



**MATT GOLDWASSER
UPOM MALIK
BENJAMIN JOHNSTON**

SQL

ANALIZA DANYCH ZA POMOCĄ ZAPYTAŃ

WYDANIE II

WARSZTATY PRAKTYCZNE

Packt

Helion

NAUKA O DANYCH I SZTUCZNA INTELIGENCJA

Tytuł oryginału: The Applied SQL Data Analytics Workshop: Develop your practical skills and prepare to become a professional data analyst, 2nd Edition

Tłumaczenie: Tomasz Walczak

ISBN: 978-83-283-8475-0

Copyright © Packt Publishing 2020. First published in the English language under the title 'The Applied SQL Data Analytics Workshop - Second Edition – (9781800203679)'.

Polish edition copyright © 2022 by Helion S.A.
All rights reserved.

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 jakiegokolwiek 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 wydawca 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 wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <https://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/sqlan2_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](#)

Spis treści

Wprowadzenie	9
Rozdział 1. Wprowadzenie do SQL-a dla analityków	33
Wprowadzenie	33
Świat danych	34
Rodzaje danych	34
Analityka danych i statystyka	35
Rodzaje statystyki	36
Zadanie 1.01 — klasyfikowanie nowego zbioru danych	37
Metody z obszaru statystyki opisowej	37
Rozkład danych	38
Ćwiczenie 1.01 — tworzenie histogramu	38
Ćwiczenie 1.02 — obliczanie kwartyli dla sprzedaży dodatków	43
Tendencja centralna	46
Ćwiczenie 1.03 — obliczanie miar tendencji centralnej dla sprzedaży dodatków	47
Dyspersja	48
Ćwiczenie 1.04 — obliczanie dyspersji dla sprzedaży dodatków	49
Analiza dwuczynnikowa	50
Wykresy punktowe	50
Ćwiczenie 1.05 — obliczanie współczynnika korelacji Pearsona dla dwóch zmiennych	56
Zadanie 1.02 — eksplorowanie danych sprzedażowych z salonu samochodowego	61
Praca z niepełnymi danymi	61
Testy istotności statystycznej	62
Często używane testy istotności statystycznej	63
Relacyjne bazy danych i SQL	64
Wady i zalety baz SQL-owych	64
Podstawowe typy danych w SQL-u	66
Typy liczbowe	66
Typy znakowe	66
Typ logiczny	67
Daty i godziny	67
Struktury danych — format JSON i tablice	68

Wczytywanie tabel — kwerenda SELECT	68
Podstawowa budowa i działanie kwerendy SELECT	