uwzględnianych repozytoriów informacji. Różne systemy WI są projektowane w celu eliminowania konkretnych problemów związanych z kombinacją określonych cech. Oto krótki opis tych cech:

**Typy użytkowników.** Użytkownicy znacznie się różnią ze względu na umiejętność interakcji z systemami obliczeniowymi. Te umiejętności zależą od wielu czynników, takich jak wykształcenie, kultura i wcześniejsze doświadczenia ze środowiskami obliczeniowymi. Użytkownik może być *ekspertem* (np. kuratorem lub bibliotekarzem), który szuka konkretnych i sprecyzowanych informacji, rozumie zakres i strukturę dostępnego repozytorium i formułuje zapytania adekwatne do zadania. Może też być *laikiem* o ogólnych potrzebach związanych z informacjami. Takie osoby nie potrafią formułować wysoce adekwatnych zapytań na potrzeby wyszukiwania. Możliwe, że student próbuje znaleźć informacje na nowy temat, badacz stara się połączyć różne punkty widzenia na kwestie historyczne, akademik weryfikuje twierdzenia innych naukowców lub klient chce kupić ubranie. Projektowanie systemów dostosowanych do różnych grup użytkowników jest ważnym zagadnieniem w dziedzinie WI, które zwykle analizuje się w obszarze nazywanym HCIR (ang. *Human-Computer Information Retrieval*).

**Typy danych.** Systemy wyszukiwania można dostosować do konkretnych typów danych. Przykładowo, z problemem wyszukiwania informacji na konkretny temat można zmierzyć się skuteczniej za pomocą niestandardowych systemów budowanych

z myślą o zbieraniu i wyszukiwaniu informacji powiązanych z konkretnym tematem. Repozytorium informacji może mieć organizację hierarchiczną opartą na hierarchii pojęć lub tematów. Tematyczne *systemy specyficzne dla dziedziny* lub *pionowe systemy WI* nie są tak rozbudowane ani zróżnicowane jak ogólna sieć WWW, która zawiera informacje na każdy temat. Ponieważ zbiory danych specyficzne dla dziedziny istnieją i są dostępne za pomocą konkretnych procesów, można z nich korzystać w wydajniejszy sposób za pomocą wyspecjalizowanych systemów. Rodzaje danych można opisać za pomocą różnych wymiarów, takich jak *szybkość generowania, różnorodność, ilość i wiarygodność*. Opisaliśmy je w podrozdziale 25.1.

**Rodzaje potrzeb związanych z informacjami.** W kontekście przeszukiwania internetu rodzaje potrzeb użytkowników można podzielić na nawigację, informacje i transakcje<sup>3</sup>. **Wyszukiwanie nawigacyjne** dotyczy znajdowania konkretnych informacji (np. strony Uniwersytetu Jagiellońskiego), które są szybko potrzebne użytkownikowi. **Wyszukiwanie informacyjne** związane jest z szukaniem aktualnych informacji na dany temat (np. badań prowadzonych na Wydziale Informatyki Uniwersytetu Jagiellońskiego; jest to klasyczne zadanie w systemach WI). Celem **wyszukiwania transakcyjnego** jest dotarcie do witryny, w której ma miejsce dalsza interakcja prowadząca do jakichś zdarzeń (takich jak dołączenie do sieci społecznościowej, zakupy produktów, dokonywanie rezerwacji przez internet, dostęp do baz danych itd.).

**Poziomy skali.** Oto słowa laureata Nagrody Nobla Herberta Simona:

*To, co konsumuje informacja, jest dość oczywiste — konsumuje ona uwagę odbiorców. Dlatego bogactwo informacji skutkuje ubóstwem uwagi i koniecznością wydajnego skupienia tej uwagi w obliczu nadmiaru źródeł informacji, które mogą ją konsumować<sup>4</sup>.*

Nadmiar źródeł informacji skutkuje wysokim stosunkiem szumu do sygnału w systemach WI. Zwłaszcza w internecie, gdzie indeksowane są miliardy stron, tworzone są interfejsy WI powiązane z wydajnymi, skalowalnymi i odpornymi na błędy algorytmami rozproszonego wyszukiwania, indeksowania, buforowania i scalania danych. Wyszukiwarki w systemach WI mogą działać tylko dla konkretnych zbiorów dokumentów. **Korporacyjne systemy wyszukiwania** obejmują rozwiązania do WI przeszukujące różne encje w firmowym **intranecie**, który składa się z sieci komputerów danej korporacji. Przeszukiwanymi encjami mogą być e-maile, korporacyjne dokumenty, podręczniki, wykresy, prezentacje, a także raporty dotyczące ludzi, spotkań i projektów. Korporacyjne systemy wyszukiwania w dużych, globalnych firmach zwykle uwzględniają setki milionów encji. W mniejszej skali działają osobiste systemy informacyjne, np. na desktopach i laptopach, nazywane **wyszukiwarkami desktopowymi** (np. Google Desktop lub OS X Spotlight). Służą one do wyszukiwania plików, katalogów i różnego rodzaju encji przechowywanych na komputerze. Istnieją też inne systemy, które korzystają z technologii P2P, np. z protokołu BitTorrent, i umożliwiają wymianę muzyki w postaci plików audio. Dostępne są też wyspecjalizowane wyszukiwarki dźwiękowe, np. od firm Yahoo! i Lycos.

---

<sup>3</sup> Szczegóły znajdziesz w pracy Brodera (2002).

<sup>4</sup> Za: Herbert A. Simon (1971), *Designing Organizations for an Information-Rich World*.

### 27.1.2. Porównanie baz danych i systemów WI

W dziedzinie informatyki bazy danych i systemy WI są ściśle ze sobą powiązane. Bazy danych służą do wyszukiwania informacji strukturalnych za pomocą dobrze zdefiniowanych formalnych języków reprezentowania danych i operowania nimi. Języki te są oparte na teoretycznych modelach danych, a ponadto opracowane zostały wydajne algorytmy działania operatorów pozwalające na szybkie wykonywanie złożonych zapytań. Z kolei WI dotyczy wyszukiwania danych niestukturalnych, zwykle przy użyciu ogólnych zapytań oraz bez ściśle zdefiniowanej logicznej reprezentacji schematycznej. Wybrane najważniejsze różnice między bazami danych i systemami WI są wymienione w tabeli 27.1.

TABELA 27.1. Porównanie baz danych i systemów WI

Bazy danych	Systemy WI
<ul style="list-style-type: none"> <li>• Dane strukturalne</li> <li>• Oparte na schemacie</li> <li>• Dominujący jest model relacyjny (stosuje się też modele obiektowy, hierarchiczny i sieciowy)</li> <li>• Model ustrukturyzowanych zapytań</li> <li>• W operacjach używa się wielu metadanych</li> <li>• Dla zapytań zwracane są dane</li> <li>• Wyniki są oparte na precyzyjnym dopasowaniu (zawsze są poprawne)</li> </ul>	<ul style="list-style-type: none"> <li>• Dane niestukturalne</li> <li>• Bez stałego schematu; różne modele danych (np. model przestrzeni wektorowej)</li> <li>• Modele swobodnie wprowadzanych zapytań</li> <li>• W operacjach używa się wielu danych</li> <li>• Dla żądań wyszukiwania zwracane są listy dokumentów lub wskaźniki</li> <li>• Wyniki są oparte na przybliżonym dopasowywaniu i pomiarze skuteczności (mogą być nieprecyzyjne i porządkowane według trafności)</li> </ul>

Bazy danych mają stałe schematy zdefiniowane w modelu danych takim jak model relacyjny. Systemy WI nie mają stałego modelu danych, a dane lub dokumenty są w nich postrzegane za pomocą jakiegoś schematu, np. modelu przestrzeni wektorowej, aby ułatwić przetwarzanie zapytań (patrz podrozdział 27.2). Bazy danych oparte na modelu relacyjnym korzystają z języka SQL do przetwarzania zapytań i transakcji. Zapytania są odwzorowywane na operacje algebry relacyjnej i algorytmy wyszukiwania (patrz rozdział 19.), a jako wynik zapytania zwracają nową relację (tabelę), będącą dokładną odpowiedzią na zapytanie z uwzględnieniem aktualnego stanu bazy. W systemach WI nie ma stałego języka definiowania struktury (schematu) dokumentu lub operowania dokumentami. Zapytania mają tu zwykle postać słów kluczowych lub swobodnie wpisywanych zdań w języku naturalnym. Wyniki zapytań w systemach WI to listy identyfikatorów dokumentów, fragmenty obiektów tekstowych lub multimedialnych (obrazów, filmów itd.) albo listy odsyłaczy do stron internetowych.

Wynik dla zapytania w bazie danych jest precyzyjny. Jeśli w relacji nie znaleziono pasujących rekordów (krotek), wynik jest pusty (wartość null). Z kolei odpowiedź na żądanie użytkownika w zapytaniu w systemie WI reprezentuje próbę wyszukania przez taki system informacji najbardziej adekwatnych do danego zapytania. Systemy baz danych przechowują obszerne metadane i umożliwiają korzystanie z nich do optymalizowania zapytań, natomiast operacje w systemach WI uwzględniają tylko wartości danych i częstotliwość ich występowania. Czasem przeprowadzane są złożone analizy statystyczne, aby ustalić *adekwatność* każdego dokumentu lub jego fragmentu względem żądania użytkownika.



### 27.1.3. Krótka historia WI

Wyszukiwanie informacji to standardowe zadanie wykonywane od czasów starożytnych cywilizacji. Polega na projektowaniu sposobów porządkowania, składowania i katalogowania dokumentów oraz rekordów. W starożytności do zapisu informacji używano nośników takich jak zwoje papirusowe i tabliczki kamienne. Pozwalało to zachować wiedzę i przekazywać ją przez pokolenia. Wraz z pojawieniem się bibliotek publicznych i pras drukarskich rozpoczęła się ewolucja działających na dużą skalę metod tworzenia, zbierania, archiwizowania i udostępniania dokumentów. Powstanie komputerów i automatycznych systemów składowania danych wymagało zastosowania wcześniejszych metod do skomputeryzowanych rozwiązań. W latach 50. powstało kilka technik. Jedną z nich została opisana w przełomowej pracy H.P. Luhn<sup>5</sup>, który zaproponował używanie słów i liczby ich wystąpień jako jednostek indeksowania dokumentów oraz stosowanie miar pokrywania się słów w zapytaniach i dokumentach jako kryterium wyszukiwania. Szybko zauważono, że składowanie dużych ilości tekstu nie stanowi problemu. Trudniejsze było wyszukiwanie i pobieranie takich informacji dla użytkowników o specyficznych potrzebach związanych z informacjami. Metody wykorzystujące statystyki rozkładu słów doprowadziły do wybierania słów kluczowych na podstawie cech ich rozkładu<sup>6</sup> i do schematów z ważonymi słowami kluczowymi.

We wczesnych eksperymentach z systemami wyszukiwania dokumentów, np. w systemie SMART<sup>7</sup> z lat 60., zastosowano do indeksowania *strukturę plików odwróconych* opartą na słowach kluczowych i ich wagach (indeksy odwrócone opisano w punkcie 17.6.4). Organizacja sekwencyjna okazała się nieadekwatna, gdy zapytania wymagały szybkich odpowiedzi (zwracanych niemal w czasie rzeczywistym). Odpowiednia organizacja takich plików stała się ważnym obszarem badań. Powstały schematy klasyfikowania i klastrowania dokumentów. Wyzwaniem pozostało przeprowadzanie eksperymentów nad wyszukiwaniem w odpowiedniej skali, ponieważ duże zbiory tekstów były niedostępne. Wraz z powstaniem sieci WWW szybko się to zmieniło. W 1992 r. instytut NIST (ang. *National Institute of Standards and Technology*) zainicjował konferencję TREC (ang. *Text Retrieval Conference*) w ramach programu TIPSTER<sup>8</sup>, którego celem było opracowanie platformy oceny metod wyszukiwania informacji i ułatwienie transferu technologii na potrzeby rozwoju produktów z dziedziny WI.

**Wyszukiwarki** są praktycznym zastosowaniem wyszukiwania informacji do dużych kolekcji dokumentów. Dzięki szybkiemu postępowi w technologiach komputerowych i komunikacyjnych ludzie mają obecnie interaktywny dostęp do olbrzymich ilości generowanych przez użytkowników materiałów dostępnych w sieci WWW. Spowodowało to błyskawiczny rozwój technologii wyszukiwarek, które próbują w czasie rzeczywistym wykrywać różnego rodzaju materiały znajdujące się w tej sieci. Moduł wyszukiwarki odpowiedzialny za wykrywanie, analizowanie i indeksowanie nowych dokumentów to **robot indeksujący** (ang. *crawler*). Istnieją różne typy wyszukiwarek dla specyficznych dziedzin wiedzy. Przykładowo, w latach 70. uruchomiono bazę danych do przeszukiwania literatury

---

<sup>5</sup> Patrz Luhn (1957), „A statistical approach to mechanized encoding and searching of literary information”.

<sup>6</sup> Patrz Salton, Yang i Yu (1975).

<sup>7</sup> Szczegółowe informacje znajdziesz w Buckley i in. (1993).

<sup>8</sup> Szczegółowe informacje znajdziesz w Harman (1992).

medycznej, obecnie powiązaną z wyszukiwarką PubMed<sup>9</sup> i zapewniającą dostęp do ponad 24 milionów streszczeń prac.

Choć wciąż dokonywane są postępy na drodze do dostosowania wyników wyszukiwania do potrzeb użytkowników końcowych, wyzwaniem pozostaje szybkie udostępnianie adekwatnych informacji wysokiej jakości, precyzyjnie dopasowanych do potrzeb konkretnych osób.

## 27.1.4. Tryby interakcji w systemach WI

Na początku podrozdziału 27.1 *wyszukiwanie informacji* zdefiniowano jako proces wyszukiwania dokumentów w kolekcji w odpowiedzi na zgłoszone przez użytkownika zapytanie (lub żądanie wyszukiwania). Taka kolekcja zwykle składa się z dokumentów zawierających dane niestrukturalne. Innego rodzaju dokumenty to obrazy, nagrania audio, filmy i mapy. Dane mogą być rozproszone niejednorodnie po dokumentach bez ściśle określonej struktury. **Zapytanie** to zbiór **pojęć** (nazywanych **słowami kluczowymi**) używany przez wyszukiującego do określenia potrzebnych informacji. Przykładowo, pojęcia *bazy danych* i *systemy operacyjne* można potraktować jako zapytanie do bibliograficznej bazy danych z dziedziny nauk komputerowych. Żądanie informacji lub zapytanie wymagające wyszukiwania może być też wyrażeniem albo pytaniem w języku naturalnym, np. „Jaka waluta obowiązuje w Chinach?” lub „Znajdź włoskie restauracje w Gdańsku”.

Istnieją dwa główne tryby interakcji z systemami WI: wyszukiwanie i przeglądanie. Choć ich cel jest podobny, realizuje się je za pomocą różnych zadań. **Wyszukiwanie** dotyczy pobierania adekwatnych informacji z repozytorium dokumentów za pomocą zapytania do systemu WI. W **przeglądaniu** istotna jest eksploracja użytkownika, który wyświetla podobne lub powiązane dokumenty na podstawie oceny ich adekwatności albo nawiguje po takich dokumentach. W trakcie przeglądania informacje potrzebne użytkownikowi nie są definiowane z góry i mogą się zmieniać. Rozważ następujący scenariusz przeglądania: użytkownik jako słowo kluczowe podaje „Kraków”. System WI wyszukuje odsyłacze do adekwatnych dokumentów dotyczących różnych aspektów związanych z Krakowem. Użytkownik w jednym ze zwróconych dokumentów natrafia na pojęcie „AGH” i za pomocą pewnej techniki dostępu (np. kliknięcia tego pojęcia w dokumencie z wbudowanym odsyłaczem) wyświetla dokumenty o uczelni AGH z tej samej lub innej witryny (albo repozytorium). Potem użytkownik wpisuje słowo „Sport” prowadzące do informacji o różnych programach sportowych na uczelni AGH. Ostatecznie kończy wyszukiwanie, dochodząc do jesiennego planu zajęć drużyny pływackiej AGH, co bardzo go interesuje. Tego rodzaju aktywność użytkownika to przeglądanie. **Hiperłącza** służą do łączenia stron WWW i korzysta się z nich przede wszystkim do przeglądania. **Etykiety** (ang. *anchor*) to fragmenty tekstu w dokumentach służące jako etykiety hiperłączy i mające bardzo istotne znaczenie w trakcie przeglądania.

**Wyszukiwanie w sieci WWW** łączy oba aspekty (przeglądania i wyszukiwania), a obecnie jest jednym z głównych zastosowań WI. Strony WWW są odpowiednikami dokumentów. Wyszukiwarki przechowują indeksowane repozytorium stron WWW, zwykle z użyciem indeksów odwróconych (patrz podrozdział 27.5). W odpowiedzi na żądanie wyszuki-

---

<sup>9</sup> Patrz <http://www.ncbi.nlm.nih.gov/pubmed/>.

wania pobierane są strony WWW najbardziej adekwatne dla użytkownika. Czasem wyniki są porządkowane malejąco według poziomu adekwatności. **Ranking strony WWW** w wyszukanym zbiorze to miara jej adekwatności względem zapytania, na którego podstawie wygenerowano dany zbiór wyników.

## 27.1.5. Ogólny proces WI

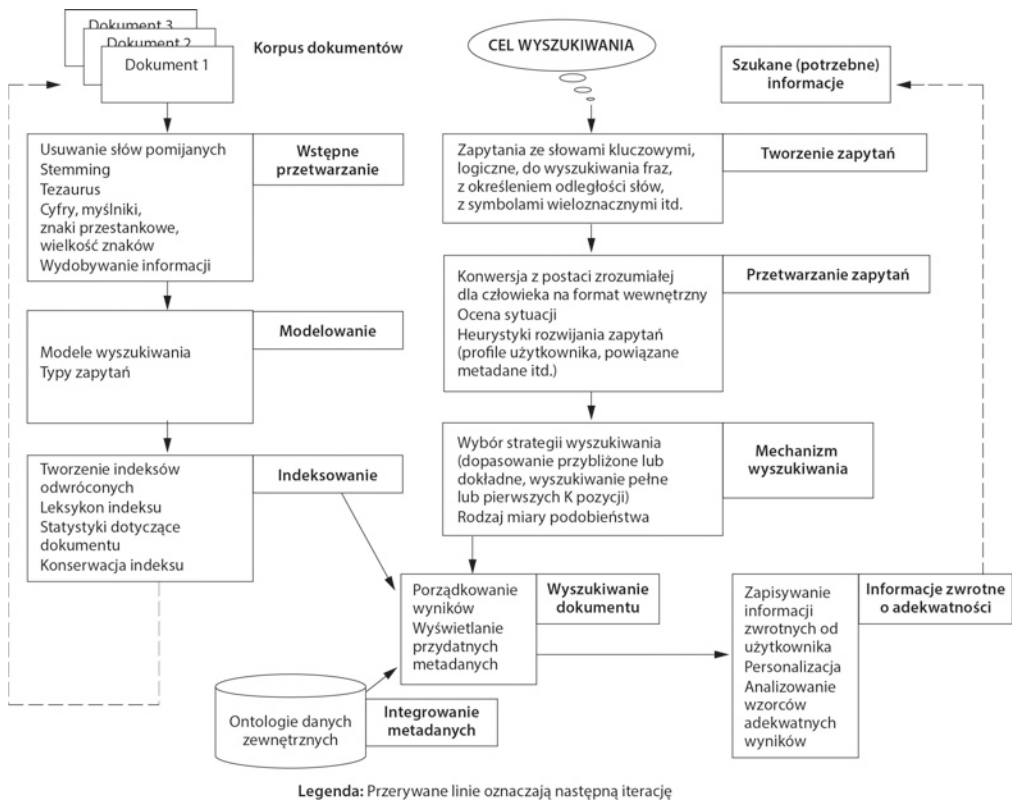
Jak wcześniej wspomniano, dokumenty składają się z niestukturalnego tekstu w języku naturalnym, obejmującego ciągi znaków w języku polskim lub innym. Standardowe przykłady dokumentów to depesze prasowe (np. agencji AP lub Reuters), korporacyjne podręczniki i raporty, ogłoszenia rządowe, artykuły ze stron WWW, blogi, tweety, książki i artykuły z magazynów. Są dwa podstawowe podejścia do WI: statystyczne i semantyczne.

W **podejściu statystycznym** dokumenty są analizowane i dzielone na porcje tekstu (słowa, zdania lub  $n$ -gramy, czyli wszystkie podciągi o długości  $n$  znaków w tekście lub dokumencie), a każde słowo lub zdanie jest zliczane, ważone i mierzone pod kątem adekwatności lub znaczenia. Słowa i ich właściwości są następnie porównywane z pojęciami z zapytania ze względu na poziom dopasowania, aby wygenerować uporządkowaną listę wynikowych dokumentów zawierających szukane wyrazy. Techniki statystyczne są dodatkowo dzielone według stosowanej metody. Trzy główne techniki statystyczne to podejścia: logiczne, oparte na przestrzeni wektorowej i probabilistyczne (patrz podrozdział 27.2).

W **podejściach semantycznych** wykorzystywane są techniki oparte na wiedzy, gdzie wykorzystuje się składniowe, leksykalne, zdaniowe, dyskursywne i pragmatyczne poziomy zrozumienia wiedzy. W praktyce w podejściach semantycznych uwzględnia się też analizy statystyczne, aby usprawnić proces wyszukiwania.

Na rysunku 27.1 pokazane są różne etapy pracy systemów WI. Kroki widoczne po lewej stronie rysunku są zwykle procesami offline, przygotowującymi zbiór dokumentów na potrzeby wydajnego wyszukiwania. Te etapy to: wstępne przetwarzanie dokumentów, modelowanie dokumentów i indeksowanie. Po prawej stronie rysunku widoczny jest proces interakcji użytkownika z systemem WI związany z zapytaniem, przeglądaniem lub wyszukiwaniem. Przedstawione są tu wykonywane kroki: formowanie zapytania, przetwarzanie zapytania, działanie mechanizmu wyszukiwania, pobieranie dokumentu i generowanie informacji zwrotnych o adekwatności. W każdym polu podkreślamy ważne zagadnienia i problemy. W dalszej części rozdziału opisujemy wybrane zagadnienia związane z różnymi zadaniami z procesu WI z rysunku 27.1.

Na rysunku 27.2 widoczny jest uproszczony potok przetwarzania z systemów WI. W celu wyszukiwania dokumentów najpierw są one reprezentowane w odpowiedniej do tego postaci. Ważne pojęcia i ich cechy są wydobywane z dokumentów i umieszczane w indeksie dokumentu, gdzie słowa, wyrażenia i ich cechy są zapisywane w tabeli, której wiersze zawierają referencje do słów występujących w poszczególnych dokumentach. Taki indeks jest następnie przekształcany na indeks odwrócony (patrz rysunek 27.4), w którym słowa i wyrażenia pozwalają znaleźć dokumenty. Na podstawie słów z zapytania dokumenty zawierające te pojęcia i cechy tych dokumentów (data utworzenia, autor, typ dokumentu) są pobierane z indeksu odwróconego i porównywane z zapytaniem. Ta operacja prowadzi do utworzenia uporządkowanej listy wyświetlanej użytkownikowi. Użytkownik



RYSUNEK 27.1. Ogólna platforma WI

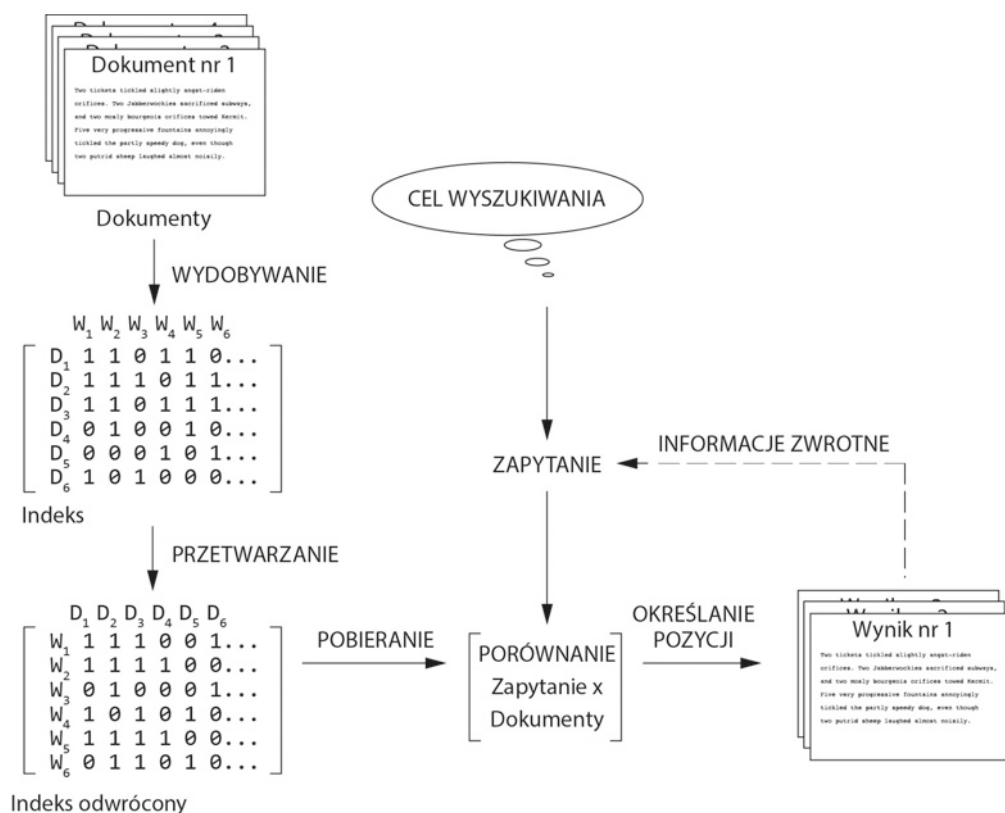
może następnie przekazać informacje zwrotne na temat wyników, co prowadzi do pośredniego lub bezpośredniego zmodyfikowania i rozwinięcia zapytania w celu pobrania wyników bardziej adekwatnych dla danej osoby. Większość systemów WI umożliwia interaktywne wyszukiwanie ze stopniowym dopracowywaniem zapytania i wyników.

## 27.2. Modele wyszukiwania

W tym podrozdziale pokrótce opiszemy ważne modele WI. Używane są tu trzy główne modele statystyczne — logiczny, przestrzeni wektorowej i probabilistyczny — a także model semantyczny.

### 27.2.1. Model logiczny

W tym modelu dokumenty są reprezentowane jako zbiór *wyrażeń*. Zapytania są formułowane jako kombinacja wyrażeń za pomocą standardowych operatorów logicznych z teorii zbiorów, takich jak I, LUB i NIE. Wyszukiwanie i adekwatność są w tym modelu traktowane zerojedynkowo. Dlatego pobrane elementy to wynik „dokładnego dopasowania” adekwatnych dokumentów. Nie ma tu określania pozycji znalezionych dokumentów. Wszystkie



RYSUNEK 27.2. Uproszczony potok WI

wyszukane dokumenty są traktowane jako równie istotne. Jest to poważne uproszczenie, w którym nie są uwzględniane liczby wystąpień wyrażeń w dokumencie ani podobieństwo innych słów do wyrażeń z zapytania.

W logicznych modelach wyszukiwania brak jest zaawansowanych algorytmów rankingowych. Należą one do najwcześniejszych i najprostszych modeli WI. Pozwalają łatwo dodawać metadane i pisać zapytania dopasowujące zawartość dokumentu oraz inne jego cechy, takie jak data utworzenia, autor i typ.

## 27.2.2. Model oparty na przestrzeni wektorowej

Ten model zapewnia platformę umożliwiającą stosowanie wag wyrażeń, tworzenie rankingów znalezionych dokumentów i uwzględnianie adekwatności informacji zwrotnych. Jeśli poszczególne wyrażenia są traktowane jak wymiary, każdy dokument jest reprezentowany za pomocą  $n$ -wymiarowego wektora wartości. Można tu stosować wartości logiczne, aby odzwierciedlić występowanie lub brak danego wyrażenia w dokumencie; można też posłużyć się liczbą reprezentującą wagę lub liczbę wystąpień wyrażenia w dokumencie. **Właściwości** (ang. *features*) to podzbiór wyrażeń w zbiorze dokumentów uznany za najistotniejszy w kontekście WI w tym zbiorze. Proces wyboru ważnych wyrażeń (właściwości) i ich cech w postaci rzadkiej (ograniczonej) listy spośród bardzo dużej liczby dostępnych wyrażeń (leksy-

kon może obejmować setki tysięcy wyrażen) jest niezależny od specyfikacji modelu. Zapytanie jest podawane jako wektor wyrażen (wektor właściwości) porównywany z wektorami dokumentów pod kątem podobieństwa i adekwatności.

Funkcja oceny podobieństwa porównująca dwa wektory nie jest stała. Można wykorzystać różne funkcje oceny podobieństwa. Często używany jest tu kosinus kąta między wektorami zapytania i dokumentu. Gdy kąt jest mniejszy, kosinus zbliża się do wartości jeden. Oznacza to, że podobieństwo zapytania do wektora dokumentu rośnie. Dla wyrażen (właściwości) stosowane są wagi proporcjonalne do liczby wystąpień. Odzwierciedla to znaczenie wyrażen w obliczeniach adekwatności. Różni się to od modelu logicznego, gdzie liczba wystąpień słów w dokumencie nie ma wpływu na ocenę adekwatności.

W modelu wektorowym waga  $w_{ij}$  wyrażenia  $i$  z dokumentu ( $j$ ) jest reprezentowana na podstawie jakiejś odmiany opisanego dalej schematu TF (ang. *term frequency*) lub TF-IDF (ang. *term frequency – inverse document frequency*). **TF-IDF** to statystyczna miara wagi używana do oceny znaczenia słów z dokumentu w kolekcji dokumentów. Zwykle stosowany jest tu następujący wzór:

$$\text{kosinus}(d_j, q) = \frac{\langle d_j \times q \rangle}{\|d_j\| \times \|q\|} = \frac{\sum_{i=1}^{|V|} w_{ij} \times w_{iq}}{\sqrt{\sum_{i=1}^{|V|} w_{ij}^2} \times \sqrt{\sum_{i=1}^{|V|} w_{iq}^2}}$$

W tym wzorze korzystamy z następujących symboli:

- $d_j$  to wektor dokumentu  $j$ ;
- $q$  to wektor zapytania;
- $w_{ij}$  to waga wyrażenia  $i$  w dokumencie  $j$ ;
- $w_{iq}$  to waga wyrażenia  $i$  w wektorze zapytania  $q$ ;
- $|V|$  to liczba wymiarów wektora; odpowiada ona łącznej liczbie ważnych słów kluczowych (właściwości).

W TF-IDF w celu ustalenia wagi wyrażenia w dokumencie używany jest iloczyn znormalizowanej częstotliwości wystąpień wyrażenia  $i$  ( $TF_{ij}$ ) w dokumencie  $D_j$  i odwrotności częstotliwości występowania wyrażenia  $i$  w zbiorze dokumentów ( $IDF_i$ ). Zasada jest taka, że wyrażenia ujmujące istotę dokumentu często w nim występują (ich TF jest wysoka), ale żeby dobrze odróżniały dany dokument od pozostałych, muszą pojawiać się w niewielkiej liczbie dokumentów w ogólnej populacji (ich IDF też powinna być wysoka).

Wartości IDF można łatwo obliczyć dla stałej kolekcji dokumentów. W wyszukiwarkach internetowych przybliżone wartości IDF są obliczane na podstawie reprezentatywnej próbki dokumentów. Można wykorzystać następujące wzory:

$$TF_{ij} = \frac{f_{ij}}{\sum_{i=1 \text{ do } |V|} f_{ij}}$$

$$IDF_i = \log(N / n_i)$$

Oto znaczenie symboli z tych wzorów:

- $TF_{ij}$  to znormalizowana częstotliwość występowania wyrażenia  $i$  w dokumencie  $D_j$ ;
- $f_{ij}$  to liczba wystąpień wyrażenia  $i$  w dokumencie  $D_j$ ;



- $IDF_i$  to odwrotność częstotliwości występowania wyrażenia  $i$  w zbiorze dokumentów;
- $N$  to liczba dokumentów w kolekcji;
- $n_i$  to liczba dokumentów, w których występuje wyrażenie  $i$ .

Zauważ, że jeśli wyrażenie  $i$  występuje we wszystkich dokumentach, to  $n_i = N$ , a  $IDF_i = \log(1)$  wynosi zero, przez co znaczenie wyrażenia jest zerowe i powstaje sytuacja, w której może nastąpić dzielenie przez zero. Waga wyrażenia  $i$  w dokumencie  $j$ ,  $w_{ij}$ , jest w niektórych technikach obliczana na podstawie wartości TF-IDF. Aby zapobiec dzieleniu przez zero, we wzorach takich jak wcześniejszy wzór na kosinus do mianownika często dodawane jest jeden.

Czasem znaczenie dokumentu dla zapytania,  $\text{znac}(D_j, Q)$ , jest bezpośrednio mierzone jako suma wartości TF-IDF wyrażen z zapytania  $Q$ :

$$\text{znac}(D_j, Q) = \sum_{i \in Q} TF_{ij} \times IDF_i$$

Współczynnik normalizacji (podobny do mianownika we wzorze na kosinus) jest uwzględniany w samym wzorze na TF-IDF. Pozwala to mierzyć znaczenie dokumentu dla zapytania na podstawie iloczynu skalarnego wektorów zapytania i dokumentu.

Algorytm Rocchia<sup>10</sup> to oparty na modelu przestrzeni wektorowej dobrze znany algorytm obliczania adekwatności informacji zwrotnych, w którym początkowy wektor zapytania i jego wagi są modyfikowane w reakcji na adekwatne dokumenty wskazane przez użytkownika. Algorytm ten zmienia pierwotny wektor zapytania  $q$  w nowy wektor  $q_e$  w następujący sposób:

$$q_e = \alpha q + \frac{\beta}{|D_r|} \sum_{d_r \in D_r} d_r - \frac{\gamma}{|D_{nr}|} \sum_{d_{nr} \in D_{nr}} d_{nr}$$

Tu  $D_r$  oznacza dokumenty adekwatne, a  $D_{nr}$  — dokumenty nieadekwatne. Wyrażenia z dokumentów adekwatnych i nieadekwatnych są dodawane do pierwotnego wektora zapytania z wagami dodatnimi i ujemnymi, aby uzyskać zmodyfikowany wektor zapytania. Parametrami równania są  $\alpha$ ,  $\beta$  i  $\gamma$ . Sumowanie  $d_r$  reprezentuje sumowanie wszystkich adekwatnych wyrażen z dokumentu  $d_r$ . Sumowanie  $d_{nr}$  oznacza sumowanie wszystkich nieadekwatnych wyrażen z dokumentu  $d_{nr}$ . Wartości podanych parametrów określają, jak informacje zwrotne wpływają na pierwotne zapytanie. Parametry te mogą zostać ustalone na podstawie eksperymentów przeprowadzanych metodą prób i błędów.

### 27.2.3. Model probabilistyczny

Pomiary podobieństwa w modelu przestrzeni wektorowej są przeprowadzane w uproszczony sposób. Przykładowo, w modelu zakłada się, że dokumenty bliższe zapytaniu według odległości kosinusowej są bardziej adekwatne względem wektora zapytania. W modelu probabilistycznym stosowane jest bardziej konkretne i definitywne podejście. Ranking dokumentów jest tworzony na podstawie szacowanego prawdopodobieństwa adekwatności

---

<sup>10</sup> Patrz Rocchio (1971).



z uwzględnieniem zapytania i dokumentu. Jest to podstawa opracowanej przez Robertsona<sup>11</sup> *zasady probabilistycznego tworzenia rankingu*.

W modelu probabilistycznym system WI musi zdecydować, czy dokumenty należą do **zbioru adekwatnych** czy **zbioru nieadekwatnych** dokumentów dla danego zapytania. Aby podjąć tę decyzję, zakłada się, że istnieją wstępnie zdefiniowane zbiory dokumentów adekwatnych i nieadekwatnych, a zadanie polega na obliczeniu prawdopodobieństwa, że dany dokument należy do pierwszego z nich, i porównaniu tej wartości z prawdopodobieństwem przynależności tego dokumentu do drugiego zbioru.

Gdy dana jest reprezentacja dokumentu,  $D$ , oszacowanie jego adekwatności ( $R$ ) i nieadekwatności ( $NR$ ) wymaga obliczenia warunkowego prawdopodobieństwa  $P(R|D)$  i  $P(NR|D)$ . To warunkowe prawdopodobieństwo można obliczyć ze wzoru Bayesa<sup>12</sup>:

$$P(R|D) = P(D|R) \times P(R)/P(D)$$

$$P(NR|D) = P(D|NR) \times P(NR)/P(D)$$

Dokument  $D$  jest uznawany za adekwatny, jeśli  $P(R|D) > P(NR|D)$ . Jeśli pominąć stałą  $P(D)$ , jest to odpowiednik stwierdzenia, że dokument jest adekwatny, gdy:

$$P(D|R) \times P(R) > P(D|NR) \times P(NR)$$

Stosunek prawdopodobieństwa  $P(D|R)/P(D|NR)$  służy do oceny prawdopodobieństwa tego, że dokument o reprezentacji  $D$  należy do zbioru adekwatnych dokumentów.

Do oszacowania  $P(D|R)$  na podstawie obliczeń  $P(t_i|R)$  dla wyrażenia  $t_i$  przyjmuje się założenie *niezależności wyrażań* (*naiwne założenie Bayesa*). Stosunki prawdopodobieństwa  $P(D|R)/P(D|NR)$  dla dokumentów są używane jako przybliżenie do tworzenia rankingów wyników. Odbywa się to przy założeniu, że dokumenty z wysokich pozycji z wysokim prawdopodobieństwem należą do zbioru adekwatnych dokumentów<sup>13</sup>.

Popularnym probabilistycznym algorytmem rankingowym jest **BM25** (ang. *Best Match 25*), w którym przyjmowane są rozsądne założenia i szacunki co do modelu probabilistycznego oraz stosowane są rozszerzenia uwzględniające w modelu wagi wyrażań z zapytań i dokumentów. Schemat wyznaczania wag wyewoluował na podstawie kilku wersji systemu **Okapi**<sup>14</sup>.

W systemie Okapi waga dokumentu  $d_j$  i zapytania  $q$  jest obliczana na podstawie podanego dalej wzoru. Oto stosowane w nim symbole:

- $t_i$  to wyrażenie;
- $f_{ij}$  to liczba wystąpień wyrażenia  $t_i$  w dokumencie  $d_j$ ;
- $f_{iq}$  to liczba wystąpień wyrażenia  $t_i$  w zapytaniu  $q$ ;
- $N$  to łączna liczba dokumentów w kolekcji;
- $d_{fi}$  to liczba dokumentów zawierających wyrażenie  $t_i$ ;

<sup>11</sup> Opis systemu Cheshire II znajdziesz w pracy Robertsona (1997).

<sup>12</sup> Twierdzenie Bayesa to standardowa technika pomiaru prawdopodobieństwa; patrz np. Howson i Urbach (1993).

<sup>13</sup> Szczegółowy opis tej kwestii znajdziesz w pracy Crofta i in. (2009) na s. 246 – 247.

<sup>14</sup> System Okapi, City University of London; patrz Robertson, Walker i Hancock-Beaulieu (1995).

- $dl_j$  to długość (w bajtach) dokumentu  $d_j$ ;
- $avdl$  to średnia długość dokumentów w kolekcji.

Ocena adekwatności dokumentu  $d_j$  dla zapytania  $q$  w systemie Okapi jest wyrażona podanym niżej wzorem z parametrami  $k_1$  (od 1,0 do 2,0),  $b$  (zwykle 0,75) i  $k_2$  (od 1 do 1000):

$$okapi(d_j, q) = \sum_{i \in q, d_j} \ln \frac{N - df_i + 0,5}{df_i + 0,5} \times \frac{(k_1 + 1)f_{ij}}{k_1 \left( 1 - b + b \frac{dl_j}{avdl} \right) + f_{ij}} \times \frac{(k_2 + 1)f_{iq}}{k_2 + f_{iq}}$$

## 27.2.4. Model semantyczny

Niezależnie od tego, jak zaawansowane stały się przedstawione modele statystyczne, mogą one pominąć wiele adekwatnych dokumentów, ponieważ nie uwzględniają pełnego znaczenia zapytania użytkownika ani potrzeb związanych z informacjami. W modelu semantycznym proces dopasowywania dokumentów do zapytania odbywa się na poziomie konceptualnym i stosuje się tu dopasowywanie semantyczne zamiast dopasowywania wyrażen z indeksu (słów kluczowych). Pozwala to wyszukiwać adekwatne dokumenty powiązane w znaczący sposób z innymi dokumentami ze zbioru wynikowego — i to nawet wtedy, gdy te powiązania nie są z natury widoczne ani ujęte w statystyczny sposób.

Podejścia semantyczne obejmują różne poziomy analiz, np.: morfologiczną, składniową i semantyczną, w celu skuteczniejszego wyszukiwania dokumentów. W **analizie morfologicznej** analizowane są rdzeń i afiks w celu określenia, jakimi częściami mowy (rzeczownikami, czasownikami, przymiotnikami itd.) są dane słowa. Po analizie morfologicznej przeprowadzana jest **analiza składniowa**, w ramach której parsowane i analizowane są kompletne zdania z dokumentu. W ostatnim kroku metody semantyczne muszą rozwiązać niejednoznaczności i/lub wygenerować adekwatne synonimy na podstawie **relacji semantycznych** między jednostkami dokumentu z różnych poziomów strukturalnych (słowami, akapitami, stronami i całymi dokumentami).

Opracowanie zaawansowanych systemów semantycznych wymaga złożonych baz wiedzy z informacjami semantycznymi, a także heurystyk wyszukiwania. Takie systemy często wymagają technik z obszarów sztucznej inteligencji i systemów eksperckich. Do użytku w *systemach WI opartych na wiedzy* wykorzystujących modele semantyczne opracowano bazy wiedzy takie jak Cyc<sup>15</sup> i WordNet<sup>16</sup>. Baza wiedzy Cyc to reprezentacja obszernej wiedzy zdroworozsądkowej. Obecnie obejmuje 15,94 miliona asercji, 498 271 pojęć atomowych oraz 441 159 pochodnych pojęć nieatomowych służących do wnioskowania na temat obiektów i zdarzeń życia codziennego. WordNet to obszerny tezaurus (obejmujący ponad 117 000 pojęć). Jest on bardzo popularny i wykorzystywany w wielu systemach. Stale się go rozwija (patrz punkt 27.4.3).

<sup>15</sup> Patrz Lenat (1995).

<sup>16</sup> Szczegółowy opis tezaury WordNet znajdziesz w pracy Millera (1990).

## 27.3. Typy zapytań w systemach WI

W procesie indeksowania z dokumentem wiązane są różne słowa kluczowe. Zwykle obejmują one słowa, frazy i inne cechy dokumentów, np.: datę utworzenia, nazwiska autorów i rodzaj dokumentu. Są one używane w systemie WI do budowania indeksów odwróconych (patrz podrozdział 27.5), używanych później w trakcie wyszukiwania. Zapytania formułowane przez użytkowników są porównywane ze zbiorem słów kluczowych z indeksu. Większość systemów WI umożliwia też stosowanie operatorów logicznych i innych do budowania złożonych zapytań. Język zapytań obejmujący takie operatory wzbogaca możliwości określania potrzeb związanych z informacjami.

### 27.3.1. Zapytania oparte na słowach kluczowych

Zapytania oparte na słowach kluczowych to najprostsza i najczęściej stosowana forma zapytań w systemach WI. Użytkownik wpisuje tylko zestaw słów kluczowych, aby wyszukać dokumenty. Wpisane wyrażenia są automatycznie łączone logicznym operatorem I. Zapytanie takie jak „zagadnienia bazodanowe” na początku listy wyszukanych wyników zwraca dokumenty zawierające oba słowa: „zagadnienia” i „bazodanowe”. Ponadto większość systemów wyszukuje też dokumenty zawierające w tekście tylko jedno z tych słów („zagadnienia” lub „bazodanowe”). Niektóre systemy na etapie wstępnego przetwarzania usuwają najczęściej występujące słowa (takie jak *i*, *a* itd.; są to tzw. **słowa pomijane**) i przesyłają do silnika WI przefiltrowane słowa kluczowe z zapytania. Większość systemów WI nie uwzględnia kolejności występowania słów w zapytaniu. Wszystkie modele wyszukiwania obsługują zapytania oparte na słowach kluczowych.

### 27.3.2. Zapytania logiczne

Niektóre systemy WI umożliwiają stosowanie razem ze słowami kluczowymi operatorów I, LUB, NIE, (), + i -. Operator I wymaga znalezienia obu słów, LUB dopuszcza znalezienie dowolnego ze słów, NIE oznacza, że wszystkie rekordy obejmujące podane po nim słowo zostaną odrzucone, () powoduje zagnieżdżenie operatorów logicznych za pomocą nawiasu, + to odpowiednik I (wymaga podania wyrażenia i musi zostać umieszczony bezpośrednio przed szukanym zwrotem), - to odpowiednik I NIE (powoduje wykluczenie dokumentów z danym wyrażeniem i musi znajdować się bezpośrednio przed niepożądanym zwrotem). Za pomocą tych operatorów i ich kombinacji można budować złożone zapytania logiczne. Są one przetwarzane zgodnie z klasycznymi regułami algebry boolowskiej. Porządkowanie wyników jest tu niemożliwe, ponieważ dokument albo jest zgodny z zapytaniem (adekwatny), albo niezgodny (nieadekwatny). Dokument jest pobierany na podstawie zapytania logicznego, jeśli to zapytanie zwraca wartość logiczną „prawda” oznaczającą dokładne dopasowanie dokumentu. Użytkownicy zwykle nie stosują kombinacji złożonych operatorów logicznych, a systemy WI obsługują ograniczoną wersję takich operatorów. Modele wyszukiwania logicznego potrafią bezpośrednio obsługiwać różne implementacje operatorów logicznych z opisywanych zapytań.

### 27.3.3. Zapytania do wyszukiwania fraz

Gdy dokumenty są na potrzeby wyszukiwania reprezentowane za pomocą odwróconego indeksu słów kluczowych, względna kolejność pojęć w dokumencie zostaje utracona. Aby przeprowadzić wyszukiwanie dokładnych fraz, należy je zapisać w indeksie odwróconym lub uwzględnić w inny sposób (na podstawie względnych pozycji wystąpień słów w dokumencie). Zapytanie do wyszukiwania fraz składa się z serii słów tworzących tę frazę. Zwykle taka fraza znajduje się między cudzysłowami. Każdy zwracany dokument musi obejmować przynajmniej jedno wystąpienie danej frazy. Wyszukiwanie fraz jest bardziej restrykcyjną i specyficzną odmianą opisanego dalej wyszukiwania z określeniem odległości słów. Przykładowym zapytaniem może być tu „konceptyjne projektowanie baz danych”. Jeśli w danym modelu wyszukiwania frazy są indeksowane, tego rodzaju zapytania można obsłużyć za pomocą dowolnego modelu. W modelach semantycznych można też wykorzystać tezaurs fraz do szybkiego słownikowego ich wyszukiwania.

### 27.3.4. Zapytania z określaniem odległości słów

Wyszukiwanie z określaniem odległości słów wymaga uwzględnienia, jak blisko szukane pojęcia powinny znajdować się w rekordzie. Najczęściej stosowaną odmianą tego podejścia jest wyszukiwanie fraz, gdzie pojęcia muszą występować w ściśle określonej kolejności. Inne operatory odległości pozwalają określić, jak blisko siebie pojęcia mają się znajdować. Niektóre operatory wyznaczają też kolejność szukanych pojęć. W każdej wyszukiwarce operatory odległości mogą być zdefiniowane inaczej. Ponadto w wyszukiwarkach używane są różne nazwy operatorów, np.: NEAR, ADJ lub AFTER. Czasem podawana jest seria pojedynczych słów wraz z dozwoloną maksymalną odległością między nimi. Modele oparte na przestrzeni wektorowej, przechowujące także informacje na temat pozycji i odległości tokenów (słów), umożliwiają solidne zaimplementowanie zapytań tego typu. Jednak zapewnianie obsługi złożonych operatorów odległości jest obliczeniowo kosztowne, ponieważ wymaga czasochłonnego wstępnego przetwarzania dokumentów, dlatego lepiej nadaje się dla mniejszych kolekcji dokumentów niż dla sieci WWW.

### 27.3.5. Zapytania z symbolami wieloznacznymi

Wyszukiwanie z symbolami wieloznacznymi zwykle ma zapewniać obsługę wyrażeń regularnych i przeszukiwania tekstu z użyciem dopasowywania do wzorca. W systemach WI zaimplementowane mogą być pewne rodzaje wyszukiwania z użyciem symboli wieloznacznych — zwykle wyszukiwanie słów z dowolnymi znakami końcowymi (np. wyrażenie ‘filtr\*’ pasuje do słów *filtry*, *filtrowanie*, *filtrować* itd.). Zapewnienie pełnej obsługi wyszukiwania z użyciem symboli wieloznacznych w wyszukiwarkach internetowych wymaga poniesienia kosztów wstępnego przetwarzania i w wielu współczesnych wyszukiwarkach ta technika nie jest dostępna<sup>17</sup>. Modele wyszukiwania nie zapewniają bezpośredniej obsługi zapytań tego typu. Lucene<sup>18</sup> uwzględnia niektóre rodzaje zapytań z użyciem symboli wieloznacznych. Parser zapytań w systemie Lucene przetwarza rozbudowane zapytanie logiczne uwzględniające wszystkie kombinacje i rozwinięcia słów z indeksu.

<sup>17</sup> Więcej szczegółów znajdziesz na stronie [http://www.livinginternet.com/w/wu\\_expert\\_wild.htm](http://www.livinginternet.com/w/wu_expert_wild.htm).

<sup>18</sup> Patrz <http://lucene.apache.org/>.

### 27.3.6. Zapytania w języku naturalnym

Istnieją wyszukiwarki dla języka naturalnego, które próbują zrozumieć strukturę i znaczenie zapytań napisanych w języku naturalnym (zwykle w postaci pytań lub zwykłego tekstu). Jest to obszar, w którym aktywnie prowadzone są badania. Wykorzystuje się tu techniki takie jak płytkie semantyczne parsowanie tekstu lub przeformułowywanie zapytań na bazie zrozumienia języka naturalnego. System próbuje na podstawie znalezionych wyników formułować odpowiedzi na takie zapytania. Niektóre systemy wyszukiwania zaczynają udostępniać języki z interfejsem w języku naturalnym, aby udzielały odpowiedzi na pytania konkretnego rodzaju — np. o definicje i fakty, wymagające podania definicji pojęć technicznych lub faktów, które można pobrać z wyspecjalizowanych baz danych. Zwykle udzielenie odpowiedzi na takie pytania jest łatwiejsze, ponieważ występują wyraźne wzorce lingwistyczne, które stanowią wskazówkę co do zdań określonego typu (np. „zdefiniowane jako” lub „dotyczy”). Obsługę zapytań tego typu mogą zapewniać modele semantyczne.

## 27.4. Wstępne przetwarzanie tekstu

W tym podrozdziale opiszemy często stosowane techniki wstępnego przetwarzania tekstu, będące częścią etapu przetwarzania tekstu z rysunku 27.1.

### 27.4.1. Usuwanie słów pomijanych

**Słowa pomijane** to wyrazy bardzo często używane w języku i odgrywające ważną rolę w budowaniu zdań, ale rzadko wpływające na znaczenie tych zdań. Wyrazy, które występują w przynajmniej 80% dokumentów z kolekcji, są zwykle nazywane *słowami pomijanymi* i uznawane za potencjalnie nieprzydatne. Z powodu powszechności występowania i funkcji tych słów nie wpływają one w znacznym stopniu na adekwatność dokumentu względem zapytania. Przykłady takich słów to: *i, w, się, też, dla, a, lub, oraz, albo, cię i nie*.

Słowa pomijane należy usunąć z dokumentu przed indeksowaniem. Jako takie słowa uznaje się zwykle rodzajniki, przyimki, spójniki i zaimki. Zapytania przed rozpoczęciem właściwego procesu wyszukiwania wymagają wstępnego przetworzenia w celu usunięcia słów pomijanych. Usunięcie takich słów prowadzi do wyeliminowania potencjalnie kłopotliwych indeksów i zmniejszenia wielkości indeksu o ok. 40% lub więcej. Jednak operacja ta może wpływać na zwracane dane, jeśli słowa pomijane są integralną częścią zapytania. Przykładowo, jeśli szukane jest wyrażenie „Być albo nie być”, usunięcie słów pomijanych spowoduje, że zapytanie stanie się bezsensowne, ponieważ wszystkie wyrazy są tu słowami pomijanymi. Dlatego w wielu wyszukiwarkach słowa pomijane nie są usuwane.

### 27.4.2. Stemming

**Temat** słowa jest zdefiniowany jako słowo otrzymane po odcięciu przedrostka i przyrostka pierwotnego słowa. Na przykład „obraz” to temat wyrazów *podobrazie, obrazować, obrazkowy i zobrazować*. Różne przyrostki i przedrostki są bardzo częste w języku polskim i służą np. do układania czasowników, odmiany przez przypadki i tworzenia liczby mnogiej.

**Stemming** polega na ograniczaniu różnych postaci słowa utworzonych w wyniku odmiany i sprowadzaniu ich do wspólnego tematu.

Można zastosować algorytm stemmingu, aby sprowadzić dowolne słowo do jego tematu. W języku angielskim najbardziej znanym algorytmem stemmingu jest algorytm Martina Portera. Stemmer Portera<sup>19</sup> to uproszczona wersja techniki Lovina, używająca ograniczonego zestawu ok. 60 reguł (spośród 260 wzorców przyrostków z techniki Lovina) i porządkująca je w zbiory. Konflikty powstałe w ramach jednego podzbioru reguł są eliminowane przed przejściem do następnego. Użycie stemmingu do wstępnego przetwarzania danych skutkuje zmniejszeniem indeksu i zwiększeniem list wyników (czasem kosztem precyzji).

### 27.4.3. Korzystanie z tezaurya

**Tezaurus** składa się ze wstępnie przygotowanej listy ważnych pojęć i głównych słów, które opisują każde pojęcie z określonej dziedziny wiedzy. Dla każdego pojęcia z listy tworzony jest też zbiór synonimów i powiązanych słów<sup>20</sup>. Tak więc w trakcie wstępnego przetwarzania synonim można zastąpić odpowiadającym mu pojęciem. Ten etap wstępnego przetwarzania pozwala łatwiej utworzyć standardowy leksykon na potrzeby indeksowania i wyszukiwania. Korzystanie z tezaurya, nazywanego też *zbiorem synonimów*, ma duży wpływ na czułość systemów WI. Proces ten może być skomplikowany, ponieważ wiele słów może w poszczególnych kontekstach mieć inne znaczenie.

**UMLS**<sup>21</sup> to obszerny tezaurus biomedyczny obejmujący miliony pojęć (*metatezaurus*) oraz sieć semantyczną metapojęć i relacji wyznaczających strukturę metatezaurya (patrz rysunek 27.3). Pojęciom są przypisywane etykiety z sieci semantycznej. Ten tezaurus pojęć zawiera synonimy nazw medycznych, hierarchie bardziej ogólnych i bardziej precyzyjnych określeń, a także inne relacje między słowami i pojęciami, dzięki czemu jest bardzo rozbudowanym źródłem do wyszukiwania informacji w dokumentach medycznych. Na rysunku 27.3 pokazano fragment sieci semantycznej z tezaurya UMLS.

**WordNet**<sup>22</sup> to ręcznie opracowany tezaurus grupujący słowa w zbiory ścisłych synonimów (ang. *synsets*). Zbiory synonimów są podzielone na rzeczowniki, czasowniki, przymiotniki i przysłówki. W każdej z tych kategorii zbiory synonimów są powiązane odpowiednimi związkami, takimi jak klasa-podklasa lub związek „jest-czymś” w przypadku rzeczowników.

WordNet jest oparty na pomyśle używania w indeksach kontrolowanego leksykonu i eliminowania w ten sposób nadmiarowości. Pomaga też użytkownikom w określaniu wyrażen pozwalających tworzyć poprawne zapytania.

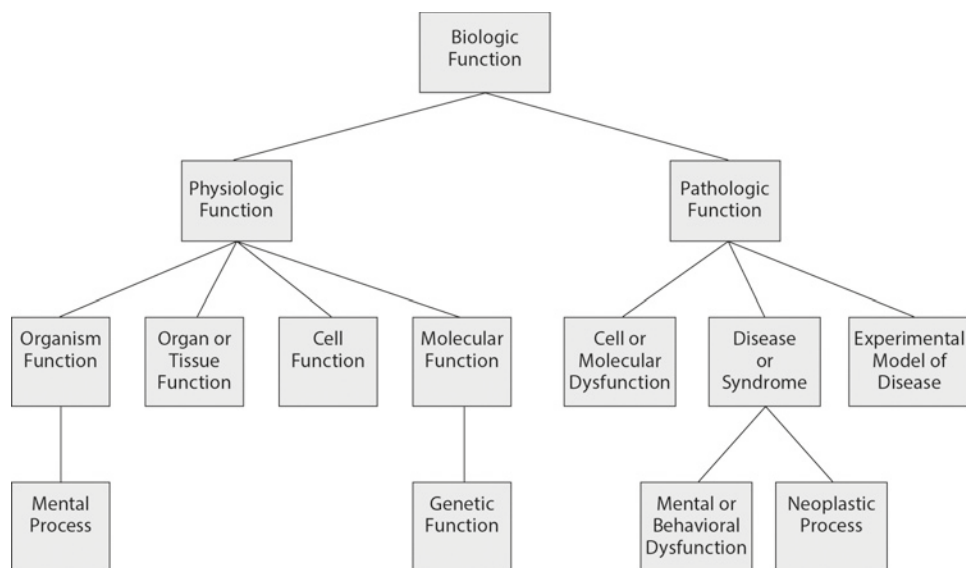
---

<sup>19</sup> Patrz Porter (1980).

<sup>20</sup> Patrz Baeza-Yates i Ribeiro-Neto (1999).

<sup>21</sup> Unified Medical Language System z National Library of Medicine.

<sup>22</sup> Szczegółowe omówienie tezaurya WordNet znajdziesz w pracy Fellbauma (1998).



RYSUNEK 27.3. Fragment sieci semantycznej z tezaursu UMLS — hierarchia wyrażenia „Biologic Function” Źródło: *UMLS Reference Manual*, National Library of Medicine.

## 27.4.4. Inne etapy przetwarzania: cyfry, myślniki, znaki przestankowe, wielkość znaków

Cyfry, daty, numery telefonów, adresy e-mail, adresy URL i inne standardowe typy tekstu mogą, ale nie muszą być usuwane w trakcie wstępnego przetwarzania. Wyszukiwarki internetowe indeksują jednak takie dane, aby wykorzystać je w metadanych dokumentów w celu poprawy precyzji i czułości (szczegółowe definicje *precyzji* i *czułości* znajdziesz w podrozdziale 27.6).

Myślniki i znaki przestankowe mogą być traktowane w różny sposób. Można wykorzystać albo całe wyrażenie z myślnikami i znakami przestankowymi, albo usunąć takie symbole. W niektórych systemach symbole reprezentujące myślniki i znaki przestankowe są usuwane lub zastępowane spacjami. W poszczególnych systemach WI stosowane są inne reguły przetwarzania. Automatyczne przetwarzanie myślników może być trudne. Można potraktować to zadanie jak problem z dziedziny klasyfikacji lub zastosować reguły heurystyczne. Przykładowo, mechanizm StandardTokenizer w systemie Lucene<sup>23</sup> traktuje myślnik jak ogranicznik dzielący słowa. Wyjątkiem jest sytuacja, gdy w tokenie występuje liczba; wtedy słowa nie są rozdzielane (np. AK-47, numery telefonów). Wiele określeń specyficznych dla dziedziny, np. nazwy w katalogu produktów lub numery wersji produktu, obejmuje myślniki. Gdy wyszukiwarka przeszukuje sieć WWW na potrzeby indeksowania, trudno jest w automatyczny sposób poprawnie traktować myślniki. Dlatego rozwijane są prostsze strategie ich przetwarzania.

<sup>23</sup> Więcej informacji o narzędziu StandardTokenizer zawiera strona <https://lucene.apache.org/>.



Większość systemów WI nie uwzględnia wielkości znaków w trakcie wyszukiwania i przekształca wszystkie litery w tekście na wielkie lub małe. Warto też zauważyć, że wiele etapów wstępnego przetwarzania tekstu jest specyficznych dla języka. Dotyczy to np. akcentów i znaków diakrytycznych oraz nietypowych rozwiązań z konkretnych języków.

### 27.4.5. Wydobywanie informacji

**Wydobywanie informacji** (ang. *information extraction*) to ogólne pojęcie dotyczące wydobywania strukturalnych treści z tekstu. Przykładowe zadania z tej dziedziny to identyfikowanie rzeczowników, faktów, zdarzeń, ludzi, miejsc i związków. Takie operacje są też nazywane *rozpoznawaniem nazwanych jednostek* i wykorzystuje się w nich podejścia oparte na regułach z użyciem tezaurusów, wyrażeń regularnych, gramatyk lub technik probabilistycznych. W kontekście WI oraz wyszukiwarek technologie wydobywania informacji najczęściej służą do identyfikowania nazwanych jednostek i wykorzystują analizy, dopasowywanie i kategoryzację tekstu, aby zwiększyć adekwatność systemu wyszukiwania. Technologie określające części mowy są stosowane w celu semantycznego opisywania dokumentów za pomocą wydobytych właściwości, co pozwala zwiększyć adekwatność wyszukiwania.

## 27.5. Indeksy odwrócone

Najprostszy sposób wyszukiwania wystąpień wyrażeń z zapytania w kolekcji tekstu polega na sekwencyjnym skanowaniu tekstu. Tego rodzaju wyszukiwanie jest odpowiednie tylko dla małych kolekcji tekstu. Większość systemów WI przetwarza kolekcje tekstu w celu utworzenia indeksów i operuje na strukturze danych w postaci indeksu odwróconego (zadanie indeksowania na rysunku 27.1). Struktura indeksu odwróconego obejmuje informacje o leksykonie i dokumencie. **Leksykon** (ang. *vocabulary*) to zbiór różnych pojęć występujących w zbiorze dokumentów i używanych w zapytaniach. Każde takie pojęcie jest powiązane z informacjami na temat zawierających je dokumentów, np.: identyfikatorami tych dokumentów, liczbą wystąpień i pozycjami tych wystąpień w dokumentach. W najprostszej postaci pojęcia z leksykonu są słowami lub tokenami dokumentów. W niektórych sytuacjach pojęcia z leksykonu to frazy, *n*-gramy, encje, odsyłacze, nazwy, daty lub ręcznie określone deskryptory z dokumentów i/lub stron WWW. Dla każdego pojęcia z leksykonu można w sekcji z informacjami o dokumentach zapisać identyfikatory dokumentów, lokalizacje wystąpień w każdym dokumencie i inne ważne dane.

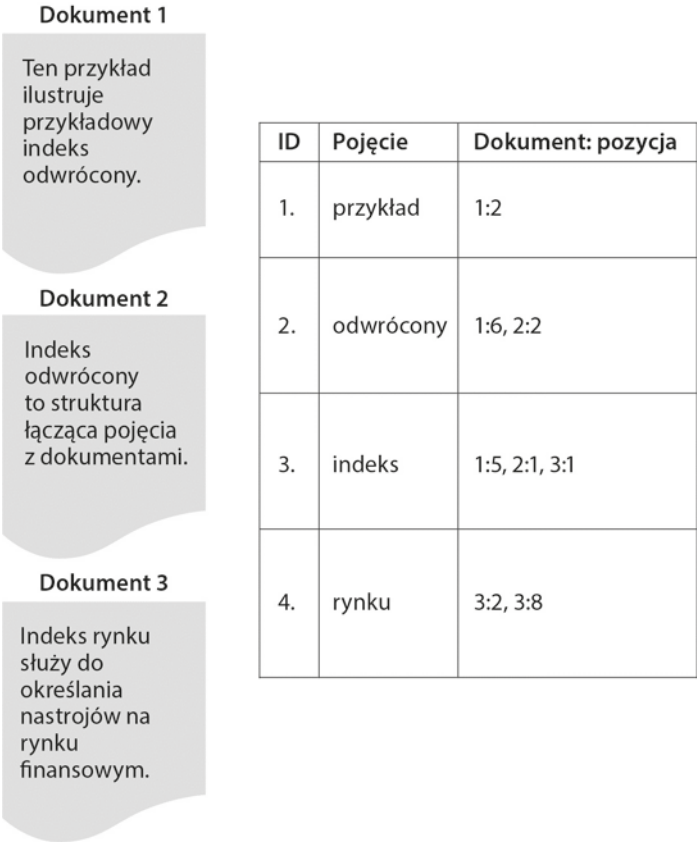
Do pojęć z dokumentu przypisywane są wagi reprezentujące szacowaną przydatność danego pojęcia jako deskryptora odróżniającego określony dokument od innych z tej samej kolekcji. Dzięki wagom pojęcie może być lepszym deskryptorem jednego dokumentu niż innych (patrz podrozdział 27.2).

**Indeks odwrócony** dla kolekcji dokumentów to struktura danych, która łączy różne pojęcia z listą wszystkich dokumentów zawierających dane pojęcia. Proces tworzenia indeksu odwróconego obejmuje etapy wydobywania i przetwarzania z rysunku 27.2. Pobrany tekst jest najpierw wstępnie przetwarzany, a dokumenty są reprezentowane za pomocą pojęć z leksykonu. W tabelach wyszukiwania dokumentów zapisywane są dotyczące tych dokumentów statystyki. Zwykle obejmują one liczbę konkretnych pojęć

w poszczególnych dokumentach i ich zbiorach, pozycje wystąpień pojęć w dokumentach i długość dokumentów. W trakcie indeksowania pojęciom przypisywane są wagi zależne od różnych kryteriów używanych dla kolekcji. Przykładowo, czasem pojęcia z tytułów dokumentów mają większe wagi niż pojęcia z innych części dokumentów.

Jednym z najpopularniejszych schematów określania wag jest opisana w podrozdziale 27.2 miara TF-IDF. Dla danego pojęcia ten schemat wag określa w pewnym zakresie dokumenty, w których to pojęcie występuje często, od dokumentów z małą lub zerową liczbą wystąpień tego pojęcia. Wagi są normalizowane, aby uwzględnić różną długość dokumentów. To zapewnia, że dłuższe dokumenty (z proporcjonalnie większą liczbą wystąpień danego słowa) nie są faworyzowane przy wyszukiwaniu w porównaniu z krótszymi dokumentami (z proporcjonalnie mniejszą liczbą wystąpień pojęcia). Przetworzone strumienie (tabele) dokumenty-pojęcia są następnie odwracane do postaci strumieni (tabel) pojęcia-dokumenty na potrzeby dalszych kroków WI.

Na rysunku 27.4 pokazano wektory pozycji pojęć w dokumentach dla czterech przykładowych słów — *przykład*, *odwrócony*, *indeks* i *rynek*. Obrazuje to pozycje wystąpień każdego pojęcia w trzech dokumentach.



RYСУNEK 27.4. Przykładowy indeks odwrócony

Oto podsumowanie kroków tworzenia indeksu odwróconego:

- (1) Podział dokumentu na pojęcia z leksykonu w wyniku podziału na tokeny, oczyszczania, usuwania słów pomijanych, stemmingu i/lub zastosowania dodatkowego tezaursusa jako leksykonu.
- (2) Zbieranie statystyk dotyczących dokumentów i zapisywanie ich w tabeli wyszukiwania dokumentów.
- (3) Odwracanie strumienia dokumenty-pojęcia w celu uzyskania strumienia pojęcia-dokumenty wraz z dodatkowymi informacjami, takimi jak częstotliwość występowania pojęć, pozycje pojęć i wagi pojęć.

Wyszukiwanie dla zbioru pojęć z zapytania adekwatnych dokumentów na podstawie indeksu odwróconego to zwykle proces trzyetapowy:

- (1) **Przeszukiwanie leksykonu.** Jeśli zapytanie obejmuje wiele pojęć, są one oddzielane i traktowane niezależnie. Każde pojęcie jest wyszukiwane w leksykonie. Do zoptymalizowania procesu wyszukiwania można wykorzystać różne struktury danych (takie jak odmiany B+-drzew) lub mieszanie. Pojęcia z zapytania można też uporządkować leksykograficznie, aby zmniejszyć ilość zajmowanej pamięci.
- (2) **Wyszukiwanie informacji o dokumentach.** Wyszukiwane są informacje o dokumentach powiązane z każdym pojęciem.
- (3) **Operowanie na znalezionych informacjach.** Wektor z informacjami o dokumencie uzyskany w kroku 2. jest teraz dodatkowo przetwarzany, aby uwzględnić różne formy logiki zapytań. W tym kroku przetwarzane są różne rodzaje zapytań (prefiksowe, zakresowe, kontekstowe i z określaniem odległości słów), aby wygenerować wynik końcowy na podstawie zbioru dokumentów zwróconego w kroku 2.

## 27.5.1. Wprowadzenie do systemu Lucene

Lucene to aktywnie rozwijany otwarty system indeksowania i wyszukiwania, który zyskał popularność zarówno w środowisku akademickim, jak i w zastosowaniach komercyjnych. Służy przede wszystkim do indeksowania, ale wykorzystuje indeksy do wspomagania wyszukiwania. Biblioteka systemu Lucene jest napisana w Javie i automatycznie zapewnia skalowalność i wysoką wydajność. Lucene to system będący podstawą innego bardzo popularnego w firmach narzędzia do wyszukiwania, Solr<sup>24</sup>. Solr wzbogaca system Lucene o wiele dodatkowych możliwości. Przykładowo, udostępnia interfejsy internetowe pozwalające indeksować dokumenty w wielu różnych formatach.

W czekającej na publikację pracy Moczara (2015) opisano zarówno system Lucene, jak i system Solr.

**Indeksowanie.** W systemie Lucene dokumenty muszą przejść przez proces indeksowania, zanim będzie możliwe ich przeszukiwanie. Dokument w systemie Lucene składa się ze zbioru pól. Pola mają określony typ danych, mogą być zapisywane w indeksie i są nieco podobne do kolumn z tabel bazy danych. Pole może zawierać dane binarne, liczbowe lub tekstowe.

---

<sup>24</sup> Patrz <http://lucene.apache.org/solr/>.

Pola tekstowe obejmują albo całe fragmenty nieprzetworzonego tekstu, albo sekwencje przetworzonych jednostek leksykalnych (strumienie tokenów). Strumienie tokenów są generowane za pomocą różnego rodzaju algorytmów tokenizacji i filtrowania. Przykładowo, `StandardTokenizer` to jeden z dostępnych w Lucene mechanizmów tokenizacji, wykorzystujący do podziału tekstu na słowa segmentację opartą na standardzie Unicode. Istnieją też inne mechanizmy tokenizacji; np. `WhitespaceTokenizer` dzieli tekst na podstawie spacji. W Lucene łatwo jest rozszerzyć mechanizmy tokenizacji i filtry, aby opracować niestandardowe algorytmy analizy tekstu na potrzeby tokenizacji i filtrowania. Te algorytmy są bardzo ważne do osiągnięcia pożądaných wyników wyszukiwania. Lucene udostępnia interfejsy API i różne implementacje wielu szybkich oraz wydajnych algorytmów tokenizacji i filtrowania. Te algorytmy zostały rozszerzone pod kątem różnych języków i dziedzin oraz obejmują implementacje algorytmów przetwarzania języka naturalnego, np.: algorytmów stemmingu, lematyzacji na podstawie słownika, analizy morfologicznej, analizy fonetycznej.

**Wyszukiwanie.** Dzięki rozbudowanemu interfejsowi API wyszukiwania zapytania są wykonywane na podstawie dokumentów, po czym zwracana jest lista uporządkowanych wyników. Zapytania są porównywane z wektorami pojęć z indeksu odwróconego, aby uzyskać ocenę adekwatności na podstawie modelu przestrzeni wektorowej (patrz punkt 27.2.2). Lucene udostępnia wysoce konfigurowalny interfejs API wyszukiwania, w którym można tworzyć zapytania z symbolami wieloznacznymi, zapytania logiczne, zapytania z określaniem odległości słów i zapytania zakresowe. W Lucene domyślny algorytm oceny wyników tworzy ich ranking za pomocą odmian techniki TF-IDF. Aby przyspieszyć wyszukiwanie, Lucene przechowuje zależne od dokumentów współczynniki normalizacji wstępnie generowane w czasie indeksowania. Te współczynniki są nazywane normami wektorów pojęć dla pól dokumentów. Wstępnie obliczone normy przyspieszają proces oceny wyników w Lucene. Algorytmy dopasowywania zapytań wywołują funkcje, które w czasie przetwarzania zapytania wykonują bardzo niewiele obliczeń.

**Zastosowania.** Jednym z powodów wielkiej popularności systemu Lucene jest łatwość stosowania go do obsługi różnych zbiorów dokumentów i systemów w kontekście indeksowania dużych zbiorów dokumentów niestukturalnych. Na podstawie Lucene zbudowana jest przeznaczona dla firm wyszukiwarka Solr. Solr to internetowa aplikacja serwerowa zapewniająca obsługę wyszukiwania fasetowego (ang. *faceted search*; patrz poświęcony temu zagadnieniu punkt 27.8.1) i przetwarzania dokumentów w niestandardowych formatach (takich jak PDF, HTML itd.) oraz usługi sieciowe związane z różnymi funkcjami API służącymi do indeksowania i wyszukiwania w systemie Lucene.

## 27.6. Miary oceny adekwatności wyników wyszukiwania





Bez odpowiednich technik oceny nie da się porównać i zmierzyć adekwatności różnych modeli wyszukiwania i systemów WI w celu wprowadzenia usprawnień. Techniki oceny w systemach WI mierzą *adekwatność tematyczną* i *adekwatność dla użytkownika*. **Adekwatność tematyczna** mierzy, w jakim stopniu temat wyniku pasuje do tematu zapytania. Przełożenie potrzeb z zakresu informacji na „perfekcyjne” zapytania to zadanie umysłowe, a wielu użytkowników nie potrafi skutecznie formułować zapytań pozwalających znaleźć wyniki lepiej dopasowane do potrzeb. Ponadto ponieważ duża część zapytań użytkowników ma charak-

ter informacyjny, nie istnieje stały zestaw właściwych odpowiedzi, jakie należy wyświetlić użytkownikowi. **Adekwatność dla użytkownika** opisuje jakość znalezionych wyników na podstawie potrzeb użytkownika. Uwzględniane są tu pośrednie czynniki takie jak percepcja użytkownika, kontekst, szybkość, środowisko użytkownika i potrzeby związane z wykonywanym zadaniem. Ocena adekwatności dla użytkownika może też obejmować analizy subiektywne i badanie wykonywanych przez użytkownika zadań wyszukiwania. Pozwala to uwzględnić niektóre aspekty pośrednich czynników związanych z tendencyjnością użytkowników w trakcie oceny wyników.

Wyszukiwanie informacji w sieci WWW nie obejmuje zerojedynkowej oceny, czy dokument jest adekwatny, czy nieadekwatny do zapytania (w logicznym modelu wyszukiwania stosuje się takie oceny; patrz punkt 27.2.1). Zamiast tego dla użytkownika generowany jest ranking dokumentów. Dlatego niektóre miary oceny dotyczą porównywania różnych rankingów generowanych przez systemy WI. Niektóre z tych miar omawiamy dalej.

## 27.6.1. Czułość i precyzja

Miary czułości i precyzji są oparte na zerojedynkowej ocenie adekwatności (każdy dokument jest adekwatny lub nie dla zapytania). **Czułość** to liczba wyszukanych adekwatnych dokumentów podzielona przez łączną liczbę wszystkich adekwatnych dokumentów z bazy danych. **Precyzja** to liczba wyszukanych adekwatnych dokumentów podzielona przez łączną liczbę znalezionych dokumentów. Na rysunku 27.5 znajduje się graficzna reprezentacja określeń *wyszukane* i *adekwatne*. Pokazano też, jak wyniki wyszukiwania są związane z czterema różnymi zbiorami dokumentów.

		Adekwatne?	
		Tak	Nie
Wyszukane?	Tak	 Trafienia PP	 Fałszywe alarmy FP
	Nie	Chybienia FN 	Poprawne odrzucenia PN 

RYSUNEK 27.5. Wyszukane i adekwatne wyniki wyszukiwania

Oto notacja z rysunku 27.5:

- PP: prawdziwie pozytywne.
- FP: fałszywie pozytywne.
- FN: fałszywie negatywne.
- PN: prawdziwie negatywne.

Określenia *prawdziwie pozytywne*, *fałszywie pozytywne*, *fałszywie negatywne* i *prawdziwie negatywne* są używane w dowolnego rodzaju zadaniach klasyfikacji do porównywania ustalonej kategorii elementu z pożądaną poprawną kategorią. Używając pojęcia *trafienia* dla dokumentów, które zostały poprawnie dopasowane do żądania użytkownika, można zdefiniować *czułość* i *precyzję* w następujący sposób:

$$\text{czułość} = |\text{trafienia}|/|\text{adekwatne}|$$

$$\text{precyzja} = |\text{trafienia}|/|\text{wyszukane}|$$

Czułość i precyzję można też zdefiniować w kontekście uporządkowanych wyników wyszukiwania. Załóżmy, że każdej pozycji rankingu odpowiada jeden dokument. Czułość  $r(i)$  dla pozycji  $i$  oraz dokumentu  $d_i^q$  (to wyszukany dokument na pozycji  $i$  dla zapytania  $q$ ) to odsetek adekwatnych dokumentów na pozycjach od  $d_1^q$  do  $d_i^q$  w zbiorze wyników dla danego zapytania. Niech  $S_i$  o liczności  $|S_i|$  będzie zbiorem adekwatnych dokumentów na pozycjach od  $d_1^q$  do  $d_i^q$ . Niech  $|D_q|$  będzie wielkością zbioru adekwatnych dokumentów dla danego zapytania. W takim scenariuszu  $|S_i| \leq |D_q|$ . Wtedy:

$$\text{czułość dla wyszukiwania z rankingiem } r_i = |S_i| / |D_q|$$

Precyzja  $p_i$  dla pozycji  $i$  oraz dokumentu  $d_i^q$  to odsetek adekwatnych dokumentów na pozycjach od  $d_1^q$  do  $d_i^q$  w zbiorze wyników:

$$\text{precyzja dla wyszukiwania z rankingiem } p_i = |S_i| / i$$

W tabeli 27.2 pokazano miary  $p(i)$ ,  $r(i)$  oraz średniej precyzji (opis w następnym punkcie). Widać tu, że czułość można zwiększyć, wyświetlając użytkownikowi więcej wyników. Jednak to podejście grozi spadkiem precyzji. W przykładzie liczba dokumentów adekwatnych dla zapytania to 10. Pokazane są pozycja w rankingu i adekwatność poszczególnych dokumentów. Precyzję i czułość można obliczyć dla każdej pozycji w rankingu (patrz dwie ostatnie kolumny). W tabeli 27.2 widać, że czułość dla wyszukiwania z rankingiem rośnie monotonicznie, natomiast precyzja fluktuuje.

TABELA 27.2. Precyzja i czułość dla wyszukiwania z rankingiem

Numer dokumentu	Pozycja w rankingu (i)	Adekwatne?	Precyzja(i)	Czułość(i)
10	1	Tak	1/1 = 100%	1/10 = 10%
2	2	Tak	2/2 = 100%	2/10 = 20%
3	3	Tak	3/3 = 100%	3/10 = 30%
5	4	Nie	3/4 = 75%	3/10 = 30%
17	5	Nie	3/5 = 60%	3/10 = 30%
34	6	Nie	3/6 = 50%	3/10 = 30%
215	7	Tak	4/7 = 57,1%	4/10 = 40%
33	8	Tak	5/8 = 62,5%	5/10 = 50%
45	9	Nie	5/9 = 55,5%	5/10 = 50%
16	10	Tak	6/10 = 60%	6/10 = 60%

## 27.6.2. Średnia precyzja

Średnia precyzja jest obliczana na podstawie precyzji dla każdego adekwatnego dokumentu z rankingu. Ta miara jest przydatna do obliczania jednej wartości precyzji, co pozwala porównywać różne algorytmy wyszukiwania dla zapytania  $q$ .

$$P_{\text{śred}} = \sum_{d_i^q \in D_q} p(i) / |D_q|$$

Rozważ przykładowy poziom precyzji dla adekwatnych dokumentów z tabeli 27.2. Średnia precyzja (wartość  $P_{\text{śred}}$ ) dla przykładu z tabeli 27.2 wynosi  $P(1) + P(2) + P(3) + P(7) + P(8) + P(10) = 79,93\%$  (w tych obliczeniach uwzględniane są tylko adekwatne dokumenty). Wiele dobrych algorytmów zapewnia wysoką średnią precyzję dla pierwszych  $k$  dokumentów przy małych wartościach  $k$  i niskim poziomie czułości.

## 27.6.3. Krzywa czułość/precyzja

Krzywą czułość/precyzja można narysować na podstawie poziomu czułości i precyzji dla każdej pozycji z rankingu. Oś  $x$  reprezentuje tu czułość, a oś  $y$  — precyzję. Zamiast uwzględniać precyzję i czułość dla każdej pozycji, najczęściej rysuje się krzywą dla poziomów czułości  $r(i)$  równych 0%, 10%, 20%, ..., 100%. Ta krzywa ma zwykle ujemne nachylenie, co odzwierciedla odwrotną zależność między precyzją a czułością.

## 27.6.4. Miara F

Miara F to średnia harmoniczna wartości precyzji ( $p$ ) i czułości ( $r$ ):

$$\frac{1}{F} = \frac{\frac{1}{p} + \frac{1}{r}}{2}$$

Wysoka precyzja prawie zawsze jest osiągnięta kosztem czułości i na odwrót. To od aplikacji zależy, czy system należy dostroić pod kątem wysokiej precyzji, czy pod kątem wysokiej czułości. Miara F jest zwykle używana jako pojedynczy wskaźnik łączący precyzję i czułość w celu porównywania różnych zbiorów wyników:

$$F = \frac{2pr}{p+r}$$

Jedną z właściwości średniej harmonicznej jest to, że dla dwóch liczb jest ona zwykle bliższa mniejszej z tych wartości. Tak więc miara F jest automatycznie bliższa mniejszej wartości spośród czułości i precyzji. Dlatego aby miara F była wysoka, zarówno precyzja, jak i czułość muszą być wysokie.

$$F = \frac{2}{\frac{1}{p} + \frac{1}{r}}$$



## 27.7. Wyszukiwanie i analizy w sieci WWW<sup>25</sup>

Powstanie sieci WWW spowodowało, że miliony użytkowników zaczęły szukać informacji przechowywanych w bardzo wielu aktywnych witrynach. Aby zapewnić dostęp do tych informacji, wyszukiwarki takie jak Google, Bing i Yahoo! muszą przetwarzać i indeksować witryny oraz zbiory dokumentów, tworząc bazy danych z indeksami. Ponadto, z powodu zmiennej natury sieci WWW, wyszukiwarki muszą regularnie uaktualniać indeksy wraz z powstawaniem nowych witryn i aktualizowaniem lub usuwaniem istniejących. Ponieważ w sieci WWW istnieje wiele milionów stron poświęconych różnym zagadnieniom, wyszukiwarki muszą stosować różne zaawansowane techniki, takie jak np. analizy odsyłaczy, aby określać wartość stron.

Istnieją też inne rodzaje wyszukiwarek oprócz tych, które regularnie przeszukują sieć WWW i automatycznie tworzą indeksy. Są to sterowane przez ludzi wyszukiwarki pionowe i metawyszukiwarki. Są one budowane za pomocą systemów komputerowych, które wspomagają ludzi w tworzeniu indeksów. Takie wyszukiwarki obejmują ręcznie opracowywane wyspecjalizowane katalogi, które są hierarchicznie uporządkowanymi indeksami pomagającymi użytkownikom docierać do różnych zasobów w sieci WWW. **Wyszukiwarki pionowe** to niestandardowe, specyficzne dla dziedziny wyszukiwarki, które przeszukują i indeksują konkretne zbiory dokumentów w sieci WWW i zwracają wyniki z tych zbiorów. **Metawyszukiwarki** są budowane na bazie wyszukiwarek. Kierują zapytania jednocześnie do różnych wyszukiwarek, po czym agregują wyniki z tych źródeł i zwracają je.

Innym źródłem umożliwiającym przeszukiwanie dokumentów w sieci WWW są **biblioteki cyfrowe**. Na ogólnym poziomie można je zdefiniować jako zbiory elektronicznych zasobów i usług służące do udostępniania materiałów w różnych formatach. Takimi zbiorami mogą być katalogi bibliotek uniwersyteckich, katalogi grup powiązanych uczelni (tak działa np. system State of Florida University) lub kompilacje różnych zewnętrznych zasobów z sieci WWW (np. narzędzie Google Scholar lub system IEEE/ACM). Takie interfejsy zapewniają jednolity dostęp do treści różnego rodzaju, np.: książek, artykułów, danych dźwiękowych i nagrań wideo, umieszczonych w różnych systemach baz danych i zdalnych repozytoriach. Te cyfrowe kolekcje, podobnie jak prawdziwe biblioteki, są zarządzane za pomocą katalogów i uporządkowane w kategorie, co pozwala znaleźć je za pomocą internetu. Biblioteki cyfrowe „obejmują osobiste, rozproszone i scentralizowane zbiory takie jak internetowe publicznie dostępne katalogi, bibliograficzne bazy danych, rozproszone bazy dokumentów, akademickie i fachowe listy dyskusyjne, magazyny elektroniczne, inne internetowe bazy danych, fora i tablice ogłoszeń”<sup>26</sup>.

### 27.7.1. Analizy danych internetowych i ich związki z WI

Obok przeglądania i przeszukiwania sieci WWW inną ważną aktywnością ściśle związaną z WI jest *analizowanie* lub *eksploracja* sieci WWW pod kątem nowych interesujących informacji (eksplorację danych z plików i baz omówimy w rozdziale 28.). Zastosowanie tech-

---

<sup>25</sup> Dziękujemy Praneshowi P. Ranganathanowi i Hariemu P. Kumarowi za wkład w powstanie tego podrozdziału.

<sup>26</sup> Covi i Kling (1996), s. 672.

niki analizy danych do wykrywania i analizowania przydatnych informacji z sieci WWW to **analizy danych internetowych**. W ostatnich kilku latach sieć WWW stała się ważnym repozytorium informacji do wielu codziennych zastosowań dla pojedynczych użytkowników, a także istotną platformą handlu elektronicznego i działania sieci społecznościowych. Dlatego stanowi ciekawy obszar zastosowań z dziedziny analizy danych. Dziedzina eksploracji i analiz danych internetowych integruje wiele obszarów obejmujących wyszukiwanie informacji, analizy tekstu, przetwarzanie języka naturalnego, eksplorację danych, uczenie maszynowe i analizy statystyczne.

Celem analiz danych internetowych jest poprawa i spersonalizowanie adekwatności wyników wyszukiwania oraz identyfikowanie trendów, które mogą się okazać wartościowe dla różnych firm i organizacji. Dalej szczegółowo omówimy te cele.

- **Wyszukiwanie adekwatnych informacji.** Ludzie zwykle szukają w sieci WWW konkretnych informacji, wpisując słowa kluczowe w wyszukiwarce, przeglądając portale informacyjne i korzystając z usług. Usługi wyszukiwania są mocno narażone na problemy z adekwatnością wyników, ponieważ wyszukiwarki muszą *od początku* uwzględniać i próbować realizować potrzeby milionów użytkowników. Wyniki nieadekwatne dla użytkowników skutkują niską precyzją (patrz podrozdział 27.6). W sieci WWW nie da się określić, czy *czułość* jest wysoka (patrz podrozdział 27.6), ponieważ nie da się zindeksować wszystkich stron internetowych. Ponadto pomiar czułości nie ma w tej sytuacji sensu, ponieważ użytkownika interesuje tylko kilka pierwszych dokumentów. Wyniki najbardziej adekwatne dla użytkownika znajdują się zwykle wśród kilku pierwszych rezultatów.
- **Personalizowanie informacji.** Różni ludzie mają odmienne preferencje dotyczące treści i prezentacji. Do dostosowywania i personalizacji aplikacji i usług sieciowych używane są różne narzędzia (np.: monitorowanie kliknięć, śledzenie ruchu gałek ocznych, bezpośrednie lub pośrednie uczenie się na podstawie profilu użytkownika albo dynamiczne łączenie usług za pomocą internetowych interfejsów API). System personalizacji zwykle korzysta z algorytmów, które na podstawie informacji o użytkownikach (zbieranych za pomocą różnych narzędzi) generują specyficzne dla nich wyniki wyszukiwania. Sieć WWW staje się bogatym środowiskiem, w którym ludzie pozostawiają ślady, gdy poruszają się po niej, klikają, zaznaczają „lubię”, komentują i kupują produkty w tej wirtualnej przestrzeni. Takie informacje mają wysoką wartość komercyjną, dlatego wiele firm oferujących różnego rodzaju produkty eksploruje i sprzedaje te informacje na potrzeby targetowania klientów.
- **Wyszukiwanie informacji w sieciach społecznościowych.** Przy ponad miliardzie pobrań aplikacji serwisu Facebook na różne urządzenia z systemem Android można sobie wyobrazić, jak popularne stały się ostatnio różne sieci społecznościowe. Ludzie budują tzw. kapitał społeczny w wirtualnych światach takich jak Twitter i Facebook. **Kapitał społeczny** związany jest z cechami organizacji społecznych takimi jak sieci powiązań, normy i zaufanie społeczne; wspomagają one koordynację i kooperację dla osiągnięcia wzajemnych korzyści. Badacze zachowań społecznych studiują kapitał społeczny i to, jak wykorzystać ten cenny zasób, aby zapewnić społeczeństwu różne profity. W punkcie 27.8.2 opiszemy pokrótce różne aspekty wyszukiwania danych w sieciach społecznościowych.

Analizy danych internetowych można podzielić na trzy kategorie: **analizy struktury sieci WWW** (polegają na wykrywaniu wiedzy na podstawie hiperłączy reprezentujących strukturę sieci WWW), **analizy treści w sieci WWW** (dotyczą wydobywania przydatnych informacji i wiedzy z treści stron WWW) i **analizy użytkownika witryn** (ang. *web usage analysis*; polegają na eksploracji wzorców dostępu do stron na podstawie dzienników użytkownika, gdzie rejestrowana jest aktywność wszystkich użytkowników).

## 27.7.2. Analizy struktury sieci WWW

Sieć WWW to olbrzymi korpus informacji, jednak znalezienie w nim wysokiej jakości danych adekwatnych do potrzeb użytkownika jest bardzo trudne. Zbiór stron WWW potraktowany jako całość w zasadzie nie ma wspólnej struktury. Występują w nim różnice w stylu pisania i treści. Ta różnorodność utrudnia precyzyjne lokalizowanie potrzebnych informacji. Wyszukiwarki oparte na indeksie stały się jednym z podstawowych narzędzi, za pomocą których użytkownicy szukają informacji w sieci WWW. Wyszukiwarki internetowe **indeksują** (ang. *crawl*) sieć WWW i tworzą indeks na potrzeby jej przeszukiwania. Gdy użytkownik — podając słowa kluczowe — określa, jakich informacji potrzebuje, wyszukiwarki internetowe sprawdzają swoje repozytoria z indeksami i jako wyniki wyszukiwania zwracają odsyłacze lub adresy URL ze skróconą zawartością stron. Dla określonego zapytania adekwatne mogą być tysiące stron. Problem powstaje, gdy użytkownikowi zwracana jest tylko niewielka część najbardziej adekwatnych wyników. Wcześniejsze omówienie zapytań i rankingów opartych na adekwatności w systemach WI (patrz podrozdziały 27.2 i 27.3) dotyczy także wyszukiwarek internetowych. Algorytmy tworzące rankingi analizują strukturę odsyłaczy w sieci WWW.

Strony WWW, w odróżnieniu od standardowych zbiorów tekstów, zawierają połączenia (w postaci hiperłączy) z innymi stronami lub dokumentami. Umożliwia to użytkownikom przechodzenie między stronami. **Hiperłączy** ma dwa komponenty: **stronę docelową** i **tekst etykiety** opisujący odsyłacz. Przykładowo, ktoś może umieścić na swojej stronie odsyłacz do witryny Yahoo! z tekstem etykiety „Moja ulubiona witryna”. Taki tekst można traktować jak pośrednie ułatwienie. Zawiera on ważne „ukryte” informacje od człowieka. Można przyjąć, że osoba umieszczająca na swojej stronie odsyłacze do innych stron jest jakoś z nimi powiązana. Wyszukiwarki starają się filtrować wyniki na podstawie ich adekwatności i wartości. Istnieje wiele nadmiarowych hiperłączy takich jak odsyłacze do strony głównej umieszczone na każdej stronie witryny. Wyszukiwarki muszą eliminować takie hiperłączy z wyników wyszukiwania.

**Koncentrator** (ang. *hub*) to strona lub witryna z odsyłaczami do zbioru ważnych witryn (autorytatywnych) z danej dziedziny. **Strona autorytatywna** (ang. *authority*) to taka, do której prowadzi wiele dobrych koncentratorów. Z kolei dobry koncentrator to strona z odsyłaczami do wielu stron autorytatywnych. Te założenia są wykorzystywane w algorytmie tworzenia rankingu wyników HITS. Kilka algorytmów tworzenia rankingów po krótko opiszemy w dalszym punkcie.

### 27.7.3. Analizowanie struktury odsyłaczy na stronach internetowych

Celem **analizy struktury sieci WWW** jest wygenerowanie strukturalnej reprezentacji witryny i stron WWW. Analizy struktury sieci WWW dotyczą głównie wewnętrznej struktury dokumentów i struktury odsyłaczy (hiperłączy) na poziomie wielu dokumentów. Gdy dana jest kolekcja powiązanych ze sobą dokumentów, można wykryć interesujące i bogate w informacje fakty opisujące połączenia między tymi dokumentami. Analizy struktury sieci WWW pomagają też w nawigacji oraz umożliwiają porównywanie i integrowanie schematów różnych stron WWW. Ten aspekt analizy struktury sieci WWW ułatwia kategoryzowanie dokumentów i grupowanie na podstawie ich struktury.

**Algorytm tworzenia rankingu PageRank.** Wcześniej wyjaśniono, że algorytmy tworzenia rankingu służą do porządkowania wyników wyszukiwania na podstawie adekwatności i wartości. Firma Google korzysta ze znanego algorytmu **PageRank**<sup>27</sup>, opartego na „ważności” każdej strony. Wszystkie strony mają określoną liczbę odsyłaczy wyjściowych (krawędzie wyjściowe) i wejściowych (krawędzie wejściowe). Bardzo trudno jest ustalić wszystkie odsyłacze wejściowe dla strony WWW, natomiast można stosunkowo łatwo znaleźć wszystkie odsyłacze wyjściowe. W algorytmie PageRank strony z dużą liczbą odsyłaczy wejściowych są ważniejsze (bardziej autorytatywne) niż strony z mniejszą liczbą tych odsyłaczy. Jednak nie wszystkie odsyłacze wejściowe są ważne. Odsyłacz wejściowy ze strony, która jest wiarygodnym źródłem informacji, jest ważniejszy od odsyłacza z jakiejś przypadkowej strony. Dlatego strona ma duże znaczenie, kiedy strony z odsyłaczami wejściowymi do niej sumarycznie mają duże znaczenie. PageRank to próba sprawdzenia, jak dobre przybliżenie „znaczenia” strony można otrzymać na podstawie struktury odsyłaczy.

Tworzenie rankingu stron to proces iteracyjny. Ranking PageRank dla strony WWW to suma rankingów PageRank wszystkich stron z odsyłaczami wejściowymi do niej. Algorytm PageRank traktuje sieć WWW jak *model Markova*. Fikcyjny internauta odwiedza nieskończony ciąg stron, losowo klikając odsyłacze. Ranking PageRank strony pozwala oszacować, jak często internauta trafi na określoną stronę. PageRank to miara niezależnego od zapytania znaczenia strony (węzła). Przykładowo, niech  $P(X)$  będzie rankingiem PageRank dowolnej strony  $X$ ,  $C(X)$  — liczbą odsyłaczy wyjściowych ze strony  $X$ , a  $d$  — współczynnikiem tłumienia (wartością z przedziału  $0 < d < 1$ ). Zwykle  $d$  jest równe 0,85. Ranking PageRank strony  $A$  można wtedy obliczyć w następujący sposób:

$$P(A) = (1 - d) + d(P(T_1)/C(T_1) + P(T_2)/C(T_2) + \dots + P(T_n)/C(T_n))$$

W tym wzorze  $T_1, T_2, \dots, T_n$  to strony prowadzące do strony  $A$  (zawierają „cytaty” ze strony  $A$ ). PageRank wyznacza rozkład prawdopodobieństwa dla stron WWW, dlatego suma rankingów PageRank dla wszystkich stron to jeden.

**Algorytm tworzenia rankingu HITS.** Zaproponowany przez Jona Kleinberga algorytm HITS<sup>28</sup> to inny rodzaj algorytmu tworzenia rankingu wykorzystujący strukturę odsyłaczy w sieci WWW. W tym algorytmie przyjęto, że dobrym koncentratorem jest dokument z odsyłaczami do wielu koncentratorów, a stroną autorytatywną jest dokument z odno-

<sup>27</sup> Algorytm PageRank został zaproponowany przez Lawrence’a Page’a (1998) i Sergeya Brina, założycieli firmy Google. Więcej informacji znajdziesz na stronie <http://en.wikipedia.org/wiki/PageRank>.

<sup>28</sup> Patrz Kleinberg (1999).

śnikami wejściowymi z wielu innych autorytatywnych stron. Algorytm działa w dwóch głównych krokach: próbkowania i przekazywania wag. W ramach próbkowania tworzony jest analizowany zbiór stron  $S$ . Zbiór ten ma następujące cechy:

- (1)  $S$  jest stosunkowo mały.
- (2)  $S$  obejmuje dużo adekwatnych stron.
- (3)  $S$  zawiera większość najbardziej autorytatywnych stron.

W ramach przekazywania wag dla każdego dokumentu rekurencyjnie obliczane są oceny dla koncentratorów i autorytatywność:

- (1) Inicjowanie ocen dla koncentratorów i autorytatywności dla wszystkich stron ze zbioru  $S$  przez przypisanie im liczby 1.
- (2) Dopóki nie zostanie osiągnięta zbieżność ocen poszczególnych koncentratorów i autorytatywności:
  - a. Dla każdej strony z  $S$  obliczana jest autorytatywność = suma ocen koncentratorów wszystkich stron *prowadzących do* danej strony.
  - b. Dla każdej strony z  $S$  obliczana jest ocena dla koncentratora = suma autorytatywności wszystkich stron, *do których prowadzi* dana strona.
  - c. Normalizowanie ocen dla koncentratora i autorytatywności w taki sposób, aby suma wszystkich ocen dla koncentratorów w  $S$  i suma wszystkich poziomów autorytatywności w  $S$  były równe 1.

#### 27.7.4. Analizy treści w sieci WWW

Wcześniej wspomniano, że **analizy treści w sieci WWW** dotyczą procesu wykrywania przydatnych informacji na podstawie treści, danych i dokumentów w sieci WWW. **Dane z treści w sieci WWW** to dane niestrukturalne, takie jak swobodnie wprowadzany tekst dokumentów przechowywanych w formie elektronicznej, dane półstrukturalne (zwykle dokumenty HTML z osadzonymi danymi graficznymi) i dane strukturalne, takie jak dane w tabelach i strony w HTML-u, XML-u lub innych językach znaczników wygenerowane jako dane wyjściowe z baz. **Treści w sieci WWW** to rzeczywiste dane na stronie WWW przeznaczone dla użytkownika odwiedzającego tę stronę. Zwykle tymi danymi są tekst i grafika, ale spotykane są też inne formaty.

Najpierw opiszemy wstępne zadania z obszaru analizy treści w sieci WWW, a następnie przyjrzymy się tradycyjnym operacjom kategoryzowania i grupowania stron.

**Wydobywanie danych strukturalnych.** Dane strukturalne w sieci WWW są często bardzo istotne, ponieważ reprezentują najważniejsze informacje. Takie dane to np. tabele strukturalne z planem lotów między dwoma miastami. Istnieje kilka technik wydobywania danych strukturalnych. Jedna z nich polega na utworzeniu **nakładki** — programu, który wygląda jak inna strukturalna reprezentacja informacji ze strony i pozwala pobrać odpowiednie treści. Inna technika to ręczne pisanie dla każdej strony programu do wydobywania danych na podstawie zaobserwowanych wzorców formatowania witryny. Jest to bardzo praco- i czasochłonne. Trzecia technika to **uczenie nakładki** (ang. *wrapper induction* lub *wrapper learning*). Polega to na tym, że użytkownik najpierw ręcznie opisuje zbiór stron treningowych, a następnie system uczenia generuje reguły (na pod-

stawie tych stron) stosowane do wydobywania docelowych elementów z innych stron. Czwarta metoda to podejście automatyczne, nastawione na wyszukiwanie wzorców i gramatyki na stronach WWW oraz **generowanie nakładek** do automatycznego wydobywania danych.

**Integrowanie informacji z sieci WWW.** Sieć WWW jest przeogromna i obejmuje miliardy dokumentów opracowanych przez wiele różnych osób i organizacji. Z tego powodu strony WWW zawierające podobne informacje mogą mieć inną składnię i opisywać te same zagadnienia w innych słowach. Wymaga to integrowania informacji z różnorodnych stron WWW. Dwa popularne podejścia integrowania informacji to:

- (1) **Integrowanie z użyciem interfejsu zapytań**, co pozwala kierować zapytania do wielu internetowych baz danych, które nie są widoczne w zewnętrznych interfejsach i są ukryte w „głębokiej sieci WWW”. **Głęboka sieć WWW**<sup>29</sup> składa się ze stron, które nie istnieją do czasu ich dynamicznego zbudowania w wyniku konkretnej operacji przeszukiwania bazy danych, co powoduje wygenerowanie informacji dla danej strony (patrz rozdział 11.). Ponieważ tradycyjne roboty indeksujące wyszukiwarek nie potrafią sprawdzać i zapisywać informacji z takich stron, głęboka sieć WWW jest na razie dla nich niedostępna.
- (2) **Dopasowywanie schematów**, np. integrowanie katalogów w celu utworzenia globalnego schematu dla aplikacji. Przykładem tego podejścia jest dopasowywanie i łączenie w jednym rekordzie danych z różnych źródeł dzięki wzajemnemu powiązaniu rekordów ze stanem zdrowia z różnych systemów. Efekt to jeden globalny rekord ze stanem zdrowia.

Trwają aktywne badania nad tymi podejściami, a szczegółowe omawianie takich technik wykracza poza zakres tego tekstu. Więcej informacji znajdziesz w wybranych publikacjach przedstawionych na końcu rozdziału.

**Integrowanie informacji na podstawie ontologii.** To zadanie polega na wykorzystaniu ontologii do skutecznego łączenia informacji z wielu różnorodnych źródeł. Ontologie (czyli formalne modele reprezentacji danych z bezpośrednio zdefiniowanymi pojęciami i łączącymi je nazwanymi związkami) służą do radzenia sobie z problemami semantycznej niejednorodności źródeł danych. Do integrowania informacji z użyciem ontologii stosuje się różne rodzaje podejść.

- W **podejściach z jedną ontologią** używa się jednej globalnej ontologii, która zapewnia wspólny leksykon do określania semantyki. Te metody sprawdzają się, jeśli wszystkie integrowane źródła danych przedstawiają prawie ten sam obraz dziedziny wiedzy. Przykładowo, UMLS (opis w punkcie 27.4.3) można wykorzystać jako wspólną ontologię dla aplikacji biomedycznych.
- W **podejściu z wieloma ontologiami** każde źródło informacji jest opisane za pomocą jego ontologii. Teoretycznie „ontologia źródła danych” może być połączeniem kilku innych ontologii, jednak nie można zakładać, że różne „ontologie źródła danych” są oparte na wspólnym leksykonie. Radzenie sobie z wieloma częściowo pokrywającymi się i potencjalnie sprzecznymi ontologiami to poważny problem dotyczący wielu zastosowań, w tym tych z obszaru bioinformatyki i innych skomplikowanych dziedzin.

---

<sup>29</sup> Głęboka sieć WWW została zdefiniowana przez Bergmana (2001).



**Budowanie hierarchii pojęć.** Często stosowanym sposobem porządkowania wyników wyszukiwania jest tworzenie list uporządkowanych liniowo dokumentów. Jednak dla niektórych użytkowników i aplikacji lepszym sposobem wyświetlania wyników jest tworzenie grup powiązanych dokumentów. Jednym ze sposobów porządkowania dokumentów w wynikach wyszukiwania (i ogólnie organizowania informacji) jest tworzenie **hierarchii pojęć**. Dokumenty w wynikach wyszukiwania są wtedy hierarchicznie porządkowane w grupy. Powiązane techniki porządkowania dokumentów to **kategoryzowanie** i **grupowanie** (patrz rozdział 28.). Grupowanie polega na tworzeniu grup dokumentów mających wiele cech wspólnych.

**Podział stron na segmenty i wykrywanie szumu.** W dokumentach internetowych znajduje się wiele elementów nadmiarowych, takich jak reklamy i panele nawigacyjne. Informacje i tekst z tych elementów należy potraktować jak szum i wyeliminować przed kategoryzowaniem dokumentów na podstawie treści. Dlatego przed uruchomieniem algorytmów kategoryzowania lub grupowania dla zbioru dokumentów należy usunąć obszary lub bloki zawierające szum.

### 27.7.5. Podejścia analizowania treści w sieci WWW

Dwa główne podejścia analizowania treści w sieci WWW są oparte (1) na agentach (podejście WI) i (2) na bazach danych (podejście bazodanowe).

**Podejście oparte na agentach** obejmuje budowanie zaawansowanych systemów sztucznej inteligencji, które potrafią autonomicznie (lub częściowo autonomicznie) działać na rzecz użytkownika w celu wykrywania i przetwarzania informacji z sieci WWW. Systemy analizy sieci WWW oparte na agentach można podzielić na trzy kategorie:

- **Inteligentne agenty internetowe.** Są to agenty programowe, które wyszukują adekwatne informacje na podstawie cech z danej dziedziny (i czasem także profilu użytkownika) używanych do porządkowania i interpretowania znalezionych danych. Przykładowo, inteligentny agent pobiera informacje o produkcie z witryn wielu dostawców, posługując się tylko ogólnymi danymi o dziedzinie produktu.
- **Filtrowanie i kategoryzowanie informacji** to następna technika wykorzystująca agenty internetowe do kategoryzowania dokumentów w sieci WWW. Te agenty korzystają z technik WI, a także z informacji semantycznych opartych na odsyłaczach z różnych dokumentów. Na tej podstawie porządkują dokumenty w hierarchii pojęć.
- **Spersonalizowane agenty internetowe** to następny rodzaj agentów. Wykorzystują one osobiste preferencje użytkowników do porządkowania wyników wyszukiwania lub do wykrywania informacji i dokumentów, które mogą okazać się wartościowe dla danej osoby. Preferencje użytkownika można też określać na podstawie wcześniejszych wyborów lub zachowań osób, których preferencje są uznawane za podobne jak danego użytkownika.

W **podejściu opartym na bazie danych** celem jest ustalenie struktury witryny lub przekształcenie jej do postaci bazy danych, tak aby możliwe stało się lepsze zarządzanie informacjami i zgłaszanie zapytań w sieci WWW. W takich analizach treści internetowych przede wszystkim próbuje się modelować i integrować dane w sieci WWW, tak aby możliwe było wykonywanie zapytań bardziej złożonych niż wyszukiwanie oparte na słowach



kluczowych. Taki efekt można osiągnąć, określając schemat dokumentów internetowych lub budując hurtownię dokumentów, internetową bazę wiedzy albo wirtualną bazę danych. W podejściu opartym na bazach danych można wykorzystać np. model OEM (ang. *Object Exchange Model*)<sup>30</sup>, w którym dane półstrukturalne są reprezentowane w grafie z etykietami. Dane z modelu OEM są traktowane jak graf, w którym obiekty są wierzchołkami, a do każdego przypisane są etykiety. Każdy obiekt jest identyfikowany na podstawie identyfikatora i wartości, która może być albo atomowa (liczba całkowita, ciąg znaków, obraz GIF, dokument HTML), albo złożona (zestaw referencji do obiektów).

Główny nacisk w podejściu opartym na bazach danych jest położony na stosowanie wielopoziomowych baz danych i internetowych systemów zapytań. W **wielopoziomowej bazie danych** na najniższym poziomie znajduje się baza zawierająca proste półstrukturalne informacje zapisane w różnych repozytoriach internetowych (np. w dokumentach hipertekstowych). Na wyższych poziomach generowane są metadane lub uogólnienia (na podstawie niższych poziomów), porządkowane w strukturalnych kolekcjach takich jak relacyjne lub obiektowe bazy danych. W **internetowych systemach obsługi zapytań** (ang. *web query systems*) informacje o treści i strukturze dokumentów internetowych są pobierane i porządkowane za pomocą technik podobnych do tych używanych w bazach danych. Następnie do wyszukiwania dokumentów internetowych i zgłaszania dotyczących ich zapytań można stosować języki zapytań podobne do języka SQL. Zapytania tego rodzaju to połączenie zapytań strukturalnych (opartych na organizacji dokumentów hipertekstowych) i opartych na treści.

## 27.7.6. Analizy użytkowania witryn

**Analizy użytkowania witryn** polegają na wykorzystaniu technik analizy danych do wykrywania wzorców użytkowania witryn na podstawie danych z sieci WWW. Celem jest zrozumienie i lepsze zaspokajanie potrzeb aplikacji internetowych. Ten proces nie wpływa bezpośrednio na wyszukiwanie informacji. Jest jednak ważny w kontekście poprawy i wzbogacenia doświadczeń użytkownika w trakcie wyszukiwania.

**Dane o użytkowaniu witryn** dotyczą wzorców używania stron WWW, np.: adresów IP, stron źródłowych oraz daty i godziny dostępu do stron przez użytkowników, grupy osób lub aplikacje. Analizy użytkowania witryn zwykle obejmują trzy podstawowe etapy: wstępne przetwarzanie, wykrywanie wzorców i analizy wzorców.

- (1) **Wstępne przetwarzanie.** Wstępne przetwarzanie przekształca informacje dotyczące statystyk i wzorców użytkowania na postać, którą można wykorzystać w metodach wykrywania wzorców. Przykładowo, pojęcie *odstona strony* oznacza wyświetlenie strony lub odwiedzin na niej. Istnieje kilka różnych rodzajów technik wstępnego przetwarzania:
  - **Wstępne przetwarzanie danych o użytkowaniu** polega na analizie dostępnych danych o wzorcach użytkowania typowych dla użytkowników, aplikacji lub grup osób. Ponieważ te dane są często niekompletne, jest to trudny proces. Potrzebne są techniki oczyszczania danych, aby wyeliminować wpływ nieistotnych elementów na wyniki analiz. Dane o użytkowaniu często są identyfikowane

---

<sup>30</sup> Patrz Kosala i Blockeel (2000).

za pomocą adresów IP i obejmują strumienie kliknięć zapisywane na serwerze. Lepsze dane są dostępne, jeśli mechanizm śledzenia użytkownika jest zainstalowany po stronie klienta.

- **Wstępne przetwarzanie treści** to proces przekształcania tekstu, obrazów, skryptów i innych materiałów na postać, którą można wykorzystać do analiz użytkownika. Ten proces często polega na analizie treści, np. na kategoryzowaniu lub grupowaniu. Techniki grupowania i kategoryzowania pozwalają grupować informacje o użytkowniku dla stron WWW podobnego typu, dzięki czemu można wykrywać wzorce użytkownika dla konkretnych klas stron opisujących określone tematy. Można też kategoryzować odsłony stron zgodnie z ich celem (np.: zakupy, szukanie informacji lub inne zastosowania).
- **Wstępne przetwarzanie struktury** polega na parsowaniu i zmianie formatowania informacji o hiperłączach i strukturze łączącej wyświetlane strony. Trudnością jest tu to, że struktura witryny może być dynamiczna i wymagać budowania w każdej sesji serwera.

(2) **Wykrywanie wzorców.** Techniki wykrywania wzorców są oparte na metodach z obszaru statystyki, uczenia maszynowego, rozpoznawania wzorców, analizy danych, eksploracji danych itp. Te techniki są dostosowywane tak, aby uwzględniały specyficzną wiedzę i cechy analiz sieci WWW. Przykładowo, przy wykrywaniu reguł asocjacji (patrz podrozdział 28.2) w analizach koszyka zakupów przyjmuje się, że kolejność produktów jest nieistotna. Jednak kolejność dostępu do stron WWW jest ważna i w analizach użytkownika witryn należy ją uwzględnić. Dlatego wykrywanie wzorców obejmuje eksplorację sekwencji odsłon stron. Gdy do wykrywania wzorców stosowane są dane o użytkowniku witryn, wykonuje się następujące rodzaje eksploracji danych:

- **Analizy statystyczne.** Techniki statystyczne to najczęściej stosowane metody zdobywania wiedzy o użytkownikach witryn internetowych. Dzięki analizie dzienników sesji można zastosować miary statystyczne takie jak średnia, mediana i liczba wystąpień do parametrów takich jak odsłony stron, czas przeglądania na stronę, długość ścieżek nawigacji między stronami i inne aspekty istotne w analizach użytkownika witryn.
- **Reguły asocjacji.** W analizach użytkownika witryn reguły asocjacji dotyczą zbiorów stron, które są używane razem (dla wartości wsparcia przekraczającej określony poziom progowy; więcej o regułach asocjacji znajdziesz w podrozdziale 28.2). Takie strony nie muszą być bezpośrednio powiązane ze sobą hiperłączami. Przykładowo, w ramach wykrywania reguł asocjacji można znaleźć korelację między użytkownikami odwiedzającymi strony z produktami elektronicznymi i osobami wyświetlającymi strony ze sprzętem sportowym.
- **Grupowanie.** W obszarze użytkownika witryn można wykryć dwa ciekawe rodzaje grup: grupy użytkowników i grupy stron. **Grupy użytkowników** obejmują osoby o podobnych wzorcach przeglądania stron. Taka wiedza jest przydatna przede wszystkim do wyciągania wniosków o demograficznych cechach użytkowników na potrzeby segmentacji rynku w sklepach internetowych lub w celu udostępniania odbiorcom spersonalizowanych materiałów. **Grupy stron** są oparte na treści. Strony o podobnej zawartości są ze sobą łączone. Grupowanie tego rodzaju można wykorzystać w wyszukiwarkach internetowych i narzędziach wspomagających przeglądanie stron WWW.

- **Kategoryzowanie.** W sieci WWW jednym z celów jest opracowanie profilu użytkowników należących do określonej klasy lub kategorii. Wymaga to określania i wyboru cech, które najlepiej opisują właściwości danej klasy lub kategorii użytkowników. Oto przykładowy ciekawy wzorzec, jaki można wykryć: 60% użytkowników, którzy złożyli internetowe zamówienie z kategorii /Produkt/Książka, należy do grupy wiekowej 18 – 25 i mieszka w wynajmowanym mieszkaniu.
  - **Wzorce sekwencyjne.** Tego rodzaju wzorce pozwalają identyfikować sekwencje odsłon stron WWW, na których podstawie można prognozować następny zestaw stron odwiedzanych przez użytkowników z określonej kategorii. Te wzorce mogą być używane przez marketingowców do zamieszczania na stronach reklam dostosowanych do docelowych odbiorców. Inny rodzaj wzorców sekwencyjnych dotyczy produktów kupowanych zwykle po nabyciu konkretnego towaru. Przykładowo, po zakupie komputera często nabywana jest drukarka.
  - **Modelowanie zależności.** Celem modelowania zależności jest ustalenie i zamodelowanie ważnych zależności między różnymi zmiennymi w sieci WWW. Może to być np. zbudowanie modelu reprezentującego różne etapy, przez jakie przechodzi internauta w trakcie zakupów w sklepie internetowym. Taki model jest oparty na działaniach użytkowników i może odróżniać przypadkowych odwiedzających od osób poważnie zainteresowanych zakupem.
- (3) **Analizy wzorców.** Ostatni krok polega na odfiltrowaniu reguł lub wzorców, które na podstawie wykrytych wzorców są uznawane za nieistotne. Często stosowaną techniką analizy wzorców jest używanie języka zapytań (takiego jak SQL) do wykrywania różnych wzorców i związków. Inna metoda obejmuje wczytywanie danych o użytkowaniu stron do hurtowni danych za pomocą narzędzi ETL i wykonywanie operacji OLAP do przeglądania danych na wielu wymiarach (patrz podrozdział 29.3). Często stosuje się tu techniki wizualizacji, takie jak przedstawianie wzorców w formie graficznej lub przypisywanie kolorów różnym wartościom, co pozwala wyróżnić wzorce lub trendy w danych.

### 27.7.7. Praktyczne zastosowania analiz użytkowania witryn

**Analityka internetowa.** Analityka internetowa ma pomóc w zrozumieniu i optymalizacji użytkowania witryn WWW. Wymaga zbierania, analizowania i monitorowania danych o skuteczności użytkowania witryn. Wewnętrzna analityka internetowa mierzy skuteczność witryny w kontekście komercyjnym. Takie dane są zwykle porównywane z kluczowymi wskaźnikami wydajności w celu pomiaru skuteczności lub wydajności całej witryny. Pozwala to usprawnić witrynę lub strategię marketingowe.

**Spamowanie w sieci WWW.** Dla firm i osób prywatnych coraz ważniejsze jest, by ich witryny i strony pojawiały się na początku listy wyników wyszukiwania. Aby osiągnąć ten efekt, trzeba zrozumieć używane w wyszukiwarkach algorytmy tworzenia rankingów i umieszczać informacje na stronie w taki sposób, by strona zajmowała wysokie pozycje dla zapytań z odpowiednimi słowami kluczowymi. Wąska linia oddziela dozwolone optymalizowanie stron w celach biznesowych od spamowania. **Spamowanie w sieci WWW** jest de-

finiowane jako celowe promowanie swojej strony za pomocą manipulowania wynikami zwracanymi przez wyszukiwarki. Analizy sieci WWW można wykorzystać do wykrywania takich stron i odrzucania ich z wyników wyszukiwania.

**Bezpieczeństwo w sieci WWW.** Analizy można wykorzystać do wyszukiwania ciekawych wzorców użytkowania witryn. Jeśli wykorzystana została luka w witrynie, można to wykryć za pomocą analiz użytkowania witryn. Pozwala to projektować lepiej zabezpieczone serwisy. Przykładowo, „tylną furtkę” lub wyciekanie informacji z serwerów WWW można wykryć za pomocą technik analiz sieci WWW związanych z nietypowymi danymi w dzienniku aplikacji internetowej. Techniki analizy bezpieczeństwa takie jak wykrywanie włamań i ataków DoS są oparte na analizach dostępu do stron WWW.

**Internetowe roboty indeksujące.** Są to programy, które odwiedzają strony WWW i tworzą kopie wszystkich otwartych stron, aby wyszukiwarka mogła je przetwarzać w celu indeksowania pobranych stron i umożliwienia szybkiego wyszukiwania. Innym zastosowaniem robotów indeksujących jest automatyczne sprawdzanie witryn i zarządzanie nimi. Robot indeksujący może badać i sprawdzać poprawność kodu w języku HTML oraz odsyłaczy w witrynie WWW. Inne, niewłaściwe zastosowanie robotów indeksujących związane jest ze zbieraniem adresów e-mail i innych danych osobowych ze stron WWW. Te informacje są później wykorzystywane do wysyłania e-maili ze spamem.

## 27.8. Trendy w wyszukiwaniu informacji

W tym podrozdziale opiszemy kilka zagadnień uwzględnianych w nowych badaniach nad WI.

### 27.8.1. Wyszukiwanie fasetowe

Wyszukiwanie fasetowe to technika umożliwiająca zintegrowane wyszukiwanie i nawigowanie. Umożliwia ona użytkownikom eksplorację danych dzięki ich filtrowaniu. Jest często stosowana w witrynach i aplikacjach sklepów internetowych oraz pozwala użytkownikom nawigować po wielowymiarowej przestrzeni danych. Wyszukiwanie fasetowe służy zwykle do obsługi trzech lub więcej wymiarów kategoryzacji. Te wymiary umożliwiają **systemowi kategoryzacji fasetowej** klasyfikowanie obiektów na różne sposoby na podstawie rozmaitych kryteriów taksonomicznych. Przykładowo, stronę WWW można kategoryzować na wiele sposobów: według treści (linie lotnicze, muzyka, wiadomości itd.), przeznaczenia (sprzedaż, informacje, rejestracja itd.), lokalizacji, używanego języka (HTML, XML itd.) i innych aspektów. Obiekt można więc kategoryzować na różne sposoby na podstawie wielu taksonomii.

**Fasety** definiują właściwości lub cechy klasy obiektów. Te właściwości muszą wzajemnie się wykluczać i reprezentować wszystkie obiekty. Przykładowo, kolekcję dzieł sztuki można pokategoryzować według następujących fasetów: artysta (nazwisko autora), epoka (czas utworzenia dzieła), typ (obraz, rzeźba, fresk itd.), kraj pochodzenia, technika (olej, akwarela, kamień, metal, mieszana itd.), kolekcja (w której znajduje się dane dzieło) itd.

W wyszukiwaniu fasetowym stosowana jest kategoryzacja fasetowa umożliwiaująca użytkownikowi nawigowanie po informacjach wieloma ścieżkami odpowiadającymi różnym sposobom uporządkowania fasetów. Różni się to od tradycyjnych taksonomii, w których hierarchia kategorii jest stała. Projekt Flamenco<sup>31</sup> z Uniwersytetu Kalifornijskiego w Berkeley jest jednym z wczesnych przykładowych systemów wyszukiwania fasetowego. Większość witryn sklepów internetowych, np. Amazon i Expedia, udostępnia w swoich interfejsach wyszukiwanie fasetowe, aby umożliwić szybkie porównywanie i nawigowanie z użyciem różnych aspektów związanych z kryteriami wyszukiwania.

## 27.8.2. Wyszukiwanie społecznościowe

Nawigację i przeglądanie w sieci WWW tradycyjnie traktuje się jak szukanie informacji przez jednego użytkownika. Jest to podejście różne od przyjmowanego we wcześniejszych badaniach bibliotekoznawców, którzy analizowali nawyki użytkowników szukających informacji. W takich badaniach wykazano, że dodatkowe osoby mogą być cennym źródłem informacji w trakcie poszukiwań prowadzonych przez jednego użytkownika. Nowsze badania wskazują na to, że w trakcie wyszukiwania informacji w sieci WWW użytkownicy często ze sobą współpracują. Z niektórych badań wynika, że duże grupy użytkowników bezpośrednio angażują się w wyszukiwanie w sieci WWW. W niektórych sytuacjach (np. w firmach) ma to postać aktywnej współpracy wielu stron; w innych scenariuszach (prawdopodobnie w większości przypadków wyszukiwania) użytkownicy wchodzi w interakcje zdalnie, asynchronicznie, a nawet nieświadomie i pośrednio.

Wyszukiwanie informacji w sieci WWW z udziałem społeczności (wyszukiwanie społecznościowe) to nowe zjawisko wspomagane nowymi technologiami internetowymi. **Kolektywne wyszukiwanie społecznościowe** obejmuje różne sposoby aktywnego angażowania się w działania związane z wyszukiwaniem, takie jak wyszukiwanie zespołowe w tej samej lokalizacji, zdalna współpraca nad zadaniami wyszukiwania, wyszukiwanie z wykorzystaniem sieci społecznościowych, używanie sieci eksperckich, stosowanie eksploracji danych społecznościowych lub inteligencji kolektywnej do usprawnienia procesu wyszukiwania, a także wykorzystanie interakcji społecznych do wspomagania wyszukiwania informacji i nadawania sensu (ang. *sense making*). Wyszukiwanie społecznościowe może odbywać się synchronicznie, asynchronicznie, w tej samej lokalizacji lub w zdalnie współdzielonej przestrzeni roboczej. Psychologowie społeczni eksperymentalnie potwierdzili, że dyskusje społeczne poprawiają zdolności poznawcze. Ludzie w grupach mogą generować rozwiązania (odpowiedzi na pytania), proponować skorzystanie z bazy danych lub wiedzy innych osób (metawiedza), a także potwierdzać poprawność i trafność pomysłów. Ponadto grupy społeczne mogą pomóc w przypomnieniu sobie czegoś i przeformułowaniu problemu. **Uczestnictwo prowadzone** (ang. *guided participation*) to proces, w którym ludzie współtworzą wiedzę razem z innymi osobami z ich społeczności. Obecnie wyszukiwanie informacji w sieci WWW jest zwykle samodzielnie wykonywanym zadaniem. W nowych pracach nad wyszukiwaniem kolektywnym opisano kilka ciekawych odkryć i potencjał tego podejścia w zakresie lepszego dostępu do informacji. Użytkownicy coraz częściej korzystają z sieci społecznościowych takich jak Facebook do szukania opinii i informacji oraz czytają recenzje produktów przed dokonaniem zakupu.

---

<sup>31</sup> Yee (2003) omawia metadane fasetowe używane do wyszukiwania obrazów.

### 27.8.3. Wyszukiwanie informacji w dialogach

**Wyszukiwanie informacji w dialogach** (ang. *conversational information access*) to interaktywny i kolektywny sposób wyszukiwania informacji. Uczestnicy angażują się w naturalną międzyludzką konwersację, a inteligentne agenty słuchają jej w tle i **wydobywają intencję**, aby zapewnić uczestnikom dostosowane do potrzeb informacje. Agenty mogą nawiązywać bezpośrednie lub dyskretne interakcje z użytkownikami za pomocą sprzętu mobilnego lub urządzeń komunikacyjnych typu wearable (urządzenia ubieralne). Interakcje tego rodzaju wymagają technologii takich jak identyfikowanie rozmówców, wykrywanie słów kluczowych, automatyczne rozpoznawanie mowy, semantyczne zrozumienie konwersacji i analizowanie dialogu, aby szybciej zapewnić użytkownikom adekwatne kwestie do omawiania. Wymienione technologie sprawiają, że wyszukiwanie informacji zmienia się z samodzielnie wykonywanego zadania w działanie kolektywne. Ponadto wyszukiwanie informacji staje się bardziej ukierunkowane na cel, gdy agenty korzystają z wielu technologii do zbierania adekwatnych informacji, a uczestnicy przekazują agentom informacje zwrotne.

### 27.8.4. Probabilistyczne modelowanie tematu

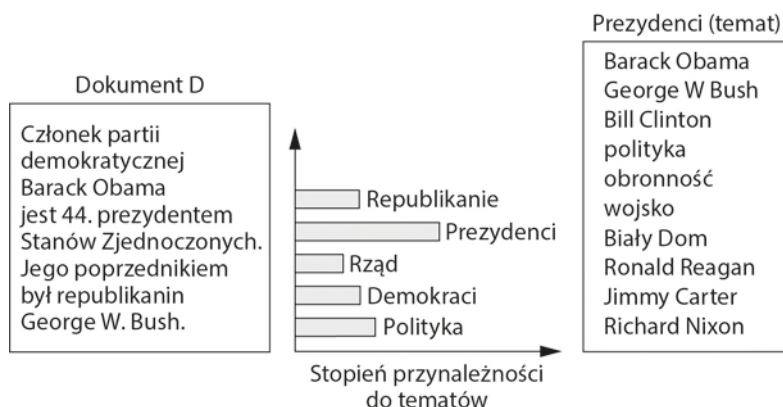
Bezprecedensowy wzrost ilości informacji generowanych z powodu powstania sieci WWW doprowadził do problemów z porządkowaniem danych w kategoriach, które mają ułatwić poprawne i wydajne udostępnianie informacji. Przykładowo, międzynarodowe agencje informacyjne, takie jak Reuters i Associated Press, codziennie zbierają z całego świata informacje dotyczące biznesu, sportu, polityki, technologii itd. Wydajne uporządkowanie takich ilości informacji jest poważnym wyzwaniem. Wyszukiwarki tradycyjnie uwzględniają słowa (w dokumentach) i odsyłacze (między dokumentami), aby zapewnić dostęp do informacji w sieci WWW. Porządkowanie informacji na bazie tematów i motywów dokumentów umożliwia użytkownikom nawigowanie po dużych ilościach informacji na podstawie tematów interesujących daną osobę.

Aby rozwiązać ten problem, w ostatniej dekadzie opracowano kategorię algorytmów uczenia maszynowego do **probabilistycznego modelowania tematów**. Te algorytmy mogą automatycznie porządkować duże kolekcje dokumentów na podstawie odpowiednich tematów. Wspaniałą cechą tych algorytmów jest to, że działają w pełni autonomicznie. To oznacza, że nie wymagają żadnych zbiorów treningowych ani informacji od ludzi, aby przeprowadzać ekstrapolację tematyczną. Oto zasady działania algorytmów tej klasy: w dokumentach dotyczących Baracka Obamy mogą się znaleźć wzmianki o innych prezydentach, innych zagadnieniach związanych z rządem lub konkretnych tematach politycznych. Artykuł o jednym z filmów z serii *Iron Man* może obejmować różne odniesienia do innych postaci ze świata fantastyki naukowej z komiksów Marvela, a jego ogólną tematyką jest fantastyka naukowa. Naturalną strukturę dokumentów można wydobyć za pomocą metod modelowania i szacowania probabilistycznego. Oto inny przykład: założmy, że każdy dokument dotyczy zbioru różnych tematów poruszanych w różnych proporcjach. Przykładowo, dokument o polityce może też opisywać prezydentów i historię Ameryki. Ponadto każdy temat jest powiązany ze zbiorem słów.

Na podstawie rysunku 27.6 można odgadnąć, że dokument D ze wzmiankami o prezydentach Stanów Zjednoczonych Baracku Obamie i George'u W. Bushu może dotyczyć tematów „prezydenci”, „polityka”, „demokraci”, „republikanie” i „rząd”. Tematy są po-



wiązane z ustalonym leksykonem. Ten leksykon jest ustalany na podstawie kolekcji dokumentów, dla których tworzone są modele tematów. Zwykle określana jest liczba tematów, jakie mają zostać utworzone na podstawie kolekcji. Dla każdego tematu słowa są porządkowane w inny sposób na podstawie tego, jak często dane wyrazy pojawiają się w różnych dokumentach na określony temat. Na rysunku 27.6 poziome słupki reprezentujące stopień przynależności do tematów powinny się sumować do jedności. Dokument D należy przede wszystkim do tematu „prezydenci”, co widać na wykresie z poziomymi słupkami. Na rysunku pokazane są tematy pokrewne tematowi „prezydenci” i lista słów powiązanych z tym ostatnim.



RYSUNEK 27.6. Dokument D i stopień jego przynależności do tematów

Probabilistyczne modelowanie tematów pozwala oszacować rozkład tematów za pomocą algorytmu uczenia, w którym zakłada się, że dokumenty mają stopień przynależności do różnych tematów. Szacunki tych stopni są wyznaczane na podstawie algorytmów próbkowania i maksymalizowania oczekiwań. Do generowania modeli tematów służy algorytm LDA (ang. *latent Dirichlet allocation*)<sup>32</sup>. Stosowane jest tu podejście generatywne, zgodnie z którym dokumenty są połączeniem ukrytych tematów, a tematy wynikają z rozkładu słów. W modelu generatywnym na podstawie ukrytych parametrów losowo generowane są obserwowalne dane. Te ukryte (nieobserwowalne) parametry to rozkład Dirichleta<sup>33</sup>, prawdopodobieństwo *a priori* występowania słów i tematów, rozkłady tematów i rozkłady słów dla tematów. Do dopasowywania tych ukrytych parametrów na podstawie zaobserwowanych danych (słów z dokumentów) stosuje się metody wnioskowania bayesowskiego, np. próbkowanie Gibbsa<sup>34</sup>.

## 27.8.5. Systemy odpowiadania na pytania

Odpowiadanie na pytania stało się popularnym tematem badań z powodu gwałtownego rozwoju technologii asystentów wirtualnych (np. Siri Apple'a i Cortana Microsoftu). Te technologie są rozwinięciem systemów IVR (ang. *interactive voice response*), które korzystają

<sup>32</sup> Patrz Blei, Ng i Jordan (2003).

<sup>33</sup> Kotz, Balakrishnan i Johnson (2000).

<sup>34</sup> German i German (1984).



przede wszystkim na technikach rozpoznawania mowy takich jak wykrywanie słów kluczowych. Odpowiadanie na pytania wymaga zaawansowanego zrozumienia pytań w języku naturalnym. Niedawno firma IBM zapisała się w historii, opracowując system odpowiadania na pytania Watson, który wziął udział w popularnym programie telewizyjnym *Jeopardy!*<sup>35</sup> i pokonał w nim ludzkich przeciwników. Odpowiadanie na pytania stało się praktyczną dziedziną inżynierii, obejmującą techniki takie jak parsowanie, rozpoznawanie nazwanych jednostek (ang. *named entity recognition* — NER), określanie głównych zagadnień, określanie rodzaju odpowiedzi, określanie związków, wnioskowanie ontologiczne oraz algorytmy wyszukiwania, indeksowania i kategoryzowania. Techniki odpowiadania na pytania obejmują też inżynierię wiedzy z wykorzystaniem dużych niestukturalnych korpusów danych, takich jak kolekcje dokumentów w sieci WWW i bazy strukturalne obejmujące wiedzę z różnych dziedzin. Wspomniane kolekcje dokumentów są zwykle na tyle duże, że wymagają zastosowania narzędzi i technologii z obszaru big data (niektóre z nich omówiono w rozdziale 25.). W dalszych podpunktach opiszemy główne zagadnienia związane z odpowiadaniem na pytania.

**Rodzaje pytań.** W systemach odpowiadania na pytania ważne jest, aby poznać kategorię lub typ pytania, ponieważ strategie udzielania odpowiedzi w dużym stopniu zależą od rodzaju pytania. Niektóre kategorie nie wykluczają się nawzajem, dlatego wymagają stosowania strategii hybrydowych. Na ogólnym poziomie pytania można podzielić na następujące kategorie:

**Pytania o fakty.** Tego rodzaju pytania wymagają określenia w dokumencie lub bazie właściwego zdania, które zapewnia poprawną odpowiedź na pytanie. Przykładowe pytania tego typu to: „Kto jest prezydentem Stanów Zjednoczonych?”, „W jakim mieście urodził się Elvis Presley?”, „Gdzie znajduje się międzynarodowy port lotniczy im. Hartsfielda Jacksona?” lub „O której godzinie zajdzie dziś słońce?”.

**Pytania o listy.** Pytania tego typu wymagają list odpowiedzi z faktami spełniającymi określony warunek. Oto przykłady: „Podaj tytuły trzech sztuk autorstwa Szekspira”, „Podaj nazwiska aktorów grających rolę Jamesa Bonda w serii filmów o agencie 007” i „Wymień trzy czerwone warzywa”.

**Pytania o definicję.** Pytania tego typu dotyczą definicji i znaczenia pojęć oraz określania kluczowych informacji i cech dotyczących danego pojęcia. Oto przykłady: „Czym jest gaz obojętny?”, „Kim był Aleksander Wielki?” lub „Czym jest stopa LIBOR?”.

**Pytania o opinie.** W pytaniach tego rodzaju ważne są różne opinie na temat danej kwestii. Oto przykłady: „Które państwa powinny mieć możliwość testowania broni atomowej?” i „Jakie jest w Arabii Saudyjskiej nastawienie do terroryzmu na Środkowym Wschodzie?”.

W ostatnich latach uczestnicy projektów środowisk badawczych i akademickich zaczęli zalecać stosowanie wspólnych miar, architektur, narzędzi i metodyk do opracowania podstaw, które pozwolą wspomagać i usprawniać techniki odpowiadania na pytania.

**Architektury.** Większość zaawansowanych architektur odpowiadania na pytania obejmuje potoki składające się z następujących etapów:

---

<sup>35</sup> Patrz Ferrucci i in. (2010).

**Analizowanie pytania.** Ten etap obejmuje analizowanie pytań i przekształcanie ich na strukturalne reprezentacje analizowanego tekstu na potrzeby przetwarzania go na dalszych etapach. Typy odpowiedzi są określane na podstawie przetworzonej reprezentacji pytania za pomocą niektórych lub wszystkich z następujących technik: płytkie parsowanie semantyczne, wykrywanie głównych elementów, określanie typu odpowiedzi, rozpoznawanie nazwanych jednostek i wykrywanie tożsamych referencji.

- **Płytkie parsowanie semantyczne.** Jest to proces przypisywania znaczników z podstawowego poziomu do struktur zdania za pomocą metody nadzorowanego uczenia maszynowego. Do zdań automatycznie przypisywane są ramy w wyniku próby dopasowania tekstu do elementów „KTO zrobił CO, KOMU, KIEDY, GDZIE, DLACZEGO i JAK”.
- **Wykrywanie głównych elementów.** Na zdjęciu niektóre fragmenty przyciągają uwagę, natomiast inne pozostają w tle. Fragmenty przyciągające uwagę to główne elementy. Podobnie w odpowiadaniu na pytania występują główne słowa zawierające odniesienia do odpowiedzi. Przykładowo, w pytaniu „Która sztuka Szekspira jest tragedią o kochankach?” można wyróżnić główne słowa „sztuka Szekspira” na podstawie reguły „która X”, gdzie X to wyrażenie rzeczownikowe w zdaniu. W systemach odpowiadania słowa główne są używane do wykrywania ukierunkowanego wyszukiwania i wspomagania określania odpowiedzi.
- **Kategoryzowanie typu odpowiedzi.** Ten etap pomaga ustalić kategorie odpowiedzi. W poprzednim przykładzie pierwsze ze słów głównych, „sztuka”, wyznacza typ odpowiedzi na pytanie. W odpowiadaniu na pytania stosuje się różne techniki uczenia maszynowego, aby ustalić typ odpowiedzi.
- **Rozpoznawanie nazwanych jednostek.** Ta faza służy do przypisywania elementów z tekstu do zdefiniowanych kategorii, takich jak osoba, miejsce, zwierzę, kraj, rzeka, kontynent.
- **Wykrywanie tożsamych referencji.** To zadanie dotyczy identyfikowania w tekście wielu wyrażen dotyczących tej samej rzeczy. Przykładowo, w zdaniu „Janek powiedział, że on sam to zrobi” zaimek „on” dotyczy Janka i jest tożsamą referencją w tekście.

**Generowanie zapytań.** Na tym etapie analizowany tekst jest wykorzystywany do wygenerowania wielu zapytań (za pomocą technik normalizacji i rozszerzania zapytań), kierowanych do jednej lub kilku wyszukiwarek, w której mogą być dostępne odpowiedzi. Przykładowo, dla pytania „Która sztuka Szekspira dotyczy tragedii kochanków?” rozszerzonymi zapytaniami mogą być „romans autorstwa Szekspira”, „sztuki Szekspira”, „romans tragedia autor Szekspir”, „romans gatunek tragedia autor Szekspir” itd. Zwykle ustalone słowa kluczowe, typy odpowiedzi, informacje o synonimach i nazwane jednostki są stosowane w różnych kombinacjach do uzyskania różnych zapytań.

**Wyszukiwanie.** Na tym etapie zapytania są przesyłane do różnych wyszukiwarek i pobierane są odpowiednie fragmenty. Używane mogą być wyszukiwarki internetowe (np. Google lub Bing) oraz te działające lokalnie (takie jak Lucene lub Indri<sup>36</sup>).

---

<sup>36</sup> <http://www.lemurproject.org/indri/>.

**Generowanie kandydujących odpowiedzi.** Do wyszukanych fragmentów stosowane są mechanizmy pobierania nazwanych jednostek, a te jednostki porównuje się z oczekiwanymi typami odpowiedzi, aby uzyskać kandydujące odpowiedzi. W zależności od pożądanej szczegółowości stosowane są algorytmy generowania odpowiedzi kandydujących i dopasowywania ich do typu odpowiedzi (np. algorytmy płytkiego dopasowywania do wzorców i algorytmy dopasowywania strukturalnego). W płytkim dopasowywaniu do wzorca używane są szablony wyrażeń regularnych z argumentami z pytania. Szablony te dopasowuje się do jednostek leksykalnych z pobranych fragmentów w celu wydobywania odpowiedzi. Przykładowo, słowa główne są dopasowywane do fragmentów potencjalnie zawierających odpowiedzi w celu ustalenia odpowiedzi kandydujących. W zdaniu „*Romeo i Julia* to sztuka Szekspira, która jest tragedią o kochankach” słowa „*Romeo i Julia*” mogą prawie bezpośrednio zastąpić człon „która” z pytania „Która sztuka Szekspira jest tragedią o kochankach?”. W dopasowywaniu strukturalnym pytania i wyszukiwane fragmenty są parsowane oraz dopasowywane na podstawie składni i znaczenia w celu znalezienia odpowiedzi kandydujących. Zdania takiego jak „Szekspir napisał tragedię o romansie Romea i Julii” nie da się powiązać ze wspomnianym pytaniem na płytkim poziomie, jednak dzięki odpowiedniemu parsowaniu i dopasowaniu możliwe jest wykrycie powiązania strukturalnego.

**Ocena odpowiedzi.** Na tym etapie szacowany jest poziom pewności dla odpowiedzi kandydujących. Podobne odpowiedzi są ze sobą scalane. Źródła wiedzy można ponownie wykorzystać, aby zebrać dowody na potwierdzenie prawdziwości różnych odpowiedzi kandydujących.

## 27.9. Podsumowanie

W tym rozdziale omówiliśmy ważną dziedzinę nazywaną wyszukiwaniem informacji (WI), ściśle powiązaną z bazami danych. Wraz z powstaniem sieci WWW z niesamowitą szybkością zaczęły się pojawiać niestrukturalne dane z tekstem, obrazem, dźwiękiem i wideo. Choć SZBD bardzo dobrze radzą sobie z danymi strukturalnymi, dane niestrukturalne zawierające różnorodne typy danych są przechowywane głównie w doraźnie tworzonych repozytoriach w sieci WWW dostępnych przede wszystkim za pomocą systemów WI. Google, Yahoo! i podobne wyszukiwarki to systemy WI oferujące zwykłym użytkownikom łatwy dostęp do nowinek z tej dziedziny. Te systemy zapewniają użytkownikom bogatsze i ciągle usprawniane mechanizmy wyszukiwania.

W podrozdziale 27.1 zaczęliśmy od przedstawienia dziedziny WI (punkt 27.1.1) oraz porównania WI i technologii baz danych (punkt 27.1.2). W punkcie 27.1.3 opisaliśmy krótką historię WI, a w punkcie 27.1.4 omówiliśmy sposoby interakcji z systemami WI: tryby zapytań i przeglądania.

W podrozdziale 27.2 zaprezentowaliśmy różne modele wyszukiwania stosowane w systemach WI, w tym logiczny, przestrzeni wektorowej, probabilistyczny i semantyczny. Te modele umożliwiają pomiar tego, czy dokument jest adekwatny do zapytania użytkownika, i zapewniają heurystyczne miary podobieństwa. W podrozdziale 27.3 przedstawiliśmy różne typy zapytań. Obok najczęściej stosowanych zapytań opartych na słowach

kluczowych istnieją też inne typy: zapytania logiczne, z wyszukiwaniem fraz, z określoną odległością między słowami, w języku naturalnym i inne; wymagają one bezpośredniej obsługi w modelach wyszukiwania danych. W systemach WI ważne jest wstępne przetwarzanie tekstu. W podrozdziale 27.4 opisano różne operacje, takie jak usuwanie słów pomijanych, stemming i wykorzystanie tezauryusa. Dalej omówiono tworzenie i stosowanie indeksów odwróconych (podrozdział 27.5), które są istotą systemów WI i wpływają na wydajność wyszukiwania. Następnie, w podrozdziale 27.6, przedstawiono różne miary oceny, np.: czułość, precyzję i miarę F. Służą one do pomiaru jakości wyników zapytań do systemów WI. Opisano też otwarty system indeksowania i wyszukiwania Lucene oraz jego rozszerzenie — Solr. Pokrótce objaśniono też informacje zwrotne na temat adekwatności. Ważne jest, aby modyfikować i usprawniać wyszukiwanie dostosowanych do użytkownika informacji na podstawie interakcji z nim i jego zaangażowania w proces wyszukiwania.

W podrozdziale 27.7 przedstawiono dość szczegółowe wprowadzenie do analiz w sieci WWW w kontekście WI. To omówienie podzielono na analizy treści, struktury i użytkowania witryn. Opisano wyszukiwanie w sieci WWW, w tym analizę struktur odsyłaczy (punkt 27.7.3) i algorytmy tworzenia rankingów wyszukiwania (np. PageRank i HITS). W końcowej części pokrótce opisaliśmy bieżące trendy, w tym wyszukiwanie fasetowe, wyszukiwanie społecznościowe i wyszukiwanie informacji w dialogach. Ponadto zaprezentowaliśmy modelowanie probabilistyczne tematów dokumentów i popularny algorytm LDA. Rozdział zakończyliśmy omówieniem systemów odpowiadania na pytania (punkt 27.7.5), które zyskują dużą popularność i są stosowane w narzędziach takich jak Siri Apple'a i Cortana Microsoftu.

Ten rozdział to tylko wprowadzenie do obszernej dziedziny. Zainteresowani Czytelnicy powinni zapoznać się z bibliografią z końca rozdziału, gdzie znajdują specjalistyczne teksty poświęcone WI i wyszukiwarkom.

## Pytania powtórkowe

- 27.1. Czym są dane strukturalne, a czym dane niestrukturalne? Podaj znane Ci z doświadczenia przykłady danych obu tych rodzajów.
- 27.2. Podaj ogólną definicję *wyszukiwania informacji* (WI). Co obejmuje wyszukiwanie informacji w kontekście danych dostępnych w sieci WWW?
- 27.3. Omów rodzaje danych i typy użytkowników powiązane z dzisiejszymi systemami WI.
- 27.4. Czym jest wyszukiwanie *nawigacyjne, informacyjne i transakcyjne*?
- 27.5. Podaj dwa podstawowe tryby interakcji z systemem WI. Opisz je i przedstaw przykłady.
- 27.6. Wyjaśnij wspomniane w tabeli 27.1 podstawowe różnice między bazami danych a systemami WI.
- 27.7. Opisz główne komponenty systemu WI przedstawione na rysunku 27.1.
- 27.8. Czym są biblioteki cyfrowe? Jakiego rodzaju dane zwykle się w nich znajdują?
- 27.9. Podaj nazwy kilku bibliotek cyfrowych, z jakich korzystałeś. Co zawierają te biblioteki i z jakiego okresu dane zawierają?

- 27.10. Przedstaw krótką historię WI i wymień przełomowe osiągnięcia w tej dziedzinie.
- 27.11. Czym jest model logiczny w WI? Jakie są jego ograniczenia?
- 27.12. Czym jest model przestrzeni wektorowej w WI? Jak tworzony jest wektor reprezentujący dokument?
- 27.13. Zdefiniuj sposób określania wag słów kluczowych w dokumencie za pomocą metody TF-IDF. Dlaczego w wadze pojęcia ważne jest uwzględnienie IDF?
- 27.14. Czym są modele probabilistyczny i semantyczny w WI?
- 27.15. Zdefiniuj *czułość* i *precyzję* w systemach WI.
- 27.16. Zdefiniuj *precyzję* i *czułość* dla pozycji *i* w rankingu wyników.
- 27.17. Jak zdefiniowana jest miara F w WI? W jaki sposób określa ona zarówno precyzję, jak i czułość?
- 27.18. Podaj różne rodzaje zapytań w systemie WI. Opisz każdy z nich z przykładami.
- 27.19. Wymień sposoby przetwarzania zapytań do wyszukiwania fraz i z określeniem odległości słów.
- 27.20. Opisz szczegółowy proces WI przedstawiony na rysunku 27.2.
- 27.21. Czym są usuwanie słów pomijanych i stemming? Dlaczego te procesy są niezbędne do lepszego wyszukiwania informacji?
- 27.22. Czym jest tezaurus? W jaki sposób przydaje się w WI?
- 27.23. Czym jest wydobywanie informacji? Podaj różne sposoby wydobywania informacji ze strukturalnego tekstu.
- 27.24. Czym jest leksykon w systemach WI? Jaką rolę odgrywa w indeksowaniu dokumentów?
- 27.25. Utwórz pięć dokumentów zawierających po ok. trzech zdań każdy. Dokumenty powinny obejmować podobną treść. Zbuduj indeks odwrócony z wszystkimi ważnymi tematami (słowami kluczowymi) z tych dokumentów.
- 27.26. Opisz proces generowania wyników wyszukiwania z użyciem indeksu odwróconego.
- 27.27. Zdefiniuj *informacje zwrotne na temat adekwatności*.
- 27.28. Opisz trzy rodzaje omówionych w tym rozdziale analiz użytkowania witryn.
- 27.29. Wymień ważne zadania związane z analizowaniem treści w sieci WWW. Opisz w kilku zdaniach każde z tych zadań.
- 27.30. Jakie trzy wykorzystujące agenty metody analiz treści w sieci WWW wymieniono w tym rozdziale?
- 27.31. Czym jest oparte na bazach danych analizowanie treści w sieci WWW? Czym są internetowe systemy obsługi zapytań?
- 27.32. Podaj popularne algorytmy tworzenia rankingów i określania znaczenia stron WWW. Który algorytm został zaproponowany przez założycieli firmy Google?
- 27.33. Na jakiej podstawowej zasadzie działa algorytm PageRank?
- 27.34. Czym są koncentratory i strony autorytatywne? W jaki sposób są one wykorzystywane w algorytmie HITS?
- 27.35. Czego możesz się dowiedzieć z analiz użytkowania witryn? Jakie dane są generowane w ramach takich analiz?

- 27.36. Jakie techniki eksploracji są często stosowane do danych o użytkowaniu witryn? Podaj przykłady.
- 27.37. Jakie są zastosowania eksploracji danych o użytkowaniu witryn?
- 27.38. Czym jest adekwatność wyszukiwania? Jak jest ustalana?
- 27.39. Zdefiniuj *wyszukiwanie fasetowe*. Wymyśl zestaw fasetów dla bazy danych dotyczącej budynków różnego rodzaju. Dwa przykładowe fasety to „wartość lub cena budynku” i „typ budynku (mieszkalny, biurowy, magazyn, fabryka itd.)”.
- 27.40. Czym jest wyszukiwanie społecznościowe? Co obejmuje kolektywne wyszukiwanie społecznościowe?
- 27.41. Zdefiniuj i objaśnij pojęcie *wyszukiwanie informacji w dialogach*.
- 27.42. Zdefiniuj *modelowanie tematów*.
- 27.43. Jak działają systemy odpowiadania na pytania?

## Wybrane publikacje

Wyszukiwanie informacji i pokrewne technologie to dziedziny, w których w środowiskach komercyjnym i akademickim stale prowadzone są badania i pojawiają się nowinki. Dostępnych jest wiele książek na temat WI, gdzie znajdziesz szczegółowe omówienie zagadnień pokrótce wprowadzonych w tym rozdziale. Książka *Search Engines: Information Retrieval in Practice* Crofta, Metzlera i Strohmanna (2009) zawiera praktyczny przegląd zagadnień i zasad z obszaru wyszukiwarek. *Introduction to Information Retrieval* Manninga, Raghavana i Schutzego (2008) to miarodajna pozycja na temat wyszukiwania informacji. Innym wprowadzeniem do WI jest książka *Modern Information Retrieval* — autorzy: Ricardo Baeza-Yates i Berthier Ribeiro-Neto (1999). Zawiera ona szczegółowe omówienie różnych aspektów technologii WI. Klasyczne pozycje Geralda Saltona (1968) i van Rijsbergena (1979) zawierają doskonały opis podstawowych badań w dziedzinie WI do końca lat 60. Salton przedstawia też model przestrzeni wektorowej w kontekście WI. Manning i Schutze (1999) prezentują wyczerpujące podsumowanie technologii z obszaru języka naturalnego i wstępnego przetwarzania tekstu. *Interactive Information Retrieval in Digital Environments* autorstwa Xie (2008) przedstawia godne uwagi, skoncentrowane na użytkownikach podejście do WI. Książka *Managing Gigabytes* Wittena, Moffata i Bella (1999) zawiera szczegółowe omówienie technik indeksowania. W książce *TREC* Voorhees i Harman (2005) przedstawiają opis kolekcji testowych i procedur oceny w kontekście konkursu TREC.

Broder (2002) dzieli zapytania w sieci WWW na trzy różne kategorie: nawigacyjne, informacyjne i transakcyjne. Przedstawia też szczegółową taksonomię wyszukiwania w sieci WWW. Covi i Kling (1996) wyjaśniają ogólną definicję bibliotek cyfrowych i omawiają organizacyjne aspekty skutecznego korzystania z takich bibliotek. Luhn (1957) wykonał w latach 50. w firmie IBM przełomowe prace nad automatycznym indeksowaniem i analityką biznesową. System SMART (Salton i in., 1993), opracowany na Uniwersytecie Cornella, był jednym z pierwszych zaawansowanych systemów WI, w których używano w pełni zautomatyzowanego indeksowania pojęć, grupowania hierarchicznego i rankingów dokumentów na podstawie podobieństwa do zapytania. W systemie SMART dokumenty i zapytania były reprezentowane za pomocą wektorów z ważonymi pojęciami zgodnie z modelem przestrzeni wektorowej.



Porterowi (1980) przypisuje się opracowanie słabych i silnych algorytmów stemmingu, które stały się standardem. Robertson (1997) opracował zaawansowany schemat wag w systemie Okapi z City University of London; system ten stał się bardzo popularny w konkursach TREC. Lenat (1995) uruchomił w latach 80. projekt Cyc, który miał dodać logikę formalną i bazy wiedzy do systemów przetwarzania informacji. Prace nad tezaurem WordNet były kontynuowane w latach 90. i wciąż trwają. Zagadnienia i zasady związane z WordNetem zostały poruszone w książce Fellbauma (1998). Rocchio (1971) opisuje algorytm pobierania informacji zwrotnych na temat adekwatności. Algorytm ten został też przedstawiony w książce Saltona (1971) *The SMART Retrieval System — Experiments in Automatic Document Processing*.

Abiteboul, Buneman i Suci (1999) przedstawiają rozbudowane omówienie danych w sieci WWW. W ich książce nacisk położony jest na dane półstrukturalne. Atzeni i Mendelzon (2000) napisali w magazynie „VLDB” artykuł redakcyjny na temat baz danych i sieci WWW. Atzeni i in. (2002) proponują modele i zmiany na potrzeby danych w sieci WWW. Abiteboul i in. (1997) omawiają język zapytań Lord służący do zarządzania danymi półstrukturalnymi.

Praca Chakrabartiego (2002) to doskonała książka na temat wykrywania wiedzy w sieci WWW. Pozycja Liu (2006) składa się z kilku części, a w każdej z nich znajduje się obszerny przegląd zagadnień związanych z analizami danych w sieci WWW i ich zastosowaniami. Świetne artykuły przeglądowe na temat analiz w sieci WWW opublikowali Kosala i Blockeel (2000) oraz Liu i in. (2004). Etzioni (1996) prezentuje wartościowe wprowadzenie do eksploracji danych w sieci WWW oraz opisuje zadania i problemy związane z tą dziedziną. Doskonałym przeglądem problemów, technik i nowinek z obszaru wyszukiwania w kontekście treści stron WWW i analiz użytkowania witryn jest praca Colleya i in. (1997). Colley (2003) koncentruje się na eksploracji wzorców użytkowania witryn z wykorzystaniem struktury sieci WWW. Spiliopoulou (2000) szczegółowo omawia analizy użytkowania witryn. Eksplorację danych w sieci WWW na podstawie struktury stron opisano w pracach Madrii i in. (1999) oraz Chakrabortiego i in. (1999). Algorytmy obliczania rankingu stron WWW zostały zaprezentowane przez Page’a i in. (1999), który opisał słynny algorytm PageRank, oraz Kleinberga (1998), który przedstawił algorytm HITS.

Harth, Hose i Schenkel (2014) prezentują techniki obsługi zapytań o dane z odsyłaczami w sieci WWW i zarządzania takimi danymi. Przedstawiają też potencjał takich technik w obszarze badań i zastosowań komercyjnych. Ferrucci i in. (2010) dość szczegółowo opisują technologie odpowiadania na pytania. Ferrucci opracował system Watson w firmie IBM. Bikel i Zitouni (2012) przedstawiają obszerny podręcznik tworzenia stabilnych i precyzyjnych wielojęzycznych systemów przetwarzania języka naturalnego. Blei, Ng i Jordan (2003) opisują przegląd modelowania tematów i algorytmu LDA. Szczegółowe i praktyczne omówienie technologii Lucene i Solr znajdziesz w oczekującej na publikację książce Moczara (2015).





## Elementy eksploracji danych

Przez ostatnich kilka dekad wiele organizacji wygenerowało ogromne ilości danych reprezentowanych w sposób umożliwiający przetwarzanie komputerowe w postaci plików i baz danych. Do przetwarzania tych danych możemy wykorzystywać istniejące technologie baz danych, które obsługują takie języki baz danych jak SQL. SQL jest jednak językiem strukturalny, którego autorzy przyjęli, że użytkownik dysponuje niezbędną wiedzą na temat struktury bazy danych. SQL obsługuje operacje algebry relacyjnej, za pomocą których użytkownik może selekcjonować interesujące go wiersze i kolumny tabel danych lub złączać (w oparciu o wartości wspólnych pól) powiązane ze sobą informacje przechowywane w różnych tabelach. W następnym rozdziale przekonamy się, że dopiero *technologia hurtowni danych* zapewnia wiele funkcji, w tym między innymi konsolidację, agregację i generowanie podsumowań danych. Hurtownie danych umożliwiają przeglądanie tych samych informacji w wielu wymiarach. W tym rozdziale skierujemy naszą uwagę w stronę innego, bardzo popularnego ostatnio obszaru badań i rozwoju, jakim jest eksploracja danych (ang. *data mining*). Jak nietrudno się domyślić, termin **eksploracja danych** odwołuje się do odkrywania nowych informacji (w postaci wzorców lub reguł) w oparciu o istniejące, duże ilości danych. Możliwości stosowania technik eksploracji danych w praktyce wymagają efektywnych metod przeszukiwania ogromnych plików lub baz danych. Choć relacyjne SZBD udostępniają niektóre mechanizmy eksploracji danych, eksploracja *nie jest* obecnie dobrze zintegrowana z systemami zarządzania bazami danych. Świat biznesu jest dziś zafascynowany możliwościami eksploracji danych, a dziedzina ta jest często nazywana **analitiką biznesową** lub **analitiką danych**.

W tym rozdziale omówimy krótko obecny stan rozwoju szeroko rozumianej technologii eksploracji danych, w której wykorzystuje się techniki znane z takich dziedzin jak uczenie maszynowe, statystyka, sieci neuronowe oraz algorytmy genetyczne. Skupimy się na cechach charakterystycznych odkrywanych informacji, rodzajach problemów napotykanych podczas podejmowania prób eksploracji zawartości baz danych oraz zastosowaniach tej dziedziny. Przeanalizujemy także sporą liczbę dostępnych obecnie komercyjnych narzędzi tego typu (patrz punkt 28.7) i opiszemy kilka nierozwiązanych problemów badawczych, które opóźniają rozwój technologii eksploracji danych.

## 28.1. Przegląd technologii eksploracji danych

W raportach niezależnych ośrodków, w tym w popularnym raporcie Gartnera<sup>1</sup>, eksploracja danych zostało uznane za jedną z najbardziej obiecujących technologii niedalekiej przyszłości. W tym podrozdziale przedstawimy związki eksploracji danych z nieco szerszym pojęciem *odkrywania wiedzy* i przedstawimy przykład, który dobrze ilustruje różnice pomiędzy obiema technologiami.

### 28.1.1. Eksploracja danych kontra hurtownie danych

Celem technologii hurtowni danych (patrz rozdział 29.) jest wspomaganie podejmowania decyzji w oparciu o przechowywane dane. Techniki eksploracji danych mogą być stosowane w połączeniu z hurtowniami danych — takie rozwiązanie może pomóc w podejmowaniu pewnych typów decyzji. Eksploracja danych może dotyczyć także operacyjnych baz danych, gdzie do odpowiednich działań można wykorzystywać pojedyncze transakcje. Aby zwiększyć efektywność tych technik, należy jednak użyć hurtowni danych z odpowiednio zagregowanymi lub podsumowanymi zbiorami informacji. Eksploracja danych może pomóc w odnajdywaniu nowych wzorców, których odkrycie nie byłoby możliwe wyłącznie w oparciu o wykonywanie zapytań czy przetwarzania danych lub metadanych składowanych w hurtowni. Ewentualne zastosowanie tego typu mechanizmów powinno więc być brane pod uwagę już w fazie projektowania hurtowni danych. Możliwości stosowania technik eksploracji danych są uzależnione także od odpowiednich narzędzi, które powinny ułatwiać ich wykorzystywanie do badania informacji składowanych w hurtowniach danych. W praktyce, w przypadku ogromnych systemów zawierających terabajty danych, skuteczne wykorzystywanie aplikacji eksploracji danych będzie zależało przede wszystkim od użytej konstrukcji hurtowni danych.

### 28.1.2. Eksploracja danych jako część procesu odkrywania wiedzy

Procesy **odkrywania wiedzy w bazach danych** (ang. *Knowledge Discovery in Databases*, w skrócie *KDD*) obejmują zwykle działania bardziej złożone niż tylko stosowanie technik eksploracji danych. Na proces odkrywania wiedzy składa się aż sześć następujących etapów<sup>2</sup>: selekcja danych, oczyszczanie danych, wzbogacanie danych, transformacja lub kodowanie danych, eksploracja danych oraz raportowanie i prezentowanie odkrytych informacji.

Przykładowo, przeanalizujmy transakcyjną bazę danych utrzymywaną przez wyspecjalizowanego dystrybutora dóbr konsumenckich. Przypuśćmy, że dane klienta obejmują jego nazwisko, kod pocztowy, numer telefonu, datę zakupu, kod produktu, cenę, ilość oraz łączną wartość zamówienia. Zastosowanie procesu odkrywania wiedzy dla tak skonstruowanej bazy danych o klientach może doprowadzić do wielu ciekawych wniosków. W czasie *selekcji danych* można wybrać informacje na temat konkretnych produktów lub kategorii produktów, sklepów rozlokowanych w określonym mieście lub regionie kraju

---

<sup>1</sup> Raport Gartnera to jedna z wielu publikacji przeglądowych na temat technologii. Takie publikacje umożliwiają menedżerom w korporacjach omawianie i wybór technologii eksploracji danych.

<sup>2</sup> Znaczna część naszej analizy tego zagadnienia opiera się na książce Adriaansa i Zantingego (1996).

itp. Zastosowany w dalszej kolejności proces *oczyszczania danych* może poprawić nieprawidłowe kody pocztowe lub wyeliminować rekordy z błędnymi numerami telefonicznymi. *Wzbogacanie* danych polega zwykle na ich rozszerzaniu w oparciu o dodatkowe źródła informacji. Przykładowo, mając dane nazwiska i numery telefonów klientów, można podjąć próbę uzyskania takich informacji jak wiek, zarobki czy zdolność kredytowa i dołączyć je do odpowiednich rekordów. *Transformacja* i kodowanie danych mogą mieć na celu ograniczenie ilości informacji przeznaczonych do dalszych analiz. Przykładowo, kody produktów można pogrupować według kategorii na sprzęt audio, sprzęt wideo, materiały eksploatacyjne, elektroniczne gadżety, kamery, akcesoria itp. Kody pocztowe można agregować do postaci reprezentującej okręgi pocztowe i regiony kraju, natomiast dochody klientów można podzielić na odpowiednie przedziały. Na rysunku 29.1 pokażemy, że proces ETL danych powinien poprzedzać proces tworzenia hurtowni danych. Jeśli więc techniki eksploracji danych mają być stosowane dla istniejącej hurtowni danych sieci sklepów, możemy oczekiwać, że oczyszczanie i porządkowanie informacji zostało już wykonane. Dopiero po wykonaniu wszystkich opisanych przed chwilą czynności przygotowawczych można przystąpić do zastosowania technik *eksploracji danych*, które mają posłużyć do odkrycia rozmaitych reguł i wzorców.

Uzyskany wynik eksploracji danych może prowadzić do odkrycia następujących typów *nowych* informacji:

- **Reguły asocjacyjne** — przykładowo, za każdym razem, gdy klient kupuje sprzęt wideo, kupuje także inne elektroniczne gadżety.
- **Wzorce sekwencyjne** — przykładowo, przypuśćmy, że klient kupuje aparat i w ciągu trzech miesięcy wraca do sklepu po fotograficzne materiały eksploatacyjne, po czym (w ciągu następnych sześciu miesięcy) bardzo często dokupuje do aparatu jakieś akcesoria. Obserwacja tych zachowań umożliwia zdefiniowanie sekwencyjnego wzorca transakcji. Klient, który kupuje więcej niż dwa produkty w okresie obniżonej sprzedaży, z dużym prawdopodobieństwem zjawi się w sklepie przynajmniej raz w okresie przedświątecznych zakupów.
- **Drzewa klasyfikacyjne** — przykładowo, klientów można sklasyfikować według częstotliwości wizyt w sklepie, sposobów zapłaty, wartości zakupów lub podobieństwa nabywanych towarów; na podstawie tak skonstruowanych klas można wyznaczyć pewne zależności statystyczne.

Przedstawiony przykład sklepu detalicznego pokazuje, że proces eksploracji danych musi być poprzedzony ich przygotowaniem — dopiero wtedy będzie możliwe znalezienie przydatnych informacji, które być może będą miały bezpośredni wpływ na podejmowane decyzje biznesowe.

Wyniki procesu eksploracji danych mogą być prezentowane w rozmaitych formatach: list, prezentacji graficznych, tabel podsumowujących lub wizualizacji.

### 28.1.3. Cele eksploracji danych i odkrywania wiedzy

Eksploracja danych zwykle jest przeprowadzana z myślą o osiągnięciu z góry określonych celów lub znalezieniu nowych zastosowań dla istniejących zasobów. Ogólnie rzecz biorąc, cele te można podzielić na następujące kategorie: przewidywanie, identyfikacja, klasyfikacja oraz optymalizacja.

- **Przewidywanie.** Eksploracja danych może pokazać, jak pewne atrybuty w ramach istniejących danych będą się zmieniały w przyszłości. Przykładem takiego zastosowania technik odkrywania wiedzy jest analiza transakcji w sklepie detalicznym pod kątem przyszłych zakupów w warunkach ewentualnych promocji, wysokości sprzedaży, którą dany sklep może wygenerować w określonym okresie, oraz ewentualnych zysków i strat spowodowanych wycofaniem ze sprzedaży określonej linii produktów. W tego typu zastosowaniach techniki eksploracji danych są wykorzystywane w ścisłym połączeniu z logiką biznesową. W kontekście badań naukowych podobne techniki można wykorzystać do analizy fal sejsmicznych celem przewidywania z dużym prawdopodobieństwem trzęsień ziemi.
- **Identyfikacja.** Znalezione wzorce danych mogą być wykorzystywane do wykrywania istnienia określonych elementów, zdarzeń lub działań. Przykładowo, włamywacze próbujący złamać zabezpieczenia systemu komputerowego mogą być zidentyfikowani dzięki analizie uruchamianych programów, operacji dostępu do plików oraz generowanego obciążenia procesora. W zastosowaniach biologicznych istnienie określonego genu można zweryfikować w oparciu o analizę odpowiednich ciągów symboli nukleotydów w sekwencji DNA. Pewną formą identyfikacji jest *uwierzytelnianie*. Techniki uwierzytelniania pozwalają stwierdzić, czy dana osoba faktycznie jest użytkownikiem, za którego się podaje, i należy do właściwej klasy użytkowników — stosowanie tych metod wymaga porównywania parametrów, obrazów lub sygnałów z zawartością bazy danych.
- **Klasyfikacja.** Eksploracja danych może się wiązać z takim partycjonowaniem informacji, które umożliwi identyfikowanie rozmaitych klas lub kategorii w oparciu o wybrane kombinacje parametrów. Przykładowo, klientów supermarketu można skategoryzować na osoby szukające promocji, osoby robiące zakupy w pośpiechu, lojalnych klientów przywiązanych do danej sieci handlowej, klientów zwracających uwagę na markę towaru oraz klientów przychodzących do sklepu bardzo rzadko. Podobna klasyfikacja może być wykorzystywana w różnych analizach transakcji zakupów dokonywanych przez klientów w przeszłości. Niekiedy kategorie stworzone według ogólnej wiedzy dziedzinowej mogą występować w roli danych wejściowych dla procesu dekompozycji problemu eksploracji i znacznie ten proces uprościć. Przykładowo, zdrowa żywność, popularne produkty podawane na imprezach oraz artykuły wykorzystywane do przygotowywania szkolnych obiadów tworzą różne kategorie działalności biznesowej supermarketu. Warto więc poddać osobnej analizie związki występujące wewnątrz poszczególnych kategorii i pomiędzy nimi. Taka kategoryzacja może służyć odpowiedniemu kodowaniu informacji, zanim jeszcze staną się przedmiotem dalszych analiz w procesie eksploracji danych.
- **Optymalizacja.** Jednym z potencjalnych celów eksploracji danych jest optymalizacja wykorzystania ograniczonych zasobów, w tym czasu, przestrzeni, pieniędzy czy materiałów, oraz maksymalizacja wartości zmiennych wyjściowych, takich jak wysokość sprzedaży lub zysk przy określonym zbiorze ograniczeń. Można przyjąć, że techniki eksploracji danych w pewnym sensie przypominają funkcje celu, które wykorzystuje się podczas rozwiązywania problemów badawczych związanych z optymalizacją wybranych parametrów w określonych warunkach.

Pojęcie *eksploracji danych* zyskało ogromną popularność i jest wykorzystywane w szerokim znaczeniu. Termin ten obejmuje nie tylko analizę statystyczną i optymalizację, ale także uczenie maszynowe. Nie istnieje wyraźne rozróżnienie pomiędzy tymi dziedzinami, zatem szczegółowe omawianie wszelkich możliwych aplikacji zaliczanych do technologii eksploracji danych wykracza poza zakres tematyczny tej książki. Czytelnicy zainteresowani głębszym poznaniem tej tematyki powinni skorzystać ze specjalistycznych książek.

### 28.1.4. Rodzaje wiedzy odkrywanej w procesie eksploracji danych

Pojęcie *wiedzy* jest powszechnie interpretowane jako określenie pewnego stopnia inteligencji. Istnieje jednak różnica pomiędzy surowymi (nieprzetworzonymi) danymi, właściwymi informacjami oraz wiedzą — przejście do kolejnej postaci wymaga oczywiście dodatkowego przetworzenia. Wiedza jest często klasyfikowana według podziału na wiedzę indukcyjną i dedukcyjną. **Wiedza dedukcyjna** ma postać nowych informacji otrzymanych przez zastosowanie wobec danych wejściowych *zdefiniowanych wcześniej* logicznych reguł wnioskowania. Eksploracja danych może być źródłem **wiedzy indukcyjnej**, która obejmuje reguły i wzorce odkryte w procesie analizy istniejących danych. Wiedza może być reprezentowana w wielu postaciach — w formacie niestrukturalnym mogą to być reguły i wyrażenia rachunku zdań; natomiast w formacie strukturalnym reprezentacja wiedzy może mieć postać drzew decyzyjnych, sieci semantycznych, sieci neuronowych lub hierarchii klas. Wiedza odkrywana w procesie eksploracji danych najczęściej jest opisywana według następujących sposobów:

- **Reguły asocjacyjne** — Takie reguły wskazują na korelację pomiędzy obecnością pewnego zbioru elementów a innym zakresem wartości przechowywanych w innym zbiorze zmiennych. Przykładowo, (1) kobieta kupująca w sklepie torebkę prawdopodobnie kupi także buty; (2) zdjęcie rentgenowskie mające cechy *a* i *b* najprawdopodobniej będzie wskazywało na własność *c* badanego pacjenta.
- **Hierarchie klasyfikacji** — Celem tej techniki jest przetworzenie istniejącego zbioru zdarzeń lub transakcji i stworzenie na jego podstawie hierarchii klas. Przykładowo, (1) populację można podzielić na pięć klas według przedziałów zdolności kredytowej określanej w oparciu o historię spłat wcześniejszych kredytów i pożyczek. (2) Istnieje możliwość opracowania modelu dla czynników określających celowość umieszczania sklepów w określonych lokalizacjach w skali od 1 do 10. (3) Fundusze inwestycyjne można sklasyfikować według takich danych jak wzrost notowań, przychód oraz poziom stabilności.
- **Wzorce sekwencyjne** — Ta technika sprowadza się do poszukiwania sekwencji zarejestrowanych działań lub zdarzeń. Przykładowo, jeśli u pacjenta, który rok temu przeszedł kardiologiczną operację wszczepienia *bypassów* z powodu zablokowanych arterii i aneurysmu aorty, zostanie wykryte duże stężenie krwi w moczu, najprawdopodobniej u takiego pacjenta w ciągu 18 miesięcy zostanie stwierdzona niewydolność nerek. Wykrywanie wzorców sekwencyjnych jest równoważne odkrywaniu związków pomiędzy zdarzeniami, dla których istnieją wyraźne relacje czasowe.

- **Wzorce w ramach szeregów czasowych** — Istotne podobieństwa często można wykryć wśród pozycji w **szeregach czasowych** mających postać sekwencji danych gromadzonych w regularnych odstępach czasu. Mogą to być np. informacje na temat dziennej sprzedaży lub sesyjny kurs zamknięcia dla akcji określonych spółek. Przykładowo, (1) dla akcji przedsiębiorstwa użyteczności publicznej ABC Energia i firmy finansowej XYZ Inwestycje odkryto ten sam wzorec cen zamknięcia akcji w roku 2014. (2) Dwa produkty charakteryzują się tym samym wzorcem sprzedaży w okresie letnim, ale zupełnie innym wzorcem w okresie zimowym. (3) Wzorec magnetycznych wiatrów słonecznych może być wykorzystywany do przewidywania warunków atmosferycznych na Ziemi.
- **Grupowanie (łączenie w klastry)** — Dla danej na wejściu populacji zdarzeń lub elementów można zastosować technikę partycjonowania (segmentacji) do zbiorów skupiających „podobne” elementy. Przykładowo, (1) cała populacja danych o leczeniu chorób może zostać podzielona na grupy według zaobserwowanych podobieństw skutków ubocznych podjętej terapii. (2) Populację dorosłych obywateli Polski można skategoryzować w pięć grup obejmujących klientów od *najbardziej zainteresowanych zakupem* do *najmniej zainteresowanych zakupem* nowego produktu. (3) Dostęp użytkowników do pewnego zbioru dokumentów na witrynie internetowej (np. utrzymywanej przez cyfrową bibliotekę) można poddać analizie uwzględniającej zdefiniowane dla tych dokumentów słowa kluczowe — w ten sposób być może uda się odkryć klastry lub kategorie użytkowników.

W przypadku większości zastosowań technik eksploracji danych oczekiwana wiedza ma postać kombinacji kilku spośród wymienionych powyżej typów. Każdy z tych typów szczegółowo omówimy w kolejnych podrozdziałach.

## 28.2. Reguły asocjacyjne

### 28.2.1. Model koszyka klienta supermarketu, poziom wsparcia i poziom ufności

Jedną z głównych technologii zaliczanych do eksploracji danych wiąże się z odkrywaniem reguł asocjacyjnych. Stosując tę technologię, należy przyjąć, że baza danych jest zbiorem transakcji, z których każda dotyczy określonego zbioru elementów. Typowym przykładem obszaru stosowania tych technik są **dane na temat koszyka klienta supermarketu**. Każdy koszyk klienta supermarketu odpowiada zbiorowi elementów, które dany klient kupił podczas jednej wizyty w supermarkecie. Spróbujmy przeanalizować cztery losowo dobrane transakcje tego rodzaju (patrz rysunek 28.1).

**Reguła asocjacyjna** ma postać  $X \Rightarrow Y$ , gdzie  $X = \{x_1, x_2, \dots, x_n\}$  oraz  $Y = \{y_1, y_2, \dots, y_m\}$  są takimi zbiorami elementów, że dla każdego  $i$  oraz  $j$  zmienne  $x_i$  oraz  $y_j$  są różnymi elementami. Tak wyrażona asocjacja określa, że jeśli dany klient kupił towary ze zbioru  $X$ , najprawdopodobniej kupi także towary ze zbioru  $Y$ . Ogólnie, każda reguła asocjacyjna ma postać  $LS$  (lewa strona)  $\Rightarrow PS$  (prawa strona), gdzie  $LS$  i  $PS$  są zbiorami elementów. Zbiór  $LS \cup PS$



Identyfikator-transakcji	Czas	Kupione-towary
101	6:35	mleko, chleb, ciastka, sok
792	7:38	mleko, sok
1130	8:05	mleko, jajka
1735	8:40	chleb, ciastka, kawa

RYSUNEK 28.1. Przykładowe transakcje modelu koszyka klienta supermarketu

jest nazywany **zbiorem elementów** (ang. *itemset*), ponieważ skupia elementy kupowane przez klientów. Aby reguła asocjacyjna stanowiła interesujące źródło informacji dla analityka stosującego techniki eksploracji danych, musi spełniać określone warunki wyrażane za pomocą odpowiednich mierników korzyści. Do dwóch najbardziej popularnych mierników tego typu należą poziom wsparcia i poziom ufności.

**Poziom wsparcia** (ang. *support*) dla reguły w postaci  $LS \Rightarrow PS$  jest miernikiem definiowanym względem zbioru elementów — określa częstotliwość wystąpień tego zbioru w bazie danych. Oznacza to, że poziom wsparcia (wyrażany w procentach) jest stosunkiem liczby transakcji zawierających wszystkie elementy łącznego zbioru elementów ( $LS \cup RS$ ) do liczby wszystkich transakcji zarejestrowanych w bazie danych. Jeśli poziom wsparcia jest niski, oznacza to, że nie ma jednoznacznych dowodów na łączne występowanie elementów zbioru  $LS \cup RS$ , ponieważ zbiór ten występuje tylko w stosunkowo niewielkiej części wszystkich transakcji. Pojęcie poziomu wsparcia jest niekiedy stosowane zamiennie z pojęciem *poziomu dominacji* (ang. *prevalence*).

**Poziom ufności** (ang. *confidence*) jest własnością implikacji reprezentowanej przez regułę asocjacyjną. Poziom ufności reguły  $LS \Rightarrow PS$  jest równy ilorazowi poziomu wsparcia sumy zbiorów  $LS \cup RS$  przez poziom wsparcia zbioru  $LS$ . Można przyjąć, że poziom ufności jest prawdopodobieństwem tego, że elementy tworzące zbiór  $RS$  zostaną kupione przez danego klienta pod warunkiem, że klient ten kupi także elementy tworzące zbiór  $LS$ . Pojęcie poziomu ufności jest niekiedy stosowane zamiennie z pojęciem *sily* (ang. *strength*) reguły asocjacyjnej.

Aby lepiej zrozumieć praktyczne znaczenie mierników poziomu wsparcia i poziomu ufności, przyjrzyjmy się następującej parze przykładowych reguł:  $Mleko \Rightarrow Sok$  oraz  $Chleb \Rightarrow Sok$ . Jeśli raz jeszcze spojrzymy na cztery przykładowe transakcje z rysunku 28.1, przekonamy się, że poziom wsparcia zbioru {Mleko, Sok} wynosi 50%, natomiast poziom wsparcia zbioru {Chleb, Sok} wynosi tylko 25%. Poziom ufności implikacji (reguły)  $Mleko \Rightarrow Sok$  jest równy 66,7% (ponieważ spośród trzech transakcji zakupu mleka tylko dwie transakcje obejmowały także zakup soku) zaś poziom ufności implikacji  $Chleb \Rightarrow Sok$  wynosi 50% (ponieważ tylko jedna z dwóch transakcji zakupu chleba obejmuje także zakup soku).

Jak widać, poziom wsparcia nie musi iść w parze z poziomem ufności. Celem procesu eksploracji reguł asocjacyjnych jest więc wygenerowanie wszystkich możliwych implikacji tego typu, których poziom wsparcia i poziom ufności przekracza określone wcześniej wartości progowe. Oznacza to, że cały problem tworzenia tych reguł można rozbić na dwa główne podproblemy:

- (1) Wygenerowanie wszystkich zbiorów elementów, których poziom wsparcia przekracza określony próg. Zbiory te są nazywane **dużymi** (lub **częstymi**) **zbiorami elementów** (ang. *large (frequent) itemsets*). Warto pamiętać, że określenie *duży* odnosi się w tym przypadku do dużego wsparcia.

- (2) Dla każdego dużego zbioru elementów generowane są wszystkie reguły spełniające warunek odpowiednio wysokiego poziomu ufności — proces ten przebiega w sposób następujący: dla dużego zbioru elementów  $X$  i zbioru  $Y \subset X$ , niech  $Z = X - Y$ ; jeśli wówczas poziom wsparcia zbioru  $X$  podzielony przez poziom wsparcia zbioru  $Z$  jest większy od minimalnego poziomu ufności, implikacja  $Z \Rightarrow Y$  (czyli  $X - Y \Rightarrow Y$ ) jest uznawana za poprawną regułę asocjacyjną.

Generowanie reguł z wykorzystaniem wszystkich dużych zbiorów elementów i określonego z góry minimalnego poziomu wsparcia jest stosunkowo proste. Okazuje się jednak, że samo odkrycie wszystkich tych zbiorów wraz z wartościami ich poziomów wsparcia stanowi poważny problem w sytuacji, gdy liczność kompletnego zbioru elementów składowanych w bazie danych jest bardzo duża. Typowy supermarket oferuje swoim klientom tysiące towarów. Liczba różnych zbiorów elementów wynosi  $2^m$ , gdzie  $m$  jest liczbą elementów, zatem wyznaczenie poziomu wsparcia dla wszystkich możliwych zbiorów elementów jest bardzo kosztowne obliczeniowo. Aby ograniczyć przestrzeń przeszukiwania kombinatorycznego, algorytmy odkrywające reguły asocjacyjne wykorzystują następujące własności zbiorów elementów:

- Podzbiór dużego zbioru elementów także musi być duży (oznacza to, że dla każdego podzbioru dużego zbioru elementów musi być przekroczony określony próg poziomu wsparcia).
- Z powyższego warunku wynika, że nadzbiór małego zbioru elementów także jest mały (a więc nie może przekraczać określonego poziomu wsparcia).

Pierwsza własność jest nazywana **dolnym domknięciem**. Drugą cechą określa się mianem właściwości **antymonotoniczności**, która ułatwia ograniczanie przestrzeni poszukiwania potencjalnych rozwiązań. Oznacza to, że kiedy algorytm odkryje, że dany zbiór elementów jest mały (nie jest dużym zbiorem elementów), wówczas wynikiem każdej operacji rozszerzenia tego zbioru — polegającej na dodaniu do niego jednego lub więcej elementów — będzie także mały zbiór elementów.

## 28.2.2. Algorytm Apriori

Pierwszym algorytmem o właściwościach dolnego domknięcia i antymonotoniczności był **algorytm Apriori** (algorytm 28.1).

Działanie algorytmu 28.1 zilustrujemy na przykładzie danych o transakcjach przedstawionych na rysunku 28.1 i przy założeniu, że minimalny poziom wsparcia wynosi 0,5. Zbiór kandydujących jednoelementowych zbiorów obejmuje następujące produkty: {mleko, chleb, sok, ciastka, jajka, kawa}; poziom wsparcia tych produktów wynosi odpowiednio: 0,75; 0,5; 0,5; 0,5; 0,25 oraz 0,25. Pierwsze cztery elementy kwalifikują się do zbioru  $L_1$ , ponieważ wyznaczony dla każdego z tych elementów poziom wsparcia jest większy lub równy wartości progowej 0,5. W pierwszej iteracji pętli powtarzaj-dopóki rozszerzamy jednoelementowy zbiór częsty w celu stworzenia kandydującego dwuelementowego zbioru częstego ( $C_2$ ). Zbiór ten składa się z następujących par elementów: {mleko, chleb}, {mleko, sok}, {chleb, sok}, {mleko, ciastka}, {chleb, ciastka} oraz {sok, ciastka}. Warto zwrócić uwagę na fakt, iż np. para {mleko, jajka} nie występuje w zbiorze  $C_2$ , ponieważ jednoelementowy zbiór {jajka} jest mały (zgodnie z własnością antymonotoniczności)

i nie występuje w zbiorze  $L_1$ . W procesie przeszukiwania zbioru transakcji obliczamy, że poziomy wsparcia dla sześciu zbiorów zawierających się w zbiorze  $C_2$  wynoszą odpowiednio: 0,25; 0,5; 0,25; 0,25; 0,5 oraz 0,25. Okazuje się, że tylko drugi i piąty zbiór dwuelementowy, czyli odpowiednio pary {mleko, sok} oraz {chleb, ciastka}, spełniają warunek poziomu wsparcia przekraczającego lub równego wartości progowej 0,5. Te dwa dwuelementowe zbiory można uznać za zbiory duże, a więc przypisać do zbioru  $L_2$ .

ALGORYTM 28.1. Algorytm Apriori do znajdowania częstych (dużych) zbiorów elementów

**Dane wejściowe:** baza danych  $D$  zawierająca informacje o  $m$  transakcjach oraz minimalny poziom wsparcia  $p\_min$  reprezentowany w postaci ułamka liczby  $m$ .

**Dane wyjściowe:** zbiory częstych zbiorów:  $L_1, L_2, \dots, L_k$

**Początek** /\* Kroki i instrukcje są numerowane w celu poprawy czytelności \*/

- (1) Dla każdego elementu  $i_1, i_2, \dots, i_n$  z osobna oblicz poziom wsparcia( $i_j$ ) równy liczbie wystąpień tego elementu ( $i_j$ ) podzielonej przez liczbę wszystkich transakcji ( $m$ ) — wyznaczenie jednego takiego poziomu będzie wymagało jednokrotnego przeszukania bazy danych i zliczenia wszystkich transakcji, w których występuje element  $i_j$ .
- (2) Zbiór  $C_1$  jest zbiorem kandydujących jednoelementowych częstych zbiorów zawierających odpowiednio elementy  $i_1, i_2, \dots, i_n$ .
- (3) Podzbiór elementów zbioru  $C_1$  zawierający element  $i_j$ , gdzie poziom wsparcia elementu  $i_j \geq p\_min$ , staje się podzbiorem zbioru  $L_1$  jednoelementowych częstych zbiorów.
- (4)  $k = 1$ ;  
     *koniec* = fałsz;  
     **powtarzaj**
- (5)  $L_{k+1}$  = (zbiór pusty);
- (6) Utwórz kandydujące  $(k+1)$ -elementowe zbiory częste (jako podzbiory zbioru  $C_{k+1}$ ) przez połączenie elementów zbioru  $L_k$  (zawierającego  $k-1$  elementów należących do obu zbiorów). W ten sposób, przez selektywne rozszerzanie o jeden element  $k$ -elementowych zbiorów częstych, powstają kandydujące  $(k+1)$ -elementowe zbiory częste.
- (7) Dodatkowo, rozważaj w roli potencjalnych składowych zbioru  $C_{k+1}$  tylko te  $k+1$  elementów, dla których każdy podzbiór z liczącością równą  $k$  występuje w zbiorze  $L_k$ .
- (8) Przeszukaj jednokrotnie bazę danych i oblicz poziom wsparcia dla każdego elementu zbioru  $C_{k+1}$ ; jeśli poziom wsparcia dla danego elementu zbioru  $C_{k+1} \geq p\_min$ , dodaj ten element do zbioru  $L_{k+1}$ .
- (9) Jeśli zbiór  $L_{k+1}$  jest pusty, przypisz zmiennej *koniec* wartość prawda;  
     w przeciwnym przypadku, przypisz zmiennej  $k$  wartość  $k+1$ .  
     **dopóki zmienna koniec ma wartość fałsz;**  
     **Koniec;**

W kolejnej iteracji pętli *powtarzaj-dopóki* konstruujemy kandydujące trójelementowe zbiory częste przez rozszerzanie zbiorów zawierających się w zbiorze  $L_2$  o dodatkowe elementy. Okazuje się jednak, że nie istnieje takie rozszerzenie zbiorów elementów zawierających się w zbiorze  $L_2$ , dla którego wszystkie dwuelementowe podzbiory zawierałyby się w zbiorze  $L_2$ . Rozważmy np. zbiór {mleko, sok, chleb}; dwuelementowy zbiór {mleko, chleb} nie zawiera się w  $L_2$ , zatem zbiór {mleko, sok, chleb} nie może być trójelementowym zbiorem częstym (ponieważ nie spełnia warunku dolnego domknięcia). W tym momencie algorytm kończy pracę ze zbiorem  $L_1$  w postaci {{mleko}, {chleb}, {sok}, {ciastka}} oraz zbiorem  $L_2$  równym {{mleko, sok}, {chleb, ciastka}}.

Z myślą o odkrywaniu reguł asocjacyjnych zaproponowano wiele innych algorytmów. Poszczególne rozwiązania różnią się przede wszystkim sposobem generowania kandydujących zbiorów elementów oraz metodami wyznaczania poziomów wsparcia dla tych zbiorów. Niektóre algorytmy wykorzystują takie struktury danych jak bitmapy i tzw. drzewa mieszające do przechowywania informacji na temat zbiorów elementów. Dotychczas zaproponowano wiele algorytmów wielokrotnie przeglądających bazę danych, ponieważ ogromna liczba potencjalnych zbiorów elementów równa  $2^m$  może uniemożliwić ustawienie wszystkich niezbędnych liczników podczas pojedynczego przeglądania bazy danych. W kolejnych punktach tego podrozdziału przeanalizujemy trzy udoskonalone (względem omówionego przed chwilą algorytmu Apriori) algorytmy eksploracji reguł asocjacyjnych: algorytm próbkujący, algorytm odkrywania częstych wzorców w drzewie oraz algorytm partycjonujący.

### 28.2.3. Algorytm próbkujący

Zasadnicza koncepcja **algorytmu próbkującego** (ang. *sampling algorithm*) opiera na wybieraniu z bazy danych transakcji niewielkich próbek, z których każda mieści się w pamięci operacyjnej komputera, i określaniu na ich podstawie częstych zbiorów elementów. Jeśli te częste zbiory elementów tworzą nadzbiory częstych zbiorów elementów dla całej bazy danych, wówczas możemy określić rzeczywisty częsty zbiór elementów, przeglądając resztę bazy danych i obliczając dokładny poziom wsparcia dla zbiorów elementów zawierających się w tym nadzbiorze. Nadzbiór częstych zbiorów elementów często można odkryć w oparciu o próbkę, stosując np. algorytm Apriori z obniżonym progiem minimalnego wsparcia.

W niektórych (rzadko spotykanych) przypadkach część częstych zbiorów elementów może nie zostać odkryta podczas pierwszego przeglądu bazy danych — wówczas niezbędne jest wykonanie pełnego przeglądu bazy po raz drugi. Decyzję o tym, czy jakiegokolwiek częste zbiory elementów zostały pominięte, należy podjąć w oparciu o tzw. *negatywną granicę* (ang. *negative border*). Własność negatywnej granicy odnosi się do zbioru  $S$  częstych zbiorów elementów oraz zbioru elementów  $I$ , i reprezentuje takie minimalne zbiory elementów zawierające się w zbiorze wszystkich podzbiorów zbioru  $I$ , które jednocześnie nie zawierają się w zbiorze  $S$ . Podstawowa zasada mówi, że negatywna granica zbioru częstych zbiorów elementów obejmuje najbliższe zbiory elementów, które także mogą być zbiorami częstymi. Przeanalizujemy przypadek, w którym zbiór  $X$  nie zawiera się w zbiorze częstych zbiorów elementów. Jeśli wszystkie podzbiory zbioru  $X$  zawierają się w zbiorze częstych zbiorów elementów, wówczas zbiór  $X$  będzie należał do negatywnej granicy.

Zilustrujemy powyższe rozważania prostym przykładem. Rozważmy zbiór elementów  $I = \{A, B, C, D, E\}$  i przyjmijmy, że połączone częste zbiory elementów o licznosciach równych od 1 do 3 mają postać:  $S = \{\{A\}, \{B\}, \{C\}, \{D\}, \{AB\}, \{AC\}, \{BC\}, \{AD\}, \{CD\}, \{ABC\}\}$ . Negatywną granicą jest w tym przypadku zbiór  $\{\{E\}, \{BD\}, \{ACD\}\}$ . Zbiór  $\{E\}$  jest jedynym jednoelementowym zbiorem, który nie zawiera się w zbiorze  $S$ ;  $\{BD\}$  jest jedynym dwuelementowym zbiorem, który (w przeciwieństwie do swoich jednoelementowych podzbiorów) nie zawiera się w zbiorze  $S$ ; natomiast zbiór  $\{ACD\}$  jest jedynym trójelementowym zbiorem, którego wszystkie jedno- i dwuelementowe podzbiory zawierają się w zbiorze  $S$ . Określenie negatywnej granicy jest w tym przypadku ważne z uwagi na konieczność wyznaczenia poziomu wsparcia dla należących do niej zbiorów elementów — tylko w ten sposób możemy się upewnić, że żadne duże zbiory elementów nie zostały pominięte podczas analizowania próbkowanych danych.

Poziom wsparcia dla negatywnej granicy jest określany podczas skanowania pozostałej części bazy danych. Jeśli na tym etapie odkryjemy, że zbiór elementów  $X$ , który stanowi część negatywnej granicy, zawiera się w zbiorze wszystkich częstych zbiorów elementów, wówczas należy przyjąć, że istnieje możliwość, że nadzbiór zbioru  $X$  także jest zbiorem częstym. W takim przypadku niezbędne jest wykonanie drugiego przeglądu zawartości bazy danych, który pozwoli nam się upewnić, że wszystkie częste zbiory elementów zostały odnalezione.

## 28.2.4. Drzewa częstych wzorców i algorytm ich tworzenia

**Drzewa częstych wzorców** (nazywane też FP-drzewami od ang. *frequent-pattern*) zaprojektowano jako alternatywę dla algorytmów opartych na strategii Apriori, które mogą generować i testować bardzo duże liczby kandydujących zbiorów elementów. Przykładowo, dla tysiąca jednoelementowych zbiorów częstych algorytm Apriori będzie musiał wygenerować aż

$$\binom{1000}{2}$$

czyli 499,5 tysiąca kandydujących zbiorów dwuelementowych. Właśnie **algorytm tworzenia drzew częstych wzorców** jest jednym z rozwiązań, które pozwalają wyeliminować konieczność generowania tak dużej liczby kandydujących zbiorów elementów.

Algorytm ten w pierwszej kolejności generuje skompresowaną wersję bazy danych, mającą postać drzewa częstych wzorców. Takie drzewo zawiera tylko najpotrzebniejsze informacje o zbiorach elementów i umożliwia efektywne odkrywanie częstych zbiorów elementów. Faktyczny proces eksploracji danych opiera się wówczas na strategii dziel i zwyciężaj (ang. *divide-and-conquer*), przewidującej podział zadania eksploracji na zbiory mniejszych zadań, z których każde operuje na warunkowym drzewie częstych wzorców będącym podzbiorem (projekcją) oryginalnego drzewa. Najpierw musimy przeanalizować proces konstruowania drzewa częstych wzorców. W pierwszej kolejności należy dokonać przeglądu bazy danych i na tej podstawie wyznaczyć jednoelementowe zbiory częste wraz z ich poziomami wsparcia. W algorytmie odkrywania częstych wzorców poziom wsparcia jest wyrażany za pomocą *liczby transakcji zawierających dany element*, zamiast w postaci odpowiedniego ułamka łącznej liczby transakcji. Jednoelementowe zbiory częste są następnie sortowane w porządku nierosnącym według ich poziomów wsparcia. Bezpośrednio potem tworzony jest korzeń drzewa częstych wzorców — początkowym oznaczeniem tego korzenia jest „pusta” etykieta. Podczas drugiego przeszu-

kiwania bazy danych dla każdej zarejestrowanej w tej bazie transakcji  $T$  sortowane są wszystkie jednoelementowe zbiory częste, które występują w danej transakcji (podobnie jak posortowane wcześniej wszystkie jednoelementowe zbiory częste odkryte w bazie danych). Można przyjąć, że posortowana lista zbiorów elementów dla transakcji  $T$  składa się z pierwszego elementu (nagłówka) i pozostałych elementów (reszty). Informacje o zbiorze elementów (nagłówek, reszta) są rekurencyjnie wstawiane do drzewa częstych wzorców, poczynawszy od węzła korzenia:

- (1) Jeśli bieżący węzeł  $N$  drzewa częstych wzorców ma potomka z nazwą elementu równą *nagłówek*, zwiększ o 1 licznik przypisany do węzła  $N$ ; w przeciwnym przypadku stwórz nowy węzeł  $N$  z licznikiem równym 1, połącz ten węzeł z jego przodkiem oraz z tabelą nagłówków elementów (tabela będzie wykorzystywana do efektywnego przeszukiwania drzewa).
- (2) Jeśli reszta nie jest pusta, powtórz krok (1), stosując w roli posortowanej listy tylko resztę — stary nagłówek jest usuwany, nowym nagłówkiem staje się pierwszy element reszty listy, natomiast pozostałe elementy stają się nową resztą tej listy.

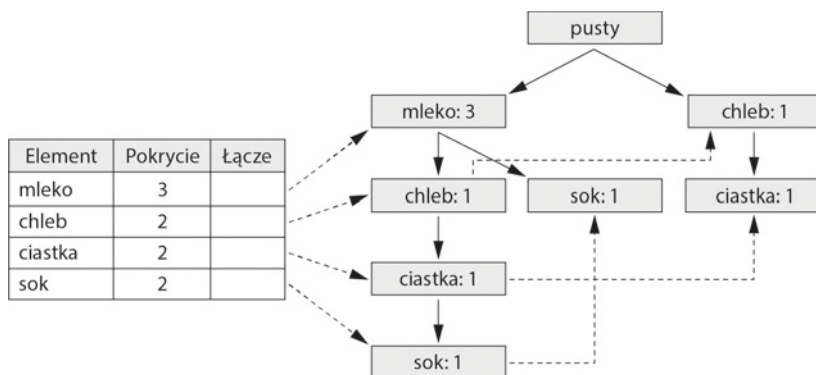
Tabela nagłówków elementów tworzona w procesie budowy drzewa częstych wzorców zawiera trzy pola dla każdego wiersza reprezentującego częsty element: identyfikator elementu, liczba poziomu wsparcia oraz łączy węzła. Dwa pierwsze pola, identyfikator elementu i liczba reprezentująca poziom wsparcia, nie wymagają wyjaśnień. Łączy węzła jest wskaźnikiem do wystąpienia odpowiedniego elementu w drzewie częstych wzorców. Ponieważ takie drzewo może zawierać więcej niż jedno wystąpienie tego samego elementu, wszystkie te elementy są powiązane z tabelą za pomocą łączy węzła wskazującego na początek odpowiedniej listy elementów. Proces budowy drzewa częstych wzorców zilustrujemy na przykładzie danych o transakcjach z rysunku 28.1. Przyjmijmy, że minimalny poziom wsparcia to 2. Po pierwszej operacji przeszukania czterech transakcji otrzymujemy następujące jednoelementowe zbiory częste wraz z odpowiednimi poziomami wsparcia:  $\{(\text{mleko}, 3)\}$ ,  $\{(\text{chleb}, 2)\}$ ,  $\{(\text{ciastka}, 2)\}$ ,  $\{(\text{sok}, 2)\}$ . Podczas drugiego skanowania bazy danych każda z opisanych w nich transakcji zostanie ponownie przetworzona.

Dla pierwszej transakcji tworzona jest posortowana lista  $T = \{\text{mleko}, \text{chleb}, \text{ciastka}, \text{sok}\}$ . Elementy listy  $T$  są jednoelementowymi zbiorami częstymi z pierwszej transakcji. Elementy należące do tego zbioru są posortowane zgodnie z nierosnącym porządkiem liczników przypisanych do jednoelementowych zbiorów odkrytych podczas pierwszego skanowania informacji o transakcjach (mleko jest pierwsze, chleb drugi itd.). Następnie tworzymy pusty węzeł korzenia dla drzewa częstych wzorców i wstawiamy „mleko” do węzła potomnego korzenia, „chleb” do węzła potomnego względem „mleka”, „ciastka” do węzła potomnego względem „chleba” oraz „sok” do węzła potomnego względem „ciastek”. Modyfikujemy wpisy dotyczące odpowiednich częstych elementów w tabeli nagłówków elementów.

Dla drugiej transakcji mamy posortowaną listę  $\{\text{mleko}, \text{sok}\}$ . Jeśli zaczniemy przeglądanie drzewa od korzenia, od razu zauważymy, że istnieje węzeł potomny oznaczony etykietą „mleko”, zatem przechodzimy do tego węzła i aktualizujemy jego licznik (aby uwzględnić drugą transakcję obejmującą zakup mleka). Ponieważ dla bieżącego węzła nie istnieje węzeł potomny oznaczony etykietą „sok”, tworzymy nowy węzeł (właśnie z etykietą „sok”) i aktualizujemy tabelę nagłówków elementów.



Dla trzeciej transakcji istnieje tylko jeden jednoelementowy zbiór elementów: {mleko}. Także w tym przypadku rozpoczęcie przeglądania drzewa od węzła korzenia od razu doprowadzi do odkrycia węzła potomnego oznaczonego etykietą „mleko”, należy więc przejść do tego węzła, zwiększyć jego licznik i dostosować odpowiedni wpis w tabeli nagłówków elementów. Ostatnia transakcja zawiera dwa częste elementy: {chleb, ciastka}. Analiza węzła korzenia pozwala stwierdzić, że nie istnieje bezpośredni węzeł potomny oznaczony etykietą „chleb”. Oznacza to, że musimy utworzyć nowego potomka korzenia, zainicjalizować jego licznik i wstawić węzeł potomny (oznaczony etykietą „ciastka”, także z zainicjalizowanym licznikiem) względem nowego węzła. Po zaktualizowaniu zawartości tabeli nagłówków elementów ostatecznie otrzymujemy drzewo częstych wzorców i tabelę w postaci przedstawionej na rysunku 28.2. Jeśli przeanalizujemy tak skonstruowane drzewo częstych wzorców, przekonamy się, że faktycznie reprezentuje ono oryginalne transakcje w skompresowanym formacie (a więc w postaci uwzględniającej tylko te elementy z każdej transakcji, które są dużymi zbiorami jednoelementowymi).



RYСУNEK 28.2. Drzewo częstych wzorców i odpowiadająca mu tabela nagłówków elementów

Algorytm 28.2 jest wykorzystywany do eksploracji drzewa częstych wzorców w poszukiwaniu częstych wzorców. Jeśli dysponujemy takim drzewem, możemy łatwo znaleźć wszystkie częste wzorce zawierające określony częsty element — wystarczy rozpocząć przeszukiwanie od tabeli nagłówków elementów i zlokalizować w niej odpowiedni element, po czym wykorzystać odpowiednie łącza węzłów w drzewie częstych wzorców. Algorytm rozpoczyna swoją pracę od jednoelementowego zbioru częstego (wzorca przyrostka), następnie konstruuje dla niego warunkową bazę wzorców i wreszcie buduje jego warunkowe drzewo częstych wzorców. Warunkowa baza wzorców jest tworzona na podstawie zbioru ścieżek przedrostków (prefiksów), częsty element występuje w takim przypadku w roli przyrostka (sufiksu). Przykładowo, jeśli raz jeszcze przeanalizujemy element „sok”, od razu zauważymy, że w przedstawionym na rysunku 28.2 drzewie częstych wzorców istnieją dwie ścieżki prowadzące do węzłów reprezentujących ten element: (mleko, chleb, ciastka, sok) oraz (mleko, sok). Dwie powiązane ścieżki przedrostków mają następującą postać: (mleko, chleb, ciastka) oraz (mleko). Na podstawie wzorców reprezentowanych w warunkowej bazie wzorców konstruowane jest warunkowe drzewo częstych wzorców. Bezpośrednio potem rekursywnie wywołujemy procedurę eksploracji. Częste wzorce są tworzone przez łączenie wzorca przyrostkowego z częstymi wzorcami wygenerowanymi na podstawie warunkowego drzewa częstych wzorców.



Algorytm 28.2. Algorytm odkrywania częstych wzorców w drzewie

**Dane wejściowe:** drzewo częstych wzorców oraz próg minimalnego wsparcia ( $min\_p$ ).

**Dane wyjściowe:** częste wzorce (zbiory elementów).

Procedura przeszukiwanie\_drzewa(*drzewo*, *alfa*);

**Początek**

Jeśli drzewo zawiera pojedynczą ścieżkę  $P$ , to:

dla każdej kombinacji  $\beta$  węzłów należących do tej ścieżki:

wygeneruj wzorec ( $\beta \cup \alpha$ )

z poziomem wsparcia równym minimalnemu poziomowi węzłów kombinacji  $\beta$ ;

w przeciwnym przypadku:

dla każdego elementu  $i$  w nagłówku danego drzewa:

**początek**

wygeneruj wzorec  $\beta = (i \cup \alpha)$  z poziomem wsparcia równym poziomowi elementu  $i$ ;

skonstruuj warunkową bazę wzorców  $\beta$ ;

skonstruuj warunkowe drzewo częstych wzorców  $\beta$ , tzw.  $\beta\_drzewo$ ;

jeśli  $\beta\_drzewo$  nie jest puste, to:

uruchom procedurę przeszukiwanie\_drzewa( $\beta\_drzewo$ ,  $\beta$ );

**koniec;**

**Koniec;**

Do zilustrowania rzeczywistego funkcjonowania tego algorytmu wykorzystamy przykładowe dane z rysunku 28.1 oraz drzewo częstych wzorców przedstawione na rysunku 28.2. Procedura przeszukiwanie\_drzewa jest wywoływana z dwoma parametrami: oryginalnym drzewem częstych wzorców oraz wartością pustą dla zmiennej  $\alpha$ . Ponieważ oryginalne drzewo częstych wzorców zawiera więcej niż jedną ścieżkę, uruchamiamy drugi blok (rozpoczynający się od wyrażenia w *przeciwnym przypadku*;) pierwszej instrukcji warunkowej. Zaczynamy od częstego elementu „sok”. Przeanalizujemy częste elementy, poczynawszy od najniższego poziomu wsparcia (a więc od ostatniego do pierwszego wpisu w tabeli nagłówków). Zmienna  $\beta$  ma przypisywaną etykietę „sok” z poziomem wsparcia równym 2.

Zgodnie z informacją zakodowaną w łączy węzła w tabeli nagłówków elementów, konstruujemy warunkową bazę wzorca składającą się z dwóch ścieżek (z elementem „sok” występującym w roli przyrostka). Są to odpowiednio ścieżki: (mleko, chleb, ciastka: 1) oraz (mleko: 1). Warunkowe drzewo częstych wzorców składa się tylko z jednego węzła (mleko: 2). Wynika to z faktu, że poziom wsparcia dla węzłów reprezentujących chleb i ciastka wynosi tylko 1, a więc jest niższy od przyjętego na początku minimalnego progu wsparcia na poziomie 2. Algorytm jest wywoływany rekurencyjnie z drzewem częstych wzorców składającym się tylko z jednego węzła (mleko: 2) oraz wartością  $\beta$  z przypisaną etykietą „sok”. Ponieważ takie drzewo częstych wzorców ma tylko jedną ścieżkę, algorytm generuje wszystkie kombinacje wartości zmiennej  $\beta$  i węzłów w jedynej ścieżce (w tym przypadku jest to para {mleko, sok} z poziomem wsparcia równym 2).

Kolejnym wykorzystywanym elementem są „ciastka”. Zmienna  $\beta$  ma przypisywaną wartość „ciastka” z poziomem wsparcia równym 2. Zgodnie ze wskazaniem łączy węzła w tabeli nagłówków elementów konstruujemy warunkową bazę wzorca składającą się z dwóch ścieżek: (mleko, chleb: 1) oraz (chleb: 1). Warunkowe drzewo częstych wzorców

zawiera tylko jeden węzeł (chleb: 2). Nasz algorytm jest więc rekurencyjnie wywoływany z drzewem częstych wzorców składającym się właśnie ze wspomnianego węzła oraz wartości „ciastka” w zmiennej *beta*. Ponieważ przekazane drzewo częstych wzorców ma tylko jedną ścieżkę, algorytm generuje wszystkie kombinacje wartości zmiennej *beta* i węzłów w tej ścieżce, czyli {chleb, ciastka} z wsparciem na poziomie równym 2. Bezpośrednio potem poddajemy analizie częsty element „chleb”. Zmienna *beta* ma standardowo przypisywaną wartość „chleb” z poziomem wsparcia 2. Zgodnie ze wskazaniem odpowiedniego łączy węzła w tabeli nagłówków elementów, konstruujemy warunkową bazę wzorca składającą się tylko z jednej ścieżki: (mleko: 1). Warunkowe drzewo częstych wzorców jest puste, ponieważ wsparcie nie przekracza ustalonego wcześniej minimalnego progu. Efektem otrzymania pustego drzewa częstych wzorców jest także brak możliwości wygenerowania tych wzorców.

Ostatnim rozważanym częstym elementem jest „mleko”. Element ten jest reprezentowany przez najwyższy wpis w tabeli nagłówków elementów i jako taki ma pustą warunkową bazę wzorca oraz puste warunkowe drzewo częstych wzorców. W efekcie nie są dodawane żadne częste wzorce. Wynikiem wykonania całego algorytmu są więc następujące częste wzorce (lub zbiory elementów) wraz z ich poziomami wsparcia: {{mleko: 3}, {chleb: 2}, {ciastka: 2}, {sok: 2}, {mleko, sok: 2}, {chleb, ciastka: 2}}.

### 28.2.5. Algorytm partycjonujący

Poniżej przedstawiono podsumowanie założeń i zasad funkcjonowania innego algorytmu, nazwanego **algorytmem partycjonującym**<sup>3</sup>. Jeśli dysponujemy bazą danych ze stosunkowo niewielką liczbą potencjalnie dużych zbiorów elementów, powiedzmy nie przekraczającą kilku tysięcy, wówczas poziom wsparcia dla wszystkich tych zbiorów można wyznaczyć podczas pojedynczego skanowania (przeszukiwania) tej bazy z wykorzystaniem technik partycjonowania. Partycjonowanie polega na dzieleniu bazy danych na rozłączne podzbiory, które są następnie traktowane jak osobne bazy danych i dla których w ramach jednej operacji skanowania można wygenerować wszystkie duże zbiory elementów (nazywane *lokalnymi zbiorami częstymi*). Takie rozwiązanie umożliwia efektywne wykorzystanie algorytmu *Apriori* dla każdej partycji, pod warunkiem, że partycje te mieszczą się w głównej pamięci operacyjnej komputera. Podział bazy danych na partycje odbywa się w sposób umożliwiający ich składowanie w całości w szybkiej pamięci operacyjnej. Jedyną wadą metody opartej na partycjonowaniu jest konieczność stosowania dla poszczególnych partycji takich poziomów wsparcia, które mają nieco inne znaczenie od odpowiednich wartości stosowanych w oryginalnej bazie danych. Minimalny poziom wsparcia, który jest wykorzystywany do określania lokalnych (dużych) zbiorów elementów, jest uzależniony od rozmiaru partycji, nie od rozmiaru bazy danych. Wartość progu minimalnego wsparcia nie ulega co prawda zmianie, jednak faktyczne wsparcie jest wyznaczane tylko dla poszczególnych partycji.

Na końcu pierwszej fazy algorytmu wyznaczana jest suma wszystkich częstych zbiorów elementów dla każdej z partycji. W ten sposób powstają globalne kandydujące częste zbiory elementów dla całej bazy danych. Scalone listy mogą zawierać pewne fałszywe

---

<sup>3</sup> Więcej szczegółowych informacji na temat tego algorytmu, opis struktur danych wykorzystywanych do jego implementowania oraz porównanie jego wydajności z innymi algorytmami można znaleźć w książce Savasere’a i in. (1995).

trafienia. Oznacza to, że niektóre z uwzględnionych na liście zbiorów elementów, które są zbiorami częstymi (dużymi) w jednej partycji, mogą nie zostać zakwalifikowane do tej grupy w wielu innych partycjach i w efekcie słusznie nie przekraczać progu minimalnego wsparcia na poziomie całej (oryginalnej) bazy danych. Warto przy tym pamiętać, że tak skonstruowana lista nigdy nie zawiera fałszywych pominięć — żadne duże zbiory elementów nie mogą być pominięte. Globalne, kandydujące duże zbiory elementów zidentyfikowane podczas pierwszej fazy są weryfikowane na drugim etapie pracy algorytmu — ich rzeczywisty poziom wsparcia jest wyznaczany dla całej bazy danych. Na końcu tej fazy algorytm identyfikuje wszystkie globalne duże zbiory elementów. Łatwo zauważyć, że algorytm partycjonujący doskonale pasuje do zapewniających większą efektywność implementacji równoległych lub rozproszonych. W dostępnej literaturze zasugerowano wiele dodatkowych udoskonaleń tego algorytmu<sup>4</sup>.

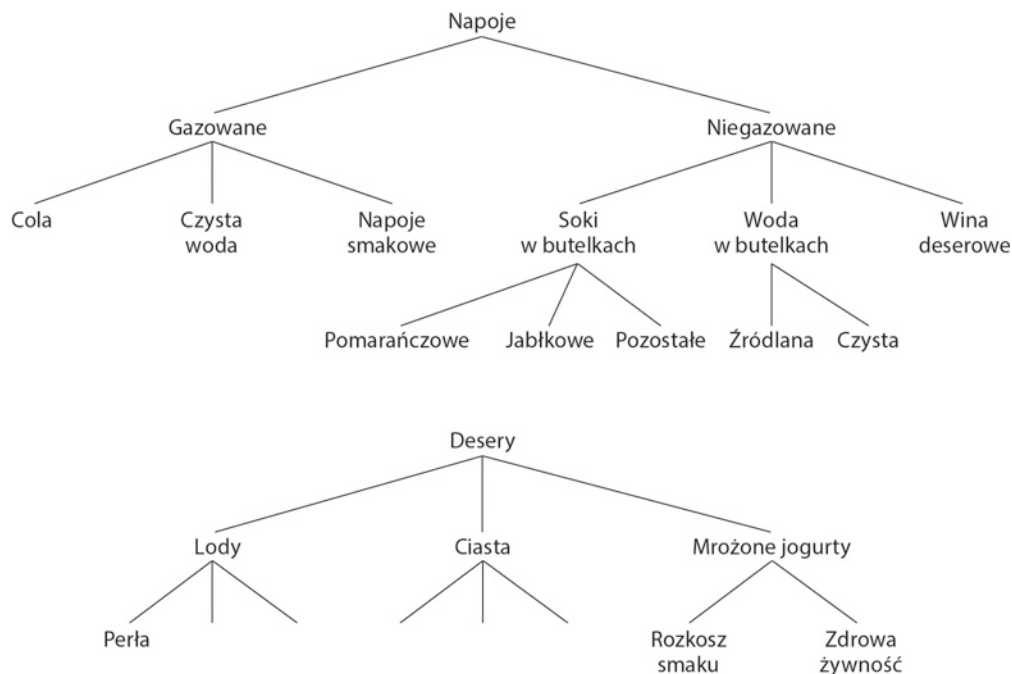
### 28.2.6. Pozostałe typy reguł asocjacyjnych

**Reguły asocjacyjne w hierarchiach.** Istnieją pewne typy asocjacji, które są szczególnie interesujące podczas bardzo szczegółowych analiz. Takie asocjacje mogą występować pomiędzy hierarchiami elementów. W wielu przypadkach istnieje możliwość podzielenia kompletnego zbioru elementów na rozłączne hierarchie według naturalnych cech ściśle związanych z określoną dziedziną wiedzy. Przykładowo, żywność w supermarkecie, książki w sklepie wydziału akademickiego lub artykuły w sklepie sportowym można skategoryzować w klasy lub podklasy i tym samym stworzyć odpowiednie hierarchie. Przeanalizujmy teraz rysunek 28.3, na którym przedstawiono klasyfikację towarów sprzedawanych w supermarkecie. Widać tam dwie hierarchie — odpowiednio napoje i desery. Całe te grupy nie muszą oczywiście tworzyć asocjacji w postaci napoje  $\Rightarrow$  desery lub desery  $\Rightarrow$  napoje. Nie oznacza to jednak, że nie mogą istnieć asocjacje w postaci (jogurt marki Zdrowa Żywność  $\Rightarrow$  woda w butelkach) lub (lody marki Perła  $\Rightarrow$  wina deserowe), których poziom ufności i poziom wsparcia jest na tyle duży, by wykryć poprawne reguły asocjacyjne.

Oznacza to, że jeśli dla obszaru aplikacji istnieje naturalna klasyfikacja zbiorów elementów w rozłączne hierarchie, odkrywanie asocjacji w ramach poszczególnych hierarchii nie jest szczególnie interesujące. Znacznie większą wartość mają wówczas asocjacje pomiędzy tymi hierarchiami — asocjacje tego typu mogą dotyczyć grup elementów na różnych poziomach analizowanych hierarchii.

**Asocjacje wielowymiarowe.** Odkrywanie reguł asocjacyjnych wiąże się z wyszukiwaniem wzorców w pliku. Na rysunku 28.1 przedstawiliśmy przykład pliku z danymi o transakcjach klientów wyrażonymi w trzech wymiarach: *Identyfikator-transakcji*, *Czas* i *Kupione-towary*. Przedstawione do tej pory zadania i algorytmy eksploracji danych dotyczyły jednak tylko jednego wymiaru — towarów zakupionych przez klienta. Następująca przykładowa reguła obejmuje etykiety pochodzące z pojedynczego wymiaru: *Kupione-towary*(mleko)  $\Rightarrow$  *Kupione-towary*(sok). Być może bardziej interesujące byłoby odkrycie reguł asocjacyjnych dotyczących wielu wymiarów; przykładowo, taka reguła asocjacyjna może mieć postać: *Czas*(6:30...8:00)  $\Rightarrow$  *Kupione-towary*(mleko). Tego typu reguły są nazywane *wielowymiarowymi regułami asocjacyjnymi*. Wymiary reprezentują albo atrybuty rekordów pliku, albo

<sup>4</sup> Patrz publikacje Cheunga i in. (1996) oraz Lina i Dunhama (1998).



RYSUNEK 28.3. Klasyfikacja elementów w supermarkecie

(w modelu relacyjnym) kolumny lub wiersze relacji, i mogą mieć charakter nominalny (kategorie) lub ilościowy. Atrybuty nominalne (kategorie) mają skończone zbiory wartości, pomiędzy którymi nie istnieją żadne relacje porządku. Atrybuty ilościowe reprezentują dane numeryczne, pomiędzy którymi naturalnie występuje relacja porządku (np. relacja mniejszości). Wymiar *Kupione-towary* jest przykładem atrybutu nominalnego, natomiast wymiary *Identyfikator-transakcji* i *Czas* są przykładami atrybutów ilościowych.

Jedną z możliwych strategii obsługi atrybutów ilościowych jest podzielenie ich wartości pomiędzy rozłączne przedziały oznaczone odpowiednimi etykietami. Taki podział może mieć charakter statyczny i opierać się na typowej wiedzy dziedzinowej. Przykładowo, w koncepcji opartej na hierarchii można pogrupować wartości reprezentujące wynagrodzenie pracowników w trzech rozłącznych grupach: niskie wynagrodzenie ( $0 < pensja < 1999$ ), średnie wynagrodzenie ( $2000 < pensja < 5999$ ) oraz wysokie wynagrodzenie ( $pensja > 5999$ ). Po wprowadzeniu takiej klasyfikacji możemy użyć do odkrywania reguł asocjacyjnych typowego algorytmu Apriori lub jednej z jego odmian, ponieważ dotychczasowy atrybut ilościowy jest teraz reprezentowany tak, jak typowy atrybut nominalny. Jeszcze inne podejście do partycjonowania przewiduje grupowanie wartości atrybutów według rozkładu danych; przykładowo, można zastosować partycjonowanie wg równej głębokości i każdej z utworzonych w ten sposób partycji przypisać wartości całkowite. Partycjonowanie przeprowadzane na tym etapie może być stosunkowo precyzyjne, a więc generować dużą liczbę przedziałów. Następnie, już w procesie eksploracji danych, można te partycje łączyć z sąsiednimi partycjami, jeśli np. okaże się, że wyznaczony dla nich poziom wsparcia nie przekracza wartości minimalnej. W tej roli można wykorzystywać algorytmy z rodziny Apriori (używane także podczas eksploracji danych).

**Asocjacje negatywne.** Problem odkrywania asocjacji negatywnych jest trudniejszy do rozwiązania niż omawiane do tej pory odkrywanie asocjacji pozytywnych. Przykładowa asocjacja negatywna może mieć następującą postać: „60% klientów kupujących chipsy nie kupuje wody w butelkach” (w tym przypadku 60% odnosi się do poziomu ufności dla negatywnej reguły asocjacyjnej). W bazie danych zawierającej 10 tysięcy elementów istnieje aż  $2^{10000}$  możliwych kombinacji elementów, z których większość nawet raz nie występuje w bazie danych. Jeśli asocjacja negatywna ma uwzględniać brak określonej kombinacji elementów w bazie danych, wówczas możemy mieć miliony reguł asocjacyjnych, których prawe strony (zbiór PS) nie zawierają żadnych istotnych informacji. Zasadniczym problemem jest wówczas znalezienie tylko *interesujących* reguł negatywnych. Ogólnie rzecz biorąc, interesują nas przypadki, w których dwa określone zbiory elementów bardzo rzadko występują w tej samej transakcji. Z odkrywaniem tych przypadków wiążą się dwa poważne problemy:

- (1) Dla łącznej liczby elementów równej 10 tysięcy prawdopodobieństwo jednoczesnego kupienia dwóch dowolnych elementów wynosi  $(1/10\,000) \cdot (1/10\,000) = 10^{-8}$ . Jeśli stwierdzimy, że faktyczny poziom wsparcia dla wspólnego występowania tej pary wynosi zero, uzyskany wynik nie będzie znacząco odbiegał od oczekiwań i jako taki nie będzie stanowił interesującej (negatywnej) asocjacji.
- (2) Drugi problem jest znacznie poważniejszy. Przeglądamy bazę danych o transakcjach w poszukiwaniu kombinacji z bardzo niskim poziomem wsparcia, a w jednej bazie danych takich kombinacji (także z zerowym poziomem wsparcia) mogą istnieć miliony. Przykładowo, zbiór danych dla 10 milionów transakcji zawiera 2,5 miliarda dwuskładnikowych kombinacji, które nie zawierają 10 tysięcy elementów. Oznacza to, że można w oparciu o taką bazę danych wygenerować miliardy zupełnie nieprzydatnych reguł.

Aby wygenerowane negatywne reguły asocjacyjne były dla nas istotne (zawierały przydatne informacje), musimy wykorzystać naszą wiedzę na temat zbiorów elementów jeszcze zanim przystąpimy do odkrywania tych reguł. Jednym z rozwiązań jest zastosowanie hierarchii. Przypuśćmy, że wykorzystujemy w tym celu przedstawione na rysunku 28.4 hierarchie napojów bezalkoholowych i chipsów.



RYSUNEK 28.4. Prosta hierarchia napojów bezalkoholowych i chipsów

Założmy, że w bazie danych odkryto silną asocjację pozytywną pomiędzy napojami bezalkoholowymi a chipsami. Jeśli okaże się, że istnieje wysokie wsparcie dla następującej reguły: zakup chipsów Dnios przeważnie oznacza, że klient kupi także napój Topsy i *nie* kupi *ani* napoju Joke, *ani* napoju Wakeup, takie spostrzeżenie będzie oczywiście interesujące. Wynika to z faktu, że naturalne byłoby oczekiwanie, że jeśli istnieje silna asocjacja pomiędzy transakcjami zakupu napoju Topsy i chipsów Dnios, powinna także istnieć silna asocjacja pomiędzy chipsami Dnios i napojem Joke lub pomiędzy chipsami Dnios i napojem Wakeup<sup>5</sup>.

<sup>5</sup> Dla uproszczenia zakładamy, że transakcje rozkładają się równomiernie pomiędzy składowymi analizowanej hierarchii.

Przypuśćmy, że w przedstawionych na rysunku 28.3 grupach mrożony jogurt oraz woda w butelkach istnieje odpowiednio podział rynku w stosunku 80:20 pomiędzy markami Rozkosz Smaku a Zdrowa Żywność oraz podział w stosunku 60:40 pomiędzy markami Źródłana i Czysta. Informacja o takim podziale oznacza, że w transakcjach obejmujących zakup mrożonego jogurtu i wody w butelce istnieje prawdopodobieństwo warunkowe na poziomie 48% pomiędzy zakupem mrożonego jogurtu marki Rozkosz Smaku a wodą w butelce marki Źródłana. Jeśli jednak wyznaczony poziom wsparcia dla takiej reguły będzie wynosił tylko 20%, będzie to oznaczało, że nasze odkrycie stanowi istotną negatywną asocjację pomiędzy jogurtem marki Rozkosz Smaku a butelkowaną wodą marki Źródłana. Asocjacja ta może być interesująca dla analityka zachowań klientów.

Problem związany z odkrywaniem negatywnych asocjacji jest szczególnie istotny w sytuacjach podobnych do zaprezentowanej powyżej, gdzie wiedza dziedzinowa ma postać hierarchii generalizacji elementów (w tym przypadku są to przedstawione na rysunku 28.3 hierarchie napojów i deserów), istniejących asocjacji pozytywnych (np. istniejąca pomiędzy grupami mrożonych jogurtów i wód w butelkach) oraz rozkładu elementów (np. pomiędzy poszczególnymi markami w ramach odpowiednich grup). Odpowiednie badania nad technologiami baz danych w tym kontekście przeprowadziła specjalna grupa naukowców w ośrodku Georgia Tech (patrz notka bibliograficzna). Zakres możliwych odkryć w obszarze asocjacji negatywnych jest ograniczony z powodu niedostatecznej znajomości hierarchii i rozkładów elementów. Innym ważnym utrudnieniem jest wykładniczy wzrost liczby takich asocjacji wraz ze wzrostem liczby elementów i transakcji w bazie danych.

### 28.2.7. Dodatkowe problemy związane z regułami asocjacyjnymi

Odkrywanie reguł asocjacyjnych w rzeczywistych bazach danych dodatkowo komplikują następujące czynniki:

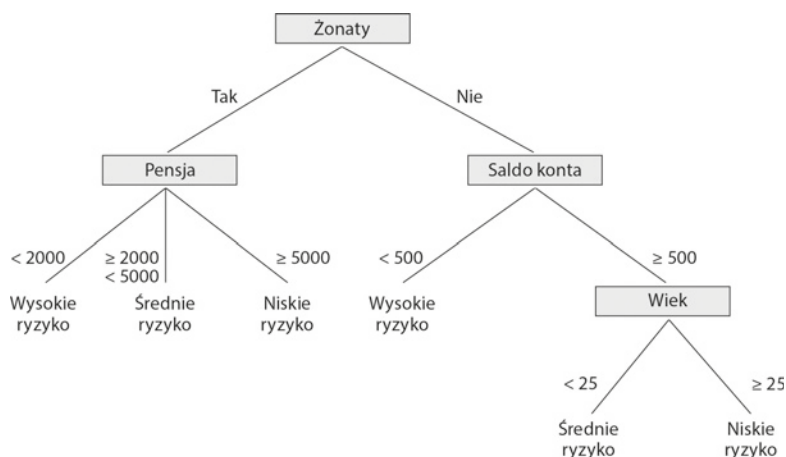
- W większości przypadków istnieje nie tylko problem znacznej liczności zbiorów elementów, ale także bardzo dużej ilości transakcji. Niektóre operacyjne bazy danych w placówkach handlowych i przedsiębiorstwach telekomunikacyjnych gromadzą dziesiątki milionów transakcji dziennie.
- Transakcje często są zmienne w takich aspektach jak lokalizacja geograficzna czy sezony handlowe, co może bardzo utrudnić proces próbkowania.
- Klasyfikacja elementów może dotyczyć wielu wymiarów. Oznacza to, że przeprowadzanie procesu odkrywania nowych informacji (w szczególności reguł negatywnych) wyłącznie w oparciu o wiedzę dziedzinową jest wyjątkowo trudne.
- Także jakość danych może podlegać częstym zmianom; w bardzo wielu zastosowaniach technik eksploracji danych źródłem najpoważniejszych problemów są brakujące, błędne, sprzeczne oraz nadmiarowe dane.



## 28.3. Klasyfikacja

**Klasyfikacja** jest procesem uczenia modelu opisującego różne klasy danych. Przyjmuje się, że klasy te są znane z góry. Przykładowo, w aplikacji wykorzystywanej w banku, klienci, którzy złożą wnioski o wydanie kart kredytowych mogą zostać zakwalifikowani do grup *wysokiego ryzyka*, *średniego ryzyka* oraz *niskiego ryzyka*. W związku z tym działania w tym zakresie często są nazywane **uczeniem z nauczycielem** lub **uczeniem nadzorowanym** (ang. *supervised learning*). Kiedy odpowiedni model zostanie zbudowany, można go wykorzystać do sklasyfikowania danych. Pierwszym krokiem uczenia modelu jest zastosowanie treningowego zbioru danych, który przeszedł już proces klasyfikacji. Każdy rekord w danych treningowych zawiera atrybut (nazywany etykietą *klasy*) wskazujący klasę, do której dany rekord należy. Wygenerowany w ten sposób model zazwyczaj ma postać drzewa decyzyjnego lub zbioru reguł. Do najtrudniejszych zadań związanych z takim modelem i generującym go algorytmem należy zapewnienie odpowiedniej zdolności przydzielania nowych danych do właściwych klas — istotnym problemem jest koszt obliczeniowy i skalowalność tego typu algorytmów.

W tym podrozdziale przeanalizujemy strategię, w której model ma postać drzewa decyzyjnego. **Drzewo decyzyjne** jest po prostu graficzną reprezentacją opisu każdej z klas; innymi słowy, jest ono jedną z możliwych reprezentacji reguł klasyfikacji. Na rysunku 28.5 przedstawiono przykładowe drzewo decyzyjne. Z konstrukcji tego drzewa wynika, że jeśli klient banku jest *żonaty* i jego miesięczna pensja wynosi 5 tysięcy złotych lub więcej, ryzyko przyznania mu karty kredytowej jest niskie. Jest to jedna z reguł, która opisuje klasę *niskie ryzyko*. Pozostałe reguły dla tej i dwóch pozostałych klas można określić, przeglądając drzewo decyzyjne począwszy od jego korzenia i skończywszy w odpowiednim węźle liścia. Algorytm 28.3 zawiera procedurę konstruowania drzewa decyzyjnego w oparciu o zbiór danych treningowych. Początkowo wszystkie próbki treningowe znajdują się w korzeniu budowanego drzewa. Próbkę tę są następnie rekursywnie partycjonowane według wartości wybranych atrybutów. Do partycjonowania próbek w poszczególnych węzłach należy wykorzystywać te atrybuty, które zapewniają najlepsze kryteria podziału, czyli np. maksymalizują wskaźnik przyrostu informacji.



RYSunEK 28.5. Przykład drzewa decyzyjnego dla aplikacji wspomagającej decyzję o przyznawaniu kart kredytowych



## Algorytm 28.3. Algorytm indukcji drzewa decyzyjnego

**Dane wejściowe:** zbiór rekordów treningowych:  $R_1, R_2, \dots, R_m$  oraz zbiór atrybutów:  $A_1, A_2, \dots, A_n$ .

**Dane wyjściowe:** drzewo decyzyjne.

Procedura Buduj\_drzewo(*Rekordy*, *Atrybuty*);

Początek

Stwórz węzeł  $N$ ;

Jeśli wszystkie rekordy (w zbiorze *Rekordy*) należą do tej samej klasy  $C$ , to:  
zwróć  $N$  jako węzeł liścia z etykietą klasy  $C$ ;

Jeśli zbiór *Atrybuty* jest pusty, to:

zwróć  $N$  jako węzeł liścia z etykietą klasy  $C$  w taki sposób, aby większość rekordów

(elementów zbioru *Rekordy*) należała do tej klasy;

Wybierz w zbiorze *Atrybuty* atrybut  $A_i$  (z *najwyższym* wskaźnikiem przyrostu informacji);

Oznacz węzeł  $N$  atrybutem  $A_i$ ;

Dla każdej znanej wartości  $V_j$  atrybutu  $A_i$ :

początek

Dodaj do węzła  $N$  gałąź dla warunku  $A_i = V_j$ ;

Przypisz zmiennej  $S_j$  podzbiór zbioru rekordów, w których  $A_i = V_j$ ;

Jeśli zbiór  $S_j$  jest pusty, to:

dodaj do drzewa taki liść  $L$  z etykietą klasy  $C$ , aby należała do niego większość rekordów ze zbioru *Rekordy*, i zwróć  $L$ ;

w przeciwnym przypadku, dodaj węzeł zwrócony przez procedurę

Buduj\_drzewo( $S_j$ ,

*Atrybuty* -  $A_i$ );

koniec;

Koniec;

Zanim przystąpimy do omawiania przykładu ilustrującego działanie algorytmu 28.3, musimy bardziej szczegółowo wyjaśnić wspomniany wskaźnik **przyrostu informacji**. Użycie **entropii** w roli takiego wskaźnika wynika z przyjętego celu minimalizowania informacji potrzebnych do sklasyfikowania przykładowych danych w wynikowych partycjach, a więc — tym samym — minimalizowania oczekiwanej liczby testów warunkowych niezbędnych do sklasyfikowania nowych rekordów. Ilość oczekiwanych informacji, które będą potrzebne do sklasyfikowania pojedynczej próbki danych treningowych złożonych z  $s$  próbek — gdzie atrybut *Klasa* obejmuje  $n$  wartości  $v_1, \dots, v_n$ , a  $s_i$  jest liczbą próbek należących do klasy oznaczonej etykietą  $v_i$  — można wyliczyć za pomocą następującego wzoru:

$$I(s_1, s_2, \dots, s_n) = - \sum_{i=1}^n p_i \log_2 p_i$$

gdzie  $p_i$  jest prawdopodobieństwem tego, że losowa próbka należy do klasy oznaczonej etykietą  $v_i$ . Przybliżeniem wartości  $p_i$  jest iloraz  $s_i/s$ . Przeanalizujemy teraz atrybut  $A$  z wartościami  $\{v_1, \dots, v_m\}$ , który będzie wykorzystywany w roli atrybutu testowego dla podziału danych w ramach drzewa decyzyjnego. Atrybut  $A$  partycjonuje próbki na podzbiory  $S_1, \dots, S_m$ , gdzie próbki przydzielone do każdego z podzbiorów  $S_j$  zawierają w atrybucie  $A$  wartość  $v_j$ .

Każdy podzbiór  $S_j$  może zawierać próbki należące do dowolnej z klas. Liczba tych próbek w podzbiorze  $S_j$ , które należą do klasy  $i$ , oznaczamy symbolem  $s_{ij}$ . Entropię związaną z atrybutem  $A$  wykorzystanym w roli atrybutu testowego można zdefiniować za pomocą następującego zbioru:

$$E(A) = \sum_{j=1}^m \frac{s_{1j} + \dots + s_{nj}}{s} \times I(s_{1j}, s_{2j}, \dots, s_{nj})$$

Wartość  $I(s_{1j}, \dots, s_{nj})$  można zdefiniować, wykorzystując przedstawiony wcześniej wzór dla  $I(s_1, \dots, s_n)$ , w którym moglibyśmy zastąpić prawdopodobieństwo  $p_i$  prawdopodobieństwem  $p_{ij} = s_{ij}/s_j$ . Przyrost informacji związany z partycjonowaniem według atrybutu  $A$  —  $Przyrost(A)$  — możemy teraz zdefiniować w postaci następującej różnicy:  $I(s_1, \dots, s_n) - E(A)$ . Do zilustrowania działania tego algorytmu wykorzystamy przykładowe dane treningowe z rysunku 28.6.

IdR	Żonaty	Pensja	Saldo konta	Wiek	Możliwość przyznania kredytu
1	nie	≥5000	<500	≥25	tak
2	tak	≥5000	≥500	≥25	tak
3	tak	2000...5000	<500	<25	nie
4	nie	<2000	≥500	<25	nie
5	nie	<2000	<500	≥25	nie
6	tak	2000...5000	≥500	≥25	tak

RYСУNEK 28.6. Przykładowe dane treningowe dla algorytmu klasyfikacji

Atrybut *IdR* reprezentuje identyfikator rekordu, który jest wewnętrznie używany do unikatowego odwoływania się do pojedynczych wpisów. Będziemy ten atrybut wykorzystywali do identyfikowania poszczególnych rekordów w ramach naszych przykładowych danych. W pierwszej kolejności musimy wyznaczyć średni oczekiwany przyrost informacji, który będzie nam potrzebny do sklasyfikowania pojedynczego rekordu ze zbioru danych treningowych złożonych z sześciu rekordów w postaci  $I(s_1, s_2)$ , gdzie pierwsza wartość etykiety klasy odpowiada wynikowi *tak*, natomiast druga — wynikowi *nie*. Zatem:

$$I(3,3) = -0,5 \log_2 0,5 - 0,5 \log_2 0,5 = 1$$

Możemy teraz obliczyć entropię dla każdego z czterech atrybutów. Dla wartości „tak” atrybutu *Żonaty* otrzymujemy:  $s_{11} = 2$ ,  $s_{21} = 1$  oraz  $I(s_{11}, s_{21}) = 0,92$ . Dla wartości „nie” atrybutu *Żonaty* otrzymujemy:  $s_{12} = 1$ ,  $s_{22} = 2$  oraz  $I(s_{12}, s_{22}) = 0,92$ . Oznacza to, że ilość oczekiwanych informacji, które są niezbędne do sklasyfikowania próbki w oparciu o wartość atrybutu *Żonaty* (występujący w roli atrybutu partycjonowania) wynosi:

$$E(\text{Żonaty}) = 3/6 I(s_{11}, s_{21}) + 3/6 I(s_{12}, s_{22}) = 0,92$$

Oznacza to, że przyrost informacji —  $Przyrost(\text{Żonaty})$  — wyniesie  $1 - 0,92 = 0,08$ . Jeśli wykonamy podobne kroki w odniesieniu do przyrostu informacji dla pozostałych trzech atrybutów, otrzymamy następujące wyniki:

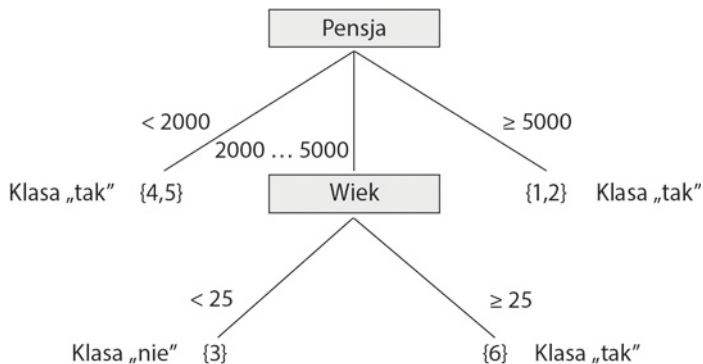
$$E(\text{Pensja}) = 0,33 \text{ oraz } Przyrost(\text{Pensja}) = 0,67$$

$$E(\text{Saldo konta}) = 0,92 \text{ oraz } Przyrost(\text{Saldo konta}) = 0,08$$

$$E(\text{Wiek}) = 0,54 \text{ oraz } Przyrost(\text{Wiek}) = 0,46$$

Ponieważ największy przyrost zaobserwowano dla atrybutu *Pensja*, właśnie ten atrybut wybieramy do roli atrybutu partycjonującego. Efektem tego wyboru jest utworzenie korzenia drzewa z etykietą *Pensja* i trzema gałęziami: po jednej dla każdej wartości wybranego atrybutu. W przypadku dwóch z trzech wartości (konkretnie, *Pensja* < 2000 oraz *Pensja* ≥ 5000) wszystkie próbki, które przeszły proces partycjonowania (rekordy z identyfikatorami *IdR* 4 i 5 dla pensji poniżej 2000 oraz rekordy z identyfikatorami *IdR* 1 i 2 dla pensji powyżej 5000), są umieszczane wewnątrz tych samych klas, czyli odpowiednio *negatywna opinia o zdolności kredytowej* oraz *pozytywna opinia o zdolności kredytowej*. Te węzły są więc liśćmi. Jedyną gałęzią, która wymaga dalszego rozwinięcia, jest gałąź reprezentująca wartości od 2000 do 5000 i zawierająca dwie próbki — rekordy danych treningowych z identyfikatorami *IdR* równymi 3 i 6. Kontynuując nasz proces w oparciu o te dwa rekordy, odkrywamy, że *Przyrost(Żonaty)* wynosi 0, *Przyrost(Saldo konta)* wynosi 1 oraz *Przyrost(Wiek)* wynosi 1.

W takim przypadku możemy wybrać zarówno atrybut *Wiek*, jak i atrybut *Saldo konta*, ponieważ przyrost informacji dla obu tych argumentów jest identyczny i maksymalny. Dodajemy do drzewa decyzyjnego węzeł z etykietą *Wiek* i dwoma gałęziami: dla wieku poniżej 25. roku życia oraz dla wieku od 25. roku życia wzwyż. Każda z nowych gałęzi dzieli pozostałe dane w taki sposób, że gałęzie te zawierają po jednym rekordzie, a więc także po jednej klasie. Na samym końcu tworzymy dwa nowe węzły liści. Ostateczną wersję skonstruowanego w ten sposób drzewa decyzyjnego przedstawiono na rysunku 28.7.



RYSUNEK 28.7. Drzewo decyzyjne skonstruowane na bazie przykładowych danych treningowych. węzły liści są reprezentowane przez zbiór wykorzystywanych w procesie partycjonowania identyfikatorów rekordów

## 28.4. Grupowanie

Omawiane przed chwilą zadanie klasyfikacji danych, które polegało na ich partycjonowaniu w oparciu o sklasyfikowane wcześniej próbki treningowe, miało na celu ułatwienie eksploracji danych. Często jednak wygodniejszym rozwiązaniem jest podzielenie danych bez konieczności uprzedniego tworzenia próbek treningowych; ta technika często jest nazywana **uczeniem bez nauczyciela** lub **uczeniem nienadzorowanym** (ang. *unsupervised learning*). Przykładowo, podczas przeprowadzania analiz biznesowych duże znaczenie może

mieć wskazanie grup klientów, którzy realizują zbliżone wzorce kupowania; natomiast w świecie medycyny istotna może być możliwość identyfikowania grup pacjentów, którzy wykazują podobne reakcje na podawane leki. Celem grupowania jest skupianie rekordów w odpowiednich podzbiorach w sposób zapewniający z jednej strony podobieństwo wszystkich rekordów w grupie, a z drugiej ich odmienność względem rekordów umieszczonych w pozostałych grupach. Tworzone w ten sposób grupy przeważnie są *rozłączne*.

Istotnym aspektem procesu grupowania danych jest wykorzystywana funkcja podobieństwa. Jeśli grupowanie ma dotyczyć danych numerycznych, zwykle można stosować funkcję podobieństwa opartą na odległości. Przykładowo, do określania podobieństwa można użyć funkcji odległości euklidesowej. Przyjmijmy, że mamy dwa  $n$ -wymiarowe punkty danych (rekordy):  $r_j$  i  $r_k$ . Wartości tych rekordów w  $i$ -tym wymiarze oznaczamy odpowiednio za pomocą symboli  $r_{ji}$  oraz  $r_{ki}$ . Odległość euklidesową pomiędzy punktami  $r_j$  i  $r_k$  w przestrzeni  $n$ -wymiarowej obliczamy za pomocą wzoru:

$$\text{Odległość}(r_j, r_k) = \sqrt{|r_{j1} - r_{k1}|^2 + |r_{j2} - r_{k2}|^2 + \dots + |r_{jn} - r_{kn}|^2}$$

Im mniejsza odległość dzieli tę parę punktów, tym większe jest ich podobieństwo. Klasycznym algorytmem grupowania jest tzw. **algorytm k-średnich** (ang. *k-means algorithm*); patrz algorytm 28.4.

#### Algorytm 28.4. Grupujący algorytm k-średnich

**Dane wejściowe:** baza danych  $D$  z  $m$  rekordami  $r_1, \dots, r_m$  oraz docelowa liczba grup  $k$ .

**Dane wyjściowe:** taki zbiór  $k$  grup, który minimalizuje kryterium kwadratu błędu.

##### Początek

Losowo wybierz  $k$  rekordów będących środkami (ang. *centroids*)  $k$  klastrów; powtarzaj:

każdy rekord  $r_i$  przypisz do takiej grupy, dla której odległość pomiędzy rekordem  $r_i$  a elementem centralnym (środkiem) jest najmniejsza ze wszystkich  $k$  grup;

dla każdej grupy ponownie oblicz środek (średnią) w oparciu o przydzielone do niej rekordy;

dopóki kolejne operacje powodują zmiany w rozkładzie rekordów;

##### Koniec;

Przedstawiony algorytm rozpoczyna się od losowego wybrania  $k$  rekordów, które przez chwilę będą pełniły rolę środków (średnich)  $m_1, \dots, m_k$  odpowiednich grup:  $C_1, \dots, C_k$ . Wszystkie rekordy są umieszczane w odpowiednich grupach według odległości dzielących te rekordy od środków (średnich) grup. Jeśli odległość pomiędzy środkiem  $m_i$  grupy  $C_i$  a rekordem  $r_j$  jest najmniejsza spośród wszystkich środków grup, rekord  $r_j$  jest umieszczany właśnie w grupie  $C_i$ . Kiedy wszystkie rekordy zostaną wstępnie umieszczone w poszczególnych grupach, algorytm ponownie wyznacza środek (średnią) każdej z grup. Cały proces jest następnie powtarzany — algorytm ponownie analizuje każdy z rekordów i umieszcza go w grupie, której środek znajduje się najbliżej danego rekordu. Kompletnie grupowanie rekordów może wymagać wykonania wielu takich iteracji, jednak z każdym kolejnym przebiegiem algorytm zbliża się do ostatecznego wyniku (choć jego działanie może się zakończyć po osiągnięciu lokalnego optimum). Warunkiem zakończenia pracy algorytmu zazwyczaj jest kryterium kwadratu błędu. Dla grup  $C_1, \dots, C_k$  ze średnimi  $m_1, \dots, m_k$  błąd jest definiowany za pomocą następującego wzoru:

$$\text{Błąd} = \sum_{i=1}^k \sum_{\forall r_j \in C_i} \text{Odległość}(r_j, m_i)^2$$

Przeanalizujemy teraz działanie algorytmu 28.4 na przykładzie dwuwymiarowych rekordów z rysunku 28.8. Przyjmijmy, że docelowa liczba grup ( $k$ ) wynosi 2. Załóżmy, że algorytm losowo wybiera rekord z identyfikatorem ( $IdR$ ) 3 dla grupy  $C_1$  oraz rekord z identyfikatorem 6 dla grupy  $C_2$  — oba rekordy będą stanowiły początkowe środki tych grup. Pozostałe rekordy zostaną przydzielone do odpowiednich grup w ramach pierwszej iteracji pętli *powtarzaj-dopóki*. Odległość dzieląca rekord oznaczony identyfikatorem 1 od środka grupy  $C_1$  wynosi 22,4; natomiast odległość pomiędzy tym rekordem a środkiem grupy  $C_2$  wynosi 32,0; zatem rekord ten jest przydzielany do pierwszej grupy. Odległość dzieląca rekord oznaczony identyfikatorem 2 od środka klastra  $C_1$  wynosi 10,0; zaś odległość pomiędzy tym rekordem a środkiem klastra  $C_2$  wynosi tylko 5,0; zatem rekord ten jest przydzielany do bliższej mu grupy  $C_2$ . Rekord z identyfikatorem 4 jest oddalony od grup  $C_1$  i  $C_2$  odpowiednio o 26,9 i 36,1 jednostek, zatem algorytm przydziela go do grupy  $C_1$ . Odległość dzieląca rekord oznaczony identyfikatorem 5 od środka grupy  $C_1$  wynosi 20,6; natomiast odległość pomiędzy tym rekordem a środkiem grupy  $C_2$  wynosi 29,2; zatem rekord ten jest przydzielany do pierwszej grupy. Algorytm musi teraz wyznaczyć nowe średnie (środki) dla obu grup. Średnia dla grupy  $C_1$  zawierającej  $n$  rekordów w  $m$  wymiarach ma postać następującego wektora:

$$\bar{C}_i = \left( \frac{1}{n} \sum_{\forall r_j \in C_i} r_{j1}, \dots, \frac{1}{n} \sum_{\forall r_j \in C_i} r_{jm} \right)$$

IdR	Wiek	Czas współpracy (w latach)
1	30	5
2	50	25
3	50	15
4	25	5
5	30	10
6	55	25

RYSUNEK 28.8. Dwuwymiarowe rekordy wykorzystane w zaprezentowanym przykładzie grupowania danych (kolumna IdR nie jest brana pod uwagę w procesie grupowania)

Nowym środkiem grupy  $C_1$  jest wektor (33,75; 8,75), natomiast nowym środkiem grupy  $C_2$  — wektor (52,5; 25). W drugiej iteracji grupowana szóstka rekordów zostaje rozdzielona pomiędzy dwie grupy w następujący sposób: rekordy oznaczone identyfikatorami 1, 4 i 5 są umieszczane w grupie  $C_1$ , natomiast rekordy oznaczone identyfikatorami 2, 3 i 6 trafiają do grupy  $C_2$ . Wyznaczone ponownie środki grup  $C_1$  i  $C_2$  wynoszą odpowiednio (28,3; 6,7) oraz (51,7; 21,7). W kolejnej iteracji wszystkie rekordy pozostają w dotychczasowych grupach, zatem działanie algorytmu zostaje zakończone.

W tradycyjnych algorytmach grupowania zakłada się, że cały przetwarzany zbiór danych mieści się w głównej pamięci operacyjnej komputera. Pracownicy wielu ośrodków badawczych skupili się w ostatnim czasie na opracowywaniu algorytmów, które zapewnią niezbędną efektywność i skalowalność także dla bardzo dużych baz danych. Jeden z takich algorytmów został nazwany BIRCH. Jest to algorytm hybrydowy, który przewiduje

zarówno wykorzystywanie technik grupowania hierarchicznego (włącznie z budową drzewiastej reprezentacji danych), jak i stosowanie dodatkowych metod grupowania dla węzłów liści skonstruowanego drzewa. Algorytm BIRCH pobiera na wejściu dwa parametry. Pierwszy określa ilość dostępnej pamięci operacyjnej, drugi definiuje początkowy próg maksymalnego promienia generowanych grup. W głównej pamięci operacyjnej składa się opisowe informacje o grupach, w tym środek (średnią) każdej z grup oraz ich promienie (zakłada się, że grupy mają kształt sferyczny). Przekazany na wejściu próg promienia wpływa oczywiście na liczbę wygenerowanych grup. Przykładowo, jeśli wartość tego progu jest duża, wówczas stosunkowo nieliczne grupy będą zawierały po wiele rekordów. Algorytm próbuje utrzymywać taką liczbę grup, która nie będzie prowadziła do przekroczenia ustalonego z góry progu promienia. Jeśli ilość dostępnej pamięci operacyjnej okaże się niewystarczająca, wówczas próg promienia jest podwyższany.

Algorytm BIRCH sekwencyjnie odczytuje rekordy danych i wstawia je kolejno do przechowywanej w pamięci operacyjnej struktury drzewiastej, która w założeniu ma utrwalić strukturę podziału danych. Rekordy są wstawiane do odpowiednich węzłów liści (potencjalnych grup) w oparciu o obliczone odległości dzielące te rekordy od punktów centralnych grup. Dla węzła liścia, do którego wstawiono nowy rekord, może zaistnieć konieczność dalszego podziału w zależności od zaktualizowanej pozycji punktu centralnego oraz promienia grupy (i jego relacji względem ustalonego parametru progowego). Co więcej, ewentualny podział węzła liścia wiąże się z koniecznością umieszczenia w pamięci dodatkowych informacji — jeśli dostępna przestrzeń pamięciowa zostanie wówczas wyczerpana, należy zwiększyć próg minimalnego promienia grup. Zwiększenie progu promienia może z kolei skutkować ograniczeniem liczby grup, ponieważ część węzłów drzewa może zostać scalona.

Podsumowując, algorytm BIRCH jest efektywną metodą grupowania danych z liniową złożonością obliczeniową (liniowo zależną od liczby rekordów przeznaczonych do pogrupowania).

## 28.5. Strategie rozwiązywania pozostałych problemów związanych z eksploracją danych

### 28.5.1. Odkrywanie wzorców sekwencyjnych

Odkrywanie wzorców sekwencyjnych opiera się na pojęciu sekwencji zbiorów elementów. Zakładamy, że takie transakcje jak omawiane do tej pory zakupy w supermarkecie (a więc zgodne z modelem koszyka klienta supermarketu) można łatwo uporządkować według czasu dokonania zakupów. Każde takie uporządkowanie prowadzi do otrzymania sekwencji zbiorów elementów. Przykładowo, taka **sekwencja zbiorów elementów** (oparta w tym przypadku na trzech kolejnych wizytach w sklepie tego samego klienta) może mieć następującą postać: {mleko, chleb, sok}, {chleb, jajka}, {ciastka, mleko, kawa}. **Poziom wsparcia** dla sekwencji  $S$  zbiorów elementów jest odsetkiem danego zbioru sekwencji  $U$ , dla którego zbiór  $S$  jest podsekwencją. W przedstawionym przykładzie zbiory {mleko, chleb, sok}{chleb,

jajka} oraz {chleb, jajka}{ciastka, mleko, kawa} są traktowane właśnie jak **podsekwencje**. Problem identyfikowania wzorców sekwencyjnych polega więc na znajdowaniu wszystkich takich podsekwencji względem danych na wejściu zbiorów sekwencji, których poziom wsparcia dorównuje lub przewyższa zdefiniowany przez użytkownika minimalny próg wsparcia. Sekwencja  $S_1, S_2, S_3, \dots$  jest **predyktorem** (ang. *predictor*) faktu, iż klient kupujący zbiór elementów  $S_1$  najprawdopodobniej kupi także zbiór elementów  $S_2$ , następnie  $S_3$ , itd. Tego typu przewidywania bazują na częstości (wsparciu) tej sekwencji w przeszłości. Istnieje wiele różnych algorytmów zaprojektowanych specjalnie z myślą o wykrywaniu sekwencji w danych.

## 28.5.2. Odkrywanie wzorców w szeregach czasowych

**Szeregi czasowe** są sekwencjami zdarzeń, z których każde może należeć do określonego typu transakcji. Przykładowo, kursy zamknięcia dla akcji lub udziałów funduszu inwestycyjnego są zdarzeniami rejestrowanymi codziennie (w każdym dniu roboczym) dla każdej akcji i każdego udziału. Sekwencja takich wartości dla akcji spółki giełdowej lub funduszu inwestycyjnego tworzy szereg czasowy. Okazuje się, że szeregi czasowe stwarzają duże pole dla poszukiwania rozmaitych wzorców — głównie przez analizę sekwencji i podsekwencji (jak w poprzednim punkcie). Przykładowo, moglibyśmy spróbować znaleźć okres, w czasie którego kurs akcji wzrastał lub pozostawał na tym samym poziomie przez  $n$  dni, najdłuższy okres, w którym zmiana ceny względem poprzedniego kursu zamknięcia nie przekraczała 1%, lub kwartał, w którym dane akcje przyniosły procentowo największy zysk lub największą stratę. Szeregi czasowe można ze sobą porównywać w oparciu o ustalone mierniki podobieństwa — dzięki nim można np. zidentyfikować firmy, których kurs akcji kształtuje się w podobny sposób. Analiza i eksploracja szeregów czasowych stanowią rozszerzoną funkcjonalność zarządzania danymi czasowymi (patrz rozdział 26.).

## 28.5.3. Regresja

*Regresja* jest jednym ze specjalnych zastosowań reguły klasyfikacji. Jeśli regułę klasyfikacji potraktujemy jako funkcję zależną od zmiennych, która odwzorowuje te zmienne w docelową zmienną klasy, możemy tę regułę nazwać **regułą regresji**. Z ogólnym zastosowaniem regresji mamy do czynienia wtedy, gdy — zamiast odwzorowywać krotkę danych z relacji do określonej klasy — na podstawie tej krotki przewidujemy wartość zmiennej. Przykładowo, przeanalizujemy relację:

TESTY\_LAB(id pacjenta, badanie 1, badanie 2, ..., badanie  $n$ )

która zawiera wartości zgromadzone podczas przeprowadzania serii  $n$  badań jednego pacjenta. Zmienną docelową, której wartość chcielibyśmy przewidzieć, jest  $P$ , czyli prawdopodobieństwo przeżycia pacjenta. W takim przypadku nasza reguła dla regresji powinna mieć następującą postać:

(badanie 1 w przedziale<sub>1</sub>) i (badanie 2 w przedziale<sub>2</sub>) i ... i  
(badanie  $n$  w przedziale <sub>$n$</sub> )  $\Rightarrow P = x$

lub

$x < P \leq y$



Alternatywa po prawej stronie powyższej reguły zależy od tego, czy możemy przewidzieć unikatową wartość zmiennej  $P$ , czy tylko przedział wartości tej zmiennej. Jeśli przyjmiemy, że  $P$  jest funkcją w postaci:

$$P = f(\text{test 1, test 2, ..., test } n)$$

będziemy ją mogli nazywać **funkcją regresji** dla przewidywanej wartości  $P$ . Ogólnie, jeśli funkcja ma postać:

$$Y = f(x_1, x_2, \dots, x_n)$$

oraz  $f$  jest funkcją liniową zależną od zmiennych  $x_i$ , proces wyznaczania wartości tej funkcji na podstawie danego na wejściu zbioru krotek  $\langle x_1, x_2, \dots, x_n, y \rangle$  jest nazywany **regresją liniową**. Metoda regresji liniowej jest często wykorzystywana w technikach statystycznych do dopasowywania zbioru obserwacji lub punktów w  $n$  wymiarach do docelowej zmiennej  $y$ .

Analiza regresji jest bardzo popularnym narzędziem w rękach osób interpretujących dane w wielu obszarach badań. Odkrycie funkcji, która umożliwi przewidywanie wartości docelowej zmiennej, jest w tym przypadku odpowiednikiem operacji eksploracji danych.

## 28.5.4. Sieci neuronowe

**Sieci neuronowe** są jedną z technik przetwarzania danych z badań nad sztuczną inteligencją; technika ta opiera się na tzw. regresji uogólnionej i zapewnia iteracyjną metodę jej wyznaczania. Sieci neuronowe wykorzystują mechanizm dopasowywania krzywej do wywodzenia funkcji na podstawie danego na wejściu zbioru próbek. Jest to przykład *rozwiązania opartego na uczeniu*, gdzie początkiem procesu wnioskowania i uczenia jest przekazanie specjalnej próbki testowej. Zastosowanie takiej metody uczenia oznacza, że odpowiedzi generowane dla nowych danych wejściowych mogą być przewidywane na podstawie znanych próbek. Takie przybliżone wyniki zależą jednak od rzeczywistego modelu (wewnętrznej reprezentacji dziedziny problemu) opracowanego przez metodę uczenia.

Sieci neuronowe można podzielić na dwie ogólne kategorie: sieci nadzorowane i sieci nienadzorowane. Metody adaptacyjne, które próbują ograniczać liczbę generowanych na wyjściu błędów (przez generowanie i weryfikowanie próbek), są nazywane metodami **uczenia nadzorowanego**; natomiast techniki, które opracowują wewnętrzne reprezentacje bez generowania przykładowych danych wyjściowych, są nazywane metodami **uczenia nienadzorowanymi**.

Sieci neuronowe same się przystosowują (udoskonalają), co oznacza, że mogą podlegać procesowi uczenia na podstawie informacji dotyczących określonego problemu. Ponieważ sieci neuronowe doskonale sprawdzają się w zadaniach klasyfikacji, stanowią przydatne narzędzie podczas eksploracji danych. Nie są one jednak pozbawione problemów. Warto pamiętać, że chociaż sieci neuronowe *się uczą*, nie zapewniają dobrej reprezentacji *tego, czego* zdołały się nauczyć. Generowane przez nie dane wyjściowe mają zwykle postać trudnych do zrozumienia informacji ilościowych. Kolejnym ograniczeniem jest fakt, że budowane przez sieci neuronowe wewnętrzne reprezentacje nie są unikatowe. Należy także zaznaczyć, że sieci neuronowe słabo radzą sobie z modelowaniem danych mających postać szeregów czasowych. Mimo tych utrudnień sieci neuronowe są coraz bardziej popularne i coraz częściej znajdują zastosowanie w rozwiązaniach tworzonych przez producentów komercyjnych.

### 28.5.5. Algorytmy genetyczne

**Algorytmy genetyczne** (ang. *Genetic Algorithms* — *GA*) stanowią klasę dobranych losowo procedur przeszukiwania, które zapewniają możliwość adaptacyjnego i zaawansowanego przeglądania zróżnicowanych topologii przestrzeni przeszukiwania. Modelowane wraz z rozwojem biologicznych badań nad mechanizmami ewolucyjnymi i wprowadzone po raz pierwszy przez Hollanda<sup>6</sup>, algorytmy genetyczne znalazły zastosowanie w tak zróżnicowanych dziedzinach jak analiza obrazów, szeregowanie zadań i projektowanie inżynierskie.

Idea algorytmów genetycznych pochodzi od koncepcji badania genetyki człowieka z tradycyjnego czteroliterowego alfabetu (wynikającego z istnienia nukleotydów A, C, T i G) ludzkiego kodu genetycznego. Konstrukcja algorytmu genetycznego wiąże się z wynalezieniem takiego alfabetu, który umożliwi kodowanie rozwiązań dla problemu decyzyjnego w postaci ciągów znaków należących do tego alfabetu. Takie ciągi są w tym przypadku reprezentacjami osobników. Funkcja przystosowania definiuje, które rozwiązania mogą przetrwać, a które zostaną odrzucone. Metody łączenia rozwiązań są modelowane po przeprowadzeniu operacji krzyżowania i łączenia ciągów pochodzących od bezpośrednich przodków jednostki. W ten sposób powstaje początkowa, mocno zróżnicowana populacja — od tego momentu rozpoczyna się gra ewolucyjna, w której wśród ciągów mogą pojawiać się mutacje. Ciągi te są łączone i generują nowe pokolenie osobników; najlepiej przystosowane osobniki przeżywają i mutują tak długo, aż powstanie rodzina skutecznych rozwiązań.

Rozwiązania generowane przez algorytmy genetyczne (GA) różnią się od wyników zwracanych przez większość innych technik przeszukiwania następującymi własnościami:

- Mechanizm przeszukiwania stosowany w algorytmie genetycznym wykorzystuje w każdym kolejnym pokoleniu całe zbiory rozwiązań, nie pojedyncze rozwiązania.
- Przeszukiwanie w przestrzeni ciągów reprezentuje znacznie większe możliwości w zakresie przetwarzania równoległego niż przeszukiwanie przestrzeni zakodowanych rozwiązań.
- Pamięć przeszukiwania jest reprezentowana jedynie przez zbiór rozwiązań dostępnych dla danego pokolenia.
- Algorytm genetyczny jest algorytmem losowym, ponieważ mechanizmy przeszukiwania wykorzystują operatory probabilistyczne.
- Podczas przechodzenia pomiędzy kolejnymi pokoleniami algorytm genetyczny odnajduje bliską optymalnej równowagę pomiędzy gromadzeniem a eksploatacją wiedzy przez manipulowanie zakodowanymi rozwiązaniami.

Algorytmy genetyczne są wykorzystywane do rozwiązywania problemów i podczas grupowania danych. Oferowane przez nie możliwości w zakresie równoległego rozwiązywania problemów czynią z nich bardzo przydatne narzędzia także w dziedzinie eksploracji danych. Do niewątpliwych wad algorytmów tego typu należy duża nadprodukcja pojedynczych rozwiązań, losowy charakter procesu przeszukiwania oraz wysokie wymagania odnośnie przetwarzania komputerowego. Ogólnie rzecz biorąc, otrzymanie jakichkolwiek istotnych informacji za pomocą algorytmu genetycznego wymaga znacznego obciążenia dostępnej mocy przetwarzania komputera.

---

<sup>6</sup> Ideę algorytmów genetycznych wprowadził Holland (1975) w swojej przełomowej pracy pt. *Adaptation in Natural and Artificial Systems*.

## 28.6. Zastosowania technik eksploracji danych

Technologie eksploracji danych mogą być z powodzeniem wykorzystywane do wspomagania decyzji w bardzo zróżnicowanych uwarunkowaniach biznesowych. W szczególności, do przewidywanych obszarów zastosowań, w których znaczenie tego typu technologii będzie stale wzrastało, należą:

- **Marketing** — zastosowania technologii eksploracji obejmują tu takie działania jak analiza zachowań klienta w oparciu o wzorce zakupów (w tym określanie możliwie skutecznych strategii marketingowych włącznie z akcjami reklamowymi, planowanie rozmieszczenia sklepów oraz wskazywanie grup odbiorców dla materiałów reklamowych przesyłanych pocztą); segmentacja klientów, sklepów i produktów oraz projektowanie katalogów, wnętrz sklepów czy kampanii reklamowych.
- **Finanse** — zastosowania metod eksploracji danych w finansach obejmują analizę zdolności kredytowej klientów, segmentację należności, analizę wyników finansowych uzyskanych z inwestycji (np. w akcje, bony skarbowe czy fundusze), ocenę sposobów finansowania oraz wykrywanie nadużyć.
- **Przemysł** — potencjalne zastosowania tego typu technologii w przemyśle mogą polegać na optymalizacji wykorzystania takich zasobów jak maszyny, siła robocza czy materiały; optymalnym projektowaniu procesów produkcyjnych, rozkładu urządzeń czy samych produktów (np. samochodów odpowiadających wymaganiom klientów).
- **Ochrona zdrowia** — zastosowania eksploracji danych w medycynie obejmują odkrywanie wzorców w obrazach radiologicznych, analizę eksperymentalnych danych z biochipów na potrzeby grupowania danych i wiązanie ich z zaobserwowanymi schorzeniami, analizę efektów ubocznych leków oraz badanie efektywności poszczególnych kuracji; optymalizację procesów w ramach szpitala czy odkrywanie związków pomiędzy danymi o stanie zdrowia pacjentów a kwalifikacjami opiekujących się nimi personelu.

## 28.7. Komercyjne narzędzia eksploracji danych

Komercyjne narzędzia przeznaczone do eksploracji danych wykorzystują obecnie wiele popularnych technik odkrywania wiedzy. Należą do nich między innymi reguły asocjacyjne, grupowanie, sieci neuronowe, poszukiwanie sekwencji oraz analiza statystyczna. Większość tych technik omówiliśmy wcześniej. Oprogramowanie tego typu często wykorzystuje także drzewa decyzyjne, które stanowią wygodną reprezentację dla reguł stosowanych podczas klasyfikowania lub grupowania; stosuje się też analizę statystyczną, która może obejmować regresję i wiele innych przydatnych technik. Współczesne komercyjne produkty wykorzystują takie zaawansowane metody jak algorytmy genetyczne, wnioskowanie na podstawie przykładów, sieci Bayesa, regresja nieliniowa, optymalizacja kombinatoryczna, dopasowywanie wzorców oraz logika rozmyta. O niektórych z tych metod wspominaliśmy w tym rozdziale.

Większość narzędzi eksploracji danych wykorzystuje interfejs *ODBC* (ang. *Open DataBase Connectivity*). Interfejs ten jest uważany za standardowy mechanizm pośredniczący w dostępie do baz danych — umożliwia współpracę aplikacji z większością popularnych pakietów tego typu, w tym z bazami danych Access, dBASE, Informix, Oracle i SQL Server. Niektóre z tych pakietów oprogramowania zawierają dodatkowo interfejsy zaprojektowane specjalnie z myślą o konkretnych rozwiązaniach: do najbardziej popularnych należą sterowniki baz danych Oracle, Access oraz SQL Server. Większość opisanych w tym podrozdziale narzędzi pracuje w środowisku Microsoft Windows, kilka działa w systemie operacyjnym UNIX. Obecnie można zaobserwować dążenie do dostosowania niemal wszystkich produktów do współpracy z dominującym środowiskiem Microsoft Windows. W przypadku jednego z narzędzi (Data Surveyor) wspomnieliśmy o zgodności z obiektywnym standardem ODMG — szczegółowe omówienie tego standardu można znaleźć w rozdziale 12.

Wymienione programy zostały zaprojektowane z myślą o przetwarzaniu szeregowym na pojedynczym komputerze. W wielu przypadkach możliwa jest praca w trybie klient-serwer. Istnieją także produkty umożliwiające przetwarzanie danych w architekturach równoległych i działanie w ramach narzędzi *OLAP* (ang. *OnLine Analytical Processing*).

### 28.7.1. Interfejs użytkownika

Większość narzędzi oferuje środowisko z graficznym interfejsem użytkownika (ang. *Graphical User Interface* — *GUI*). Niektóre produkty dodatkowo implementują wyszukane techniki wizualizacji danych i reguł (np. MineSet firmy SGI), a nawet możliwość interaktywnego manipulowania danymi. Interfejsy tekstowe są spotykane coraz rzadziej i dotyczą raczej narzędzi dostępnych dla platformy UNIX (np. pakietu Intelligent Miner firmy IBM).

### 28.7.2. Interfejs programowy aplikacji

Interfejs programowy aplikacji (ang. *Application Programming Interface* — *API*) jest przeważnie narzędziem opcjonalnym. Większość produktów nie zezwala na wywoływanie ich wewnętrznych funkcji. Niektóre narzędzia oferują jednak programistom aplikacji możliwość ponownego wykorzystywania ich kodu. Do najbardziej popularnych interfejsów tego typu należą biblioteki języka programowania C oraz biblioteki dołączane dynamicznie (ang. *Dynamic Link Libraries* — *DLL*). Niektóre narzędzia zawierają nawet specjalne języki poleceń dla baz danych.

W tabeli 28.1 wymieniliśmy jedenaście reprezentatywnych narzędzi do eksploracji danych. Obecnie różni producenci na całym świecie oferują setki komercyjnych pakietów stworzonych właśnie z myślą o eksploracji danych. Do walki o ten rynek oprogramowania włączyły się firmy spoza Stanów Zjednoczonych: produkty Data Surveyor i Polyanalyst pochodzą odpowiednio z Holandii i Rosji.

### 28.7.3. Kierunki przyszłego rozwoju

Narzędzia do eksploracji danych stale ewoluują, a ich twórcy starają się uwzględniać najnowsze osiągnięcia naukowców zajmujących się tą dziedziną. Wiele z tych narzędzi wykorzystuje najnowsze algorytmy pochodzące z takich obszarów badań jak sztuczna inteli-

gencja, statystyka czy optymalizacja. Szybkie przetwarzanie jest obecnie możliwe dzięki współczesnym technikom obsługi baz danych (np. przetwarzaniu rozproszonemu) w architekturach klient-serwer, w równoległych bazach danych oraz w hurtowniach danych. W przyszłości rozwój aplikacji tego typu z pewnością będzie prowadził do pełniejszego wykorzystania możliwości, jakie daje internet. Coraz większą popularność zyskują także rozwiązania hybrydowe, a samo przetwarzanie będzie lepiej wykorzystywało wszystkie dostępne zasoby. Do zwiększenia efektywności w tym zakresie przyczyni się rozwój środowisk przetwarzania równoległego i rozproszonego. Udoskonalenia istniejących rozwiązań są szczególnie istotne z uwagi na stale zwiększającą się ilość informacji składowanych w bazach danych.

TABELA 28.1. Kilka reprezentatywnych narzędzi do eksploracji danych

Firma	Produkt	Technika	Platforma	Interfejs*
Acknosoft	Kate	Drzewa decyzyjne, wnioskowanie na podstawie przykładów	Windows, UNIX	Microsoft Access
Angoss	Knowledge SEEKER	Drzewa decyzyjne, metody statystyczne	Windows	ODBC
Business Objects	Business Miner	Sieci neuronowe, uczenie maszynowe	Windows	ODBC
CrossZ	QueryObject	Analiza statystyczna, algorytm optymalizacji	Windows, MVS, UNIX	ODBC
Data Distilleries	Data Surveyor	Zróżnicowana, może łączyć różne techniki eksploracji danych	UNIX	ODBC, zgodny z ODMG
DBMiner Technology Inc.	DBMiner	Analiza OLAP, asocjacje, klasyfikacja, algorytmy grupujące	Windows	Microsoft 7.0 OLAP Manager
IBM	Intelligent Miner	Klasyfikacja, reguły asocjacyjne, modele przewidywania	UNIX (AIX)	IBM DB2
Megaputer Intelligence	Polyanalyst	Gromadzenie wiedzy symbolicznej, programowanie ewolucyjne	Windows, OS/2	ODBC, Oracle, DB2
NCR	Management Discovery Tool (MDT)	Reguły asocjacyjne	Windows	ODBC
Purple Insight	MineSet	Drzewa decyzyjne, reguły asocjacyjne	UNIX (Irix)	Oracle, Sybase, Informix
SAS	Enterprise Miner	Drzewa decyzyjne, reguły asocjacyjne, sieci neuronowe, regresja, grupowanie	UNIX (Solaris), Windows, Macintosh	ODBC, Oracle, AS/400

\* ODBC: Open DataBase Connectivity; ODMG: Object Data Management Group

Głównym kierunkiem eksploracji danych jest analizowanie terabajtów i petabajtów w tzw. systemach big data, które omówiliśmy w rozdziale 25. Te systemy obejmują własne narzędzia i biblioteki do eksploracji danych, np. Mahout działający na szczegółowo opisanej platformie Hadoop. Obszar eksploracji danych będzie też ściśle powiązany z danymi przechowywanymi w chmurze w hurtowniach danych i udostępnianymi za pomocą

serwerów OLAP do eksploracji wtedy, gdy będą potrzebne. Rozrastanie się multimedialnych baz danych to jedna kwestia; ważne jest też to, że składowanie i wyszukiwanie obrazów to powolne operacje. Warto zwrócić uwagę na malejący koszt pamięci dyskowych, który powoduje, że zwiększa się możliwość składowania i przetwarzania ogromnych ilości danych, także w stosunkowo niewielkich firmach. Z czasem programy do drążenia danych będą musiały obsługiwać coraz większe zbiory danych w coraz większej liczbie przedsiębiorstw.

Większość pakietów oprogramowania obsługującego eksplorację danych będzie wykorzystywała standard ODBC do odczytywania niezbędnych informacji z biznesowych baz danych (można oczekiwać, że standardy wejściowe specyficzne dla poszczególnych producentów i pakietów z czasem będą traciły na znaczeniu). Bezwzględnie konieczne jest opracowanie nowych standardów dla takich obiektów jak obrazy i inne dane multimedialne, które w niedalekiej przyszłości będą stanowiły informacje źródłowe dla procesów eksploracji danych.

## 28.8. Podsumowanie

W tym rozdziale dokonaliśmy przeglądu technik składających się na ważną dziedzinę eksploracji danych, która wykorzystuje technologię baz danych do odkrywania dodatkowej wiedzy lub wzorców zawartych w danych. Przedstawiliśmy także prosty przykład odkrywania wiedzy w bazach danych, które jest działaniem nieco szerszym niż eksploracja danych. Spośród wielu różnych technik eksploracji danych skupiliśmy się na: odkrywaniu reguł asocjacyjnych, klasyfikacji oraz grupowaniu. Dla każdej z nich zaprezentowaliśmy algorytmy, których działanie zilustrowaliśmy za pomocą odpowiednich przykładów.

W tym rozdziale poświęciliśmy także trochę miejsca na krótkie omówienie innych technik, w tym sieci neuronowych (związanych głównie ze sztuczną inteligencją) i algorytmów genetycznych. Ponieważ eksploracja danych wciąż jest przedmiotem aktywnych badań, spróbowaliśmy nakreślić oczekiwane kierunki rozwoju tej technologii. Należy się spodziewać, że w przyszłości technologia baz danych będzie w znacznym stopniu przystosowana właśnie do działań związanych z eksploracją informacji. Na końcu tego rozdziału podsumowaliśmy jedenaście z setek dostępnych narzędzi do eksploracji danych; można jednak oczekiwać, że w przyszłości zarówno ich liczba, jak i funkcjonalność ulegną znacznemu zwiększeniu.

## Pytania powtórkowe

- 28.1. Jakie są różne fazy procesu odkrywania wiedzy w oparciu o bazy danych? Opisz kompletny scenariusz aplikacji, w której na podstawie istniejącej bazy danych o transakcjach może być odkrywana nowa wiedza.
- 28.2. Jakie cele lub zadania stawia się przed technologią eksploracji danych?
- 28.3. Wymień pięć typów wiedzy otrzymywanej w wyniku stosowania technik eksploracji danych.

- 28.4. Czym (w sensie rodzaju odkrywanej wiedzy) są reguły asocjacyjne? Podaj definicję *poziomu wsparcia* i *poziomu ufności* oraz wykorzystaj oba wspomniane elementy do zdefiniowania reguły asocjacyjnej.
- 28.5. Czym jest własność dolnego domknięcia? W jaki sposób ta własność może pomóc w opracowaniu efektywnego algorytmu znajdowania reguł asocjacyjnych (np. w odniesieniu do odnajdywania dużych zbiorów elementów)?
- 28.6. Który czynnik zdecydował o opracowaniu algorytmu odkrywania częstych wzorców w drzewie z myślą o eksploracji reguł asocjacyjnych?
- 28.7. Przedstaw przykład reguły asocjacyjnej pomiędzy hierarchiami i opisz taką regułę.
- 28.8. Czym są negatywne reguły asocjacyjne w kontekście hierarchii z rysunku 28.3?
- 28.9. Z jakimi najważniejszymi utrudnieniami wiąże się eksploracja reguł asocjacyjnych z ogromnych baz danych?
- 28.10. Czym są reguły klasyfikacji? Jakie są związki tych reguł z drzewami decyzyjnymi?
- 28.11. Czym jest entropia i jak może być wykorzystywana w procesie budowania drzew decyzyjnych?
- 28.12. Czym różni się grupowanie od klasyfikacji?
- 28.13. Opisz sieci neuronowe i algorytmy genetyczne w kontekście technik eksploracji danych. Wymień najważniejsze utrudnienia w stosowaniu tych technik.

## Ćwiczenia

- 28.14. Zastosuj algorytm Apriori dla następującego zbioru danych:

Identyfikator transakcji	Kupione towary
101	mleko, chleb, jajka
102	mleko, sok
103	sok, masło
104	mleko, chleb, jajka
105	kawa, jajka
106	kawa
107	kawa, sok
108	mleko, chleb, ciastka, jajka
109	ciastka, masło
110	mleko, chleb

Kompletny zbiór elementów ma następującą postać: {mleko, chleb, ciastka, jajka, masło, kawa, sok}. Użyj wartości 0,2 w roli progu minimalnego poziomu wsparcia.

- 28.15. Przedstaw dwie reguły, których poziom ufności jest większy lub równy 0,7 dla zbioru elementów zawierającego trzy produkty z ćwiczenia 28.14.
- 28.16. Dla algorytmu partycjonującego udowodnij, że każdy znajdujący się w bazie danych częsty zbiór elementów musi występować przynajmniej w jednej partycji w roli lokalnego częstego zbioru elementów.



- 28.17. Przedstaw drzewo częstych wzorców, które można zbudować na podstawie danych przedstawionych w ćwiczeniu 28.14.
- 28.18. Zastosuj algorytm odkrywania częstych wzorców w drzewie skonstruowanym w ćwiczeniu 28.17 i przedstaw częste zbiory elementów.
- 28.19. Zastosuj dla poniższego zbioru rekordów danych algorytm klasyfikacji. Atrybutem podziału na klasy jest w tym przypadku *Dotychczasowy klient*.

IdR	Wiek	Miasto	Płeć	Wykształcenie	Dotychczasowy klient
101	20..30	PO	K	licencjackie	TAK
102	20..30	WR	M	wyższe	TAK
103	31..40	PO	K	licencjackie	TAK
104	51..60	PO	K	licencjackie	NIE
105	31..40	WA	M	średnie	NIE
106	41..50	PO	K	licencjackie	TAK
107	41..50	PO	K	wyższe	TAK
108	20..30	WA	M	licencjackie	TAK
109	20..30	PO	K	średnie	NIE
110	20..30	PO	K	licencjackie	TAK

- 28.20. Przeanalizuj następujący zbiór dwuwymiarowych rekordów:

IdR	Wymiar 1	Wymiar 2
1	8	4
2	5	4
3	2	4
4	2	6
5	2	8
6	8	6

Załóżmy, że mamy także dwa różne schematy grupowania: (1) Grupa<sub>1</sub> zawiera rekordy {1, 2, 3} i Grupa<sub>2</sub> zawiera rekordy {4, 5, 6} oraz (2) Grupa<sub>1</sub> zawiera rekordy {1, 6} i Grupa<sub>2</sub> zawiera rekordy {2, 3, 4, 5}. Który schemat grupowania jest lepszy? Odpowiedź uzasadnij.

- 28.21. Użyj algorytmu k-średnich do pogrupowania danych z ćwiczenia 28.20. Możesz przyjąć, że docelowa liczba grup (k) wynosi 3, oraz że rekordy z identyfikatorami IdR równymi 1, 3 i 5 zostały losowo wybrane do roli początkowych środków (średnich) grup.
- 28.22. Algorytm k-średnich wykorzystuje miarę podobieństwa, którą najczęściej jest po prostu odległość dzieląca rekord od środka grupy. Jeśli jednak wartości atrybutów rekordów nie mają charakteru ilościowego, tylko nominalny — przykładowo, atrybut *Poziom zarobków* z wartościami {niski, średni, wysoki}, atrybut *Żonaty* z wartościami {Tak, Nie} lub atrybut *Województwo* z wartościami {dolnośląskie, kujawsko-pomorskie, ..., zachodniopomorskie} — stosowanie miary odległości nie ma oczywiście racji bytu. Zdefiniuj taką miarę podobieństwa,

która będzie bardziej pasowała do grupowania rekordów zawierających dane nominalne.

## Wybrane publikacje

Literatura poświęcona eksploracji danych wywodzi się i dotyczy rozmaitych dziedzin badań, włącznie ze statystyką, optymalizacją matematyczną, uczeniem maszynowym i sztuczną inteligencją. Chen i in. (1996) przedstawili dobre podsumowanie perspektywy rozwoju baz danych w kierunku eksploracji danych. Książka opracowana przez Hana i Kambera (2006) stanowi doskonałe źródło szczegółowej wiedzy na temat algorytmów i technik stosowanych w obszarze eksploracji danych. W ośrodku badawczym IBM Almaden opracowano znaczną część stosowanych do dzisiaj pojęć i algorytmów, a także przeprowadzono wiele eksperymentów w zakresie wydajności poszczególnych rozwiązań. Agrawal i in. (1993) przedstawili pierwszą poważną rozprawę na temat reguł asocjacyjnych. Algorytm Apriori dla danych o koszyku klienta supermarketu zaproponował Agrawal i Srikant (1994); algorytm ten został udoskonalony przez wprowadzenie mechanizmu partycjonowania w książce Savaserego i in. (1995); Toivonen (1996) przedstawił propozycję próbkowania jako jeden ze sposobów ograniczenia czasu przetwarzania. Cheung i in. (1996) rozszerzyli technikę partycjonowania o możliwość stosowania w środowiskach rozproszonych; Lin i Dunhan (1998) zaproponowali techniki rozwiązywania problemów związanych z błędnymi danymi wejściowymi. Agrawal i in. (1993b) omówili perspektywy rozwoju metod odkrywania reguł asocjacyjnych w kontekście ich efektywności. Mannila i in. (1994), Park i in. (1995) oraz Amir i in. (1997) zaprezentowali kolejne wydajne algorytmy związane z regułami asocjacyjnymi. Han i in. (2000) zaprezentowali omówiony w tym rozdziale algorytm odkrywania częstych wzorców w drzewie. Srikant (1995) zaproponował metodę eksploracji reguł uogólnionych. Savasere i in. (1998) jako pierwsi przedstawili kompletne rozwiązanie w zakresie eksploracji asocjacji negatywnych. Agrawal i in. (1996) opisali system Quest firmy IBM. Sarawagi i in. (1998) przedstawili propozycję implementacji, w której reguły asocjacyjne są zintegrowane z systemem zarządzania bazą danych. Piatesky-Shapiro i Frawley (1992) opublikowali cykl artykułów poświęconych różnym zagadnieniom związanym z odkrywaniem wiedzy. Zhang i in. (1996) zaprezentowali algorytm BIRCH stworzony z myślą o grupowaniu informacji składowanych w ogromnych bazach danych. Informacje o uczeniu drzew decyzyjnych i zaprezentowanym w tym rozdziale algorytmie klasyfikacji można znaleźć w książce Mitchella (1997).

Adriaans i Zantinge (1996), Fayyad i in. (1997) oraz Weiss i Indurkha (1998) poświęcili swoje książki różnym aspektom eksploracji danych i zastosowaniom tych technik w przewidywaniu faktów. Idea algorytmów genetycznych pochodzi jeszcze z książki Hollanda (1975); dobry przegląd tego typu algorytmów można znaleźć w publikacji Srinivasa i Patnaika (1994). Do tej pory wydano mnóstwo książek poświęconych sieciom neuronowym; rozległe wprowadzenie do tej tematyki przedstawił Lippman (1987).

Tan, Steinbach i Kumar (2006) prezentują obszernie wprowadzenie do eksploracji danych ze szczegółowym wykazem literatury. Zachęcamy też Czytelników do zapoznania się z materiałami z dwóch ważnych dorocznych konferencji z obszaru eksploracji danych: KDD (ang. *Knowledge Discovery and Data Mining Conference*), prowadzonej od 1995 r., i SDM (ang. *SIAM International Conference on Data Mining*), organizowanej od 2001 r. Odsyłacze do zakończonych konferencji znajdziesz na stronie <http://dblp.uni-trier.de>.

## Przegląd hurtowni danych i rozwiązań OLAP

Hurtownie danych to bazy, w których dane są zapisywane i przechowywane na potrzeby wspomagania podejmowania decyzji niezależnie od danych z baz transakcyjnych. Zwykle bazy transakcyjne przechowują dane przez ograniczony czas; po jego upływie dane tracą bezpośrednią użyteczność i są archiwizowane. Z kolei hurtownie danych przechowują dane z wielu lat, aby umożliwić analizowanie historycznych informacji. Hurtownie danych zapewniają takie możliwości w zakresie składowania danych, funkcjonalności i wykonywania interaktywnych zapytań, które wykraczają daleko poza możliwości tradycyjnych, transakcyjnych baz danych. Ze stałym wzrostem możliwości związana jest konieczność zwiększenia wydajności dostępu do danych w bazach. We współczesnych organizacjach użytkownicy danych często nie mają bezpośredniego kontaktu z właściwymi źródłami danych. Wielu użytkowników potrzebuje dostępu do danych w trybie tylko do odczytu, przy czym niezbędny jest szybki dostęp do danych w ilościach, jakich nie da się w wygodny sposób pobrać na komputer danej osoby. Wykorzystywane informacje często pochodzą z wielu baz danych. Ponieważ stosunkowo duży odsetek analiz przeprowadzanych przez użytkowników ma powtarzalny i przewidywalny charakter, producenci oprogramowania i pracownicy działów wsparcia przystąpili do projektowania systemów obsługujących takie operacje. W hurtowniach danych stosuje się inne modele i struktury oraz typy technologii składowania i wyszukiwania niż w bazach transakcyjnych. Ponadto z obu typów rozwiązań korzystają inne grupy użytkowników. Kadra zarządzająca przedsiębiorstwami na średnim i wyższych szczeblach zgłasza obecnie ogromne zapotrzebowanie na systemy wspomagające podejmowanie decyzji poprzez udostępnianie odpowiednio szczegółowych informacji. Tego typu funkcjonalność oferują tzw. *hurtownie danych*, rozwiązania oparte na technice *przetwarzania analitycznego dostępnego bezpośrednio* (ang. *OnLine Analytical Processing — OLAP*) oraz *eksploracji danych*. W rozdziale 28. przedstawiliśmy już podstawy technik eksploracji danych. W tym rozdziale zaprezentujemy szeroki przegląd technologii wykorzystujących hurtownie danych i rozwiązania OLAP.

### 29.1. Wprowadzenie, definicje i terminologia

W rozdziale 1. zdefiniowaliśmy *bazę danych* jako zbiór powiązanych ze sobą danych, a *system bazy danych* jako połączenie samej bazy danych i jej oprogramowania. Hurtownia danych także jest zbiorem informacji wraz z obsługującym te informacje oprogramowaniem, jednak istnieje wyraźna różnica pomiędzy hurtowniami danych a systemami baz danych.

Funkcjonowanie tradycyjnych (relacyjnych, obiektowych, sieciowych i hierarchicznych) baz danych opiera się na przetwarzaniu transakcji. *Hurtownie danych* mają nieco inne cechy, które predestynują je przede wszystkim do aplikacji wspomagania decyzji — z reguły są zoptymalizowane do wyszukiwania danych, a nie, jak tradycyjne systemy baz danych, do przetwarzania transakcji.

Ponieważ hurtownie danych zostały wdrożone w bardzo wielu organizacjach z myślą o zaspokajaniu konkretnych potrzeb, nie istnieje jedna, standardowa definicja pojęcia *hurtowni danych*. Artykuły w profesjonalnych czasopismach i książki drukowane przez popularne wydawnictwa zawierają mnóstwo różnych interpretacji tej nazwy. Wielu producentów rozwiązań informatycznych wykorzystuje popularność tego terminu do marketingu produktów, które w wiążą się z hurtowniami danych. Istnieją także całe rzesze konsultantów oferujących rozmaite usługi pod szyldem hurtowni danych. Warto jednak pamiętać, że hurtownie danych różnią się od tradycyjnych baz danych zarówno strukturą, jak i sposobem funkcjonowania, wydajnością oraz przeznaczeniem.

W. H. Inmon<sup>1</sup> scharakteryzował koncepcję **hurtowni danych** w następujący sposób: *skoncentrowany na konkretnym zadaniu, zintegrowany, trwały, zmienny w czasie zbiór danych wykorzystywany do wspomagania podejmowania decyzji*. Hurtownie danych zapewniają dostęp do danych dla takich działań jak przeprowadzanie skomplikowanych analiz, odkrywanie wiedzy i podejmowanie decyzji dzięki zapytaniom **jednorazowym** i **składowanym**. **Zapytania składowane** to z góry zdefiniowane zapytania o parametrach, które mogą często się powtarzać. Takie zapytania pozwalają wydajnie spełniać wymagania dotyczące danych i informacji należących do firmy. Hurtownie danych są wykorzystywane do wielu różnych zastosowań — od rozwiązań OLAP, przez systemy wspomagania decyzji (DSS), aż po eksplorację danych. Dalej definiujemy te zastosowania.

**OLAP** (ang. *OnLine Analytical Processing*) jest pojęciem stosowanym do opisywania analizy skomplikowanych danych składowanych w hurtowni danych. W rękach doświadczonych analityków narzędzia OLAP umożliwiają szybkie i proste kierowanie zapytań o dane analityczne składowane w hurtowniach danych i **składowicach danych** (ang. *data mart*; są to analityczne bazy danych podobne do hurtowni, ale ze zdefiniowanym węższym zakresem danych).

Rozwiązania **DSS** (ang. *Decision-Support Systems*), określane także za pomocą skrótu **EIS** (ang. *Executive Information Systems*; nie należy tego określenia mylić z innym rozwinięciem skrótu *EIS*: *Enterprise Integration Systems*) lub **MIS** (ang. *management information systems*), wspomagają osoby odpowiedzialne za podejmowanie kluczowych decyzji, udostępniające dane (analityczne) wyższego poziomu. Technika eksploracji danych (którą szczegółowo omówiliśmy w rozdziale 28.) jest stosowana w procesie *odkrywania wiedzy*, który polega na przeglądaniu danych w poszukiwaniu nieoczekiwanych nowych wniosków (przypomina to poszukiwanie pereł wiedzy w oceanie danych).

Tradycyjne bazy danych obsługują nie tylko technikę **przetwarzania transakcji bezpośredniego** (ang. *OnLine Transaction Processing — OLTP*), która obejmuje takie operacje jak wstawianie, aktualizacja czy usuwanie danych, ale także wymagania odnośnie informacji zwracanych przez zapytania. Tradycyjne relacyjne bazy danych są

---

<sup>1</sup> Wprowadzenie pojęcia *hurtowni danych* przypisuje się właśnie Inmonowi (1992). Praca Inmona i in. (2008) jest zatytułowana *DW 2.0: The architecture for the next generation of Data Warehousing*.

optymalizowane do przetwarzania zapytań, które mogą dotyczyć stosunkowo niewielkiej części bazy danych, oraz do wykonywania transakcji wstawiających lub aktualizujących kilka krotek przetwarzanej relacji. Oznacza to, że systemów tego typu nie można w prosty sposób przygotować do efektywnego działania w roli rozwiązań OLAP, DSS czy eksploracji danych. Zupełnie inaczej jest w przypadku hurtowni danych, które są projektowane specjalnie do obsługi efektywnego wyszukiwania, przetwarzania i prezentowania danych do takich celów jak ich dalsza analiza lub wykorzystanie podczas wspomagania decyzji. W porównaniu z tradycyjnymi bazami danych, hurtownie danych zawierają ogromną ilość danych pochodzących z różnych źródeł, które mogą obejmować bazy danych zbudowane w oparciu o różne modele danych, a niekiedy także pliki składowane w niezależnych systemach i platformach.

## 29.2. Właściwości hurtowni danych

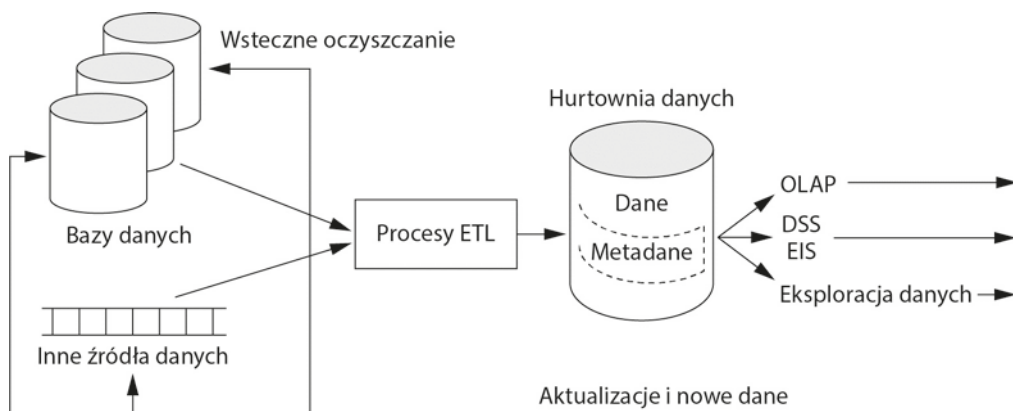
Omówienie hurtowni danych i elementów, które je odróżniają od transakcyjnych baz danych, wymaga przeanalizowania odpowiedniego modelu danych. W rozwiązaniach OLAP i technologiach wspomagania decyzji dobrze sprawdza się wielowymiarowy model danych (który szczegółowo omówimy w podrozdziale 29.3). W przeciwieństwie do systemów wielobazowych (ang. *multi-database*), które oferują dostęp do różnych i często heterogenicznych baz danych, hurtownia danych jest często miejscem składowania zintegrowanych danych pochodzących z wielu źródeł — dane te są zwykle przetworzone do postaci umożliwiających ich przechowywanie zgodnie z modelem wielowymiarowym. Inaczej niż w większości transakcyjnych baz danych, w rozwiązaniach opartych na hurtowniach danych zazwyczaj obsługiwane są analizy szeregów czasowych i trendów (w tym analizy typu „co, jeśli” i analizy prognostyczne), które wymagają znacznie większej ilości danych historycznych, niż zazwyczaj utrzymuje się w transakcyjnych bazach danych.

W porównaniu z transakcyjnymi bazami danych hurtownie danych są znacznie bardziej stabilne. Oznacza to, że informacje w nich zawarte zwykle się nie zmieniają. Takie dane są często nazywane danymi tylko do odczytu, dodawania i oczyszczania. Hurtownię danych można traktować jak system niedziałający w czasie rzeczywistym, w którym okresowo zapisywane są dane. Transakcje w systemach transakcyjnych są jednostkami przetwarzania przeznaczonymi do zmiany stanu bazy danych; inaczej jest w hurtowniach danych, gdzie aktualizacje dotyczą znacznie większych zbiorów informacji i są przeprowadzane zgodnie z dokładnie przemyślaną strategią odświeżania (zwykle w sposób przyrostowy). Operacje aktualizacji informacji składowanych w hurtowniach danych są obsługiwane przez proces **ETL**, który wykonuje szeroko zakrojone wstępne przetwarzanie i jest widoczny na rysunku 29.1. Pojęcie hurtowni danych można także opisać bardziej ogólnymi słowami: *jest to zbiór technologii wspomagania decyzji przeznaczonych dla pracowników wykorzystujących wiedzę (dyrektorów wykonawczych, menadżerów, analityków) do podejmowania lepszych i szybszych decyzji*<sup>2</sup>. Na rysunku 29.1 przedstawiono ogólny przegląd koncepcyjnej struktury hurtowni danych. Przedstawiony schemat obejmuje cały proces pracy hurtowni danych. Proces ten składa się z możliwych operacji oczyszczania i po-

---

<sup>2</sup> Doskonały opis tego zagadnienia — oparty właśnie na tej definicji początkowej — opracowali i opisali Chaudhuri i Dayal (1997).

nowego formatowania danych przed umieszczeniem w hurtowni danych. Proces ten jest wykonywany za pomocą narzędzi ETL. W tle tego procesu istnieje możliwość wykorzystywania technik OLAP, eksploracji danych i DSS do generowania na podstawie zapisanych danych nowych informacji (np. reguł lub innych metadanych), które — co także przedstawiono na rysunku 29.1 — wracają do hurtowni danych. Rysunek pokazuje również, że do źródeł informacji dla hurtowni danych mogą należeć także pliki.



RYSUNEK 29.1. Ogólna struktura hurtowni danych

Oto ważne cechy hurtowni danych, podane razem z definicją pojęcia OLAP w 1993 r. i mające zastosowanie także dziś<sup>3</sup>:

- wielowymiarowa perspektywa koncepcyjna,
- nieograniczona liczba wymiarów i poziomów agregacji,
- nieograniczone operacje międzywymiarowe,
- dynamiczna obsługa rzadkich macierzy,
- architektura klient-serwer,
- obsługa jednoczesnego dostępu wielu użytkowników,
- dostępność,
- przezroczystość,
- intuicyjny interfejs manipulowania danymi,
- analizy indukcyjne i dedukcyjne,
- elastyczność generowanych raportów.

Ponieważ hurtownie danych obejmują zwykle ogromne ilości danych, zasadniczo są one większe od źródłowych baz danych o rząd (lub nawet dwa rzędy) wielkości. Ilość danych (często wyrażana w terabajtach, a nawet petabajtach) powoduje, że efektywne zarządzanie informacjami wymaga stosowania korporacyjnych hurtowni danych, wirtualnych hurtowni danych, logicznych hurtowni danych oraz składnic danych (nazywanych także tematycznymi hurtowniami danych):

<sup>3</sup> Nazwę *OLAP* wprowadzili Codd i Salley (1993), którzy wymienili podane tu cechy.



- **Korporacyjne hurtownie danych** mają zwykle postać ogromnych projektów wymagających poważnych inwestycji zarówno w zakresie czasu, jak i środków.
- **Wirtualne hurtownie danych** oferują perspektywy reprezentujące zawartość wykorzystywanych na co dzień operacyjnych baz danych — dla zapewnienia wydajności perspektywy te są zwykle materializowane.
- **Logiczne hurtownie danych** wykorzystują federacje, rozpraszanie danych i techniki wirtualizacji.
- **Składowe dane** (ang. *data marts*) są generalnie przeznaczone dla wyznaczonych jednostek organizacyjnych (np. działów firmy) i są ściślej związane z obszarami zainteresowań tych jednostek.

Oto inne określenia często spotykane w kontekście hurtowni danych:

- **Operacyjne zbiory danych (OZD)**. Ta nazwa jest często stosowana dla pośrednich wersji bazy przed oczyszczeniem, agregacją i przekształceniem danych zapisywanych w bazie.
- **Analityczne zbiory danych (AZD)**. Są to bazy budowane na potrzeby analizy danych. Zwykle bazy OZD są rekonfigurowane i zmieniane w bazy AZD w procesie oczyszczania, agregowania i przekształcania danych.

## 29.3. Modelowanie danych dla hurtowni danych

Modele wielowymiarowe wykorzystują istniejące związki łączące dane do przekazywania danych w wielowymiarowych macierzach nazywanych *kostkami danych* (w przypadku macierzy więcej niż trójwymiarowych, często stosuje się określenie *hiperkostek danych*). Dane w naturalny sposób pasujące do formatowania wymiarowego mogą być przetwarzane przez zapytania wykonywane w wielowymiarowej macierzy znacznie efektywniej niż w relacyjnym modelu danych. Przykładami trzech wymiarów w korporacyjnej hurtowni danych mogą być okresy rozliczeń podatkowych, produkty oraz obszary działalności.

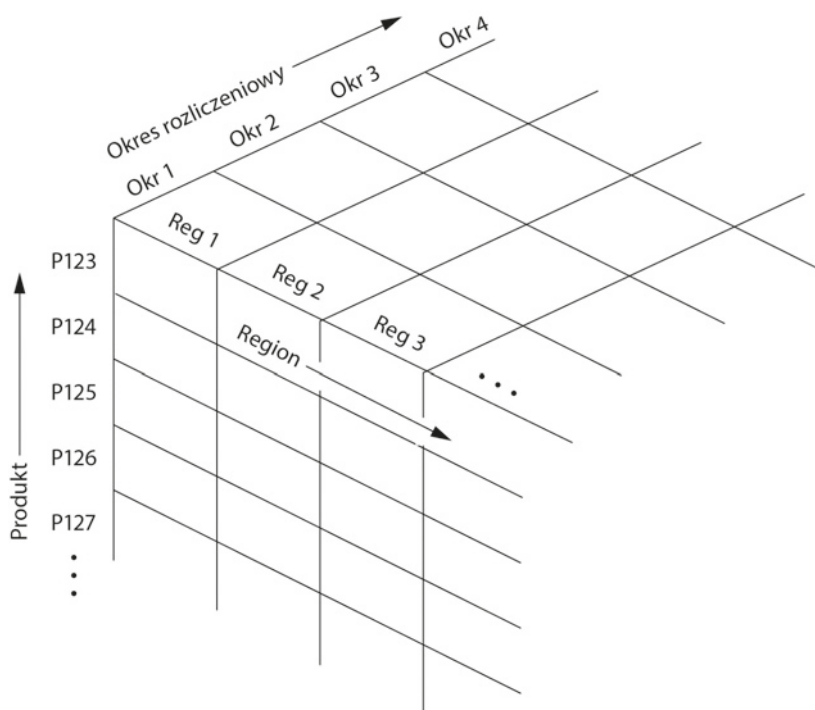
Standardowy arkusz kalkulacyjny jest oczywiście macierzą dwuwymiarową. Przykładem takiej macierzy może być arkusz kalkulacyjny reprezentujący dane o regionalnej sprzedaży poszczególnych produktów firmy w określonym czasie. Produkty mogą być reprezentowane przez wiersze, natomiast wartości sprzedaży dla poszczególnych obszarów mogą mieć postać kolumn (taką dwuwymiarową organizację tego arkusza przedstawiono na rysunku 29.2). Dodanie do tej macierzy wymiaru czasu (np. okresów rozliczeniowych z urzędem skarbowym) byłoby równoznaczne ze stworzeniem macierzy trójwymiarowej, która może być reprezentowana za pomocą kostki danych.

Na rysunku 29.3 przedstawiono trójwymiarową kostkę danych organizującą dane o sprzedaży produktów według okresów rozliczeniowych z urzędem skarbowym i regionów objętych działalnością firmy. Każda komórka tej kostki może zawierać dane dla określonego produktu, określonego okresu rozliczeniowego oraz określonego regionu. Dołączenie dodatkowego wymiaru spowodowałoby powstanie hiperkostki, warto jednak pamiętać, że przekroczenie liczby trzech wymiarów uniemożliwiłoby wizualizację i graficzne prezentowanie otrzymanej struktury. Przechowywane w ten sposób dane mogą być bezpośrednio



	Region			
	Reg 1	Reg 2	Reg 3	...
Produkt	P123			
	P124			
	P125			
	P126			
	⋮			

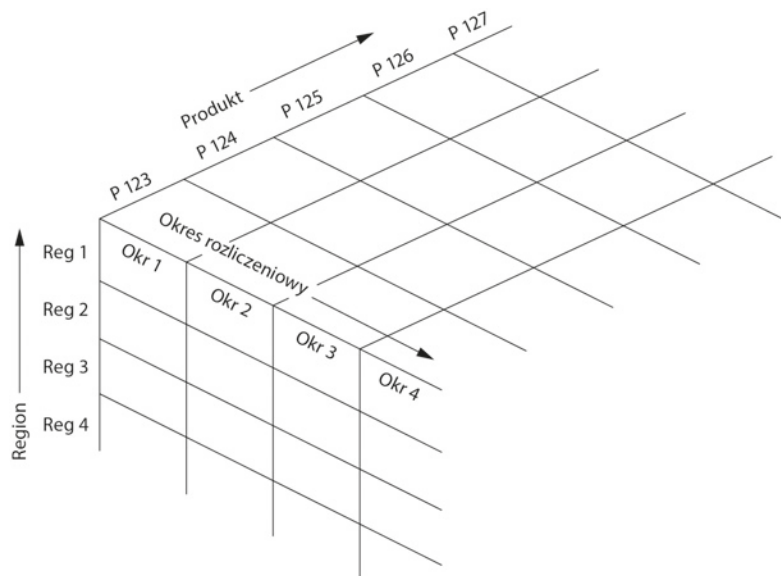
RYSUNEK 29.2. Model macierzy dwuwymiarowej



RYSUNEK 29.3. Model trójwymiarowej kostki danych

przetwarzane za pomocą zapytań operujących na dowolnych kombinacjach wymiarów, co pozwala uniknąć skomplikowanych zapytań wykonywanych na bazach danych. Istnieją nawet narzędzia do przeglądania danych zgodnie z wybranymi przez użytkownika wymiarami.

Zmiana jednej hierarchii wymiarowej (orientacji) na inną jest bardzo prosta — wykorzystuje się w tym celu technikę nazywaną **przestawianiem** (lub *obracaniem*). Technika ta przewiduje możliwość obracania kostki danych w taki sposób, aby przedstawiała różne orientacje istniejących osi. Przykładowo, możemy tak przestawić kostkę danych, aby zawierała przychody ze sprzedaży regionalnej w postaci wierszy, sumy sprzedaży w kolejnych kwartałach w postaci kolumn oraz produkty firmy w trzecim wymiarze (patrz rysunek 29.4). Technika ta jest więc równoważna posiadaniu osobnych tabel sprzedaży regionalnej dla każdego produktu, gdzie każda z tabel zawiera dane o wartości sprzedaży jednego produktu w określonych regionach i w określonych okresach rozliczeniowych. Dwuwymiarowy obraz kostki o trzech wymiarach (lub większej ich liczbie) to **wycinek** (ang. *slice*). Dwuwymiarowy widok dla produktów i regionów na rysunku 29.2 to wycinek trójwymiarowej kostki z rysunku 29.3. Określenie „przecinanie i rzutowanie” oznacza systematyczną redukcję danych do postaci ich mniejszych porcji lub obrazów, co pozwala zobaczyć informacje z różnych perspektyw.



Rysunek 29.4. Przestawiona wersja kostki danych z rysunku 29.3

Modele wielowymiarowe prowadzą do perspektyw hierarchicznych, w których dane są wyświetlane zgodnie z techniką zwijania (ang. *roll-up*) i rozwijania (ang. *drill-down*). **Widok zwinięty** przenosi hierarchię na wyższy poziom i grupuje dane w poszczególnych wymiarach w większe jednostki (np. sumuje dane tygodniowe do postaci zestawień kwartalnych lub rocznych). Taką perspektywę przedstawiono na rysunku 29.5, gdzie poszczególne produkty zostały pogrupowane w bardziej ogólnych kategoriach produktów. Przedstawiony na rysunku 29.6 **widok rozwinięty** oferuje dokładnie odwrotne rozwiązanie — rozбивa dotychczasowe dane do postaci bardziej szczegółowej (w tym przypadku rozkłada informacje o sprzedaży w poszczególnych państwach na odpowiednie informacje o wynikach w konkretnych regionach tych państw, a poszczególne produkty rozбивa na podgrupy podzielone według stylów. W hurtowniach danych możliwość **rozwijania** jest zwykle ograniczona do najniższego poziomu zagregowanych danych skła-

		Region <span>→</span>		
		Region 1	Region 2	Region 3
Kategorie produktów ↓	Produkty 1XX			
	Produkty 2XX			
	Produkty 3XX			
	Produkty 4XX			

RYSUNEK 29.5. Operacja zwijania

dowanych w danej hurtowni. Przykładowo, na rysunku 29.6 dane z niższego poziomu mogą reprezentować „sumę sprzedaży produktów w podstylu A stylu P123 w kolorze czarnym w podregionie 1 regionu o kodzie 30-022”. Możliwe, że dane zagregowane na tym poziomie są przechowywane w operacyjnym zbiorze danych. Niektóre SZBD, np. Oracle, udostępniają tabele zagnieżdżone. Pozwala to na dostęp do niższych poziomów danych i umożliwia obsługę większej liczby etapów rozwijania.

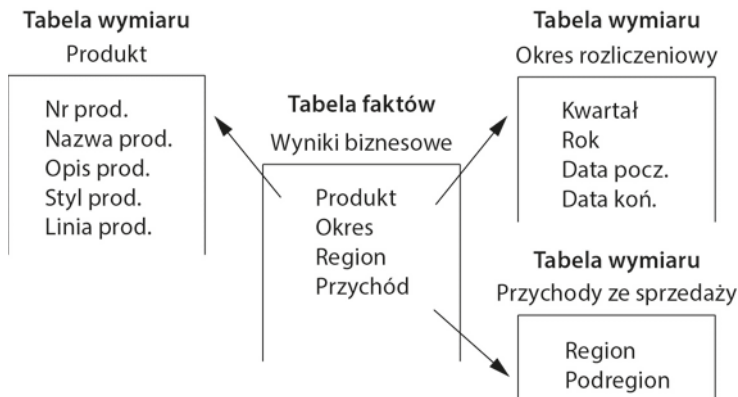
		Region 1				Region 2	
		Podreg 1	Podreg 2	Podreg 3	Podreg 4	Podreg 1	
Style P123	A						
	B						
	C						
	D						
Style P124	A						
	B						
	C						
Style P125	A						
	B						
	C						
	D						

RYSUNEK 29.6. Operacja rozwijania

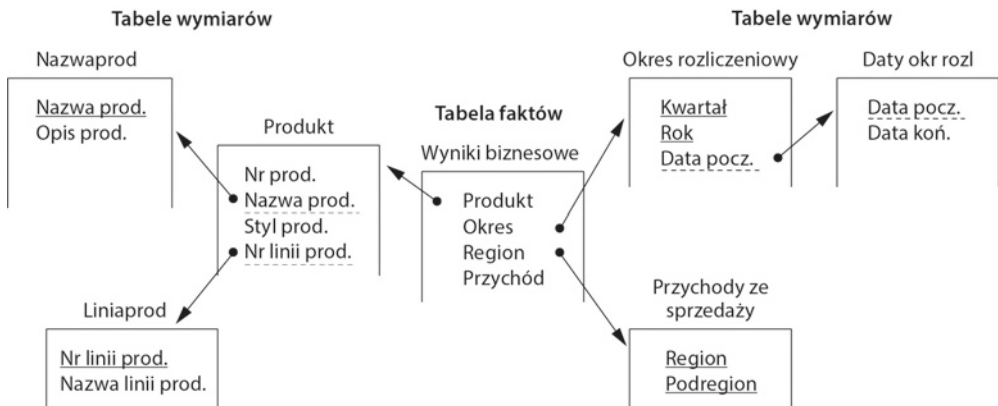
**Wielowymiarowy model** składowania danych wymaga stosowania dwóch rodzajów tabel: tabel wymiarów oraz tabel faktów. **Tabela wymiaru** składa się z krotek atrybutu danego wymiaru. **Tabelę faktów** możemy traktować jako strukturę zawierającą po jednej krotce dla każdego zarejestrowanego w hurtowni danych faktu. Fakt zawiera jedną lub więcej zmierzonych lub zaobserwowanych zmiennych, które identyfikuje za pomocą wskaźników do odpowiednich tabel wymiarów. Tabela faktów zawiera właściwe dane i informacje umożliwiające identyfikowanie każdej z krotek we właściwych wymiarach.

Tabelę faktów można też potraktować jak zagregowaną perspektywę z danymi transakcyjnymi, gdzie każda tabela wymiaru reprezentuje „dane nadrzędne”, do których należą określone transakcje. W systemach baz danych obsługujących model wielowymiarowy jest on implementowany za pomocą specjalnego oprogramowania o nazwie *wielowymiarowe bazy danych*, którego tu nie omawiamy. Model wielowymiarowy w naszym ujęciu jest oparty na umieszczeniu hurtowni jako relacyjnej bazy w relacyjnym SZBD.

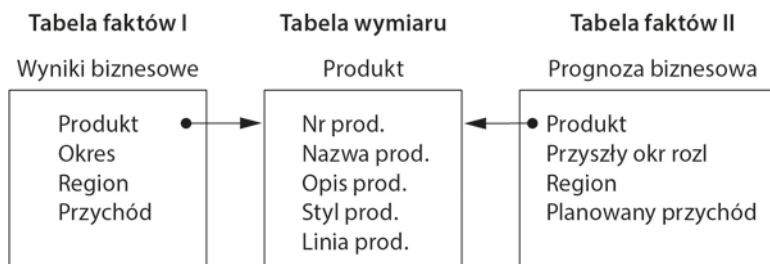
Na rysunku 29.7 przedstawiono przykład tabeli faktów, która może być przeglądana z perspektywy wielowymiarowych tabel. Dwa popularne wielowymiarowe schematy to schemat gwiazdy i schemat płątka śniegu. **Schemat gwiazdy** składa się z tabeli faktów i osobnych tabel dla każdego wymiaru (patrz rysunek 29.7). **Schemat płątka śniegu** jest pewną odmianą schematu gwiazdy — różnica polega na tym, że tabele wymiarów ze schematu gwiazdy są (wskutek przeprowadzenia normalizacji) zorganizowane w hierarchię (patrz rysunek 29.8). **Konstelacja faktów** jest zbiorem tabel faktów, które współdzielą niektóre tabele wymiarów. Na rysunku 29.9 przedstawiono konstelację faktów z parą tabel faktów reprezentujących wyniki biznesowe i prognozy biznesowe. Obie tabele faktów współdzielią tę samą tabelę wymiaru produktów. Konstelacja faktów ogranicza zbiór zapytań, które można wykonać na hurtowni danych.



RYСУNEK 29.7. Schemat gwiazdy z tabelami faktów i wymiarów



RYСУNEK 29.8. Schemat płatk śniegu



RYSUNEK 29.9. Przykładowa konstelacja faktów

Hurtownie danych wykorzystują także techniki indeksowania zasobów, które zapewniają większą wydajność dostępu do informacji (techniki indeksowania omówiliśmy w rozdziale 17.). Często spotykana w tego typu rozwiązaniach technika, nazywana **indeksowaniem bitmapowym** (ang. *bitmap indexing*), konstruuje wektor bitowy dla każdej indeksowanej wartości należącej do danej dziedziny (kolumny). Technika ta doskonale się sprawdza w przypadku dziedzin o niskiej liczności. W każdym wektorze ustawiany jest jeden bit na  $j$ -tej pozycji, jeśli  $j$ -ty wiersz zawiera indeksowaną wartość. Przykładowo, wyobraźmy sobie spis samochodów zawierający 100 tysięcy pozycji, dla których stworzono indeks bitmapowy na kolumnie przechowującej rozmiar samochodu. Jeśli przyjmujemy, że istnieją cztery możliwe rozmiary samochodów — ekonomiczny, kompaktowy, miejski i rodzinny — taki indeks będzie się składał z czterech wektorów bitowych, z których każdy będzie zawierał 100 tysięcy bitów (12,5 KB); rozmiar całego indeksu bitmapowego będzie więc równy ok. 50 KB. Indeksowanie bitmapowe może być źródłem znacznych korzyści zarówno pod względem szybkości wykonywania operacji wejścia-wyjścia, jak i zajmowanej przestrzeni — jedynym warunkiem jest mała liczność dziedziny wartości. Dzięki tak skonstruowanym wektorom bitowym indeks bitmapowy może w zasadniczy sposób poprawić efektywność takich operacji jak porównywanie danych, agregacja czy złączanie. Przykładowe zapytanie z użyciem schematu gwiazdy pokazano w podrzdziale 19.8; przedstawiono tam także przekształcanie schematu gwiazdy na potrzeby skutecznego wykonywania operacji z użyciem indeksów bitmapowych.

W schemacie gwiazdy dane wymiarowe mogą być indeksowane do krotek zawartych w tabeli faktów z wykorzystaniem **indeksowania złączeń** (ang. *join indexing*). Indeksy złączeń są tradycyjnymi indeksami stosowanymi do utrzymywania związków pomiędzy wartościami kluczy głównych a wartościami kluczy obcych. Takie indeksy mogą być wykorzystywane w schemacie gwiazdy do łączenia wartości w tabeli wymiaru z odpowiadającymi im wierszami w tabeli faktów. Przykładowo, wyobraźmy sobie tabelę faktów reprezentujących sprzedaż produktów, dla której istnieją tabele wymiarów: miasto sprzedaży oraz okres rozliczeniowy. Gdybyśmy stworzyli indeks złączenia dla miasta, to dla każdego miasta reprezentowanego w hurtowni danych taki indeks utrzymywałby identyfikatory krotek zawierających dane miasto. Indeksy złączeń mogą obejmować więcej niż jeden wymiar.

Hurtownie danych mogą ułatwiać dostęp do podsumowań danych — możliwości w tym zakresie wynikają z trwałości informacji składowanych w hurtowniach danych oraz stopnia przewidywalności wykonywanych na tych danych analiz. Takie przedstawianie informacji jest realizowane w oparciu o dwie różne strategie: (1) budowa mniejszych tabel zawierających podsumowania danych takich jak kwartalne zestawienia wyników sprzedaży lub przychody według linii produktów oraz (2) stosowanie różnych poziomów kodowania

w istniejących tabelach (np. tygodniowe, kwartalne lub roczne kodowanie wyników sprzedaży). Dla porównania, dodatkowe koszty związane z tworzeniem i utrzymywaniem tego typu agregacji stanowiłyby zapewne barierę nie do pokonania w często modyfikowanych, transakcyjnych bazach danych.

Celem popularnego w korporacjach podejścia **MDM** (ang. *master data management*) jest definiowanie standardów, procesów, strategii i własności danych powiązanych z kluczowymi encjami w organizacji. Tabele wymiarów (które w hurtowni danych dotyczą fizycznych obiektów takich jak klienci, regiony lub kategorie produktów) reprezentują dane nadrzędne. Ponieważ wymiary są wspólne dla wielu faktów lub dla używanych do tworzenia raportów składnic danych, projektanci hurtowni danych zwykle muszą poświęcać dużo czasu na porządkowanie i ujednolicanie wymiarów (np. na eliminowanie definicyjnych i pojęciowych różnic między wieloma źródłowymi systemami, z których pochodzą dane wymiarów). Dlatego struktury tabel zawierających wymiary są dobrymi kandydatami do przechowywania specjalnych kopii danych nadrzędnych, które można wykorzystać w innych środowiskach.

## 29.4. Budowanie hurtowni danych

Podczas konstruowania hurtowni danych projektanci powinni mieć jak najszerzy ogłąd przewidywanych zastosowań projektowanych rozwiązań. W fazie projektowania nie można oczywiście przewidzieć wszystkich zapytań czy analiz. Efektem całego procesu projektowania bez wątpienia powinna być właściwa obsługa **zapytań tworzonych *ad hoc***, a więc możliwość dostępu do danych składających się z dowolnej sensownej kombinacji wartości atrybutów z tabel wymiarów lub faktów. Przykładowo, firma marketingowa mogłaby wymagać zupełnie innej organizacji wykorzystywanej przez siebie hurtowni danych niż charytatywna fundacja non profit, której działania skupiają się na poszukiwaniu sponsorów. Przewidywane techniki stosowania hurtowni danych powinny determinować wybór właściwego schematu.

Gromadzenie informacji dla hurtowni danych wymaga wykonania następujących kroków:

- (1) Dane muszą być pozyskiwane z wielu heterogenicznych źródeł, takich jak bazy danych lub inne materiały zawierające np. dane z rynków finansowych lub informacje na temat środowiska naturalnego.
- (2) Dane muszą być formatowane w sposób umożliwiający spójne składowanie w hurtowni danych. Nazwy, znaczenia i dziedziny danych pochodzących z niezwiązanych ze sobą źródeł muszą być do siebie dostosowane. Przykładowo, podwykonawcy realizujący zamówienia ogromnej korporacji mogą wykorzystywać inne kalendarze fiskalne, gdzie kwartały kończą się w różnych dniach, co w oczywisty sposób utrudnia agregowanie danych finansowych dla poszczególnych kwartałów. Różne typy kart kredytowych mogą zgłaszać transakcje do systemu w inny sposób, co utrudnia naliczanie wszystkich zakupów dokonywanych kartą. Tego typu niezgodności formatów muszą zostać wyeliminowane.
- (3) Dane muszą być oczyszczone, aby zapewnić ich poprawność. Oczyszczanie danych jest zawiłym i skomplikowanym procesem, który w wielu przypadkach stanowi najbardziej pracochłonny etap całego procesu konstruowania hurtowni danych.

W przypadku danych wejściowych mechanizm oczyszczania należy stosować przed zapisaniem nowych informacji w hurtowni danych. Ponieważ dane wejściowe muszą być analizowane pod kątem spójności z pozostałymi danymi i odpowiednio formatowane, projektanci hurtowni danych powinni wykorzystać możliwość weryfikowania poprawności i jakości danych. Rozpoznawanie błędnych i niekompletnych danych jest trudne do zautomatyzowania, a oczyszczanie danych, które wymaga stosowania mechanizmów automatycznej korekcji błędów, bywa jeszcze trudniejsze. Niektóre aspekty tego zagadnienia (np. sprawdzanie dziedzin wartości) są co prawda stosunkowo łatwe do zakodowania w mechanizmach oczyszczających, jednak automatyczne wykrywanie pozostałych problemów związanych z danymi może być znacznie bardziej wymagające (przykładowo, nietrudno sobie wyobrazić, że jedno z wymagań będzie określało kombinację `Miasto = 'Warszawa'` i `Województwo = 'wielkopolskie'` jako błędną). Po zajęciu się wszystkimi tego typu problemami projektant hurtowni danych musi skoordynować wczytywanie do hurtowni podobnych danych pochodzących z różnych źródeł. Kiedy zatrudnieni w organizacji menadżerowie danych stwierdzą, że ich dane są poddawane procesowi oczyszczania przed zapisaniem w hurtowni danych, z pewnością będą chcieli wprowadzić w tych danych odpowiednie aktualizacje, które w przyszłości wyeliminują konieczność ponownego oczyszczania. Działania polegające na zwracaniu oczyszczonych danych do źródła są nazywane **wstecznym oczyszczaniem** (patrz rysunek 29.1).

- (4) Dane muszą być dostosowane do modelu danych zastosowanego w tworzonej hurtowni. Dane pochodzące z różnych źródeł muszą być odpowiednio reprezentowane w modelu danych hurtowni. W niektórych sytuacjach konieczne będzie przekonwertowanie tych danych ze stosowanego w źródłowej bazie danych modelu relacyjnego, obiektowego lub sieciowego i (lub) hierarchicznego do modelu wielowymiarowego.
- (5) Dane muszą być wczytane do hurtowni danych. Sama ilość danych gromadzonych w hurtowni powoduje, że proces ich wczytywania to trudne zadanie. Często niezbędne jest stosowanie narzędzi monitorujących proces wczytywania danych oraz metod przywracania prawidłowego stanu w razie wystąpienia niekompletnych lub niepoprawnych operacji wczytywania. W przypadku ogromnych ilości danych składowanych w hurtowni aktualizowanie przyrostowe jest zwykle jedynym wykonalnym rozwiązaniem. Stosowana polityka odświeżania zawartości hurtowni danych najczęściej stanowi kompromis, który jest wynikiem zbioru uzyskanych odpowiedzi na wymienione poniżej pytania:
  - Na ile aktualne mają być informacje składowane w hurtowni danych?
  - Czy hurtownia danych może być odłączana od swoich źródeł danych?  
Jeśli tak, to na jak długo?
  - Jakie zależności istnieją pomiędzy danymi?
  - Jakie są możliwości w zakresie składowania danych?
  - Jakie zdefiniowano wymagania odnośnie rozproszenia hurtowni danych (np. dotyczące replikacji i partycjonowania)?
  - Jaki jest planowany czas wczytywania danych (włącznie z ich oczyszczaniem, formatowaniem, kopiowaniem, przesyłaniem oraz odbudową indeksów)?



Dane w hurtowni mogą pochodzić z wielu źródeł, lokalizacji geograficznych i (lub) stref czasowych. Dlatego wczytywanie danych trzeba starannie zaplanować i przeprowadzić. Kolejność wczytywania danych do hurtowni jest bardzo istotna. Wczytanie danych w niewłaściwym porządku może prowadzić do naruszenia ograniczeń integralności lub reguł semantycznych. Oba te problemy mogą skutkować niepowodzeniem wczytywania danych. Przykładowo, dane nadrzędne (nowe lub zmodyfikowane), takie jak dane klientów lub produktów, trzeba wczytać przed dotyczącymi ich transakcjami, a dane z faktury należy wczytać przed powiązanymi z nimi danymi rozrachunkowymi.

Wspomnieliśmy już, że bazy danych muszą jednocześnie zapewniać wydajne przetwarzanie transakcji i obsługiwać wymagania użytkowników odnośnie wykonywanych zapytań (w tym zapytań tworzonych *ad hoc*); nieco inaczej jest w przypadku hurtowni danych, które są zwykle optymalizowane z myślą o zapewnieniu szybkiego dostępu do potrzebnych danych osobom podejmującym decyzje. Stosowane w hurtowniach danych metody składowania informacji odzwierciedlają właśnie tę właściwość i obejmują następujące procesy:

- Składowanie danych zgodnie z modelem danych zastosowanym w hurtowni.
- Tworzenie i utrzymywanie wymaganych struktur danych.
- Tworzenie i utrzymywanie właściwych ścieżek dostępu.
- Zapewnienie mechanizmów obsługi danych zmieniających się w czasie, np. w przypadku dodawania nowych danych.
- Obsługa aktualizacji zawartości hurtowni danych.
- Odświeżanie danych.
- Usuwanie przestarzałych danych.

Nawet jeśli na początku budowaniu hurtowni zostanie poświęcona odpowiednia ilość czasu, ogromna ilość danych składowanych w hurtowni sprawia, że jej ponowne, kompletne wypełnienie danymi jest później po prostu niemożliwe. Istnieją rozwiązania alternatywne polegające na selektywnym (częściowym) odświeżaniu danych i oddzielaniu różnych wersji hurtowni (co wymaga z kolei podwojenia przestrzeni składowania danych z hurtowni!). W sytuacji, gdy hurtownia danych wykorzystuje mechanizm przyrostowego odświeżania danych, może istnieć potrzeba okresowego usuwania przestarzałych informacji; przykładowo, z hurtowni danych przechowującej informacje o ostatnich dwunastu okresach rozliczeniowych można raz w roku (lub nawet co kwartał) usuwać dane dotyczące najbardziej odległych wyników.

Hurtownie danych muszą być projektowane także ze szczególnym uwzględnieniem środowiska, w którym będą funkcjonowały. Do najważniejszych elementów składających się na takie środowisko należą:

- przewidywane zastosowania,
- dopasowanie modelu danych,
- właściwości dostępnych zasobów,
- projekt komponentu metadanych,
- projekt podziału na moduły,
- projektowane możliwości zarządzania i modyfikowania,
- kwestie związane z architekturą rozproszoną i równoległą.

Dalej kolejno omówimy każdy z tych elementów. Projekt hurtowni danych jest początkowo uzależniony głównie od przewidywań w zakresie zastosowań rozwiązania, a więc tego, kto i jak będzie korzystał z hurtowni. Na początku ważny jest wybór modelu danych zgodnego z tymi zastosowaniami. Na tym etapie bierze się pod uwagę zarówno przewidywane zastosowania, jak i charakterystyki źródeł danych. Odpowiedni projekt podziału hurtowni danych na moduły jest niezbędny z przyczyn praktycznych, ponieważ tylko rozbiecie całego systemu na mniejsze części umożliwi w przyszłości jego modyfikowanie w odpowiedzi na zmiany w organizacji i jej środowisku informacyjnym. Dobry projekt hurtowni danych musi przewidywać jak największe możliwości w zakresie jej utrzymania, które pozwolą osobom zarządzającym hurtownią efektywnie planować i wprowadzać odpowiednie zmiany, stale zapewniając optymalną obsługę użytkowników.

Terminu *metadane* użyliśmy kilkakrotnie od rozdziału 1. — zdefiniowaliśmy je jako opis bazy danych, włącznie z definicją schematu bazy danych. W hurtowni danych kluczowym składnikiem jest tzw. **repozytorium metadanych**. Repozytorium metadanych zawiera zarówno techniczne, jak i biznesowe metadane. **Techniczne metadane** obejmują szczegóły procesu gromadzenia i przetwarzania informacji, struktur ich składowania, opisy danych, operacje i metody utrzymania hurtowni danych, a także funkcjonalność obsługi dostępu do zawartości tej hurtowni. **Biznesowe metadane** zawierają istotne szczegóły reguł biznesowych i organizacyjnych, które są obsługiwane przez hurtownię danych.

Architektura rozproszonego środowiska informatycznego organizacji jest głównym elementem decydującym o właściwościach całego projektu hurtowni danych. Istnieją dwa podstawowe typy architektur rozproszonych: hurtownie rozproszone i hurtownie federacyjne. **Rozproszone hurtownie danych** mają wszystkie cechy rozproszonych baz danych związane z replikacją, partycjonowaniem, komunikacją oraz spójnością danych. Architektura rozproszona może być szczególnie korzystna pod względem wydajności hurtowni danych, w tym lepszego równoważenia obciążeń, skalowalności oraz większej dostępności. Taka architektura przewiduje, że w każdym węźle rozproszonego systemu jest utrzymywana pojedyncza replika (kopia) repozytorium metadanych. Idea tworzenia **federacyjnych hurtowni danych** przypomina rozwiązania oparte na federacyjnych bazach danych — oznacza to, że federacyjna hurtownia danych ma postać zdecentralizowanej konfederacji autonomicznych hurtowni danych, z których każda zawiera własne repozytorium metadanych. Mając świadomość ogromnych wyzwań, jakie stoją przed hurtowniami danych, nietrudno zauważyć, że takie federacje powinny się składać z mniejszych komponentów, np. składnic danych.

Firmy stają się niezadowolone z tradycyjnych technik i technologii z obszaru hurtowni danych. Nowe wymagania z dziedziny analityki skutkują powstawaniem nowych rozwiązań analitycznych, takich jak Netezza firmy IBM, Greenplum firmy EMC, Hana firmy SAP i ParAccel firmy Tableau Software. Analizy w zakresie big data sprawiły, że w nowej generacji hurtowni używana jest platforma Hadoop i inne wyspecjalizowane bazy danych (np. grafowe i z parami klucz-wartość). Omówienie technologii z obszaru big data opartych na Hadoopie znajdziesz w rozdziale 25. Platformy wirtualizacji danych, np. firmy Cisco<sup>4</sup>, w przyszłości umożliwią budowanie logicznych hurtowni danych.

---

<sup>4</sup> Patrz opis platformy wirtualizacji danych firmy Cisco: <http://www.compositesw.com/products-services/data-virtualization-platform/>.

## 29.5. Typowe funkcje hurtowni danych

Hurtownie danych istnieją po to, aby ułatwiać skomplikowane, związane z dużą ilością danych oraz często wykonywane zapytania *ad hoc*. Hurtownie danych muszą więc gwarantować lepszą i bardziej wydajną obsługę zapytań, niż oczekuje się od transakcyjnych baz danych. Moduł dostępu do danych, który jest częścią hurtowni danych, powinien oferować rozszerzoną funkcjonalność arkusza kalkulacyjnego, efektywny mechanizm przetwarzania zapytań, obsługę wbudowanych zapytań i zapytań konstruowanych *ad hoc*, możliwości eksploracji danych oraz obsługę perspektyw zmaterializowanych. W szczególności, rozszerzona funkcjonalność arkusza kalkulacyjnego powinna obejmować mechanizmy współpracy z tradycyjnymi aplikacjami tego typu (np. aplikacją Excel firmy Microsoft), a także z programami opartymi na rozwiązaniach OLAP. Takie rozszerzone arkusze kalkulacyjne często oferują następujące elementy funkcjonalności:

- **Zwijanie:** Dane są podsumowywane w postaci postępującej generalizacji (przykładowo, dane tygodniowe mogą być połączone w dane kwartalne, roczne itd.).
- **Rozwijanie:** Ta technika umożliwia odśłanianie bardziej szczegółowych poziomów danych (jest więc odwrotnością techniki zwijania).
- **Przestawianie:** Stosowanie tabel przestawnych (*rotowanie* danych).
- **Przecinanie i rzutowanie:** Wykonywanie operacji projekcji na poszczególnych wymiarach.
- **Sortowanie:** Dane są sortowane według wartości porządkowych.
- **Selekcja:** Dane są filtrowane na podstawie wartości lub zakresu.
- **Atrybuty pochodne (obliczane):** Atrybuty są wyznaczane za pomocą operacji wykonywanych zarówno na wartościach składowanych, jak i na innych wartościach pochodnych.

Ponieważ hurtownie danych są wolne od ograniczeń typowych dla środowisk transakcyjnych, gwarantują znacznie większą efektywność mechanizmów przetwarzania zapytań. Do narzędzi i technik stosowanych w hurtowniach danych należy transformacja zapytań, logiczne mnożenie i sumowanie indeksów, specjalne funkcje **ROLAP** (relacyjny OLAP; od ang. *Relational OLAP*) oraz **MOLAP** (wielowymiarowy OLAP; od ang. *Multidimensional OLAP*), rozszerzenia języka SQL, zaawansowane metody złączania oraz inteligentne skanowanie (podobnie jak w przypadku wielokrotnie wtrącanych zapytań).

Dostępne są też rozwiązania **HOLAP** (hybrydowy OLAP), łączące funkcje ROLAP i MOLAP. Na potrzeby tworzenia podsumowań rozwiązania HOLAP wykorzystują kostki danych (za pomocą funkcji MOLAP), co zwiększa wydajność. Gdy potrzebne są szczegółowe informacje, system HOLAP może rozwijać kostkę do postaci podstawowych danych relacyjnych (z wykorzystaniem komponentu ROLAP).

Poprawa wydajności hurtowni danych wynika także z przetwarzania równoległego. Do najpopularniejszych równoległych architektur serwerów należy symetryczna architektura wieloprocessorowa (ang. *Symmetric MultiProcessor* — *SMP*), architektura klastrowa, architektura masowego przetwarzania równoległego (ang. *Massively Parallel Processing* — *MPP*) oraz rozmaite kombinacje wymienionych architektur.

Pracownicy wykorzystujący wiedzę oraz osoby odpowiedzialne za podejmowanie decyzji wykorzystują do eksploracji danych zarówno zapytania parametryczne, jak i zapytania tworzone *ad hoc*. Oznacza to, że stosowany w hurtowni danych moduł dostępu do danych musi zapewniać dostęp do zapytań strukturalnych (zarówno tych parametrycznych, jak i tworzonych *ad hoc*). Połączenie tych elementów tworzy tzw. zarządzane środowisko zapytań. Same mechanizmy eksploracji danych opierają się na technikach, które wywodzą się z analizy statystycznej oraz sztucznej inteligencji. Analiza statystyczna może być oczywiście stosowana z wykorzystaniem zaawansowanych arkuszy kalkulacyjnych, wyszukanego oprogramowania statystycznego lub programów napisanych specjalnie do realizacji określonych zadań. Bardzo popularne są takie techniki jak wyznaczanie opóźnień (ang. *lagging*), średnie kroczące czy analiza regresji. Do klasyfikowania danych i odkrywania wiedzy (w oparciu o zawartość hurtowni danych), która nie zawsze jest zgodna z oczekiwaniami analityków lub nie daje się łatwo wyrażać w postaci zapytań, stosuje się techniki sztucznej inteligencji (np. algorytmy genetyczne, sieci neuronowe). Zagadnienia związane z eksploracją danych szczegółowo omówiliśmy w rozdziale 28.

## 29.6. Hurtownie danych kontra perspektywy

Niektórzy sądzą, że hurtownie danych są tylko rozszerzeniem perspektyw z baz danych. Wspominaliśmy w tej książce o mechanizmie materializowanych perspektyw, które są jednym ze sposobów realizacji wymagań w zakresie lepszego dostępu do danych (omówienie perspektyw zawarto w punkcie 7.3). Zmaterializowane perspektywy są wykorzystywane właśnie ze względu na poprawę wydajności. W punkcie 19.2.4 opisano, jak perspektywy zmaterializowane są utrzymywane i stosowane na potrzeby optymalizowania zapytań. Perspektywy są jednak jedynie podzbiorem funkcji i możliwości hurtowni danych. Podobieństwo między perspektywami i hurtowniami danych polega przede wszystkim na tym, że oba rozwiązania prezentują zawartość baz danych z przeznaczeniem tylko do odczytu oraz mają charakter tematyczny. Hurtownie danych różnią się jednak od perspektyw w następujących aspektach:

- Hurtownie danych mają postać trwałych struktur; nie są jedynie materializowane na żądanie.
- Hurtownie danych rzadko mają charakter struktur relacyjnych, znacznie częściej są zgodne z wielowymiarowym modelem danych.
- Hurtownie danych mogą być indeksowane w celu poprawy wydajności. Indeksów nie można stosować dla perspektyw w oderwaniu od reprezentowanych baz danych.
- Hurtownie danych we właściwy dla siebie sposób oferują specyficzną funkcjonalność; perspektywy tego nie umożliwiają.
- Hurtownie danych przechowują ogromne ilości zintegrowanych (i często czasowych) danych — zwykle tych danych jest więcej niż w pojedynczej bazie danych; natomiast perspektywy stanowią jedynie formę reprezentacji informacji zawartych w bazach danych.
- W hurtowniach dane są zapisywane z wielu źródeł za pomocą złożonego procesu ETL, obejmującego oczyszczanie, usuwanie i podsumowywanie informacji. Perspektywy są tworzone na podstawie bazy za pomocą zdefiniowanego zapytania.

## 29.7. Trudności z implementowaniem hurtowni danych

Konstruowanie, administracja i kontrola jakości hurtowni danych wiąże się z kilkoma istotnymi problemami. Ogromne znaczenie ma właściwe zarządzanie całym projektem (na który składa się projektowanie, konstruowanie i implementacja hurtowni danych) — nigdy nie należy lekceważyć działań zaliczanych do zarządzania projektem. Budowa korporacyjnej hurtowni danych w ogromnej organizacji jest wielkim przedsięwzięciem, w którym przejście od początkowej koncepcji do ostatecznej implementacji może zająć lata. Trudności z tym związane i długi czas wczytywania danych do hurtowni stał się źródłem popularności (zarówno w sensie rozwoju, jak i pomyślnie zakończonych wdrożeń) alternatywy dla korporacyjnych hurtowni danych — tzw. składnic danych, które mogą stanowić atrakcyjny wybór dla tych organizacji, które pilnie potrzebują dostępu do rozwiązań OLAP, DSS i (lub) eksploracji danych.

Administracja hurtownią danych wymaga sporego zaangażowania, które zwykle jest uzależnione od rozmiaru i poziomu złożoności hurtowni. Organizacja podejmująca próbę administrowania hurtownią danych musi dobrze zrozumieć i realistycznie ocenić skomplikowaną materię takiej administracji. Chociaż hurtownie danych są projektowane z myślą o dostępie tylko do odczytu, hurtownia danych nie będzie statyczną strukturą, dopóki nie zostaną powstrzymane zmiany wprowadzane w jej źródłach informacji. W wielu przypadkach należy oczekiwać, że źródłowe bazy danych będą stale zmieniane; schemat hurtowni danych oraz moduł pozyskiwania informacji muszą więc być przygotowane na częste aktualizacje uwzględniające zmiany w źródłach danych.

Ważnym zagadnieniem w rozwiązaniach opartych na hurtowniach danych jest problem kontroli jakości danych. Zarówno jakość, jak i spójność danych (przede wszystkim danych wymiarów, które są związane z zarządzaniem danymi nadrzędnymi) ma zasadnicze znaczenie dla użytkowników hurtowni. Dane przekazywane do hurtowni przechodzą co prawda proces oczyszczania, jednak samo oczyszczanie nie zdejmuje z administratora i projektanta bazy danych odpowiedzialności za właściwą jakość i spójność składowanych informacji. Największe problemy sprawia mieszanie danych pochodzących z heterogenicznych i całkowicie odmiennych źródeł, ponieważ administrator ma wówczas do czynienia z różnymi konwencjami nazewnictwa, różnymi definicjami dziedzin wartości, różnymi numerami identyfikacyjnymi itp. Za każdym razem gdy źródłowa baza danych ulega zmianie, administrator hurtowni danych musi rozważyć ewentualny wpływ tych zmian na pozostałe elementy zarządzanej przez siebie hurtowni danych.

Prognozy dotyczące przyszłych zastosowań hurtowni danych powinny być opracowywane przed przystąpieniem do jej konstruowania, co wcale nie oznacza, że nie powinny podlegać stałym korektom odpowiadającym zmieniającym się wymaganiom. Wraz ze zmianami wzorców przyszłego wykorzystania tworzonej hurtowni danych na bardziej czytelne i bliższe ostatecznego kształtu można dostosowywać ścieżki składowania i dostępu tak, aby pozostawały zoptymalizowane do obsługi rzeczywistych zastosowań tej hurtowni w danej organizacji. Tego typu czynności powinny być wielokrotnie powtarzane w całym cyklu życia hurtowni danych, ponieważ tylko w ten sposób można zapewnić zgodność tej hurtowni z oczekiwaniami użytkowników. Projekt hurtowni danych powinien także uwzględniać możliwość dodawania i odłączania źródeł danych bez konieczności wprowadzania zasadniczych zmian w strukturze hurtowni. Źródła danych i same dane będą się zmieniać, a w hurtowni trzeba uwzględnić takie modyfikacje. Dostosowywanie do-

stępnym źródłem danych do modelu danych zastosowanego w hurtowni będzie wymagało stałego zaangażowania i jest w równym stopniu sztuką, jak i nauką. Ponieważ technologie informatyczne stale podlegają błyskawicznym zmianom, zarówno wymagania, jak i możliwości hurtowni danych będą z czasem wielokrotnie ewoluowały. Co więcej, sama technologia hurtowni danych będzie się zmieniała — można przyjąć, że struktura i funkcjonalność poszczególnych modułów będzie nieustannie udoskonalana. Łatwa do przewidzenia zmienność technologii stanowi istotną przesłankę do budowania systemów w pełni modułowych.

Administracja hurtownią danych wymaga znacznie szerszych umiejętności niż administracja tradycyjną bazą danych. Różne jednostki dużej organizacji często postrzegają dane w odmienny sposób. Zatrudnienie jednej osoby nie wystarczy — niezbędne jest zaangażowanie zespołu ekspertów z doskonałym przygotowaniem technicznym i niemałym doświadczeniem w rozmaitych obszarach. Taki zespół musi posiadać szczegółową wiedzę biznesową oraz na temat reguł, regulacji, ograniczeń i polityk stosowanych w firmie. Podobnie jak w przypadku administrowania bazą danych, także administracja hurtownią danych nie ogranicza się wyłącznie do działań technicznych; znaczną część zadań realizowanych przez administratora stanowi efektywna współpraca ze wszystkimi pracownikami organizacji, którzy są zainteresowani korzystaniem z hurtowni danych. Zasadniczym problemem, przed jakim stoją administratorzy baz danych przydzieleni do administracji hurtowniami danych, jest znacznie szerszy zakres obowiązków i — tym samym — zwiększona odpowiedzialność.

Kluczowe znaczenie ma zarówno jakość projektu funkcji zarządzania hurtownią danych, jak i dobór zespołu zarządzającego, który będzie te funkcje realizował. Zarządzanie hurtownią danych w ogromnej organizacji zawsze stanowi poważne wyzwanie. Istnieje obecnie wiele komercyjnych narzędzi stworzonych specjalnie z myślą o obsłudze funkcji zarządzania. Efektywne zarządzanie hurtownią danych z pewnością zależy od zaangażowania zespołu, który musi dysponować szerokimi umiejętnościami technicznymi, którego działania muszą być dobrze koordynowane i który musi być odpowiednio kierowany. Równie ważne jak opracowanie rozwiązań przygotowanych na nieustającą ewolucję technologii hurtowni danych jest uwzględnienie konieczności rozszerzania (w razie potrzeby) umiejętności zespołu zarządzającego już istniejącą hurtownią danych.

## 29.8. Podsumowanie

W tym rozdziale omówiliśmy zagadnienia związane z technologią nazywaną hurtowniami danych. Pracę z hurtowniami danych można postrzegać jako proces wymagający rozmaitych czynności przygotowawczych. Inaczej jest w przypadku eksploracji danych (patrz rozdział 28.), którą można traktować jak mechanizm uzyskiwania wiedzy na podstawie wartości istniejącej hurtowni danych. Najpierw, w podrozdziale 29.1, wprowadziliśmy kluczowe pojęcia związane z hurtowniami danych, zdefiniowaliśmy technologie takie jak *OLAP* i *DSS* oraz porównaliśmy je z modelem *OLTP*. Zaprezentowaliśmy też ogólną architekturę systemów hurtowni danych. W podrozdziale 29.2 omówiliśmy podstawowe cechy i różne rodzaje hurtowni danych. W podrozdziale 29.3 opisaliśmy modelowanie danych w hurtowniach za pomocą wielowymiarowego modelu danych. Przedstawiliśmy tam różne rodzaje tabel i schematów. W podrozdziale 29.4 szczegółowo opisaliśmy procesy i aspekty



projektowe związane z budowaniem hurtowni danych. W podrozdziale 29.5 zaprezentowaliśmy typowe specjalne funkcje hurtowni danych. W podrozdziale 29.6 porównaliśmy perspektywy z modelu relacyjnego z wielowymiarowym ujęciem danych w hurtowniach. Wreszcie w podrozdziale 29.7 opisaliśmy trudności związane z implementowaniem hurtowni danych i administrowaniem nimi.

## Pytania powtórkowe

- 29.1. Czym jest hurtownia danych? Czym hurtownie danych różnią się od baz danych?
- 29.2. Zdefiniuj następujące pojęcia: *OLAP* (ang. *OnLine Analytical Processing*), *ROLAP* (ang. *Relational OLAP*), *MOLAP* (ang. *Multidimensional OLAP*) oraz *DSS* (ang. *Decision Support System*).
- 29.3. Opisz najważniejsze właściwości hurtowni danych. Podziel wymienione charakterystyki na te, które zalicza się do funkcjonalności hurtowni danych, oraz te uznawane za pochodne korzyści stosowania tej funkcjonalności przez użytkowników.
- 29.4. Czym jest wielowymiarowy model danych? Jak wykorzystuje się ten model w technologiach hurtowni danych?
- 29.5. Zdefiniuj następujące pojęcia: *schemat gwiazdy*, *schemat płatka śniegu*, *konstelacja faktów* oraz *składnice danych*.
- 29.6. Jakiego rodzaju indeksy buduje się dla hurtowni danych? Zilustruj zastosowanie każdego z wymienionych typów w oparciu o przykład.
- 29.7. Opisz kroki składające się na proces budowy hurtowni danych.
- 29.8. Które elementy odgrywają najważniejszą rolę podczas projektowania hurtowni danych?
- 29.9. Opisz funkcje, które mogą być realizowane przez użytkownika na hurtowni danych, i zilustruj za pomocą przykładowej wielowymiarowej hurtowni danych wyniki wykonania tych funkcji.
- 29.10. Jakie są cechy wspólne relacyjnych *perspektyw* oraz hurtowni danych i składnic danych? Co różni relacyjne perspektywy od wymienionych struktur?
- 29.11. Wymień trudności związane z implementowaniem hurtowni danych.
- 29.12. Wymień nierozwiązane zagadnienia i problemy będące przedmiotem badań w zakresie hurtowni danych.
- 29.13. Czym jest zarządzanie danymi nadrzędnymi? W jaki sposób zadanie to jest powiązane z hurtowniami danych?
- 29.14. Czym są logiczne hurtownie danych? Poszukaj w internecie informacji o platformie wirtualizacji danych firmy Cisco. Wyjaśnij, w jaki sposób może ona pomóc w budowaniu logicznych hurtowni danych.

## Wybrane publikacje

Wprowadzenie pojęcia hurtowni danych przypisuje się Inmonowi (1992, 2005). Codd i Salley (1993) spopularyzowali termin *przetwarzania analitycznego dostępnego bezpośrednio* (OLAP) i zdefiniowali zbiór własności, które powinny być spełnione przez hurtownie



danych obsługujące rozwiązania OLAP. Kimball (1996) jest znany z wkładu w rozwój dziedziny hurtowni danych. Mattison (1996) napisał jedną z wielu książek poświęconych hurtowniom danych, która zawiera nie tylko analizę technik stosowanych w hurtowniach danych, ale także wskazówki dotyczące strategii ich wdrażania w przedsiębiorstwach. Ponniah (2010) opracował bardzo dobry praktyczny przegląd procesów budowania hurtowni danych: od zbierania wymagań po konserwację wdrożonych rozwiązań. Praca Jukica i in. (2013) to bogate źródło informacji o modelowaniu hurtowni danych. Publikacja Bischoffa i Alexandera (1997) stanowi kompilację rad udzielanych przez ekspertów z dziedziny hurtowni danych. Chaudhuri i Dayal (1997) przedstawili doskonały podręcznik dotyczący tych zagadnień, natomiast Widom (1995) skupił się na kilku aktualnych problemach badawczych w tym zakresie.

# XII

---

**Dodatkowe zagadnienia z obszaru  
baz danych: bezpieczeństwo**



## Bezpieczeństwo w bazach danych

Niniejszy rozdział omawia techniki używane do ochrony baz danych przed różnymi zagrożeniami. Przedstawione są też schematy zapewniania uprawnień dostępu uwierzytelnionym użytkownikom. Zaprezentujemy wybrane zagrożenia dotyczące baz danych, np. wstrzykiwanie kodu w języku SQL. W końcowej części rozdziału pokażemy, jak różne rodzaje zabezpieczeń działają w popularnym relacyjnym SZBD Oracle. Podrozdział 30.1 przedstawia wstępne informacje o kwestiach bezpieczeństwa, zagrożeniach wobec baz danych, a także o środkach zapobiegawczych; zagadnienia te omówimy dalej w rozdziale. Podrozdział 30.2 omawia mechanizmy używane do nadawania i odbierania uprawnień w relacyjnych systemach baz danych i w języku SQL. Mechanizmy te są nazywane **dyspozycyjnym systemem kontroli dostępu** (ang. *discretionary access control*). Podrozdział 30.3 przedstawia mechanizm pozwalający na wymuszenie wielu poziomów bezpieczeństwa, który częściej występuje we współczesnych systemach bazodanowych i jest znany jako **obowiązkowy system kontroli dostępu** (ang. *mandatory access control*). Wprowadza także pojęcie ról w **zarządzaniu kontrolą dostępu**, a także zabezpieczeń opartych na etykietach i wierszach. W podrozdziale 30.3 pokrótce omówiono też kontrolę dostępu do danych w formacie XML. Podrozdział 30.4 zawiera omówienie ważnego zagrożenia dla baz danych, czyli wstrzykiwania kodu w języku SQL, oraz proponuje zabezpieczenia przed nim. Podrozdział 30.5 krótko omawia problem bezpieczeństwa statystycznych baz danych. Podrozdział 30.6 wprowadza problematykę kontroli przepływu i ukrytych kanałów. Podrozdział 30.7 stanowi związane omówienie kwestii szyfrowania i infrastruktury klucza asymetrycznego (publicznego). Opisano tam także certyfikaty cyfrowe. Podrozdział 30.8 to wprowadzenie do technik zapewniania prywatności, a w podrozdziale 30.9 omówione są aktualne trudności w dziedzinie zabezpieczeń baz danych. Podrozdział 30.10 to opis zabezpieczeń opartych na etykietach w bazach Oracle. Podrozdział 30.11 podsumowuje zawartość całego rozdziału. Czytelnik zainteresowany jedynie podstawowymi mechanizmami bezpieczeństwa baz danych może znaleźć interesujące go informacje w podrozdziałach 30.1 i 30.2.

## 30.1. Wprowadzenie do bezpieczeństwa baz danych<sup>1</sup>

### 30.1.1. Rodzaje zabezpieczeń

Bezpieczeństwo baz danych jest bardzo szerokim zagadnieniem, obejmujących wiele kwestii, w tym:

- Kwestie prawne i etyczne dotyczące dostępności informacji. Niektóre informacje mogą być określone jako prywatne i nie mogą być dostępne dla nieupoważnionych osób lub organizacji. W Stanach Zjednoczonych obowiązują liczne prawa dotyczące prywatności informacji.
- Polityki bezpieczeństwa na poziomie państwowym, instytucjonalnym lub firmowym dotyczące danych, które nie mogą być publicznie ujawniane (na przykład wyciągi bankowe lub dane medyczne).
- Kwestie systemowe takie jak systemowe *poziomy zabezpieczeń*, na których wdrażane są różne funkcje kontroli dostępu (na przykład zabezpieczenia realizowane na poziomie sprzętu, systemu operacyjnego lub systemu baz danych).
- W przypadku niektórych organizacji — potrzebę wprowadzenia wielu różnych *poziomów dostępu* oraz klasyfikacji w stosunku do ich wszystkich danych i użytkowników (na przykład podział na dane ściśle tajne, tajne, poufne i jawne). Polityka bezpieczeństwa takiej organizacji musi uwzględniać kontrolę dostępu do danych o różnych stopniach jawności.

**Zagrożenia wobec baz danych.** Zagrożenia wobec baz danych mogą spowodować utratę lub uszkodzenie niektórych lub wszystkich z następujących obszarów bezpieczeństwa: integralność, dostępność i poufność.

- **Utrata integralności.** Integralność bazy danych odnosi się do wymagania, aby dane były zabezpieczone przed niewłaściwymi modyfikacjami. Modyfikacje obejmują tworzenie, dodawanie, aktualizację, zmianę statusu danych oraz ich usuwanie. Integralność jest utracona, jeżeli wskutek celowego lub przypadkowego działania dane zostały zmodyfikowane w nieautoryzowany sposób. Jeśli utrata integralności danych lub systemu nie zostanie skorygowana, to dalsza praca uszkodzonego systemu może spowodować jego niedokładne lub błędne działanie albo skutkować oszustwem.
- **Utrata dostępności.** *Dostępność bazy danych* oznacza możliwość uzyskania przez człowieka lub program dostępu do obiektów, do których odczytywania są uprawnieni. *Utrata dostępności* ma miejsce, gdy dany użytkownik lub program nie może korzystać z takich obiektów.
- **Utrata poufności.** Poufność bazy danych odnosi się do ochrony danych przed nieautoryzowanym ujawnieniem. Skutki uzyskania nieautoryzowanego dostępu do niejawnych informacji mogą być różne, od naruszenia prawa do ochrony korespondencji po zagrożenie bezpieczeństwa państwowego. Nieautoryzowane, przypadkowe lub nieprzewidziane naruszenie poufności może skutkować utratą zaufania, niezręczną sytuacją, ale też konsekwencjami prawnymi wobec organizacji zarządzającej bazą danych.

---

<sup>1</sup> Dziękujemy Fariborzowi Farahmandowi, Bharathowi Rengarajanowi i Frankowi Rietcie za istotny wkład w opracowanie tego i dalszych podrozdziałów z tego rozdziału.

**Bezpieczeństwo baz danych nie jest odizolowanym zagadnieniem.** Gdy analizujesz zagrożenia związane z bazami danych, ważne jest, aby pamiętać, że sam SZBD nie może odpowiadać za utrzymanie poufności, integralności i dostępności danych. Baza danych jest częścią sieci usług, w tym aplikacji, serwerów WWW, zapór, mechanizmów przerywania połączeń SSL i systemów monitorowania bezpieczeństwa. Ponieważ bezpieczeństwo całego systemu jest często zależne od najsłabszego ogniwa, włamanie do bazy danych może nastąpić nawet wtedy, gdy ona sama została idealnie zabezpieczona.

Aby chronić bazy danych przed powyższymi zagrożeniami, stosuje się *cztery sposoby zabezpieczeń*: kontrola dostępu, kontrola wnioskowania, kontrola przepływu i szyfrowanie. Każdy z nich zostanie omówiony dalej w tym rozdziale.

W wieloużytkownikowym systemie baz danych SZBD musi zapewnić mechanizmy pozwalające wybranym użytkownikom lub grupom użytkowników na dostęp do wybranych fragmentów bazy danych bez możliwości korzystania z pozostałej jej części. Jest to szczególnie istotne, gdy jedna zintegrowana baza danych jest używana przez wielu różnych użytkowników wewnątrz przedsiębiorstwa. Na przykład, informacje dotyczące wynagrodzeń pracowników i raporty dotyczące wydajności powinny być utajnione przed większością użytkowników systemu. SZBD zazwyczaj zawiera **podsystem uwierzytelniania i zabezpieczeń** bazy danych, który jest odpowiedzialny za zapewnienie bezpieczeństwa fragmentów bazy danych przed nieautoryzowanym dostępem. Można rozróżnić dwa typy mechanizmów zabezpieczeń bazy danych:

- **Dyspozycyjny mechanizm bezpieczeństwa.** Polega na nadawaniu użytkownikom uprawnień określających dostęp do określonych plików, rekordów i atrybutów w określonym trybie (odczytu, dodawania, usuwania lub modyfikowania).
- **Obowiązkowy mechanizm bezpieczeństwa.** Jest używany do wprowadzenia wielu poziomów bezpieczeństwa poprzez przypisywanie danych i użytkowników do różnych klas (poziomów) bezpieczeństwa. Typowa polityka bezpieczeństwa polega na umożliwieniu użytkownikowi na określonym poziomie bezpieczeństwa dostępu tylko do danych przydzielonych do tego samego (lub niższego) poziomu. Rozszerzeniem tej koncepcji jest *mechanizm bezpieczeństwa oparty o role*, który polega na określaniu dostępu do danych na podstawie zdefiniowanych ról użytkowników. Kontrolę dostępu opartą na rolach opisano w punkcie 30.4.2.

Omówimy dyspozycyjny model bezpieczeństwa w podrozdziale 30.2, a model obowiązkowy i model z użyciem ról w podrozdziale 30.3.

### 30.1.2. Środki kontroli

Do zapewnienia bezpieczeństwa danych w bazie stosowane są cztery podstawowe środki kontroli:

- kontrola dostępu,
- kontrola wnioskowania,
- kontrola przepływu,
- szyfrowanie danych.

Problemem związanym z zabezpieczeniami dotyczącym większości systemów komputerowych jest zapobieganie nieautoryzowanemu dostępowi do samego systemu komputerowego, który to dostęp może mieć na celu uzyskanie informacji lub dokonanie złośliwych zmian we fragmencie bazy danych. Mechanizmy zabezpieczeń bazy danych muszą pozwalać na ograniczanie dostępu do całego systemu. Ta funkcja nazywana jest **kontrolą dostępu** (ang. *access control*) i jest realizowana poprzez tworzenie kont użytkowników i haseł, które pozwalają na kontrolowanie procesu logowania przez system baz danych. Omówimy techniki kontroli dostępu w punkcie 30.1.3.

**Statystyczne bazy danych** są używane do dostarczania informacji statystycznych lub podsumowań wartości na podstawie różnych kryteriów. Na przykład, baza danych o ludności może zapewniać dostęp do statystyk o grupach wiekowych, poziomach zamożności, wielkości gospodarstw domowych, poziomach wykształcenia itp. Użytkownicy statystycznej bazy danych, tacy jak statystycy rządowi lub pracownicy firm przeprowadzających badania rynku, mogą uzyskać dostęp do danych statystycznych, ale nie do szczegółowych danych na temat określonych osób. Zabezpieczenia tego typu baz danych muszą zapewniać brak dostępu do danych o pojedynczych osobach. Czasami można wnioskować lub łączyć dane statystyczne oparte na różnych kryteriach wyboru w celu uzyskania informacji o konkretnej osobie; zabezpieczenia bazy danych powinny również chronić przed takimi działaniami. Ten problem, nazywany **bezpieczeństwem statystycznych baz danych**, jest omówiony pokrótce w podrozdziale 30.4. Odpowiadające mu zabezpieczenia nazywamy **kontrolą wnioskowania** (ang. *inference control*).

Kolejnym zagadnieniem bezpieczeństwa jest **kontrola przepływu** (ang. *flow control*), która zabezpiecza przed takim przepływem danych, który pozwalałby na dostęp do nich nieautoryzowanemu użytkownikowi. Problem ten jest omawiany w podrozdziale 30.6. Kanały, które pozwalają na niejawny przepływ danych w sposób naruszający politykę bezpieczeństwa, nazywane są **kanałami ukrytymi** (ang. *covert channels*). Omówimy pokrótce związane z nimi kwestie w punkcie 30.6.1.

Ostatnim środkiem kontroli jest **szyfrowanie danych**, używane do zabezpieczania poufnych danych (takich jak numery kart kredytowych) podczas przesyłania przez jakikolwiek rodzaj sieci komunikacyjnej. Szyfrowanie może również służyć do zapewnienia dodatkowej ochrony fragmentom bazy danych. Dane są **szyfrowane** za pomocą wybranego algorytmu. Nieautoryzowany użytkownik po uzyskaniu dostępu do systemu bazy danych nie będzie mógł ich odszyfrować, w odróżnieniu od autoryzowanych użytkowników znających algorytm (i klucz) pozwalające na rozszyfrowanie danych. Dla zastosowań wojskowych opracowano techniki szyfrowania, które są bardzo trudne do złamania. Jednak szyfrowane rekordy baz danych występują obecnie w organizacjach prywatnych, rządowych i wojskowych. Regulacje stanowe i federalne nakazują szyfrowanie wszystkich systemów przechowujących chronione prawem dane osobowe. Oto np. wyciąg z prawa stanu Georgia (OCGA 10-1-911):

„Dane osobowe” oznaczają imię (lub inicjał) i nazwisko w połączeniu z podanymi dalej elementami danych, jeśli albo nazwisko, albo wymienione elementy nie są zaszyfrowane lub zmodyfikowane:

- numer ubezpieczenia społecznego;
- numer prawa jazdy lub numer stanowej karty identyfikacyjnej;



- numer konta, numer karty kredytowej lub numer karty debetowej, jeśli możliwe jest wykorzystanie takiego numeru bez dodatkowych danych identyfikacyjnych, kodów dostępu lub haseł;
- hasła do kont, numery identyfikacyjne lub inne kody dostępu.

Ponieważ prawa definiujące, czym są dane osobowe, różnią się między stanami, systemy muszą chronić prywatność użytkowników i odpowiednio korzystać ze środków zapewniających prywatność. Sama dyspozycyjna kontrola dostępu (patrz podrozdział 30.2) może nie wystarczyć. W podrozdziale 30.7 pokrótce omówimy techniki szyfrowania, w tym popularne metody, takie jak szyfrowanie z użyciem klucza publicznego (często używane w internetowych transakcjach z wykorzystaniem baz danych) i podpisów cyfrowych (stosowanych w komunikacji osobistej).

Pełne omówienie zagadnień bezpieczeństwa systemów komputerowych i baz danych wykracza poza zakres zagadnień poruszanych w tej książce. Prezentowany jest tutaj jedynie zwięzły przegląd technik zabezpieczania baz danych. Pomijamy tu obszernie zagadnienie zabezpieczeń sieci i komunikacji. Czytelnik zainteresowany tym tematem może skorzystać z książek wymienionych w bibliografii zamieszczonej pod koniec rozdziału.

### 30.1.3. Bezpieczeństwo a administrator bazy danych

Jak już zostało wspomniane w rozdziale 1., administrator bazy danych (DBA) jest centralną postacią zarządzającą systemem bazy danych. Do zadań DBA należy nadawanie uprawnień użytkownikom, którzy mają korzystać z systemu, i klasyfikacja użytkowników oraz danych według przyjętej polityki bezpieczeństwa przedsiębiorstwa. DBA ma w bazie danych **konto administratora**, nazywane też **kontem systemowym** lub **kontem superużytkownika**, które oferuje możliwości niedostępne dla zwykłego użytkownika systemu bazy danych<sup>2</sup>. Komendy dostępne tylko dla DBA obejmują operacje nadawania i odbierania uprawnień poszczególnym kontom, użytkownikom, grupom użytkowników oraz wykonywanie następujących czynności:

- (1) **Tworzenie kont** — ta akcja pozwala na tworzenie nowych kont i haseł dla użytkowników lub grup użytkowników potrzebujących dostępu do SZBD.
- (2) **Nadawanie uprawnień** — pozwala administratorowi na nadawanie określonych uprawnień poszczególnym kontom.
- (3) **Odbieranie uprawnień** — pozwala na odebranie (anulowanie) określonych uprawnień wcześniej przypisanych do konkretnego konta.
- (4) **Określanie poziomów bezpieczeństwa** — polega na przypisywaniu kont użytkowników do odpowiednich poziomów bezpieczeństwa.

Administrator jest odpowiedzialny za ogólne bezpieczeństwo systemu bazy danych. Działanie 1. z powyższej listy jest używane do ograniczania dostępu do całego systemu bazy danych, podczas gdy działania 2. i 3. są używane do określania *dyspozycyjnej* autoryzacji w bazie danych, a działanie 4. do określania autoryzacji *obowiązkowej*.

---

<sup>2</sup> Takie konto jest podobne do przypisywanych administratorom kont *root* lub *superuser*, które pozwalają na używanie komend systemu operacyjnego niedostępnych dla pozostałych użytkowników.

### 30.1.4. Ochrona dostępu, konta użytkowników i audyty bazy danych

Kiedy osoba (lub grupa osób) potrzebuje dostępu do systemu bazy danych, to osoba ta (lub grupa) musi wystąpić o stworzenie konta użytkownika. Administrator bazy danych tworzy wówczas nowy **identyfikator konta** oraz **hasło**, o ile użytkownik rzeczywiście ma prawo korzystać z zasobów bazy danych. Użytkownik musi się **zalogować** do systemu bazy danych podając identyfikator konta oraz hasło za każdym razem, kiedy potrzebuje dostępu do zgromadzonych informacji. Następnie SZBD sprawdza, czy identyfikator i hasło są ważne, i jeżeli są, zezwala na używanie systemu i zgromadzonych w nim danych. Programy komputerowe są również traktowane jak użytkownicy i wymaga się od nich zalogowania (patrz rozdział 10.).

Najprostszym sposobem kontrolowania listy użytkowników oraz ich kont i haseł jest stworzenie zaszyfrowanej tabeli lub pliku z dwiema kolumnami: *IdentyfikatorKonta* i *Hasło*. Tabela taka może być łatwo zarządzana przez SZBD. Podczas tworzenia nowego konta do tabeli dodawana jest nowa krotka, a przy kasowaniu konta — odpowiadająca mu krotka jest usuwana.

System bazy danych musi również przechowywać zapis wszystkich operacji wykonanych przez danego użytkownika podczas **sesji logowania**, składający się z listy wszystkich komend wykonywanych przez użytkownika na bazie danych od momentu zalogowania się do wylogowania. Podczas logowania użytkownika system bazy danych może zapisać identyfikator konta użytkownika i przypisać go do komputera lub urządzenia, z którego użytkownik się zalogował. Wszystkie operacje przeprowadzone za pośrednictwem tego komputera lub urządzenia są przypisywane zalogowanemu użytkownikowi aż do momentu wylogowania. Szczególnie istotne jest przechowywanie informacji o operacjach modyfikujących zawartość bazy danych. Dzięki temu po wykryciu manipulacji w bazie danych administrator ma możliwość sprawdzenia, kto je spowodował.

Do przechowywania zapisu wszystkich modyfikacji i użytkowników, którzy je wprowadzili, można dostosować *systemowy dziennik zdarzeń*. Przypomnijmy z rozdziałów 20. i 22., że **dziennik systemowy** zawiera wpisy dla każdej operacji przeprowadzonej na bazie danych, dzięki czemu można odzyskać dane po błędnej transakcji lub awarii systemu. Można rozszerzyć wpisy dziennika tak, aby zawierały również identyfikator konta użytkownika i identyfikator komputera lub urządzenia użytego do przeprowadzenia danej operacji. Przy podejrzeniu jakiegokolwiek zafałszowania danych w bazie wykonuje się **audyt bazy danych**, który polega na przejrzeniu wszystkich operacji zapisanych w dzienniku zdarzeń w określonym przedziale czasu. Jeśli podczas audytu zostanie stwierdzona nieautoryzowana lub niedozwolona operacja, to administrator może określić, kto jej dokonał. Audyty baz danych są szczególnie istotne w przypadku informacji niejawnych, zmienianych przez liczne transakcje i wielu użytkowników; dotyczy to np. bankowych baz danych modyfikowanych przez tysiące kasjerów. Dziennik bazy danych używany głównie do celów związanych z bezpieczeństwem danych jest często nazywany **raportem audytu** (ang. *audit trail*).

### 30.1.5. Dane wrażliwe i typy ujawnień

**Poziom wrażliwości danych** to miara znaczenia danych przypisywana im przez ich właściciela w celu określenia potrzeby ochrony. Niektóre bazy zawierają tylko dane wrażliwe, natomiast inne mogą w ogóle ich nie obejmować. Zarządzanie tak skrajnymi bazami jest stosunkowo łatwe, ponieważ można wykorzystać do tego kontrolę dostępu (patrz następny podrozdział). Zadanie jest trudniejsze, gdy niektóre dane są wrażliwe, natomiast inne — nie.

Dane mogą zostać uznane za wrażliwe z kilku powodów.

- (1) **Dane z natury wrażliwe.** Dane mogą być na tyle poufne lub ujawniające informacje o danej osobie, że stają się wrażliwe. Jest to np. wynagrodzenie pracownika lub informacja o tym, czy dana osoba jest zarażona wirusem HIV lub chora na AIDS.
- (2) **Dane z wrażliwego źródła.** Źródło danych może wskazywać na potrzebę poufności; mogą one pochodzić np. od informatora, którego tożsamość musi pozostać tajna.
- (3) **Opisane jako wrażliwe.** Właściciel danych może bezpośrednio opisać je jako wrażliwe.
- (4) **Z wrażliwymi atrybutami lub rekordami.** Konkretny atrybut lub rekord może zostać uznany za wrażliwy. Dotyczy to np. atrybutu wynagrodzenia pracownika lub historii rekordów z wynagrodzeniami w bazie z danymi pracowników.
- (5) **Wrażliwe w powiązaniu z wcześniej ujawnionymi danymi.** Niektóre dane nie są wrażliwe same w sobie, ale stają się takie w obecności innych danych. Mogą to być np. dokładne informacje o współrzędnych geograficznych miejsca, gdzie nastąpiło zdarzenie uznane później za poufne.

To administrator bazy danych i administrator zabezpieczeń odpowiadają za zespołowe wymuszanie polityk bezpieczeństwa w organizacji. Określają one, czy poszczególnym użytkownikom lub ich grupom należy zapewnić dostęp do określonego atrybutu bazy danych (*elementu tabeli* lub *elementu danych*). Przed podjęciem decyzji o tym, czy ujawnienie danych jest bezpieczne, trzeba uwzględnić kilka czynników. Trzy najważniejsze z nich to dostępność danych, akceptowalność dostępu i wiarygodność uwierzytelnienia.

- (1) **Dostępność danych.** Jeśli użytkownik aktualizuje pole, staje się ono niedostępne i inne osoby nie powinny widzieć takich danych. Taka blokada jest tylko tymczasowa i ma gwarantować, że żaden użytkownik nie zobaczy żadnych nieprawidłowych informacji. Zwykle jest to zapewniane za pomocą mechanizmów sterowania współbieżnego (patrz rozdział 21.).
- (2) **Akceptowalność dostępu.** Dane powinny być ujawniane tylko uwierzytelnionym użytkownikom. Administrator bazy danych może odmówić dostępu nawet wtedy, gdy żądanie użytkownika nie dotyczy bezpośrednio wrażliwego elementu danych, ale może skutkować ujawnieniem informacji o danych wrażliwych, do których ta osoba nie ma uprawnień.
- (3) **Wiarygodność uwierzytelnienia.** Przed przyznaniem dostępu można uwzględnić także pewne zewnętrzne cechy związane z użytkownikiem. Użytkownik może np. mieć uprawnienia dostępu tylko w godzinach pracy. System może też śledzić wcześniejsze zapytania, aby zapewnić, że określona kombinacja zapytań nie spowoduje ujawnienia wrażliwych danych. Jest to istotne zwłaszcza w zapytaniach do statystycznych baz danych (patrz podrozdział 30.5).

Określenie *precyzja* w kontekście zabezpieczeń dotyczy udostępniania możliwie wielu danych i ochrony wyłącznie podzbioru wrażliwych danych. Oto definicje *zabezpieczeń* i *precyzji*:

- **Zabezpieczenia.** Środki chroniące dane przed uszkodzeniem i zapewniające odpowiednią kontrolę dostępu do danych. Zapewnianie bezpieczeństwa wymaga ujawniania tylko zwykłych danych i odrzucania zapytań dotyczących wrażliwych pól.
- **Precyzja.** Ma służyć ochronie wszystkich wrażliwych danych przy jednoczesnym ujawnianiu lub udostępnianiu możliwie wielu zwykłych danych. Warto zauważyć, że to znaczenie *precyzji* nie jest związane ze zdefiniowaną w punkcie 27.6.1 precyzją w wyszukiwaniu informacji.

Idealnym połączeniem jest zapewnienie pełnego bezpieczeństwa przy maksymalnej precyzji. Jeśli chcesz zachować bezpieczeństwo, musisz w pewnym stopniu poświęcić precyzję. Dlatego zwykle niezbędny jest kompromis między bezpieczeństwem a precyzją.

### 30.1.6. Związki między bezpieczeństwem a prywatnością informacji

Szybki wzrost zakresu zastosowań technologii informatycznych w przemyśle, jednostkach rządowych i na uczelniach rodzi trudne pytania i problemy związane z ochroną i wykorzystaniem danych osobowych. Pytania o to, *kto* ma *jakie* prawa do informacji o ludziach i w *jakich* celach może korzystać z tych danych, stają się coraz ważniejsze, gdy zbliżamy się do momentu, w którym technicznie możliwe jest dowiedzenie się prawie wszystkiego o niemal każdym.

W ramach projektowania rozwiązań z zakresu prywatności na potrzeby przyszłości trzeba uwzględnić wymiary filozoficzne, prawne i praktyczne. Zagadnienia związane z dostępem do zasobów (bezpieczeństwo) i właściwym wykorzystaniem informacji (prywatność) w dużym stopniu się pokrywają. Teraz zdefiniujemy różnicę między *bezpieczeństwem* a *prywatnością*.

**Bezpieczeństwo** w technologiach informatycznych dotyczy wielu aspektów ochrony systemu przed nieuprawnionym użyciem. Obejmuje to kwestie uwierzytelniania użytkowników, szyfrowania informacji, kontroli dostępu, polityk pracy, zapór i wykrywania włamań. Dla naszych celów ograniczymy omawianie bezpieczeństwa do tego, jak dobrze system potrafi zabezpieczać dostęp do przechowywanych informacji. Kwestia **prywatności** wykracza poza bezpieczeństwo. Prywatność związana jest z tym, w jakim stopniu używanie danych osobowych rejestrowanych w systemie jest zgodne z bezpośrednimi lub pośrednimi założeniami dotyczącymi stosowania tych danych. Dla użytkownika końcowego prywatność może być ważna w dwóch ujęciach: *zapobiegania rejestrowaniu* danych osobowych i *zapewniania właściwego wykorzystania* takich danych.

Na potrzeby tego rozdziału prezentujemy prostą, ale przydatną definicję **prywatności**: *możliwość kontrolowania przez użytkowników warunków, na jakich ich dane osobowe są rejestrowane i użytkowane*. W podsumowaniu można stwierdzić, że bezpieczeństwo jest oparte na technologiach zapewniających właściwą ochronę informacji. Bezpieczeństwo jest niezbędne do zapewnienia prywatności. Prywatność wymaga mechanizmów zapew-

niających zgodność z podstawowymi zasadami i innymi bezpośrednio opisanymi politykami. Jedną z głównych zasad polega na tym, że użytkowników należy informować o zbieraniu danych oraz z góry powiadamiać o tym, jak będą wykorzystywane te informacje; trzeba też dawać im realną możliwość zatwierdzenia lub odrzucenia takiego zastosowania danych. Powiązane zagadnienie, **zaufanie**, dotyczy zarówno bezpieczeństwa, jak i prywatności. Zaufanie rośnie, gdy bezpieczeństwo i prywatność są zapewnione.

## 30.2. Dyspozycyjna kontrola dostępu polegająca na nadawaniu i odbieraniu uprawnień

Typowym sposobem wprowadzania **zabezpieczeń dyspozycyjnych** (ang. *discretionary security*) w systemach baz danych jest nadawanie i odbieranie **uprawnień**. Rozważmy system uprawnień w kontekście relacyjnego systemu bazy danych. W szczególności, omówimy system uprawnień podobny do tego, który został wprowadzony do języka SQL (patrz rozdziały 7. i 8.). Wiele obecnych relacyjnych SZBD używa odmian tej metody. Podstawowa koncepcja polega na zastosowaniu w języku zapytań wyrażeń, które pozwolą administratorowi i wybranym użytkownikom na nadawanie i odbieranie uprawnień.

### 30.2.1. Typy uprawnień dyspozycyjnych

W SQL2 i nowszych wersjach tego języka<sup>3</sup> pojęcie **identyfikatora uwierzytelnienia** jest używane w odniesieniu do konta użytkownika (lub grupy kont). Dla uproszczenia będziemy tu używali w miejsce wyrażenia *identyfikator uwierzytelnienia* zamiennie słów *użytkownik* i *konto*. System baz danych musi pozwalać na określanie dostępu określonych użytkowników do poszczególnych relacji. Można również kontrolować operacje. Posiadanie konta nie oznacza zatem od razu możliwości wykorzystania wszystkich funkcji oferowanych przez system bazy danych. Można wydzielić dwa nieformalne poziomy nadawania uprawnień:

- **poziom konta** — na tym poziomie administrator określa uprawnienia przypisane do konta, niezależnie od relacji istniejących w bazie danych;
- **poziom relacji (tabeli)** — administrator może kontrolować dostęp do każdej relacji lub perspektywy określonej w strukturze bazy danych.

Uprawnienia na **poziomie konta** dotyczą możliwości samego konta i mogą zawierać takie uprawnienia jak `CREATE SCHEMA` lub `CREATE TABLE` pozwalające na tworzenie schematu lub relacji; `CREATE VIEW` do tworzenia perspektyw; `ALTER` pozwalające na modyfikacje schematu takie jak dodawanie i usuwanie atrybutów relacji; `DROP` pozwalające na usuwanie relacji i perspektyw; `MODIFY` pozwalające na dodawanie, kasowanie i modyfikowanie krotek; `SELECT` pozwalające na odczytywanie informacji zapisanych w bazie danych poprzez polecenie `SELECT`. Należy zwrócić uwagę, że te uprawnienia dotyczą całego konta. Jeśli dany użytkownik nie ma uprawnienia `CREATE TABLE`, to nie będzie mógł tworzyć żadnych

---

<sup>3</sup> Uprawnienia dyspozycyjne zostały wprowadzone w wersji SQL2 i są dostępne także w nowszych wersjach języka SQL.

relacji. Uprawnienia na poziomie konta *nie są* częścią standardu SQL2 i mogą być określane dowolnie przez producentów SZBD. We wcześniejszych wersjach SQL istniało uprawnienie CREATETAB pozwalające na tworzenie tabel (relacji).

Drugi poziom uprawnień dotyczy **poziomu relacji** i dotyczy zarówno rzeczywistych relacji określonych w strukturze bazy, jak i relacji wirtualnych (perspektyw). Określanie tych uprawnień *jest* zdefiniowane w SQL2. W dalszych rozważaniach będziemy używali terminu *relacja* w odniesieniu zarówno do relacji określonych w strukturze bazy danych (nazywanych też relacjami bazowymi), jak i do perspektyw, chyba że zostanie zaznaczone inaczej. Uprawnienia na poziomie relacji określają, jakie typy komend mogą być wykonywane przez określonych użytkowników na określonych relacjach. Niektóre uprawnienia dotyczą również pojedynczych kolumn (atrybutów) relacji. Komendy SQL2 pozwalają na określanie uprawnień *jedynie na poziomie relacji i atrybutów*. Mimo dość dużej ogólności, utrudnia to tworzenie kont z ograniczonymi uprawnieniami. Nadawanie i cofanie poszczególnych uprawnień odbywa się według modelu uprawnień dla zabezpieczeń dyspozycyjnych znanego jako **model macierzy dostępu**, gdzie wiersze macierzy  $M$  przedstawiają *podmioty* uprawnień (użytkowników, konta, programy) a kolumny odpowiadają ich *przedmiotom* (relacjom, rekordom, kolumnom, perspektywom, operacjom). Każda pozycja  $M(i, j)$  w macierzy określa typ uprawnień (odczyt, zapis, zmiana) podmiotu  $i$  do przedmiotu  $j$ .

Na potrzeby kontrolowania uprawnień dotyczących relacji każda relacja  $R$  w bazie danych ma określonego **właściciela** — zazwyczaj jest nim użytkownik, który tę relację stworzył. Właścicielowi relacji nadawane są *wszystkie* uprawnienia do niej. W SQL2 administrator może określić właściciela całego schematu, przypisując mu po jego stworzeniu określone konto użytkownika za pomocą komendy CREATE SCHEMA (patrz punkt 7.1.1). Właściciel może przekazać swoje uprawnienia do którejkolwiek z posiadanych relacji innym użytkownikom poprzez **nadanie** uprawnień ich kontom. W SQL dla każdej relacji  $R$  można przypisać następujące uprawnienia:

- **SELECT (odczyt) do relacji  $R$**  — nadaje użytkownikowi uprawnienia odczytu. W języku SQL pozwala to na użycie komendy SELECT do odczytywania krotek relacji  $R$ .
- **MODIFY do relacji  $R$**  — daje użytkownikowi możliwość modyfikacji krotek relacji  $R$ . W SQL to uprawnienie jest dalej podzielone na uprawnienia UPDATE, DELETE i INSERT, odnoszące się do odpowiednich komend SQL (patrz podrozdział 7.4), które dany użytkownik może zastosować do relacji  $R$ . Dodatkowo dla uprawnień UPDATE i INSERT można określić uprawnienia do modyfikacji tylko określonych atrybutów relacji.
- **REFERENCES do relacji  $R$**  — pozwala na *odwoływanie* się do relacji  $R$  przy określaniu więzów integralności. To uprawnienie również może być zawężone do pojedynczych atrybutów.

Należy zwrócić uwagę, że do stworzenia perspektywy użytkownik musi mieć uprawnienia SELECT do *wszystkich relacji* wykorzystywanych w jego definicji.

### 30.2.2. Określanie uprawnień przy użyciu perspektyw

Mechanizm **perspektyw** jest istotnym elementem w *schemacie zabezpieczeń dyspozycyjnych*. Jeżeli właściciel  $A$  relacji  $R$  chce udostępnić użytkownikowi  $B$  tylko niektóre atrybuty swojej relacji, to  $A$  może utworzyć perspektywę  $V$  zawierającą wybrane atrybuty relacji  $R$ ,



a następnie nadać użytkownikowi *B* uprawnienia SELECT do perspektywy *V*. To samo odnosi się do ograniczenia dostępu użytkownika *B* do niektórych krotek relacji *R*. Wystarczy zdefiniować za pomocą zapytania nową perspektywę *V'*, zawierającą tylko te krotki relacji *R*, które mają być dostępne dla użytkownika *B*. Zastosowanie perspektyw do definiowania zabezpieczeń zostanie zilustrowane odpowiednim przykładem w punkcie 30.2.5.

### 30.2.3. Cofanie uprawnień

W niektórych przypadkach użytkownik powinien mieć pewne uprawnienia tylko tymczasowo. Na przykład, właściciel relacji może chcieć nadać uprawnienia SELECT użytkownikowi tylko na czas wykonywania jakiegoś zadania, a następnie je cofnąć. Potrzebny jest zatem mechanizm pozwalający na **cofanie** (anulowanie) uprawnień. W języku SQL do anulowania uprawnień służy komenda REVOKE. Działanie tej komendy będzie zilustrowane przykładem w punkcie 30.2.5.

### 30.2.4. Propagacja uprawnień poprzez opcję GRANT

Kiedy właściciel *A* relacji *R* nadaje uprawnienia do *R* innemu użytkownikowi *B*, to uprawnienia te mogą być nadane z *opcją* GRANT lub *bez niej*. Jeśli opcja GRANT jest nadana użytkownikowi, to oznacza to, że *B* może swoje uprawnienia nadawać innym użytkownikom. Przypuśćmy, że *B* ma nadane przez *A* uprawnienia z opcją GRANT, a następnie nadaje swoje uprawnienia do relacji *R* kolejnemu użytkownikowi *C*, również z opcją GRANT. W ten sposób uprawnienia do *R* mogą być **propagowane** na inne konta bez wiedzy właściciela relacji. Jeśli *A* anuluje uprawnienia nadane *B*, to wszystkie propagowane za pośrednictwem *B* uprawnienia *powinny zostać automatycznie cofnięte* przez system.

Istnieje możliwość otrzymania przez użytkownika uprawnień do tej samej relacji z dwóch różnych źródeł. Na przykład *A4* może otrzymać uprawnienia UPDATE *R* zarówno od *A2*, jak i *A3*. W takiej sytuacji jeśli *A2* anuluje nadane przez siebie uprawnienia, *A4* nadal będzie je miał, ponieważ zostały mu nadane również przez *A3*. Jeżeli później *A3* również cofnie swoje uprawnienia — *A4* całkowicie traci uprawnienia do *R*. System bazy danych pozwalający na propagację uprawnień musi więc przechowywać pełny zapis informacji o tym, jak uprawnienia były nadawane, aby późniejsze ich cofanie było realizowane prawidłowo i kompletnie.

### 30.2.5. Przykład

Przypuśćmy, że DBA tworzy cztery konta — *A1*, *A2*, *A3*, i *A4* — i chce, aby tylko *A1* miał możliwość tworzenia relacji w bazie danych. DBA musi więc wykonać następującą komendę:

```
GRANT CREATETAB TO A1;
```

Uprawnienia CREATETAB (tworzenia tabeli) daje użytkownikowi *A1* możliwość tworzenia nowych tabel (relacji) w bazie danych, jest więc uprawnieniem na *poziomie konta*. Niegdyś uprawnienie CREATETAB było częścią SQL, obecnie decyzja o jego implementacji jest pozostawiona producentom SZBD. Warto zauważyć, że *A1*, *A2* itd. mogą być użytkownikami, np. Janem z działu informatycznego lub Marią z działu marketingu, ale też aplikacjami lub programami needing dostępu do bazy.



W SQL2 ten sam rezultat można osiągnąć komendą `CREATE SCHEMA` w następujący sposób:

```
CREATE SCHEMA PRZYKŁAD AUTHORIZATION A1;
```

Teraz konto *A1* może tworzyć nowe tabele w schemacie o nazwie *PRZYKŁAD*. Kontynuując przykład, przypuśćmy, że *A1* tworzy dwie podstawowe relacje (tabele) *PRACOWNIK* i *DZIAŁ* pokazane na rysunku 30.1. *A1* jest **właścicielem** tych dwóch relacji, a zatem ma do nich *wszystkie uprawnienia*.

#### PRACOWNIK

IMIĘNAZWISKO	PESEL	DATAUR	ADRES	PŁEĆ	PENSJA	NRDZ
--------------	-------	--------	-------	------	--------	------

#### DZIAŁ

NUMERDZ	NAZWADZ	PESELKIEROWNIKA
---------	---------	-----------------

Rysunek 30.1. Schematy relacji *PRACOWNIK* i *DZIAŁ*

Następnie przypuśćmy, że użytkownik *A1* chce nadać użytkownikowi *A2* uprawnienia do dodawania i usuwania krotek w obu relacjach. *A1* nie chce jednak, aby *A2* mógł przekazywać swoje uprawnienia innym. *A1* powinien więc wykonać następującą komendę:

```
GRANT INSERT, DELETE ON PRACOWNIK, DZIAŁ TO A2;
```

Zauważmy, że konto właściciela relacji (*A1*) automatycznie ma opcję `GRANT` pozwalającą na nadawanie innym swoich uprawnień. Jednak *A2* nie może nadawać uprawnień `SELECT` ani `DELETE` do tabel *PRACOWNIK* i *DZIAŁ*, ponieważ *A2* nie została nadana opcja `GRANT`.

Przypuśćmy dalej, że *A1* chce pozwolić użytkownikowi *A3* na odczytywanie informacji z obu tabel i chce umożliwić propagowanie tych uprawnień dalej. *A1* może to zrealizować za pomocą komendy

```
GRANT SELECT ON PRACOWNIK, DZIAŁ TO A3 WITH GRANT OPTION;
```

Wyrażenie `WITH GRANT OPTION` oznacza, że *A3* może teraz propagować swoje uprawnienia na inne konta używając polecenia `GRANT`. Na przykład, *A3* może nadać uprawnienia `SELECT` do relacji *PRACOWNIK* użytkownikowi *A4*, wykonując następującą komendę:

```
GRANT SELECT ON PRACOWNIK TO A4;
```

Zauważmy, że *A4* nie może propagować swoich uprawnień na inne konta, ponieważ nie została mu nadana opcja `GRANT`.

Przypuśćmy teraz, że *A1* decyduje się cofnąć uprawnienia `SELECT` do relacji *PRACOWNIK* nadane wcześniej *A3*. *A1* wykonuje więc komendę:

```
REVOKE SELECT ON PRACOWNIK FROM A3;
```

SZBD musi teraz cofnąć uprawnienie `SELECT` do relacji *PRACOWNIK* użytkownikowi *A3*, a także *automatycznie cofnąć* uprawnienie `SELECT` również użytkownikowi *A4*, ponieważ *A3*, który nadał mu to uprawnienie, sam go już nie posiada.

Dalej, przypuśćmy, że *A1* chce nadać ponownie użytkownikowi *A3* ograniczoną możliwość przeszukiwania relacji *PRACOWNIK* i pozwolić mu na propagowanie tych uprawnień.

Ograniczenie polega na możliwości odczytania tylko atrybutów IMIĘNAZWISKO, DATAUR oraz ADRES, i to tylko dla krotek o wartości NRZD równej 5. A1 tworzy więc perspektywę:

```
CREATE VIEW A3PRACOWNIK AS
SELECT IMIĘNAZWISKO, DATAUR, ADRES
FROM PRACOWNIK
WHERE NRZD = 5;
```

Po stworzeniu perspektywy A1 może nadać A3 uprawnienia SELECT do A3pracownik w następujący sposób:

```
GRANT SELECT ON A3PRACOWNIK TO A3 WITH GRANT OPTION;
```

W końcu przypuścmy, że A1 chce pozwolić A4 na aktualizację tylko atrybutu PENSJA w tabeli PRACOWNIK. A1 może uzyskać to następującą komendą:

```
GRANT UPDATE ON PRACOWNIK (PENSJA) TO A4;
```

Dla uprawnień UPDATE i INSERT można określić atrybuty, które mogą być modyfikowane lub dodawane. Pozostałe uprawnienia (SELECT, DELETE) nie pozwalają na określenie atrybutów, których dotyczą, ale można tę samą funkcjonalność łatwo osiągnąć, tworząc odpowiednią perspektywę, zawierającą tylko wybrane atrybuty relacji. Uprawnienia UPDATE i INSERT są wzbogacone o możliwość zdefiniowania listy atrybutów, których dotyczą, ponieważ nie zawsze możliwe jest modyfikowanie perspektywy (patrz rozdział 8.).

### 30.2.6. Określanie ograniczeń propagacji uprawnień

Opracowano techniki ograniczania propagacji uprawnień, aczkolwiek nie zostały one jeszcze zaimplementowane w większości dostępnych SZBD i *nie są częścią* standardu SQL. Ograniczenie **propagacji poziomej** do danej liczby  $i$  oznacza, że użytkownik  $B$  posiadający opcję GRANT może przekazać swoje uprawnienia najwyżej  $i$  innym użytkownikom. **Propagacja pionowa** jest nieco bardziej skomplikowana i ogranicza głębokość nadawania uprawnień. Nadanie uprawnień z pionową propagacją równą zero oznacza przekazanie uprawnień *bez* opcji GRANT. Jeśli użytkownik  $A$  nadaje uprawnienia użytkownikowi  $B$  z pionową propagacją równą liczbie  $j > 0$ , oznacza to, że użytkownik  $B$  posiada opcję GRANT, ale może przekazać swoje uprawnienia tylko z pionową propagacją *mniejszą niż*  $j$ . W rezultacie, pionowa propagacja ogranicza ciąg opcji GRANT, które mogą być nadawane przez jednego użytkownika następnemu w oparciu o jedno oryginalne źródło uprawnień.

Zilustrujemy teraz krótkim przykładem ograniczenia propagacji poziomej i pionowej, które — przypomnijmy — *nie są obecnie dostępne* w SQL ani w większości systemów relacyjnych. Przypuścmy, że A1 nadaje A2 uprawnienia SELECT do relacji PRACOWNIK z poziomą propagacją równą 1 i pionową propagacją równą 2. A2 może teraz nadać uprawnienia SELECT najwyżej jednemu użytkownikowi, ponieważ propagacja pozioma jest równa 1. Ponadto, A2 może nadawać uprawnienia jedynie z pionową propagacją równą 0 (czyli bez opcji GRANT) lub 1, ponieważ A2 musi zmniejszyć podczas przekazywania uprawnień swoją propagację pionową o co najmniej 1. Oprócz tego propagacja pozioma nie może być wyższa niż podana pierwotnie w trakcie przyznawania uprawnień. Przykładowo, jeśli A nada uprawnienia użytkownikowi B z propagacją poziomą równą  $j > 0$ , oznacza to, że B może przyznać uprawnienia innym kontom tylko z propagacją poziomą *niższą lub równą*  $j$ . Jak widać w przytoczonym przykładzie, mechanizmy poziomej i pionowej propagacji pozwalają na skuteczne ograniczenie propagacji uprawnień.

### 30.3. Realizacja zabezpieczeń wielopoziomowych za pomocą obowiązkowej kontroli dostępu i zabezpieczeń opartych na rolach

Mechanizm zabezpieczeń dyspozycyjnych oparty na nadawaniu i odbieraniu uprawnień do określonych relacji jest tradycyjnie głównym mechanizmem kontroli dostępu w relacyjnych systemach baz danych. Jest to metoda typu „wszystko albo nic” — albo użytkownik posiada pewne uprawnienia, albo nie. W wielu aplikacjach istnieje potrzeba wprowadzenia *dodatkowej polityki bezpieczeństwa*, która klasyfikuje dane i użytkowników w oparciu o klasy bezpieczeństwa. Takie podejście, nazywane **obowiązkową kontrolą dostępu**, jest często *łączone* z zabezpieczeniami dyspozycyjnymi opisanymi w podrozdziale 30.2. Należy zwrócić uwagę, że większość dostępnych na rynku relacyjnych SZBD oferuje jedynie mechanizm dyspozycyjnej kontroli dostępu. Jednak potrzeba wielopoziomowych zabezpieczeń istnieje w zastosowaniach rządowych, wojskowych i wywiadowczych, a także w wielu aplikacjach przemysłowych i korporacyjnych. Z powodu ważnych obaw dotyczących prywatności w wielu systemach stosowane są poziomy określające, jaki dostęp mają poszczególne konta do prywatnych informacji (nazywanych danymi osobowymi). Niektórzy producenci SZBD, np. Oracle, oferują specjalne, przeznaczone do zastosowań rządowych wersje relacyjnych SZBD, obejmujące obowiązkową kontrolę dostępu.

Typowe **klasy bezpieczeństwa** to ściśle tajne ( $\acute{S}T$ ), tajne ( $T$ ), poufne ( $P$ ) i jawne ( $J$ ), gdzie  $\acute{S}T$  jest najwyższym poziomem bezpieczeństwa, a  $J$  najniższym. Istnieją również bardziej skomplikowane schematy klasyfikacji, w których klasy bezpieczeństwa tworzą kratę. Dla uproszczenia, w naszych rozważaniach będziemy używali czterech poziomów bezpieczeństwa, przy czym  $\acute{S}T \geq T \geq P \geq J$ . Powszechnie używany model zabezpieczeń wielopoziomowych, znany jako *model Bella LaPaduli*<sup>4</sup>, klasyfikuje każdy **podmiot** (użytkowników, konta, programy) i **przedmiot** zabezpieczeń (relacje, krotki, kolumny, perspektywy, operacje) do jednej z klas bezpieczeństwa  $\acute{S}T$ ,  $T$ ,  $P$  lub  $J$ . Będziemy określać **uprawnienia** (klasyfikację) podmiotu  $S$  jako **class(S)**, a **klasyfikację** przedmiotu  $O$  jako **class(O)**. Klasyfikacja pozwala na wprowadzenie dwóch ograniczeń:

- (1) Podmiot  $S$  może czytać przedmiot  $O$  tylko wtedy, gdy  $class(S) \geq class(O)$ .  
To ograniczenie jest nazywane **prostą własnością bezpieczeństwa**.
- (2) Podmiot  $S$  może zapisywać przedmiot  $O$  tylko wtedy, gdy  $class(S) \leq class(O)$ .  
To ograniczenie jest nazywane **własnością z gwiazdką** (lub po prostu **\*-własnością**).

Pierwsze ograniczenie jest intuicyjne i wymusza oczywistą zasadę, że nikt nie może odczytywać danych zaklasyfikowanych do wyższej klasy bezpieczeństwa niż ta przypisana czytającemu. Druga zasada jest mniej intuicyjna. Zabrania ona zapisywania (modyfikacji) danych o klasyfikacji niższej niż przypisana użytkownikowi. Pogwałcenie tej zasady prowadziłoby do przepływu informacji z wyższych do niższych poziomów bezpieczeństwa, co jest niezgodne z podstawowymi zasadami zabezpieczeń wielopoziomowych. Przykładowo, użytkownik przydzielony do klasy  $\acute{S}T$  mógłby stworzyć kopię informacji z klasyfikacją  $\acute{S}T$ , zapisując ją jako nowy obiekt klasy  $J$  i czyniąc ją tym samym widoczną dla wszystkich użytkowników systemu.

<sup>4</sup> Publikacja Bella i La Padulli (1976) to raport techniczny korporacji MITRE dotyczący bezpiecznych systemów komputerowych używających systemu Multics.

W celu wprowadzenia zasad zabezpieczeń wielopoziomowych do relacyjnych baz danych zazwyczaj traktuje się wartości atrybutów i krotki jako przedmioty mechanizmu bezpieczeństwa. Każdy atrybut  $A$  jest zatem skojarzony w schemacie z **atrybutem klasyfikacji**  $C$  i każdy atrybut każdej krotki jest przydzielony do jednej z klas bezpieczeństwa. Dodatkowo, w niektórych modelach dodatkowy **atrybut bezpieczeństwa**  $TC$  jest nadawany również całej krotce. Opisywane tu podejście to *model wielopoziomowy*, ponieważ umożliwia klasyfikowanie obiektów za pomocą wielu poziomów bezpieczeństwa. Schemat **wielopoziomowej relacji**  $R$  z  $n$  atrybutami wyglądałby następująco:

$$R(A_1, C_1, A_2, C_2, \dots, A_n, C_n, TC)$$

gdzie każda wartość  $C_i$  przedstawia *atrybut klasyfikacji* związany z atrybutem  $A_i$ .

Wartość atrybutu  $TC$  dla każdej krotki  $k$  — będąca *najwyższą* wartością z wszystkich wartości atrybutów klasyfikacji  $TC$  z wszystkich krotek  $k$  — określa ogólną klasyfikację całej krotki, podczas gdy  $C_i$  określają dokładniejszą klasyfikację dla każdego z atrybutów. Wartością  $TC$  każdej krotki  $k$  jest *najwyższa* z wszystkich wartości atrybutów klasyfikacji  $C_i$  z  $k$ .

**Rzeczywistym kluczem** (ang. *apparent key*) dla relacji wielopoziomowej jest zestaw atrybutów, który tworzy klucz główny w zwykłej (jednopoziomowej) relacji. Relacja wielopoziomowa będzie zawierała inne dane w zależności od poziomów bezpieczeństwa przeglądających ją podmiotów (użytkowników). W niektórych przypadkach możliwe jest zapisanie krotki w relacji z wyższą klasą bezpieczeństwa i stworzenie odpowiednich krotek w niższej klasie. Taką operację nazywamy **filtrowaniem**. Zazwyczaj konieczne jest zapisywanie dwóch lub więcej krotek z różnych klas bezpieczeństwa z tym samym *rzeczywistym kluczem*. Prowadzi to do pojęcia **wielopostaciowości** (ang. *polyinstantiation*)<sup>5</sup>, gdzie kilka krotek może mieć tę samą wartość klucza rzeczywistego, ale różne wartości atrybutów dla użytkowników na różnych poziomach bezpieczeństwa.

Zilustrujemy wprowadzone pojęcia prostym przykładem relacji wielopoziomowej przedstawionej na rysunku 30.2(a), na którym wartość atrybutów klasyfikacji jest pokazana obok wartości atrybutów krotek. Przypuśćmy, że atrybut *Nazwisko* jest kluczem rzeczywistym, i rozważmy zapytanie `SELECT * FROM PRACOWNIK`. Użytkownik z klasy bezpieczeństwa  $T$  otrzyma tę samą relację, która jest widoczna na rysunku 30.2(a), ponieważ klasy wszystkich krotek są niższe lub równe  $T$ . Użytkownik z poziomu  $P$  nie będzie jednak mógł odczytać wartości atrybutu *Pensja* dla krotki *Nowak* oraz *WydaźnośćPracy* dla *Kowalski*, ponieważ są przypisane do wyższych poziomów bezpieczeństwa. Krotki zostaną **odfiltrowane** do postaci zaprezentowanej na rysunku 30.2(b), gdzie *Pensja* i *WydaźnośćPracy* mają wartości *null*. Dla użytkownika przypisanego do klasy bezpieczeństwa  $J$  filtrowanie krotki *Kowalski* pozwoli jedynie na odczytanie atrybutu *Nazwisko*, a pozostałe będą wyświetlone jako *null* (rysunek 30.2(c)). Filtrowanie wprowadza więc wartość *null* w miejsce atrybutów, których poziom bezpieczeństwa był wyższy niż poziom bezpieczeństwa użytkownika.

<sup>5</sup> Pojęcie to jest zbliżone do sytuacji, gdzie kilka wersji w bazie danych odnosi się do tego samego rzeczywistego obiektu.

## (a) PRACOWNIK

Nazwisko	Pensja		WydajnośćPracy		TC
Kowalski J	3000	P	Srednia	T	T
Nowak P	5000	T	Dobra	P	T

## (b) PRACOWNIK

Nazwisko	Pensja		WydajnośćPracy		TC
Kowalski J	3000	P	NULL	P	P
Nowak P	NULL	P	Dobra	P	P

## (c) PRACOWNIK

Nazwisko	Pensja		WydajnośćPracy		TC
Kowalski J	NULL	J	NULL	J	J

## (d) PRACOWNIK

Nazwisko	Pensja		WydajnośćPracy		TC
Kowalski J	3000	P	Srednia	T	T
Kowalski J	3000	P	Bardzo dobra	P	P
Nowak P	5000	T	Dobra	P	T

RYСУNEK 30.2. Wielopoziomowa relacja ilustrująca zabezpieczenia wielopoziomowe.

- (a) Oryginalne krotki PRACOWNIK. (b) Perspektywa relacji po przefiltrowaniu dla użytkowników na poziomie P. (c) Perspektywa relacji po przefiltrowaniu dla użytkowników na poziomie J. (d) Wielopostaciowość krotki Kowalski

Ogólnie, zasada **integralności encji** dla relacji wielopoziomowych mówi, że wszystkie atrybuty będące elementami klucza rzeczywistego muszą być niepuste i muszą mieć *tę samą* klasę bezpieczeństwa w każdej krotce. Ponadto wszystkie pozostałe wartości atrybutów w krotce muszą być przypisane do wyższej lub tej samej klasy bezpieczeństwa co atrybuty klucza. Ograniczenie to wymusza, aby użytkownik mógł odczytać klucz krotki, jeśli ma uprawnienia do odczytu któregośkolwiek z jej elementów. Inne zasady integralności, czyli **integralność wartości null** i **integralność międzypostaciowa**, polegają na tym, że jeśli wartość krotki widoczna na niższym poziomie bezpieczeństwa może być odfiltrowana z oryginalnej krotki na wyższym poziomie bezpieczeństwa, to w relacji wielopoziomowej wystarczy przechowywać krotkę o wyższej klasie bezpieczeństwa.

Aby dokładniej zilustrować pojęcie wielopostaciowości, przypuśćmy, że użytkownik o poziomie bezpieczeństwa *P* próbuje zmienić wartość atrybutu *WydajnośćPracy* dla Kowalski z rysunku 30.2 na 'BardzoDobra'. Odpowiada to następującemu zapytaniu SQL:

```
UPDATE PRACOWNIK
SET WydajnośćPracy = 'BardzoDobra'
WHERE Nazwisko = 'Kowalski';
```

Ponieważ perspektywa prezentowana użytkownikom z klasy bezpieczeństwa  $P$  (patrz rysunek 30.2(b)) pozwala na taką zmianę, system nie powinien jej odrzucić. Inaczej użytkownik *domyśliłby się*, że dla krotki *Kowalski* istnieje jakaś niepusta wartość w miejscu wyświetlanej wartości *null*. Jest to przykład uzyskiwania informacji poprzez **kanał ukryty**, co nie powinno być możliwe w systemach o wysokim stopniu zabezpieczeń (patrz punkt 30.6.1). Użytkownik jednak nie może zmienić istniejącej wartości atrybutu *Wyda\_jność* ↪ *Pracy*, która jest sklasyfikowana do wyższego poziomu bezpieczeństwa. Rozwiązaniem jest **wielopostaciowość**, czyli stworzenie nowej instancji krotki *Kowalski* na poziomie bezpieczeństwa  $P$  (tak jak to pokazano na rysunku 30.2(d)). Jest to konieczne, ponieważ nowa krotka nie może być odfiltrowywana na poziomie  $T$ .

Aby obsługiwać powyższą i podobne sytuacje, trzeba zmodyfikować działanie podstawowych operacji aktualizujących dane w modelu relacyjnym (INSERT, DELETE, UPDATE). Wykracza to jednak poza zakres tej książki. Zainteresowanego czytelnika odsyłamy do bibliografii zamieszczonej na końcu rozdziału.

### 30.3.1. Porównanie dyspozycyjnego i obowiązkowego modelu bezpieczeństwa

Dyspozycyjna kontrola dostępu charakteryzuje się wysokim poziomem elastyczności, co czyni ją użyteczną w wielu różnych zastosowaniach. Podstawową wadą modelu dyspozycyjnego jest jego podatność na ataki takie jak konie trojańskie zaszyte w kodzie aplikacji. Powodem tego jest fakt, że model ten nie kontroluje rozpowszechniania ani używania danych po uzyskaniu do nich dostępu przez użytkownika. Z drugiej strony, obowiązkowa kontrola dostępu zapewnia wysoki stopień zabezpieczenia poprzez zablokowanie niepożądanego przepływu danych. Jest więc odpowiednia do aplikacji wojskowych, które wymagają wysokiego stopnia zabezpieczeń. Jednak mechanizmy zabezpieczeń obowiązkowych są zbyt sztywne, gdyż wymagają przydzielania wszystkich podmiotów i przedmiotów działania bazy danych do określonych klas bezpieczeństwa. Ich zastosowanie jest przez to ograniczone do niewielu środowisk. W wielu rzeczywistych sytuacjach preferowany jest model dyspozycyjny ze względu na jego lepszy stosunek bezpieczeństwa do użyteczności.

### 30.3.2. Kontrola dostępu oparta na rolach

Kontrola dostępu oparta na rolach pojawiła się w latach dziewięćdziesiątych jako skuteczna technologia do zarządzania bezpieczeństwem w dużych systemach korporacyjnych. Jej podstawowym założeniem jest przypisanie uprawnień określonym **rolom** (a nie poszczególnym użytkownikom) i powiązanie użytkowników z rolami. Role mogą być tworzone i usuwane za pomocą komend CREATE ROLE i DESTROY ROLE. Uprawnienia można nadawać i odbierać rolom (a także, w razie potrzeby, pojedynczym użytkownikom) za pomocą omawianych w poprzednim rozdziale 30.2 komend GRANT i REVOKE. Przykładowo, firma może utworzyć role dla menedżera produktu, kupca, asystenta ds. obsługi korespondencji, menedżera ds. obsługi klienta itd. Do każdej roli można przypisać wiele osób. Uprawnienia wspólne dla roli są przypisywane do nazwy danej roli i nadawane automatycznie wszystkim osobom przypisanym do tej roli.

Kontrolę dostępu opartą na rolach można stosować razem z tradycyjnymi technikami dyspozycyjnymi i obowiązkowymi. Gwarantuje ona, że tylko uwierzytelnieni użytkownicy o właściwych rolach otrzymają dostęp do określonych danych lub zasobów. Użyt-



kownicy nawiązują sesje, w ramach których mogą aktywować podzbiór przypisanych im ról. Z każdą sesją może być powiązanych wiele ról, ale tylko jeden użytkownik. Wiele SZBD obsługuje role, do których można przypisywać uprawnienia.

Podział zadań to następny ważny wymóg w różnych popularnych SZBD. Taki podział jest potrzebny, aby zapobiec wykonywaniu przez jednego użytkownika operacji wymagających udziału przynajmniej dwóch osób (ma to zapobiegać oszustwom). Jedną z metod podziału zadań jest wzajemne wykluczanie się ról. Dwie role **wzajemnie się wykluczają**, jeśli nie mogą być jednocześnie wykorzystywane przez tego samego użytkownika. **Wzajemne wykluczanie się ról** można podzielić na dwie kategorie: *statyczne (na etapie uwierzytelniania)* i *dynamiczne (na etapie wykonywania)*. W podejściu statycznym dwie role, które wzajemnie się wykluczają, nie mogą jednocześnie należeć do użytkownika w momencie jego uwierzytelniania. W podejściu dynamicznym użytkownik może mieć wykluczające się role na etapie uwierzytelniania, ale nie mogą one być jednocześnie aktywne. Inną odmianą modelu wzajemnego wykluczania się ról jest podejście z wyklucaniem całkowitym i częściowym.

**Hierarchia ról** jest naturalnym sposobem organizowania ról w celu odzwierciedlenia łańcuchów odpowiedzialności i władzy w organizacji. Standardowo, pomniejsze role u dołu są połączone z coraz istotniejszymi rolami w miarę przesuwania się w górę hierarchii. Diagramy hierarchii wprowadzają częściowy porządek, czyli relacje w hierarchii są zwrotne, przechodnie i niesymetryczne. Oznacza to, że jeśli użytkownik ma daną rolę, automatycznie otrzymuje role z niższych poziomów hierarchii. Definiowanie hierarchii obejmuje wybór jej rodzaju i ról oraz późniejsze zaimplementowanie hierarchii w wyniku przypisania ról innym rolom. Hierarchię ról można zaimplementować tak:

```
GRANT ROLE pracownik_etatowy TO typ_pracownika1  
GRANT ROLE stażysta TO typ_pracownika2
```

Ten przykład ilustruje, jak przypisać role `pracownik_etatowy` i `stażysta` do dwóch typów pracowników.

Inną kwestią związaną z bezpieczeństwem jest **zarządzanie tożsamością**. **Tożsamość** określa unikatową „nazwę” danej osoby. Ponieważ prawne nazwy ludzi nie zawsze są unikatowe, tożsamość musi obejmować wystarczające dodatkowe informacje, aby pełna nazwa była unikatowa. Uwierzytelnianie tożsamości i obsługa ich schematu to **zarządzanie tożsamością**. Dziedzina ta pozwala organizacjom skutecznie uwierzytelniać osoby i zarządzać oferowanym im dostępem do poufnych informacji. Te funkcje są widoczne w wymogach biznesowych we wszystkich branżach i organizacjach każdej wielkości. Administratorzy zarządzania tożsamością muszą stale dbać o spełnienie wymagań właścicieli aplikacji, a jednocześnie kontrolować koszt i zwiększać wydajność rozwiązań informatycznych.

Inną istotną cechą w systemach opartych na rolach jest możliwość wprowadzenia czasowych ograniczeń, takich jak czas i okres trwania aktywacji ról oraz uruchamianie roli na pewien okres poprzez aktywację innej roli. Używanie modelu wykorzystującego role jest istotnym elementem rozwiązującym kluczowe problemy związane z bezpieczeństwem aplikacji opartych na WWW. Role mogą być też przypisane kolejnym etapom ciągu prac w taki sposób, że użytkownik przypisany do jednej z ról związanych z określonym typem pracy może uczestniczyć tylko w czynnościach związanych z określonym etapem.



Model oparty na rolach posiada wiele użytecznych cech takich jak elastyczność, neutralność polityk bezpieczeństwa, lepsza obsługa zarządzania bezpieczeństwem i administracji, naturalne wymuszanie hierarchicznej struktury w organizacjach oraz inne, powodujące, że jest to dobry model do opracowywania bezpiecznych aplikacji WWW. Dla odmiany, modelom dyspozycyjnym i obowiązkowym brakuje niektórych tych możliwości. Poza tym, model oparty na rolach może realizować tradycyjne polityki systemów dyspozycyjnych i obowiązkowych. Ponadto, model ten zapewnia w naturalny sposób obsługę problemów związanych z wykonywaniem zadań i definiowaniem procedur pracy, a także pozwala tworzyć polityki definiowane przez użytkowników i specyficzne dla organizacji. Łatwiejsze zarządzanie użytkownikami korzystającymi z internetu stało się kolejną przyczyną powodzenia zabezpieczeń opartych na rolach.

### 30.3.3. Zabezpieczenia oparte na etykietach i kontrola dostępu na poziomie wierszy

W wielu popularnych relacyjnych SZBD stosuje się obecnie kontrolę dostępu na poziomie wierszy. Pozwala ona wprowadzać zaawansowane reguły kontroli dostępu na podstawie analizy danych wiersz po wierszu. W tym podejściu każdemu wierszowi danych przypisywana jest etykieta, która przechowuje informacje na temat wrażliwości danych. Kontrola dostępu na poziomie wierszy pozwala na tworzenie bardziej szczegółowych zabezpieczeń, ponieważ uprawnienia mogą być ustawiane dla każdego wiersza, a nie tylko dla tabeli lub kolumny. Początkowo użytkownik otrzymuje od administratora bazy danych domyślną etykietę sesji. Używane poziomy odpowiadają hierarchii poziomów wrażliwości danych na ujawnienie lub uszkodzenie. Celem jest zachowanie prywatności lub bezpieczeństwa. Etykiety mają zapobiegać temu, by nieuwierzytelnieni użytkownicy wyświetlali lub modyfikowali określone dane. Użytkownik o niskim poziomie uwierzytelnienia, zwykle reprezentowanym przez niską wartość, nie ma dostępu do danych z wyższych poziomów. Jeśli wiersz nie ma przypisanej etykiety, jest ona mu przypisywana automatycznie zgodnie z etykietą sesji użytkownika.

Polityka zdefiniowana przez administratora to **polityka zabezpieczeń opartych na etykietach**. Gdy aplikacja uzyskuje dostęp do danych objętych polityką lub zgłasza zapytanie o nie, polityka jest stosowana automatycznie. Jeśli używana jest polityka, do każdego wiersza w schemacie dodawana jest nowa kolumna. Dla każdego wiersza znajduje się w niej etykieta określająca wrażliwość danego wiersza w używanej polityce. Podobnie jak w obowiązkowej kontroli dostępu, gdzie każdy użytkownik ma ustalone uprawnienia, tu każda osoba posiada tożsamość. Jest ona porównywana z przypisaną każdemu wierszowi etykietą, aby stwierdzić, czy użytkownik może zobaczyć zawartość danego wiersza. Użytkownik może jednak samodzielnie zapisać wartość etykiety — w ramach pewnych ograniczeń i zgodnie z wytycznymi dla tego konkretnego wiersza. Etykiecie można przypisać wartość pomiędzy aktualną etykietą sesji użytkownika a poziomem minimalnym. Administrator bazy danych ma uprawnienia do przypisania początkowej domyślnej etykiety wiersza.

Wymagania w modelu zabezpieczeń opartych na etykietach są uzupełnieniem wymagań dyspozycyjnej kontroli dostępu. Dlatego aby użytkownik uzyskał dostęp do wiersza, musi spełniać wymogi dyspozycyjnej kontroli dostępu, a dodatkowo zabezpieczeń opar-

tych na etykietach. Wymogi dyspozycyjnej kontroli dostępu gwarantują, że użytkownik jest uwierzytelniony do wykonania danej operacji na schemacie. W większości aplikacji zabezpieczenia oparte na etykietach są potrzebne tylko dla niektórych tabel. Dla większości tabel aplikacji wystarczająca jest ochrona zapewniana przez dyspozycyjną kontrolę dostępu.

Polityki zabezpieczeń są zwykle tworzone przez menedżerów i pracowników działu zasobów ludzkich. Takie polityki są wysokopoziomowe, neutralne ze względu na technologię i powiązane z zagrożeniami. Powstają na bazie instrukcji od zarządu, który oczekuje określenia użytecznych, rozsądnych i korzystnych procedur organizacyjnych, wytycznych oraz działań. Z politykami zwykle powiązane są definicje kar i środków zapobiegawczych stosowanych w sytuacji naruszenia zasad. Polityki są następnie interpretowane i przekształcane na polityki uwzględniające etykiety przez **administratora etykiet zabezpieczeń**. Taki administrator definiuje etykiety zabezpieczeń dla danych i poziomy uwierzytelniania dla użytkowników. Te etykiety i poziomy wyznaczają dostęp do określonych chronionych obiektów.

Załóżmy, że użytkownik posiada uprawnienia do wykonywania operacji `SELECT` na tabeli. Gdy użytkownik wykona taką operację, zabezpieczenia oparte na etykietach automatycznie sprawdzą każdy wiersz zwracany przez zapytanie, aby ustalić, czy użytkownik ma uprawnienia do wyświetlenia danych. Przykładowo, jeśli poziom użytkownika to 20, może on zobaczyć wszystkie wiersze z poziomu 20 lub niższego. Ten poziom określa wrażliwość informacji z wiersza. Im bardziej wrażliwy wiersz, tym wyższa wartość jego etykiety. Takie zabezpieczenia można skonfigurować także na potrzeby testów bezpieczeństwa związanych z instrukcjami `UPDATE`, `DELETE` i `INSERT`.

### 30.3.4. Kontrola dostępu dla danych w formacie XML

Ponieważ XML jest używany w aplikacjach komercyjnych i naukowych na całym świecie, trwają prace nad wprowadzeniem w nim standardów bezpieczeństwa. Wyniki tych prac to m.in. podpisy cyfrowe i standardy szyfrowania dla plików XML. W specyfikacji *Signature Syntax and Processing* opisano składnię języka XML służącą do reprezentowania powiązań między podpisami kryptograficznymi a dokumentami XML lub innymi zasobami elektronicznymi. Ta specyfikacja obejmuje też procedury do obliczania i sprawdzania podpisów dokumentów XML. Protokół tworzenia podpisów cyfrowych dokumentów XML różni się od innych protokołów podpisywania wiadomości, takich jak OpenPGP (ang. *Pretty Good Privacy*; jest to usługa zapewniania poufności i uwierzytelniania, którą można stosować do poczty elektronicznej i aplikacji przechowujących pliki), ponieważ umożliwia podpisywanie tylko wybranych fragmentów drzewa dokumentu XML (patrz rozdział 13.) zamiast całych dokumentów. Ponadto specyfikacja podpisów dokumentów XML definiuje mechanizmy dodawania kontrasygnat i przekształcania (do postaci kanonicznej), gwarantujące, że dwa wystąpienia tego samego tekstu dadzą ten sam skrót używany do generowania podpisu, nawet jeśli ich reprezentacje nieco się różnią (np. rodzajem odstępów).

Specyfikacja XML Encryption Syntax and Processing definiuje leksykon i reguły przetwarzania pozwalające chronić poufność całych dokumentów XML oraz ich fragmentów, a także danych w innych formatach. Zaszyfrowana treść i dodatkowe informacje o przetwarzaniu przeznaczone dla odbiorcy są reprezentowane w dobrze uformowanym ko-

dzie w języku XML. Dlatego można je przetwarzać za pomocą narzędzi dla formatu XML. W odróżnieniu od innych popularnych technologii zapewniania poufności, takich jak SSL (ang. *Secure Sockets Layer*; najpopularniejszy protokół bezpieczeństwa w internecie) i wirtualne sieci prywatne, szyfrowanie w formacie XML dotyczy też fragmentów dokumentów i danych w pamięci trwałej. Systemy baz danych takie jak PostgreSQL lub Oracle obsługują obiekty w formacie JSON (ang. *JavaScript Object Notation*) jako format danych i udostępniają dla nich podobne narzędzia jak te opisane wcześniej dla formatu XML.

### 30.3.5. Polityki kontroli dostępu dla aplikacji sieciowych i mobilnych

Środowiska z publicznie dostępnymi aplikacjami sieciowymi stanowią wyjątkowe wyzwanie w zakresie zabezpieczania baz danych. Niektóre takie systemy odpowiadają za poufne lub prywatne informacje. Do tych rozwiązań należą sieci społecznościowe, serwery API aplikacji mobilnych i platformy transakcyjne sklepów internetowych.

W środowiskach do obsługi handlu elektronicznego (ang. *e-commerce*) transakcje są przeprowadzane elektronicznie. Wymaga to zaawansowanych polityk kontroli dostępu wykraczających poza tradycyjne SZBD. W środowiskach tradycyjnych baz danych kontrola dostępu jest zwykle oparta na zestawie uwierzytelnień podanych przez pracowników ds. bezpieczeństwa lub użytkowników zgodnie z politykami bezpieczeństwa. Ten prosty model nie nadaje się do dynamicznych środowisk spotykanych w handlu elektronicznym. Ponadto w takich środowiskach ochrony potrzebują nie tylko tradycyjne dane, ale też wiedza i doświadczenie. Takie cechy szczególne wymagają więcej swobody w zakresie tworzenia polityk kontroli dostępu. Mechanizmy kontroli dostępu muszą być wystarczająco elastyczne, by obsługiwać szeroki wachlarz niejednorodnych obiektów żądających ochrony.

Ponieważ wiele systemów obsługi rezerwacji, biletów, płatności i zakupów przetwarza informacje chronione prawnie, do ochrony takich informacji niezbędna jest architektura zabezpieczeń wykraczająca poza prostą kontrolę baz danych. Gdy nieupoważniona osoba uzyska dostęp do chronionych informacji, oznacza to wyciek danych, co ma poważne konsekwencje prawne i finansowe. Tą nieupoważnioną osobą może być pracownik konkurencji, która aktywnie stara się wykraść chronione informacje. Może to być też osoba z wewnątrz, która przekroczyła swoje uprawnienia lub w niewłaściwy sposób udostępniła chronione informacje innym. Przykładowo, nieodpowiednie zarządzanie danymi z kart kredytowych doprowadziło do poważnych wycieków danych w dużych sklepach.

W środowiskach tradycyjnych baz danych kontrola dostępu jest zwykle oparta na zestawie uwierzytelnień określanych przez pracowników ds. bezpieczeństwa. Jednak w aplikacjach sieciowych bardzo często zdarza się, że użytkownikiem jest sama aplikacja, a nie odpowiednio uwierzytelniona osoba. Może to spowodować, że mechanizmy kontroli dostępu w SZBD zostaną ominięte, a baza danych stanie się zwykłym relacyjnym magazynem danych dla systemu. W takich środowiskach zagrożenia typu wstrzykiwanie kodu w języku SQL (co opiszemy szczegółowo w podrozdziale 30.4) stają się dużo bardziej niebezpieczne, ponieważ mogą skutkować wyciekiem wszystkich danych i nie są ograniczone do informacji, do których dostęp umożliwia określone konto.

Aby chronić takie systemy przed wyciekiem danych, pierwszym wymogiem jest rozbudowana polityka zabezpieczania informacji, wykraczająca poza techniczne mechanizmy kontroli dostępu znane z popularnych SZBD. Taka polityka musi chronić nie tylko zwykłe dane, ale też procesy, wiedzę i doświadczenie.

Drugim, powiązanim wymogiem jest **kontrola dostępu oparta na treści**. Pozwala ona stworzyć polityki kontroli dostępu, które uwzględniają treść chronionych obiektów. Aby umożliwić kontrolę dostępu opartą na treści, polityka musi pozwalać na tworzenie warunków na podstawie treści obiektów.

Trzeci wymóg dotyczy niejednorodności użytkowników. Wymaga to polityk kontroli dostępu opartych na cechach i kategoriach użytkowników, a nie na pojedynczych konkretnych danych (np. identyfikatorach użytkowników). Możliwym rozwiązaniem, pozwalającym lepiej uwzględnić profile użytkowników w tworzeniu polityk kontroli dostępu, są **dane uwierzytelniające** (ang. *credentials*). Stanowią one zestaw dotyczących użytkownika cech (np.: wiek, stanowisko, rola w organizacji) istotnych ze względu na bezpieczeństwo. Przykładowo, wykorzystując dane uwierzytelniające, można formułować polityki takie jak *tylko pracownicy etatowi zatrudnieni od przynajmniej pięciu lat mają dostęp do dokumentów dotyczących wewnętrznych mechanizmów systemu*.

Oczekuje się, że XML będzie odgrywał ważną rolę w kontroli dostępu w sklepach internetowych<sup>6</sup>, ponieważ staje się standardowym językiem reprezentacji danych w wymianie dokumentów w sieci WWW. XML staje się też językiem handlu elektronicznego. Dlatego, z jednej strony, trzeba zapewnić bezpieczeństwo dokumentów XML, tworząc dostosowane do tego mechanizmy kontroli dostępu. Z drugiej strony, informacje o kontroli dostępu (polityki kontroli dostępu i dane uwierzytelniające użytkowników) można zapisywać za pomocą samego języka XML. Język **DSML** (ang. *Directory Services Markup Language*) pozwala zapisać informacje o usługach z katalogu za pomocą składni języka XML. Jest to podstawa do utworzenia standardu komunikowania się z usługami z katalogu na potrzeby podawania i sprawdzania danych uwierzytelniających użytkowników. Jednolitą prezentację chronionych obiektów i polityk kontroli dostępu można stosować też na potrzeby samych polityk i danych uwierzytelniających. Przykładowo, niektóre elementy danych uwierzytelniających (np. nazwisko użytkownika) mogą być dostępne dla każdego, natomiast inne — tylko dla użytkowników z określonej grupy. Ponadto stosowanie opartego na XML-u języka do podawania danych uwierzytelniających i polityk kontroli dostępu pozwala ułatwić bezpieczne przesyłanie danych uwierzytelniających i eksportowanie polityk.

## 30.4. Wstrzykiwanie kodu w języku SQL

Wstrzykiwanie kodu w języku SQL to jedno z najczęstszych zagrożeń w systemach baz danych. Omówimy je szczegółowo w tym podrozdziale. Oto wybrane inne często przeprowadzane ataki na bazy danych:

---

<sup>6</sup> Patrz Thuraisingham i in. (2001).

- **Niedozwolone podniesienie uprawnień.** Ten atak polega na tym, że napastnik próbuje podnieść swoje uprawnienia, wykorzystując słabe punkty w systemach baz danych.
- **Nadużycie uprawnień.** Niedozwolone podniesienie uprawnień dotyczy nieuwierzytelnionego użytkownika, natomiast ten atak jest przeprowadzany przez użytkownika z uprawnieniami. Przykładowo, administrator, który może modyfikować informacje o studentach, może wykorzystać je do zmiany ocen studenta bez pozwolenia wykładowcy.
- **Odmowa usług. Atak przez odmowę usług** (ang. *denial of service* — DoS) to próba uniemożliwienia dostępu do zasobów ich docelowym użytkownikom. Jest to ogólna kategoria ataków, w której dostęp do aplikacji sieciowych lub danych jest blokowany w wyniku przepełnienia bufora lub zużycia zasobów.
- **Słaby system uwierzytelniania.** Jeśli system uwierzytelniania jest słaby, napastnik może podać się za uprawnionego użytkownika, zdobywając dane uwierzytelniające tego ostatniego.

### 30.4.1. Metody wstrzykiwania kodu w języku SQL

W rozdziale 11. wyjaśniliśmy, że programy i aplikacje sieciowe korzystające z bazy danych mogą przysyłać do niej dane i polecenia, a także wyświetlać dane pobrane z bazy w przeglądarce. W **ataku przez wstrzyknięcie kodu w języku SQL** napastnik wstrzykuje za pomocą aplikacji tekstowe dane wejściowe, które zmieniają instrukcję w języku SQL zgodnie z celami napastnika. Taki atak może powodować różnorakie szkody — np. w wyniku niedozwolonej modyfikacji bazy lub pobrania poufnych danych. Może też posłużyć do wykonania instrukcji z poziomu systemu, które spowodują odmowę usługi dla aplikacji. W tym podrozdziale omawiamy rodzaje ataków tego typu.

**Modyfikowanie kodu w języku SQL.** Atak przez modyfikację kodu (jest to najczęstszy rodzaj ataków ze wstrzykiwaniem) zmienia w aplikacji polecenie w języku SQL — np. przez dodanie warunków do klauzuli `WHERE` zapytania lub przez rozbudowanie zapytania o dodatkowe komponenty z operacjami na zbiorach takimi jak `UNION`, `INTERSECT` lub `MINUS`. Możliwe są też inne rodzaje ataków z modyfikacją kodu. Typowy atak tego rodzaju ma miejsce w czasie logowania się do bazy. Załóżmy, że prosta procedura uwierzytelniania wywołuje poniższe zapytanie i sprawdza, czy zwrócone zostały jakieś wiersze:

```
SELECT * FROM users WHERE username = 'jacek' and PASSWORD = 'hasłojacka';
```

Napastnik może spróbować zmodyfikować instrukcję w języku SQL w następujący sposób:

```
SELECT * FROM users WHERE username = 'jacek' and (PASSWORD = 'hasłojacka' or  
↳ 'x' = 'x');
```

W ten sposób napastnik, który wie, że 'jacek' to poprawny login jednego z użytkowników, może zalogować się do systemu bazy danych na to konto, nie znając hasła. Może wtedy zrobić wszystko, do czego uprawniony jest Jacek.

**Wstrzykiwanie kodu.** Atak tego typu to próba dodania instrukcji w języku SQL do istniejącego polecenia w wyniku wykorzystania błędu spowodowanego przetwarzaniem nieprawidłowych danych. Napastnik może wstrzyknąć kod do programu, aby zmienić jego działanie.

Wstrzykiwanie kodu to popularna technika włamywania się do systemów w celu zdobycia informacji.

**Wstrzykiwanie wywołań funkcji.** W ataku tego rodzaju wywołanie funkcji bazy danych lub systemu operacyjnego jest wstawiane do podatnej na atak instrukcji w języku SQL, aby zmodyfikować dane lub uruchomić wywołanie systemowe z wysokimi uprawnieniami. Przykładowo, można uruchomić funkcję, która wykonuje jakieś działania związane z komunikacją w sieci. W zapytaniach w języku SQL można też wywoływać funkcje z niestandardowych pakietów baz danych (lub dowolne niestandardowe funkcje z bazy). Napastnik może wykorzystać przede wszystkim dynamicznie tworzone zapytania w języku SQL (patrz rozdział 10.), ponieważ są one tworzone w czasie wykonywania programu.

Przykładowo, tabela `dual` jest używana w klauzuli `FROM` w bazie Oracle, gdy użytkownik chce uruchomić kod w języku SQL niepowiązany logicznie z żadną tabelą. Aby pobrać dzisiejszą datę, można zastosować instrukcję:

```
SELECT SYSDATE FROM dual;
```

Następny przykład pokazuje, że nawet najprostsze instrukcje w języku SQL mogą być podatne na atak:

```
SELECT TRANSLATE('dane wyjściowe', 'źródłowy ciąg', 'docelowy ciąg') FROM dual;
```

Tu wywołanie `TRANSLATE` służy do zastępowania ciągu znaków innym ciągiem. Funkcja `TRANSLATE` zastępuje znaki 'źródłowy ciąg' znakami 'docelowy ciąg' jeden po drugim. To oznacza, że litera `ź` zostanie zastąpiona literą `d`, litera `r` literą `o`, litera `ó` literą `c` itd.

Tego rodzaju instrukcje w języku SQL mogą być podatne na atak przez wstrzyknięcie. Przyjrzyj się następującemu przykładowi:

```
SELECT TRANSLATE(" || UTL_HTTP.REQUEST ('http://129.107.2.1/') || ",  
                '98765432', '9876') FROM dual;
```

Użytkownik może wprowadzić ciąg `(" || UTL_HTTP.REQUEST ('http://129.107.2.1/') || ")`, gdzie `||` to operator łączenia, co powoduje zażądanie strony z serwera WWW. Pakiet `UTL_HTTP` generuje wywołania protokołu HTTP (ang. *Hypertext Transfer Protocol*) w języku SQL. Obiekt `REQUEST` przyjmuje jako parametr adres URL (tu jest to `'http://129.107.2.1/'`), komunikuje się z daną witryną i zwraca pobrane z niej dane (zwykle kod w języku HTML). Napastnik może sprepować wprowadzany łańcuch znaków i adres URL, aby dodać inne funkcje i wykonać niedozwolone operacje. Tu zastosowaliśmy fikcyjny przykład konwersji wartości `'98765432'` na `'9876'`, jednak celem napastnika może być dostęp do zasobu URL i uzyskanie poufnych informacji. Napastnik może następnie pobrać przydatne informacje z serwera bazy danych (zlokalizowanego pod adresem URL przekazanym jako parametr) i przesłać je na serwer WWW (wywołujący funkcję `TRANSLATE`).

## 30.4.2. Zagrożenia związane ze wstrzykiwaniem kodu w języku SQL

Wstrzykiwanie kodu w języku SQL jest szkodliwe, a dla napastników motywujące są możliwości, jakie daje im ta technika. Oto niektóre zagrożenia związane z atakami tego rodzaju:



- **Badanie cech bazy danych.** Napastnik może ustalić typ używanej na zapleczu bazy, dzięki czemu może przeprowadzić atak z wykorzystaniem słabych punktów typowych dla danego SZBD.
- **Odmowa usług.** Napastnik może zalać serwer żadaniami, blokując w ten sposób usługi dla uprawnionych użytkowników. Może też usunąć niektóre dane.
- **Pomijanie uwierzytelniania.** Jest to jedno z najczęstszych zagrożeń, pozwalające napastnikowi uzyskać dostęp do bazy danych z poziomu konta uwierzytelnionego użytkownika i wykonać wszystkie pożądane operacje.
- **Identyfikowanie możliwych do wstrzyknięcia parametrów.** W ataku tego rodzaju napastnik zbiera ważne informacje na temat typu i struktury bazy używanej na zapleczu aplikacji sieciowej. Ten atak jest możliwy dzięki temu, że domyślna strona błędu zwracana przez serwery aplikacji często zawiera za dużo informacji.
- **Zdalne wykonywanie instrukcji.** Ten atak zapewnia napastnikom narzędzie do wykonywania dowolnych operacji na bazie danych. Zdalny użytkownik może np. wykonywać procedury składowane i funkcje za pomocą zdalnego interaktywnego interfejsu z obsługą języka SQL.
- **Podwyższanie uprawnień.** W ataku tego rodzaju logiczne luki w bazie są wykorzystywane do podniesienia poziomu dostępu.

### 30.4.3. Techniki ochrony przed wstrzykiwaniem kodu w języku SQL

Ochronę przed wstrzykiwaniem kodu w języku SQL można uzyskać, stosując określone reguły programowania we wszystkich procedurach i funkcjach dostępnych z poziomu sieci WWW. W tym punkcie omawiamy niektóre z tych technik.

**Wiązanie zmiennych (za pomocą instrukcji sparametryzowanych).** Używanie zmiennych wiązanych (*parametrów*; patrz rozdział 10.) chroni przed atakami przez wstrzyknięcie, a dodatkowo poprawia wydajność.

Przyjrzyj się następującemu przykładowi wykorzystującemu Javę i JDBC:

```
PreparedStatement stmt = conn.prepareStatement( "SELECT * FROM  
    PRACOWNIK WHERE ID_PRACOWNIKA=? AND HASŁO=?");  
stmt.setString(1, id_pracownika);  
stmt.setString(2, hasło);
```

Zamiast umieszczać dane wejściowe od użytkownika w instrukcji, te dane należy powiązać z parametrem. W tym przykładzie wartości wejściowe 1 i 2 są przypisywane do zmiennych (wiązane z nimi) `employee_id` i `password`, zamiast być bezpośrednio przekazywane jako parametry w postaci ciągów znaków.

**Filtrowanie (sprawdzanie poprawności) danych wejściowych.** Tę technikę można wykorzystać do usuwania znaków ucieczki z wejściowych ciągów znaków za pomocą funkcji `Replace` języka SQL. Przykładowo, ogranicznik w postaci apostrofu (') można zastąpić cudzysłowem ("). Ta technika pozwala zapobiec niektórym atakom przez modyfikację kodu w języku SQL, ponieważ znaki ucieczki można wykorzystać do przeprowadzenia takich ataków. Jednak ponieważ znaków ucieczki jest wiele, technika ta nie jest niezawodna.



**Bezpieczeństwo funkcji.** Dostęp do funkcji (zarówno standardowych, jak i niestandardowych) z baz danych powinien być ograniczony, ponieważ można je wykorzystać do ataków przez wstrzyknięcie funkcji w języku SQL.

## 30.5. Wprowadzenie do bezpieczeństwa statystycznych baz danych

Statystyczne bazy danych służą głównie do opracowywania statystyk dotyczących różnych grup społecznych. Sama baza może przechowywać poufne dane dotyczące konkretnych osób, które powinny być chronione przed niepowołanym dostępem. Użytkownicy powinni jednak mieć możliwość odczytywania danych statystycznych takich jak średnie, sumy, liczności, wartości największe, najmniejsze i odchylenia. Omówienie opracowanych technologii pozwalających na ochronę informacji o pojedynczych osobach wykracza poza zakres niniejszej książki. Zilustrujemy jedynie problem za pomocą prostego przykładu odnoszącego się do relacji z rysunku 30.3. Jest to relacja OSOBA o atrybutach NAZWISKO, PESEL, DOCHÓD, ADRES, MIASTO, WOJEWÓDZTWO, KODPOCZTOWY, PŁEĆ i STOPIEŃ\_NAUKOWY.

OSOBA

NAZWISKO	PESEL	DOCHÓD	ADRES	MIASTO	WOJEWÓDZTWO	KODPOCZTOWY	PŁEĆ	STOPIEŃ_NAUKOWY
----------	-------	--------	-------	--------	-------------	-------------	------	-----------------

RYСУNEK 30.3. Schemat relacji OSOBA ilustrującej zabezpieczenia statystycznej bazy danych

**Populacja** jest zestawem krotek relacji (tabeli), które spełniają określone warunki wyboru. Każdy warunek wyboru na relacji OSOBA określa więc pewną populację. Na przykład warunek `PŁEĆ="M"` określa populację mężczyzn, warunek `((PŁEĆ="K") AND (STOPIEŃ_NAUKOWY="mgr" OR STOPIEŃ_NAUKOWY="dr"))` określa populację kobiet, których stopień naukowy to magister lub doktor, a warunek `MIASTO="Gliwice"` — populację osób mieszkających w Gliwicach.

Zapytania statystyczne polegają na zastosowaniu funkcji statystycznych do pewnej populacji, na przykład odczytaniu liczności populacji lub średniej pensji określonej grupy osób. Użytkownicy statystycznej bazy danych nie mogą jednak odczytywać informacji takich jak dochód konkretnych osób. Techniki **zabezpieczenia statystycznej bazy danych** muszą uniemożliwiać takie działania. Można to osiągnąć, zabraniając odczytu wartości poszczególnych atrybutów i dopuszczając jedynie zapytania używające statystycznych funkcji agregujących takich jak COUNT, SUM, MIN, MAX, AVERAGE i STANDARD DEVIATION. Takie zapytania nazywamy **zapytaniami statystycznymi**.

Odpowiedzialność za zapewnienie poufności danych o poszczególnych osobach spoczywa na systemie bazy danych, który równocześnie musi zapewniać użytkownikom dostęp do podsumowań statystycznych. Zapewnienie **ochrony prywatności** w statystycznych bazach danych jest najistotniejsze. Typowy przykład jej naruszenia prezentujemy poniżej.

W niektórych przypadkach możliwe jest **wywnioskowanie** wartości pojedynczych krotek z ciągu zapytań statystycznych, szczególnie gdy zapytania dotyczą niewielkiej liczby krotek. Rozważmy dwa następujące zapytania:

```
Z1: SELECT COUNT(*) FROM OSOBA
    WHERE <warunek>;
Z2: SELECT AVG(DOCHÓD) FROM OSOBA
    WHERE <warunek>;
```

Przypuśćmy teraz, że jesteśmy zainteresowani znalezieniem informacji o dochodach „Joanny Kowalskiej”; wiemy, że ma stopień doktora i że mieszka w Katowicach w województwie śląskim. Możemy więc wykonać zapytanie *Z1* z następującym warunkiem:

```
(STOPIEŃ_NAUKOWY="dr" AND PŁEĆ="K" AND MIASTO="Katowice" AND  
WOJEWÓDZTWO="śląskie")
```

Jeżeli otrzymamy dla takiego zapytania wynik równy 1, to możemy wykonać zapytanie *Z2* z tym samym warunkiem i w rezultacie otrzymać dochody Joanny Kowalskiej. Nawet w przypadku, kiedy *Z1* nie daje 1, tylko inną niewielką liczbę (na przykład 2 lub 3), możemy za pomocą zapytań statystycznych i funkcji *MAX*, *MIN* i *AVERAGE* określić zakres możliwych wartości dochodu dla poszukiwanej osoby.

Możliwość wynioskowania z zapytań statystycznych danych dotyczących pojedynczej osoby jest zmniejszone, jeśli system nie zezwala na zapytania, gdy liczność wybranej populacji jest niższa od pewnej ustalonej wartości granicznej. Inną techniką chroniącą dane pojedynczych osób jest ograniczenie możliwości wykonywania ciągu zapytań dotyczących tej samej populacji. Można również wprowadzić do wyników pewne drobne niedokładności czy celowy *szum* utrudniający odczytanie prywatnych informacji z danych statystycznych. Kolejną techniką jest podzielenie bazy danych. Wymaga się wówczas, aby dane były przechowywane w grupach krotek o wielkości większej od ustalonej wielkości minimalnej, a zapytania dotyczyły całych grup lub zestawów grup krotek. Zainteresowanych czytelników odsyłamy do bibliografii zamieszczonej na końcu rozdziału.

## 30.6. Wprowadzenie do kontroli przepływu

**Kontrola przepływu** reguluje obieg informacji pomiędzy obiektami bazy danych. Przepływ pomiędzy obiektem *X* a obiektem *Y* definiujemy jako odczytanie danych z obiektu *X* i zapisanie ich do *Y*. **Kontrola przepływu** polega na sprawdzaniu, czy informacja zawarta w niektórych obiektach nie przepływa jawnie ani niejawnie do obiektów słabiej chronionych. Dzięki temu użytkownik nie może pośrednio otrzymać w *Y* danych, do których nie ma bezpośredniego dostępu w *X*. Aktywną kontrolę przepływu zaczęto stosować na początku lat 70. Większość mechanizmów kontroli przepływu wykorzystuje klasy bezpieczeństwa; przepływ informacji od nadawcy do odbiorcy jest możliwy tylko wtedy, jeśli klasa bezpieczeństwa odbiorcy jest nie niższa niż klasa bezpieczeństwa nadawcy. Przykłady kontroli przepływu to zapobieganie udostępnianiu poufnych danych klientów przez program usługowy czy blokowanie przesyłania tajnych danych wojskowych do użytkownika o nieznanej klasyfikacji.

**Polityka przepływu** określa kanały, którymi można przysyłać informacje. Najprostsza forma polityki przepływu zakłada istnienie dwóch klas bezpieczeństwa: poufnej (*P*) oraz jawnej (*J*) i pozwala na każdy przepływ danych z wyjątkiem przepływu z klasy *P* do klasy *J*. Taka polityka może rozwiązywać problem izolacji danych, gdy program usługowy pracuje z takimi danymi jak informacje o klientach, gdzie niektóre wpisy mogą być poufne. Przykładowo, usługa obliczająca podatek dochodowy może wyświetlić dane adresowe i opłatę za skorzystanie z usługi, ale nie przychód i odliczenia.

Uwierzytelnianie użytkowników na potrzeby dostępu do zasobów jest realizowane za pośrednictwem mechanizmów kontroli dostępu, co oznacza, że wykonywane mogą być tylko dozwolone operacje. Kontrola przepływu może być realizowana za pomocą rozbudowanych mechanizmów kontroli dostępu, które przypisują każdemu wykonywanemu programowi jego klasę bezpieczeństwa (*uprawnienia*). Program może odczytać określony segment pamięci tylko wtedy, gdy jego klasa bezpieczeństwa nie jest niższa niż klasa przypisana segmentowi. Program może zapisywać dane do segmentu, jeżeli posiada klasę bezpieczeństwa nie wyższą niż klasa segmentu. Automatycznie ogranicza to możliwość przekazywania informacji przez użytkowników z wyższych poziomów bezpieczeństwa na niższe. Przykładowo, program mający przypisaną klasę bezpieczeństwa *tajne* może odczytywać wszystkie dane poufne i jawne, a wszystkie zapisywane przez niego dane będą *tajne* lub *ściśle tajne*.

Można wyróżnić dwa rodzaje przepływu danych: *jawny* — będący wynikiem instrukcji przypisania takich jak  $Y := f(X_1, \dots, X_n)$ ; oraz *niejawny* — wywoływany przez instrukcje warunkowe takie jak  $\text{if } f(X_{m+1}, \dots, X_n) \text{ then } y := f(X_1, \dots, X_m)$ .

Mechanizmy kontroli przepływu muszą zapewniać wykonywanie tylko uprawnionych przepływów danych (zarówno jawnych, jak i niejawnych). Bezpieczny przepływ informacji musi spełniać zestaw warunków. Warunki określające autoryzowane przepływy danych mogą być wyrażone przy użyciu relacji przepływów pomiędzy klasami i przypisane do informacji. (Przepływ informacji z *A* do *B* istnieje, gdy informacja powiązana z *A* wpływa na informację powiązaną z *B*. Przepływ wynika z operacji, które powodują przepływ informacji z jednego obiektu do drugiego). Relacje te mogą, dla każdej z klas, określać zestaw innych klas, do których dane mogą być przekazywane, lub mogą definiować relacje pomiędzy klasami opisujące kanały przepływu danych. Ogólnie mechanizmy kontroli przepływu działają poprzez przypisanie każdemu obiektowi etykiety określającej jego klasę bezpieczeństwa. Etykiety te mogą być później wykorzystane do weryfikacji relacji przepływu zdefiniowanych w modelu.

### 30.6.1. Ukryte kanały

Ukryte kanały pozwalają na przepływ informacji naruszający bezpieczeństwo lub ustaloną politykę. W szczególności, **kanał ukryty** pozwala na przepływanie informacji z obszaru o wyższym poziomie bezpieczeństwa do niższego. Ukryte kanały można ogólnie przypisać do dwóch kategorii: **kanały zapisu** i **kanały czasowe**. Różnią się one tym, że w **kanałach czasowych** informacja jest przenoszona przy synchronizacji pewnych zdarzeń i procesów, podczas gdy **kanały zapisu** nie wymagają synchronizacji, ponieważ dane są przenoszone za pośrednictwem informacji systemowych lub innych fragmentów systemu normalnie niedostępnych dla użytkownika.

Dla prostego przykładu kanału ukrytego rozważmy rozproszony system bazy danych, którego dwa węzły mają odpowiednio klasyfikację *tajny* (*T*) i *jawny* (*J*). Aby zrealizować transakcję, obydwa węzły muszą wyrazić na nią zgodę. Węzły mogą wzajemnie realizować tylko operacje zgodne z *\*-własnością*, która stanowi, że *T* nie może zapisywać ani przysyłać danych do *J*. Jeśli jednak węzły uzgodnią stworzenie pomiędzy sobą ukrytego kanału, to po rozpoczęciu przez *J* transakcji, *T* może przysyłać dane do *J* w pewien uzgodniony ukryty sposób, naruszając tym samym *\*-własność*. Można to osiągnąć, powtarzając

wywoływanie transakcji, która powoduje ukryte przesyłanie informacji do *J*. Przeciwdziałania takie jak blokowanie omówione w rozdziałach 21. i 22. ograniczają równoczesne zapisywanie informacji przez użytkowników o różnych poziomach bezpieczeństwa, likwidując tym samym ukryte kanały zapisu. Systemy operacyjne i rozproszone systemy baz danych zapewniają kontrolę nad wykonywaniem operacji wielowątkowych, pozwalającą na współdzielenie zasobów bez możliwości wkraczania jednego programu lub procesu w obszar pamięci lub inne zasoby systemu używane przez inny program, co pozwala na zlikwidowanie ukrytych kanałów czasowych. W dobrze zaprojektowanych dużych systemach ukryte kanały zwykle nie stanowią poważnego problemu. Nie wyklucza to jednak możliwości opracowania przez użytkowników sposobu na ukryte przesyłanie informacji.

Niektórzy eksperci zajmujący się bezpieczeństwem twierdzą, że sposobem na uniknięcie ukrytych kanałów jest nieudostępnianie programistom rzeczywistych danych po tym, jak oprogramowanie zostało wdrożone w środowisku produkcyjnym. Przykładowo, programista banku nie musi mieć dostępu do listy nazwisk i sald na rachunkach klientów, a programiści biura maklerskiego nie muszą znać rejestru zakupów i sprzedaży. Podczas testowania i tworzenia oprogramowania dostęp programistów do pewnych testowych danych jest uzasadniony, ale po zakończeniu implementacji — już nie.

## 30.7. Szyfrowanie i infrastruktura klucza publicznego

Przedstawione wcześniej metody kontroli dostępu i przepływu danych, pomimo niewątpliwych zalet, nie są w stanie ochronić bazy danych przed pewnymi zagrożeniami. Przypuśćmy, że po wysłaniu danych okazuje się, że zostały one przechwycone przez nieuprawnionego użytkownika. W takiej sytuacji stosując szyfrowanie możemy ukryć przesłane dane tak, że nawet pomimo przechwycenia transmisji nie zostaną one ujawnione. **Szyfrowanie** polega na przekształcaniu danych na postać **kryptogramu**, która nie jest łatwa do zrozumienia dla nieuprawnionych osób. Zwiększa to poziom bezpieczeństwa i prywatności w momencie złamania kontroli dostępu, ponieważ w sytuacji utraty lub kradzieży danych nieuprawniona osoba nie może łatwo zrozumieć zaszyfrowanych informacji.

Po tym wprowadzeniu podajemy standardowe definicje, których będziemy się trzymać<sup>7</sup>:

- *Kryptogram*: zaszyfrowane dane.
- *Tekst jawny*: zrozumiałe dane o określonym znaczeniu, które mogą zostać wczytane lub użyte bez deszyfrowania.
- *Szyfrowanie*: proces przekształcania tekstu jawnego na kryptogram.
- *Deszyfrowanie*: proces przekształcania kryptogramu na tekst jawny.

Szyfrowanie polega na zastosowaniu wobec danych **algorytmu szyfrującego** z pewnym wybranym **kluczem szyfrowania**. Aby odzyskać oryginalne dane, należy je **odszyfrować** za pomocą odpowiedniego **klucza deszyfrującego**.

---

<sup>7</sup> Za Departamentem Handlu Stanów Zjednoczonych.

### 30.7.1. Szyfry DES i AES

**DES** (ang. *Data Encryption Standard* — standard szyfrowania danych) jest systemem opracowanym przez rząd Stanów Zjednoczonych do użytku publicznego. Został ogólnie zaakceptowany jako standard kryptograficzny zarówno w USA, jak i w wielu innych krajach. DES zapewnia szyfrowanie kanału pomiędzy nadawcą A i odbiorcą B. Algorytm DES jest staranną i złożoną kombinacją dwóch podstawowych operacji kryptograficznych: podstawiania i permutacji (transpozycji). Siła algorytmu wynika z powtarzania wspomnianych operacji w 16 cyklach. Tekst jawny (oryginalna forma wiadomości) jest szyfrowany w blokach 64-bitowych. Co prawda klucz ma długość 64 bitów, ale w rzeczywistości może być dowolną liczbą 56-bitową. Po zakwestionowaniu skuteczności tego szyfru Narodowy Instytut Standardów (*NIST* — ang. *National Institute of Standards*) wprowadził zaawansowany standard szyfrowania (**AES** — ang. *Advanced Encryption Standard*). Ten algorytm, zamiast 56-bitowych, używa 128-bitowych bloków danych i może używać kluczy o długości 128, 192 i 256 bitów. AES wykorzystuje większą niż DES pulę kluczy, a więc jego złamanie zajmuje znacznie więcej czasu. W dzisiejszych systemach AES z długimi kluczami to domyślnie stosowany standard. Jest to także standard w produktach do pełnego szyfrowania dysków. Narzędzia FileVault Apple'a i BitLocker Microsoftu używają kluczy 256- lub 128-bitowych. TripleDES to technika rezerwowa, stosowana w starszych systemach, które nie obsługują nowoczesnych standardów szyfrowania.

### 30.7.2. Algorytmy z kluczem symetrycznym

Klucz symetryczny to jeden klucz używany zarówno do szyfrowania, jak i do deszyfrowania. Dzięki stosowaniu klucza symetrycznego możliwe jest rutynowe stosowanie szybkiego szyfrowania i deszyfrowania do wrażliwych danych z bazy. Komunikat zaszyfrowany za pomocą klucza może zostać odszyfrowany tylko przy użyciu tego samego klucza. Algorytmy używane do szyfrowania z kluczem symetrycznym to **algorytmy z kluczem tajnym**. Ponieważ takie algorytmy stosuje się przede wszystkim do szyfrowania treści komunikatów, można je też nazwać **algorytmami szyfrowania treści**.

Główną wadą algorytmów z kluczem tajnym jest konieczność przekazywania tego klucza. Możliwym rozwiązaniem jest generowanie klucza tajnego na podstawie hasła użytkownika i tej samej funkcji po stronie nadawcy oraz odbiorcy. Jest to *algorytm szyfrowania na podstawie hasła*. Siła szyfrowania z użyciem klucza symetrycznego zależy od długości tego klucza. Jeśli używany jest ten sam algorytm, dane zaszyfrowane za pomocą dłuższego klucza są trudniejsze do złamania niż wtedy, gdy stosowany jest krótszy klucz.

### 30.7.3. Szyfrowanie kluczem publicznym (asymetrycznym)

W 1976 roku Diffie i Hellman zaproponowali nowy typ kryptosystemu nazwany **szyfrowaniem kluczem publicznym**. Algorytmy klucza publicznego są oparte raczej na funkcjach matematycznych niż na operacjach na wzorach bitów. W tych algorytmach wyeliminowano jedną z wad szyfrowania kluczem symetrycznym, związaną z koniecznością bezpiecznej wymiany wspólnego klucza między nadawcą a odbiorcą. Systemy z kluczem publicznym do szyfrowania i deszyfrowania używają dwóch kluczy. *Klucz publiczny* może być przesyłany w niezabezpieczony sposób, natomiast *klucz prywatny* w ogóle nie jest przekazywany.

Takie rozwiązania, używające do wykonywania uzupełniających się operacji (szyfrowania i deszyfrowania) dwóch powiązanych kluczy (publicznego i prywatnego), to **algorytmy szyfrowania kluczem asymetrycznym**. Zastosowanie dwóch kluczy niesie ze sobą istotne konsekwencje w obszarach poufności, dystrybucji kluczy i uwierzytelniania. Dwa klucze używane w szyfrowaniu kluczem publicznym są nazywane **kluczem publicznym** i **kluczem prywatnym**. Klucz prywatny jest kluczem tajnym, ale nazywamy go *prywatnym* (a nie tajnym lub *sekretnym* jak w przypadku konwencjonalnego szyfrowania) dla uniknięcia mylenia z kluczem tradycyjnych szyfrów. Oba klucze są powiązane matematycznie, ponieważ jeden służy do szyfrowania, a drugi — do deszyfrowania. Jednak uzyskanie klucza prywatnego na podstawie klucza publicznego jest bardzo trudne.

Schemat (*infrastruktura*) szyfrowania z kluczem publicznym składa się z sześciu elementów:

- (1) **Tekst jawny** — dane lub wiadomość, która jest podawana algorytmowi jako dane wejściowe.
- (2) **Algorytm szyfrujący** — algorytm szyfrujący dokonuje różnych transformacji tekstu jawnego.
- (3) i (4). **Klucze prywatny i publiczny** — para kluczy dobranych tak, że jeden może być użyty do szyfrowania, a drugi do odszyfrowywania. Szczegóły transformacji dokonywanych przez algorytm szyfrujący zależą od klucza publicznego lub prywatnego dostarczonego jako jedna z danych wejściowych. Przykładowo, jeśli komunikat został zaszyfrowany za pomocą klucza publicznego, może zostać odszyfrowany tylko przy użyciu odpowiedniego klucza prywatnego.
- (5) **Kryptogram** — zaszyfrowana wiadomość dostarczona w wyniku działania algorytmu szyfrującego. Zależy od oryginalnego tekstu jawnego oraz od klucza. Dla danej wiadomości dwa różne klucze dadzą dwa różne kryptogramy.
- (6) **Algorytm deszyfrujący** — algorytm ten odczytuje kryptogram oraz odpowiedni klucz i przetwarza je w oryginalny tekst jawny.

Jak sugerują nazwy, klucz publiczny jest udostępniany publicznie innym użytkownikom, podczas gdy klucz prywatny pozostaje znany tylko jego właścicielowi. Algorytm kryptograficzny ogólnego zastosowania zależy od jednego klucza szyfrującego i innego (ale powiązanego z pierwszym) klucza deszyfrującego. Podstawowe kroki są następujące:

- (1) Każdy użytkownik tworzy parę kluczy używanych do szyfrowania i deszyfrowania wiadomości.
- (2) Każdy użytkownik umieszcza jeden z dwóch kluczy w publicznym rejestrze lub innym ogólnodostępnym pliku. Jest to klucz publiczny. Drugi klucz pozostaje prywatny.
- (3) Jeśli nadawca chce wysłać prywatną wiadomość do odbiorcy, to szyfruje wiadomość przy użyciu publicznego klucza odbiorcy.
- (4) Kiedy adresat otrzymuje wiadomość, deszyfruje ją za pomocą swojego klucza prywatnego. Nikt poza odbiorcą nie może odszyfrować wiadomości, ponieważ tylko odbiorca zna swój klucz prywatny.



**Algorytm klucza publicznego RSA.** Jeden z pierwszych schematów klucza publicznego został stworzony w 1978 roku przez Rona Rivesta, Adiego Shamira i Lena Adlemana w MIT<sup>8</sup> i został nazwany od inicjałów ich nazwisk **schematem RSA**. Zyskał on sobie od tamtego czasu uznanie i stał się najczęściej używanym szyfrem z kluczem publicznym. RSA wykorzystuje teorię liczb i zagadnienia rozkładu dużych liczb na czynniki pierwsze. Algorytm pracuje w arytmetyce modularnej mod  $n$ .

Do szyfrowania i deszyfrowywania używane są dwa klucze —  $d$  i  $e$ . Istotną własnością jest to, że mogą one być wzajemnie zamieniane.  $n$  jest wybraną dużą liczbą całkowitą, która jest iloczynem dwóch różnych dużych liczb pierwszych  $a$  i  $b$  ( $n = a \cdot b$ ). Klucz szyfrujący  $e$  jest losowo wybraną liczbą z przedziału 1 do  $n$ , która jest względnie pierwsza z  $(a-1) \cdot (b-1)$ . Blok tekstu jawnego  $P$  jest szyfrowany jako  $P^e = P \bmod n$ . Ponieważ potęgowanie jest wykonywane w grupie mod  $n$ , faktoryzacja  $P^e$  w celu odkrycia oryginalnej wiadomości jest problemem trudnym. Klucz deszyfrujący  $d$  jest tak dobrany, że  $(P^e)^d \bmod n = P$ . Można go obliczyć, korzystając z równania  $d \cdot e = 1 \bmod ((a-1) \cdot (b-1))$ . Uprawniony użytkownik będący w posiadaniu klucza  $d$  może zatem bez trudu rozszyfrować wiadomość, obliczając  $(P^e)^d \bmod n = P$ , bez potrzeby faktoryzacji  $P^e$ .

### 30.7.4. Podpis cyfrowy

Podpis cyfrowy jest przykładem zastosowania technologii kryptograficznych do zapewnienia uwierzytelniania w handlu elektronicznym. Podobnie jak odręczny, **podpis elektroniczny** (cyfrowy) pozwala na dołączenie do tekstu unikatowego dla konkretnej osoby znacznika. Znacznik powinien być trwały, co oznacza, że inne osoby powinny mieć możliwość późniejszego sprawdzenia, czy podpis rzeczywiście należy do nadawcy wiadomości.

Podpis cyfrowy składa się z ciągu symboli. Jeśli podpis cyfrowy danej osoby wyglądałby zawsze tak samo, to byłby bardzo prosty do podrobienia poprzez zwykłe skopiowanie ciągu znaków. Dlatego podpis cyfrowy musi być za każdym razem inny. Podpis będący funkcją podpisywanej wiadomości oraz aktualnego czasu spełnia to wymaganie. Aby podpis był unikatowy dla każdego użytkownika i równocześnie trudny do sfalszowania, musi zależeć również od pewnej tajnej liczby znanej tylko podpisującemu. Uogólniając, podpis cyfrowy musi zależeć od podpisywanej wiadomości oraz od unikatowej sekretnej liczby przypisanej użytkownikowi. Adresat podpisanej wiadomości nie musi znać żadnych tajnych liczb. Technologia klucza publicznego jest najlepszym rozwiązaniem przy tworzeniu cyfrowego podpisu opisanego powyżej.

### 30.7.5. Certyfikaty cyfrowe

Certyfikat cyfrowy służy do łączenia wartości klucza publicznego z tożsamością osoby lub usługi, która posiada klucz prywatny do cyfrowo podpisanego dokumentu. Certyfikaty są wydawane i podpisywane przez jednostki certyfikujące. Certyfikat dotyczy podmiotu otrzymującego go od takiej jednostki. Zamiast wymagać od wszystkich stron uwierzytelniania każdego użytkownika, stosuje się oparte na certyfikatach cyfrowych uwierzytelnianie przez niezależną jednostkę.

---

<sup>8</sup> Rivest i in. (1978).



Sam certyfikat cyfrowy zawiera informacje różnego rodzaju. Są to np. informacje o jednostce certyfikującej i właścicielu certyfikatu. Na poniższej liście wymienione są wszystkie dane ujęte w certyfikacie:

- (1) Informacje o właścicielu certyfikatu. Mają one postać unikatowego identyfikatora — nazwy wyróżniającej (ang. *distinguished name*) właściciela. Obejmuje ona nazwę właściciela, organizację i inne informacje.
- (2) Klucz publiczny właściciela.
- (3) Data wydania certyfikatu.
- (4) Okres ważności (podawany w występujących w każdym certyfikacie polach *Valid From* — czyli ważny od i *Valid To* — czyli ważny do).
- (5) Informacje o wydawcy certyfikatu.
- (6) Podpis cyfrowy jednostki certyfikującej. Wszystkie wymienione informacje są szyfrowane za pomocą funkcji tworzącej skrót wiadomości; funkcja ta generuje podpis cyfrowy. Ten podpis potwierdza, że powiązanie między właścicielem certyfikatu a kluczem publicznym jest poprawne.

## 30.8. Problemy z prywatnością i jej zachowywanie

Zachowanie prywatności danych staje się coraz większym wyzwaniem dla ekspertów od bezpieczeństwa i prywatności baz danych. Pod niektórymi względami w celu zachowania prywatności danych powinniśmy posunąć się do ograniczenia eksploracji i analiz danych na dużą skalę. Najczęściej stosowane techniki radzenia sobie z takimi problemami polegają na unikaniu budowania ogromnych centralnych hurtowni danych będących pojedynczym repozytorium z kluczowymi informacjami. Jest to jedna z przeszkód do zbudowania ogólnokrajowych rejestrów pacjentów cierpiących na różne poważne choroby. Inne rozwiązanie polega na celowym modyfikowaniu i zniekształcaniu danych.

Gdy wszystkie dane są dostępne w jednej hurtowni, naruszenie bezpieczeństwa tylko jednego repozytorium skutkuje ujawnieniem wszystkich informacji. Unikanie centralnych hurtowni i stosowanie rozproszonych algorytmów eksploracji danych minimalizuje jednak wymianę danych potrzebną do opracowania globalnie poprawnych modeli. Modyfikowanie, zniekształcanie i anonimizowanie danych pozwala dodatkowo ograniczyć związane z prywatnością zagrożenia w procesie eksploracji danych. Na potrzeby tych metod można usuwać informacje o tożsamości z ujawnionych danych i wprowadzać w danych szum. Jednak stosując te techniki, należy zwrócić uwagę na jakość wynikowych danych w bazie, ponieważ dane te mogą zostać zanadto zmodyfikowane. Konieczna jest umiejętność oszacowania błędów, jakie mogły się pojawić w wyniku tych modyfikacji.

Prywatność to ważny obszar badań w dziedzinie zarządzania bazami danych. Jest to skomplikowane zagadnienie z powodu międzydyscyplinarnego charakteru i problemów związanych z subiektywnością w interpretowaniu prywatności, zaufania itd. W ramach przykładu rozważ rejestry i transakcje medyczne oraz prawne, wymagające spełnienia określonych wymogów z zakresu prywatności. Coraz większą uwagę poświęca się też kontroli dostępu i prywatności w urządzeniach mobilnych. SZBD wymagają niezawodnych technik wydajnego składowania informacji związanych z bezpieczeństwem w ma-

łych urządzeniach, a także metod negocjowania zaufania. Ważnym problemem jest to, gdzie przechowywać informacje związane z tożsamością użytkowników, profilami, danymi uwierzytelniającymi i uprawnieniami, a także jak korzystać z tych danych do niezawodnego identyfikowania użytkowników. Ponieważ w środowiskach baz danych generowane są duże strumienie danych, trzeba zaprojektować wydajne techniki kontroli dostępu i zintegrować je z technikami przetwarzania, aby umożliwić stałe zgłaszanie zaapytań. Trzeba też zagwarantować prywatność danych o lokalizacji użytkowników, rejestrowanych za pomocą czujników i sieci telekomunikacyjnych.

## 30.9. Wyzwania związane z utrzymaniem bezpieczeństwa baz danych

Analizując szybki wzrost liczby zagrożeń dla baz danych i zasobów w postaci informacji, trzeba podjąć badania nad licznymi problemami, takimi jak: jakość danych, prawo własności intelektualnej i odporność baz danych. Pokróćce omówimy tu prace, jakie trzeba wykonać w kilku ważnych obszarach, którymi zajmują się badacze.

### 30.9.1. Jakość danych

Spółeczność użytkowników baz danych potrzebuje technik i rozwiązań organizacyjnych do oceny oraz potwierdzania jakości danych. Takie metody mogą obejmować proste mechanizmy, np. ikony jakości zamieszczane w witrynach WWW. Potrzebne są też rozwiązania umożliwiające skuteczniejsze sprawdzanie integralności semantycznej danych i narzędzia do oceny jakości danych oparte na technikach takich jak łączenie zapisów. Ponadto niezbędne są techniki odtwarzania danych na poziomie aplikacji do automatycznego poprawiania nieprawidłowych informacji. Z tymi zadaniami zmagają się obecnie producenci narzędzi ETL powszechnie stosowanych do wczytywania danych do hurtowni (patrz podrozdział 29.4).

### 30.9.2. Prawa własności intelektualnej

Wraz z rozpowszechnieniem się internetu i intranetów prawne oraz informacyjne aspekty danych stały się dla organizacji istotnym problemem. Aby sobie z nimi poradzić, zaproponowano techniki dodawania znaków wodnych (ang. *watermarking*) do danych relacyjnych. Głównym celem dodawania cyfrowych znaków wodnych jest ochrona treści przed nieuprawnionym powielaniem i dystrybucją. Odbywa się to dzięki umożliwieniu udowodnienia praw do treści. Tradycyjnie dodawanie cyfrowych znaków wodnych wymagało dziedziny z akceptowalnym szumem, w ramach której obiekt można było modyfikować, zachowując jego najważniejsze cechy. Potrzebne są jednak badania, aby ocenić niezawodność takich technik i przeanalizować różne podejścia do zapobiegania naruszaniu praw własności intelektualnej.

### 30.9.3. Odporność baz danych

Systemy baz danych muszą działać i wykonywać swoje funkcje nawet przy ograniczonych możliwościach oraz mimo destrukcyjnych zdarzeń takich jak ataki w ramach wojny informacyjnej. SZBD, obok zapobiegania w każdy możliwy sposób atakom i wykrywania ich, powinien sobie radzić z następującymi zadaniami:

- **Izolowanie.** Natychmiastowe podejmowanie działań, aby zablokować napastnikowi dostęp do systemu i odizolować problem w celu zapobieżenia jego rozprzestrzenianiu się.
- **Ocena uszkodzeń.** Określanie zakresu problemu, w tym niesprawnych funkcji i uszkodzonych danych.
- **Rekonfigurowanie.** Rekonfiguracja w celu umożliwienia pracy w trybie awaryjnym na czas odtwarzania bazy.
- **Naprawa.** Odtwarzanie uszkodzonych lub utraconych danych i naprawa lub ponowna instalacja niesprawnych funkcji systemu w celu wznowienia pracy w normalnym trybie.
- **Eliminowanie błędów.** W możliwym zakresie należy zidentyfikować słabe punkty wykorzystane w ataku i podjąć kroki, aby zapobiec jego powtórzeniu.

Celem napastnika w wojnie informacyjnej jest zakłócenie działalności organizacji, co osiąga się poprzez uszkodzenie systemów informatycznych. Obiektem ataku może być sam system lub dane. Choć ataki powodujące całkowite wyłączenie systemu są poważne i dramatyczne, muszą zostać przeprowadzone w odpowiednim momencie, jeśli napastnik chce zrealizować swoje zamiary. Takie ataki natychmiast powodują koncentrację uwagi i wysiłków nad przywróceniem pracy systemu, zdiagnozowaniem przebiegu ataku oraz wprowadzeniem środków zapobiegawczych.

Do tej pory zagadnienia związane z odpornością baz nie zostały w wystarczającym stopniu zbadane. Potrzebnych jest jeszcze wiele badań nad technikami i metodami gwarantującymi odporność systemów baz danych.

## 30.10. Zabezpieczenia oparte na etykietach w bazach Oracle

Ograniczenie dostępu do całej tabeli lub odizolowanie wrażliwych danych w odrębnych bazach to kosztowne operacje. **Zabezpieczenia oparte na etykietach w bazach Oracle** eliminują konieczność wykonywania takich operacji, ponieważ umożliwiają kontrolę dostępu na poziomie wierszy. Ten mechanizm jest dostępny od wersji Oracle Database 11g Release 1 (11.1) Enterprise Edition. Z każdą tabelą lub perspektywą bazy danych można powiązać politykę bezpieczeństwa. Jest ona stosowana za każdym razem, gdy tabela lub perspektywa jest modyfikowana lub zgłaszane jest zapytanie o nią. Programiści mogą łatwo dodawać do aplikacji opartych na bazach Oracle mechanizmy kontroli dostępu wykorzystujące etykiety. Zabezpieczenia oparte na etykietach to umożliwiający modyfikację sposobu kontrolowania dostępu do wrażliwych danych. Etykiety są przypisywane zarówno do użytkowników, jak i do danych. W bazach Oracle te etykiety służą do zapewniania bezpieczeństwa.

### 30.10.1. Technologia wirtualnych prywatnych baz danych

**Wirtualne prywatne bazy danych** (ang. *virtual private databases* — VPD) to mechanizm baz Oracle z wersji Enterprise Edition, który pozwala dodawać predykaty do instrukcji użytkownika, aby w automatyczny sposób ograniczyć dostęp do danych i pozwolić na niego tylko określonym użytkownikom i aplikacjom. Technika VPD umożliwia wymuszaną przez serwer i precyzyjną kontrolę dostępu w celu tworzenia bezpiecznych aplikacji.

Mechanizm VPD zapewnia kontrolę dostępu na podstawie polityk. Wymuszają one kontrolę dostępu na poziomie obiektów lub wierszy. Mechanizm VPD udostępnia interfejs API pozwalający dodawać polityki bezpieczeństwa do tabel lub perspektyw bazy danych. Za pomocą PL/SQL, podstawowego języka programowania używanego w aplikacjach opartych na bazach Oracle, programiści i administratorzy zabezpieczeń mogą wprowadzać polityki bezpieczeństwa z użyciem procedur składowanych<sup>9</sup>. Polityki VPD pozwalają programistom usunąć mechanizmy zabezpieczania dostępu z aplikacji i scentralizować je w bazie danych Oracle.

Mechanizm VPD jest włączany w wyniku powiązania polityki bezpieczeństwa z tabelą, perspektywą lub aliasem. Administrator do wiązania funkcji realizujących polityki z obiektami baz danych używa pakietu DBMS\_RLS z języka PL/SQL. W momencie dostępu do obiektu powiązanego z polityką bezpieczeństwa uruchamiana jest funkcja realizująca daną politykę. Funkcja ta zwraca predykat: klauzulę WHERE, która jest dodawana do instrukcji w języku SQL, co powoduje *automatyczną* i *dynamiczną* modyfikację dostępu użytkownika do danych. Zabezpieczenia oparte na etykietach w bazach Oracle to technika wymuszania bezpieczeństwa na poziomie wierszy za pomocą polityk bezpieczeństwa.

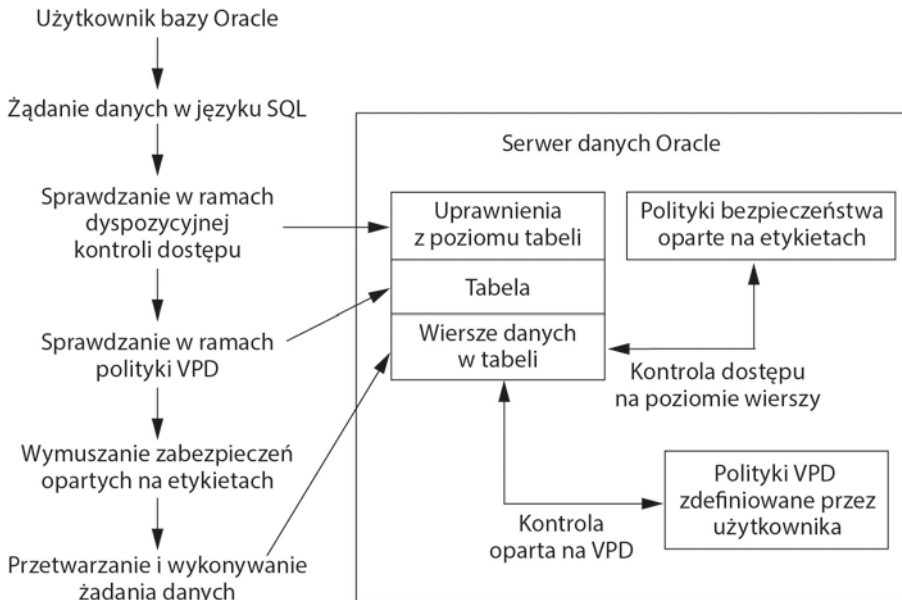
### 30.10.2. Architektura zabezpieczeń opartych na etykietach

Zabezpieczenia oparte na etykietach w bazach Oracle wykorzystują technologię VPD udostępnioną w produkcie Oracle Database 11.1 Enterprise Edition. Na rysunku 30.4 pokazano, jak wygląda dostęp do danych w tym modelu. Widoczna jest tam sekwencja operacji sprawdzania w ramach dyspozycyjnej kontroli dostępu i zabezpieczeń opartych na etykietach.

Na rysunku 30.4 przedstawiona jest sekwencja operacji sprawdzania w ramach dyspozycyjnej kontroli dostępu i zabezpieczeń opartych na etykietach. Po lewej stronie rysunku pokazany jest użytkownik aplikacji wysyłający w ramach sesji żądanie w języku SQL do bazy Oracle Database 11g Release 1 (11.1). SZBD Oracle sprawdza uprawnienia użytkownika w ramach dyspozycyjnej kontroli dostępu, upewniając się, że dana osoba ma uprawnienia SELECT do żądanej tabeli. Następnie sprawdza, czy z tą tabelą powiązana jest polityka VPD (pozwala to ustalić, czy tabela jest chroniona za pomocą zabezpieczeń opartych na etykietach). Jeśli tak jest, do pierwotnej instrukcji w języku SQL dodawana jest modyfikacja z mechanizmu VPD (klauzula WHERE), określająca zbiór wierszy dostępnych użytkownikowi. Następnie zabezpieczenia sprawdzają etykiety każdego wiersza w celu ustalenia podzbioru wierszy dostępnych użytkownikowi (wyjaśniono to w następnym punkcie). Tak zmodyfikowane zapytanie jest przetwarzane, optymalizowane i wykonywane.

---

<sup>9</sup> Procedury składowane są opisane w punkcie 8.2.2.



RYSUNEK 30.4. Architektura zabezpieczeń opartych na etykietach w bazach Oracle. Za: Oracle (2007)

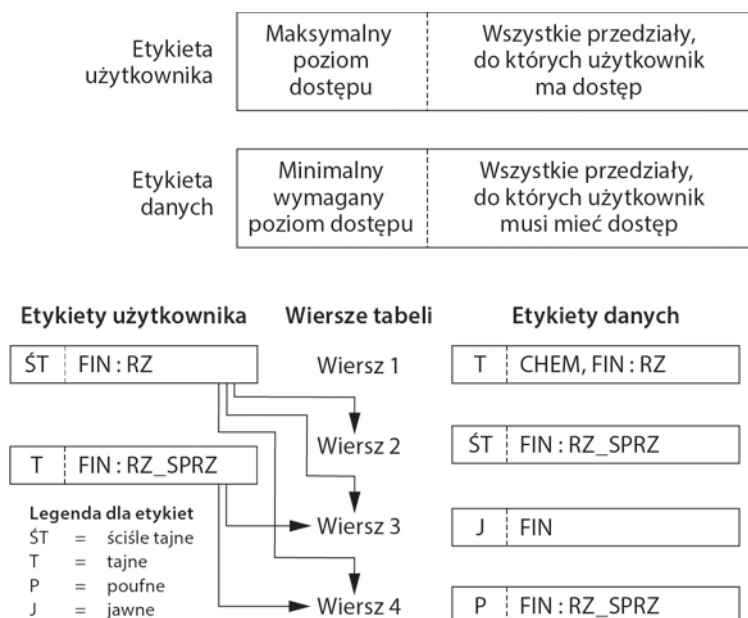
### 30.10.3. Współdziałanie etykiet danych i etykiet użytkowników

Etykieta użytkownika określa, do jakich informacji użytkownik ma uprawnienia dostępu. Wyznacza też typ tego dostępu (do odczytu lub do zapisu). Etykieta wiersza określa wrażliwość zawartych w nim informacji, a także ich właściciela. Gdy dostęp do tabeli w bazie jest zależny od etykiety, dostęp do wiersza można uzyskać tylko wtedy, jeśli etykieta użytkownika spełnia kryteria zdefiniowane w definicji polityki. Przyznanie lub odmowa dostępu wynika z porównania etykiety danych z etykietą sesji użytkownika.

Przedziały (ang. *compartments*) umożliwiają precyzyjniejszą klasyfikację wrażliwości danych z etykietami. Wszystkie dane powiązane z jednym projektem można przypisać do tego samego przedziału. Przedziały są opcjonalne. Etykieta może obejmować zero lub więcej przedziałów.

Grupy służą do identyfikowania organizacji jako właścicieli danych. Służą do tego odpowiednie etykiety grup. Grupy są hierarchiczne. Daną grupę można powiązać z grupą nadrzędną.

Jeśli najwyższy poziom użytkownika to **TAJNE**, ta osoba ma dostęp do wszystkich danych z poziomów **TAJNE**, **POUFNE** i **JAWNE**. Użytkownik nie ma wtedy dostępu do danych z poziomem **ŚCIŚLE\_TAJNE**. Na rysunku 30.5 pokazano, jak etykiety danych i etykiety użytkowników współdziałają w kontekście kontroli dostępu w ramach zabezpieczeń opartych na etykietach w bazach Oracle.



RYСУNEK 30.5. Etykiety danych i użytkowników w bazach Oracle. Za: Oracle (2007)

Na rysunku 30.5 użytkownik 1 ma dostęp do wierszy 2, 3 i 4, ponieważ jego maksymalny poziom to ŚT (ŚCIŚLE\_TAJNE). Użytkownik ma dostęp do przedziału FIN (finanse), a dostęp do grupy RZ (region zachodni) hierarchicznie zapewnia dostęp do grupy RZ\_SPRZ (region zachodni, sprzedaż). Użytkownik nie ma dostępu do wiersza 1, ponieważ nie ma uprawnień do przedziału CHEM (chemia). Aby użytkownik uzyskał dostęp do wiersza, musi mieć uprawnienia do wszystkich przedziałów podanych w etykiecie wiersza. Z tego przykładu wynika, że użytkownik 2 ma dostęp do wierszy 3 i 4, ponieważ jego maksymalny poziom to T (niższy niż ŚT z wiersza 2). Choć użytkownik 2 ma uprawnienia do przedziału FIN, może uzyskać dostęp tylko do grupy RZ\_SPRZ, przez co nie ma dostępu do wiersza 1.

## 30.11. Podsumowanie

W niniejszym rozdziale omówiono kilka technik zabezpieczających systemy baz danych. W podrozdziale 30.1 przedstawiono wprowadzenie do bezpieczeństwa baz danych. W podrozdziale 30.1.1 opisano różne zagrożenia dla baz danych (utrata integralności, dostępności i poufności). W podrozdziale 30.1.2 omówiono mechanizmy kontroli służące do radzenia sobie z tymi problemami: kontrolę dostępu, kontrolę wnioskowania, kontrolę przepływu i szyfrowanie. W dalszej części podrozdziału 30.1 przedstawiono różne zagadnienia z obszaru bezpieczeństwa, w tym wrażliwość danych, typy wycieków, zależności między bezpieczeństwem a precyzją wyników, gdy użytkownik żąda informacji, oraz zależności między bezpieczeństwem a prywatnością informacji.

Wprowadzenie zabezpieczeń wiąże się z kontrolowaniem dostępu do całego systemu bazy danych, a także nadzorowaniem dostępu do jego poszczególnych funkcji i obszarów. Pierwsza czynność jest zazwyczaj realizowana poprzez przypisywanie użytkownikom kont oraz haseł. Druga — poprzez wdrożenie systemu nadawanych i odbieranych poszczególnym użytkownikom uprawnień do używania poszczególnych fragmentów bazy danych. Taki sposób kontrolowania dostępu (opisany w podrozdziale 30.2) często jest nazywany dyspozycyjną kontrolą dostępu. Przedstawione zostały niektóre instrukcje języka SQL służące do nadawania i odbierania uprawnień, a także kilka ilustrujących ich użycie przykładów. W podrozdziale 30.3 przedstawiono również mechanizmy obowiązkowej kontroli dostępu wprowadzającej zabezpieczenia wielopoziomowe. Wymagają one przydzielenia wszystkich użytkowników oraz danych do poszczególnych klas bezpieczeństwa oraz wprowadzenia reguł uniemożliwiających przepływ informacji z wyższych poziomów bezpieczeństwa do niższych. Zaprezentowane zostały podstawowe pojęcia z zakresu zabezpieczeń wielopoziomowych w relacyjnych bazach danych, takie jak filtrowanie i wielopostaciowość. Została również przedstawiona kontrola dostępu oparta na rolach (punkt 30.3.2), która przypisuje użytkownikom różne uprawnienia w zależności od odgrywanej przez nich roli w systemie. Omówiono też hierarchie ról, wzajemne wykluczanie się ról oraz zabezpieczenia oparte na wierszach i etykietach. W podrozdziale 30.4 przedstawiono najważniejsze kwestie dotyczące wstrzykiwania kodu w języku SQL, metody, w których możliwe są takie ataki, i rozmaite zagrożenia powiązane z tą techniką. Dalej opisano różne sposoby zapobiegania takim atakom.

W podrozdziale 30.5 krótko omówiono problem kontrolowania dostępu do danych w statystycznych bazach danych, związany z ochroną danych osobowych indywidualnych osób przy jednoczesnym zapewnieniu dostępu do danych statystycznych związanych z populacją. W podrozdziale 30.6 omówiono kwestie związane z kontrolą przepływu i problemem ukrytych kanałów, a w podrozdziale 30.7 — szyfrowanie danych oraz infrastruktury klucza publicznego i prywatnego. W punkcie 30.7.3 objaśniono algorytmy klucza symetrycznego i korzystanie z popularnej infrastruktury klucza publicznego opartej na kluczu asymetrycznym. W punktach 30.7.4 i 30.7.5 opisano podpisy i certyfikaty cyfrowe. W podrozdziale 30.8 podkreślono znaczenie prywatności i wskazano kilka technik jej zapewniania. W podrozdziale 30.9 opisano różne wyzwania związane z bezpieczeństwem, w tym jakość danych, prawa własności intelektualnej i odporność danych. Na zakończenie rozdziału, w podrozdziale 30.10, przedstawiono implementowanie polityk bezpieczeństwa za pomocą połączenia bezpieczeństwa opartego na etykietach i technologii VPD w bazach Oracle 11g.

## Pytania powtórkowe

- 30.1. Omów następujące pojęcia: *autoryzacja bazy danych, kontrola dostępu, szyfrowanie danych, konto uprzywilejowane (systemowe), audyt bazy danych, raport audytu*.
- 30.2. Które konto jest wyznaczane jako właściciel relacji? Jakie uprawnienia ma takie konto?
- 30.3. Jak można zastosować mechanizm perspektyw w mechanizmach zabezpieczeń?
- 30.4. Omów rodzaje uprawnień z poziomu konta i z poziomu relacji.



- 30.5. Co rozumiemy przez nadanie uprawnień? Co rozumiemy przez cofnięcie uprawnień?
- 30.6. Omów system propagacji uprawnień i ograniczenia związane z limitami poziomej i pionowej propagacji.
- 30.7. Wymień typy uprawnień dostępne w SQL.
- 30.8. Jaka jest różnica pomiędzy *dyspozycyjną* i *obowiązkową* kontrolą dostępu?
- 30.9. Jakie są typowe klasy bezpieczeństwa? Omów prostą własność bezpieczeństwa i \*-własność. Wyjaśnij, dlaczego te własności są istotne w wielopoziomowym modelu zabezpieczeń.
- 30.10. Opisz wielopoziomowy relacyjny model danych. Zdefiniuj następujące pojęcia: *klucz rzeczywisty*, *wielopostaciowość*, *filtrowanie*.
- 30.11. Jakie są zalety zabezpieczeń dyspozycyjnych i obowiązkowych?
- 30.12. Czym jest kontrola dostępu oparta na rolach? Jakie są jej zalety w stosunku do schematu dyspozycyjnego i obowiązkowego?
- 30.13. Podaj dwa typy wzajemnego wykluczania w kontroli dostępu opartej na rolach.
- 30.14. Czym jest kontrola dostępu na poziomie wierszy?
- 30.15. Czym są zabezpieczenia oparte na etykietach? Jak administrator może je wymuszać?
- 30.16. Podaj różne typy ataków przez wstrzyknięcie kodu w języku SQL.
- 30.17. Jakie zagrożenia są związane z atakami przez wstrzyknięcie kodu w języku SQL?
- 30.18. Jakie środki można przedsięwziąć, aby zabezpieczyć się przed atakami przez wstrzyknięcie kodu w języku SQL?
- 30.19. Co to jest statystyczna baza danych? Omów specyfikę jej zabezpieczeń.
- 30.20. Jaki jest związek prywatności z bezpieczeństwem w statystycznej bazie danych? Jakie środki można zastosować do zapewnienia ochrony prywatności danych w tego typu bazach danych?
- 30.21. Czym jest kontrola przepływu w kontekście bezpieczeństwa? Jakie istnieją rodzaje przepływu danych?
- 30.22. Co nazywamy kanałami ukrytymi? Podaj przykład takiego kanału.
- 30.23. Jaki jest cel szyfrowania? Jakie procesy są powiązane z szyfrowaniem danych i z ich późniejszym odzyskiwaniem?
- 30.24. Podaj przykład algorytmu szyfrującego i omów jego działanie.
- 30.25. Powtórz poprzednie pytanie w odniesieniu do algorytmu RSA.
- 30.26. Czym jest algorytm klucza symetrycznego w zabezpieczeniach opartych na kluczu?
- 30.27. Co to jest infrastruktura klucza publicznego? W jaki sposób zapewnia bezpieczeństwo?
- 30.28. Co to jest podpis cyfrowy? W jaki sposób działa?
- 30.29. Jakie informacje obejmuje certyfikat cyfrowy?

## Ćwiczenia

- 30.30. Jak można zachować prywatność danych w bazie?
- 30.31. Wymień aktualne problemy w dziedzinie bezpieczeństwa baz danych.
- 30.32. Rozważ schemat relacyjnej bazy danych z rysunku 5.5. Przypuśćmy, że wszystkie relacje zostały stworzone przez użytkownika X (będącego również ich właścicielem), który następnie nadał następujące uprawnienia użytkownikom A, B, C, D i E:
- a) Użytkownik A może odczytywać lub modyfikować każdą relację z wyjątkiem CZŁONEK\_RODZINY i może przekazywać swoje uprawnienia innym.
  - b) Użytkownik B może odczytywać wszystkie atrybuty PRACOWNIK i DZIAŁ z wyjątkiem atrybutów PENSJA, PESELKIEROWNIKA i DATAPRZEJKIEROWNICTWA.
  - c) Użytkownik C może odczytywać i modyfikować tabelę PRACUJE\_NAD, ale tylko odczytywać atrybuty IMIĘ, INICJAŁDRIMIENIA, NAZWISKO i PESEL tabeli PRACOWNIK oraz NAZWAPROJ i NUMERPROJ z tabeli PROJEKT.
  - d) Użytkownik D może odczytywać wszystkie atrybuty PRACOWNIK i CZŁONEK\_RODZINY, a dodatkowo może modyfikować zawartość tabeli CZŁONEK\_RODZINY.
  - e) Użytkownik E może odczytywać dowolny atrybut tabeli PRACOWNIK, ale tylko dla krotek o wartości NRDZ=3.
  - f) Napisz wyrażenia SQL nadające te uprawnienia. Użyj perspektyw tam, gdzie to będzie potrzebne
- 30.33. Przypuśćmy, że uprawnienia (a) z ćwiczenia 30.32 są nadane z opcją GRANT, ale w taki sposób, że konto A może przekazać swoje uprawnienia najwyżej 5 innym użytkownikom i każdy z nich może przekazać uprawnienia dalej, ale już *bez* opcji GRANT. Jakie będą w tym przypadku ograniczenia propagacji poziomej i pionowej?
- 30.34. Rozważ relację z rysunku 30.2(d). Jak wyglądałaby dla użytkownika z klasy bezpieczeństwa J? Przypuśćmy, że taki użytkownik próbuje zmienić dochód Kowalskiego na 5000 zł; jaki byłby skutek takiego działania?

## Wybrane publikacje

Autoryzacja oparta na nadawaniu i odbieraniu uprawnień została zaproponowana w eksperymentalnym systemie baz danych SYSTEM R i jest opisana w pracy Griffithsa i Wade'a (1976). Kilka książek omawia ogólnie bezpieczeństwo w bazach danych i systemach komputerowych, w tym publikacje Leissa (1982a), Fernandez i in. (1981) oraz Fuginiego i in. (1995). Natan (2005) napisał praktyczną książkę na temat problemów z implementacją zabezpieczeń i audytów we wszystkich popularnych relacyjnych SZBD.

Wiele publikacji omawia różne techniki projektowania i ochrony statystycznych baz danych. Są wśród nich prace takich autorów jak McLeish (1989), Chin i Ozsoyoglu (1981), Leiss (1982), Wong (1984) i Denning (1980). Ghosh (1984) omawia zastosowanie statystycznych baz danych do kontroli jakości. Istnieje sporo publikacji dotyczących kryptografii i szyfrowania danych, w tym prace Diffiego i Hellmana (1979), Rivesta i in. (1978), Akla (1983), Pfleegera i Pfleeger (2007), Omury i in. (1990), Stallinga (2000) oraz Iyera i in. (2004).

Halfond i in. (2006) pomagają zrozumieć ataki przez wstrzykiwanie kodu w języku SQL i związane z nimi zagrożenia. Raport firmy Oracle (2007a) wyjaśnia, że bazy Oracle są mniej narażone na takie ataki niż bazy SQL Server. W tej samej publikacji pokrótce objaśniono też, jak zapobiegać takim atakom. Inne proponowane rozwiązania omówiono w pracach Boyda i Keromytisa (2004), Halfonda i Orso (2005) oraz McClure'a i Krügera (2005).

Bezpieczeństwo wielopoziomowe omawiali Jajodia i Sandhu (1991), Denning i in. (1987), Smith i Winslett (1992), Stachour i Thuraisingham (1990), Lunt i in. (1990) oraz Bertino i in. (2001). Przegląd kwestii badawczych w zakresie bezpieczeństwa baz danych prezentują Lunt i Fernandez (1990), Jajodia i Sandhu (1991), Bertino i in. (1998), Castano i in. (1995) oraz Thuraisingham i in. (2001). Wpływ zabezpieczeń wielopoziomowych na kontrolę współbieżności jest omówiony w pracy Atluriego i in. (1997). Bezpieczeństwo w bazach danych obiektowych i semantycznych oraz w systemach nowej generacji omówione jest przez Rabbitiego i in. (1991), Jajodię i Kogana (1990) oraz Smitha (1990). Oh (1999) prezentuje model bezpieczeństwa łączący schemat dyspozycyjny z obowiązkowym. Modele bezpieczeństwa dla aplikacji działających przez WWW oraz kontrolę dostępu opartą na rolach omawia Joshi i in. (2001). Kwestie bezpieczeństwa dla menadżerów w kontekście handlu elektronicznego i potrzeby szacowania ryzyka są omawiane przez Farahmunda i in. (2005). Kontrola dostępu na poziomie wierszy została szczegółowo omówiona w pracach firm Oracle (2007b) i Sybase (2005). W ostatniej z tych publikacji opisano też hierarchię ról i wzajemne wykluczanie. W Oracle (2009) wyjaśniono, jak zarządzanie tożsamością jest stosowane w bazach Oracle.

Nowe rozwiązania oraz przyszłe wyzwania z zakresu bezpieczeństwa i prywatności w bazach danych omówiono w Bertino i Sandhu (2005). Rząd USA (1978), OECD (1980) i NRC (2003) to wzorcowe źródła ilustrujące spojrzenie na prywatność przez ważne jednostki rządowe. Karat i in. (2009) opisują model polityk bezpieczeństwa i prywatności. XML i kontrolę dostępu opisano w pracy Naedelego (2003). Bardziej szczegółowe są prace na temat technik zapewniania prywatności (Vaidya i Clifton, 2004), praw własności intelektualnej (Sion i in., 2004) i odporności baz danych (Jajodia i in., 1999). Technologia VPD i zabezpieczenia oparte na etykietach z baz Oracle są szczegółowo omówione w Oracle (2007b).

Agrawal i in. (2002) zdefiniowali hipokratyczne bazy danych służące do zachowywania prywatności w informacjach z obszaru służby zdrowia. k-anonimowość to technika zapewniania prywatności omówiona w pracach Bayardo i Agrawala (2005) oraz Cirianiego i in. (2007). Oparte na k-anonimowości techniki eksploracji danych z zachowaniem prywatności zostały przeanalizowane w pracy Cirianiego i in. (2008). Vimercati i in. (2014) opisują szyfrowanie i fragmentację jako możliwe techniki zapewniania poufności danych w chmurze.

---

## Dodatki



# A

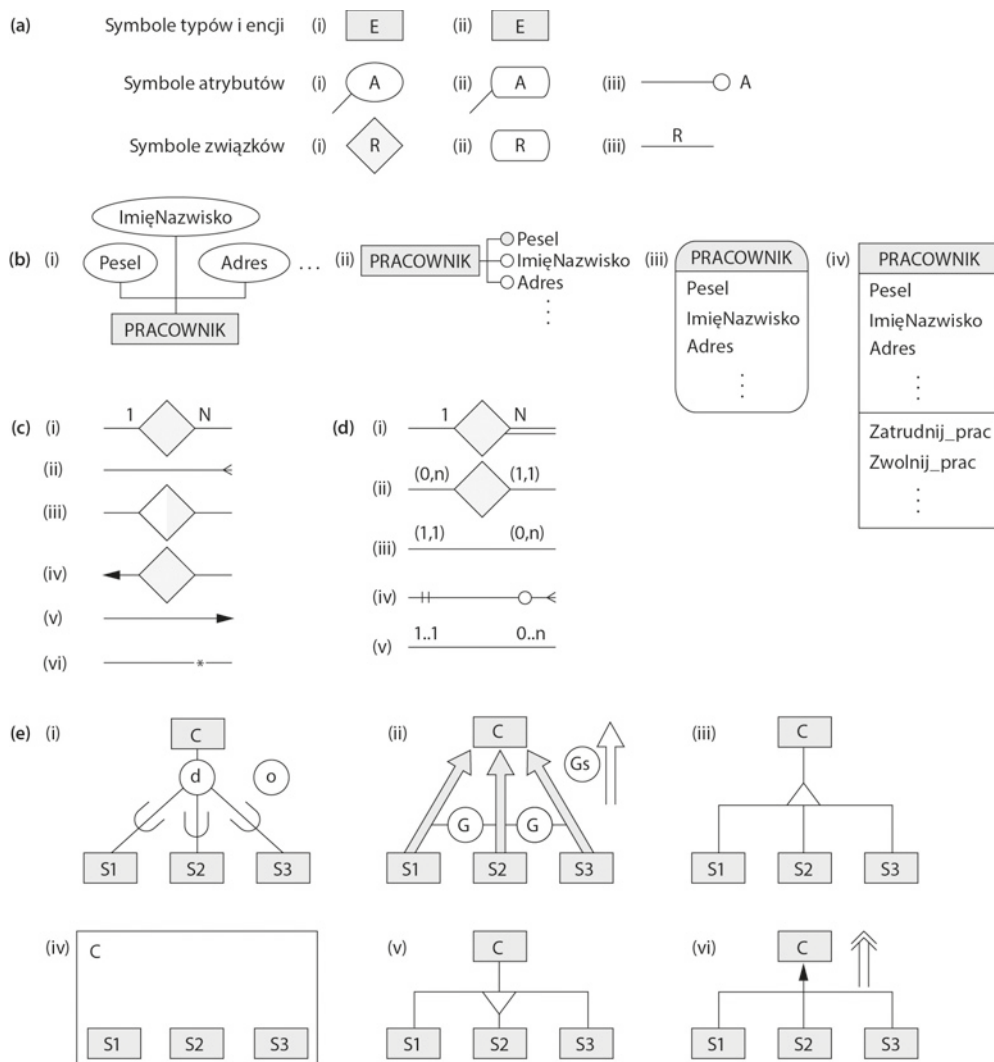
## Alternatywne notacje modeli związków encji

Na rysunku A.1 przedstawiono kilka różnych notacji diagramowych, służących do reprezentowania pojęć modeli ER i EER. Niestety, nie istnieje żadna standardowa notacja: różni projektanci baz danych stosują różne notacje. Podobnie, różne metodologie CASE (ang. computer-aided software engineering — inżynieria oprogramowania wspomagana komputerowo) oraz OOA (ang. object-oriented analysis — analiza obiektowa) wykorzystują różne notacje. Niektóre z nich są związane z modelami uwzględniającymi dodatkowe pojęcia i ograniczenia oprócz zdefiniowanych w ramach modeli ER i EER, które opisano w rozdziałach od 7. do 9., a inne modele oferują mniejszą liczbę takich pojęć i ograniczeń. Notacja, która została użyta w rozdziale 7., dość dokładnie oddaje koncepcje oryginalnej, często wykorzystywanej notacji ER. Poniżej zostaną omówione pewne alternatywne rozwiązania.

Na rysunku A.1(a) przedstawiono różne notacje służące do reprezentowania typów i klas encji, atrybutów oraz związków. W rozdziałach od 7. do 9. używano symboli oznaczonych na rysunku A.1(a) jako (i), czyli prostokąta, owalu i rombu. Symbole oznaczone jako (ii) są podobne, ale używa się ich w przypadku innych metodologii w celu reprezentowania trzech różnych pojęć. Symbol w postaci linii prostej (iii) reprezentujący związki jest używany przez kilka narzędzi i metodologii.

Na rysunku A.1(b) przedstawiono pewne notacje służące do łączenia atrybutów z typami encji. Wcześniej używaliśmy notacji (i). Notacja (ii) w celu reprezentowania atrybutów wykorzystuje trzecią formę (iii) z rysunku A.1(a). Ostatnie dwie notacje — (iii) i (iv) — są popularne w przypadku metodologii obiektowych i niektórych narzędzi CASE. Ostatnia forma notacji służy do reprezentowania zarówno atrybutów, jak i metod klas, oddzielonych poziomą linią.

Na rysunku A.1(c) przedstawiono różne notacje służące do reprezentowania współczynników liczności związków binarnych. W rozdziałach od 7. do 9. była wykorzystywana notacja (i). Notacja (ii) — znana jako notacja *kurzej stopki* — jest dość powszechnie stosowana. Notacja (iv) wykorzystuje strzałkę jako odwołanie funkcyjne (od N do 1) i przypomina wykorzystywaną przez nas notację kluczy obcych w modelu relacyjnym (patrz rysunek 9.2). Notacja (v) — używana w przypadku *diagramów Bachmana* — wykorzystuje strzałkę w *odwrotnym kierunku* (od 1 do N). W związkach typu 1:1 notacja (ii) wykorzystuje linię prostą bez „kurzej stopki”. Notacja (iii) stosuje metodę polegającą na połowicznym wypełnieniu rombu, zaś notacja (iv) umieszcza strzałki po obu stronach. W związkach typu M:N notacja (ii) wykorzystuje „kurze stopki” po obu stronach linii, notacja (iii) polega na zaciemnieniu obu połówek rombu, zaś notacja (iv) nie przedstawia żadnych strzałek.



RYSUNEK A.1. Alternatywne formy notacji. (a) Symbole typów i klas encji, atrybutów oraz związków. (b) Reprezentowanie atrybutów. (c) Reprezentowanie współczynników liczości. (d) Różne notacje dla wartości minimalnych i maksymalnych. (e) Notacje służące do reprezentowania specjalizacji i generalizacji

Na rysunku A.1(d) przedstawiono kilka odmian reprezentowania więzów wartości minimalnych i maksymalnych, które są używane w celu wyświetlania zarówno współczynników liczości, jak i poziomu uczestnictwa — całkowitego i częściowego. Najczęściej posługiwaliśmy się notacją (i). Notacja (ii) stanowi notację alternatywną, którą zastosowano na rysunku 7.15 i omówiono w podrozdziale 7.7.4. Warto przypomnieć, że stosowana notacja określała ograniczenie mówiące o tym, że każda encja musi występować w co najmniej min oraz najwyżej maks instancji związku. Stąd w przypadku związku typu 1:1 obie wartości maksymalne wynoszą 1, zaś w przypadku związku typu M:N obie wartości maksymalne wynoszą n. Wartość minimalna większa od 0 określa uczestnictwo



zupełne (zależność występowania). W przypadku metodologii wykorzystujących linię prostą w celu reprezentowania związków często stosuje się *odwrotne rozmieszczenie* więzów (min, maks), co przedstawiono w formie notacji (iii). Odmianą tego podejścia często stosowaną w niektórych narzędziach (i w UML-u) jest notacja (v). Kolejną popularną techniką — wykorzystującą to samo rozmieszczenie co w przypadku (iii) — jest reprezentowanie wartości *min* jako symbolu „o” (kółka, oznaczającego 0) lub jako symbolu „|” (pionowa kreska, oznaczająca 1), zaś wartości *max* jako symbolu „|” (pionowa kreska, oznaczająca 1) lub „kurzej stopki” (oznaczającej n), co przedstawiono w formie notacji (iv).

Na rysunku A.1(e) przedstawiono pewne notacje służące do reprezentowania specjalizacji i generalizacji. Notacja (i) była wykorzystywana w rozdziale 8., gdzie symbol d umieszczony w kółku oznacza, że podklasy (S1, S2 oraz S3) są rozłączne (ang. *disjoint*), zaś symbol o oznacza, że klasy nakładają się. Notacja (ii) wykorzystuje symbol G (od generalizacji) w celu określenia rozłączności oraz Gs w celu określenia nakładania. Pewne notacje wykorzystują strzałki z grotem zaciemnionym, a inne z grotem pustym (pokazanym obok). Notacja (iii) wykorzystuje symbol trójkąta wskazującego na nadklasę, zaś notacja (v) symbol trójkąta wskazującego na podklasy. Istnieje również możliwość wykorzystania obu notacji w ramach jednej metodologii, gdzie konwencja notacji (iii) określa generalizację, zaś konwencja notacji (v) — specjalizację. Notacja (iv) umieszcza prostokąty reprezentujące podklasy w prostokącie reprezentującym nadklasę. Spośród notacji opartych na zapisie (vi) niektóre wykorzystują strzałki z liniami pojedynczymi, a inne — strzałki z liniami podwójnymi (co pokazano obok).

Notacje przedstawione na rysunku A.1 ukazują tylko niektóre symbole diagramowe, których się używa lub proponowano używać w conceptualnych schematach baz danych. Używane są też inne notacje, jak również różne kombinacje wymienionych powyżej. Przydatną rzeczą byłoby określenie standardu, którego wszyscy musieliby się trzymać, co pozwoliłoby zapobiec nieporozumieniom i zmniejszyło panujący bałagan.



# B

## Parametry dysków

Najważniejszym parametrem określającym wydajność działania dysku jest czas potrzebny do zlokalizowania dowolnego bloku dyskowego przy danym jego adresie, a następnie przesłania bloku między dyskiem a pamięcią główną. Jest to **średni czas dostępu** (ang. *random access time*). Należy tu uwzględnić trzy składowe:

- (1) **Czas wyszukiwania ( $s$ )**. Jest to czas potrzebny do mechanicznego ustawienia głowicy odczytująco-zapisującej nad odpowiednią ścieżką w przypadku dysków z głowicą ruchomą (w przypadku dysków z głowicą stałą jest to czas potrzebny do elektronicznego przełączenia się na odpowiednią głowicę). W przypadku głowic ruchomych czas ten może być różny w zależności od odległości między bieżącą ścieżką znajdującą się pod głowicą a ścieżką określoną w adresie bloku. Zazwyczaj producenci dysków podają średni czas wyszukiwania w milisekundach. Typowe wartości tego parametru sięgają od 4 do 10 ms. Jest to główny *winny* opóźnienia związanego z przesyłaniem bloków między dyskiem a pamięcią.
- (2) **Opóźnienie rotacyjne ( $rd$ )**. Kiedy głowica odczytująco-zapisująca znajdzie się nad odpowiednią ścieżką, użytkownik musi poczekać, aż znajdzie się pod nią początek wymaganego bloku. W przeciętnym przypadku wymaga to czasu związanego z półobrotem dysku, ale rzeczywiste wartości sięgają od dostępu natychmiastowego (jeżeli od razu po zakończeniu wyszukiwania początek wymaganego bloku znajduje się na pozycji pod głowicą) po pełny obrót dysku (jeżeli początek wymaganego bloku przesunie się pod głowicą tuż po zakończeniu wyszukiwania). Jeżeli prędkość obrotowa dysku wynosi  $p$  obrotów na minutę (rpm), to średnie opóźnienie rotacyjne ( $rd$ ) wynosi

$$rd = (1/2) \cdot (1/p) \text{ min} = (60 \cdot 1000) / (2 \cdot p) \text{ ms} = 30000/p \text{ ms}$$

Typową wartością parametru  $p$  jest 10000 obr/min, co daje opóźnienie rotacyjne  $rd = 3$  ms. W przypadku dysków z głowicą stałą, gdzie czas wyszukiwania można zaniedbać, ta składowa powoduje największe opóźnienie pod względem transmisji bloku dyskowego.

- (3) **Czas transmisji bloku ( $btt$ )**. Kiedy głowica odczytująco-zapisująca znajdzie się nad początkiem wymaganego bloku, potrzebny jest pewien okres czasu w celu przesłania danych z bloku. Ów czas transmisji bloku zależy od rozmiaru bloku, rozmiaru ścieżki oraz prędkości obrotowej. Jeżeli **szybkość transmisji** dla dysku wynosi  $tr$  bajtów/ms, zaś rozmiar bloku wynosi  $B$  bajtów, to

$$btt = B/tr \text{ ms}$$

Jeżeli rozmiar ścieżki wynosi 50 kB, zaś  $p$  wynosi 3600 obr/min, to szybkość transmisji w bajtach/ms wynosi:

$$tr = (50 \cdot 1000) / (60 \cdot 1000 / 3600) = 3000 \text{ bajtów/ms}$$

W takim przypadku  $btt = B/3000$  ms, gdzie  $B$  jest rozmiarem bloku określonym w bajtach.

Średni czas wymagany do znalezienia i przesłania bloku przy danym jego adresie można oszacować jako:

$$(s + rd + btt) \text{ ms}$$

Jest tak zarówno w przypadku odczytu, jak i zapisu bloku. Główną metodą redukcji tego czasu jest przesłanie kilku bloków przechowywanych na jednej lub wielu ścieżkach tego samego cylindra. W takim przypadku czas wyszukiwania występuje tylko w przypadku pierwszego bloku. W celu przesłania  $k$  kolejnych *nieciągłych* bloków znajdujących na tym samym cylindrze potrzeba około

$$s + (k \cdot (rd + btt)) \text{ ms}$$

W takim przypadku potrzebne są dwa lub więcej buforów w pamięci głównej, ponieważ w sposób ciągły odczytujemy lub zapisujemy  $k$  bloków, co omówiono w rozdziale 17. Czas transmisji przypadający na jeden blok zostaje jeszcze bardziej zredukowany, kiedy przesyłane są *kolejne bloki* na tej samej ścieżce lub cylindrze. Eliminuje to opóźnienie rotacyjne dla wszystkich bloków oprócz pierwszego, więc oszacowanie czasu potrzebnego do przesłania  $k$  kolejnych bloków wynosi

$$s + rd + (k \cdot btt) \text{ ms}$$

Bardziej dokładne oszacowanie uwzględnia szczeliny międzyblokowe (patrz podrozdział 17.2.1), zawierające informacje, które pozwalają głowicy odczytująco-zapisującej na określenie, który blok ma właśnie być odczytywany. Zazwyczaj producenci dysków podają parametr **zbiorczej prędkości transmisji danych** (ang. *bulk transfer rate*, **btr**), który uwzględnia rozmiar szczeliny w przypadku odczytu kolejnych bloków. Jeżeli rozmiar ten wynosi  $G$  bajtów, to

$$btr = (B / (B + G)) \cdot tr \text{ bajtów/ms}$$

Zbiorcza prędkość transmisji danych określa poziom transmisji *użytecznych bajtów* z bloków danych. Głowica odczytująco-zapisująca musi przesunąć się nad wszystkimi danymi umieszczonymi na ścieżce przy obrocie dysku, w tym nad danymi związanymi ze szczelinami międzyblokowymi, które stanowią informacje sterujące, a nie rzeczywiste dane. W przypadku użycia tego parametru czas potrzebny na przesłanie użytecznych danych w ramach jednego bloku spośród kilku kolejnych bloków wynosi  $B/btr$ . Stąd szacowany czas odczytu  $k$  bloków przechowywanych obok siebie w ramach tego samego cylindra wynosi

$$s + rd + (k \cdot (B / btr)) \text{ ms}$$

Kolejnym parametrem jest **czas przepisania** (ang. *rewrite time*). Jest on przydatny w sytuacjach, gdy wczytujemy blok z dysku do bufora pamięci głównej, aktualizujemy bufor, a następnie zapisujemy bufor z powrotem do oryginalnego bloku dyskowego. W wielu przypadkach czas wymagany do zaktualizowania bufora w pamięci głównej jest krótszy od czasu związanego z jednym obrotem dysku. Jeżeli wiadomo, że bufor jest gotowy do

przepisania, system może pozostawić głowicę nad tą samą ścieżką i w czasie kolejnego obrotu dysku zaktualizowany bufor jest przepisywany z powrotem na dysk. Stąd czas przepisania  $T_{rw}$  zwykle określa się jako czas związany z pojedynczym obrotem dysku:

$$T_{rw} = 2 \cdot rd \text{ ms} = 60000/p \text{ ms}$$

Podsumowując, poniżej przedstawiono listę omówionych parametrów oraz używanych symboli:

czas wyszukiwania:	$s$ ms
opóźnienie rotacyjne:	$rd$ ms
czas transmisji bloku:	$btt$ ms
czas przepisania:	$T_{rw}$ ms
prędkość transmisji:	$tr$ bajtów/ms
zbiorcza prędkość transmisji:	$btr$ bajtów/ms
rozmiar bloku:	$B$ bajtów
rozmiar szczeliny międzyblokowej:	$G$ bajtów
szybkość dysku	$p$ rpm (obrotów na minutę)



# C

## Omówienie języka QBE

Język *QBE* (ang. *Query-By-Example*, zapytanie przez przykład) jest ważny dlatego, że jest jednym z pierwszych graficznych języków zapytań z minimalną składnią opracowanych dla potrzeb systemów bazodanowych. Został opracowany przez firmę IBM i jest dostępny jako produkt komercyjny stanowiący część opcji interfejsu *QMF* (ang. *Query Management Facility*) systemu bazodanowego DB2. Język ten został również zaimplementowany w SZBD Paradox i ma związek z interfejsem typu wskaż i naciśnij (ang. *point-and-click*) SZBD Access. Różni się on od języka SQL tym, że użytkownik nie musi bezpośrednio określać pełnego zapytania — zamiast tego, zapytanie jest formułowane poprzez wypełnienie **szablonów** relacji wyświetlanych na ekranie monitora. Na rysunku C.1 przedstawiono, jak mogą wyglądać takie szablony dla bazy danych z rysunku 5.6. Użytkownik nie musi pamiętać nazw atrybutów lub relacji, ponieważ są one wyświetlane jako część takich szablonów. Ponadto użytkownik nie musi trzymać się żadnych rygorystycznych reguł składniowych dotyczących określania zapytań; zamiast tego, w celu skonstruowania **przykładu** związanego z pobieraniem lub aktualizacją danych wpisuje stałe i zmienne w kolumnach szablonu. Język QBE, jak zostanie pokazane, jest związany z domenowym rachunkiem relacyjnym i w przypadku jego oryginalnej specyfikacji dowiedziono, że jest relacyjnie kompletny.

### PRACOWNIK

NAZWISKO	IMIĘ	INICJAŁDRIM	PESEL	DATAUR	ADRES	PŁEĆ	PENSJA	PESEL_PRZEŁ	NRDZ
----------	------	-------------	-------	--------	-------	------	--------	-------------	------

### DZIAŁ

NAZWADZ	NUMERDZ	PESELKIEROWNIKA	DATAPRZEJKIEROWNICTWA
---------	---------	-----------------	-----------------------

### LOKALIZACJE\_DZIAŁÓW

NUMERDZ	LOKALIZACJADZ
---------	---------------

### PROJEKT

NAZWAPROJ	NUMERPROJ	LOKALIZACJAPROJ	NR_DZ
-----------	-----------	-----------------	-------

### PRACUJE\_NAD

PESELPRAC	NR_PROJ	GODZINY
-----------	---------	---------

### CZŁONEK\_RODZINY

PESELPRAC	IMIĘ_CZŁONKA_RODZINY	PŁEĆ	DATAUR	STOPIEŃ_POKREWIEŃSTWA
-----------	----------------------	------	--------	-----------------------

RYСУNEK C.1. Schemat relacyjny z rysunku 5.6 w postaci reprezentowanej w języku QBE



## C.1. Podstawowe mechanizmy pobierania danych w języku QBE

W języku QBE zapytania pobierające dane określa się poprzez wypełnienie jednego lub wielu wierszy w szablonych tabel. W przypadku zapytania dotyczącego pojedynczej relacji wpisuje się stałe lub **elementy przykładowe** (jest to pojęcie z języka QBE) w kolumnach szablonu relacji. Element przykładowy oznacza zmienną domenową i jest określany jako przykładowa wartość poprzedzona znakiem podkreślenia (  ). Ponadto, przedrostek P. (określany mianem operatora „P kropka”) jest wstawiany w określonych kolumnach w celu wskazania, że chcemy wydrukować (lub wyświetlić) wartości z tych kolumn w wyniku. Stałe określają wartości, które muszą dokładnie w tej samej postaci występować w kolumnach.

Przykładowo, weźmy pod uwagę zapytanie Z0: *Pobierz datę urodzenia i adres Jana B. Nowaka*. Na rysunkach od C.2(a) do C.2(d) przedstawiono, w jaki sposób zapytanie to można określić w stopniowo coraz bardziej zwięzłej formie w języku QBE. Na rysunku C.2(a) przykład pracownika jest przedstawiany jako typ wiersza, którym jesteśmy zainteresowani. Pozostawiając imiona i nazwisko (Jan B. Nowak) jako stałe w kolumnach IMIĘ, INICJAŁDRIM oraz NAZWISKO, określamy w nich dokładne dopasowanie. Wszystkie pozostałe kolumny zostały poprzedzone symbolem podkreślenia, określającym, że są one zmiennymi domenowymi (elementami przykładowymi). Przedrostek P. umieszczono w kolumnach DATAUR i ADRES w celu określenia, że ich wartości chcemy przesłać na wyjście.

(a) PRACOWNIK

IMIĘ	INICJAŁDRIM	NAZWISKO	PESEL	DATAUR	ADRES	PŁEĆ	PENSJA	PESELPRZEŁOŻ	NRDZ
Jan	B	Nowak	_65010912345	P_1965-01-09	ul. Nowa 11, 00-900 Warszawa	_M	_3000	_55120834698	_3

(b) PRACOWNIK

IMIĘ	INICJAŁDRIM	NAZWISKO	PESEL	DATAUR	ADRES	PŁEĆ	PENSJA	PESELPRZEŁOŻ	NRDZ
Jan	B	Nowak		P_1965-01-09	ul. Nowa 11, 00-900 Warszawa				

(c) PRACOWNIK

IMIĘ	INICJAŁDRIM	NAZWISKO	PESEL	DATAUR	ADRES	PŁEĆ	PENSJA	PESELPRZEŁOŻ	NRDZ
Jan	B	Nowak		P_X	P_Y				

(d) PRACOWNIK

IMIĘ	INICJAŁDRIM	NAZWISKO	PESEL	DATAUR	ADRES	PŁEĆ	PENSJA	PESELPRZEŁOŻ	NRDZ
Jan	B	Nowak		P.	P.				

RYСУNEK C.2. Cztery sposoby określenia zapytania Z0 w języku QBE

Zapytanie Z0 można skrócić zgodnie z rysunkiem C.2(b). Nie ma potrzeby określania wartości przykładowych dla kolumn, którymi nie jesteśmy zainteresowani. Ponadto ze względu na fakt, że wartości przykładowe są całkowicie dowolne, możemy po prostu określić nazwy zmiennych, jak pokazano to na rysunku C.2(c). Wreszcie możemy również w ogóle opuścić wartości przykładowe, jak na rysunku C.2(d), i określić jedynie przedrostek P. w pobieranych kolumnach.

W celu sprawdzenia, na ile zapytania pobierające dane w języku QBE są podobne do domenowego rachunku relacyjnego, wystarczy porównać rysunek C.2(d) z zapytaniem Z0 (uproszczonym) w ramach rachunku domenowego, czyli:

Z0: {uv | PRACOWNIK(qrstuvwxyz) and q='Jan' and r='B' and s='Nowak'}

Każdą kolumnę szablonu języka QBE można traktować jako *niejawną zmienną domenową*, więc IMIĘ odpowiada zmiennej domenowej *q*, INICJAŁDRIMIENIA odpowiada zmiennej *r*, ..., zaś NRZDZ odpowiada zmiennej *z*. W przypadku zapytania QBE kolumny z przedrostkiem P. odpowiadają zmiennym określonym po lewej stronie znaku | w zapisie rachunku domenowego, natomiast kolumny z wartościami stałymi odpowiadają zmiennym krotki z nałożonymi równościowymi warunkami wyboru. Warunek PRACOWNIK(qrstuvwxyz) oraz kwantyfikator egzystencjalne są w języku QBE niejawne, ponieważ używany jest szablon odpowiadający relacji PRACOWNIK.

W języku QBE interfejs użytkownika pozwala najpierw na wybranie tabel (relacji) potrzebnych do sformułowania zapytania poprzez wyświetlenie listy nazw wszystkich relacji. Następnie są wyświetlane szablony dla wybranych relacji. Użytkownik przechodzi do odpowiednich kolumn w szablonych i określa zapytanie. Przechodzenie między szablony i wykonywanie określonych funkcji ułatwiają klawisze skrótu.

Poniżej zostaną przedstawione przykłady ilustrujące podstawowe mechanizmy języka QBE. Operatory porównania inne niż = (takie jak > lub >=) mogą być wstawiane do kolumny przed wpisaniem wartości stałej. Przykładowo, zapytanie Z0A: *Wyświetl numery PESEL pracowników, którzy pracują ponad 20 godzin tygodniowo nad projektem o numerze 1* można określić w sposób przedstawiony na rysunku C.3(a). W przypadku bardziej skomplikowanych warunków użytkownik może zażądać wyświetlenia **poła warunku**, które jest tworzone po naciśnięciu określonego klawisza funkcyjnego. Użytkownik może wówczas wpisać warunek złożony<sup>1</sup>.

PRACUJE_NAD			
(a)	PESELPRAC	NR_PROJ	GODZINY
	P.		> 20

PRACUJE_NAD			
(b)	PESELPRAC	NR_PROJ	GODZINY
	P.	_PX	_HX

WARUNKI			
	_HX > 20 and (_PX = 1 or _PX = 2)		

PRACUJE_NAD			
(c)	PESELPRAC	NR_PROJ	GODZINY
	P.	1	> 20
	P.	2	> 20

RYSUNEK C.3. Określanie warunków złożonych w języku QBE. (a) To samo zapytanie Z0A. (b) Zapytanie języka QBE z polem warunku. (c) Zapytanie języka QBE bez pola warunku

<sup>1</sup> Negacja zapisywana za pomocą symbolu ¬ nie jest dozwolona w polu warunku.

Przykładowo, zapytanie Z0B: *Wyświetl numery PESEL pracowników, którzy pracują ponad 20 godzin tygodniowo nad projektem o numerze 1 lub 2* można określić tak, jak na rysunku C.3(b).

Pewne warunki złożone można określać bez użycia pola warunku. Obowiązująca reguła określa, że wszystkie warunki określone na tym samym wierszu szablonu relacji są łączone operatorem logicznym **and** (wszystkie z nich muszą być spełnione przez wybraną krotkę), natomiast warunki określone na odrębnych wierszach są połączone operatorem **or** (musi być spełniony przynajmniej jeden z nich). Stąd zapytanie Z0B można również określić w sposób przedstawiony na rysunku C.3(c), wstawiając dwa odrębne wiersze w szablonie.

Weźmy teraz pod uwagę zapytanie Z0C: *Wyświetl numery PESEL pracowników, którzy pracują zarówno nad projektem o numerze 1, jak i 2*. Nie można go określić tak, jak na rysunku C.4(a), który wyświetla pracowników pracujących nad *któryms* z projektów 1 lub 2. Zmienna przykładowa `_ES` zostaje powiązana z wartościami PESELPRAC w krotkach `<-`, `1`, `->`, *jak również* z wartościami w krotkach `<-`, `2`, `->`. Na rysunku C.4(b) pokazano, w jaki sposób należy określić zapytanie Z0C w sposób poprawny, gdzie warunek (`_EX = _EY`) z polu wyboru sprawia, że zmienne `_EX` i `_EY` są wiązane tylko z identycznymi wartościami PESELPRAC.

PRACUJE\_NAD

(a)

PESELPRAC	NR_PROJ	GODZINY
P._ES	1	
P._ES	2	

PRACUJE\_NAD

(b)

PESELPRAC	NR_PROJ	GODZINY
P._EX	1	
P._EY	2	

WARUNKI

<code>_EX = _EY</code>
------------------------

RYSUNEK C.4. Określenie pracowników pracujących nad dwoma projektami. (a) Niepoprawne określenie warunku iloczynu logicznego AND. (b) Określenie poprawne

Ogólnie rzecz biorąc, kiedy zapytanie zostanie określone, wartości wynikowe są wyświetlane w szablonie pod odpowiednimi kolumnami. Jeżeli wynik zawiera więcej wierszy niż może być wyświetlone na ekranie, większość implementacji języka QBE oferuje klawisze funkcyjne pozwalające na przewijanie wierszy w górę i w dół. Podobnie jeżeli szablon lub kilka szablonów nie mieści się pod względem szerokości na ekranie, istnieje możliwość ich przewijania.

Operację złączenia w języku QBE określa się używając *tej samej zmiennej*<sup>2</sup> w kolumnach, które mają podlegać złączeniu. Przykładowo, zapytanie Z1: *Wyświetl imię i nazwisko oraz adres wszystkich pracowników, którzy pracują dla działu badawczego* można określić w sposób przedstawiony na rysunku C.5(a). W ramach pojedynczego zapytania można określić dowolną liczbę złączeń. Można także określić **tabelę wyników** w celu wyświetlenia wyników zapytania złączeniowego, co przedstawiono na rysunku C.5(a). Jest to konieczne w sytuacji, gdy wynik uwzględnia atrybuty pochodzące z dwóch lub więcej relacji. Jeżeli nie zostanie określona żadna tabela wyników, system generuje wynik zapytania w postaci kolumn różnych relacji, co może powodować problemy z jego interpretacją. Rysunek C.5(a) ilustruje również funkcję języka QBE służącą do określenia, że powinny zostać pobrane wszystkie atrybuty relacji, co zapewnia umieszczenie operatora P. pod nazwą relacji w jej szablonie.

W celu złączenia tabeli z samą sobą określamy różne zmienne reprezentujące różne odwołania do tabeli. Przykładowo, zapytanie Z8: *Dla każdego pracownika pobierz jego imię i nazwisko oraz imię i nazwisko jego bezpośredniego przełożonego* można określić w sposób przedstawiony na rysunku C.5(b), gdzie zmienne o nazwach rozpoczynających się od litery P odwołują się do pracownika, zaś te, których nazwy rozpoczynają się od R odwołują się do przełożonego.

(a) PRACOWNIK

IMIĘ	INICJAŁDRIM	NAZWISKO	PESEL	DATAUR	ADRES	PŁEĆ	PENSJA	PESELPRZEŁOŻ	NRDZ
_IM		_NZ			_ADR				_DX

DZIAŁ

NAZWADZ	NUMERDZ	PESELKIEROWNIKA	DATAPRZEJKIEROWNICTWA
Badawczy	_DX		

WYNIK			
P.	_IM	_NZ	_ADR

(b) PRACOWNIK

IMIĘ	INICJAŁDRIM	NAZWISKO	PESEL	DATAUR	ADRES	PŁEĆ	PENSJA	PESELPRZEŁOŻ	NRDZ
_P1		_P2						_XPESEL	
_R1		_R2	_XPESEL						

WYNIK				
P.	_P1	_P2	_R1	_R2

RYСУNEK C.5. Ilustracja operacji złączenia i relacji wynikowych w języku QBE. (a) Zapytanie Z1. (b) Zapytanie Z8

<sup>2</sup> W podręcznikach języka QBE zmienna jest określana mianem **elementu przykładowego**.

## C.2. Grupowanie, agregacje i modyfikacje bazy danych w języku QBE

Weźmy teraz pod uwagę zapytania wymagające użycia funkcji grupujących lub agregujących. W kolumnie można określić operator grupowania G. w celu wskazania, że krotki powinny zostać pogrupowane na podstawie wartości danej kolumny. Można określać funkcje, takie jak AVG., SUM., CNT. (count), MAX. oraz MIN. W języku QBE funkcje AVG., SUM. i CNT. są domyślnie stosowane względem odrębnych (ang. *distinct*) wartości w ramach grupy. Jeżeli chcemy, aby funkcje te zostały zastosowane względem wszystkich wartości, musimy użyć przedrostka ALL<sup>3</sup>. Taka konwencja *różni się* od konwencji języka SQL, gdzie domyślnym postępowaniem jest zastosowanie funkcji względem wszystkich wartości.

Na rysunku C.6(a) przedstawiono zapytanie Z23, które zlicza liczbę *odrębnych* wartości pensji w ramach relacji PRACOWNIK. Zapytanie Z23A (rysunek C.6(b)) zlicza liczbę wszystkich wartości pensji, co jest równoważne zliczeniu liczby pracowników (pod warunkiem, że każdemu pracownikowi jest przypisana pensja). Na rysunku C.6(c) przedstawiono zapytanie Z24, które pobiera numer każdego działu oraz liczbę pracowników i średnią pensję w każdym dziale. Stąd kolumna NRDZ jest używana w celu grupowania zgodnie z funkcją G.. W ramach jednej kolumny można określić kilka operatorów G., P. i ALL. Na rysunku C.6(d) przedstawiono zapytanie Z26, które wyświetla nazwę każdego projektu oraz liczbę pracujących nad nim pracowników w projektach, nad którymi pracuje więcej niż dwóch pracowników.

(a) PRACOWNIK

IMIĘ	INICJAŁDRIM	NAZWISKO	PESEL	DATAUR	ADRES	PLEĆ	PENSJA	PESELPRZEŁOŻ	NRDZ
							P.CNT.		

(b) PRACOWNIK

IMIĘ	INICJAŁDRIM	NAZWISKO	PESEL	DATAUR	ADRES	PLEĆ	PENSJA	PESELPRZEŁOŻ	NRDZ
							P.CNT.ALL		

(c) PRACOWNIK

IMIĘ	INICJAŁDRIM	NAZWISKO	PESEL	DATAUR	ADRES	PLEĆ	PENSJA	PESELPRZEŁOŻ	NRDZ
			P.CNT.ALL				P.AVG.ALL		P.G.

(d) PROJEKT

NAZWAPROJ	NUMERPROJ	LOKALIZACJAPROJ	NR_DZ
P.	_PX		

PRACUJE\_NAD

PESELPRAC	NR_PROJ	GODZINY
P.CNT.EX	G._PX	

WARUNKI

CNT_EX > 2
------------

RYСУNEK C.6. Funkcje i grupowanie w języku QBE. (a) Zapytanie Z23. (b) Zapytanie Z23A. (c) Zapytanie Z24. (d) Zapytanie Z26

<sup>3</sup> Przedrostek ALL w przypadku języka QBE nie ma związku z kwantyfikatorem uniwersalnym.

Język QBE posiada symbol negacji  $\neg$ , który jest używany w sposób podobny do funkcji NOT EXISTS języka SQL. Na rysunku C.7 przedstawiono zapytanie Z6, które wyświetla listę imion i nazwisk pracowników nieposiadających członków rodziny. Symbol negacji  $\neg$  mówi tyle, że wybieramy wartości zmiennej  $\_PX$  z relacji PRACOWNIK tylko wówczas, gdy nie występują one w relacji CZŁONEK\_RODZINY. Ten sam efekt można osiągnąć, umieszczając operator  $\neg$   $\_PX$  w kolumnie PESELPRAC.

PRACOWNIK									
IMIĘ	INICJAŁDRIM	NAZWISKO	PESEL	DATAUR	ADRES	PŁEĆ	PENSJA	PESELPRZEŁ	NRDZ
P.		P.	$\_SX$						

CZŁONEK_RODZINY				
PESELPRAC	IMIĘ_CZŁONKA_RODZINY	PESEL	DATAUR	STOPIEŃ_POKREWIEŃSTWA
$\neg$ $\_SX$				

RYSUNEK C.7. Ilustracja negacji w zapytaniu Z6

Chociaż język QBE w swojej oryginalnej postaci miał obsługiwać konstrukcje równoważne funkcjom EXISTS i NOT EXISTS z języka SQL, jednak jego implementacja w ramach QMF (w systemie DB2) *nie* oferuje takiej obsługi. Stąd omawiana tu wersja języka QBE interfejsu QMF *nie jest relacyjnie kompletna*. Zapytania takie jak Z3: *Znajdź pracowników, którzy pracują nad wszystkimi projektami prowadzonymi przez wydział o numerze 5* nie mogą być określone.

Istnieją trzy operatory języka QBE służące do modyfikowania bazy danych: I. dla wstawiania (od *insert*), D. dla usuwania (od *delete*) oraz U. dla aktualizowania (od *update*). Operatory wstawiania i usuwania określa się w kolumnie szablonu pod nazwą relacji, natomiast operator aktualizacji określa się pod kolumnami, które mają zostać zaktualizowane. Na rysunku C.8(a) przedstawiono, w jaki sposób można wstawić nową krotkę do relacji PRACOWNIK. W przypadku usuwania, najpierw wstawiamy operator D., a następnie określamy usuwane krotki poprzez warunek (rysunek C.8(b)). W celu zaktualizowania krotki określamy atrybut U. pod nazwą atrybutu, a później jego nową wartość. Należy również wybrać aktualizowaną krotkę lub krotki w normalny sposób. Na rysunku C.8(c) przedstawiono operację aktualizacji, która zwiększa pensję *Jana Nowaka* o 10 procent, jak również przenosi go do działu o numerze 4.

(a) PRACOWNIK

	IMIĘ	INICJAŁDRIM	NAZWISKO	PESEL	DATAUR	ADRES	PŁEĆ	PENSJA	PESELPRZEŁOŻ	NRDZ
I.	Ryszard	K	Mariański	52123065339	1952-12-30	ul. Dębowa98, Kęty	M	3700	41062013258	4

(b) PRACOWNIK

	IMIĘ	INICJAŁDRIM	NAZWISKO	PESEL	DATAUR	ADRES	PŁEĆ	PENSJA	PESELPRZEŁOŻ	NRDZ
D.				52123065339						

(c) PRACOWNIK

	IMIĘ	INICJAŁDRIM	NAZWISKO	PESEL	DATAUR	ADRES	PŁEĆ	PENSJA	PESELPRZEŁOŻ	NRDZ
	Jan		Nowak					U_5*1.1		U.4

RYSUNEK C.8. Modyfikowanie bazy danych w języku QBE. (a) Wstawianie. (b) Usuwanie. (c) Aktualizowanie

Język QBE posiada również możliwości definiowania danych. Tabele bazy danych mogą być określane interaktywnie, a definicja tabeli może zostać zaktualizowana przez dodanie, zmianę nazwy lub usunięcie kolumny. Można także określać różnorodne charakterystyki każdej kolumny, takie jak to, czy jest kluczem danej relacji, jakiego jest typu oraz czy na danym polu powinien zostać utworzony indeks. Język QBE posiada również możliwość definiowania perspektyw, autoryzacji, przechowywania definicji zapytań dla celów przyszłego użycia itd.

Język QBE nie wykorzystuje *liniowego* stylu języka SQL; jest raczej *dwuwymiarowym* językiem, ponieważ użytkownicy określają zapytanie, przechodząc po całym ekranie. Testy przeprowadzone wśród użytkowników wykazały, że język QBE jest łatwiejszy do nauczenia od SQL, szczególnie dla osób nieposiadających wykształcenia technicznego. W tym sensie język QBE był *pierwszym* przyjaznym dla użytkownika *wizualnym* językiem relacyjnych baz danych.

Ostatnio opracowano dla systemów komercyjnych baz danych kilka innych interfejsów przyjaznych dla użytkownika. Używanie menu, elementów graficznych i formularzy staje się codziennością. Operacją podobną do używania języka QBE jest częściowe wypełnianie formularzy w celu uruchomienia wyszukiwania. Wizualne języki zapytań, które wciąż nie są zbyt popularne, prawdopodobnie będą oferowane wraz z komercyjnymi relacyjnymi bazami danych w przyszłości.



# D

---

## Bibliografia

### Skróty używane w bibliografii:

**ACM:** Association for Computing Machinery

**AFIPS:** American Federation of Information Processing Societies

**ASPLOS:** Sprawozdania z International Conference on Architectural Support for Programming Languages and Operating Systems

**CACM:** Communications of the ACM (czasopismo)

**CIKM:** Sprawozdania z International Conference on Information and Knowledge Management

**DASFAA:** Sprawozdania z International Conference on Database Systems for Advanced Applications

**DKE:** Data and Knowledge Engineering, Elsevier Publishing (czasopismo)

**EDBT:** Sprawozdania z International Conference on Extending Database Technology

**EDS:** Sprawozdania z International Conference on Expert Database Systems

**ER Conference:** Sprawozdania z International Conference on Entity-Relationship Approach (obecnie używaną nazwą jest International Conference on Conceptual Modeling)

**ICDCS:** Sprawozdania z IEEE International Conference on Distributed Computing Systems

**ICDE:** Sprawozdania z IEEE International Conference on Data Engineering

**IEEE:** Institute of Electrical and Electronics Engineers

**IEEE Computer:** Czasopismo komputerowe IEEE CS

**IEEE CS:** IEEE Computer Society

**IFIP:** International Federation for Information Processing

**JACM:** Journal of the ACM

**KDD:** Knowledge Discovery in Databases

**LNCS:** Lecture Notes in Computer Science

**NCC:** Sprawozdania z National Computer Conference (publikowane przez AFIPS)

**OOPSLA:** Sprawozdania z ACM Conference on Object-Oriented Programming Systems, Languages, and Applications

**OSDI:** USENIX Symposium on Operating Systems Design and Implementation

**PAMI:** Pattern Analysis and Machine Intelligence

**PODS:** Sprawozdania z ACM Symposium on Principles of Database Systems

**SIGMETRICS:** Sprawozdania z ACM International Conference on Measurement and Modeling of Computer Systems

**SIGMOD:** Sprawozdania z ACM SIGMOD International Conference on Management of Data

**SOSP:** ACM Symposium on Operating System Principles

**TKDE:** IEEE Transactions on Knowledge and Data Engineering (czasopismo)

**TOCS:** ACM Transactions on Computer Systems (czasopismo)

**TODS:** ACM Transactions on Database Systems (czasopismo)

**TOIS:** ACM Transactions on Information Systems (czasopismo)

**TOOIS:** ACM Transactions on Office Information Systems (czasopismo)

**TPDS:** IEEE Transactions of Parallel and Distributed Systems (czasopismo)

**TSE:** IEEE Transactions on Software Engineering (czasopismo)

**VLDB:** Sprawozdania z International Conference on Very Large Data Bases (wydania od roku 1981 udostępniane przez wydawnictwo Morgan Kaufmann, Menlo Park, California)

## Format zapisu pozycji w bibliografii

Tytuły książek są zapisywane pogrubieniem, na przykład **Database Computers**. Nazwy sprawozdań konferencyjnych są zapisywane kursywą, na przykład *ACM Pacific Conference*. Nazwy czasopism są zapisywane pogrubieniem, na przykład **TODS** lub **Information Systems**. W przypadku cytatów z czasopism podawany jest numer tomu oraz numer wydania (w ramach tomu, o ile występuje), a także data wydania. Przykładowo, zapis „**TODS**, 3:4, grudzień 1978” odnosi się do wydania czasopisma *ACM Transactions on Database Systems* z grudnia 1978 w tomie 3, o numerze 4. Odwołania do artykułów znajdujących się w książkach lub w sprawozdaniach konferencyjnych, które są oddzielnie wymienione w bibliografii, występują w formie zapisu „w:” z odpowiednim tytułem, na przykład „w: VLDB [1978]” lub „w: Rustin [1974]”. Numery stron (zapisywane w skrócie jako „s.”) są podawane na końcu każdej pozycji, tam gdzie to możliwe. W przypadku pozycji autorstwa więcej niż czterech osób podawany jest jedynie pierwszy autor oraz skrót „i in.”. W wyborze publikacji umieszczonym na końcu każdego rozdziału używano zapisu „i in.” wówczas, gdy autorów było więcej niż dwóch.

## Publikacje

Abadi, D. J., Madden, S. R., Hachem, N. [2008] „Column Stores vs. Row Stores: How Different Are They Really?”, w: *SIGMOD* [2008].

Abbott R., Garcia-Molina H. [1989] „Scheduling Real-Time Transactions with Disk Resident Data”, w: *VLDB* [1989].

Abiteboul S., Kanellakis P. [1989] „Object Identity as a Query Language Primitive”, w: *SIGMOD* [1989].

- Abiteboul S., Hull R., Vianu V. [1995] **Foundations of Databases**, Addison-Wesley, 1995.
- Abramova, V., Bernardino, J. [2013] „NoSQL Databases: MongoDB vs Cassandra”, *Proc. Sixth Int. Conf. on Comp. Sci. and Software Eng. (C3S2E'13)*, Porto, Portugalia, lipiec 2013, s. 14 – 22.
- Abrial J. [1974] „Data Semantics”, w: Klimbie i Koffeman [1974].
- Acharya, S., Alonso, R., Franklin, M., Zdonik, S. [1995] „Broadcast Disks: Data Management for Asymmetric Communication Environments”, w: *SIGMOD* [1995].
- Adam N., Gongopadhyay A. [1993] „Integrating Functional and Data Modeling in a Computer Integrated Manufacturing System”, w: *ICDE* [1993].
- Adriaans P., Zantinge D. [1996] **Data Mining**, Addison-Wesley, 1996.
- Afsarmanesh H., McLeod D., Knapp D., Parker A. [1985] „An Extensible Object-Oriented Approach to Databases for VLSI/CAD”, w: *VLDB* [1985].
- Afrati, F., Ullman, J. [2010] „Optimizing Joins in a MapReduce Environment”, w: *EDBT* [2010].
- Agneesswaran, V.S. [2014] **Big Data Analytics Beyond Hadoop: Real-Time Applications with Storm, Spark, and More Hadoop Alternatives**, Pearson FT Press, 2014, 240 s.
- Agrawal D., ElAbbad A. [1990] „Storage Efficient Replicated Databases”, **TKDE**, 2:3, wrzesień 1990.
- Agrawal, R. i in. [2008] „The Claremont Report on Database Research”, tekst dostępny na stronie: <http://db.cs.berkeley.edu/claremont/claremontreport08.pdf>, maj 2008.
- Agrawal R., Gehani N. [1989] „ODE: The Language and the Data Model”, w: *SIGMOD* [1989].
- Agrawal R., Srikant R. [1994] „Fast Algorithms for Mining Association Rules in Large Databases”, w: *VLDB* [1994].
- Agrawal R., Gehani N., Srinivasan J. [1990] „OdeView: The Graphical Interface to Ode”, w: *SIGMOD* [1990].
- Agrawal R., Imielinski T., Swami A. [1993] „Mining Association Rules Between Sets of Items in Databases”, w: *SIGMOD* [1993].
- Agrawal R., Imielinski T., Swami A. [1993b] „Database Mining: A Performance Perspective”, **TKDE** 5:6, grudzień 1993.
- Agrawal R., Mehta M., Shafer J., Srikant R. [1996] „The Quest Data Mining System”, w: *KDD* [1996].
- Ahad R., Basu A. [1991] „ESQL: A Query Language for the Relational Model Supporting Image Domains”, w: *ICDE* [1991].
- Ahmed R. i in. [2006] „Cost-Based Query Transformation in Oracle”, w: *VLDB* [2006].
- Ahmed R. i in. [2014] „Of Snowstorms and Bushy Trees”, w: *VLDB* [2014].
- Aho A., Ullman J. [1979] „Universality of Data Retrieval Languages”, *Proceedings of the POPL Conference*, San Antonio TX, ACM, 1979.
- Aho A., Beeri C., Ullman J. [1979] „The Theory of Joins in Relational Databases”, **TODS**, 4:3, wrzesień 1979.
- Aho A., Sagiv Y., Ullman J. [1979a] „Efficient Optimization of a Class of Relational Expressions”, **TODS**, 4:4, grudzień 1979.
- Akl S. [1983] „Digital Signatures: A Tutorial Survey”, **IEEE Computer**, 16:2, luty 1983.

- Alagic, S. [1999] „A Family of the ODMG Object Models”, w: *Advances in Databases and Information Systems, Third East European Conference, ADBIS'99*, Maribor, Slovenia, J. Eder, I. Rozman, T. Welzer (red.), wrzesień 1999, LNCS, nr 1691, Springer.
- Alashqur A., Su S., Lam H. [1989] „OQL: A Query Language for Manipulating Object-Oriented Databases”, w: *VLDB* [1989].
- Albano A., Cardelli L., Orsini R. [1985] „GALILEO: A Strongly Typed Interactive Conceptual Language”, **TODS**, 10:2, czerwiec 1985.
- Albrecht J. H. [1996] „Universal GIS Operations”, University of Osnabrück, Niemcy, praca doktorska, 1996.
- Allen F., Loomis M., Mannino M. [1982] „The Integrated Dictionary/Directory System”, **ACM Computing Surveys**, 14:2, czerwiec 1982.
- Allen, J. [1983] „Maintaining Knowledge about Temporal Intervals”, w: **CACM** 26:11, listopad 1983, s. 832 – 843.
- Alonso G., Agrawal D., El Abbadi A., Mohan C. [1997] „Functionalities and Limitations of Current Workflow Management Systems”, **IEEE Expert**, 1997.
- Amir A., Feldman R., Kashi, R. [1997] „A New and Versatile Method for Association Generation”, **Information Systems**, 22:6, wrzesień 1997.
- Ananthanarayanan, G. i in. [2012] „PACMan: Coordinated Memory Caching for Parallel Jobs”, w: *USENIX Symp. on Networked Systems Design and Implementation (NSDI)*, 2012.
- Anderson S. i in. [1981] „Sequence and Organization of the Human Mitochondrial Genome”, **Nature**, 290:457 – 465, 1981.
- Andrews T., Harris C. [1987] „Combining Language and Database Advances in an Object-Oriented Development Environment”, *OOPSLA*, 1987.
- ANSI [1975] American National Standards Institute Study Group on Data Base Management Systems: Interim Report, FDT, 7:2, ACM, 1975.
- ANSI [1986] American National Standards Institute: **The Database Language SQL**, Document ANSI X3.135, 1986.
- ANSI [1986a] American National Standards Institute: **The Database Language NDL**, Document ANSI X3.133, 1986.
- ANSI [1989] American National Standards Institute: **Information Resource Dictionary Systems**, Document ANSI X3.138, 1989.
- Antenucci, J. i in. [1998] **Geographic Information Systems: A Guide to the Technology**, Chapman and Hall, maj 1998.
- Anwar T., Beck H., Navathe S. [1992] „Knowledge Mining by Imprecise Querying: A Classification Based Approach”, w: *ICDE* [1992].
- Apers P., Hevner A., Yao S. [1983] „Optimization Algorithms for Distributed Queries”, **TSE**, 9:1, styczeń 1983.
- Apweiler, R., Martin, M., O'Donovan, C., Prues, M. [2003] „Managing Core Resources for Genomics and Proteomics”, **Pharmacogenomics**, 4:3, maj 2003, s. 343 – 350.
- Aref, W. i in. [2004] „VDBMS: A Testbed Facility or Research in Video Database Benchmarking”, w: **Multimedia Systems (MMS)**, 9:6, czerwiec 2004, s. 98 – 115.

- Arisawa, H., Catarci, T. [2000] *Advances in Visual Information Management, Proc. Fifth Working Conf. On Visual Database Systems*, Arisawa, H., Catarci, T. (red.), Fujkuoka, Japonia, *IFIP Conference Proceedings 168*, Kluwer, 2000.
- Armstrong W. [1974] „Dependency Structures of Data Base Relationships”, *Proceedings of the 1F1P Congress*, 1974.
- Ashburner, M. i in. [2000] „Gene Ontology: Tool for the unification of biology”, **Nature Genetics**, tom 25, maj 2000, s. 25 – 29.
- Astrahan M. i in. [1976] „System R: A Relational Approach to Data Base Management”, **TODS**, 1:2, czerwiec 1976.
- Atkinson M., Buneman P. [1987] „Types and Persistence in Database Programming Languages”, w: **ACM Computing Surveys**, 19:2, czerwiec 1987.
- Atkinson M. i in. [1990] *The Object-Oriented Database System Manifesto, Proc. Deductive and Object Oriented Database Conf. (DOOD)*, Kioto, Japonia, 1990.
- Atluri V. i in. [1997] „Multilevel Secure Transaction Processing: Status and Prospects”, w: **Database Security: Status and Prospects**, Chapman and Hall, 1997, s. 79 – 98.
- Atzeni P., De Antonellis V. [1993] **Relational Database Theory**, Benjamin/Cummings, 1993.
- Atzeni P., Mecca G., Merialdo P. [1997] „To Weave the Web”, w: *VLDB* [1997].
- Bachman C. [1969] „Data Structure Diagrams”, **Data Base** (biuletyn ACM SIGFIDET), 1:2, marzec 1969.
- Bachman C. [1973] „The Programmer as a Navigator”, **CACM**, 16:1, listopad 1973.
- Bachman C. [1974] „The Data Structure Set Model”, w: Rustin [1974].
- Bachman C., Williams S. [1964] „A General Purpose Programming System for Random Access Memories”, *Proceedings of the Fall Joint Computer Conference*, AFIPS, 26, 1964.
- Badal D., Popek G. [1979] „Cost and Performance Analysis of Semantic Integrity Validation Methods”, w: *SIGMOD* [1979].
- Badrinath, B., Imielinski, T. [1992] „Replication and Mobility”, *Proc. Workshop on the Management of Replicated Data 1992*: s. 9 – 12.
- Badrinath B., Ramamritham K. [1992] „Semantics-Based Concurrency Control: Beyond Commutativity”, **TODS**, 17:1, marzec 1992.
- Bahga, A., Madiseti, V. [2013] **Cloud Computing—A Hands On Approach**, (<http://www.cloudcomputingbook.info>), 2013, 454 s.
- Baeza-Yates R., Larson P. A. [1989] „Performance of B+-trees with Partial Expansions”, **TKDE**, 1:2, czerwiec 1989.
- Baeza-Yates R., Ribero-Neto B. [1999] **Modern Information Retrieval**, Addison-Wesley, 1999.
- Balbin I., Ramamohanrao K. [1987] „A Generalization of the Different Approach to Recursive Query Evaluation”, **Journal of Logic Programming**, 15:4, 1987.
- Bancilhon, F. [1985] „Naive Evaluation of Recursively Defined Relations,” w: **On Knowledge Base Management Systems** (Brodie, M., Mylopoulos, J., red.), warsztaty Islamorada 1985, Springer, s. 165 – 178.
- Bancilhon F., Buneman P., red. [1990] **Advances in Database Programming Languages**, ACM Press, 1990.

- Bancilhon, F., Ferran, G. [1995] „The ODMG Standard for Object Databases”, DASFAA 1995, Singapur, s. 273 – 283.
- Bancilhon F., Ramakrishnan R. [1986] „An Amateur’s Introduction to Recursive Query Processing Strategies”, w: *SIGMOD* [1986].
- Bancilhon F., Delobel C., Kanellakis P., red. [1992] **Building an Object-Oriented Database System: The Story of O2**, Morgan Kaufmann, 1992.
- Bancilhon F., Maier D., Sagiv Y., Ullman J. [1986] „Magic sets and other strange ways to implement logic programs”, *PODS* [1986].
- Banerjee J. i in. [1987] „Data Model Issues for Object-Oriented Applications”, **TOOIS**, 5:1, styczeń 1987.
- Banerjee J., Kim W., Kim H., Korth H. [1987a] „Semantics and Implementation of Schema Evolution in Object-Oriented Databases”, w: *SIGMOD* [1987].
- Barbara, D. [1999] „Mobile Computing and Databases – A Survey”, **TKDE**, 11:1, styczeń 1999.
- Baroody A., DeWitt D. [1981] „An Object-Oriented Approach to Database System Implementation”, **TODS**, 6:4, grudzień 1981.
- Barrett T. i in. [2005] „NCBI GEO: mining millions of expression profiles—database and tools”, **Nucleic Acid Research**, 33: numer poświęcony bazom danych, 2005, s. 562 – 566.
- Barrett, T. i in. [2007] „NCBI GEO: mining tens of millions of expression profiles—database and tools update”, w: **Nucleic Acids Research**, 35:1, styczeń 2007.
- Barsalou T., Siambela N., Keller A., Wiederhold G. [1991] „Updating Relational Databases Through Object-Based Views”, w: *SIGMOD* [1991].
- Bassiouni M. [1988] „Single-Site and Distributed Optimistic Protocols for Concurrency Control”, **TSE**, 14:8, sierpień 1988.
- Batini C., Ceri S., Navathe S. [1992] **Database Design: An Entity-Relationship Approach**, Benjamin/Cummings, 1992.
- Batini C., Lenzerini M., Navathe, S. [1987] „A Comparative Analysis of Methodologies for Database Schema Integration”, **ACM Computing Surveys**, 18:4, grudzień 1987.
- Batory D. i in. [1988] „GENESIS: An Extensible Database Management System”, **TSE**, 14:11, listopad 1988.
- Batory, D., Buchmann, A. [1984] „Molecular Objects, Abstract Data Types, and Data Models: A Framework”, w: *VLDB* [1984].
- Bay, H., Tuytelaars, T., Gool, L. V. [2006] „SURF: Speeded Up Robust Features”, w: *Proc. Ninth European Conference on Computer Vision*, maj 2006.
- Bayer R., McCreight E. [1972] „Organization and Maintenance of Large Ordered Indexes”, **Acta Informatica**, 1:3, luty 1972.
- Bayer R., Graham M., Seegmuller G., red. [1978] **Operating Systems: An Advanced Course**, Springer-Verlag, 1978.
- Beck H., Anwar T., Navathe S. [1994] „A Conceptual Clustering Algorithm for Database Schema Design”, **TKDE**, 6:3, czerwiec 1994.
- Beck H., Gala S., Navathe S. [1989] „Classification as a Query Processing Technique in the CANDIDE Semantic Data Model”, w: *ICDE* [1989].

- Beeri C., Ramakrishnan R. [1987] „On the Power of Magic”, w: *PODS* [1987].
- Beeri C., Fagin R., Howard J. [1977] „A Complete Axiomatization for Functional and Multivalued Dependencies”, w: *SIGMOD* [1977].
- Bellamkonda, S. i in., [2009], „Enhanced Subquery Optimization in Oracle”, w: *VLDB* [2009]
- Bell, D.E., Lapadula, L.J. [1976] **Secure computer system: Unified exposition and Multics Interpretation**, raport techniczny MTR-2997, MITRE Corp., Bedford, MA, marzec 1976.
- Ben-Zvi J. [1982] „The Time Relational Model”, praca doktorska, University of California, Los Angeles, 1982.
- Benson D., Boguski M., Lipman D., Ostell J., „GenBank”, **Nucleic Acids Research**, 24:1, 1996.
- Benson, D., Karsch-Mizrachi, I., Lipman, D. i in. [2002] „GenBank”, **Nucleic Acids Research**, 36:1, styczeń 2008.
- Berg B., Roth J. [1989] **Software for Optical Disk**, Meckler, 1989.
- Bergman, M. K. [2001] „The Deep Web: Surfacing Hidden Value”, **The Journal of Electronic Publishing**, 7:1, sierpień 2001.
- Berners-Lee T., Caillian R., Grooff J., Pollermann B. [1992] „World-Wide Web: The Information Universe”, **Electronic Networking: Research, Applications and Policy**, 1:2, 1992.
- Berners-Lee T., Caillian R., Lautonen A., Nielsen H., Secret A. [1994] „The World Wide Web”, **CACM**, 13:2, sierpień 1994.
- Bernstein P. [1976] „Synthesizing Third Normal Form Relations from Functional Dependencies”, **TODS**, 1:4, grudzień 1976.
- Bernstein, P. and Goodman, N. [1983] “Multiversion Concurrency Control—Theory and Algorithms,” **TODS**, 8:4, s. 465 –483.
- Bernstein P., Goodman N. [1980] „Timestamp-Based Algorithms for Concurrency Control in Distributed Database Systems”, w: *VLDB* [1980].
- Bernstein P., Goodman N. [1981a] „Concurrency Control in Distributed Database Systems”, **ACM Computing Surveys**, 13:2, czerwiec 1981.
- Bernstein P., Goodman N. [1981b] „The Power of Natural Semijoins”, **SIAM Journal of Computing**, 10:4, grudzień 1981.
- Bernstein P., Goodman N. [1984] „An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases”, **TODS**, 9:4, grudzień 1984.
- Bernstein P., Blaustein B., Clarke E. [1980] „Fast Maintenance of Semantic Integrity Assertions Using Redundant Aggregate Data”, w: *VLDB* [1980].
- Bernstein P., Hadzilacos V., Goodman N. [1987] **Concurrency Control and Recovery in Database Systems**, Addison-Wesley, 1988.
- Bertino E. [1992] „Data Hiding and Security in Object-Oriented Databases”, w: *ICDE* [1992].
- Bertino, E. [1998] „Data Security”, w: **DKE** 25:1 – 2, s. 199 – 216.
- Bertino, E. Sandhu, R. [2005] „Security—Concepts, Approaches, and Challenges”, w: **IEEE Transactions on Dependable Secure Computing (TDSC)**, 2:1, 2005, s. 2 – 19.
- Bertino, E. Guerrini, G. [1998] „Extending the ODMG Object Model with Composite Objects”, *OOPSLA*, Vancouver, Kanada, 1998, s. 259 – 270.



- Bertino E., Kim W. [1989] „Indexing Techniques for Queries on Nested Objects”, **TKDE**, 1:2, czerwiec 1989.
- Bertino E., Catania B., Ferrari E. [2001] „A Nested Transaction Model for Multilevel Secure Database Management Systems”, **ACM Transactions on Information and System Security**, 4:4, listopad 2001, s. 321 – 370.
- Bertino E., Negri M., Pelagatti G., Sbattella L. [1992] „Object-Oriented Query Languages: The Notion and the Issues”, **TKDE**, 4:3, czerwiec 1992.
- Bertino E., Pagani E., Rossi G. [1992] „Fault Tolerance and Recovery in Mobile Computing Systems”, w: Kumar i Han [1992].
- Bertino F., Rabbitti, Gibbs S. [1988] „Query Processing in a Multimedia Environment”, **TOIS**, 6, 1988.
- Bhargava B., Helal A. [1993] „Efficient Reliability Mechanisms in Distributed Database Systems”, *CIKM*, listopad 1993.
- Bhargava B., Reidl J. [1988] „A Model for Adaptable Systems for Transaction Processing”, w: *ICDE* [1988].
- Bikel, D., Zitouni, I. [2012] **Multilingual Natural Language Processing Applications: From Theory to Practice**, IBM Press, 2012.
- Biliris A. [1992] „The Performance of Three Database Storage Structures for Managing Large Objects”, w: *SIGMOD* [1992].
- Biller H. [1979] „On the Equivalence of Data Base Schemas — A Semantic Approach to Data Translation”, **Information Systems**, 4:1, 1979.
- Bischoff J., Alexander T., red., **Data Warehouse: Practical Advice from the Experts**, Prentice-Hall, 1997.
- Biskup J., Dayal U., Bernstein P. [1979] „Synthesizing Independent Database Schemas”, w: *SIGMOD* [1979].
- Bitton, D., Gray, J. [1988] „Disk Shadowing”, w: *VLDB* [1988], s. 331 – 338.
- Bjork A. [1973] „Recovery Scenario for a DB/DC System”, *Proceedings of the ACM National Conference*, 1973.
- Bjorner D., Lovengren H. [1982] „Formalization of Database Systems and a Formal Definition of IMS”, w: *VLDB* [1982].
- Blaha, M., Rumbaugh, J. [2005] **Object-Oriented Modeling and Design with UML**, wydanie drugie, PrenticeHall, 2005.
- Blaha M., Premerlani W. [1998] **Object-Oriented Modeling and Design for Database Applications**, Prentice-Hall, 1998.
- Blakely, J., Larson, P., Tompa, F.W. [1986] „Efficiently Updating Materialized Views”, w: *SIGMOD* [1986], s. 61 – 71.
- Blakeley J., Martin N. [1990] „Join Index, Materialized View, and Hybrid-Hash Join: A Performance Analysis”, w: *ICDE* [1990].
- Blakeley J., Coburn N., Larson P. [1989] „Updated Derived Relations: Detecting Irrelevant and Autonomously Computable Updates”, **TODS**, 14:3, wrzesień 1989.
- Blasgen M. i in. [1981] „System R: An Architectural Overview”, **IBM Systems Journal**, 20:1, styczeń 1981.

- Blasgen M., Eswaran K. [1976] „On the Evaluation of Queries in a Relational Database System”, **IBM Systems Journal**, 16:1, styczeń 1976.
- Blei, D.M., Ng, A.Y., Jordan, M.I. [2003] „Latent Dirichlet Allocation”, **Journal of Machine Learning Research**, 3, marzec 2003, s. 993 – 1022.
- Bleier R., Vorhaus A. [1968] „File Organization in the SDC TDMS”, *Proceedings of the IFIP Congress*.
- Bocca J. [1986] „EDUCE — A Marriage of Convenience: Prolog and a Relational DBMS”, *Proceedings of the Third International Conference on Logic Programming*, Springer-Verlag, 1986.
- Bocca J. [1986a] „On the Evaluation Strategy of EDUCE”, w: *SIGMOD* [1986].
- Bodorick P., Riordon J., Pyra J. [1992] „Deciding on Correct Distributed Query Processing”, **TKDE**, 4:3, czerwiec 1992.
- Boncz, P., Zukowski, M., Nes, N. [2005] „MonetDB/X100: Hyper-Pipelining Query Execution”, w: *Proc. Conf. on Innovative Data Systems Research CIDR* [2005].
- Bonnet, P., Gehrke, J., Seshadri, P. [2001] „Towards Sensor Database Systems”, w: *Proc. 2nd Int. Conf. on Mobile Data Management*, Hong Kong, Chiny, LNCS 1987, Springer, styczeń 2001, s. 3 – 14.
- Booch G., Rumbaugh J., Jacobson I., **Unified Modeling Language User Guide**, Addison-Wesley, 1999.
- Borges, K., Laender, A., Davis, C. [1999] „Spatial data integrity constraints in object oriented geographic data modeling”, *Proc. 7th ACM International Symposium on Advances in Geographic Information Systems*, 1999.
- Borgida A., Brachman R., McGuinness D., Resnick L. [1989] „CLASSIC: A Structural Data Model for Objects”, w: *SIGMOD* [1989].
- Borkin S. [1978] „Data Model Equivalence”, w: *VLDB* [1978].
- Bossomaier, T., Green, D. [2002] **Online GIS and Metadata**, Taylor and Francis, 2002.
- Boukerche, A., Tuck, T. [2001] „Improving Concurrency Control in Distributed Databases with Predeclared Tables”, w: *Proc. Euro-Par 2001: Parallel Processing, 7th International Euro-Par Conference*, Manchester, Wielka Brytania, Sierpień 28 – 31, 2001, s. 301 – 309.
- Boutselakis, H. i in. [2003] „E-MSD: the European Bioinformatics Institute Macromolecular Structure Database”, **Nucleic Acids Research**, 31:1, styczeń 2003, s. 458 – 462.
- Bouzeghoub M., Metais E. [1991] „Semantic Modelling of Object-Oriented Databases”, w: *VLDB* [1991].
- Boyce R., Chamberlin D., King W., Hammer M. [1975] „Specifying Queries as Relational Expressions”, **CACM**, 18:11, listopad 1975.
- Boyd, S., Keromytis, A. [2004] „SQLrand: Preventing SQL injection attacks”, w: *Proc. 2nd Applied Cryptography and Network Security Conf. (ACNS 2004)*, czerwiec 2004, s. 292 – 302.
- Braam, P., Schwan, P. [2002] Lustre: The intergalactic file system, *Proc. Ottawa Linux Symposium*, czerwiec 2002 (<http://ols.fedoraproject.org/OLS/Reprints-2002/braam-reprint.pdf>).
- Bracchi G., Paolini P., Pelagatti G. [1976] „Binary Logical Associations in Data Modelling”, w: Nijssen [1976].
- Brachman R., Levesque H. [1984] „What Makes a Knowledge Base Knowledgeable? A View of Databases from the Knowledge Level”, w: *EDS* [1984].
- Brandon, M. i in. [2005] MITOMAP: A human mitochondrial genome database—2004 Update, **Nucleic Acid Research**, 34:1, styczeń 2005.

- Bratbergsengen K. [1984] „Hashing Methods and Relational Algebra Operators”, w: *VLDB* [1984].
- Bray O. [1988] **Computer Integrated Manufacturing — The Data Management Strategy**, Digital Press, 1988.
- Breitbart, Y., Komondoor, R., Rastogi, R., Seshadri, S., Silberschatz, A. [1999] „Update Propagation Protocols for Replicated Databases”, w: *SIGMOD* [1999], s. 97 – 108.
- Breitbart Y., Silberschatz A., Thompson G. [1990] „Reliable Transaction Management in a Multidatabase System”, w: *SIGMOD* [1990].
- Brinkhoff, T., Kriegel, H.-P., Seeger, B. [1993] „Efficient Processing of Spatial Joins Using R-trees”, w: *SIGMOD* [1993].
- Broder, A. [2002] „A Taxonomy of Web Search”, w: **SIGIR Forum**, 36:2, wrzesień 2002, s. 3 – 10.
- Brodeur, J., Bédard, Y., Proulx, M. [2000] „Modelling Geospatial Application Databases Using UML-Based Repositories Aligned with International Standards in Geomatics”, *Proc. 8th ACM International Symposium on Advances in Geographic Information Systems*, Waszyngton, DC, ACM Press, 2000, s. 39 – 46.
- Brodie M., Mylopoulos J., red. [1985] **On Knowledge Base Management Systems**, Springer-Verlag, 1985.
- Brodie M., Mylopoulos J., Schmidt J., red. [1984] **On Conceptual Modeling**, Springer-Verlag, 1984.
- Brosey M., Shneiderman B. [1978] „Two Experimental Comparisons of Relational and Hierarchical Database Models”, **International Journal of Man-Machine Studies**, 1978.
- Bruno, N., Chaudhuri, S., Gravano, L. [2002] „Top-k Selection Queries Over Relational Databases: Mapping Strategies and Performance Evaluation”, **ACM TODS**, 27:2, 2002, s. 153 – 187.
- Bry F. [1990] „Query Evaluation in Recursive Databases: Bottom-up and Top-down Reconciled”, **DKE**, 5, 1990, s. 289 – 312.
- Buckley, C., Salton, G., Allan, J. [1993] „The SMART Information Retrieval Project”, w: *Proc. of the Workshop on Human Language Technology*, Human Language Technology Conference, Association for Computational Linguistics, marzec 1993.
- Bukhres O. [1992] „Performance Comparison of Distributed Deadlock Detection Algorithms”, w: *ICDE* [1992].
- Buneman P., Frankel R. [1979] „FQL: A Functional Query Language”, w: *SIGMOD* [1979].
- Burkhard W. [1976] „Hashing and Trie Algorithms for Partial Match Retrieval”, **TODS**, 1:2, czerwiec 1976, s. 175 – 187.
- Burkhard W. [1979] „Partial-match Hash Coding: Benefits of Redunancy”, **TODS**, 4:2, czerwiec 1979, s. 228 – 239.
- Bush V. [1945] „As We May Think”, *Atlantic Monthly*, 176:1, styczeń 1945. Przedruk w Kochen M., red., **The Growth of Knowledge**, Wiley, 1967.
- Butterworth, P., Otis, A., Stein, J. [1991] „The Gemstone Object Database Management System”, w: **CACM**, 34:10, październik 1991, s. 64 – 77.
- Byte [1995] Special Issue on Mobile Computing, czerwiec 1995.
- CACM [1995] Special Issue of the **Communications of the ACM**, on Digital Libraries, 38:5, maj 1995.
- CACM [1998] Special Issue of the **Communications of the ACM** on Digital Libraries: Global Scope and Unlimited Access, 41:4, kwiecień 1998.

- Cahill, M.J., Rohm, U., Fekete, A. [2008] „Serializable Isolation for Snapshot Databases”, w: *SIGMOD* [2008].
- Cammarata S., Ramachandra P., Shane D. [1989] „Extending a Relational Database with Deferred Referential Integrity Checking and Intelligent Joins”, w: *SIGMOD* [1989].
- Campbell D., Embley D., Czejdo B. [1985] „A Relationally Complete Query Language for the Entity-Relationship Model”, w: *ER Conference* [1985].
- Cardenas A. [1985] **Data Base Management Systems**, wydanie drugie, Allyn and Bacon, 1985.
- Carey M. i in. [1986] „The Architecture of the EXODUS Extensible DBMS”, w: Dittrich, Dayal [1986].
- Carey M., DeWitt D., Vandenberg S. [1988] „A Data Model and Query Language for Exodus”, w: *SIGMOD* [1988].
- Carey M., DeWitt D., Richardson J., Shekita E. [1986a] „Object and File Management in the EXODUS Extensible Database System”, w: *VLDB* [1986].
- Carey M., Franklin M., Livny M., Shekita E. [1991] „Data Caching Tradeoffs in Client-Server DBMS Architectures”, w: *SIGMOD* [1991].
- Carey, M., Kossman, D. [1998] „Reducing the breaking distance of an SQL Query Engine”, w: *VLDB* [1998], s. 158 – 169.
- Carlis, J. [1986] „HAS, a Relational Algebra Operator or Divide Is Not Enough to Conquer”, w: *ICDE* [1986].
- Carlis J., March S. [1984] „A Descriptive Model of Physical Database Design Problems and Solutions”, w: *ICDE* [1984].
- Carneiro, G., Vasconcelos, N. [2005] „A Database Centric View of Semantic Image Annotation and Retrieval”, w: *SIGIR* [2005].
- Carroll J. M., [1995] **Scenario Based Design: Envisioning Work and Technology in System Development**, Wiley, 1995.
- Casanova M., Vidal V. [1982] „Toward a Sound View Integration Method”, w: *PODS* [1982].
- Casanova M., Fagin R., Papadimitriou C. [1981] „Inclusion Dependencies and Their Interaction with Functional Dependencies”, w: *PODS* [1981].
- Casanova M., Furtado A., Tuchermann L. [1991] „A Software Tool for Modular Database Design”, **TODS**, 16:2, czerwiec 1991.
- Casanova M., Tuchermann L., Furtado A., Braga, A. [1989] „Optimization of Relational Schemas Containing Inclusion Dependencies”, w: *VLDB* [1989].
- Castano S., DeAntonello V., Fugini M.G., Pernici B. [1998] „Conceptual Schema Analysis: Techniques and Applications”, **TODS**, 23:3, wrzesień 1998, s. 286 – 332.
- Catarci T., Costabile M. F., Levialdi S., Batini C. [1997] „Visual Query Systems for Databases: A Survey”, **Journal of Visual Languages and Computing**, 8:2, czerwiec 1997, s. 215 – 260.
- Catarci T., Costabile M. F., Santucci G., Tarantino L., red. [1998] *Proceedings of the Fourth International Workshop on Advanced Visual Interfaces*, ACM Press, 1998.
- Cattell, R. [1991] **Object Data Management: Object-Oriented and Extended Relational Database Systems**, Addison-Wesley, 1991.
- Cattell, R., Barry, D. K. [2000], **The Object Data Standard: ODMG 3.0**, Morgan Kaufmann, 2000.

- Cattell R., Skeen J. [1992] „Object Operations Benchmark”, **TODS**, 17:1, marzec 1992.
- Cattell R., red. [1993] **The Object Database Standard: ODMG-93, Release 1.2**, Morgan Kaufmann, 1993.
- Cattell R., red. [1997] **The Object Database Standard: ODMG, Release 2.0**, Morgan Kaufmann, 1997.
- Cattell, R. [2010] „Scalable SQL and NoSQL data stores”, **SIGMOD Record**, 39:4, 2010.
- Ceri S., Fraternali, P. [1997] **Designing Database Applications with Objects and Rules: The IDEA Methodology**, Addison-Wesley, 1997.
- Ceri S., Owicki, S. [1983] „On the Use of Optimistic Methods for Concurrency Control in Distributed Databases”, *Proceedings of the Sixth Berkeley Workshop on Distributed Data Management and Computer Networks*, luty 1983.
- Ceri S., Pelagatti G. [1984] „Correctness of Query Execution Strategies in Distributed Databases”, **TODS**, 8:4, grudzień 1984.
- Ceri S., Pelagatti G. [1984a] **Distributed Databases: Principles and Systems**, McGraw-Hill, 1984.
- Ceri S., Tanca L. [1987] „Optimization of Systems of Algebraic Equations for Evaluating Datalog Queries”, w: *VLDB* [1987].
- Ceri S., Gottlob G., Tanca L. [1990], **Logic Programming and Databases**, Springer-Verlag, 1990.
- Ceri S., Navathe S., Wiederhold G. [1983] „Distribution Design of Logical Database Schemas”, **TSE**, 9:4, lipiec 1983.
- Ceri S., Negri M., Pelagatti G. [1982] „Horizontal Data Partitioning in Database Design”, w: *SIGMOD* [1982].
- Cesarini F., Soda G. [1991] „A Dynamic Hash Method with Signature”, **TODS**, 16:2, czerwiec 1991.
- Chakrabarti, S. [2002] **Mining the Web: Discovering Knowledge from Hypertext Data**, Morgan-Kaufmann, 2002.
- Chakrabarti, S. i in. [1999] „Mining the Web’s Link Structure”, **Computer** 32:8, sierpień 1999, s. 60 – 67.
- Chakravarthy S. [1990] „Active Database Management Systems: Requirements, State-of-the-Art, and an Evaluation”, w: *ER Conference* [1990].
- Chakravarthy S. [1991] „Divide and Conquer: A Basis for Augmenting a Conventional Query Optimizer with Multiple Query Processing Capabilities”, w: *ICDE* [1991].
- Chakravarthy S. i in. [1989] „HiPAC: A Research Project in Active, Time Constrained Database Management”, Final Technical Report, XAIT-89-02, Xerox Advanced Information Technology, sierpień 1989.
- Chakravarthy S., Anwar E., Maugis L., Mishra D. [1994] Design of Sentinel: An Object-Oriented DBMS with Event-Based Rules, **Information and Software Technology**, 36:9, 1994.
- Chakravarthy S., Karlapalem K., Navathe S., Tanaka A. [1993] „Database Supported Co-operative Problem Solving”, w: **International Journal of Intelligent Cooperative Information Systems**, 2:3, wrzesień 1993.
- Chakravarthy U., Grant J., Minker J. [1990] „Logic-Based Approach to Semantic Query Optimization”, **TODS**, 15:2, czerwiec 1990.

- Chalmers M., Chitson P. [1992] „Bead: Explorations in Information Visualization”, *Proceedings of the ACM SIGIR International Conference*, czerwiec 1992.
- Chamberlin D. i in. [1976] „SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control”, **IBM Journal of Research and Development**, 20:6, listopad 1976.
- Chamberlin D. i in. [1981] „A History and Evaluation of System R”, **CACM**, 24:10, październik 1981.
- Chamberlin D., Boyce R. [1974] „SEQUEL: A Structured English Query Language”, w: *SIGMOD* [1984].
- Chan C., Ooi B., Lu H. [1992] „Extensible Buffer Management of Indexes”, w: *VLDB* [1992].
- Chandy K., Browne J., Dissley C., Uhrig W. [1975] „Analytical Models for Rollback and Recovery Strategies in Database Systems”, **TSE**, 1:1, marzec 1975.
- Chang C. [1981] „On the Evaluation of Queries Containing Derived Relations in a Relational Database”, w: Gallaire i in. [1981].
- Chang C., Walker A. [1984] „PROSQL: A Prolog Programming Interface with SQL/ DS”, w: *EDS* [1984].
- Chang E., Katz R. [1989] „Exploiting Inheritance and Structure Semantics for Effective Clustering and Buffering in Object-Oriented Databases”, w: *SIGMOD* [1989].
- Chang, F. i in. [2006] „Bigtable: A Distributed Storage System for Structured Data”, w: *OSDI* [2006].
- Chang N., Fu K. [1981] „Picture Query Languages for Pictorial Databases”, **IEEE Computer**, 14:11, listopad 1981.
- Chang P., Myre W. [1988] „OS/2 EE Database Manager: Overview and Technical Highlights”, **IBM Systems Journal**, 27:2, 1988.
- Chang S., Lin B., Walser R. [1979] „Generalized Zooming Techniques for Pictorial Database Systems”, *NCC, AFIPS*, 48, 1979.
- Chatzoglou P. D., McCaulay L. A. [1997] „Requirements Capture and Analysis: A Survey of Current Practice”, **Requirements Engineering**, 1997, s. 75 – 88.
- Chaudhri, A., Rashid, A., Zicari, R. red. [2003] **XML Data Management: Native XML and XML-Enabled Database Systems**, Addison-Wesley, 2003.
- Chaudhuri S., Dayal U. [1997] „An Overview of Data Warehousing and OLAP Technology”, **SIGMOD Record**, tom 26, nr 1, marzec 1997.
- Chaudhuri, S., Shim, K. [1994] „Including Group-By in Query Optimization”, w: *VLDB* [1994].
- Chaudhuri, S. i in. [1995] „Optimizing Queries with Materialized Views”, w: *ICDE* [1995].
- Chen M., Yu P. [1991] „Determining Beneficial Semijoins for a Join Sequence in Distributed Query Processing”, w: *ICDE* [1991].
- Chen M., Han J., Yu P. S., [1996] „Data Mining: An Overview from a Database Perspective”, **TKDE**, 8:6, grudzień 1996.
- Chen P. [1976] „The Entity Relationship Mode — Toward a Unified View of Data”, **TODS**, 1:1, marzec 1976.
- Chen P., Patterson D. [1990]. „Maximizing performance in a striped disk array”, w: *Proceedings of Symposium on Computer Architecture, IEEE*, Nowy Jork, 1990.
- Chen P. i in. [1994] RAID High Performance, Reliable Secondary Storage, **ACM Computing Surveys**, 26:2, 1994.

- Chen Q., Kambayashi Y. [1991] „Nested Relation Based Database Knowledge Representation”, w: *SIGMOD* [1991].
- Cheng J. [1991] „Effective Clustering of Complex Objects in Object-Oriented Databases”, w: *SIGMOD* [1991].
- Cheung D. i in. [1996] „A Fast and Distributed Algorithm for Mining Association Rules”, w: *Proceedings of International Conference on Parallel and Distributed Information Systems*, PDIS [1996].
- Childs D. [1968] „Feasibility of a Set Theoretical Data Structure — A General Structure Based on a Reconstituted Definition of Relation”, *Proceedings of the IFIP Congress*, 1968.
- Chimenti D. i in. [1987] „An Overview of the LDL System”, **IEEE Data Engineering Bulletin**, 10:4, 1987, s. 52 – 62.
- Chimenti D. i in. [1990] „The LDL System Prototype”, **TKDE**, 2:1, marzec 1990.
- Chin F. [1978] „Security in Statistical Databases for Queries with Small Counts”, **TODS**, 3:1, marzec 1978.
- Chin F., Ozsoyoglu G. [1981] „Statistical Database Design”, **TODS**, 6:1, marzec 1981.
- Chintalapati R., Kumar V., Datta A. [1997] „An Adaptive Location Management Algorithm for Mobile Computing”, *Proceedings of 22nd Annual Conference on Local Computer Networks (LCN '97)*, Minneapolis, 1997.
- Chou, H.-T., DeWitt, D. [1985] „An Evaluation of Buffer Management Strategies or Relational Databases”, *VLDB* [1985], s. 127 – 141.
- Chou, H.-T., Kim W. [1986] „A Unifying Framework for Version Control in a CAD Environment”, w: *VLDB* [1986].
- Christodoulakis S. i in. [1984] „Development of a Multimedia Information System for an Office Environment”, w: *VLDB* [1984].
- Christodoulakis S., Faloutsos C. [1986] „Design and Performance Considerations for an Optical Disk-Based Multimedia Object Server”, **IEEE Computer**, 19:12, grudzień 1986.
- Chrysanthos, P. [1993] „Transaction Processing in a Mobile Computing Environment”, *Proc. IEEE Workshop on Advances in Parallel and Distributed Systems*, październik 1993, s. 77 – 82.
- Chu W., Hurley P. [1982] „Optimal Query Processing for Distributed Database Systems”, **IEEE Transactions on Computers**, 31:9, wrzesień 1982.
- Ciborra C., Migliarese P., Romano P. [1984] „A Methodological Inquiry of Organizational Noise in Socio Technical Systems”, **Human Relations**, 37:8, 1984.
- CISCO [2014] Accelerate Application Performance with the Cisco UCS Invicta Series, dokument firmy CISCO, styczeń 2014.
- Claybrook B. [1992] **File Management Techniques**, Wiley, 1992.
- Claybrook B. [1992] **OLTP: OnLine Transaction Processing Systems**, Wiley, 1992.
- Clementini, E., Di Felice, P. [2000] „Spatial Operators”, w: **SIGMOD Record** 29:3, 2000, s. 31 – 38.
- Clifford J., Tansel A. [1985] „On an Algebra for Historical Relational Databases: Two Views”, w: *SIGMOD* [1985].
- Clocksin W. F., Mellish C. S. [2004] **Programming in Prolog: Using the ISO Standard**, wydanie piąte, Springer-Verlag, 2003.



- Cloudera Inc. [2014] „Impala Performance Update: Now Reaching DBMS-Class Speed”, Justin Erickson i in. (<http://blog.cloudera.com/blog/2014/01/impala-performance-dbms-class-speed/>), styczeń 2014.
- Cockcroft, S. [1997] „A Taxonomy of Spatial Data Integrity Constraints”, *GeoInformatica*, 1997, s. 327 – 343.
- CODASYL [1978] Data Description Language Journal of Development, Canadian Government Publishing Centre, 1978.
- Codd E. [1970] „A Relational Model for Large Shared Data Banks”, **CACM**, 13:6, czerwiec 1970.
- Codd E. [1971] „A Data Base Sublanguage Founded on the Relational Calculus”, *Proceedings of the ACM SIGFIDET Workshop on Data Description, Access, and Control*, listopad 1971.
- Codd E. [1972] „Relational Completeness of Data Base Sublanguages”, w: Rustin [1972].
- Codd E. [1972a] „Further Normalization of the Data Base Relational Model”, w: Rustin [1972].
- Codd E. [1974] „Recent Investigations in Relational Database Systems”, *Proceedings of the IFIP Congress*, 1974.
- Codd E. [1978] „How About Recently? (English Dialog with Relational Data Bases Using Rendezvous Version 1)”, w: Shneiderman [1978].
- Codd E. [1979] „Extending the Database Relational Model to Capture More Meaning”, **TODS**, 4:4, grudzień 1979.
- Codd E. [1982] „Relational Database: A Practical Foundation for Productivity”, **CACM**, 25:2, grudzień 1982.
- Codd E. [1985] „Is Your DBMS Really Relational?” oraz „Does Your DBMS Run By the Rules?”, **COMPUTER WORLD**, 14 października oraz 21 października, 1985.
- Codd E. [1986] „An Evaluation Scheme for Database Management Systems That Are Claimed to Be Relational”, w: *ICDE* [1986].
- Codd E. [1990] **Relational Model for Data Management-Version 2**, Addison-Wesley, 1990.
- Codd E. F., Codd S. B., Salley C. T. [1993] „Providing OLAP (On-Line Analytical Processing) to User Analyst: An IT Mandate”, dokument dostępny pod adresem <https://pdfs.semanticscholar.org/0a93/e70589fbeb43edf65de61ffb6cd3696c4a.pdf>, 1993.
- Comer D. [1979] „The Ubiquitous B-tree”, **ACM Computing Surveys**, 11:2, czerwiec 1979.
- Comer D. [2008] **Computer Networks and Internets**, wydanie piąte, Prentice-Hall, 2008.
- Cooley, R. [2003] „The Use of Web Structure and Content to Identify Subjectively Interesting Web Usage Patterns”, **ACM Trans. On Internet Technology**, 3:2, maj 2003, s. 93 – 116.
- Cooley, R., Mobasher, B., Srivastava, J. [1997] „Web Mining: Information and Pattern Discovery on the World Wide Web”, w: *Proc. Ninth IEEE Int. Conf. on Tools with Artificial Intelligence (ICTAI)*, listopad 1997, s. 558 – 567.
- Cooley, R., Mobasher, B., Srivastava, J. [2000] „Automatic personalization based on Web usage mining”, **CACM**, 43:8, sierpień 2000.
- Corcho, C., Fernandez-Lopez, M., Gomez-Perez, A. [2003] „Methodologies, Tools and Languages for Building Ontologies. Where Is Their Meeting Point?”, **DKE**, 46:1, lipiec 2003.
- Cormen, T., Leiserson, C., Rivest, R. [1990] **Introduction to Algorithms**, MIT Press, 1990.

- Cornelio A., Navathe S. [1993] „Applying Active Database Models for Simulation”, w: *Proceedings of 1993 Winter Simulation Conference*, IEEE, Los Angeles, grudzień 1993.
- Corson, S., Macker, J. [1999] „Mobile Ad-Hoc Networking: Routing Protocol Performance Issues and Performance Considerations”, IETF Request for Comments No. 2501, styczeń 1999, tekst dostępny na stronie <http://www.ietf.org/rfc/rfc2501.txt>.
- Cosmadakis S., Kanellakis P. C., Vardi M. [1990] „Polynomial-time Implication Problems for Unary Inclusion Dependencies”, **JACM**, 37:1, 1990, s. 15 – 46.
- Covi, L., Kling, R. [1996] „Organizational Dimensions of Effective Digital Library Use: Closed Rational and Open Natural Systems Models”, **Journal of American Society of Information Science (JASIS)**, 47:9, 1996, s. 672 – 689.
- Croft, B., Metzler, D., Strohmman, T. [2009] **Search Engines: Information Retrieval in Practice**, AddisonWesley, 2009.
- Cruz I. [1992] „Doodle: A Visual Language for Object-Oriented Databases”, w: *SIGMOD* [1992].
- Curtice R. [1981] „Data Dictionaries: An Assessment of Current Practice and Problems”, w: *VLDB* [1981].
- Cuticchia A., Fasman K., Kingsbury D., Robbins R., Pearson P. [1993] „The GDB Human Genome Database Anno 1993”. **Nucleic Acids Research**, 21:13, 1993.
- Czejdo B., Elmasri R., Rusinkiewicz M., Embley D. [1987] „An Algebraic Language for Graphical Query Formulation Using an Extended Entity-Relationship Model”, *Proceedings of the ACM Computer Science Conference*, 1987.
- Dahl R., Bubenko J. [1982] „IDBD: An Interactive Design Tool for CODASYL DBTG Type Databases”, w: *VLDB* [1982].
- Dahl V. [1984] „Logic Programming for Constructive Database Systems,” w *EDS* [1984].
- Dahl V. [1984] „Logic Programming for Constructive Database Systems”, w: *EDS* [1984].
- Das S. [1992] **Deductive Databases and Logic Programming**, Addison-Wesley, 1992.
- Das, S., Antony, S., Agrawal, D. i in. [2008] „Clouded Data: Comprehending Scalable Data Management Systems”, **UCSB CS Technical Report** 2008-18, listopad 2008.
- Date C. J. [1983] **An Introduction to Database Systems**, tom 2, Addison-Wesley, 1983.
- Date C. J. [1983a] „The Outer Join”, *Proceedings of the Second International Conference on Databases (ICOD-2)*, 1983.
- Date C. J. [1984] „A Critique of the SQL Database Language”, **ACM SIGMOD Record**, 14:3, listopad 1984.
- Date, C. J. [2001] **The Database Relational Model: A Retrospective Review and Analysis: A Historical Account and Assessment of E. F. Codd's Contribution to the Field of Database Technology**, Addison-Wesley, 2001.
- Date C. J. [2004] **An Introduction Database Systems**, wydanie ósme, Addison-Wesley, 2004.
- Date C. J., Darwen H. [1993] **A Guide to the SQL Standard**, wydanie trzecie, Addison-Wesley.
- Date C. J., Fagin, R. [1992] „Simple Conditions for Guaranteeing Higher Normal Forms in Relational Databases”, **TODS**, 17:3, 1992.
- Date C., White C. [1988] **A Guide to SQL/DS**, Addison-Wesley, 1988.

- Date C., White C. [1989] **A Guide to DB2**, wydanie trzecie, Addison-Wesley, 1989.
- Davies C. [1973] „Recovery Semantics for a DB/DC System”, *Proceedings of the ACM National Conference*, 1973.
- Dayal U. i in. [1987] „PROBE Final Report”, Technical Report CCA-87-02, Computer Corporation of America, grudzień 1987.
- Dayal U., Bernstein P. [1978] „On the Updatability of Relational Views”, w: *VLDB* [1978].
- Dayal U., Hsu M., Ladin R. [1991] „A Transaction Model for Long-Running Activities”, w: *VLDB* [1991].
- DBTG [1971] **Report of the CODASYL Data Base Task Group**, ACM, kwiecień 1971.
- DeCandia, G. et al. [2007] „Dynamo: Amazon’s Highly Available Key-Value Store,” w *SOSP*, 2007.
- Deelman, E., Chervenak, A. L. [2008] „Data Management Challenges of Data-Intensive Scientific Workflows”, w: *Proc. IEEE International Symposium on Cluster, Cloud, and Grid Computing*, 2008, s. 687 – 692.
- Delcambre L., Lim B., Urban S. [1991] „Object-Centered Constraints”, w: *ICDE* [1991].
- DeMarco T. [1979] **Structured Analysis and System Specification**, Prentice-Hall, 1979.
- DeMers, M. [2002] **Fundamentals of GIS**, John Wiley, 2002.
- DeMichiel L. [1989] „Performing Operations Over Mismatched Domains”, w: *ICDE* [1989].
- Denning D. [1980] „Secure Statistical Databases with Random Sample Queries”, **TODS**, 5:3, wrzesień 1980.
- Denning D., Denning P. [1979] „Data Security”, **ACM Computing Surveys**, 11:3, wrzesień 1979, s. 227 – 249.
- Denning, D. i in. [1987] „A Multi-level Relational Data Model”, w: *Proc. IEEE Symp. On Security and Privacy*, 1987, s. 196 – 201.
- Deshpande A. [1989] „An Implementation for Nested Relational Databases”, Technical Report, praca doktorska, Indiana University, 1989.
- Devor C., Weeldreyer J. [1980] „DDTS: A Testbed for Distributed Database Research”, *Proceedings of the ACM Pacific Conference*, 1980.
- DeWitt D. i in. [1984] „Implementation Techniques for Main Memory Databases”, w: *SIGMOD* [1984].
- DeWitt D. i in. [1990] „The Gamma Database Machine Project”, **TKDE**, 2:1, marzec 1990.
- DeWitt D., Futersack P., Maier D., Velez F. [1990] „A Study of Three Alternative Workstation Server Architectures for Object-Oriented Database Systems”, w: *VLDB* [1990].
- Dhawan C. [1997] **Mobile Computing**, McGraw-Hill, 1997.
- Di, S. M. [2005] **Distributed Data Management in Grid Environments**, Wiley, 2005.
- Dietrich, B. L. i in. [2014] **Analytics Across the Enterprise: How IBM Realizes Business Value from Big Data and Analytics**, IBM Press (Pearson plc), 2014, 192 s.
- Dietrich S., Friesen O., Calliss W. [1999] „On Deductive and Object Oriented Databases: The VALIDITY Experience”, raport techniczny, Arizona State University, 1999.
- Diffie W., Hellman M. [1979] „Privacy and Authentication”, **Proceedings of the IEEE**, 67:3, marzec 1979, s. 397 – 429.

Dimitrova, N. [1999] „Multimedia Content Analysis and Indexing for Filtering and Retrieval Applications”, **Information Science**, specjalny numer poświęcony informatycznym technologiom multimedialnym, część 1, 2:4, 1999.

Dipert B., Levy M. [1993] **Designing with Flash Memory**, Annabooks, 1993.

Dittrich K. [1986] „Object-Oriented Database Systems: The Notion and the Issues”, w: Dittrich, Dayal [1986].

Dittrich K., Dayal U., red. [1986] *Proceedings of the International Workshop on Object-Oriented Database Systems*, IEEE CS, Pacific Grove, CA, wrzesień 1986.

Dittrich K., Kotz A., Mulle J. [1986] „An Event/Trigger Mechanism to Enforce Complex Consistency Constraints in Design Databases”, w: **ACM SIGMOD Record**, 15:3, 1986.

DKE [1997] Special Issue on Natural Language Processing, **DKE**, 22:1, 1997.

Dodd G. [1969] „APL — A Language for Associative Data Handling in PL/I”, *Proceedings of the Fall Joint Computer Conference*, AFIPS, 29, 1969.

Dodd G. [1969] „Elements of Data Management Systems”, **ACM Computing Surveys**, 1:2, czerwiec 1969.

Dogac A., [1998] Specjalna sekcja poświęcona handlowi elektronicznemu, **ACM Sigmod Record** 27:4, grudzień 1998.

Dogac A., Ozsu M. T., Bilins A., Sellis T., red. [1994] **Advances in Object-Oriented Database Systems**, NATO ASI Series. Series F: Computer and Systems Sciences, tom 130, Springer-Verlag, 1994.

Dos Santos C., Neuhold E., Furtado A. [1979] „A Data Type Approach to the Entity-Relationship Model”, w: *ER Conference* [1979].

Du D., Tong S. [1991] „Multilevel Extendible Hashing: A File Structure for Very Large Databases”, **TKDE**, 3:3, wrzesień 1991.

Du H., Ghanta S. [1987] „A Framework for Efficient IC/VLSI CAD Databases”, w: *ICDE* [1987].

Dumas P. i in. [1982] „MOBILE-Burotique: Prospects for the Future”, w: Naffah [1982].

Dumpala S., Arora S. [1983] „Schema Translation Using the Entity-Relationship Approach”, w: *ER Conference* [1983].

Dunham M., Helal A. [1995] „Mobile Computing and Databases: Anything New?”, **ACM SIGMOD Record**, 24:4, grudzień 1995.

Dwyer S. i in. [1982] „A Diagnostic Digital Imaging System”, *Proceedings of the IEEE CS Conference on Pattern Recognition and Image Processing*, czerwiec 1982.

Eastman C. [1987] „Database Facilities for Engineering Design”, **Proceedings of the IEEE**, 69:10, październik 1981.

EDS [1984] **Expert Database Systems**, Kerschberg L., red. (*Proceedings of the First International Workshop on Expert Database Systems*, Kiawah Island, SC, październik 1984), Benjamin/Cummings, 1986.

EDS [1986] **Expert Database Systems**, Kerschberg L., w: (*Proceedings of the First International Conference on Expert Database Systems*, Charleston, SC, kwiecień 1986), Benjamin/Cummings, 1987.

EDS [1988] **Expert Database Systems**, Kerschberg L., red. (*Proceedings of the Second International Conference on Expert Database Systems*, Tysons Corner, VA, kwiecień 1988), Benjamin/Cummings, 1989.

- Eick C. [1991] „A Methodology for the Design and Transformation of Conceptual Schemas”, w: *VLDB* [1991].
- ElAbbadi A., Toueg S. [1988] „The Group Paradigm for Concurrency Control”, w: *SIGMOD* [1988].
- ElAbbadi A., Toueg S. [1989] „Maintaining Availability in Partitioned Replicated Databases”, **TODS**, 14:2, czerwiec 1989.
- Ellis C., Nutt G. [1980] „Office Information Systems and Computer Science”, **ACM Computing Surveys**, 12:1, marzec 1980.
- Elmagarmid A. K., red. [1992] **Database Transaction Models for Advanced Applications**, Morgan Kaufmann, 1992.
- Elmagarmid, A., Helal, A. [1988] „Supporting Updates in Heterogeneous Distributed Database Systems”, w: *ICDE* [1988], s. 564 – 569.
- Elmagarmid A., Leu Y., Litwin W., Rusinkiewicz M. [1990] „A Multidatabase Transaction Model for Interbase”, w: *VLDB* [1990].
- Elmasri R., Larson J. [1985] „A Graphical Query Facility for ER Databases”, w: *ER Conference* [1985].
- Elmasri R., Wiederhold G. [1979] „Data Model Integration Using the Structural Model”, w: *SIGMOD* [1979].
- Elmasri R., Wiederhold G. [1980] „Structural Properties of Relationships and Their Representation”, *NCC, AFIPS*, 49, 1980.
- Elmasri R., Wiederhold G. [1981] „GORDAS: A Formal, High-Level Query Language for the Entity-Relationship Model”, w: *ER Conference* [1981].
- Elmasri R., Wu G. [1990] „A Temporal Model and Query Language for ER Databases”, w: *ICDE* [1990].
- Elmasri R., Wu G. [1990a] „The Time Index: An Access Structure for Temporal Data”, w: *VLDB* [1990].
- Elmasri R., James S., Kouramajian V. [1993] „Automatic Class and Method Generation for Object-Oriented Databases”, *Proceedings of the Third International Conference on Deductive and Object-Oriented Databases (DOOD-93)*, Phoenix, AZ, grudzień 1993.
- Elmasri R., Kouramajian V., Fernando S. [1993] „Temporal Database Modeling: An Object-Oriented Approach”, *CIKM*, listopad 1993.
- Elmasri R., Larson J., Navathe S. [1986] „Schema Integration Algorithms for Federated Databases and Logical Database Design”, Honeywell CSDD, raport techniczny CSC-86-9: 8212, styczeń 1986.
- Elmasri R., Srinivas P., Thomas G. [1987] „Fragmentation and Query Decomposition in the ECR Model”, w: *ICDE* [1987].
- Elmasri R., Weeldreyer J., Hevner A. [1985] „The Category Concept: An Extension to the Entity-Relationship Model”, **International Journal on Data and Knowledge Engineering**, 1:1, maj 1985.
- Engelbart D., English W. [1968] „A Research Center for Augmenting Human Intellect”, *Proceedings of the Fall Joint Computer Conference*, AFIPS, grudzień 1968.
- Epstein R., Stonebraker M., Wong E. [1978] „Distributed Query Processing in a Relational Database System”, w: *SIGMOD* [1978].

ER Conference [1979] **Entity-Relationship Approach to Systems Analysis and Design**, Chen P., red. (*Proceedings of the First International Conference on Entity-Relationship Approach*, Los Angeles, grudzień 1979), North-Holland, 1980.

ER Conference [1981] **Entity-Relationship Approach to Information Modeling and Analysis**, Chen P., red. (*Proceedings of the Second International Conference on Entity-Relationship Approach*, Washington, październik 1981), Elsevier Science, 1981.

ER Conference [1983] **Entity-Relationship Approach to Software Engineering**, Davis C., Jajodia S., Ng P., Yeh R., red. (*Proceedings of the Third International Conference on Entity-Relationship Approach*, Anaheim, CA, październik 1983), North-Holland, 1983.

ER Conference [1985] *Proceedings of the Fourth International Conference on Entity Relationship Approach*, Liu J., red, Chicago, październik 1985, IEEE CS.

ER Conference [1986] *Proceedings of the Fifth International Conference on Entity-Relationship Approach*, Spaccapietra S., red., Dijon, Francja, listopad 1986, Express-Tirages.

ER Conference [1987] *Proceedings of the Sixth International Conference on Entity-Relationship Approach*, March S., red., Nowy Jork, listopad 1987.

ER Conference [1988] *Proceedings of the Seventh International Conference on Entity-Relationship Approach*, Batini C., red., Rzym, listopad 1988.

ER Conference [1989] *Proceedings of the Eighth International Conference on Entity-Relationship Approach*, Lochovsky F., red., Toronto, październik 1989.

ER Conference [1990] *Proceedings of the Ninth International Conference on Entity-Relationship Approach*, Kangassalo H., red., Lozanna, Szwajcaria, wrzesień 1990.

ER Conference [1991] *Proceedings of the Tenth International Conference on Entity-Relationship Approach*, Teorey T., red., San Mateo, CA, październik 1991.

ER Conference [1992] *Proceedings of the Eleventh International Conference on Entity-Relationship Approach*, Pernul G., Tjoa A., red., Karlsruhe, Niemcy, październik 1992.

ER Conference [1993] *Proceedings of the Twelfth International Conference on Entity-Relationship Approach*, Elmasri R., Kouramajian V., red., Arlington, TX, grudzień 1993.

ER Conference [1994] *Proceedings of the Thirteenth International Conference on Entity-Relationship Approach*, Loucopoulos P., Theodoulidis B., red., Manchester, Anglia, grudzień 1994.

ER Conference [1995] *Proceedings of the Fourteenth International Conference on ER-OO Modeling*, Papazougrou M., Tari Z., red., Brisbane, Australia, grudzień 1995.

ER Conference [1996] *Proceedings of the Fifteenth International Conference on Conceptual Modeling*, Thalheim B., red., Cottbus, Niemcy, październik 1996.

ER Conference [1997] *Proceedings of the Sixteenth International Conference on Conceptual Modeling*, Embley D., red., Los Angeles, październik 1997.

ER Conference [1998] *Proceedings of the Seventeenth International Conference on Conceptual Modeling*, Ling T.-K., red., Singapur, listopad 1998.

ER Conference [1999] *Proc. Eighteenth Conference on Conceptual Modeling*, Akoka, J., Bouzeghoub, M., ComynWattiau, I., Métais, E. (red.): Paryż, Francja, **LNCS 1728**, Springer, 1999.

ER Conference [2000] *Proc. Nineteenth Conference on Conceptual Modeling*, Laender, A., Liddle, S., Storey, V. (red.), Salt Lake City, **LNCS 1920**, Springer, 2000.



- ER Conference [2001] *Proc. Twentieth Conference on Conceptual Modeling*, Kunii, H., Jajodia, S., Solveberg, A. (red.), Yokohama, Japonia, **LNCS** 2224, Springer, 2001.
- ER Conference [2002] *Proc. 21st Int. Conference on Conceptual Modeling*, Spaccapietra, S., March, S., Kambayashi, Y. (red.), Tampere, Finland, **LNCS** 2503, Springer, 2002.
- ER Conference [2003] *Proc. 22nd Int. Conference on Conceptual Modeling*, Song, I.-Y., Liddle, S., Ling, T.-W., Scheuermann, P. (red.), Tampere, Finlandia, **LNCS** 2813, Springer, 2003.
- ER Conference [2004] *Proc. 23rd Int. Conference on Conceptual Modeling*, Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T.-W. (red.), Szanghaj, Chiny, **LNCS** 3288, Springer, 2004.
- ER Conference [2005] *Proc. 24th Int. Conference on Conceptual Modeling*, Delacambre, L.M.L., Kop, C., Mayr, H., Mylopoulos, J., Pastor, O. (red.), Klagenfurt, Austria, **LNCS** 3716, Springer, 2005.
- ER Conference [2006] *Proc. 25th Int. Conference on Conceptual Modeling*, Embley, D., Olive, A., Ram, S. (red.), Tucson, AZ, **LNCS** 4215, Springer, 2006.
- ER Conference [2007] *Proc. 26th Int. Conference on Conceptual Modeling*, Parent, C., Schewe, K.-D., Storey, V., Thalheim, B. (red.), Auckland, Nowa Zelandia, **LNCS** 4801, Springer, 2007.
- ER Conference [2008] *Proc. 27th Int. Conference on Conceptual Modeling*, Li, Q., Spaccapietra, S., Yu, E. S. K., Olive, A. (red.), Barcelona, Hiszpania, **LNCS** 5231, Springer, 2008.
- ER Conference [2009] *Proc. 28th Int. Conference on Conceptual Modeling*, Laender, A., Castano, S., Dayal, U., Casati, F., de Oliveira (red.), Gramado, RS, Brazylia, **LNCS** 5829, Springer, 2009.
- ER Conference [2010] *Proc. 29th Int. Conference on Conceptual Modeling*, Parsons, J. et al. (red.), Vancouver, Kanada, **LNCS** 6412, Springer, 2010.
- ER Conference [2011] *Proc. 30th Int. Conference on Conceptual Modeling*, Jeusfeld, M. Delcambre, L., Ling, Tok Wang (red.), Bruksela, Belgia, **LNCS** 6998, Springer, 2011.
- ER Conference [2012] *Proc. 31st Int. Conference on Conceptual Modeling*, Atzeni, P., Cheung, D.W., Ram, Sudha (eds.), Florencja, Włochy, **LNCS** 7532, Springer, 2012.
- ER Conference [2013] *Proc. 32nd Int. Conference on Conceptual Modeling*, Ng, Wilfred, Storey, V., Trujillo, J. (red.), Hong Kong, Chiny, **LNCS** 8217, Springer, 2013.
- ER Conference [2014] *Proc. 33rd Int. Conference on Conceptual Modeling*, Yu, Eric, Dobbie, G., Jarke, M., Purao, S. (red.), Atlanta, Stany Zjednoczone, **LNCS** 8824, Springer, 2014.
- ER Conference [2015] *Proc. 34th Int. Conference on Conceptual Modeling*, Sztokholm, Szwecja, **LNCS**, Springer, w przygotowaniu.
- Erl, T. i in. [2013] **Cloud Computing: Concepts, Technology and Architecture**, Prentice Hall, 2013, 489 s.
- ESRI [2009] "The Geodatabase: Modeling and Managing Spatial Data", w: **ArcNews**, 30:4, ESRI, zima 2008/2009.
- Ester, M., Kriegel, H.-P., Jorg, S. [2001] „Algorithms and Applications for Spatial Data Mining”, w: **Research Monograph in GIS**, CRC Press [2001].
- Ester, M., Kriegel, H.-P., Sander, J., Xu, X. [1996] „A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”, w: *KDD*, 1996, AAAI Press, s. 226 – 231.
- Eswaran K., Chamberlin D. [1975] „Functional Specifications of a Subsystem for Database Integrity”, w: *VLDB* [1975].
- Eswaran K., Gray J., Lorie R., Traiger I. [1976] „The Notions of Consistency and Predicate Locks in a Data Base System”, **CACM**, 19:11, listopad 1976.



Etzioni, O. [1996] "The World-Wide Web: quagmire or gold mine?", **CACM**, 39:11, listopad 1996, s. 65 – 68.

Everett G., Dissly C., Hardgrave W. [1971] **RFMS User Manual**, TRM-16, Computing Center, University of Texas at Austin, 1981.

Fagin R. [1977] „Multivalued Dependencies and a New Normal Form for Relational Databases”, **TODS**, 2:3, wrzesień 1977.

Fagin R. [1979] „Normal Forms and Relational Database Operators”, w: *SIGMOD* [1979].

Fagin R. [1981] „A Normal Form for Relational Databases That Is Based on Domains and Keys”, **TODS**, 6:3, wrzesień 1981.

Fagin R., Nievergelt J., Pippenger N., Strong H. [1979] „Extendible Hashing — A Fast Access Method for Dynamic Files”, **TODS**, 4:3, wrzesień 1979.

Falcone S., Paton, N. [1997] „Deductive Object-Oriented Database Systems: A Survey”, *Proceedings of the 3rd International Workshop Rules in Database Systems (RIDS'97)*, Skovde, Szwecja, czerwiec 1997.

Faloutsos C. [1996] **Searching Multimedia Databases by Content**, Kluwer, 1996.

Faloutsos C. i in. [1994] „Efficient and effective querying by image content”, w: **Journal of Intelligent Information Systems**, 3:4, 1994.

Faloutsos C., Jagadish H. [1992] „On B-Tree Indices for Skewed Distributions”, w: *VLDB* [1992].

Fan, J., Gao, Y., Luo, H., Xu, G. [2004] „Automatic Image Annotation by Using Concept-sensitive Salient Objects for Image Content Representation”, w: *SIGIR*, 2004.

Farag W., Teorey, T. [1993] „FunBase: A Function-Based Information Management System”, *CIKM*, listopad 1993.

Farahmand, F., Navathe, S., Sharp, G., Enslow, P. [2003] „Managing Vulnerabilities of Information Systems to Security Incidents”, *Proc. ACM 5th International Conference on Electronic Commerce*, ICEC 2003, Pittsburgh, PA, wrzesień 2003, s. 348 – 354.

Farahmand, F., Navathe, S., Sharp, G., Enslow, P. [2005] „A Management Perspective on Risk of Security Threats to Information Systems”, **Journal of Information Technology & Management**, tom 6, s. 203 – 225, 2005.

Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R. [1997] **Advances in Knowledge Discovery and Data Mining**, MIT Press, 1997.

Fekete, A., O'Neil, E., O'Neil, P. [2004] „A Read-only Transaction Anomaly Under Snapshot Isolation”, **SIGMOD Record**, 33:3, 2004, s. 12 – 14.

Fekete, A. i in. [2005] „Making Snapshot Isolation Serializable”, **ACM TODS**, 30:2, 2005, s. 492 – 528.

Fellbaum, C., red. [1998] **WordNet: An Electronic Lexical Database**, MIT Press, 1998.

Fensel, D. [2000] „The Semantic Web and Its Languages”, **IEEE Intelligent Systems**, tom 15, nr 6, listopad/grudzień 2000, s. 67 – 73.

Fensel, D. [2003] **Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce**, wydanie drugie, Springer-Verlag, Berlin, 2003.

Fernandez E., Summers R., Wood C. [1981] **Database Security and Integrity**, Addison-Wesley, 1981.

Ferrier A., Stangret, C. [1982] „Heterogeneity in the Distributed Database Management System SIRIUS-DELTA”, w: *VLDB* [1982].

- Ferrucci, D. i in. [2010] „Building Watson: An overview of the DeepQA project”, **AI Magazine** 31:3, 2010, s. 59 – 79.
- Fishman D. i in. [1986] „IRIS: An Object-Oriented DBMS”, **TOOIS**, 4:2, kwiecień 1986.
- Flickner, M. i in. [1995] „Query by Image and Video Content: The QBIC System”, **IEEE Computer**, 28:9, wrzesień 1995, s. 23 – 32.
- Flynn, J., Pitts, T. [2000] **Inside ArcINFO 8**, wydanie drugie, On Word Press, 2000.
- Folk M. J., Zoellick B., Riccardi G. [1998] **File Structures: An Object Oriented Approach with C++**, wydanie trzecie, Addison-Wesley, 1998.
- Fonseca, F., Egenhofer, M., Davis, C., Câmara, G. [2002] „Semantic Granularity in Ontology-Driven Geographic Information Systems”, w: **Annals of Mathematics and Artificial Intelligence** 36:1 – 2, s. 121 – 151.
- Ford D., Christodoulakis S. [1991] „Optimizing Random Retrievals from CDV Format Optical Disks”, w: **VLDB** [1991].
- Ford D., Blakeley J., Bannon T. [1993] „Open OODB: A Modular Object-Oriented DBMS”, w: **SIGMOD** [1993].
- Foreman G., Zahorjan J. [1994] „The Challenges of Mobile Computing” **IEEE Computer**, kwiecień 1994.
- Fotouhi, F., Grosky, W., Stanchev, P. (red.) [2007] *Proc. of the First ACM Workshop on Many Faces of the Multimedia Semantics, MS 2007*, Augsburg, Niemcy, wrzesień 2007.
- Fowler M., Scott K. [2000] **UML distilled**, wydanie drugie, Addison-Wesley, 2000.
- Franaszek P., Robinson J., Thomasian A. [1992] „Concurrency Control for High Contention Environments”, **TODS**, 17:2, czerwiec 1992.
- Frank, A. [2003] „A linguistically justified proposal for a spatio-temporal ontology”, w: *Proc. COSIT03- Int. Conf. on Spatial Information Theory*, Ittingen, Szwajcaria, **LNCS** 2825, wrzesień 2003.
- Franklin F. i in. [1992] „Crash Recovery in Client-Server EXODUS”, w: **SIGMOD** [1992].
- Franks, B. [2012] **Taming the Big Data Tidal Wave**, Wiley, 2012, 294 s.
- Fraternali P. [1999] Tools and Approaches for Data Intensive Web Applications: A Survey, *ACM Computing Surveys*, 31:3, wrzesień 1999.
- Frenkel K. [1991] „The Human Genome Project and Informatics”, **CACM**, listopad 1991.
- Friesen O., Gauthier-Villars G., Lefelorre A., Vieille L., „Applications of Deductive Object-Oriented Databases Using DEL”, w: Ramakrishnan (1995).
- Friis-Christensen, A., Tryfona, N., Jensen, C. S. [2001] „Requirements and Research Issues in Geographic Data Modeling”, *Proc. 9th ACM International Symposium on Advances in Geographic Information Systems*, 2001.
- Fugini, M., Castano, S., Martella G., Samarati, P. [1995] **Database Security**, ACM Press i Addison-Wesley, 1995.
- Furtado A. [1978] „Formal Aspects of the Relational Model”, **Information Systems**, 3:2, 1978.
- Gadia S. [1988] „A Homogeneous Relational Model and Query Language for Temporal Databases”, **TODS**, 13:4, grudzień 1988.
- Gait J. [1988] „The Optical File Cabinet: A Random-Access File System for Write-Once Optical Disks”, **IEEE Computer**, 21:6, czerwiec 1988.

Galindo-Legaria, C., Joshi, M. [2001] „Orthogonal Optimization of Subqueries and Aggregation”, w: *SIGMOD* [2001].

Galindo-Legaria, C., Sefani, S., Waas, F. [2004] „Query Processing for SQL Updates”, w: *SIGMOD* [2004], s. 844 – 849.

Gallaire H., Minker J., red. [1978] **Logic and Databases**, Plenum Press, 1978.

Gallaire H., Minker J., Nicolas J. [1984] „Logic and Databases: A Deductive Approach”, *ACM Computing Surveys*, 16:2, czerwiec 1984.

Gallaire H., Minker J., Nicolas J., red. [1981], **Advances in Database Theory**, tom 1, Plenum Press, 1981.

Gamal-Eldin M., Thomas G., Elmasri R. [1988] „Integrating Relational Databases with Support for Updates”, *Proceedings of the International Symposium on Databases in Parallel and Distributed Systems*, IEEE CS, grudzień 1988.

Gane C., Sarson T. [1977] **Structured Systems Analysis: Tools and Techniques**, Improved Systems Technologies, 1977.

Gangopadhyay A., Adam N. [1997]. **Database Issues in Geographic Information Systems**, Kluwer Academic Publishers, 1997.

Garcia-Molina H. [1982] „Elections in Distributed Computing Systems”, **IEEE Transactions on Computers**, 31:1, styczeń 1982.

Garcia-Molina H. [1983] „Using Semantic Knowledge for Transaction Processing in a Distributed Database”, **TODS**, 8:2, czerwiec 1983.

Garcia-Molina, H., Ullman, J., Widom, J. [2000] **Database System Implementation**, Prentice-Hall, 2000.

Garcia-Molina, H., Ullman, J., Widom, J. [2009] **Database Systems: The Complete Book**, wydanie drugie, PrenticeHall, 2009.

Gartner [2014a] **Hype Cycle for Information Infrastructure**, Mark Beyer I Roxanne Edjlali, sierpień 2014, Gartner Press, 110 s.

Gartner [2014b] „The Logical Data Warehouse Will be a Key Scenario for Using Data Federation”, Eric Thoo I Ted Friedman, Gartner, wrzesień 2012, 6 s.

Gedik, B., Liu, L. [2005] „Location Privacy in Mobile Systems: A Personalized Anonymization Model”, w: *ICDCS*, 2005, s. 620 – 629.

Gehani N., Jagadish H., Shmueli O. [1992] „Composite Event Specification in Active Databases: Model and Implementation”, w: *VLDB* [1992].

Geman, S., Geman, D. [1984] „Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images”, **IEEE Transactions on Pattern Analysis and Machine Intelligence**, tom PAMI-6, nr 6, listopad 1984, s. 721 – 741.

Georgakopoulos D., Rusinkiewicz M., Sheth A. [1991] „On Serializability of Multi-database Transactions Through Forced Local Conflicts”, w: *ICDE* [1991].

Gerritsen R. [1975] „A Preliminary System for the Design of DBTG Data Structures”, **CACM**, 18:10, październik 1975.

Ghemawat, S., Gobioff, H., Leung, S. [2003] „The Google File System”, w: *SOSP* [2003].

Ghosh S. [1984] „An Application of Statistical Databases in Manufacturing Testing”, w: *ICDE* [1984].

- Ghosh S. [1986] „Statistical Data Reduction for Manufacturing Testing”, w: ICDE [1986].
- Gibson, G. i in. [1997] „File Server Scaling with NetworkAttached Secure Disks”, Sigmetrics, 1997.
- Gifford D. [1979] „Weighted Voting for Replicated Data”, *Proceedings of the Seventh ACM Symposium on Operating Systems Principles*, 1979.
- Gladney H. [1989] „Data Replicas in Distributed Information Services”, **TODS**, 14:1, marzec 1989.
- Gogolla M., Hohenstein U. [1991] „Towards a Semantic View of an Extended Entity-Relationship Model”, **TODS**, 16:3, wrzesień 1991.
- Goldberg A., Robson D. [1983] **Smalltalk-80: The Language and Its Implementation**, Addison-Wesley, 1983.
- Goldfine A., Konig P. [1988] *A Technical Overview of the Information Resource Dictionary System (IRDS)*, wyd. 2, NBS IR 88-3700, National Bureau of Standards.
- Goodchild, M. F. [1992] „Geographical Information Science”, **International Journal of Geographical Information Systems**, 1992, s. 31 – 45.
- Goodchild, M. F. [1992a] „Geographical Data Modeling”, **Computers & Geosciences** 18:4, 1992, s. 401 – 408.
- Gordillo, S., Balaguer, F. [1998] „Refining an Objectoriented GIS Design Model: Topologies and Field Data”, *Proc. 6th ACM International Symposium on Advances in Geographic Information Systems*, 1998.
- Gotlieb L. [1975] „Computing Joins of Relations”, w: SIGMOD [1975].
- Graefe G. [1993] „Query Evaluation Techniques for Large Databases”, **ACM Computing Surveys**, 25:2, czerwiec 1993.
- Graefe G., DeWitt D. [1987] „The EXODUS Optimizer Generator”, w: SIGMOD [1987].
- Graefe, G., McKenna, W. [1993] „The Volcano Optimizer Generator”, w: ICDE [1993], s. 209 – 218.
- Graefe, G. [1995] „The Cascades Framework for Query Optimization”, *Data Engineering Bulletin*, 18:3, 1995, s. 209 – 218.
- Gravano L., Garcia-Molina H. [1997] „Merging Ranks from Heterogeneous Sources”, w: VLDB [1997].
- Gray J. [1978] „Notes on Data Base Operating Systems”, w: Bayer, Graham, Seegmuller [1978].
- Gray J. [1981] „The Transaction Concept: Virtues and Limitations”, w: VLDB [1981].
- Gray J., Reuter A. [1993] **Transaction Processing: Concepts and Techniques**, Morgan Kaufmann, 1993.
- Gray, J., Helland, P., O'Neil, P., Shasha, D. [1993] „The Dangers of Replication and a Solution”, SIGMOD [1993].
- Gray, J., Horst, B., Walker, M. [1990] „Parity Striping of Disk Arrays: Low-Cost Reliable Storage with Acceptable Throughput”, w: VLDB [1990], s. 148 – 161.
- Gray J., Lorie R., Putzulo G. [1975] „Granularity of Locks and Degrees of Consistency in a Shared Data Base”, w: Nijssen [1975].
- Gray J., McJones P., Blasgen M. [1981] „The Recovery Manager of the System R Database Manager”, **ACM Computing Surveys**, 13:2, czerwiec 1981.

- Griffiths P., Wade B. [1976] „An Authorization Mechanism for a Relational Database System”, **TODS**, 1:3, wrzesień 1976.
- Grochowski E., Hoyt, R. F. [1996] „Future Trends in Hard Disk Drives”, **IEEE Transactions on Magnetism**, 32:3, maj 1996.
- Grosky W. [1994] „Multimedia Information Systems”, w: **IEEE Multimedia**, 1:1, wiosna 1994.
- Grosky W. [1997] „Managing Multimedia Information in Database Systems”, w: **CACM**, 40:12, grudzień 1997.
- Grosky W., Jain R., Mehrotra R., red. [1997], **The Handbook of Multimedia Information Management**, Prentice-Hall PTR, 1997.
- Gruber, T. [1995] „Toward principles for the design of ontologies used for knowledge sharing”, **International Journal of Human-Computer Studies**, 43:5 – 6, listopad/grudzień 1995, s. 907 – 928.
- Gupta, R., Horowitz E. [1992] **Object Oriented Databases with Applications to Case, Networks and VLSI CAD**, Prentice-Hall, 1992.
- Güting, R. [1994] „An Introduction to Spatial Database Systems”, w: **VLDB** [1994].
- Guttman A. [1984] „R-Trees: A Dynamic Index Structure for Spatial Searching”, w: **SIGMOD** [1984].
- Gwayer M. [1996] **Oracle Designer/2000 Web Server Generator Technical Overview** (wersja 1.3.2), raport techniczny, Oracle Corporation, wrzesień 1996.
- Gyssens, M., Paredaens, J., Van Gucht, D. [1990] „A graph-oriented object model for database end-user interfaces”, w: **SIGMOD** [1990].
- Haas P., Swami A. [1995] „Sampling-Based Selectivity Estimation for Joins Using Augmented Frequent Value Statistics”, w: **ICDE** [1995].
- Haas P., Naughton J., Seshadri S., Stokes L. [1995] Sampling-Based Estimation of the Number of Distinct Values of an Attribute”, w: **VLDB** [1995].
- Hachem N., Berra, P. [1992] „New Order Preserving Access Methods for Very Large Files Derived from Linear Hashing”, **TKDE**, 4:1, luty 1992.
- Hadoop [2014] serwis wiki poświęcony Hadoopowi: <http://hadoop.apache.org/>.
- Hadzilacos V. [1983] „An Operational Model for Database System Reliability”, in *Proceedings of SIGACT-SIGMOD Conference*, marzec 1983.
- Hadzilacos V. [1988] „A Theory of Reliability in Database Systems”, 1986.
- Haerder T., Reuter A. [1983] „Principles of Transaction Oriented Database Recovery — A Taxonomy”, **ACM Computing Surveys**, 15:4, wrzesień 1983, s. 287 – 318.
- Haerder T., Rothermel K. [1987] „Concepts for Transaction Recovery in Nested Transactions”, w: **SIGMOD** [1987].
- Hakonarson, H., Gulcher, J., Stefansson, K. [2003] „deCODE genetics, Inc.”, **Pharmacogenomics Journal**, 2003, s. 209 – 215.
- Halfond, W., Orso. A. [2005] „AMNESIA: Analysis and Monitoring for Neutralizing SQL-Injection Attacks”, w: *Proc. IEEE and ACM Int. Conf. on Automated Software Engineering (ASE 2005)*, listopad 2005, s. 174 – 183.
- Halfond, W., Viegas, J., Orso, A. [2006] „A Classification of SQL Injection Attacks and Countermeasures”, w: *Proc. Int. Symposium on Secure Software Engineering*, marzec 2006.

- Hall P. [1976] „Optimization of a Single Relational Expression in a Relational Data Base System”, **IBM Journal of Research and Development**, 20:3, maj 1976.
- Hamilton G., Catteli R., Fisher M. [1997] **JDBC Database Access with Java-A Tutorial and Annotated Reference**, Addison-Wesley, 1997.
- Hammer M., McLeod D. [1975] „Semantic Integrity in a Relational Data Base System”, w: *VLDB* [1975].
- Hammer M., McLeod D. [1981] „Database Description with SDM: A Semantic Data Model”, **TODS**, 6:3, wrzesień 1981.
- Hammer M., Sarin, S. [1978] „Efficient Monitoring of Database Assertions”, w: *SIGMOD* [1978].
- Han J., Kamber M., Pei, J. [2005] **Data Mining: Concepts and Techniques**, wydanie drugie, Morgan Kaufmann, San Francisco, 2005.
- Han, Y., Jiang, C. Luo, X. [2004] „A Study of Concurrency Control in Web-Based Distributed Real-Time Database System Using Extended Time Petri Nets”, *Proc. Int. Symposium on Parallel Architectures, Algorithms, and Networks*, 2004, s. 67 – 72.
- Han J., Pei J., Yin Y. [2000] „Mining Frequent Patterns without Candidate Generation”, w: *SIGMOD*, 2000.
- Hanson E. [1992] „Rule Condition Testing and Action Execution in Ariel”, w: *SIGMOD* [1992].
- Hardgrave W. [1980] „Ambiguity in Processing Boolean Queries on TDMS Tree Structures: A Study of Four Different Philosophies”, **TSE**, 6:4, lipiec 1980.
- Hardgrave W. [1984] „BOLT: A Retrieval Language for Tree-Structured Database Systems”, w: Tou [1984].
- Harel, D. [1987] „Statecharts: A Visual Formulation for Complex Systems”, w: **Science of Computer Programming**, 8:3, czerwiec 1987, s. 231 – 274.
- Harman, D. [1992] „Evaluation Issues in Information Retrieval”, **Information Processing and Management**, 28:4, s. 439 – 440.
- Harrington J. [1987] **Relational Database Management for Microcomputer: Design and Implementation**, Holt, Rinehart, and Winston, 1987.
- Harris L. [1978] „The ROBOT System: Natural Language Processing Applied to Data Base Query”, *Proceedings of the ACM National Conference*, grudzień 1978.
- Harth, A., Hose, K., Schenkel, R. [2014] **Linked Data Management**, Chapman and Hall, CRC Press, 2014, 576 s.
- Haskin R., Lorie R. [1982] „On Extending the Functions of a Relational Database System”, w: *SIGMOD* [1982].
- Hasse C., Weikum G. [1991] „A Performance Evaluation of Multi-Level Transaction Management”, w: *VLDB* [1991].
- Hayes-Roth F., Waterman D., Lenat D., red. [1983] **Building Expert Systems**, Addison-Wesley, 1983.
- Hayne S., Ram S. [1990] „Multi-User View Integration System: An Expert System for View Integration”, w: *ICDE* [1990].
- Hecht. R., Jablonski, S. [2011] „NoSQL Evaluation, A Use Case Oriented Survey”, w: *Int. Conf. on Cloud and Service Computing*, IEEE, 2011, s. 336 – 341.
- Heiler S., Zdonick, S. [1990] „Object Views: Extending the Vision”, w: *ICDE* [1990].

- Heiler S., Hardhvalal S., Zdonick S., Blaustein B., Rosenthal A. [1992] „A Flexible Framework for Transaction Management in Engineering Environment”, w: Elmagarmid [1992].
- Helal A., Hu T., Elmasri R., Mukherjee S. [1993] „Adaptive Transaction Scheduling”, *CIKM*, listopad 1993.
- Held G., Stonebraker, M. [1978] „B-Trees Reexamined”, *CACM*, 21:2, luty 1978.
- Henriksen, C., Lauzon, J. P., Morehouse, S. [1994] „Open Geodata Access Through Standards”, *Standard View Archive*, 1994, 2:3, s. 169 – 174.
- Henschen, L., Naqvi S. [1984] „On Compiling Queries in Recursive First-Order Databases”, *JACM*, 31:1, styczeń 1984.
- Hernandez H., Chan E. [1991] „Constraint-Time-Maintainable BCNF Database Schemes”, *TODS*, 16:4, grudzień 1991.
- Herot C. [1980] „Spatial Management of Data”, *TODS*, 5:4, grudzień 1980.
- Hevner A., Yao S. [1979] „Query Processing in Distributed Database Systems”, *TSE*, 5:3, maj 1979.
- Hinneburg, A., Gabriel, H.-H. [2007] „DENCLUE 2.0: Fast Clustering Based on Kernel Density Estimation”, w: *Proc. IDA'2007: Advances in Intelligent Data Analysis VII, 7th International Symposium on Intelligent Data Analysis*, Ljubljana, Słowenia, wrzesień 2007, **LNCS 4723**, Springer, 2007.
- Hoffer J. [1982] „An Empirical Investigation with Individual Differences in Database Models”, *Proceedings of the Third International Information Systems Conference*, grudzień 1982.
- Hoffer, J., Prescott, M., Topi, H. [2009] **Modern Database Management**, wydanie dziewiąte, Prentice-Hall, 2009.
- Holland J. [1975] **Adaptation in Natural and Artificial Systems**, University of Michigan Press, 1975.
- Holsapple C., Whinston A., red. [1987] **Decisions Support Systems Theory and Application**, Springer-Verlag, 1987.
- Holt, R. C. [1972] „Some Deadlock Properties of Computer Systems”, *ACM Computing Surveys*, 4:3, s. 179 – 196.
- Holtzman J. M., Goodman D. J., red. [1993] **Wireless Communications: Future Directions**, Kluwer, 1993.
- Horowitz, B. [1992] „A Run-Time Execution Model for Referential Integrity Maintenance”, w: *ICDE* [1992], s. 548 – 556.
- Hortonworks, Inc. [2014a] „Benchmarking Apache Hive 13 for Enterprise Hadoop”, Carter Shanklin, blog Hortonworks (<http://hortonworks.com/blog/benchmarking-apache-hive-13-enterprise-hadoop/>), czerwiec 2014.
- Hortonworks, Inc. [2014b] „Best Practices — Selecting Apache Hadoop Hardware”.
- Howson, C., P. Urbach, P. [1993] **Scientific Reasoning: The Bayesian Approach**, Open Court Publishing, grudzień 1993.
- Hsiao D., Kamel M. [1989] „Heterogeneous Databases: Proliferation, Issues, and Solutions”, *TKDE*, 1:1, marzec 1989.
- Hsu A., Imielinsky T. [1985] „Integrity Checking for Multiple Updates”, w: *SIGMOD* [1985].
- Hsu, M., Zhang, B. [1992] „Performance Evaluation of Cautious Waiting”, *TODS*, 17:3, s. 477 – 512.



Hull R., King R. [1987] „Semantic Database Modeling: Survey, Applications, and Research Issues”, **ACM Computing Surveys**, 19:3, wrzesień 1987.

Huxhold, W. [1991] **An Introduction to Urban Geographic Information Systems**, Oxford University Press, 1991.

IBM [1978] **QBE Terminal Users Guide**, Form Number SH20-2078-0.

IBM [1992] **Systems Application Architecture Common Programming Interface Database Level 2 Reference**, Document Number SC26-4798-01.

ICDE [1984] *Proceedings of the IEEE CS International Conference on Data Engineering*, Shuey R. red., Los Angeles, CA, kwiecień 1984.

ICDE [1986] *Proceedings of the IEEE CS International Conference on Data Engineering*, Wiederhold G., red., Los Angeles, luty 1986.

ICDE [1987] *Proceedings of the IEEE CS International Conference on Data Engineering*, Wah B., red., Los Angeles, luty 1987.

ICDE [1988] *Proceedings of the IEEE CS International Conference on Data Engineering*, Carlis, J. red., Los Angeles, luty 1988.

ICDE [1989] *Proceedings of the IEEE CS International Conference on Data Engineering*, Shuey, R., red., Los Angeles, luty 1989.

ICDE [1990] *Proceedings of the IEEE CS International Conference on Data Engineering*, Liu, M., red., Los Angeles, luty 1990.

ICDE [1991] *Proceedings of the IEEE CS International Conference on Data Engineering*, Cerccone, N., and Tsuchiya, M., red., Kobe, Japonia, kwiecień 1991.

ICDE [1992] *Proceedings of the IEEE CS International Conference on Data Engineering*, Golshani F., red., Phoenix, AZ, luty 1992.

ICDE [1993] *Proceedings of the IEEE CS International Conference on Data Engineering*, Elmagarmid A., Neuhold E., red., Wiedeń, Austria, kwiecień 1993.

ICDE [1994] *Proceedings of the IEEE CS International Conference on Data Engineering*.

ICDE [1995] *Proceedings of the IEEE CS International Conference on Data Engineering*, Yu P. S., Chen A. L. A. red., Taipei, Tajwan, 1995.

ICDE [1996] *Proceedings of the IEEE CS International Conference on Data Engineering*, Su, S. Y. W., red., New Orleans, 1996.

ICDE [1997] *Proceedings of the IEEE CS International Conference on Data Engineering*, Gray, A., Larson, P. A., red., Birmingham, Anglia, 1997.

ICDE [1998] *Proceedings of the IEEE CS International Conference on Data Engineering*, Orlando, FL, 1998.

ICDE [1999] *Proceedings of the IEEE CS International Conference on Data Engineering*, Sydney, Australia, 1999.

ICDE [2000] *Proc. IEEE CS International Conference on Data Engineering*, San Diego, CA, luty – marzec 2000.

ICDE [2001] *Proc. IEEE CS International Conference on Data Engineering*, Heidelberg, Niemcy, kwiecień 2001.

- ICDE [2002] *Proc. IEEE CS International Conference on Data Engineering*, San Jose, CA, luty – marzec 2002.
- ICDE [2003] *Proc. IEEE CS International Conference on Data Engineering*, Dayal, U., Ramamritham, K., Vijayaraman, T. M., red., Bangalore, Indie, marzec 2003.
- ICDE [2004] *Proc. IEEE CS International Conference on Data Engineering*, Boston, MA, marzec – kwiecień 2004.
- ICDE [2005] *Proc. IEEE CS International Conference on Data Engineering*, Tokio, Japonia, kwiecień 2005.
- ICDE [2006] *Proc. IEEE CS International Conference on Data Engineering*, Liu, L., Reuter, A., Whang, K.-Y., Zhang, J., red., Atlanta, GA, kwiecień 2006.
- ICDE [2007] *Proc. IEEE CS International Conference on Data Engineering*, Sztambuł, Turcja, kwiecień 2007.
- ICDE [2008] *Proc. IEEE CS International Conference on Data Engineering*, Cancun, Meksyk, kwiecień 2008.
- ICDE [2009] *Proc. IEEE CS International Conference on Data Engineering*, Szanghaj, Chiny, marzec – kwiecień 2009.
- ICDE [2010] *Proc. IEEE CS International Conference on Data Engineering*, Long Beach, CA, marzec 2010.
- ICDE [2011] *Proc. IEEE CS International Conference on Data Engineering*, Hannover, Niemcy, kwiecień 2011.
- ICDE [2012] *Proc. IEEE CS International Conference on Data Engineering*, Kementsietsidis, A., Antonio Vaz Salles, M., red., Waszyngton, D.C., kwiecień 2012.
- ICDE [2013] *Proc. IEEE CS International Conference on Data Engineering*, Jensen, C., Jermaine, C., Zhou, Xiaofang, red., Brisbane, Australia, kwiecień 2013.
- ICDE [2014] *Proc. IEEE CS International Conference on Data Engineering*, Cruz, Isabel F. i in., red., Chicago, marzec – kwiecień 2014.
- ICDE [2015] *Proc. IEEE CS International Conference on Data Engineering*, Seul, Korea, kwiecień 2015, w przygotowaniu.
- IGES [1983] *International Graphics Exchange Specification Version 2*, National Bureau of Standards, U.S. Department of Commerce, styczeń 1983.
- Imielinski T, Badrinath. B. [1994] „Mobile Wireless Computing: Challenges in Data Management”, **CACM**, 37:10, październik 1994.
- Imielinski T., Lipski W. [1981] „On Representing Incomplete Information in a Relational Database”, w: *VLDB* [1981].
- Indulska, M., Orłowska, M. E. [2002] „On Aggregation Issues in Spatial Data Management”, (ACM International Conference Proceeding Series) *Proc. Thirteenth Australasian Conference on Database Technologies*, Melbourne, 2002, s. 75 – 84.
- Informix [1998] „Web Integration Option for Informix Dynamic Server”, artykuł dostępny w witrynie <http://www.informix.com>.
- Inmon, W. H. [1992] **Building the Data Warehouse**, Wiley 1992.
- Inmon, W., Strauss, D., Neushloss, G. [2008] **DW 2.0: The Architecture for the Next Generation of Data Warehousing**, Morgan Kaufmann, 2008.

- Integrity [2004] „An Introduction to SQL Injection Attacks for Oracle Developers”, Integrity, kwiecień 2004, tekst dostępny pod adresem: <http://www.net-security.org/dl/articles/IntegrityIntrotoSQLInjectionAttacks.pdf>.
- Internet Engineering Task Force (IETF) [1999] „An Architecture Framework for High Speed Mobile Ad Hoc Network”, w: *Proc. 45th IETF Meeting*, Oslo, Norwegia, lipiec 1999, tekst dostępny na stronie: <http://www.ietf.org/proceedings/99jul/>.
- Ioannidis Y., Kang Y. [1990] „Randomized Algorithms for Optimizing Large Join Queries”, w: *SIGMOD* [1990].
- Ioannidis Y., Kang Y. [1991] „Left-Deep vs Bushy Trees: An Analysis of Strategy Spaces and Its Implications for Query Optimization”, w: *SIGMOD* [1991].
- Ioannidis Y., Wong E. [1988] „Transforming Non-Linear Recursion to Linear Recursion”, w: *EDS* [1988].
- Iossophidis J. [1979] „A Translator to Convert the DDL of ERM to the DDL of System 2000”, w: *ER Conference* [1979].
- Irani K., Purkayastha S., Teorey T. [1979] „A Designer for DBMS-Processable Logical Database Structures”, w: *VLDB* [1979].
- Iyer i in. [2004] „A Framework for Efficient Storage Security in RDBMSs”, w: *EDBT*, 2004, s. 147 – 164.
- Jacobson, I., Booch, G., Rumbaugh, J. [1999] **The Unified Software Development Process**, AddisonWesley, 1999.
- Jacobson I., Christerson M., Jonsson P., Overgaard G. [1992] **Object Oriented Software Engineering: A Use Case Driven Approach**, Addison-Wesley, 1992.
- Jagadish H. [1989] „Incorporating Hierarchy in a Relational Model of Data”, w: *SIGMOD* [1989].
- Jagadish H. [1997] „Content-Based Indexing and Retrieval”, w: Grosky i in. [1997].
- Jajodia, S., Ammann, P., McCollum, C. D. [1999] „Surviving Information Warfare Attacks”, **IEEE Computer**, 32:4, kwiecień 1999, s. 57 – 63.
- Jajodia S., Kogan B. [1990] „Integrating an Object-Oriented Data Model with Multilevel Security”, *Proc. IEEE Symposium on Security and Privacy*, maj 1990, s. 76 – 85.
- Jajodia S., Mutchler D. [1990] „Dynamic Voting Algorithms for Maintaining the Consistency of a Replicated Database”, **TODS**, 15:2, czerwiec, 1990.
- Jajodia S., Sandhu R. [1991] „Toward a Multilevel Secure Relational Data Model”, w: *SIGMOD* [1991].
- Jajodia S., Ng P., Springsteel F. [1983] „The Problem of Equivalence for Entity-Relationship Diagrams”, **TSE**, 9:5, wrzesień, 1983.
- Jardine D., red. [1977] **The ANSI/SPARC DBMS Model**, North-Holland, 1977.
- Jarke M., Koch J. [1984] „Query Optimization in Database Systems”, **ACM Computing Surveys**, 16:2, czerwiec, 1984.
- Jensen C. i in. [1994] „A Glossary of Temporal Database Concepts”, **ACM SIGMOD Record**, 23:1, marzec 1994.
- Jensen C., Snodgrass R. [1992] „Temporal Specialization”, w: *ICDE* [1992].
- Jensen, C. i in. [2001] „Location-based Services: A Database Perspective”, *Proc. ScanGIS Conference*, 2001, s. 59 – 68.

- Jhingran, A., and Khedkar, P. [1992] „Analysis of Recovery in a Database System Using a Write-ahead Log Protocol”, w: *SIGMOD* [1992].
- Jing, J., Helal, A., Elmagarmid, A. [1999] „Clientserver Computing in Mobile Environments”, **ACM Computing Surveys**, 31:2, czerwiec 1999.
- Johnson T., Shasha D. [1993] „The Performance of Current B-Tree Algorithms”, **TODS**, 18:1, marzec 1993.
- Jorwekar, S. i in. [2007] „Automating the Detection of Snapshot Isolation Anomalies”, w: *VLDB* [2007], s. 1263 – 1274.
- Joshi J. B. D., Aref W. G., Ghafoor A., Spafford E. H. [2001] „Security Models for Web-Based Applications”, **CACM**, luty 2001, s. 38 – 44.
- Jukic, N., Vrbsky, S., Nestorov, S. [2013] **Database Systems: Introduction to Databases and Data Warehouses**, Prentice Hall, 2013, 408 s.
- Jung, I.Y., Yeom, H.Y. [2008] „An efficient and transparent transaction management based on the dataworkflow of HVEM DataGrid”, *Proc. Challenges of Large Applications in Distributed Environments*, 2008, s. 35 – 44.
- Kaefer W., Schoening H. [1992] „Realizing a Temporal Complex-Object Data Model”, w: *SIGMOD* [1992].
- Kamel I., Faloutsos C. [1993] „On Packing R-trees”, *CIKM*, listopad 1993.
- Kamel N., King R. [1985] „A Model of Data Distribution Based on Texture Analysis”, w: *SIGMOD* [1985].
- Kappel G., Schrefl M. [1991] „Object/Behavior Diagrams”, w: *ICDE* [1991].
- Karlapalem K., Navathe S. B., Ammar M. [1996] „Optimal Redesign Policies to Support Dynamic Processing of Applications on a Distributed Relational Database System”, **Information Systems**, 21:4, 1996, s. 353 – 367.
- Karolchik, D. i in. [2003] „The UCSC Genome Browser Database”, w: **Nucleic Acids Research**, 31:1, styczeń 2003.
- Katz R. [1985] **Information Management for Engineering Design: Surveys in Computer Science**, Springer-Verlag, 1985.
- Katz R., Wong E. [1982] „Decompiling CODASYL DML into Relational Queries”, **TODS**, 7:1, marzec 1982.
- Kavis, M. [2014] **Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS)**, Wiley, 224 s.
- KDD [1996] *Proceedings of the Second International Conference on Knowledge Discovery in Databases and Data Mining*, Portland, Oregon, sierpień 1996.
- Ke, Y., Sukthankar, R. [2004] „PCA-SIFT: A More Distinctive Representation for Local Image Descriptors”, w: *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2004.
- Kedem Z., Silberschatz A. [1980] „Non-Two Phase Locking Protocols with Shared and Exclusive Locks”, w: *VLDB* [1980].
- Keller A. [1982] „Updates to Relational Database Through Views Involving Joins”, w: Scheuermann [1982].
- Kemp K. [1993]. „Spatial Databases: Sources and Issues”, w: **Environmental Modeling with GIS**, Oxford University Press, Nowy Jork, 1993.

- Kemper A., Wallrath M. [1987] „An Analysis of Geometric Modeling in Database Systems”, **ACM Computing Surveys**, 19:1, marzec 1987.
- Kemper A., Lockemann P., Wallrath M. [1987] „An Object-Oriented Database System for Engineering Applications”, w: *SIGMOD* [1987].
- Kemper A., Moerkotte G., Steinbrunn M. [1992] „Optimizing Boolean Expressions in Object Bases”, w: *VLDB* [1992].
- Kent W. [1978] **Data and Reality**, North-Holland, 1978.
- Kent W. [1979] „Limitations of Record-Based Information Models”, **TODS**, 4:1, marzec 1979.
- Kent W. [1991] „Object-Oriented Database Programming Languages”, w: *VLDB* [1991].
- Kerschberg L., Ting P., Yao S. [1982] „Query Optimization in Star Computer Networks”, **TODS**, 7:4, grudzień 1982.
- Ketabchi M. A., Mathur S., Risch T., Chen J. [1990] „Comparative Analysis of RDBMS and OODBMS: A Case Study”, *IEEE International Conference on Manufacturing*, 1990.
- Khan, L. [2000] „Ontology-based Information Selection”, praca doktorska, University of Southern California, sierpień 2000.
- Khoshafian S., Baker A., [1996] **Multimedia and Imaging Databases**, Morgan Kaufmann, 1996.
- Khoshafian S., Chan A., Wong A., Wong H. K. T. [1992] **Developing Client Server Applications**, Morgan Kaufmann, 1992.
- Khoury, M. [2002] „Epidemiology and the Continuum from Genetic Research to Genetic Testing”, w: **American Journal of Epidemiology**, 2002, s. 297 – 299.
- Kifer M., Lozinskii, E. [1986] „A Framework for an Efficient Implementation of Deductive Databases”, *Proceedings of the Sixth Advanced Database Symposium*, Tokio, sierpień 1986.
- Kim W. [1995] **Modern Database Systems: The Object Model, Interoperability, and Beyond**, ACM Press, Addison-Wesley, 1995.
- Kim P. [1996] „A Taxonomy on the Architecture of Database Gateways for the Web”, Working Paper TR-96-U-10, Chungnam National University, Taejon, Korea (dokument dostępny pod adresem <http://grigg.chungnam.ac.kr/projects/UniWeb>).
- Kim, S.-H., Yoon, K.-J., Kweon, I.-S. [2006] „Object Recognition Using a Generalized Robust Invariant Feature and Gestalt's Law of Proximity and Similarity”, w: *Proc. Conf. on Computer Vision and Pattern Recognition Workshop (CVPRW '06)*, 2006.
- Kim W. [1982] „On Optimizing an SQL-like Nested Query”, **TODS**, 3:3, wrzesień 1982.
- Kim W. [1989] „A Model of Queries for Object-Oriented Databases”, w: *VLDB* [1989].
- Kim W. [1990] „Object-Oriented Databases: Definition and Research Directions”, **TKDE**, 2:3, wrzesień 1990.
- Kim W. i in. [1987] „Features of the ORION Object-Oriented Database System”, Microelectronics and Computer Technology Corporation, raport techniczny ACA-ST-308-87, wrzesień 1987.
- Kim, W., Lochovsky, F., red. [1989] **Object-oriented Concepts, Databases, and Applications**, ACM Press, Frontier Series, 1989.
- Kim, W., Garza, J., Ballou, N., Woelk, D. [1990] „Architecture of the ORION Next-Generation Database System”, **TKDE**, 2:1, 1990, s. 109 – 124.

- Kim W., Reiner D., Batory D., red. [1985] **Query Processing in Database Systems**, Springer-Verlag, 1985.
- Kimball R. [1996] **The Data Warehouse Toolkit**, Wiley, Inc. 1996.
- King J. [1981] „QUIST: A System for Semantic Query Optimization in Relational Databases”, w: VLDB [1981].
- Kitsuregawa M., Nakayama M., Takagi M. [1989] „The Effect of Bucket Size Tuning in the Dynamic Hybrid GRACE Hash Join Method”, w: VLDB [1989].
- Kleinberg, J. M. [1999] „Authoritative sources in a hyperlinked environment”, **JACM** 46:5, wrzesień 1999, s. 604 – 632.
- Klimbie J., Koffeman K., red. [1974] **Data Base Management**, North-Holland, 1974.
- Klug A. [1982] „Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions”, **JACM**, 29:3, lipiec 1982.
- Knuth D. [2002] **Sztuka programowania**, tom 3: **Sortowanie i wyszukiwanie**, WNT, 2001 (Knuth D. [1998] **The Art of Computer Programming**, tom 3: **Sorting and Searching**, wydanie drugie, Addison-Wesley, 1998).
- Kogelnik A. [1998] „Biological Information Management with Application to Human Genome Data”, praca doktorska, Georgia Institute of Technology and Emory University, 1998.
- Kogelnik A. i in. [1998] „MITOMAP: A human mitochondrial genome database — 1998 update”, **Nucleic Acids Research**, 26:1, styczeń 1998.
- Kogelnik A., Navathe S., Wallace D. [1997] „GENOME: A system for managing Human Genome Project Data”. *Proceedings of Genome Informatics '97, Eighth Workshop on Genome Informatics*, Tokio, Japonia, Sponsor: Human Genome Center, University of Tokyo, grudzień 1997.
- Kohler W. [1981] „A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems”, **ACM Computing Surveys**, 13:2, czerwiec 1981.
- Konsynski B., Bracker L., Bracker W. [1982] „A Model for Specification of Office Communications”, **IEEE Transactions on Communications**, 30:1, styczeń 1982.
- Kooi, R. P., [1980] **The Optimization of Queries in Relational Databases**, praca doktorska, Case Western Reserve University, 1980: s. 1 – 159.
- Koperski, K., Han, J. [1995] „Discovery of Spatial Association Rules in Geographic Information Databases”, w: *Proc. SSD'1995, 4th Int. Symposium on Advances in Spatial Databases*, Portland, Maine, **LNCS** 951, Springer, 1995.
- Korfhage R. [1991] „To See, or Not to See: Is that the Query” w: *Proceedings of the ACM SIG1R International Conference*, czerwiec 1991.
- Korth H. [1983] „Locking Primitives in a Database System”, **JACM**, 30:1, styczeń 1983.
- Korth H., Levy E., Silberschatz A. [1990] „A Formal Approach to Recovery by Compensating Transactions”, w: VLDB [1990].
- Kosala, R., Blockeel, H. [2000] „Web Mining Research: a Survey”, **SIGKDD Explorations**, 2:1, czerwiec 2000, s. 1 – 15.
- Kotz A., Dittrich K., Mülle J. [1988] „Supporting Semantic Rules by a Generalized Event/Trigger Mechanism”, w: VLDB [1988].

- Kotz, S., Balakrishnan, N., Johnson, N. L. [2000] „Dirichlet and Inverted Dirichlet Distributions”, w: **Continuous Multivariate Distributions: Models and Applications, Vol. 1**, wydanie drugie, John Wiley, 2000.
- Krishnamurthy R., Naqvi S. [1989] „Non-Deterministic Choice in Datalog”, *Proceedings of the 3rd International Conference on Data and Knowledge Bases*, Jerozolima, czerwiec 1989.
- Krishnamurthy R., Litwin W., Kent W. [1991] „Language Features for Interoperability of Databases with Semantic Discrepancies”, w: *SIGMOD* [1991].
- Krovetz R., Croft B. [1992] „Lexical Ambiguity and Information Retrieval”, w: **TOIS**, 10, kwiecień 1992.
- Kubiatowicz, J. i in. [2000] „OceanStore: An Architecture for Global-Scale Persistent Storage”, *ASPLOS* 2000.
- Kuhn, R. M., Karolchik, D., Zweig i in. [2009] „The UCSC Genome Browser Database: update 2009”, **Nucleic Acids Research**, 37:1, styczeń 2009.
- Kulkarni K. i in. [1995] „Introducing Reference Types and Cleaning Up SQL3’s Object Model”, *ISO WG3 Report X3H2-95-456*, listopad 1995.
- Kumar A. [1991] „Performance Measurement of Some Main Memory Recovery Algorithms”, w: *ICDE* [1991].
- Kumar A., Segev A. [1993] „Cost and Availability Tradeoffs in Replicated Concurrency Control”, **TODS**, 18:1, marzec 1993.
- Kumar A., Stonebraker M. [1987] „Semantics Based Transaction Management Techniques for Replicated Data”, w: *SIGMOD* [1987].
- Kumar, D. [2007a] „Genomic medicine: a new frontier of medicine in the twenty first century”, **Genomic Medicine**, 2007, s. 3 – 7.
- Kumar, D. [2007b] „Genome mirror—2006”, **Genomic Medicine**, 2007, s. 87 – 90.
- Kumar V., Han M., red. [1992] **Recovery Mechanisms in Database Systems**, Prentice-Hall, 1992.
- Kumar V., Hsu M. [1998] **Recovery Mechanisms in Database Systems**, Prentice-Hall (PTR), 1998.
- Kumar V., Song H. S. [1998] **Database Recovery**, Kluwer Academic, 1998.
- Kung H., Robinson J. [1981] „Optimistic Concurrency Control”, **TODS**, 6:2, czerwiec 1981.
- Lacroix M., Pirotte A. [1977] „Domain-Oriented Relational Languages”, w: *VLDB* [1977].
- Lacroix M., Pirotte A. [1977a] „ILL: An English Structured Query Language for Relational Data Bases”, w: Nijssen [1977].
- Lai, M.-Y., Wilkinson, W. K. [1984] „Distributed Transaction Management in Jasmin”, w: *VLDB* [1984].
- Lamb, C. i in. [1991] „The ObjectStore Database System”, w: **CACM**, 34:10, październik 1991, s. 50 – 63.
- Lamport L. [1978] „Time, Clocks, and the Ordering of Events in a Distributed System”, **CACM**, 21:7, lipiec 1978.
- Lander, E. [2001] „Initial Sequencing and Analysis of the Genome”, **Nature**, 409:6822, 2001.
- Langerak R. [1990] „View Updates in Relational Databases with an Independent Scheme”, **TODS**, 15:1, marzec 1990.
- Lanka S., Mays E. [1991] „Fully Persistent B1-Trees”, w: *SIGMOD* [1991].



- Larson J. [1983] „Bridging the Gap Between Network and Relational Database Management Systems”, **IEEE Computer**, 16:9, wrzesień 1983.
- Larson J., Navathe S., Elmasri R. [1989] „Attribute Equivalence and its Use in Schema Integration”, **TSE**, 15:2, kwiecień 1989.
- Larson P. [1978] „Dynamic Hashing”, **BIT**, 18, 1978.
- Larson P. [1981] „Analysis of Index-Sequential Files with Overflow Chaining”, **TODS**, 6:4, grudzień 1981.
- Lassila, O. [1998] „Web Metadata: A Matter of Semantics”, **IEEE Internet Computing**, 2:4, lipiec/sierpień 1998, s. 30 – 37.
- Laurini R., Thompson D. [1992] **Fundamentals of Spatial Information Systems**, Academic Press, 1992.
- Lausen G., Vossen, G. [1997] **Models and Languages of Object Oriented Databases**, Addison-Wesley, 1997.
- Lazebnik, S., Schmid, C., Ponce, J. [2004] „Semi-Local Affine Parts for Object Recognition”, w: *Proc. British Machine Vision Conference*, Kingston University, The Institution of Engineering and Technology, Wielka Brytania, 2004.
- Lee, J., Elmasri, R., Won, J. [1998] „An Integrated Temporal Data Model Incorporating Time Series Concepts”, **DKE**, 24, 1998, s. 257 – 276.
- Lehman P., Yao S. [1981] „Efficient Locking for Concurrent Operations on B-Trees”, **TODS**, 6:4, grudzień 1981.
- Lehman T., Lindsay B. [1989] „The Starburst Long Field Manager”, w: *VLDB* [1989],
- Leiss E. [1982] „Randomizing: A Practical Method for Protecting Statistical Databases Against Compromise”, w: *VLDB* [1982].
- Leiss E. [1982a] **Principles of Data Security**, Plenum Press, 1982.
- Lenat, D. [1995] „CYC: A Large-Scale Investment in Knowledge Infrastructure”, **CACM** 38:11, listopad 1995, s. 32 – 38.
- Lenzerini M., Santucci C. [1983] „Cardinality Constraints in the Entity Relationship Model”, w: *ER Conference* [1983].
- Leung C., Hibler B., Mwara N. [1992] „Picture Retrieval by Content Description”, w: **Journal of Information Science**, 1992, s. 111 – 119.
- Levesque H. [1984] „The Logic of Incomplete Knowledge Bases”, w: Brodie i in., rozdz. 7 [1984].
- Li W.-S., Seluk Candan K., Hirata K., Hara Y. [1998] Hierarchical Image Modeling for Object-Based Media Retrieval, w: **DKE**, 27:2, wrzesień 1998, s. 139 – 176.
- Lien E., Weinberger P. [1978] „Consistency, Concurrency, and Crash Recovery”, w: *SIGMOD* [1978].
- Lieuwen L., DeWitt D. [1992] „A Transformation-Based Approach to Optimizing Loops in Database Programming Languages”, w: *SIGMOD* [1992].
- Lilien L., Bhargava B. [1985] „Database Integrity Block Construct: Concepts and Design Issues”, **TSE**, 11:9, wrzesień 1985.
- Lin J., Dunham M. H. [1998] „Mining Association Rules”, in *ICDE* [1998].

- Lindsay B. i in. [1984] „Computation and Communication in R\*: A Distributed Database Manager”, **TOCS**, 2:1, styczeń 1984.
- Lippman R. [1987] „An Introduction to Computing with Neural Nets”, **IEEE ASSP Magazine**, kwiecień 1987.
- Lipski W [1979] „On Semantic Issues Connected with Incomplete Information”, **TODS**, 4:3, wrzesień 1979.
- Lipton R., Naughton J., Schneider D. [1990] „Practical Selectivity Estimation through Adaptive Sampling”, w: *SIGMOD* [1990].
- Liskov B., Zilles S. [1975] „Specification Techniques for Data Abstractions”, **TSE**, 1:1, marzec 1975.
- Litwin W. [1980] „Linear Hashing: A New Tool for File and Table Addressing”, w: *VLDB* [1980].
- Liu, B. [2006] **Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data (Data-Centric Systems and Applications)**, Springer, 2006.
- Liu, B., Chen-Chuan-Chang, K. [2004] „Editorial: Special Issue on Web Content Mining”, **SIGKDD Explorations Newsletter** 6:2, grudzień 2004, s. 1 – 4.
- Liu K., Sunderraman R. [1988] „On Representing Indefinite and Maybe Information in Relational Databases”, w: *ICDE* [1988].
- Liu L., Meersman R. [1992] „Activity Model: A Declarative Approach for Capturing Communication Behavior in Object-Oriented Databases”, w: *VLDB* [1992].
- Lockemann P., Knutsen W. [1968] „Recovery of Disk Contents After System Failure”, **CACM**, 11:8, sierpień 1968.
- Longley, P. i in. [2001] **Geographic Information Systems and Science**, John Wiley, 2001.
- Lorie R. [1977] „Physical Integrity in a Large Segmented Database”, **TODS**, 2:1, marzec 1977.
- Lorie R., Plouffe W. [1983] „Complex Objects and Their Use in Design Transactions”, w: *SIGMOD* [1983].
- Lowe, D. [2004] „Distinctive Image Features from ScaleInvariant Keypoints”, **Int. Journal of Computer Vision**, tom 60, 2004, s. 91 – 110.
- Lozinskii E. [1986] „A Problem-Oriented Inferential Database System”, **TODS**, 11:3, wrzesień 1986.
- Lu H., Mikkilineni K., Richardson J. [1987] „Design and Evaluation of Algorithms to Compute the Transitive Closure of a Database Relation”, w: *ICDE* [1987].
- Lubars M., Potts C., Richter C. [1993] „A Review of the State of Practice in Requirements Modeling”, *IEEE International Symposium on Requirements Engineering*, San Diego, CA, 1993.
- Lucyk B. [1993] **Advanced Topics in DB2**, Addison-Wesley, 1993.
- Luhn, H. P. [1957] „A Statistical Approach to Mechanized Encoding and Searching of Literary Information”, **IBM Journal of Research and Development**, 1:4, październik 1957, s. 309 – 317.
- Lunt, T., Fernandez, E. [1990] „Database Security”, w: *SIGMOD Record*, 19:4, s. 90 – 97.
- Lunt, T. i in. [1990] „The Seaview Security Model”, **IEEE TSE**, 16:6, s. 593 – 607.
- Luo, J., Nascimento, M. [2003] „Content-based Subimage Retrieval via Hierarchical Tree Matching”, w: *Proc. ACM Int Workshop on Multimedia Databases*, Nowy Orlean, s. 63 – 69.
- Madria, S. i in. [1999] „Research Issues in Web Data Mining”, w: *Proc. First Int. Conf. on Data Warehousing and Knowledge Discovery* (Mohania, M., Tjoa, A., red.) **LNCS** 1676, Springer, s. 303 – 312.

- Madria, S., Baseer, Mohammed, B., Kumar, V., Bhowmick, S. [2007] „A transaction model and multiversion concurrency control for mobile database systems”, *Distributed and Parallel Databases (DPD)*, 22:2 – 3, 2007, s. 165 – 196.
- Maguire D., Goodchild M., Rhind D., red. [1997] **Geographical Information Systems: Principles and Applications**, tom 1 i 2, Longman Scientific and Technical, Nowy Jork.
- Mahajan S., Donahoo M. J., Navathe S. B., Ammar M., Malik S. [1998] „Grouping Techniques for Update Propagation in Intermittently Connected Databases”, w: *ICDE* [1998].
- Maier D. [1983] **The Theory of Relational Databases**, Computer Science Press, 1983.
- Maier D., Warren D. S. [1988] **Computing with Logic**, Benjamin Cummings, 1988.
- Maier D., Stein J., Otis A., Purdy A. [1986] „Development of an Object-Oriented DBMS”, *OOPSLA*, 1986.
- Malewicz, G. [2010] „Pregel: a system for large-scale graph processing”, w: *SIGMOD* [2010].
- Malley C., Zdonick S. [1986] „A Knowledge-Based Approach to Query Optimization”, w: *EDS* [1986].
- Mannila H., Toivonen H., Verkamo A. [1994] „Efficient Algorithms for Discovering Association Rules”, w: *KDD-94, AAAI Workshop on Knowledge Discovery in Databases*, Seattle, 1994.
- Manning, C., Schütze, H. [1999] **Foundations of Statistical Natural Language Processing**, MIT Press, 1999.
- Manning, C., Raghavan, P., Schütze, H. [2008] **Introduction to Information Retrieval**, Cambridge University Press, 2008.
- Manola F. [1998] „Towards a Richer Web Object Model”, w: **ACM SIGMOD Record**, 27:1, marzec 1998.
- Manolopoulos, Y., Nanopoulos, A., Papadopoulos, A., Theodoridis, Y. [2005] **R-Trees: Theory and Applications**, Springer, 2005.
- March S., Severance D. [1977] „The Determination of Efficient Record Segmentations and Blocking Factors for Shared Files”, **TODS**, 2:3, wrzesień 1977.
- Mark L., Roussopoulos N., Newsome T., Laohapipattana P. [1992] „Incrementally Maintained Network to Relational Mappings”, **Software Practice & Experience**, 22:12, grudzień 1992.
- Markowitz V., Raz Y. [1983] „ERROL: An Entity-Relationship, Role Oriented, Query Language”, w: *ER Conference* [1983].
- Martin J., Odell J. [1992] **Object Oriented Analysis and Design**, Prentice Hall, 1992.
- Martin J., Chapman K., Leben J. [1989] **DB2-Concepts, Design, and Programming**, Prentice-Hall, 1989.
- Maryanski F. [1980] „Backend Database Machines”, **ACM Computing Surveys**, 12:1, marzec 1980.
- Masunaga Y. [1987] „Multimedia Databases: A Formal Framework”, *Proceedings of the IEEE Office Automation Symposium*, kwiecień 1987.
- Mattison R., **Data Warehousing: Strategies, Technologies, and Techniques**, McGraw-Hill, 1996.
- Maune, D. F. [2001] **Digital Elevation Model Technologies and Applications: The DEM Users Manual**, ASPRS, 2001.
- McCarty, C. i in. [2005] „Marshfield Clinic Personalized Medicine Research Project (PMRP): design, methods and recruitment for a large population-based biobank”, **Personalized Medicine**, 2005, s. 49 – 70.

- McClure, R., Krüger, I. [2005] „SQL DOM: Compile Time Checking of Dynamic SQL Statements”, *Proc. 27th Int. Conf. on Software Engineering*, maj 2005.
- McKinsey [2013] **Big data: The next frontier for innovation, competition, and productivity**, McKinsey Global Institute, 2013, 216 s.
- McLeish M. [1989] „Further Results on the Security of Partitioned Dynamic Statistical Databases”, **TODS**, 14:1, marzec 1989.
- McLeod D., Heimbigner D. [1985] „A Federated Architecture for Information Systems”, **TOOIS**, 3:3, lipiec 1985.
- Mehrotra S. i in. [1992] „The Concurrency Control Problem in Multidatabases: Characteristics and Solutions”, w: *SIGMOD* [1992].
- Melton, J. [2003] **Advanced SQL: 1999—Understanding Object-Relational and Other Advanced Features**, Morgan Kaufmann, 2003.
- Melton J., Mattos N. [1996] *An Overview of SQL3 — The Emerging New Generation of the SQL Standard*, Tutorial No. T5, VLDB, Bombay, wrzesień 1996.
- Melton J., Simon A. R. [1993] **Understanding the New SQL: A Complete Guide**, Morgan Kaufmann.
- Melton, J., Simon, A. R. [2002] **SQL: 1999—Understanding Relational Language Components**, Morgan Kaufmann, 2002.
- Melton J., Bauer J., Kulkarni K. [1991] „Object ADTs (with improvements for value ADTs)”, *ISO WG3 Report X3H2-91-083*, kwiecień 1991.
- Menasce D., Popek G., Muntz R. [1980] „A Locking Protocol for Resource Coordination in Distributed Databases”, **TODS**, 5:2, czerwiec 1980.
- Mendelson A., Maier D. [1979] „Generalized Mutual Dependencies and the Decomposition of Database Relations”, w: *VLDB* [1979].
- Mendelson A., Mihaila G., Milo T. [1997] „Querying the World Wide Web”, **Journal of Digital Libraries**, 1:1, kwiecień 1997.
- Mesnier, M. i in. [2003] „Object-Based Storage”, *IEEE Communications Magazine*, sierpień 2003, s. 84 – 90.
- Metais E., Kedad Z., Comyn-Wattiau C., Bouzeghoub M., „Using Linguistic Knowledge in View Integration: Toward a Third Generation of Tools”, w: **DKE** 23:1, czerwiec 1998.
- Mihailescu, M., Soundararajan, G., Amza, C. „MixApart: Decoupled Analytics for Shared Storage Systems”, w: *USENIX Conf on File And Storage Technologies (FAST)*, 2013.
- Mikkilineni K., Su S. [1988] „An Evaluation of Relational Join Algorithms in a Pipelined Query Processing Environment”, **TSE**, 14:6, czerwiec 1988.
- Mikolajczyk, K., Schmid, C. [2005] „A performance evaluation of local descriptors”, **IEEE Transactions on PAMI**, 10:27, 2005, s. 1615 – 1630.
- Miller, G. A. [1990] „Nouns in WordNet: a lexical inheritance system”, w: **International Journal of Lexicography** 3:4, 1990, s. 245 – 264.
- Miller, H. J. [2004] „Tobler’s First Law and Spatial Analysis”, *Annals of the Association of American Geographers*, 94:2, 2004, s. 284 – 289.
- Milojicic, D. i in. [2002] *Peer-to-Peer Computing*, HP Laboratories Technical Report No. HPL-2002-57, HP Labs, Palo Alto, tekst dostępny na stronie: <http://www.hpl.hp.com/techreports/2002/HPL-2002-57R1.html>.

- Minoura T., Wiederhold G. [1981] „Resilient Extended True-Copy Token Scheme for a Distributed Database”, **TSE**, 8:3, maj 1981.
- Missikoff M., Wiederhold G. [1984] „Toward a Unified Approach for Expert and Database Systems”, w: EDS [1984].
- Mitchell T. [1997] **Machine Learning**, McGraw Hill, Nowy Jork, 1997.
- Mitschang B. [1989] „Extending the Relational Algebra to Capture Complex Objects”, w: **VLDB** [1989].
- Moczar, L. [2015] **Enterprise Lucene and Solr**, Addison Wesley, w przygotowaniu, 2015, 496 s.
- Mohan C. [1993] „IBM’s Relational Database Products: Features and Technologies”, w: **SIGMOD** [1993].
- Mohan C. i in. [1992] „ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks using Write-Ahead Logging”, **TODS**, 17:1, marzec 1992.
- Mohan C., Levine F. [1992] „ARIEL/IM: An Efficient and High-Concurrency Index Management Method Using Write-Ahead Logging”, w: **SIGMOD** [1992].
- Mohan C., Narang I. [1992] „Algorithms for Creating Indexes for Very Large Tables without Quiescing Updates”, w: **SIGMOD** [1992].
- Mohan C. i in. [1992] „ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging”, **TODS**, 17:1, marzec 1992.
- Morris K. i in. [1987] „YAWN! (Yet Another Window on NAIL!)”, w: **ICDE** [1987].
- Morris K., Ullman J., VanGelden A. [1986] „Design Overview of the NAIL! System”, *Proceedings of the Third International Conference on Logic Programming*, Springer-Verlag, 1986.
- Morris R. [1968] „Scatter Storage Techniques”, **CACM**, 11:1, styczeń 1968.
- Morsi M., Navathe S., Kim H. [1992] „An Extensible Object-Oriented Database Testbed”, w: **ICDE** [1992].
- Moss J. [1982] „Nested Transactions and Reliable Distributed Computing”, *Proceedings of the Symposium on Reliability in Distributed Software and Database Systems*, IEEE CS, lipiec 1982.
- Motro A. [1987] „Superviews: Virtual Integration of Multiple Databases”, **TSE**, 13:7, lipiec 1987.
- Mouratidis, K. i in. [2006] „Continuous nearest neighbor monitoring in road networks”, w: **VLDB** [2006], s. 43 – 54.
- Mukkamala R. [1989] „Measuring the Effect of Data Distribution and Replication Models on Performance Evaluation of Distributed Systems”, w: **ICDE** [1989].
- Mumick I., Finkelstein S., Pirahesh H., Ramakrishnan R. [1990a] „Magic Is Relevant”, w: **SIGMOD** [1990].
- Mumick I., Pirahesh H., Ramakrishnan R. [1990] „The Magic of Duplicates and Aggregates”, w: **VLDB** [1990].
- Muralikrishna M. [1992] „Improved Unnesting Algorithms for Join and Aggregate SQL Queries”, w: **VLDB** [1992].
- Muralikrishna M., DeWitt D. [1988] „Equi-depth Histograms for Estimating Selectivity Factors for Multi-dimensional Queries”, w: **SIGMOD** [1988].
- Murthy, A.C., Vavilapalli, V.K. [2014] **Apache Hadoop YARN: Moving beyond MapReduce and Batch Processing with Apache Hadoop 2**, Addison Wesley, 2014, 304 s.

- Mylopoulos J., Bernstein P., Wong H. [1980] „A Language Facility for Designing Database-Intensive Applications”, **TODS**, 5:2, czerwiec 1980.
- Naedele, M. [2003] Standards for XML and Web Services Security, **IEEE Computer**, 36:4, kwiecień 2003, s. 96 – 98.
- Naish L., Thom J. [1983] „The MU-PROLOG Deductive Database”, raport techniczny 83/10, Department of Computer Science, University of Melbourne, 1983.
- Natan R. [2005] **Implementing Database Security and Auditing: Includes Examples from Oracle, SQL Server, DB2 UDB, and Sybase**, Digital Press, 2005.
- Navathe S. [1980] „An Intuitive View to Normalize Network-Structured Data”, w: *VLDB* [1980].
- Navathe, S., Balaraman, A. [1991] „A Transaction Architecture for a General Purpose Semantic Data Model”, w: *ER* [1991], s. 511 – 541.
- Navathe S.B., Karlapalem K., Ra M.Y. [1996] „A Mixed Fragmentation Methodology for the Initial Distributed Database Design”, **Journal of Computers and Software Engineering**, 3:4, 1996.
- Navathe, S.B. i in. [1994] „Object Modeling Using Classification in CANDIDE and Its Application”, w: Dogac i in. [1994].
- Navathe S., Ahmed R. [1989] „A Temporal Relational Model and Query Language”, **Information Sciences**, 47:2, marzec 1989, s. 147 – 175.
- Navathe S., Gadgil S. [1982] „A Methodology for View Integration in Logical Database Design”, w: *VLDB* [1982].
- Navathe S., Kerschberg L. [1986] „Role of Data Dictionaries in Database Design”, **Information and Management**, 10:1, styczeń 1986.
- Navathe S., Savasere A. [1996] „A Practical Schema Integration Facility Using an Object Oriented Approach”, w: **Multidatabase Systems** (A. Elmagarmid, O. Bukhres, red.), Prentice-Hall, 1996.
- Navathe S., Schkolnick M. [1978] „View Representation in Logical Database Design”, w: *SIGMOD* [1978].
- Navathe S., Ceri S., Wiederhold G., Dou J. [1984] „Vertical Partitioning Algorithms for Database Design”, **TODS**, 9:4, grudzień 1984.
- Navathe S., Elmasri R., Larson J. [1986] „Integrating User Views in Database Design”, **IEEE Computer**, 19:1, styczeń 1986.
- Navathe, S., Patil, U., Guan, W. [2007] „Genomic and Proteomic Databases: Foundations, Current Status and Future Applications”, w: **Journal of Computer Science and Engineering**, Korean Institute of Information Scientists and Engineers (KIISE), 1:1, 2007, s. 1 – 30.
- Navathe S., Sashidhar T., Elmasri R. [1984a] „Relationship Merging in Schema Integration”, w: *VLDB* [1984].
- Negri M., Pelagatti S., Sbatella L. [1991] „Formal Semantics of SQL Queries”, **TODS**, 16:3, wrzesień 1991.
- Ng P. [1981] „Further Analysis of the Entity-Relationship Approach to Database Design”, **TSE**, 7:1, styczeń 1981.
- Ngu, A. [1989] „Transaction Modeling”, w: *ICDE* [1989], s. 234 – 241.
- Nicolas J. [1978] „Mutual Dependencies and Some Results on Undecomposable Relations”, w: *VLDB* [1978].

- Nicolas J. [1997] „Deductive Object-Oriented Databases, Technology, Products, and Applications: Where Are We?”, *Proceedings of the Symposium on Digital Media Information Base (DMIB'97)*, Nara, Japonia, listopad 1997.
- Nicolas J., Phipps G., Derr M., Ross K. [1991] „Glue-NAIL!: A Deductive Database System”, w: *SIGMOD* [1991].
- Niemiec, R. [2008] **Oracle Database 10g Performance Tuning Tips & Techniques**, McGraw Hill Osborne Media, 2008, 967 s.
- Nievergelt J. [1974] „Binary Search Trees and File Organization”, *ACM Computing Surveys*, 6:3, wrzesień 1974.
- Nievergelt J., Hinterberger H., Seveik K. [1984]. „The Grid File: An Adaptable Symmetric Multikey File Structure”, *TODS*, 9:1, marzec 1984, s. 38 – 71.
- Nijssen G., red. [1976] **Modelling in Data Base Management Systems**, North-Holland, 1976.
- Nijssen G., red. [1977] **Architecture and Models in Data Base Management Systems**, North-Holland, 1977.
- Nwosu K., Berra P., Thuraishingham B., red. [1996], **Design and Implementation of Multimedia Database Management Systems**, Kluwer Academic, 1996.
- O'Neil, P., O'Neil, P. [2001] **Database: Principles, Programming, Performance**, Morgan Kaufmann, 1994.
- Obermarck R. [1982] „Distributed Deadlock Detection Algorithms”, *TODS*, 7:2, czerwiec 1982.
- Oh Y.-C., [1999] „Secure Database Modeling and Design”, praca doktorska, College of Computing, Georgia Institute of Technology, marzec 1999.
- Ohsuga S. [1982] „Knowledge Based Systems as a New Interactive Computer System of the Next Generation”, w: **Computer Science and Technologies**, North-Holland, 1982.
- Olken, F., Jagadish, J. [2003] Management for Integrative Biology, **OMICS: A Journal of Integrative Biology**, 7:1, styczeń 2003.
- Olle T. [1978] **The CODASYL Approach to Data Base Management**, Wiley, 1978.
- Olle T., Sol H., Verrijn-Stuart A., red. [1982] **Information System Design Methodology**, North-Holland, 1982.
- Olston, C. i in. [2008] Pig Latin: A Not-So-Foreign language for Data Processing, w: *SIGMOD* [2008].
- Omiecinski E., Scheuermann P. [1990] „A Parallel Algorithm for Record Clustering”, *TODS*, 15:4, grudzień 1990.
- Omura J. K. [1990] „Novel Applications of Cryptography in Digital Communications”, **IEEE Communications** 28:5, maj 1990, s. 21 – 29.
- O'Neil, P., Graefe, G. [1995] „Multi-Table Joins Through Bitmapped Join Indices”, **SIGMOD Record**, tom 24, nr 3, 1995.
- Open GIS Consortium, Inc. [1999] „OpenGIS® Simple Features Specification for SQL”, wersja 1.1, OpenGIS Project Document 99-049, maj 1999.
- Open GIS Consortium, Inc. [2003] „OpenGIS® Geography Markup Language (GML) Implementation Specification”, wersja 3, OGC 02-023r4, 2003.
- Oracle [2005] **Oracle 10, Introduction to LDAP and Oracle Internet Directory**, 10g Release 2, Oracle Corporation, 2005.



- Oracle [2007] **Oracle Label Security Administrator's Guide, 11g (release 11.1)**, Part no. B28529-01, Oracle, tekst dostępny na stronie: [http://download.oracle.com/docs/cd/B28359\\_01/network.111/b28529/intro.htm](http://download.oracle.com/docs/cd/B28359_01/network.111/b28529/intro.htm).
- Oracle [2008] **Oracle 11 Distributed Database Concepts**, 11g Release 1, Oracle Corporation, 2008.
- Oracle [2009] „An Oracle White Paper: Leading Practices for Driving Down the Costs of Managing Your Oracle Identity and Access Management Suite”, Oracle, kwiecień 2009.
- Osborn S. [1977] **Normal Forms for Relational Databases**, praca doktorska, University of Waterloo, 1977.
- Osborn S. [1989] „The Role of Polymorphism in Schema Evolution in an Object-Oriented Database”, **TKDE**, 1:3, wrzesień 1989.
- Osborn S. [1979] „Towards a Universal Relation Interface”, w: **VLDB** [1979].
- Ozsoyoglu G., Ozsoyoglu Z., Matos V. [1985] „Extending Relational Algebra and Relational Calculus with Set Valued Attributes and Aggregate Functions”, **TODS**, 12:4, grudzień 1987.
- Ozsoyoglu Z., Yuan L. [1987] „A New Normal Form for Nested Relations”, **TODS**, 12:1, marzec 1987.
- Ozsu M. T., Valduriez P. [1999] **Principles of Distributed Database Systems**, wydanie drugie, Prentice-Hall, 1999.
- Palanisamy, B. i in. [2011] „Purlieus: locality-aware resource allocation for MapReduce in a cloud”, w: *Proc. ACM/IEEE Int. Conf for High Perf Computing, Networking, Storage and Analysis*, (SC) 2011.
- Palanisamy, B. i in. [2014] „VNCache: Map Reduce Analysis for Cloud-archived Data”, *Proc. 14th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing*, 2014.
- Palanisamy, B., Singh, A., Liu, Ling [2015] „Cost-effective Resource Provisioning for MapReduce in a Cloud”, **IEEE TPDS**, 26:5, maj 2015.
- Papadias, D. i in. [2003] „Query Processing in Spatial Network Databases”, w: **VLDB** [2003] s. 802 – 813.
- Papadimitriou C. [1979] „The Serializability of Concurrent Database Updates”, **JACM**, 26:4, październik 1979.
- Papadimitriou C. [1986] **The Theory of Database Concurrency Control**, Computer Science Press, 1986.
- Papadimitriou C., Kanellakis P. [1979] „On Concurrency Control by Multiple Versions”, **TODS**, 9:1, marzec 1974.
- Papazoglou M., Valder W. [1989] **Relational Database Management: A Systems Programming Approach**, Prentice-Hall, 1989.
- Paredaens J., Van Gucht D. [1992] „Converting Nested Algebra Expressions into Flat Algebra Expressions”, **TODS**, 17:1, marzec 1992.
- Parent C., Spaccapietra S. [1985] „An Algebra for a General Entity-Relationship Model”, **TSE**, 11:7, lipiec 1985.
- Paris J. [1986] „Voting with Witnesses: A Consistency Scheme for Replicated Files”, w: **ICDE** [1986].
- Park J., Chen M., Yu P. [1995] „An Effective Hash Based Algorithm for Mining Association Rules”, w: **SIGMOD** [1995].

- Parker Z., Poe, S., Vrbsky, S.V. [2013] „Comparing NoSQL MongoDB to an SQL DB”, *Proc. 51st ACM Southeast Conference [ACMSE '13]*, Savannah, GA, 2013.
- Paton A. W., red. [1999] **Active Rules in Database Systems**, Springer Verlag, 1999.
- Paton N. W., Diaz O. [1999] Survey of Active Database Systems, **ACM Computing Surveys**, 31:1, marzec 1999, s. 63-103.
- Patterson D., Gibson G., Katz R. [1988]. „A Case for Redundant Arrays of Inexpensive Disks (RAID)”, w: *SIGMOD* [1988].
- Paul H. i in. [1987] „Architecture and Implementation of the Darmstadt Database Kernel System”, w: *SIGMOD* [1987].
- Pavlo, A. i in. [2009] A Comparison of Approaches to Large Scale Data Analysis, w: *SIGMOD* [2009].
- Pazandak P., Srivastava J., „Evaluating Object DBMSs for Multimedia”, **IEEE Multimedia**, 4:3, s. 34 – 49.
- Pazos-Rangel, R. i in. [2006] „Least Likely to Use: A New Page Replacement Strategy for Improving Database Management System Response Time”, w: *Proc. CSR 2006: Computer Science - Theory and Applications*, St. Petersburg, Rosja, **LNCS**, tom 3967, Springer, 2006, s. 314–323.
- PDES [1991] „A High-Lead Architecture for Implementing a PDES/STEP Data Sharing Environment”, Publication Number PT 1017.03.00, PDES Inc., maj 1991.
- Pearson P. i in. [1994] „The Status of Online Mendelian inheritance in Man (OMIM) Medio 1994”, **Nucleic Acids Research**, 22:17, 1994.
- Peckham J., Maryanski F. [1988] „Semantic Data Models”, **ACM Computing Surveys**, 20:3, wrzesień 1988, s. 153 – 189.
- Peng, T., Tsou, M. [2003] **Internet GIS: Distributed Geographic Information Services for the Internet and Wireless Network**, Wiley, 2003.
- Pfleeger C.P., Pfleeger S. [2007] **Security in Computing**, wydanie czwarte, Prentice Hall, 2007.
- Phipps G., Derr M., Ross K. [1991] „Glue-NAIL!: A Deductive Database System”, w: *SIGMOD* [1991].
- Piatetsky-Shapiro G., Frauley W., red. [1991] **Knowledge Discovery in Databases**, AAAI Press/MIT Press, 1991.
- Pistor P., Anderson F. [1986] „Designing a Generalized NF2 Model with an SQL-type Language Interface”, w: *VLDB* [1986], s. 278 – 285.
- Pitoura, E., Bhargava, B. [1995] „Maintaining Consistency of Data in Mobile Distributed Environments”, w: *15th ICDCS*, maj 1995, s. 404 – 413.
- Pitoura E., Samaras G. [1998] **Data Management for Mobile Computing**, Kluwer, 1998.
- Pitoura E., Bukhres O., Elmagarmid A. [1995] „Object Orientation in Multidata-Base Systems”, **ACM Computing Surveys**, 27:2, czerwiec 1995.
- Polavarapu, N. i in. [2005] „Investigation into Biomedical Literature Screening Using Support Vector Machines”, w: *Proc. 4thInt. IEEE Computational Systems Bioinformatics Conference (CSB'05)*, sierpień 2005, s. 366 – 374.
- Ponceleon D. i in. [1999] „CueVideo: Automated Multimedia Indexing and Retrieval”, *Proc. 7th ACM Multimedia Conf.*, Orlando, Fl, październik 1999, s. 199.
- Ponniah, P. [2010] **Data Warehousing Fundamentals for IT Professionals**, wydanie drugie, Wiley Interscience, 2010, 600 s.

- Poosala V., Ioannidis Y., Haas P., Shekita E. [1996] „Improved Histograms for Selectivity Estimation of Range Predicates”, w: *SIGMOD* [1996].
- Porter, M. F. [1980] „An algorithm for suffix stripping”, **Program**, 14:3, s. 130 – 137.
- Ports, D.R.K., Grittner, K. [2012] „Serializable Snapshot Isolation in PostgreSQL”, **Proceedings of VLDB**, 5:12, 2012, s. 1850 – 1861.
- Potter B., Sinclair J., Till D. [1996] **An Introduction to Formal Specification and Z**, wydanie drugie, Prentice-Hall, 1996.
- Prabhakaran, B. [1996] **Multimedia Database Management Systems**, Springer-Verlag, 1996.
- Prasad, S. i in. [2004] „SyD: A Middleware Testbed for Collaborative Applications over Small Heterogeneous Devices and Data Stores”, *Proc. ACM/IFIP/USENIX 5th International Middleware Conference (MW-04)*, Toronto, Kanada, październik 2004.
- Price, B. [2004] „ESRI Systems Integration Technical Brief—ArcSDE High-Availability Overview”, ESRI, 2004, Rev 2 (<http://www.lincoln.ne.gov/city/pworks/gis/pdf/arcsde.pdf>).
- Rabitti F., Bertino E., Kim W., Woelk D. [1991] „A Model of Authorization for Next-Generation Database Systems”, **TODS**, 16:1, marzec 1991.
- Ramakrishnan, R., Gehrke, J. [2003] **Database Management Systems**, wydanie trzecie, McGraw-Hill, 2003.
- Ramakrishnan R., Ullman J. [1995] „Survey of Research in Deductive Database Systems”, **Journal Of Logic Programming**, 23:2, 1995, s. 125 – 149.
- Ramakrishnan R., red. [1995] **Applications of Logic Databases**, Kluwer Academic, 1995.
- Ramakrishnan R., Srivastava D., Sudarshan S. [1992] „{CORAL}: {C}ontrol, {R}elations and {L}ogic”, w: VLDB [1992].
- Ramakrishnan R., Srivastava D., Sudarshan S., Sheshadri P. [1993] „Implementation of the {CORAL} Deductive Database System”, w: SIGMOD [1993].
- Ramamoorthy C., Wah B. [1979] „The Placement of Relations on a Distributed Relational Database”, *Proceedings of the First International Conference on Distributed Computing Systems*, IEEE CS, 1979.
- Ramesh V., Ram S. [1997] „Integrity Constraint Integration in Heterogeneous Databases an Enhanced Methodology for Schema Integration”, **Information Systems**, 22:8, grudzień 1997, s. 423 – 446.
- Ratnasamy, S. i in. [2001] „A Scalable Content-Addressable Network”, SIGCOMM 2001.
- Reed D. [1983] „Implementing Atomic Actions on Decentralized Data”, **TOCS**, 1:1, luty 1983.
- Reese, G. [1997] **Database Programming with JDBC and Java**, O'Reilly, 1997.
- Reisner P. [1977] „Use of Psychological Experimentation as an Aid to Development of a Query Language”, **TSE**, 3:3, maj 1977.
- Reisner P. [1981] „Human Factors Studies of Database Query Languages: A Survey and Assessment”, **ACM Computing Surveys**, 13:1, marzec 1981.
- Reiter R. [1984] „Towards a Logical Reconstruction of Relational Database Theory”, w: Brodie i in., rozdz. 8 [1984].
- Reuter, A. [1980] „A Fast Transaction Oriented Logging Scheme for UNDO recovery”, **TSE** 6:4, s. 348 – 356.

- Revilak, S., O'Neil, P., O'Neil, E. [2011] „Precisely Serializable Snapshot Isolation (PSSI)”, w: *ICDE* [2011], s. 482 – 493.
- Ries D., Stonebraker M. [1977] „Effects of Locking Granularity in a Database Management System”, **TODS**, 2:3, wrzesień 1977.
- Rissanen J. [1977] „Independent Components of Relations”, **TODS**, 2:4, grudzień 1977.
- Rivest, R. et al. [1978] „A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”, **CACM**, 21:2, luty 1978, s. 120 – 126.
- Robbins R. [1993] „Genome Informatics: Requirements and Challenges”, *Proceedings of the Second International Conference on Bioinformatics, Supercomputing and Complex Genome Analysis*, World Scientific Publishing, 1993.
- Robertson, S. [1997] „The Probability Ranking Principle in IR”, w: **Readings in Information Retrieval** (Jones, K. S., Willett, P., red.), Morgan Kaufmann Multimedia Information and Systems Series, s. 281 – 286.
- Robertson, S., Walker, S., Hancock-Beaulieu, M. [1995] „Large Test Collection Experiments on an Operational, Interactive System: Okapi at TREC”, **Information Processing and Management**, 31, s. 345 – 360.
- Rocchio, J. [1971] „Relevance Feedback in Information Retrieval”, w: **The SMART Retrieval System: Experiments in Automatic Document Processing**, (G. Salton, red.), Prentice-Hall, s. 313 – 323.
- Rosenkrantz, D., Stearns, D., Lewis, P. [1978] System-Level Concurrency Control for Distributed Database Systems, **TODS**, 3:2, s. 178 – 198.
- Rotem, D. [1991] „Spatial Join Indices”, w: *ICDE* [1991].
- Roth M.A., Korth H.F., Silberschatz A. [1988] Extended Algebra and Calculus for non-1NF Relational Databases”, **TODS**, 13:4, 1988, s. 389 – 417.
- Roth M., Korth H. [1987] „The Design of Non-1NF Relational Databases into Nested Normal Form”, w: *SIGMOD* [1987].
- Rothnie J. i in. [1980] „Introduction to a System for Distributed Databases (SDD-1)”, **TODS**, 5:1, marzec 1980.
- Roussopoulos N. [1991] „An Incremental Access Method for View-Cache: Concept, Algorithms, and Cost Analysis”, **TODS**, 16:3, wrzesień 1991.
- Roussopoulos, N., Kelley, S., Vincent, F. [1995] „Nearest Neighbor Queries”, w: *SIGMOD* [1995], s. 71 – 79.
- Rozen S., Shasha D. [1991] „A Framework for Automating Physical Database Design”, w: *VLDB* [1991].
- Rudensteiner E. [1992] „Multiview: A Methodology for Supporting Multiple Views in Object-Oriented Databases”, w: *VLDB* [1992].
- Ruemmler C., Wilkes, J. [1994] „An Introduction to Disk Drive Modeling”, **IEEE Computer**, 27:3, marzec 1994, s.17 – 27.
- Rumbaugh J., Blaha M., Premerlam W., Eddy F., Lorensen W. [1991] **Object Oriented Modeling and Design**, Prentice-Hall, 1991.
- Rumbaugh, J., Jacobson, I., Booch, G. [1999] **The Unified Modeling Language Reference Manual**, Addison-Wesley, 1999.

- Rusinkiewicz M. i in. [1988] „OMNIBASE — A Loosely Coupled: Design and Implementation of a Multidatabase System”, **IEEE Distributed Processing Newsletter**, 10:2, listopad 1988.
- Rustin R., red. [1972] **Data Base Systems**, Prentice-Hall, 1972.
- Rustin R., red. [1974] *Proceedings of the BJNAV2*.
- Sacca D., Zaniolo C. [1987] „Implementation of Recursive Queries for a Data Language Based on Pure Horn Clauses”, *Proceedings of the Fourth International Conference on Logic Programming*, MIT Press, 1986.
- Sadri F., Ullman J. [1982] „Template Dependencies: A Large Class of Dependencies in Relational Databases and Its Complete Axiomatization”, **JACM**, 29:2, kwiecień 1982.
- Sagiv Y., Yannakakis M. [1981] „Equivalence among Relational Expressions with the Union and Difference Operators”, **JACM**, 27:4, listopad 1981.
- Sahay, S. i in. [2008] „Discovering Semantic Biomedical Relations Utilizing the Web”, w: **Journal of ACM Transactions on Knowledge Discovery from Data (TKDD)**, Special issue on Bioinformatics, 2:1, 2008.
- Sakai H. [1980] „Entity-Relationship Approach to Conceptual Schema Design”, w: *SIGMOD* [1980].
- Salem, K., Garcia-Molina, H. [1986] „Disk Striping”, w: *ICDE* [1986], s. 336 – 342.
- Salton, G. [1968] **Automatic Information Organization and Retrieval**, McGraw Hill, 1968.
- Salton, G. [1971] **The SMART Retrieval System—Experiments in Automatic Document Processing**, Prentice-Hall, 1971.
- Salton, G. [1990] „Full Text Information Processing Using the Smart System”, **IEEE Data Engineering Bulletin** 13:1, 1990, s. 2 – 9.
- Salton G., Buckley C. [1991] „Global Text Matching for Information Retrieval”, w: **Science**, 253, sierpień 1991.
- Salton, G., Yang, C. S., Yu, C. T. [1975] „A theory of term importance in automatic text analysis”, **Journal of the American Society for Information Science**, 26, s. 33 – 44 (1975).
- Salzberg B. [1988] **File Structures: An Analytic Approach**, Prentice-Hall, 1988.
- Salzberg B. i in. [1990] „FastSort: A Distributed Single-Input Single-Output External Sort”, w: *SIGMOD* [1990].
- Samet H. [1990] **The Design and Analysis of Spatial Data Structures**, Addison-Wesley, 1990.
- Samet H. [1990a] **Applications of Spatial Data Structures: Computer Graphics, Image Processing and GIS**, Addison-Wesley, 1990.
- Sammut C., Sammut R. [1983] „The Implementation of UNSW-PROLOG”, **The Australian Computer Journal**, maj 1983.
- Santucci, G. [1998] „Semantic Schema Refinements for Multilevel Schema Integration”, **DKE**, 25:3, 1998, s. 301 – 326.
- Sarasua W., O'Neill W. [1999] „GIS in Transportation”, w: Taylor, Francis [1999].
- Sarawagi S., Thomas S., Agrawal R. [1998] „Integrating Association Rules Mining with Relational Database Systems: Alternatives and Implications”, w: *SIGMOD* [1998].
- Savasere A., Omiecinski E., Navathe S. [1995] „An Efficient Algorithm for Mining Association Rules”, w: *VLDB* [1995].

- Savasere A., Omiecinski E., Navathe S. [1998] „Mining for Strong Negative Association in a Large Database of Customer Transactions”, w: *ICDE* [1998].
- Schatz B. [1995] „Information Analysis in the Net: The Interspace of the Twenty-First Century”, *Keynote Plenary Lecture at American Society for Information Science (ASIS) Annual Meeting*, Chicago, 11 października, 1995.
- Schatz B. [1997] „Information Retrieval in Digital Libraries: Bringing Search to the Net”, **Science**, nr 275, 17 stycznia 1997.
- Schek H. J., Scholl M. H. [1986] „The Relational Model with Relation-Valued Attributes”, **Information Systems**, 11:2, 1986.
- Schek H. J., Paul H. B., Scholl M. H., Weikum G. [1990] „The DASDBS Project: Objects, Experiences, and Future Projects”, **IEEE TKDE**, 2:1, 1990.
- Scheuermann P., Schiffner C., Weber H. [1979] „Abstraction Capabilities and Invariant Properties Modeling within the Entity-Relationship Approach”, w: *ER Conference* [1979].
- Schlimmer J., Mitchell T., McDermott J. [1991] „Justification Based Refinement of Expert Knowledge” w: *Platesky-Shapiro, Frawley* [1991].
- Schmarzo, B. [2013] **Big Data: Understanding How Data Powers Big Business**, Wiley, 2013, 240 s.
- Schlossnagle, G. [2005] **Advanced PHP Programming**, Sams, 2005.
- Schmidt J., Swenson J. [1975] „On the Semantics of the Relational Model”, w: *SIGMOD* [1975].
- Schneider, R. D. [2006] **MySQL Database Design and Tuning**, MySQL Press, 2006.
- Scholl, M. O., Voisard, A., Rigaux, P. [2001] **Spatial Database Management Systems**, Morgan Kaufman, 2001.
- Sciore E. [1982] „A Complete Axiomatization for Full Join Dependencies”, **JACM**, 29:2, kwiecień 1982.
- Scott, M., Fowler, K. [1997] **UML Distilled: Applying the Standard Object Modeling Language**, Addison-Wesley, 1997.
- Selinger P. i in. [1979] „Access Path Selection in a Relational Database Management System”, w: *SIGMOD* [1979].
- Senko M. [1975] „Specification of Stored Data Structures and Desired Output in DIAM II with FORAL”, w: *VLDB* [1975].
- Senko M. [1980] „A Query Maintenance Language for the Data Independent Accessing Model II”, **Information Systems**, 5:4, 1980.
- Shapiro L. [1986] „Join Processing in Database Systems with Large Main Memories”, **TODS**, 11:3, 1986.
- Shasha, D., Bonnet, P. [2002] **Database Tuning: Principles, Experiments, and Troubleshooting Techniques**, Morgan Kaufmann, wydanie poprawione, 2002.
- Shasha D., Goodman N. [1988] „Concurrent Search Structure Algorithms”, **TODS**, 13:1, marzec 1988.
- Shekhar, S., Chawla, S. [2003] **Spatial Databases, A Tour**, Prentice-Hall, 2003.
- Shekhar, S., Xong, H. [2008] **Encyclopedia of GIS**, Springer Link (w serwisie internetowym).
- Shekita E., Carey M. [1989] „Performance Enhancement Through Replication in an Object-Oriented DBMS”, w: *SIGMOD* [1989].

- Shenoy S., Ozsoyoglu Z. [1989] „Design and Implementation of a Semantic Query Optimizer”, **TKDE**, 1:3, wrzesień 1989.
- Sheth A.P., Larson J.A. [1990] „Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases”, **ACM Computing Surveys**, 22:3, wrzesień 1990, s. 183 – 236.
- Sheth A., Gala S., Navathe S. [1993] „On Automatic Reasoning for Schema Integration”, w: **International Journal of Intelligent Co-operative Information Systems**, 2:1, marzec 1993.
- Sheth A., Larson J., Cornelio A., Navathe S. [1988] „A Tool for Integrating Conceptual Schemas and User Views”, w: *ICDE* [1988].
- Shipman D. [1981] „The Functional Data Model and the Data Language DAPLEX”, **TODS**, 6:1, marzec 1981.
- Shlaer S., Mellor S. [1988] **Object-Oriented System Analysis: Modeling the World in Data**, Yourdon Press, 1988.
- Shneiderman B., red. [1978] **Databases: Improving Usability and Responsiveness**, Academic Press, 1978.
- Shvachko, K.V. [2012] „HDFS Scalability: the limits of growth”, publikacje z konferencji Usenix, **Login**, tom 35, nr 2, s. 6 – 16, kwiecień 2010  
(<https://www.usenix.org/legacy/publications/login/2010-04/openpdfs/shvachko.pdf>).
- Sibley E., Kerschberg L. [1977] „Data Architecture and Data Model Considerations”, *NCC, AFIPS*, 46, 1977.
- Siegel M., Madnick S. [1991] „A Metadata Approach to Resolving Semantic Conflicts”, w: *VLDB* [1991].
- Siegel M., Sciore E., Salveter S. [1992] „A Method for Automatic Rule Derivation to Support Semantic Query Optimization”, **TODS**, 17:4, grudzień 1992.
- SIGMOD [1974] *Proceedings of the ACM SIGMOD-SIGFIDET Conference on Data Description, Access, and Control*, Rustin R., red., maj 1974.
- SIGMOD [1975] *Proceedings of the 1975 ACM SIGMOD International Conference on Management of Data*, King F., red., San Jose, CA, maj 1975.
- SIGMOD [1976] *Proceedings of the 1976 ACM SIGMOD International Conference on Management of Data*, Rothnie J., red., Waszyngton, czerwiec 1976.
- SIGMOD [1977] *Proceedings of the 1977 ACM SIGMOD Internaitonal Conference on Management of Data*, Smith D., red., Toronto, sierpień 1977.
- SIGMOD [1978] *Proceedings of the 1978 ACM SIGMOD International Conference on Management of Data*, Lowenthal E., Dale N., red., Austin, TX, maj/czerwiec 1978.
- SIGMOD [1979] *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, Bernstein P., red., Boston, MA, maj/czerwiec 1979.
- SIGMOD [1980] *Proceedings of the 1980 ACM SIGMOD International Conference on Management of Data*, Chen P., Sprowls R., red., Santa Monica, CA, maj 1980.
- SIGMOD [1981] *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, Lien Y., red., Ann Arbor, MI, kwiecień/maj 1981.
- SIGMOD [1982] *Proceedings of the 1982 ACM SIGMOD International Conference on Management of Data*, Schkolnick M., red., Orlando, FL, czerwiec 1982.



SIGMOD [1983] *Proceedings of the 1983 ACM SIGMOD International Conference on Management of Data*, DeWitt D., Gardarin G., red., San Jose, CA, maj 1983.

SIGMOD [1984] *Proceedings of the 1984 ACM SIGMOD Internaitonal Conference on Management of Data*, Yormark E., red., Boston, MA, czerwiec 1984.

SIGMOD [1985] *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data*, Navathe S., red., Austin, TX, maj 1985.

SIGMOD [1986] *Proceedings of the 1986 ACM SIGMOD Internaitonal Conference on Management of Data*, Zaniolo C., red., Waszyngton, maj 1986.

SIGMOD [1987] *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, Dayal U., Traiger I., red., San Francisco, CA, maj 1987.

SIGMOD [1988] *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, Boral H., Larson P., red., Chicago, czerwiec 1988.

SIGMOD [1989] *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, Clifford J., Lindsay B., Maier D., red., Portland, OR, czerwiec 1989.

SIGMOD [1990] *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, Garcia-Molina H., Jagadish H., red., Atlantic City, NJ, czerwiec 1990.

SIGMOD [1991] *Proceedings of the 1991 ACM SIGMOD Internaitonal Conference on Management of Data*, Clifford J., King R., red., Denver, CO, czerwiec 1991.

SIGMOD [1992] *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, Stonebraker M., red., San Diego, CA, czerwiec 1992.

SIGMOD [1993] *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, Buneman P., Jajodia S., red., Waszyngton, czerwiec 1993.

SIGMOD [1994] *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, Snodgrass R. T., Winslett M., red., Minneapolis, MN, czerwiec 1994.

SIGMOD [1995] *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, Carey M., Schneider D. A., red., Minneapolis, MN, czerwiec 1995.

SIGMOD [1996] *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, Jagadish H. V., Mumick I. P., red., Montreal, czerwiec 1996.

SIGMOD [1997] *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, Peckham J., red., Tucson, AZ, maj 1997.

SIGMOD [1998] *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, Haas L., Tiwary A., red., Seattle, WA, czerwiec 1998.

SIGMOD [1999] *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, Faloutsos C., red., Filadelfia, PA, maj 1999.

SIGMOD [2000] *Proceedings of 2000 ACM SIGMOD International Conference on Management of Data*, Chen, W., Naughton J., Bernstein, P., red., Dallas, TX, maj 2000.

SIGMOD [2001] *Proceedings of 2001 ACM SIGMOD International Conference on Management of Data*, Aref, W., red., Santa Barbara, CA, maj 2001.

SIGMOD [2002] *Proceedings of 2002 ACM SIGMOD International Conference on Management of Data*, Franklin, M., Moon, B., Ailamaki, A., red., Madison, WI, czerwiec 2002.

- SIGMOD [2003] *Proceedings of 2003 ACM SIGMOD International Conference on Management of Data*, Halevy, Y., Zachary, G., Doan, A., red., San Diego, CA, czerwiec 2003.
- SIGMOD [2004] *Proceedings of 2004 ACM SIGMOD International Conference on Management of Data*, Weikum, G., Christian König, A., DeBloch, S., red., Paryż, Francja, czerwiec 2004.
- SIGMOD [2005] *Proceedings of 2005 ACM SIGMOD International Conference on Management of Data*, Widom, J., red., Baltimore, MD, czerwiec 2005.
- SIGMOD [2006] *Proceedings of 2006 ACM SIGMOD International Conference on Management of Data*, Chaudhari, S., Hristidis, V., Polyzotis, N., red., Chicago, IL, czerwiec 2006.
- SIGMOD [2007] *Proceedings of 2007 ACM SIGMOD International Conference on Management of Data*, Chan, C.-Y., Ooi, B.-C., Zhou, A., red., Pekin, Chiny, czerwiec 2007.
- SIGMOD [2008] *Proceedings of 2008 ACM SIGMOD International Conference on Management of Data*, Wang, J. T.-L., red., Vancouver, Kanada, czerwiec 2008.
- SIGMOD [2009] *Proceedings of 2009 ACM SIGMOD International Conference on Management of Data*, Cetintemel, U., Zdonik, S., Kossman, D., Tatbul, N., red., Providence, RI, czerwiec – lipiec 2009.
- SIGMOD [2010] *Proceedings of 2010 ACM SIGMOD International Conference on Management of Data*, Elmagarmid, A.K., Agrawal, D., red., Indianapolis, IN, czerwiec 2010.
- SIGMOD [2011] *Proceedings of 2011 ACM SIGMOD International Conference on Management of Data*, Sellis, T., Miller, R., Kementsietsidis, A., Velegrakis, Y., red., Ateny, Grecja, czerwiec 2011.
- SIGMOD [2012] *Proceedings of 2012 ACM SIGMOD International Conference on Management of Data*, Selcuk Candan, K., Chen, Yi, Snodgrass, R., Gravano, L., Fuxman, A., red., Scottsdale, Arizona, czerwiec 2012.
- SIGMOD [2013] *Proceedings of 2013 ACM SIGMOD International Conference on Management of Data*, Ross, K., Srivastava, D., Papadias, D., red., Nowy Jork, czerwiec 2013.
- SIGMOD [2014] *Proceedings of 2014 ACM SIGMOD International Conference on Management of Data*, Dyreson, C., Li, Feifei, Ozsu, T., red., Snowbird, UT, czerwiec 2014.
- SIGMOD [2015] *Proceedings of 2015 ACM SIGMOD International Conference on Management of Data*, Melbourne, Australia, maj/czerwiec 2015, w przygotowaniu.
- Silberschatz A., Korth H., Sudarshan S. [2011] **Database System Concepts**, wydanie szóste, McGraw-Hill, 2011.
- Silberschatz A., Stonebraker M., Ullman J. [1990] „Database Systems: Achievements and Opportunities”, w: **ACM SIGMOD Record**, 19:4, grudzień 1990.
- Simon, H. A. [1971] „Designing Organizations for an Information-Rich World”, w: **Computers, Communications and the Public Interest**, (Greenberger, M., red.), The Johns Hopkins University Press, 1971, (s. 37 – 72).
- Sion, R., Atallah, M., Prabhakar, S. [2004] „Protecting Rights Proofs for Relational Data Using Watermarking”, **TKDE**, 16:12, 2004, s. 1509 – 1525.
- Sklar, D. [2005] **Learning PHP5**, O'Reilly Media, Inc., 2005.
- Smith G. [1990] „The Semantic Data Model for Security: Representing the Security Semantics of an Application”, w: ICDE [1990].
- Smith, J. i in. [1981] „MULTIBASE: Integrating Distributed Heterogeneous Database Systems”, *NCC, AFIPS*, 50, 1981.

- Smith, J. R., Chang, S.-F. [1996] „VisualSEEK: A Fully Automated Content-Based Image Query System”, *Proc. 4th ACM Multimedia Conf.*, Boston, MA, listopad 1996, s. 87 – 98.
- Smith J., Chang P. [1975] „Optimizing the Performance of a Relational Algebra Interface”, **CACM**, 18:10, październik 1975.
- Smith J., Smith D. [1977] „Database Abstractions: Aggregation and Generalization”, **TODS**, 2:2, czerwiec 1977.
- Smith K., Winslett M. [1992] „Entity Modeling in the MLS Relational Model”, w: *VLDB* [1992].
- Smith P., Barnes G. [1987] **Files and Databases: An Introduction**, Addison-Wesley, 1987.
- Snodgrass R. [1987] „The Temporal Query Language TQuel”, **TODS**, 12:2, czerwiec 1987.
- Snodgrass R., Ahn I. [1985] „A Taxonomy of Time in Databases”, w: *SIGMOD* [1985].
- Snodgrass R., red. [1995] **The TSQ2 Temporal Query Language**, Kluwer, 1995.
- Soutou G. [1998] „Analysis of Constraints for N-ary Relationships”, w: *ER98*.
- Spaccapietra S., Jain R., red. [1995] *Proceedings of the Visual Database Workshop*, Lozanna, Szwajcaria, październik 1995.
- Spiliopoulou, M. [2000] „Web Usage Mining for Web Site Evaluation”, **CACM** 43:8, sierpień 2000, s. 127 – 134.
- Spooner D., Michael A., Donald B. [1986] „Modeling CAD Data with Data Abstraction and Object Oriented Technique”, w: *ICDE* [1986].
- Srikant R., Agrawal R. [1995] „Mining Generalized Association Rules”, w: *VLDB* [1995].
- Srinivas M., Patnaik L. [1994] „Genetic Algorithms: A Survey”, **IEEE Computer**, czerwiec 1994.
- Srinivasan V., Carey M. [1991] „Performance of B-Tree Concurrency Control Algorithms”, w: *SIGMOD* [1991].
- Srivastava D., Ramakrishnan R., Sudarshan S., Sheshadri P. [1993] „Coral++: Adding Object-Orientation to a Logic Database Language”, w: *VLDB* [1993].
- Srivastava J., i in. [2000] „Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data”, **SIGKDD Explorations**, 1:2, 2000.
- Stachour P., Thuraisingham B. [1990] „The Design and Implementation of INGRES”, **TKDE**, 2:2, czerwiec 1990.
- Stallings W. [1997] **Data and Computer Communications**, wydanie piąte, Prentice-Hall, 1997.
- Stallings W. [2010] **Network Security Essentials: Applications and Standards**, wydanie czwarte, Prentice Hall, 2010.
- Stevens, P., Pooley, R. [2003] **Using UML: Software Engineering with Objects and Components**, wydanie poprawione, Addison-Wesley, 2003.
- Stoesser, G. i in. [2003] „The EMBL Nucleotide Sequence Database: Major New Developments”, **Nucleic Acids Research**, 31:1, styczeń 2003, s. 17 – 22.
- Stoica, I., Morris, R., Karger, D. i in. [2001] „Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications”, *SIGCOMM* 2001.
- Stonebraker, M., Aoki, P., Litwin W., i in. [1996] „Mariposa: A Wide-Area Distributed Database System”, **VLDB J**, 5:1, 1996, s. 48 – 63.

- Stonebraker M. i in. [2005] „C-store: A column oriented DBMS”, w: *VLDB* [2005].
- Stonebraker M. [1975] „Implementation of Integrity Constraints and Views by Query Modification”, w: *SIGMOD* [1975].
- Stonebraker M. [1993] „The Miro DBMS”, w: *SIGMOD* [1993].
- Stonebraker M., Rowe L. [1986] „The Design of POSTGRES”, w: *SIGMOD* [1986].
- Stonebraker M., red. [1994] **Readings in Database Systems**, wydanie drugie, Morgan Kaufmann, 1994.
- Stonebraker M., Hanson E., Hong C. [1987] „The Design of the POSTGRES Rules System”, w: *ICDE* [1987].
- Stonebraker M., Moore D. [1996] **Object-Relational DBMSs: The Next Great Wave**, Morgan Kaufman, 1996.
- Stonebraker M., Wong E., Kreps P., Held G. [1976] „The Design and Implementation of INGRES”, **TODS**, 1:3, wrzesień 1976.
- Stroustrup, B. [1997] **The C++ Programming Language: Special Edition**, Pearson, 1997.
- Su S. [1985] „A Semantic Association Model for Corporate and Scientific-Statistical Databases”, **Information Science**, 29, 1985.
- Su S. [1988] **Database Computers**, McGraw-Hill, 1988.
- Su S., Krishnamurthy V., Lam H. [1988] „An Object-Oriented Semantic Association Model (OSAM\*)”, w: **AI in Industrial Engineering and Manufacturing: Theoretical Issues and Applications**, American Institute of Industrial Engineers, 1988.
- Subramanian V. S., Jajodia S., red. [1996] **Multimedia Database Systems: Issues and Research Directions**, Springer Verlag, 1996.
- Subrahmanian V. [1998] **Principles of Multimedia Databases Systems**, Morgan Kaufmann, 1998.
- Sunderraman R. [2007] **ORACLE 10g Programming: A Primer**, Addison-Wesley, 2007.
- Swami A., Gupta A. [1989] „Optimization of Large Join Queries: Combining Heuristics and Combinatorial Techniques”, w: *SIGMOD* [1989].
- Sybase [2005] **System Administration Guide: Volume 1 and Volume 2 (Adaptive Server Enterprise 15.0)**, Sybase, 2005.
- Tan, P., Steinbach, M., Kumar, V. [2006] **Introduction to Data Mining**, Addison-Wesley, 2006.
- Tanenbaum A. [2003] **Computer Networks**, wydanie czwarte, Prentice Hall PTR, 2003.
- Tansel A. i in, red. [1993] **Temporal Databases: Theory, Design, and Implementation**, Benjamin Cummings, 1993.
- Teorey T. [1994] **Database Modeling and Design: The Fundamental Principles**, wydanie drugie, Morgan Kaufmann, 1994.
- Teorey T., Yang D., Fry J. [1986] „A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model”, **ACM Computing Surveys**, 18:2, czerwiec 1986.
- Thomas J., Gould J. [1975] „A Psychological Study of Query by Example”, *NCC AFIPS*, 44, 1975.
- Thomas R. [1979] „A Majority Consensus Approach to Concurrency Control for Multiple Copy Data Bases”, **TODS**, 4:2, czerwiec 1979.

- Thomasian A. [1991] „Performance Limits of Two-Phase Locking”, w: *ICDE* [1991].
- Thuraisingham, B. [2001] **Managing and Mining Multimedia Databases**, CRC Press, 2001.
- Thuraisingham B. i in. [2001] „Directions for Web and E-Commerce Applications Security”, *Tenth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2001, s. 200 – 204.
- Thusoo, A. i in. [2010] Hive—A Petabyte Scale Data Warehouse Using Hadoop, w: *ICDE* [2010].
- Todd S. [1976] „The Peterlee Relational Test Vehicle — A System Overview”, *IBM Systems Journal*, 15:4, grudzień 1976.
- Toivonen H., „Sampling Large Databases for Association Rules”, w: *VLDB* [1996].
- Tou J., red. [1984] **Information Systems COINS-IV**, Plenum Press, 1984.
- Tsangaris M., Naughton J. [1992] „On the Performance of Object Clustering Techniques”, w: *SIGMOD* [1992].
- Tsichritzis D. [1982] „Forms Management”, *CACM*, 25:7, lipiec 1982.
- Tsichritzis D., Klug A., red. [1978] **The ANSI/X3/SPARC DBMS Framework**, AFIPS Press, 1978.
- Tsichritzis D., Lochovsky F. [1976] „Hierarchical Data-Base Management: A Survey”, *ACM Computing Surveys*, 8:1, marzec 1976.
- Tsichritzis D., Lochovsky F. [1982] **Data Models**, Prentice-Hall, 1982.
- Tsotras V., Gopinath B. [1992] „Optimal Versioning of Object Classes”, w: *ICDE* [1992].
- Tsou D. M., Fischer P. C. [1982] „Decomposition of a Relation Scheme into Boyce Codd Normal Form”, *SIGACT News*, 14:3, 1982, s. 23 – 29.
- U.S. Congress [1988] „Office of Technology Report, Appendix D: Databases, Repositories, and Informatics”, w: **Mapping Our Genes: Genome Projects: How Big, How Fast?**, John Hopkins University Press, 1988.
- U.S. Department of Commerce [1993] **TIGER/Line Files**, Bureau of Census, Waszyngton, 1993.
- Ullman J. [1982] **Principles of Database Systems**, wydanie drugie, Computer Science Press, 1982.
- Ullman J. [1985] „Implementation of Logical Query Languages for Databases”, *TODS*, 10:3, wrzesień 1985.
- Ullman J. [1988] **Principles of Database and Knowledge-Base Systems**, tom 1, Computer Science Press, 1988.
- Ullman J. [1989] **Principles of Database and Knowledge-Base Systems**, tom 2, Computer Science Press, 1989.
- Ullman J. D., Widom J. [1997] **A First Course in Database Systems**, Prentice-Hall, 1997.
- Uschold, M., Gruninger, M. [1996] „Ontologies: Principles, Methods and Applications”, **Knowledge Engineering Review**, 11:2, czerwiec 1996.
- Vadivelu, V., Jayakumar, R.V., Muthuvel, M., i in. [2008] „A backup mechanism with concurrency control for multilevel secure distributed database systems”, *Proc. Int. Conf. on Digital Information Management*, 2008, s. 57 – 62.
- Vaidya, J., Clifton, C., „Privacy-Preserving Data Mining: Why, How, and What For?”, **IEEE Security & Privacy (IEEE SP)**, listopad/grudzień 2004, s. 19 – 27.

- Valduriez P., Gardarin G. [1989] **Analysis and Comparison of Relational Database Systems**, Addison-Wesley, 1989.
- van Rijsbergen, C. J. [1979] **Information Retrieval**, Butterworths, 1979.
- Valiant, L. [1990] „A Bridging Model for Parallel Computation”, **CACM**, 33:8, sierpień 1990.
- Vassiliou Y. [1980] „Functional Dependencies and Incomplete Information”, w: *VLDB* [1980].
- Vélez, F., Bernard, G., Darnis, V. [1989] „The O2 Object Manager: an Overview”, w: *VLDB* [1989], s. 357 – 366.
- Verheijen G., VanBekkum J. [1982] „NIAM: An Information Analysis Method”, w: Olle i in. [1982].
- Verhofstadt J. [1978] „Recovery Techniques for Database Systems”, **ACM Computing Surveys**, 10:2, czerwiec 1978.
- Vielle L. [1986] „Recursive Axioms in Deductive Databases: The Query-Subquery Approach”, w: *EDS* [1986].
- Vielle L. [1987] „Database Complete Proof Production Based on SLD-Resolution”, w: *Proceedings of the Fourth International Conference on Logic Programming*, 1987.
- Vielle L. [1988] „From QSQ Towards QoSQ: Global Optimization of Recursive Queries”, w: *EDS* [1988].
- Vielle L. [1998] „VALIDITY: Knowledge Independence for Electronic Mediation”, artykuł na zamówienie, w: *Practical Applications of Prolog/Practical Applications of Constraint Technology (PAP/PACT'98)*, Londyn, marzec 1998.
- Vin H., Zellweger P., Swinehart D., Venkat Rangan P. [1991] „Multimedia Conferencing in the Etherphone Environment”, **IEEE Computer**, Special Issue on Multimedia Information Systems, 24:10, październik 1991.
- VLDB [1975] *Proceedings of the First International Conference on Very Large Data Bases*, Kerr D., red., Framingham, MA, wrzesień 1975.
- VLDB [1976] **Systems for Large Databases**, Lockemann P., Neuhold E., red., w: *Proceedings of the Second International Conference on Very Large Data Bases*, Bruksela, Belgia, lipiec 1976, North-Holland, 1976.
- VLDB [1977] *Proceedings of the Third International Conference on Very Large Data Bases*, Merten A., red., Tokio, Japonia, październik 1977.
- VLDB [1978] *Proceedings of the Fourth International Conference on Very Large Data Bases*, Bubenko J., Yao S., red., Berlin Zachodni, Niemcy, wrzesień 1978.
- VLDB [1979] *Proceedings of the Fifth International Conference on Very Large Data Bases*, Furtado A., Morgan H., red., Rio de Janeiro, Brazylia, październik 1979.
- VLDB [1980] *Proceedings of the Sixth International Conference on Very Large Data Bases*, Lochovsky F., Taylor R., red., Montreal, Kanada, październik 1980.
- VLDB [1981] *Proceedings of the Seventh International Conference on Very Large Data Bases*, Zaniolo C., Delobel C., red., Cannes, Francja, wrzesień 1981.
- VLDB [1982] *Proceedings of the Eighth International Conference on Very Large Data Bases*, McLeod D., Villasenor Y., red., Mexico City, wrzesień 1982.
- VLDB [1983] *Proceedings of the Ninth International Conference on Very Large Data Bases*, Schkolnick M., Thanos C., red., Florencja, Włochy, październik/listopad 1983.



VLDB [1984] *Proceedings of the Tenth International Conference on Very Large Data Bases*, Dayal U., Schlageter G., Seng L, red., Singapur, sierpień 1984.

VLDB [1985] *Proceedings of the Eleventh International Conference on Very Large Data Bases*, Pirotte A., Vassiliou Y., red., Sztokholm, Szwecja, sierpień 1985.

VLDB [1986] *Proceedings of the Twelfth International Conference on Very Large Data Bases*, Chu W., Gardarin G., Ohsuga S., red., Kioto, Japonia, sierpień 1986.

VLDB [1987] *Proceedings of the Thirteenth International Conference on Very Large Data Bases*, Stocker P., Kent W., Hammersley P., red., Brighton, Anglia, wrzesień 1987.

VLDB [1988] *Proceedings of the Fourteenth International Conference on Very Large Data Bases*, Bancilhon E., DeWitt D., red., Los Angeles, sierpień/wrzesień 1988.

VLDB [1989] *Proceedings of the Fifteenth International Conference on Very Large Data Bases*, Apers P., Wiederhold G., red., Amsterdam, sierpień 1989.

VLDB [1990] *Proceedings of the Sixteenth International Conference on Very Large Data Bases*, McLeod D., Sacks-Davis R., Schek H., red., Brisbane, Australia, sierpień 1990.

VLDB [1991] *Proceedings of the Seventeenth International Conference on Very Large Data Bases*, Lohman G., Sernadas A., Camps R., red., Barcelona, Katalonia, Hiszpania, wrzesień 1991.

VLDB [1992] *Proceedings of the Eighteenth International Conference on Very Large Data Bases*, Yuan L, red., Vancouver, Kanada, sierpień 1992.

VLDB [1993] *Proceedings of the Nineteenth International Conference on Very Large Data Bases*, Agrawal R., Baker S., Bell D.A., red., Dublin, Irlandia, sierpień 1993.

VLDB [1994] *Proceedings of the 20th International Conference on Very Large Data Bases*, Bocca J., Jarke M., Zaniolo C., red., Santiago, Chile, wrzesień 1994.

VLDB [1995] *Proceedings of the 21st International Conference on Very Large Data Bases*, Dayal U., Gray P. M. D., Nishio S., red., Zurich, Szwajcaria, wrzesień 1995.

VLDB [1996] *Proceedings of the 22nd International Conference on Very Large Data Bases*, Vijayaraman T. M., Buchman A. P., Mohan C, Sarda N. L., red., Bombaj, Indie, wrzesień 1996.

VLDB [1997] *Proceedings of the 23rd International Conference on Very Large Data Bases*, Jarke M., Carey M. J., Dittrich K. R., Lochovsky F. H., Loucopoulos, P, red., Zurich, Szwajcaria, wrzesień 1997.

VLDB [1998] *Proceedings of the 24th International Conference on Very Large Data Bases*, Gupta A., Shmueli O., Widom J., red., Nowy Jork, wrzesień 1998.

VLDB [1999] *Proceedings of the 25th International Conference on Very Large Data Bases*, Zdonik S. B., Valduriez P., Orlowska M., red., Edynburg, Szkocja, wrzesień 1999.

VLDB [2000] *Proc. 26th International Conference on Very Large Data Bases*, Abbadi, A. i in., red., Kair, Egipt, wrzesień 2000.

VLDB [2001] *Proc. 27th International Conference on Very Large Data Bases*, Apers, P. i in., red., Rzym, Włochy, wrzesień 2001.

VLDB [2002] *Proc. 28th International Conference on Very Large Data Bases*, Bernstein, P., Ionnidis, Y., Ramakrishnan, R., red., Hong Kong, Chiny, sierpień 2002.

VLDB [2003] *Proc. 29th International Conference on Very Large Data Bases*, Freytag, J. i in., red., Berlin, Niemcy, wrzesień 2003.



- VLDB [2004] *Proc. 30th International Conference on Very Large Data Bases*, Nascimento, M. i in., red., Toronto, Kanada, wrzesień 2004.
- VLDB [2005] *Proc. 31st International Conference on Very Large Data Bases*, Böhm, K. i in., red., Trondheim, Norwegia, sierpień/wrzesień 2005.
- VLDB [2006] *Proc. 32nd International Conference on Very Large Data Bases*, Dayal, U. i in., red., Seul, Korea, wrzesień 2006.
- VLDB [2007] *Proc. 33rd International Conference on Very Large Data Bases*, Koch, C. i in., red., Wiedeń, Austria, wrzesień, 2007.
- VLDB [2008] *Proc. 34th International Conference on Very Large Data Bases*, wydane jako: **Proceedings of the VLDB Endowment**, tom 1, Auckland, Nowa Zelandia, sierpień 2008.
- VLDB [2009] *Proc. 35th International Conference on Very Large Data Bases*, wydane jako: **Proceedings of the VLDB Endowment**, tom 2, Lyon, Francja, sierpień 2009.
- VLDB [2010] *Proc. 36th International Conference on Very Large Data Bases*, wydane jako: **Proceedings of the VLDB Endowment**, tom 3, Singapur, sierpień 2010.
- VLDB [2011] *Proc. 37th International Conference on Very Large Data Bases*, wydane jako: **Proceedings of the VLDB Endowment**, tom 4, Seattle, sierpień 2011.
- VLDB [2012] *Proc. 38th International Conference on Very Large Data Bases*, wydane jako: **Proceedings of the VLDB Endowment**, tom 5, Sztambuł, Turcja, sierpień 2012.
- VLDB [2013] *Proc. 39th International Conference on Very Large Data Bases*, wydane jako: **Proceedings of the VLDB Endowment**, tom 6, Riva del Garda, Trento, Włochy, sierpień 2013.
- VLDB [2014] *Proc. 39th International Conference on Very Large Data Bases*, wydane jako: **Proceedings of the VLDB Endowment**, tom 7, Hangzhou, Chiny, wrzesień 2014.
- VLDB [2015] *Proc. 40th International Conference on Very Large Data Bases*, wydane jako: **Proceedings of the VLDB Endowment**, tom 8, Kohala Coast, Hawaje, wrzesień 2015, w przygotowaniu.
- Voorhees, E., Harman, D., red., [2005] **TREC Experiment and Evaluation in Information Retrieval**, MIT Press, 2005.
- Vorhaus A., Mills R. [1967] „The Time-Shared Data Management System: A New Approach to Data Management”, System Development Corporation, Report SP-2634, 1967.
- Wallace D. [1995] „1994 William Allan Award Address: Mitochondrial DNA Variation in Human Evolution, Degenerative Disease, and Aging”, **American Journal of Human Genetics**, 57:201 – 223, 1995.
- Walton C., Dale A., Jenevein R. [1991] „A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins”, w: *VLDB* [1991].
- Wang K. [1990] „Polynomial Time Designs Toward Both BCNF and Efficient Data Manipulation”, w: *SIGMOD* [1990].
- Wang Y., Madnick S. [1989] „The Inter-Database Instance Identity Problem in Integrating Autonomous Systems”, w: *ICDE* [1989].
- Wang Y., Rowe L. [1991] „Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture”, w: *SIGMOD* [1991].
- Warren D. [1992] „Memoing for Logic Programs”, **CACM**, 35:3, **ACM**, marzec 1992.

- Weddell G. [1992] „Reasoning About Functional Dependencies Generalized for Semantic Data Models”, **TODS**, 17:1, marzec 1992.
- Weikum G. [1991] „Principles and Realization Strategies of Multilevel Transaction Management”, **TODS**, 16:1, marzec 1991.
- Weiss S., Indurkha N. [1998] **Predictive Data Mining: A Practical Guide**, Morgan Kaufmann, 1998.
- Whang K. [1985] „Query Optimization in Office By Example”, IBM Research Report RC 11571, grudzień 1985.
- Whang K., Navathe S. [1987] „An Extended Disjunctive Normal Form Approach for Processing Recursive Logic Queries in Loosely Coupled Environments”, w: *VLDB* [1987].
- Whang K., Navathe S. [1992] „Integrating Expert Systems with Database Management Systems — an Extended Disjunctive Normal Form Approach”, **Information Sciences**, 64, marzec 1992.
- Whang K., Malhotra A., Sockut G., Burns L. [1990] „Supporting Universal Quantification in a Two-Dimensional Database Query Language”, w: *ICDE* [1990].
- Whang K., Wiederhold G., Sagalowicz D. [1982] „Physical Design of Network Model Databases Using the Property of Separability”, w: *VLDB* [1982].
- White, T. [2012] **Hadoop: The Definitive Guide**, (wydanie trzecie), O'Reilly, Yahoo! Press, 2012, [<http://hadoopbook.com>].
- Widom J., „Research Problems in Data Warehousing”, **CIKM**, listopad 1995.
- Widom J., Ceri S. [1996] **Active Database Systems**, Morgan Kaufmann, 1996.
- Widom J., Finkelstein S. [1990] „Set Oriented Production Rules in Relational Database Systems”, w: *SIGMOD* [1990].
- Wiederhold G. [1984] „Knowledge and Database Management”, **IEEE Software**, styczeń 1984.
- Wiederhold, G. [1987] **File Organization for Database Design**, McGraw-Hill, 1987.
- Wiederhold G. [1995] „Digital Libraries, Value, and Productivity”, **CACM**, kwiecień 1995.
- Wiederhold G., Elmasri R. [1979] „The Structural Model for Database Design”, w: ER Conference [1979].
- Wiederhold G., Beetem A., Short G. [1982] „A Database Approach to Communication in VLSI Design”, **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, 1:2, kwiecień 1982.
- Wilkinson K., Lyngbaek P., Hasan W. [1990] „The IRIS Architecture and Implementation”, **TKDE**, 2:1, marzec 1990.
- Willshire M. [1991] „How Spacey Can They Get? Space Overhead for Storage and Indexing with Object-Oriented Databases”, w: *ICDE* [1991].
- Wilson B., Navathe S. [1986] „An Analytical Framework for Limited Redesign of Distributed Databases”, *Proceedings of the Sixth Advanced Database Symposium*, Tokio, sierpień 1986.
- Wiorkowski G., Kull D. [1992] **DB2-Design and Development Guide**, wyd. 3, Addison-Wesley, 1992.
- Witkowski, A., i in., „Spreadsheets in RDBMS for OLAP”, w: *SIGMOD* [2003].
- Wirth, N. [1985] **Algorithms and Data Structures**, Prentice-Hall, 1985.

- Witten, I. H., Bell, T. C., Moffat, A. [1994] **Managing Gigabytes: Compressing and Indexing Documents and Images**, Wiley, 1994.
- Wolfson, O. Chamberlain, S., Kalpakis, K., Yesha, Y. [2001] „Modeling Moving Objects for Location Based Services”, NSF Workshop on Infrastructure for Mobile and Wireless Systems, w: **LNCS 2538**, s. 46 – 58.
- Wong E. [1983] „Dynamic Rematerialization-Processing Distributed Queries Using Redundant Data”, **TSE**, 9:3, maj 1983.
- Wong E., Youssefi K. [1976] „Decomposition — A Strategy for Query Processing”, **TODS**, 1:3, wrzesień 1976.
- Wong H. [1984] „Micro and Macro Statistical/Scientific Database Management”, w: ICDE [1984].
- Wood J., Silver D. [1989] **J Joint Application Design: How to Design Quality Systems in 40% Less Time**, Wiley, 1989.
- Worboys, M., Duckham, M. [2004] **GIS – A Computing Perspective**, wydanie drugie, CRC Press, 2004.
- Wright, A., Carothers, A., Campbell, H. [2002] „Gene-environment interactions the BioBank UK study”, **Pharmacogenomics Journal**, 2002, s. 75 – 82.
- Wu X., Ichikawa T. [1992] „KDA: A Knowledge-Based Database Assistant with a Query Guiding Facility”, **TKDE** 4:5, październik 1992.
- <http://www.oracle.com/ocom/groups/public/@ocompublic/documents/webcontent/039544.pdf>.
- Xie, I. [2008] **Interactive Information Retrieval in Digital Environments**, IGI Publishing, Hershey, PA, 2008.
- Xie, W. [2005] „Supporting Distributed Transaction Processing Over Mobile and Heterogeneous Platforms”, praca doktorska, Georgia Tech, 2005.
- Xie, W., Navathe, S., Prasad, S. [2003] „Supporting QoS-Aware Transaction in the Middleware for a System of Mobile Devices (SyD)”, w: *Proc. 1st Int. Workshop on Mobile Distributed Computing in ICDCS '03*, Providence, RI, maj 2003.
- XML (2005): <http://www.w3.org/XML/>.
- Yan, W.P., Larson, P.A. [1995] „Eager aggregation and Lazy Aggregation”, w: **VLDB** [1995].
- Yannakakis Y. [1984] „Serializability by Locking”, **JACM**, 31:2, 1984.
- Yao S. [1979] „Optimization of Query Evaluation Algorithms”, **TODS**, 4:2, czerwiec 1979.
- Yao S., red. [1985] **Principles of Database Design**, tom 1: **Logical Organizations**, Prentice-Hall, 1985.
- Yee, K.-P. i in. [2003] „Faceted metadata for image search and browsing”, *Proc.ACM CHI 2003 (Conference on Human Factors in Computing Systems)*, Ft. Lauderdale, FL, s. 401 – 408.
- Yee, W. i in. [2002] „Efficient Data Allocation over Multiple Channels at Broadcast Servers”, *IEEE Transactions on Computers*, numer specjalny poświęcony mobilności i bazom danych, 51:10, 2002.
- Yee, W., Donahoo, M., Navathe, S. [2001] „Scaling Replica Maintenance in Intermittently Synchronized Databases”, w: **CIKM**, 2001.
- Yoshitaka, A., Ichikawa, K. [1999] „A Survey on Content-Based Retrieval for Multimedia Databases”, **TKDE**, 11:1, styczeń 1999.

Youssefi K., Wong E. [1979] „Query Processing in a Relational Database Management System”, w: VLDB [1979].

Zadeh L. [1983] „The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems”, **Fuzzy Sets and Systems**, 11, North-Holland, 1983.

Zaharia M. i in. [2012] „Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing”, w: *Proc. Usenix Symp. on Networked System Design and Implementation (NSDI)*, kwiecień 2012, s. 15 – 28.

Zaniolo C. [1976] „Analysis and Design of Relational Schemata for Database Systems”, praca doktorska, University of California, Los Angeles, 1976.

Zaniolo C. [1988] „Design and Implementation of a Logic Based Language for Data Intensive Applications”, ICLP/SLP 1988, s. 1666 – 1687.

Zaniolo, C. [1990] „Deductive Databases: Theory meets Practice”, w: EDBT, 1990, s. 1 – 15.

Zaniolo C. i in. [1986] „Object-Oriented Database Systems and Knowledge Systems”, w: *EDS* [1984].

Zaniolo C. i in. [1997] **Advanced Database Systems**, Morgan Kaufmann, 1997.

Zantinge, D., Adriaans, P. [1996] *Managing Client Server*, Addison-Wesley, 1996.

Zave P. [1997] „Classification of Research Efforts in Requirements Engineering”, **ACM Computing Surveys**, 29:4, grudzień 1997.

Zeiler, M. [1999] **Modeling Our World—The ESRI Guide to Geodatabase Design**, 1999.

Zhang, T., Ramakrishnan R., Livny M. [1996] „Birch: An Efficient Data Clustering Method for Very Large Databases”, w: *SIGMOD*, 1996.

Zhao, R., Grosky, W. [2002] „Bridging the Semantic Gap in Image Retrieval”, w: **Distributed Multimedia Databases: Techniques and Applications** (Shih, T.K., red.), Idea Publishing, 2002.

Zhou, X., Pu, P. [2002] „Visual and Multimedia Information Management”, *Proc. Sixth Working Conf. on Visual Database Systems*, Zhou, X., Pu, P. (red.), Brisbane, Australia, IFIP Conference Proceedings 216, Kluwer, 2002.

Ziauddin, M. i in. [2008] „Optimizer Plan Change Management: Improved Stability and Performance in Oracle 11g”, w: *VLDB* [2008].

Zicari R. [1991] „A Framework for Schema Updates in an Object-Oriented Database System”, w: *ICDE* [1991].

Zloof M. [1975] „Query by Example”, *NCC, AFIPS*, 44, 1975.

Zloof M. [1982] „Office By Example: A Business Language That Unifies Data, Word Processing, and Electronic Mail”, **IBM Systems Journal**, 21:3, 1982.

Zobel J., Moffat A., Sacks-Davis R. [1992] „An Efficient Indexing Technique for Full-Text Database Systems”, w: *VLDB* [1992].

Zvieli A. [1986] „A Fuzzy Relational Calculus”, w: *EDS* [1986].

---

# Skorowidz

## A

abstrakcja danych, 46, 70, 173  
adekwatność  
  dla użytkownika, 1116  
  tematyczna, 1115  
administrator bazy danych,  
  50, 1203  
adnotacje, 497  
adres sprzętowy, 613  
adresowanie jawne, 638  
AES, Advanced Encryption  
  Standard, 1228  
agenty internetowe, 1125  
agregacja, 176, 314, 753,  
  1256  
akceptowalność dostępu,  
  1205  
akcja, 279  
aksjomaty Armstronga, 566  
aktor na scenie, 50  
aktualizacje, 524  
  bezpośrednie, 884  
  natychmiastowe, 883,  
  893  
  odroczone, 882, 890  
aktywne  
  bazy danych, 278  
  implementacja, 1039  
  model uogólniony, 1035  
  projektowanie, 1039  
  wyzwalacze, 1035  
  zastosowania, 1044  
reguły, 1042  
systemy baz danych, 59

algebra  
  Allena, 1055  
  relacyjna, 293  
algorytm  
  Apriori, 1148  
  ARIES, 897, 904  
  heurystycznej  
  optymalizacji  
  algebraicznej, 765  
  nieoczekiwania, 858  
  NO-UNDO/REDO, 883  
  odwzorowujący model ER,  
  348  
  partycjonujący, 1155  
  próbujący, 1150  
  sortująco-scalający, 728  
  syntezy relacyjnej, 595  
  UNDO/NO-REDO, 883  
  UNDO/REDO, 883  
  uporządkowania, 860  
  złączenia mieszającego, 742  
algorytmy  
  genetyczne, 1169  
  normalizacyjne, 586  
  operacji projekcji, 743  
  operacji selekcji, 729  
  projektowania, 563, 579  
  równoległe, 751  
  równoległego  
  przetwarzania zapytań,  
  750  
sortowania  
  zewnętrznego, 727  
z kluczem  
  symetrycznym, 1228

alias, 239, 240  
alokacja  
  bloków, 627  
  danych, 918, 921  
analitycy systemowi, 52  
analityka internetowa, 1128  
analiza  
  częstotliwości operacji  
  aktualizacji, 710  
  danych  
  audio, 1069  
  internetowych, 1119  
  przestrzennych, 1059  
encji, 985  
mediów  
  społecznościowych, 984  
obrazów, 1067  
opisowe i prognostyczne,  
  984  
spodziewanej  
  częstotliwości, 709  
struktury  
  odsyłaczy, 1122  
  sieci WWW, 1121  
transakcji, 708  
treści w sieci WWW, 1123,  
  1125  
użytkowania witryn, 1126  
wzorców  
  czasowych, 709  
  unikatowości, 710  
wzorców, 1128  
zakończona sugestiami,  
  984  
zapytań, 708

- analizator składniowy, 721
- anomalie
  - aktualizacji, 521, 524
  - modyfikowania, 524
  - usuwania, 524
  - wstawiania, 524
- antyłączenia, 725, 744
- anulowanie kaskadowe, 828
- Apache
  - Giraph, 1016
  - HBase, 972
  - Hive, 1006
  - Pig, 1004
  - Tez, 1015
  - ZooKeeper, 975
- API, Application Programming Interface, 372, 394
- aplikacje, 40
  - baz danych, 101, 105, 371, 608
  - dla urządzeń mobilnych, 80, 101
  - obiektowe, 62
- architektura
  - bez zasobów
  - współdzielonych, 751
  - czystych baz
    - rozproszonych, 942
  - federacyjnych baz danych, 942
  - klient-serwer, 944
  - platformy YARN, 1012
  - RAID, 648
  - rozproszonych baz danych, 940
  - scentralizowana, 87
  - składowania danych, 653
  - systemów baz danych, 69, 941
    - HDFS, 994
    - Hive, 1006
  - systemów zarządzania, 87
  - trójwarstwowa, 74
    - poziom koncepcyjny, 75
    - poziom wewnętrzny, 74
    - poziom zewnętrzny, 75
  - typu klient-serwer, 87, 89
  - dwuwarstwowa, 90
  - n-warstwowa, 90
  - trójwarstwowa, 90
- z dyskami
  - współdzielonymi, 750
  - z pamięcią współdzieloną, 750
- zabezpieczeń opartych na etykietach, 1234
- asercja, 211, 277
- asocjacja, 176
  - przestrzenna, 1064
  - negatywna, 1158
- AST, Automated Storage Tiering, 656
- atak przez wstrzyknięcie kodu, 1221
- atom, 326, 334, 430
- atrybuty, 71, 107, 198, 457, 487
  - czasu życia, 1055
  - definiujące, 158
  - jednowartościowe, 108
  - klucza, 111, 534
  - klucza typu encji, 111
  - kompozytowe, 499
  - pochodne, 108
  - proste, 107
  - schematu relacji, 534
  - skomplikowane, 109
  - specyficzne, 155
  - stałe, 108
  - typów związków, 121
  - wielowartościowe, 108
  - złączenia, 308, 735
  - złożone, 107
- audio, 1067
- audyty bazy danych, 1204
- automatyczna
  - analiza danych, 1065, 1069
  - analiza obrazów, 1067
  - zmienna globalna, 407
- awarie, 818
  - łączy komunikacyjnych, 925
  - poszczególnych węzłów, 925
- B**
- B\*-drzewa, 695
- B+-drzewa, 687
  - odmiany, 694
  - usuwanie, 690, 695
  - wstawianie, 690, 693
- wystąpienie niedomiaru, 694
- wyszukiwanie, 690
- bajt, 609
- baza danych, 37
  - MongoDB, 961
  - NOSQL, 64, 955
  - UNIWERSYTET, 137, 166, 167
- bazy danych
  - administratorzy, 50
  - aktywne, 1035
  - analitycy systemowi, 52
  - architektura systemów, 69
  - bezpieczeństwo, 1197, 1199
  - czasowe, 1045
  - dedukcyjne, 1070
  - funkcje mieszające, 601
  - indeksowanie, 601
  - internetowe, 405
  - inżynierowie
    - oprogramowania, 52
  - koncepcyjne modelowanie danych, 99
  - konta użytkowników, 1204
  - modele danych
    - przestrzennych, 1059
  - multimedialne, 1065
  - normalizacja, 515
  - obiektowe, 425, 465, 1053
  - obiektowo-relacyjne, 425
  - ODMG, 460
  - odporność, 1233
  - odtworzenie, 809
  - pamięci głównej, 606
  - postaci normalne, 532
  - projektanci, 50
  - projekty, 601
  - przechowywanie, 607
  - przestrzenne, 1058
  - relacyjny model danych, 193
  - rozproszone, 911
  - sterowanie współbieżne, 809
  - struktury plikowe, 601
  - systemy zarządzania, 53
  - techniki odtwarzania, 881
  - techniki programowania, 367

- teoria projektowania, 515
  - transakcji, 809
  - użytkownicy, 51
  - używanie systemów zarządzania, 64
  - właściwości, 44
  - zapisywanie dokumentów XML, 499
  - zapytania, 719
  - zastosowania, 1031
  - BCNF, Boyce-Codd normal form, 545
  - dekompozycja relacji, 547
  - dekompozycja ze złączeniem nieaddytywnym, 582
  - B-drzewa, 648, 681–684
  - bezpieczeństwo, 1206
    - audyty, 1204
    - certyfikaty, 1230
    - cofanie uprawnień, 1209
    - dostęp do danych XML, 1218
    - infrastruktura klucza publicznego, 1227
    - klasy bezpieczeństwa, 1212
    - konta użytkowników, 1204
    - kontrola dostępu, 1207, 1212, 1215
      - na poziomie wierszy, 1217
    - kontrola przepływu, 1225
    - model
      - dyspozycyjny, 1215
      - obowiązkowy, 1215
    - ochrona dostępu, 1204
    - ograniczenia propagacji uprawnień, 1211
    - określanie uprawnień, 1208
    - podpis cyfrowy, 1230
    - polityki kontroli dostępu, 1219
    - rodzaje zabezpieczeń, 1200
    - statystycznych baz danych, 1224
    - szyfrowanie, 1227
    - środki kontroli, 1201
    - typy uprawnień, 1207
    - w bazach danych, 1199
    - w sieci WWW, 1129
    - wstrzykiwanie kodu, 1220
    - bezpieczne wyrażenia, 333
    - biblioteki
      - funkcji, 372, 390
      - klas, 388, 402
      - klas JDBC, 394
    - bieżący stan relacji, 199
    - big data, 64, 955, 983, 986
    - chmury obliczeniowe, 1019
    - interfejsy, 1024
    - model
      - danych oparty na dokumentach, 94
      - danych oparty na grafie, 94
      - danych typu klucz-wartość, 94
      - XML-owy, 94
    - niejednorodność informacji, 1023
    - niekompletne informacje, 1024
    - prywatność i poufność, 1024
    - szybkość generowania, 987
    - wiarygodność, 988
    - bit, 609
      - aktualizacji, 621
      - modyfikacji, 884
      - zajętości, 884
    - bitmapa
      - istnienia, 701
      - dla liści B+-drzew, 702
    - blok zapytania, 724
    - blokady, 848
      - binarne, 848
      - certyfikujące, 864
      - dzielone, 850
      - konwersje, 852
      - menedżer blokad, 849
      - na krótki okres czasu, 876
      - sterowanie współbieżne, 872
      - systemowe tabele, 848
      - wielotrybowe, 850
      - wyłączne, 850
    - bloki, 626
      - alokowanie, 627
    - blokowanie, 847
      - dwufazowe, 848, 853, 872
      - wielowersyjne, 864
      - rygorystyczne, 855
    - indeksowe, 875
    - predykatowe, 876
    - z wieloma poziomami ziarnistości, 869, 870
    - BM25, Best Match 25, 1105
    - bufor, 620, 727
      - danych, 815
      - dziennika, 821
      - opróżnienie, 884
      - strategie zastępowania, 823
      - zastąpienie, 884
      - zastępowanie danych, 622
    - buforowanie bloków, 619
    - dyskowych, 883
    - bufory SZBD, 813
- ## C
- CAD, Computer-Aided Design, 425
  - Cassandra, 971
  - certyfikaty cyfrowe, 1230
  - charakterystyka harmonogramów, 827, 829
  - chmura
    - optymalizacja zasobów, 1021
    - problemy z lokalnością danych, 1021
  - chmury obliczeniowe, 1019
  - ciało dokumentu, 487
  - ciągi
    - bitowe, 229
    - znaków, 229
  - CRM, customer relationship management, 92
  - czas
    - transakcji, 1048
    - transmisji bloku, 614
    - wyszukiwania, 614
  - czasowa postać normalna, 1053
  - czasowe
    - bazy danych, 1045
    - kalendarze, 1046
    - reprezentacja czasu, 1046
    - wymiary czasu, 1046
    - złączenie przecięć, 1053
  - część wspólna, 301, 302, 312



czwarta postać normalna,  
549, 589  
  dekompozycja  
  ze złączeniem  
  nieaddytywnym, 591  
czyszczenie pamięci, 896  
czynnik, 721

## D

### dane

  audio, 1067, 1069  
  eksploracja, 1141  
  dotyczące map, 1060  
  niestrukuralne, 484  
  o atrybutach, 1060  
  obrazowe, 1060  
  półstrukturalne, 484  
  przestrzenne, 1063  
  eksploracja, 1063  
  indeksowanie, 1062  
  modele, 1059  
  typy, 1059  
  zarządzanie, 1065  
  zastosowania, 1065  
samoopisujące, 45, 201  
strukturalne, 484  
tekstowe, 1067  
wideo, 1066  
wirtualne, 48  
wrażliwe, 1205  
Datalog, 1073  
  bezpieczeństwo, 1077  
  programy, 1077  
DBMS, Database Management  
  System, 40  
DDB, Distributed Database,  
  913  
decyzje projektowe, 710  
dedukcyjne  
  bazy danych, 1070  
  interpretacja reguł, 1075  
  język Datalog, 1073  
  język Prolog, 1071  
  klauzule Horna, 1074  
  operacje relacyjne, 1081  
  systemy baz danych, 59  
dedukowanie, 174  
definiowanie  
  bazy danych, 40  
  obiektów, 447, 460  
  perspektyw, 78

deklarowanie  
  kursora, 380  
  zmiennych uchwytów, 391  
dekompozycja, 574  
  na schematy, 579  
  relacji, 547, 573  
  zapytań i aktualizacji, 934  
  ze złączeniem  
  nieaddytywnym, 582,  
  591  
  binarna, 547  
  o złączeniach  
  nieaddytywnych, 578  
  testowanie, 577  
  właściwości złączenia  
  nieaddytywnego, 577  
  bezstratnego, 575  
denormalizacja, 54  
DES, Data Encryption  
  Standard, 1228  
destruktor, 433  
diagram  
  EER, 364  
  klas UML, 129, 171, 172  
  schematu związków encji,  
  72, 106  
  związków encji, ER, 101,  
  124, 128, 143–145, 506  
DKNF, domain-key normal  
  form, 594  
dokument XML, 488  
dokumentowe systemy  
  NOSQL, 959, 961  
dokumenty  
  danocentryczne, 490  
  dokumentocentryczne, 490  
  hipertekstowe, 483  
  hybrydowe, 490  
domknięcie, 316  
  zbioru, 565–568  
dopasowywanie ciągów  
  do wzorców, 244  
dopuszczalne  
  rozszerzenia, 530  
  stany relacji, 530  
dostęp  
  nieautoryzowany, 55  
  sekwencyjny, 617  
  swobodny, 613  
dostępność, 916, 957, 1200  
  aktualnych informacji, 60  
  danych, 1205

dostrajanie indeksów, 705  
DRAM, Dynamic RAM, 605  
druga postać normalna, 539,  
  541  
  definicja ogólna, 542  
drugorzędne  
  ścieżki dostępu, 665  
  urządzenia pamięciowe,  
  609  
drugorzędny mechanizm  
  składowania, 604  
drzewa, 509, 681  
  B-powiązań, 874  
  częstych wzorców, 1151  
  czwórkowe, 1063  
  decyzyjne, 1160  
  krzaczaste, 787  
  lewostronnie zagłębione,  
  787  
  niezrównoważone, 681  
  prawostronnie  
  zagłębione, 787  
  wyszukiwania, 682  
  zapytań, 313, 721, 758,  
  767  
  heurystyczna  
   optymalizacja, 760  
  kanoniczne, 760  
  końcowe, 760  
  początkowe, 760  
drzewiasta struktura danych,  
  682  
DTD, document type  
  definition, 491  
dublowanie danych, 650  
duplikat, 706  
dynamiczne  
  indeksy wielopoziomowe,  
  681  
  operatory przestrzenne,  
  1061  
  programowanie, 388  
  rozszerzanie plików, 642  
  wiążanie, 439  
dynamiczny  
  indeks wielopoziomowy,  
  681  
  język SQL, 381  
DynamoDB, 967  
  model danych, 968  
dysk twardy, HDD, 609

## dyski

- magnetyczne, 609
- bloki dyskowe, 611
- cylinder, 610
- czas przepisania, 1248
- czas transmisji bloku, 1247
- czas wyszukiwania, 1247
- główce, 614
- klastry, 613
- kontroler dysku, 614
- opóźnienie rotacyjne, 1247
- pojemność, 609
- sektory, 610
- strony, 611
- szczeliny
  - międzyblokowe, 611
- szybkość transmisji, 1247
- ścieżki, 610
- średni czas dostępu, 1247
- zbiorcza prędkość transmisji danych, 615, 1248
- zwiększanie wydajności, 615
- optyczne, 606
- SSD, 616

## dyskretna transformata

- Fouriera, DFT, 1066

## dyspozycyjny mechanizm

- bezpieczeństwa, 1201

## dziedziczenie, 152, 436, 440, 446

- atrybutów i związków, 152

- EXTENDS, 453

- jednokrotne, 162

- tabel, 446

- typów, 154, 446

- typu EXTENDS, 465

- wielokrotne, 360, 439

- wybiórce, 439

- zachowania, 453

## dziedzina, 197

- atrybutu, 198

- wartości, 111

- wartości SQL, 229

- wiedzy, 173

- dzielenie, 310, 312

- dziennik systemowy, 821, 887, 1204

## E

- ECR, Entity-Category-Relationship, 164

- EER, 347

- odwzorowywanie
  - na schemat obiektowy, 466

- EFD, enterprise flash drives, 616

- efektywne przetwarzanie
  - zapytań, 56

- egzemplarze, 70–73

- relacji, 198
- specjalizacji, 156
- związku, 115

- ekosystem Hadoopa, 998

- eksploracja danych, 1141,

- 1143, 1166, 1170

- hurtownie danych, 1142

- proces odkrywania

- wiedzy, 1142

- programy, 1172

- przestrzennych, 1063

- technologie, 1142

- ekstensja, 437, 440, 458, 478

- obiektów, 435

- relacji, 198

- elastyczność, 60, 61

- elementy

- baz danych, 813

- egzemplarza związku, 115

- proste, 489

- schematu, 226

- eliminowanie

- duplikatów, 752

- powtórzeń, 298

- zagnieżdżenia, 785

- encja, 71

- atrybuty, 107

- enkapsulacja operacji, 432, 440, 445

- ER, Entity-Relationship, 101, 347

- ERP, enterprise resource planning, 92

- etykieta, 485

- użytkownika, 1235

- ewolucja schematu, 74

## F

- fałszywe krotki, 526

- fasety, 1129

- FCIP, Fibre Channel over IP, 655

- federacyjny system baz
  - danych, FSD, 937

- fizyczne projektowanie bazy
  - danych, 608, 665, 708, 710

- format danych, 197

- formatowanie stylu, 487

- formaty przechowywania
  - rekordów, 625

- formularz, 80

- serwera PHP, 414

- fragmentacja danych, 918

- mieszana, 920

- pionowa, 919

- pozioma, 919, 958

- funkcja, 112, 436

- EXISTS, 263, 277

- map, 992

- modyfikatora, 445

- obserwatora, 445

- odległości, 1066

- randomizacji, 636

- SQLAllocHandle, 390

- UNIQUE, 265

- XMLAGG, 511

- XMLATTRIBUTES, 511

- XMLELEMENT, 511

- XMLFOREST, 511

- XMLROOT, 511

- funkcje

- agregujące, 268, 294, 314, 473, 774

- hurtowni danych, 1191

- kosztu, 777, 786

- dla operacji JOIN, 783

- dla operacji SELECT, 778

- mieszające, 601, 603

- mieszające

- partycjonowania, 741

- składowane, 399

- skryptów, 487

- w PHP, 412

- funkcjonalne modele danych, 117

**G**

generalizacja, 155, 156, 163, 169, 176  
 generator  
   fałszywych krotek, 526  
   kodu, 721  
   typów, 430  
     Array<T>, 455  
     Bag<T>, 454  
     List<T>, 454  
     Set<T>, 454  
 geograficzne systemy  
   informacyjne, GIS, 37  
 Giraph, 1016  
 GIS, Geographical Information Systems, 37, 1058  
 globalny menedżer  
   odtworzenia, 901  
   transakcji, 928  
 głosowanie, 927  
 głosowe przekazywanie  
   danych, 81  
 Google File System, 972  
 graf, 505, 509  
   krawędzie, 760  
   oczekiwania, 858  
   zapytań, 329, 760  
 graficzny interfejs  
   użytkownika, GUI, 57, 80  
 grafowe  
   bazy NOSQL, 975  
   systemy NOSQL, 960  
 GRU, Group LRU, 823  
 grupowanie, 270, 314, 1163, 1256

**H**

Hadoop, 983, 988, 998, 1017  
   zalety technologii, 1008  
 Hadoop 2, 1009  
 harmonogramy  
   konfliktowo-  
   -szeregowalne, 830  
   nieszeregowane, 830  
   rodzaje równoważności,  
   840  
   sprawdzanie  
   szeregowalności, 834  
   szeregowane, 829, 835, 873  
   ściśle, 829  
   transakcji, 825

haszowanie dynamiczne, 642  
 HBase, 972

  model danych, 972  
   operacje CRUD, 974  
   składowanie danych, 974  
   system rozproszony, 974  
   w YARN, 1016  
   wersjonowanie, 972  
 HDD, hard disk drive, 609  
 HDFS, Hadoop Distributed  
   File System, 972, 985  
   architektura systemu, 994  
   operacje wejścia-wyjścia,  
   996  
   skalowalność systemu, 997  
   wymagania, 994  
   zarządzanie replikami,  
   996

heterogeniczność  
 semantyczna, 939

heurystyczna optymalizacja,  
 760

  algebraiczna, 766  
   zapytań, 758

hierarchia  
   atrybutów złożonych, 108  
   dokumentu XML, 509  
   klas, 438  
   klasyfikacji, 1145  
   pamięciowa, 605  
   specjalizacji, 160  
   typów, 440, 436, 437

hierarchiczne perspektywy,  
 505

hierarchiczny model danych,  
 488

histogramy, 778

Hive, 1006  
   architektura wtyczek,  
   1007  
   obsługa języka SQL, 1007  
   optymalizacje, 1007  
   tabele, 1007

HTML, Hypertext Markup  
 Language, 483

hurtownia danych, 1177, 1179  
   funkcje, 1191  
   modelowanie danych, 1181  
   perspektywy, 1192  
   tworzenie, 1187

hybrydowe systemy NOSQL,  
 960

**I**

identyfikacja, 175

identyfikator  
   obiektu, 444, 448  
   uwierzytelniania, 226,  
   1207

IDL, Interface Definition  
 Language, 460

iloczyn  
   kartezjański, 303, 312  
   zbiorów, 243

implementacja  
   antyłączeń, 748  
   aktywnych baz danych,  
   1039  
   operacji agregujących, 745  
   operacji fizycznych, 749  
   złączeń, 734, 735, 746  
   częściowych, 748  
   nierównościowych, 748

indeksowane pliki  
   sekwencyjne, 680

indeksowanie  
   bitmapowe, 1186  
   danych przestrzennych,  
   1062  
   oparte na funkcji, 702  
   złączeń, 1186

indeksy, 56  
   bitmapowe, 699, 712  
   dostrajanie, 705  
   drugorzędne, 673  
   fizyczne, 703  
   główne, 666–668  
   klastrowania, 666, 670  
   logiczne, 703  
   mieszające, 712  
   na wielu kluczach, 696  
   niezageszczone, 746  
   odwrócone, 1112  
   oparte na mieszaniu, 699  
   tworzenie, 704  
   uporządkowane, 666  
   na wielu atrybutach, 697  
   wielopoziomowe, 677,  
   681  
   zageszczone, 746  
   złączeń przestrzennych,  
   1063

informacje  
 nadmiarowe, 521  
 o obiekcie, 433  
 o schemacie, 485  
 o zdarzeniach, 1047  
 w katalogu bazy danych, 733  
 w krotkach, 521  
 infrastruktura klucza  
 publicznego, 1227  
 integralność, 1200  
 encji, 209  
 odwołań, 58, 209  
 intensja  
 kolekcji encji, 110  
 relacji, 198  
 interakcja w programowaniu, 373  
 interaktywne interfejsy, 370  
 interfejs  
 Fibre Channel, 653  
 JDBC, 394  
 programowy aplikacji,  
 API, 90, 372, 1171  
 SAS, 614  
 SATA, 614  
 SCSI, 614  
 SQL/CLI, 388, 389  
 WWW, 371  
 interfejsy  
 baz danych, 77  
 dla administratorów  
 baz danych, 81  
 dla użytkowników  
 parametrycznych, 81  
 dla systemu HDFS, 1007  
 dla wielu użytkowników,  
 57  
 interaktywne, 371  
 internetowe, 80  
 języka naturalnego, 81  
 oparte na formularzach,  
 80  
 operacji, 433  
 przeglądania, 80  
 użytkownika, 1171  
 systemów zarządzania, 80  
 internet  
 wyszukiwanie informacji  
 i danych, 1093  
 internetowe bazy danych, 405  
 interpretacja relacji, 202

inżynier oprogramowania, 52  
 ISAM, Indexed Sequential  
 Access Method, 665  
 iSCSI, 655  
 iterator, 379, 454, 749  
 pozycyjny, 386  
 izolacja, 49  
 snapshotów, 825, 843,  
 866, 868

## J

jakość danych, 1232  
 Java, 383  
 JavaScript, 419  
 jawne deklarowanie zbiorów,  
 265  
 JDBC, 388, 393  
 importowanie biblioteki  
 klas, 395  
 obiekt  
 polecenia, 396  
 połączenia, 395  
 typu ResultSet, 397  
 parametr polecenia, 396  
 polecenia języka SQL, 397  
 tworzenie zmiennych, 395  
 wczytywanie sterownika,  
 395  
 wiązanie parametrów  
 polecenia, 396  
 jezioro danych, 1022  
 język  
 C, 389  
 deklarowanie zmiennych  
 uchwytów, 391  
 dołączanie biblioteki  
 funkcji, 390  
 polecenia języka SQL,  
 391  
 połączenie z bazą, 391  
 przetwarzanie wyniku  
 zapytania, 392  
 rekord polecenia, 391  
 rekord środowiska, 391  
 wiązanie parametrów  
 polecenia, 391  
 wykonywanie polecenia,  
 392  
 C++, 476  
 Datalog, 1071  
 definicji

danych, 78  
 formularzy, 80  
 obiektów, 431  
 perspektyw, 78  
 składowania, 78  
 HTM, 405  
 Java, 383  
 polecenia języka SQL,  
 393  
 JavaScript, 419  
 ODL, 431, 447, 460  
 OQL, 468  
 PHP, 405  
 PL/SQL, 372  
 Prolog, 1071  
 QBE, 1251  
 SQL, 223  
 SQL/PSM, 372  
 SQLJ, 374  
 TSQL2, 1055  
 WSDL, 504  
 XML, 483, 494  
 XSL, 504  
 XSLT, 504  
 zapytań Cypher, 976, 977  
 języki  
 manipulowania danymi,  
 78, 79  
 programowania baz  
 danych, 402  
 systemów zarządzania, 77  
 JobTracker, 1011  
 JSON, JavaScript Object  
 Notation, 420  
 JSP, Java Server Pages, 419

## K

kalendarze, 1046, 1057  
 kaskadowe przeniesienie, 214  
 katalog, 226, 883, 895  
 przesłaniania, 895  
 katalogi  
 scentralizowane, 946  
 z pełną replikacją, 946  
 z replikacją częściową,  
 946  
 kategorie, 164, 171  
 częściowe, 166  
 kompletne, 166  
 modeli danych, 71

- klasa, 170, 433
    - bazowa, 171
    - liści, 171
  - klastrowanie
    - fizyczne, 647
    - na podstawie gęstości, 1064
    - przestrzenne, 1064
  - klasy bezpieczeństwa, 1212
  - klasyfikacja, 174, 1160
    - przestrzenna, 1063
    - systemów zarządzania, 92
  - klauzula, *Patrz także* operacja, polecenie, słowo kluczowe
    - CASCADE, 234
    - CASCADE ON DELETE, 235
    - CASCADE ON UPDATE, 235
    - CHECK, 235, 278
    - FOR, 503
    - FOR UPDATE OF, 380
    - FOREIGN KEY, 234
    - FROM, 237, 261, 275
    - GROUP BY, 270, 275, 746
    - HAVING, 270, 272
    - INTO, 378
    - LET, 502, 503
    - NOT EXISTS, 278
    - ORDER BY, 246, 275, 381, 503, 727
    - PRIMARY KEY, 234
    - RETURN, 502, 503
    - SELECT, 237, 275
    - SET DEFAULT, 234
    - SET NULL, 234
    - WHERE, 237, 241, 503
    - WITH, 273
  - klauzule Horna, 1074
  - klient, 89
  - klient-serwer, 944
  - klucz, 58, 109, 205, 458, 534
    - częściowy, 122, 538
    - drugorzędny, 673
    - główny, 206, 498, 534, 667
  - kandydujący, 206, 529, 534
  - obcy, 209, 351, 498
  - publiczny, 1228
  - unikatowy, 206
  - kolekcja
    - obiektów, 435
    - trwała, 438
    - ulotna, 438
  - kolekcje jednoelementowe, 473
  - kolizja, 638
  - kolor, 1067
  - kolumnowe
    - składowanie danych, 708
    - systemy NOSQL, 959, 972
  - kolumny niepowiązane, 392
  - kompilator zapytań, 83
  - komponenty systemu Hive, 1006
  - komputer kliencki, 88
  - komunikacja, 86, 377
  - konceptyjne
    - modelowanie danych, 99
    - projektowanie
      - baz danych, 99
      - obiektowych baz danych, 465
      - relacyjnych baz danych, 465
    - reprezentacje danych, 47
  - konceptualizacja, 179
  - konserwowanie, 40
  - konstrukcja
    - AS, 266
    - CASE, 274
    - COUNT, 269
    - schematu, 72
  - konstruktor, 433
    - array, 431
    - atomów, 430
    - bag, 431
    - dictionary, 431
    - list, 431
    - set, 431
    - struktur, 430
    - typów, 430, 440
    - typów kolekcji, 430
  - konstruowanie bazy danych, 40
  - konta użytkowników, 1204
  - kontekst domyślny, 384
  - kontrola
    - dostępu, 1212
    - dla danych XML, 1218
    - na poziomie wierszy, 1217
    - oparta na rolach, 1215
    - nadmiarowości, 53
    - przepływu, 1202, 1225
  - kontroler dysku, 614
  - kontrolowana nadmiarowość, 54
  - konwersja
    - blokad, 852
    - drzewa zapytania, 762
  - kopie bezpieczeństwa, 57, 84, 902
  - korzyści ekonomiczne, 60
  - koszt
    - operacji JOIN, 783
    - operacji SELECT, 778
    - przesyłu danych, 931
  - kosztowa optymalizacja zapytań, 776
  - koszyk, 1146
  - krata specjalizacji, 160, 161
  - krawędź grafu, 329, 760
  - krotki, 198, 200, 260
    - fałszywe, 526
    - wartości, 260
    - zawieszone, 584
  - kształt, 1068
  - cursor, 379, 380
  - kwalifikator, 239
    - AS, 266
  - kwalifikowane nazwy atrybutów, 239
  - kwantyfikatory, 473
    - egzystencjalne, 327, 330
    - uniwersalne, 327, 330, 331
- ## L
- LBA, Logical Block Address, 613
  - leksykon, 1112
  - licznik, 621
  - liczność dziedziny, 199
  - limity czasu, 859
  - liniowe
    - przyspieszenie, 751
    - skalowanie, 751
  - literały, 429, 448
    - atomowe, 449
    - kolekcji, 449
    - strukturalne, 449
  - logiczna teoria, 179
  - logika trójwartościowa, 258

## Lucene

- indeksowanie, 1114
- wyszukiwanie, 1115

## Ł

- łączenie, 638, 640
  - okresów, 1056
  - operacji, 748
  - z bazą danych, 376
- łączność operacji, 764

## M

## magazyn

- danych typu klucz-  
wartość, 72, 967, 971
- NOSQL, 967
- ODS, 647

- manipulowanie bazą danych, 40

- mapowanie, 989

- MapReduce, 983, 988, 999, 1017

- model programowania, 990

- odporność na usterki, 1000
- procedura przedstawiania, 1001

- przebieg prac, 1000
- szeregowanie zadań, 1001
- środowisko

- uruchomieniowe, 999
- zalety technologii, 1008
- złączenia, 1002

- matematyczna relacja, 195

- materializacja perspektywy, 282

- MDM, master data  
management, 1187

## mechanizm

- bezpieczeństwa, 1201
- grupowania, 314
- komunikacji, 86
- potokowy, 748
- RDD, 1025
- składowania
  - drugorzędny, 604
  - podstawowy, 604
  - trzeciorzędny, 604
- SQL/PSM, 400
- wnioskowania, 173

## menadżer

- składowanych danych, 83
- sterowników, 394
- blokad, 849
- bufora, 620
- danych składowanych, 84
- odtworzenia, 901
- transakcji, 928
- węzła, 1013, 1015
- zasobów, 1014

- metadane, 40, 45, 74

- metoda, 433

- DBMIN, 823

- metodologia projektowania

- wstępująca, 518
- zstępująca, 518

## metody

- aktywnego zbioru, 823
- implementacji złączeń, 735
- odwzorowywania
  - generalizacji, 357
- odwzorowywania
  - specjalizacji, 357
- organizacji pliku, 629, 647
- podziału na dziedziny, 823
- wyszukiwania, 729, 731, 733

- miara F, 1118

## mieszanie

- dynamiczne, 644
- liniowe, 644
- partycjonowane, 697, 712
- rozszerzalne, 642
- stabilne, 969, 970
- statyczne, 641
- wewnętrzne, 636
- wielokrotne, 639
- zewnętrzne, 639

- migawka bazy danych, 73

- moc wyrażania, 324

## model

- bezpieczeństwa, 1215
- danych, 47, 70
  - fizyczny, 71
- hierarchiczny, 71, 95, 488
- kolumnowy, 94
- konceptyjny, 71
- niskopoziomowy, 71, 95
- obiektowy, 72
- oparty na dokumentach, 94

- oparty na grafie, 94

- oparty na rekordach, 72
- przestrzennych, 1059
- relacyjny, 71, 193
- reprezentacyjny, 71, 95
- rozszerzony, 1033
- samoopisujący, 72
- sieciowy, 71, 94
- typu klucz-wartość, 94
- w systemie HBase, 972
- w systemie Neo4j, 975
- wysokopoziomowy, 71, 95

- XML-owy, 94

- z DynamoDB, 968

- EER, 347

- odwzorowanie w relacje, 357

- ER, 347

- odwzorowanie w model  
relacyjny, 355

## informacji

- przestrzennych, 1060
- macierzy dostępu, 1208
- MapReduce, 983, 989, 999
- obiektoowo-relacyjny, 251, 440

- obiektoowy, 425

- ODMG, 447, 448

- oparty na przestrzeni  
wektorowej, 1102

- probabilistyczny, 1104

- programowania

- MapReduce, 990

- relacyjny, 195, 196, 347, 355

- replikacji, 957

- semantyczny, 1106

- związków encji

- rozszerzony, EER, 102, 151–191

- związków encji, ER, 101

## modelowanie

- danych, 1181
- obiektoowe, 151
- typów UNII, 164

## moduły, 53, 82

- buforujące, 56

- klienta, 69

- serwera, 69

- trwale składowane, 399

- wnioskujące, 1070

- zarządzania buforami, 83



modyfikowanie, 524  
 kodu, 1221  
 MongoDB, 961  
 cechy systemu  
 rozproszonego, 965  
 operacje CRUD, 965  
 replikacja, 965  
 sharding, 966  
 monitorowanie wydajności,  
 85  
 multimedialne bazy danych,  
 1065  
 analiza obrazów, 1067  
 dane audio, 1069  
 opis obrazów, 1069  
 wykrywanie obiektów,  
 1068

## N

nadklasa, 152  
 specjalizacji, 154  
 nadklucze, 205, 534  
 nadmiarowość, 53  
 nadtyp, 436  
 nagłówki  
 dokumentu, 487  
 pliku, 627  
 NameNode, 995  
 napęd  
 dyskowy, 609, 613  
 taśmowy, 617  
 narzędzia, 53  
 baz danych, 85  
 do eksploracji danych,  
 1172  
 eksploracji danych, 1170  
 konwersji, 85  
 narzędzie grep, 992  
 NAS, network-attached  
 storage, 654  
 nawiasy klamrowe, 538  
 nazewnictwo konstrukcji  
 schematu, 125  
 nazwany iterator, 385  
 nazwy  
 atrybutów kwalifikowane,  
 239  
 obiektów, 434  
 ograniczeń, 235  
 ról, 117  
 zastępcze, 239  
 Neo4j, 975  
 interfejsy, 979  
 język zapytań Cypher, 977  
 model danych, 975  
 system rozproszony, 979  
 niepodzielność, 50  
 niewystarczalność  
 postaci normalnych, 573  
 niezależność  
 danych, 76  
 fizyczna, 77  
 logiczna, 76  
 programu od danych, 46  
 programu od operacji, 47  
 niezgodność impedancji, 372  
 niezmiennosc, 429  
 normalizacja, 515, 517  
 danych, 54, 533  
 relacji, 532  
 NOSQL, 955  
 BigTable, 972  
 Cassandra, 971  
 cechy systemów, 957  
 dostęp do danych, 958  
 dostępność, 957  
 DynamoDB, 967  
 grafowe bazy, 975  
 HBase, 972  
 kategorie systemów, 959  
 modele replikacji, 957  
 MongoDB, 961  
 Neo4j, 975  
 para klucz-wartość, 967  
 Redis, 971  
 replikacja, 957  
 sharding plików, 958  
 skalowalność, 957  
 spójność ostateczna, 957  
 systemy kolumnowe, 972  
 twierdzenie CAP, 960  
 Voldemort, 968  
 wersjonowanie, 959  
 nośnik danych, 604  
 notacja  
 diagramowa, 1243  
 diagramów związków  
 encji, 124, 126  
 diagramu EER, 153, 158,  
 159  
 dla grafów zapytań, 329  
 drzew zapytań, 313, 758  
 grafów zapytań, 758

języka  
 Datalog, 1073  
 Prolog, 1071  
 UML, 128  
 modeli związków encji,  
 1243  
 modelu relacyjnego, 203  
 schematyczna, 531  
 z kropką, 434, 447, 453  
 numer sekwencyjny  
 dziennika, 897

## O

obiekt, 427  
 typu Connection, 395  
 typu PreparedStatement,  
 396  
 typu ResultSet, 397  
 obiektowa pamięć masowa,  
 656  
 obiektowe  
 bazy danych, 425, 465  
 czas, 1053  
 języki programowania,  
 427  
 obiektowy  
 język zapytań, OQL, 468  
 model danych, 92, 94, 251  
 obiekty, 448  
 atomowe, 448, 456  
 iteratora pozycyjnego, 387  
 kolekcji, 454  
 molekularne, 178  
 trwałe, 56, 434  
 ulotne, 434  
 wyjątków, 174  
 złożone, 178, 443  
 obiekty-fabryki, 458  
 obliczenia kognitywne, 985  
 obowiązkowy mechanizm  
 bezpieczeństwa, 1201  
 obraz, 1066  
 końcowy, 884  
 pierwotny, 829, 884  
 semantyczne opisywanie,  
 1069  
 obsługa  
 nadmiarowości, 650  
 przepełnienia pakietów,  
 640  
 transakcji, 841



- wersji atrybutów, 1053
- zarządzania transakcjami, 930
- obszar komunikacji języka, 377
- ochrona
  - bazy danych, 40
  - dostępu, 1204
- oczekiwanie ostrożne, 858
- oddzielenie programów od danych, 46
- odkrywanie wiedzy, 1142, 1143
- ODL, Object Definition Language, 431, 447, 460
  - schemat bazy danych, 461
- ODMG, Object Data Management Group, 447
  - bazy danych, 460
  - dziedziczenie, 453
  - ekstensje, 458
  - interfejsy, 454
  - klasy, 454
  - klucze, 458
  - literały, 449
  - obiekty-fabryki, 458
  - stan, 448, 452
  - wiązanie z językiem C++, 476
  - wyjątki, 454
  - zachowanie, 452
- odporność
  - baz danych, 1233
  - na błędy, 1015
  - na podział, 917
- odpowiadanie na pytania, 1132
- ODS, operational data store, 647
- odtworzalność
  - harmonogramu, 827
- odtwarzanie, 818, 827
  - aktualizacje natychmiastowe, 893
  - aktualizacje odroczone, 890
  - ARIES, 897
  - bazy danych, 809, 881
  - danych, 925
  - NO-UNDO/REDO, 890
  - obrazu pierwotnego, 829
  - po awariach, 902
- UNDO/NO-REDO, 894
  - w systemach wielu baz danych, 901
- odwołanie, 444, 446
- skrośne, 351
- odwzorowanie, 200
  - atrybutów
    - wielowartościowych, 353
  - binarnych typów
    - związków
      - 1:1, 351
      - M:N, 352
      - 1:N, 352
  - generalizacji, 357
  - kategorii, 361
  - modelu
    - danych, 104
    - EER, 357, 466
    - ER, 355
  - n-składnikowych typów
    - związków, 354
  - słabych typów encji, 350
  - specjalizacji, 357
  - typów encji, 349
  - typów unii, 361
  - współdzielonych podklas, 360
- odzyskiwanie danych, 57
- ograniczenia
  - dla krotek, 235
  - dla typów związków, 118
  - dziedziny, 204
  - generalizacji, 157
  - klucza, 205, 232
  - kompletnej specjalizacji, 159
  - kompletności, 159
  - minimalnej liczności, 120
  - modelu relacyjnego, 203
  - propagacji uprawnień, 1211
  - przejęć, 211
  - rozłączności, 158
  - specjalizacji, 157
  - stanów, 211
  - strukturalne, 114
  - strukturalne typów
    - związków, 121
  - udziału, 120
  - unikatowości, 111
  - wartości pustych, 205
- więzów integralności
  - odwołań, 232
- związków
  - trójskładnikowych, 136
- okres istnienia obiektu, 448
- ontologia, 173, 178
- opcja
  - CASCADE, 285
  - GRANT, 1209
  - RESTRICT, 285
- operacja, 47, 70
  - CARTESIAN PRODUCT, 743
  - EXCEPT, 744
  - INTERSECTION, 738, 743
  - JOIN, 267, 734, 738
    - funkcje kosztu, 783
  - MINUS, 743, 744
  - PROJECT, 738
  - REDO, 883
  - SELECT, 729
  - SET DIFFERENCE, 738, 743, 744
  - UNDO, 883
  - UNION, 738, 743
- operacje
  - agregujące, 745
  - aktualizacji, 212, 215
  - algebry relacyjnej, 301, 312
  - arytmetyczne, 244
  - binarne, 294, 301
  - binarne na relacjach, 305
  - CRUD, 959, 965
  - CRUD w systemie HBase, 974
  - części wspólnej, 301
  - dodatkowe, 819
  - domknięcia rekurencyjne, 316
  - dzielenia, 310
  - iloczynu kartezjańskiego, 303
  - konstruujące obiekty, 459
  - na plikach, 627
  - na zbiorach, 753
  - odczytu i zapisu, 813
  - porównania, 258
  - projekcji, 297, 743
  - przemienne, 297
  - przypisania, 299
  - relacyjne dodatkowe, 314

- operacje
    - rozwijania, 1184
    - równo-złączenia, 307
    - różnicy, 301
    - selekcji, 295, 729
    - sumy, 301
    - sumy zewnętrznej, 319
    - teoriomnogościowe, 743
    - unarne, 294, 295
    - usuwania, 214
    - wejścia-wyjścia, 996
    - wstawiania, 212
    - złączenia, 305
      - naturalnego, 307
      - zewnętrznego, 317
    - zmiana nazwy, 299, 300
  - operator, 53
    - antyłączenia, 726
    - grupowania, 475
    - INNER JOIN, 267
    - logiczny
      - and, 1254
      - or, 260, 1254
    - porównania
      - BETWEEN, 246
      - IN, 260
      - LIKE, 244
    - selekcji unarny, 297
  - operatory
    - kolekcji, 473
    - metryczne, 1061
    - projekttywne, 1061
    - topologiczne, 1061
    - złączeń częściowych, 725
  - opóźnienie rotacyjne, 614
  - oprogramowanie
    - komunikacyjne, 86
  - optyczne pamięci masowe, 606
  - optymalizacja
    - adaptacyjna, 800
    - drzew zapytań, 760
    - fizyczna, 789
    - selekcji, 781
    - zapytań, 719, 757, 794
      - kosztowa, 775
      - semantyczna, 803
      - w bazach Oracle, 799
      - w hurtowniach danych, 796
    - zasobów, 1021
    - złączenia, 786
  - optymalizator
    - fizyczny, 799
    - globalny, 799
    - zapytań, 83
  - OQL, 468
    - funkcje agregujące, 473
    - kolekcje
      - jednoelementowe, 473
    - kwantyfikatory, 473
    - nazwane zapytania, 472
    - operator grupowania, 475
    - perspektywy, 472
    - punkty wejścia, 469
    - wyniki zapytań, 470
    - wyrażenia kolekcji, 474
    - wyrażenia ścieżkowe, 470
    - zapytania, 469
    - zmiennie iterujące, 469
  - organizacja plików, 609
  - orientacja przeglądania, 381
  - osadzanie
    - kodu SQL, 378, 402
    - polecień, 371
      - w SQL, 374
      - w SQLJ, 383
  - osiągalność, 434
  - otwarte łącze baz danych, 90
- P**
- pakiet mieszający, 736
  - pamięć
    - DRAM, 605
    - flash, 606
    - masowa, 605, 607
    - iSCSI, 655
    - objektowa, 656
    - SSD, 616
    - nieulotna, 608
    - podręczna SZBD, 727, 883
    - RAM, 605
    - ulotna, 608
  - para klucz-wartość, 968
  - parametr MTBF, 650
  - parametry
    - dysków, 1247
    - polecenia, 396
  - partycjonowanie cykliczne, 751
  - pełny udział, 120
  - personel pomocniczy, 53
  - perspektywy, 48, 280, 472, 1192
    - aktualizacja, 282
    - materializacja, 282
    - mechanizm
      - uwierzytelniania, 284
    - przyrostowa aktualizacja, 773
    - hierarchiczne, 505
    - wewnątrzwersyjne, 284
    - zmaterializowane, 772
  - pętla while, 387
  - PHP, 405
    - automatyczna zmienna
      - globalna, 407
    - dane z formularzy, 417
    - formularze, 414
    - funkcje, 412
    - nawiązywanie połączenia, 415
    - nazwy zmiennych, 408, 409
    - programowanie baz
      - danych, 415
    - tablice, 410
    - typy danych, 409
    - wstawianie rekordów, 417
    - zmiennie, 409, 414
  - piąta postać normalna, 552, 592
  - pierwsza postać normalna, 201, 535, 541
  - platforma
    - PACMAN, 1021
    - Tez, 1015
    - YARN, 1009
  - pliki, 608, 624
    - B-drzewa, 648
    - bezpośrednie, 632
    - dynamiczne rozszerzanie, 642
    - główne, 635
    - klastrowane, 666
    - matrycowe, 697, 712, 1062
    - metoda
      - dostępu, 629
      - organizacji, 629
    - mieszanie, 636
    - dynamiczne, 644
    - liniowe, 644
    - zewnętrzne, 639

- nagłówki, 627
- nieuporządkowanych rekordów, 631
- odwrócone, 707
- operacje
  - aktualizacji, 627
  - pobierania, 627
  - zbiorowe, 629
- płaskie, 196
- podstawowe metody
  - organizacji, 648
- poleceń, 371
- posortowane, 632
- przepełnienia, 635
- rekordów mieszanych, 647
- reorganizacja, 631
- sekwencyjne, 631
  - indeksowane, 680
- statyczne, 630
- stertowe, 631
- struktury indeksowe, 665
- transakcji, 635
- uporządkowanych rekordów, 632
- współczynnikapełnienia, 646
- współużytkowanie, 654
- wybranie rekordów, 627
- względne, 632
- płaski model relacyjny, 201
- pobieranie
  - danych, 1252
  - dokumentów XML, 505
- poddrzewo, 681
- podejścia obiektowe, 423
- podklasa, 152, 170
  - dzielona, 162
  - zdefiniowana
    - przez predykat, 170
  - zdefiniowana przez użytkownika, 158, 170
- podpis cyfrowy, 1230
- podstawowa metoda
  - organizacji plików, 687
- podstawowy mechanizm
  - składowania, 604
- podsystem
  - bezpieczeństwa
    - i autoryzacji, 55
  - sterowania
    - współbieżnego, 855
  - tworzenia kopii
    - bezpieczeństwa, 57
- podtyp, 436
- podzbiory atrybutów, 205
- pojemność
  - dysku, 609
  - składowania, 605
- pokrycie minimalne, 564, 571
- pole
  - indeksujące, 665, 666, 673
  - klastrowania, 670
  - wyszukiwania, 683
- polecenia
  - osadzonego języka, 375
  - zmiany schematu, 285
- polecenie
  - ALTER, 286
  - CALL, 400
  - CLOSE CURSOR, 379
  - CREATE ASSERTION, 277
  - CREATE TABLE, 226, 228
  - CREATE TRIGGER, 277, 278
  - CREATE TYPE, 443
  - CREATE VIEW, 280
  - DELETE, 249
  - DIAGNOSTIC SIZE, 841
  - DROP, 285
  - DROP TABLE, 248
  - DROP VIEW, 282
  - EXECUTE, 382
  - FETCH, 379
  - INSERT, 247
  - ISOLATION LEVEL, 841
  - OPEN CURSOR, 379
  - PREPARE, 382
  - READ ONLY, 841
  - READ UNCOMMITTED, 841
  - READ WRITE, 841
  - SELECT, 236
  - SERIALIZABLE, 841
  - SET TRANSACTION, 841
  - UPDATE, 250
- polimorfizm, 440
  - operacji, 438
- polityka
  - kontroli dostępu, 1219
  - przepływu, 1225
- połączenie z bazą danych, 391, 415
- porównanie
  - obiektów, 429
  - zbiorów i wielozbiorów, 260
- porządkowanie
  - krotek, 200
  - wartości, 200
  - według znaczników czasu, 860, 862
  - złążeń, 787
- postaci normalne, 533, 535
- postać normalna
  - Boyce'a-Codda, 545
  - czwarta, 549, 589
  - definicje ogólne, 542
  - druga, 539
  - klucza dziedziny, DKNF, 594
  - niewystarczalność, 573
  - oparta na kluczach głównych, 532
  - piąta, 552, 592
  - pierwsza, 535
  - trzecia, 540
- poufność, 1200
- powtarzanie historii, 897
- poziom
  - izolacji, 825, 842
  - ziarnistości, 869
- praca z wieloma kopiami, 925
- pracownicy poza sceną, 52
- prawa własności
  - intelektualnej, 1232
- predykat, 202
  - definiujący podklasę, 158
  - homogeniczności, 1066
- prekompilator, 83, 371
- probabilistyka, 1104
- problem
  - aktualizacji tymczasowej, 816
  - błędnej sumy, 818
  - fantomowy, 875
  - niezgodności impedancji, 56
  - odczytu
    - niepowtarzalnego, 818
  - utraconej aktualizacji, 816
  - z lokalnością danych, 1021
  - z prywatnością, 1231
- procedury
  - arytmetyczne, 593
  - składowane, 59, 278, 398, 399

- proces
    - generalizacji, 163
    - JobTracker, 999
    - odkrywania wiedzy, 1142
    - projektowania, 103
    - TaskTracker, 999
  - procesor wykonawczy, 84
  - produkt kartezjański, 303
  - program
    - aplikacji, 373
    - klienta, 84, 373
  - programiści
    - aplikacji, 52
    - baz danych, 50
    - narzędzi, 53
    - systemu zarządzania bazą danych, 53
  - programowanie
    - baz danych, 370
    - internetowych, 405
    - w PHP, 415
    - dynamiczne, 388
    - określanie kolejności złączeń, 790
    - logiczne, 1070
    - statyczne, 388
    - z osadzeniem kodu, 378
  - projekcja, 297, 312, 574, 752
  - uogólniona, 314
  - projekt
    - ER, 123
    - fizyczny, 44
    - konceptyjny, 113
    - logiczny, 44
  - projektowanie
    - aktywnych baz danych, 1039
    - baz danych, 515
    - etapy, 104
    - fizyczne, 105
    - konceptyjne, 103, 465
    - logiczne, 104
    - metodą wstępującą, 563
    - na podstawie analiz, 563
    - relacyjne, 586
    - relacyjne przez syntezę, 563
    - relacyjnych baz danych, 347, 563
    - schematów relacji, 519
    - schematów relacyjnych baz danych, 579
  - Prolog, 1071
  - protokoły
    - optymistyczne, 847
    - sterowania
      - współbieżnego, 847
    - zapobiegania zakleszczeniom, 857
  - protokół
    - blokowania dwufazowego, 853
    - iSCSI, 655
    - SOAP, 504
    - zatwierdzania
      - dwufazowego, 901, 929
      - zatwierdzania trójfazowego, 929
  - prywatność, 1206, 1231
  - przechowywanie baz danych, 607
  - przeciążanie
    - funkcji, 446
    - operatorów, 438, 440
  - przedziały, 1235
  - przeglądy
    - indeksu, 730
    - plików, 729
  - przekształcanie
    - kwantyfikatorów, 330
    - operacji algebry relacji, 761
  - przemienność operacji, 763
  - przenoszenie, 638
  - przepełnienie pakietów, 640
  - przeplatanie, 813
    - danych, 649
    - danych na poziomie bitowym, 651
    - na poziomie blokowym, 651
  - przestrzenne bazy danych, 1058
  - przestrzeń
    - adresowa, 638
    - nazw XML, 497
    - wektorowa, 1102
  - przeszukiwanie baz, 81
  - przetwarzanie
    - oparte na zdarzeniach, 491
    - plików, 44
    - potokowe, 749
    - równoległe, 751
  - strumieniowe, 749
  - tablicowe, 801
  - tekstu, 1109
  - transakcji, 809, 811
    - dostępnego
      - bezpośrednio, 1178
      - na bieżąco, 49, 93
    - wyników zapytań, 379, 385, 392
    - z materializacją, 748
    - zapytań, 57, 719
    - zapytań rozproszonych, 931
  - przewidywanie, 1144
  - przezroczystość, 915
  - przybliżone punkty kontrolne, 898
  - przyrostowa aktualizacja perspektyw, 773
  - pula bufora, 620
  - punkty
    - kontrolne, 887
    - przybliżone, 887
    - wejścia do bazy danych, 434, 448, 469
    - zatwierdzenia transakcji, 822
- ## Q
- QBE, Query-By-Example, 1251
  - agregacje, 1256
  - grupowanie, 1256
  - modyfikacje bazy danych, 1256
  - pobieranie danych, 1252
- ## R
- rachunek relacyjny, 293, 323
    - bezpieczeństwo wyrażeń, 333
    - dziedzin, 334
    - wyrażenia, 325
    - wzory, 325
    - zapytania, 328
  - RAID, 648
    - poprawianie wydajności, 651
    - poziomy, 651
    - równoległy dostęp do dysku, 648

- zwiększanie niezawodności, 650
- RAM, Random Access Memory, 605
- RDD, 1025
- RDF, Resource Description Framework, 505
- R-drzewa, 1062
- Redis, 971
- redukcja, 989
- referencje do obiektów, 431
- regresja, 1167
- reguła zapisu Thomasa, 862
- reguły, 59
  - aktywne, 1042
  - asocjacji przestrzennej, 1064
  - asocjacyjne, 1145, 1146
    - w hierarchiach, 1156
  - biznesowe, 58
  - heurystyczne, 723
  - kolokacji przestrzennej, 1064
  - wnioskowania, 564, 565
- rejestrowanie
  - w czasie odwoływania, 897
  - zapisów, 885
  - zapisów z wyprzedzeniem, 886
- rekordy, 608, 623
  - bieżące, 628
  - danych, 41
  - fantomowe, 875
  - niesegmentowane, 626
  - o stałej długości, 624
  - o zmiennej długości, 624
  - opisu, 390
  - polecenia, 390, 391
  - połączenia, 390
  - segmentowane, 626
  - środowiska, 390, 391
  - zaczepienia, 667
- rekurencyjne
  - operacje domknięcia, 316
  - relacje, 274
- relacje, 198, 200
  - czasu rzeczywistego, 1048
  - czasu transakcji, 1050
  - dwuczasowe, 1051
  - matematyczne, 199
  - o kluczu całościowym, 549, 551
  - odwołujące, 209
  - połączone, 266
  - między danymi, 57
  - uniwersalne, 529, 564, 573
  - wirtualne, 228
  - właściwości, 199
  - wskazywane, 209
  - wynikowe, 212
  - zagnieżdżone, 537
  - zakresowe, 324
- relacyjne
  - bazy danych
    - aktualizacje, 212
    - algorytmy projektowania, 563
    - dokumenty XML, 505
    - integralność encji, 209
    - modyfikowanie, 212
    - naruszenia
      - więzów integralności, 212
    - normalizacja, 517
    - ograniczenia, 210
    - operacja aktualizacji, 215
    - operacja usuwania, 214
    - operacja wstawiania, 212
    - projektowanie, 347
    - schematy, 206, 340
    - stan, 207
    - transakcje, 212, 216
    - więzy integralności, 207
    - wprowadzenie czasu, 1048
  - operacje unarne, 295
- relacyjny
  - model danych, 92, 94, 193, 195
  - notacja, 203
  - ograniczenia, 203
  - ograniczenia dziedziny, 204
  - ograniczenia klucza, 205
  - ograniczenia wartości pustych, 205
  - pojęcia, 196
  - rachunek dziedziny, 334
  - rachunek krotek, 323
- reorganizacja plików, 85
- replikacja, 918, 921, 957
  - nadrzędna-nadrzędna, 958
  - nadrzędna-podrzędna, 957
  - w systemie MongoDB, 965
- repozytoria danych, 86
- reprezentacja
  - czasu, 1046
  - wiedzy, 151, 173
- RFID, radio-frequency identification, 986
- roboty indeksujące, 1129
- rodzaje
  - awarii, 818
  - blokad, 848
  - indeksów, 677
  - pamięci masowej, 607
  - rozproszonych systemów, 937
  - zabezpieczeń, 1200
- role, 114, 199
- rozkład skośny, 742
- rozmieszczanie rekordów plików, 623
  - w blokach, 626
- rozproszone
  - bazy danych, DDB, 911, 913
  - aktualizacje, 934
  - alokacja danych, 918, 921
  - architektura
    - rozproszona, 940
  - architektura równoległa, 940
  - autonomia, 917
  - dekompozycja zapytań, 934
  - dostępność, 916
  - fragmentacja, 918
  - klasyfikacja, 938
  - odporność na podział, 917
  - odtwierzanie danych, 925
  - optymalizacja, 930
  - przetwarzanie zapytań, 930, 933
  - przezroczystość, 915
  - replikacja, 918, 921
  - rodzaje systemów, 937

## rozproszone

- rozproszone
  - odtworzenie danych, 927
  - rozproszone sterowanie
    - współbieżne, 925, 927
  - skalowalność, 917
  - stabilność, 916
  - sterowanie
    - współbieżne, 925
  - zalety, 917
  - zarządzanie katalogiem, 946
  - zarządzanie
    - transakcjami, 928
  - odtworzenie danych, 927
  - przetwarzanie zapytań, 933
  - sterowanie współbieżne, 925, 927
  - zakleszczenie, 925
  - zatwierdzenie, 925
- rozproszony
  - magazyn danych, 968
  - system obliczeniowy, 913
  - system zarządzania
    - bazami danych, 913
- rozstrzyganie kolizji, 638
- rozszerzalny język
  - znaczników, XML, 483
- rozszerzony model
  - danych, 1033
  - związków encji, 151–191
- równoległe
  - przetwarzanie zapytań, 750
  - systemy zarządzania, 940
- równoległość
  - na poziomie operatorów, 751
  - w jednym zapytaniu, 754
  - w wielu zapytaniach, 754
- równoważność, 564
  - perspektywiczna, 839
  - zbiorów, 569
- równo-złączenie, 307, 312
- różnica zbiorów, 243, 301, 302, 312

## S

- SAN, Storage Area Networks, 653
- scalanie
  - perspektyw, 771
  - relacji, 351
- scentralizowana
  - architektura, 88
- scentralizowane systemy
  - zarządzania, 87
- schemat, 70, 72, 110
  - bazy danych, 179, 461
  - EER, 166, 185, 466
  - ER, 363
  - fragmentacji, 920
  - konceptyjny, 78, 95
  - konceptyjny EER, 167
  - obiekty, 466
  - relacji, 198, 519
  - relacji uniwersalnej, 573
  - relacyjnej bazy danych, 203, 206, 340, 521
  - SQL, 226
  - wewnętrzny, 95
  - WI, 1101
  - XML, 494
  - zewnętrzny, 95
  - związków encji, 124
- sekcje krytyczne, 849
- sekwencje operacji, 299
- selekcja, 237, 295, 312, 729, 752
- selektywność, 734
  - złączenia, 309
- semantyka, 1106
  - relacji, 519
- serwer, 89
  - DataNode, 996
  - NameNode, 995
- serwery
  - aplikacji, 84
  - bazy danych, 84, 945
  - poczty elektronicznej, 88
  - SQL, 90, 945
  - transakcji, 90
  - WWW, 88, 91
  - zapytań, 90
- serwlety Javy, 419
- sesja logowania, 1204
- sharding, 918

- plików, 958
  - w systemie MongoDB, 966
- siatki typów, 439
- sieci
  - neuronowe, 1168
  - obszarów składowania
    - danych, SAN, 653
  - protokoły składowania
    - danych, 655
- sieć semantyczna, 178
- SIFT, scale-invariant feature transform, 1068
- skalowalność, 917, 957
  - JobTrakera, 1010
  - systemu HDFS, 997
- skierowany graf acykliczny, 721
- składowanie danych, 603
  - kolumnowe relacji, 707
  - mechanizmy, 604
  - relacji, 706
  - SAN, 653
  - urządzenia, 605
- słabe typy encji, 122
- słowo kluczowe, 81
  - !ATTLIST, 493
  - AFTER, 279
  - ALL, 242, 244
  - ANY, 261
  - AS, 245
  - ASC, 246
  - BEFORE, 279
  - CHECK, 277
  - CLUSTER, 704
  - CONSTRAINT, 235
  - DESC, 246
  - DISTINCT, 242, 299, 743
  - extent, 458
  - GRANT, 1209
  - having, 476
  - IN, 400
  - INOUT, 400
  - OUT, 400
  - PRIOR, 381
  - REF, 444
  - relationship, 457
  - SOME, 261
  - UNDER, 441, 446
  - UNIQUE, 704
  - where, 469
- SOAP, Simple Object Access Protocol, 504



- sortowanie, 246, 751
  - zewnątrzne, 632, 727, 728
- spamowanie, 1128
- specjalizacja, 154, 160, 163, 170, 176
  - częściowa, 170
  - rozłączna, 170
  - zdefiniowana przez atrybut, 158, 171
- specyficzne typy encji, 155
- specyfikacja, 179
- spójność
  - ostateczna, 957
  - pamięci podręcznej, 754
- SQL, Structured Query Language, 193, 223
  - aktywne bazy danych, 278
  - asercje, 277
  - deskryptory, 226
  - dodatkowe własności, 250
  - dopasowywanie
    - podciągów znaków, 244
  - dynamiczny, 374, 381
  - dziedziny wartości, 229
  - funkcje agregujące, 268
  - grupowanie, 270
  - identyfikator
    - uwierzytelniania, 226
  - implementacja
    - perspektyw, 282
  - jawne deklarowanie
    - zbiorów, 265
  - katalog, 226
  - logika trójwartościowa, 258
  - nazwy atrybutów, 239
  - nieokreślona klauzula WHERE, 241
  - obsługa formatu XML, 511
  - obsługa transakcji, 841
  - ograniczenia, 231
  - operacje arytmetyczne, 244
  - operacje porównania, 258
  - osadzanie poleceń, 374
  - perspektywy, 280
  - polecenia zmiany
    - schematu, 285
  - poziom izolacji, 841
  - proste zapytanie, 246
  - rozmiar obszaru
    - diagnostycznego, 841
  - rozszerzenia obiektowe, 440
  - schemat, 226
  - skomplikowane zapytania, 257
  - sortowanie, 246
  - tabele, 242
  - tabele połączone, 266
  - techniki programowania, 369
  - translacja zapytań, 724
  - tryb dostępu, 841
  - typy danych, 229
  - wskazówki, 801
  - wstrzykiwanie kodu, 1220
  - wyzwalacze, 251, 277, 278
  - zapytania, 275
    - rekurencyjne, 274
    - zagnieżdżone
      - skorelowane, 262
  - zarysy, 802
  - zarządzanie instrukcjami, 802
  - zbiory, 242
  - zmienianie nazw
    - atrybutów, 265
- SQL/CLI, 388, 390
- SQL/PSM, 398, 400
- SQL-99, 1044
  - wyzwalacze, 1044
- SQLJ, 374
  - osadzanie poleceń, 383
  - używanie iteratorów, 385
- SSD, solid-state drives, 616
- stabilność, 916
- stacja czołowa NAS, 654
- stan, 448
  - awaryjny, 820
  - bazy danych, 72, 824
  - relacji, 198
  - relacyjnej bazy danych, 207
  - spójny, 824
  - transakcji, 819
  - zakończenia, 820
  - zatwierdzenia, 820
- standard
  - CORBA, 460
  - JDBC, 90
  - ODMG, 72
  - SQL/CLI, 390
- statyczna strona, 488
- statyczne programowanie, 388
- statystyczne bazy danych, 1202
- stemming, 1109
- sterowanie współbieżne, 49, 84, 809, 815, 837, 847
  - izolacja snapshotów, 866, 868
  - operacja usuwania, 875
  - rekordy fantomowe, 875
  - rozproszone bazy danych, 925
  - techniki walidacyjne, 866
  - wielowersyjne, 862
  - znaczniki czasu, 860
- sterownik JDBC, 394
- stopień
  - homogeniczności, 937
  - lokalnej autonomii, 937
  - relacji, 198, 297, 298
  - typu związku, 115, 132
- stosowanie procesów
  - specjalizacji, 163
- strategie
  - programowania, 371
  - przetwarzania zapytań, 721
  - zastępowania buforów, 815
  - zastępowania danych
    - FIFO, 622
    - LRU, 622
    - MRU, 622
    - zegarowe, 622
- stratny projekt, 576
- stronicowanie
  - z przesłanianiem, 895, 896
- strony dynamiczne, 488
- struktury
  - B-drzewa, 685
  - bazy danych, 70
  - dostępowe, 665
  - drzewiaste, 160
  - hurtowni danych, 1180
  - indeksowe, 665
  - obiektów, 448
  - plikowe, 601, 603
  - schematu mieszania, 643



struktury  
   schematu mieszania  
     dynamicznego, 645  
   składowania, 603  
 suma  
   teoriomnogościowa, 733  
   zbiorów, 243, 301, 302,  
     312  
   zewnętrzna, 319  
 sygnatura, 433  
 symbol  
   ?, 493  
   apostrofu, 245  
   gwiazdki, 241, 492  
   pionowej kreski, 493  
   podkreślenia, 244  
   procenta, 244  
   wieloznacznym \*, 502  
   zastępczy, 245, 418  
 system, 819  
   Apache HBase, 972  
   Apache ZooKeeper, 975  
   big data, 64, 94, 955  
   BigTable, 972  
   DB/DC, 86  
   DynamoDB, 967  
   Hadoop, 983  
   HBase, 972  
   HDFS, 972, 993  
   Hive, 1006  
   Lucene, 1114  
   MongoDB, 962  
   Neo4j, 975  
   NOSQL, 64, 72  
   NOSQL z parami  
     klucz-wartość, 959  
   Pig, 1005  
   Spark, 1025  
   STARBURST, 1042  
   Voldemort, 968  
   WI, 1097  
 systematyka, 179  
 systemowe tabele blokad,  
   848  
 systemy  
   bazy danych, 40  
   jednoużytkownikowe, 812  
   operacyjne, 83  
   repozytorium danych, 86  
   słownika danych, 86  
   wieloużytkownikowe, 812

wielobazodanowe, 937  
 zarządzania bazami  
   danych  
   drzewiasty model  
     danych, 92  
   homogeniczne  
     rozproszone, 93  
   koszt, 93  
   moduły, 82  
   natywny XML-owy, 92  
   obiektowo-relacyjny, 92  
   obiektowy model  
     danych, 92  
   relacyjny model danych,  
     92  
   rozproszony, 93  
   rozszerzony system  
     relacyjny, 94  
   typy ścieżek dostępu, 93  
   zarządzania szeregami  
     czasowymi, 1057  
 szacowanie  
   kosztów, 723, 781  
   selektywności warunku,  
     733  
 SZBD, system zarządzania  
   bazą danych, 40 406, 499,  
   1017  
   podsystem menedżera  
     blokad, 849  
   strategie zastępowania  
     bufora, 823  
 szeregi czasowe, 1047, 1057,  
   1167  
 szeregowalność  
   perspektywiczna, 839  
 szyfrowanie danych, 1202,  
   1227  
   kluczem publicznym,  
     1228

## Ś

ścieżki w dokumentach, 500  
 środki kontroli, 1201  
 środowisko, 82  
   uruchomieniowe  
     MapReduce, 999  
   wytwarzania aplikacji, 66

## T

tabela zmodyfikowanych  
   stron, 898  
 tabele, 196, 242  
   bazowe, 228  
   blokad, 848  
   mieszające, 636  
   połączone, 266  
   wirtualne, *Patrz*  
     perspektywy  
 tablice  
   asocjacyjne, 410  
   liczbowe, 410  
   w PHP, 410  
 taśmowe urządzenia  
   pamięciowe, 617  
 taśmy magnetyczne, 607  
 technika SQL/PSM, 398  
 techniki  
   blokowania dwufazowego,  
     848  
   mieszania, 636  
   odtworzenia, 893  
   programowania, 367, 369  
   sterowania  
     współbieżnego, 847  
   wielowersyjnego  
     sterowania  
     współbieżnego, 862  
 technologia  
   Automated Storage  
     Tiering, 656  
   NAS, 654  
   RFID, 986  
 tekstura, 1067  
 test  
   postaci normalnej, 533  
   złączenia  
     nieaddytywnego, 547  
 testowanie dekompozycji  
   binarnych, 577  
 Tez, 1015  
 tezaury, 179, 1110  
 tożsamość obiektów, 429, 440  
 transakcje, 49, 216, 809–813,  
   819  
   aktywne, 886  
   harmonogramy, 825  
   interaktywne, 876  
   izolacja, 824

niepodzielność, 824  
 niewpływające na bazy danych, 890  
 pożądane właściwości, 824  
 trwałość, 824  
 w SQL, 841  
 wielu baz danych, 901  
 zachowanie spójności, 824  
 zapuszkowane, 51  
 zarządzanie w DDB, 928  
 translacja zapytań SQL, 724  
 trwałość obiektów, 432  
 trzecia postać normalna, 540  
   definicja ogólna, 544  
   interpretacja definicji, 544  
 trzeciorzędny mechanizm składowania, 604  
 TSQL2, 1055  
 twierdzenie CAP, 960  
 tworzenie  
   dokumentu XML, 510  
   egzemplarzy, 174  
   hierarchicznych perspektyw, 505  
   indeksu, 704  
   kopii bezpieczeństwa, 57, 84, 902  
   kopii zapasowej, 85  
   relacji związku, 352  
   schematu koncepcyjnego ER, 127  
   tabel, 444  
 typ danych, 42  
   DATE, 230, 1047  
   INTERVAL, 231  
 typ  
   trójskładnikowy, 115  
   UDT, 443  
   związku, 114, 171  
     1:1, 123  
     1:N, 123  
 typy  
   danych, 111, 229, 1095  
   binarne, 115  
   ciągi bitowe, 229  
   ciągi znaków, 229  
   logiczne, 230  
   numeryczne, 229  
   przestrzenne, 1059

znacznika czasowego, 230  
 elementów, 493  
 encji, 106, 109  
 identyfikujące, 122  
 prawidłowe, 122  
 silne, 122  
 słabe, 122  
 uogólnione, 170  
 właścicielskie, 122  
 informacji, 1143  
 rekordów, 623, 647  
 ujawnień, 1205  
 unii, 164, 361  
 uprawnień  
   dyspozycyjnych, 1207  
 użytkowników, 1095  
 wierszowe, 443  
 złożone, 498  
 związków, 114, 171  
 związków  
   trójskładnikowych, 133

## U

uchwyt rekordu, 390  
 uczenie  
   nienadzorowane, 1163  
   z nauczycielem, 1160  
 udoskonalanie  
   koncepcyjne, 163  
   z dołu do góry, 163  
   z góry na dół, 163  
 udostępnianie bazy danych, 40  
 UDT, 445  
   tworzenie tabel, 444  
 udział całkowity, 120  
 ukryte kanały, 1226  
 UML, Universal Modeling Language, 101, 128  
 UMLS, 1110  
 unikatowa tożsamość, 429  
 unikatowość, 58  
 unikatowy identyfikator obiektu, OID, 429  
 uogólniona projekcja, 314  
 uogólniony typ encji, 170  
 uporządkowanie logiczne, 676  
 uprawnienia, 1207  
   poziom konta, 1207  
   poziom relacji, 1207

urządzenia  
   NAS, 654  
   pamięciowe drugorzędne, 609  
   składowania danych, 605  
 usunięcie zagnieżdżenia, 538  
 usuwanie, 524  
 utrata  
   dostępności, 1200  
   integralności, 1200  
   poufności, 1200  
 uwierzytelnianie, 284  
 użytkownicy  
   dorywczy, 51  
   końcowi, 51  
   parametryczni, 51  
   samodzielni, 52  
 używanie systemów zarządzania, 64

## V

Voldemort, 968  
   mieszanie stabilne, 969  
   spójność, 971  
   wersjonowanie, 971  
 VPD, virtual private databases, 1234

## W

walidacja, 847, 866  
 walidacyjne sterowanie współbieżne, 866  
 warstwa  
   aplikacji, 91, 945  
   prezentacji, 944  
 wartości  
   niepodzielne, 535  
   prawdziwości atomów, 334  
   puste, 109, 201, 584  
 wartość NULL, 234, 258, 267, 525  
 warunek, 279  
   koniunktywny, 731  
   selekcji, 237, 295, 328, 335  
   zachowania atrybutów, 573  
   złączenia, 237, 328, 335

- wczesne wiązanie, 439
- wersje Hadoopa, 993
- wewnętrzne algorytmy sortowania, 727
- węzły
  - relacji, 329
  - stałe, 329
- WI, wyszukiwanie informacji, 1093
- wiarygodność
  - uwierzytelnienia, 1205
- wiązanie
  - parametrów polecenia, 391, 396
  - późne, 439
  - z językiem C++, 476
- wideo, 1066
- wiedza dedukcyjna, 1145
- wielodostępne
  - przetwarzanie transakcji, 49
- wielokrotne dziedziczenie, 162
- wieloprogramowe systemy operacyjne, 813
- wielowersyjne
  - blokowanie dwufazowe, 864
  - sterowanie współbieżne, 862
- wielozbiór, 431
- krótki, 298
- wierzchołki, 681
- nadrzędne, 681
- potomne, 681
- relacji, 760
- siostrzane, 694
- stałe, 760
- wewnętrzne, 681
- więzy
  - ekstensji, 437
  - integralności, 58, 207, 212
  - encji, 209
  - odwołań, 209, 593
- wirtualne prywatne bazy danych, VPD, 1234
- właściwości
  - dekompozycji, 564
  - dekompozycji relacyjnych, 573
  - klas, 174
  - pola indeksującego, 677
  - relacji, 199
  - rodzajów indeksów, 677
  - transakcji, 824
  - złączenia
    - nieaddytywnego, 579
- właściwość
  - niepodzielności, 824
  - zachowania zależności, 533
  - zachowania zależności dekompozycji, 574
  - złączenia bezstratnego, 533, 575, 582
- wnioskowanie, 59, 173
- WordNet, 1110
- wpis dziennika typu
  - odwołania, 885, 890
  - ponowienia, 885, 890
  - UNDO, 893
- WSDL, Web Services
  - Description Language, 504
- wskazówki, 801
- wskaźnik, 626
  - danych, 684, 688
  - drzewa, 684, 687
  - typu rekordu, 625, 639
- współbieżność, 49
- współczynnik
  - bloku, 626
  - liczności, 119
  - selekcji złączenia, 739
  - zapełnienia pliku, 646
- współdzielenie danych, 49
- współużytkowanie plików, 654
- wstawianie, 524
- wstępne przetwarzanie, 1126
- tekstu, 1109
- wstrzykiwanie
  - kodu, 1220–1223
  - wywołań funkcji, 1222
- wybiórcze dziedziczenie, 439
- wybór koniunktywny, 732
- wycofanie
  - kaskadowe, 861, 888
  - transakcji, 888
- wydajność
  - dysku, 615
  - operacji złączenia, 738
- wydobywanie informacji, 1112
- wyjątek, 384
- wykonawczy procesor
  - bazy danych, 721
- wykonywanie
  - polecenia, 392
  - punktów kontrolnych, 898
  - zapytań, 767
- wykrywanie
  - krawędzi, 1068
  - obiektów na obrazach, 1068
  - wzorców, 1127
  - zakleszczeń, 858
- wymagania
  - danych, 103
  - funkcjonalne, 103
- wymiana danych, 62
- wymiary czasu rzeczywistego, 1047
- wymuszanie stosowania standardów, 59
- wyniki zapytań, 470
- wyrażenia
  - algebry relacyjnej, 299
  - bezpieczne, 333
  - kolekcji, 474
  - relacyjnego rachunku, 325
  - ścieżkowe, 447, 470
  - wewnątrzwersyjne, 299
- wyrażenie FLWOR, 502
- wyróżniona kopia danych, 925
- wyszukiwanie, 646
  - binarne, 633, 730
  - dla selekcji, 731, 733
- informacji, WI, 1093, 1178
- adekwatność
  - tematyczna, 1115
- analizowanie treści, 1125
- czułość, 1116
- fasetowe, 1129
- model logiczny, 1101
- model oparty na przestrzeni wektorowej, 1102
- model probabilistyczny, 1104
- model semantyczny, 1106, 1100
- odpowiadanie na pytania, 1132

podejście statystyczne, 1100  
 precyzja, 1116  
 społecznościowe, 1130  
 średnia precyzja, 1118  
 trendy, 1129  
 tryby interakcji, 1099  
 typy zapytań, 1107  
 w dialogach, 1131  
 w sieci WWW, 1119  
 zapytania do  
   wyszukiwania fraz, 1108  
 zapytania logiczne, 1107  
 zapytania oparte na słowach kluczowych, 1107  
 zapytania w języku naturalnym, 1109  
 zapytania z określaniem odległości słów, 1108  
 zapytania z symbolami wieloznacznymi, 1108  
 liniowe, 627, 730  
 na podstawie alternatywy logicznej, 733  
 na podstawie warunku koniunktywnego, 731  
 oparte na indeksie, 731  
 w sieci WWW, 1099  
 według treści, 1065  
 z użyciem indeksu bitmapowego, 731  
 z użyciem indeksu funkcyjnego, 731  
 z użyciem zapytań zakresowych, 731  
 wytwarzanie aplikacji, 59  
 wywołania funkcji, 388, 402  
 wyzwacze, 59, 211, 251, 277, 278, 1035  
   w Oracle, 1035  
   w SQL-99, 1044  
 wzorce, 244  
   sekwencyjne, 1145  
   w ramach szeregów czasowych, 1146  
 wzór, 326

## X

XML, Extensible Markup Language, 62, 483, 1218  
   adnotacje, 497  
   atrybuty kompozytowe, 499  
   definiowanie elementów, 492  
   dobrze uformowane dokumenty, 491  
   dokumenty bez schematu, 490  
   elementy  
     pierwszopoziomowe, 498  
   hierarchia dokumentu, 509  
   hierarchiczne  
     perspektywy, 505  
   klucze, 498  
   opis schematu, 497  
   plik schematu, 491  
   pobieranie dokumentów, 505  
   prawidłowe dokumenty, 491  
   przestrzeń nazw, 497  
   schematy, 494  
   tworzeniem dokumentu, 510  
   typy, 497  
   zapisywanie dokumentów, 499  
   złożone typy, 498  
 XML DTD, 491  
 XML/SQL, 511  
 XPath, 500  
   separatory, 501  
   ścieżki w dokumentach, 500  
 XQuery, 500  
   definiowanie zapytań, 502  
 XSL, Extensible Stylesheet Language, 504  
 XSLT, Extensible Stylesheet Language for Transformations, 504

## Y

YARN, Yet Another Resource Negotiator, 1009  
   architektura platformy, 1012  
   platforma usług, 1022

## Z

zabezpieczenia, 1200  
   oparte na etykietach, 1233, 1234  
   oparte na rolach, 1212  
   wielopoziomowe, 1212  
 zachowanie zależności dekompozycji, 574  
 zaczepienie bloku, 667  
 zagłodzenie, 859  
 zagnieżdżanie konstruktorów typów, 430  
 zakleszczenie, 856  
 zależności  
   częściowe, 539  
   dekompozycji, 574  
   funkcyjne, 528  
   nietrywialne, 566  
   oparte na funkcjach, 593  
   oparte na procedurach arytmetycznych, 593  
   pokrycie minimalne, 571  
   reguły wnioskowania, 565, 588  
   równoważność zbiorów, 569  
   trywialne, 566  
   zbiory minimalne, 570  
 istnienia, 120  
 przechodnie, 540  
 wielowartościowe, 549, 588  
 zawierania, 592  
 złączeniowe, 552, 592  
 zamiana grafów w drzewa, 509  
 zapewnianie elastyczności, 61  
 zapobieganie zakleszczeniom, 857

- zapytania, 40
  - bazowe, 275
  - czasowe, 1055
  - dekompozycja, 934
  - do wyszukiwania fraz, 1108
  - dotyczące wielu relacji, 787
  - drzewa, 758
  - eliminowanie
    - zagnieżdżenia, 770
  - etapy przetwarzania, 722
  - heurystyki optymalizacji, 758
  - interaktywne, 83
  - interpretowane, 775
  - języka SQL, 236
  - kompilowane, 775
  - koszty, 776
  - logiczne, 1107
  - nierekurencyjne, 1081
  - o najbliższego sąsiada, 1061
  - oparte na słowach
    - kluczowych, 81, 1107
  - optymalizacja, 719, 757
    - adaptacyjna, 800
    - fizyczna, 789
    - kosztowa, 792
    - w bazach Oracle, 799
    - w hurtowniach danych, 796
  - optymalizator
    - fizyczny, 799
    - globalny, 799
  - OQL, 469
  - pobierające dane, 418
  - przestrzenne, 1058–1061
  - przetwarzanie, 719
    - równoległe, 750
    - tablicowe, 801
  - przez przykład, QBE, 1251
  - rekurencyjne, 274
  - scalanie, 770
  - selekcji-projekcji-
    - złączania, 329
  - składowane, 1178
  - sposoby wykonywania, 767
  - statystyczne, 1224
  - strategie przetwarzania, 721
  - szacowanie wielkości
    - wyników, 795
  - w algebrze relacyjnej, 320
  - w czasie wykonywania programu, 381
  - w hurtowniach danych, 796
  - w języku naturalnym, 1109
  - w relacyjnym rachunku krotek, 328
  - w XML, 502
  - wskazówki indeksowania, 707
  - wyświetlanie planu wykonania, 794
  - z określaniem odległości słów, 1108
  - z symbolami
    - wieloznacznymi, 1108
  - zagnieżdżone, 260, 768
  - zagnieżdżone
    - skorelowane, 262
  - zakresowe, 731, 1061
  - zapis planu, 796
- zarysy, 802
- zarządca aplikacji, 1013, 1014
- zarządzanie
  - buforem, 620
  - danymi przestrzennymi, 1065
  - federacyjnymi systemami
    - baz danych, 938
  - ograniczeniami, 706
  - replikami, 996, 997
  - rozproszonym katalogiem, 946
  - transakcjami, 928, 930
- zastępowanie danych w buforze, 622
- zastosowania
  - aktywnych baz danych, 1044
  - baz danych, 60
  - danych przestrzennych, 1065
  - postaci normalnych, 533
  - operacji relacyjnych, 1081
  - wczesne, 60
  - współczesne, 63
- zatrzaski, 876
- zbiory, 242
  - danych RDD, 1025
  - encji, 106, 109
  - kompletne, 309
  - minimalne, 570
  - odczytu, 815
  - par, 200
  - potęgowe, 535
  - wartości, 109
  - wartości atrybutów, 111
  - zapisu, 815
  - związków, 114, 115
- zdarzenia, 279, 1038
  - punktowe, 1047
  - trwające, 1047
- zdefiniowane reguły, 59
- zgodność złączenia, 301
- ziarnistość, 847
  - elementów danych, 869
- złączanie, 752
  - mieszające hybrydowe, 742
  - mieszające równoległe
    - partycjonowane, 753
  - nierównościowe, 752
  - oparte na pakietach, 1003
  - partycjonowane
    - równościowe, 752
  - sortująco-scalające, 1002
  - z użyciem mieszania, 1003
  - z użyciem partycji, 1003
- złączenia, 305, 734
  - po stronie mappera, 1003
  - w modelu MapReduce, 1002
  - wydajność operacji, 738, 739
- złączenie
  - bezzstratne, 575
  - częściowe, 725, 933
  - dwukierunkowe, 734
  - krzyżowe, 303
  - mieszające
    - hybrydowe, 741
    - partycjonowane, 736, 741
  - naturalne, 266, 307, 312
  - nieaddytywne, 577, 579, 582
  - pętli zagnieżdżonych, 735
  - sortująco-scalające, 735
  - theta, 307, 312

wewnętrzne, 267, 309,  
318  
wielokierunkowe, 734  
z pętlą pojedynczą, 735  
z udziałem wielu tabel,  
267  
zewnątrzne, 267, 317, 746  
lewe, 318  
pełne, 319, 748  
prawo, 319  
złożone elementy, 489  
zmienianie nazw atrybutów,  
265, 300  
zmienna  
iterująca, 469  
SQLCODE, 377, 378  
SQLSTATE, 377

zmienne  
dziedziny, 334  
dzielone, 375  
instancji, 427  
komunikacji, 376  
krotek, , 237, 239, 324,  
327  
znaczniki, 485, 486, 487  
czasowe, 847, 860  
odczytu, 863  
transakcji, 857  
zapisu, 863  
znak ucieczki, 245, *Patrz także*  
symbol  
zunifikowany język  
modelowania, UML,  
101

związek  
IS-AN, 153  
JEST, 153, 170  
JEST-CZĘŚCIĄ, 176  
JEST-PODKLASĄ, 176  
JEST-POWIĄZANY, 176  
klasy-podklasy, 153  
nadklasy-podklasy, 170  
związki  
binarne, 132, 133  
pomiędzy encjami, 71  
rekurencyjne, 117, 118  
trójskładnikowe, 132–136  
ograniczenia, 136  
w postaci atrybutów, 116

## Ż

źródło danych, 394, 483



# PROGRAM PARTNERSKI

— GRUPY HELION —

- 
- A top-down view of four hands of different skin tones holding four interlocking puzzle pieces. The pieces are arranged in a 2x2 square. The top-left piece is red, the top-right is dark blue, the bottom-left is light blue, and the bottom-right is a slightly darker shade of light blue. The hands are positioned at the corners of the square, with fingers gripping the edges of the puzzle pieces.
1. ZAREJESTRUJ SIĘ
  2. PREZENTUJ KSIĄŻKI
  3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

**Dowiedz się więcej i dołącz już dzisiaj!**

<http://program-partnerski.helion.pl>

GRUPA  
**Helion** 



## KOMPLEKSOWO SZKOLIMY NOWOCZESNY BIZNES



IT



BIZNES



PROJEKTY



PROCESY

NASZE SZKOLENIA SĄ PROWADZONE  
ZGODNIE Z METODĄ

# BLENDED LEARNING

modelem kształcenia, który łączy tradycyjne szkolenie  
z dostępem do nowoczesnych narzędzi - wideokursów,  
e-booków i audiobooków

T: 609 850 372 E: SZKOLENIA@HELION.PL  
**WWW.HELIONSZKOLENIA.PL**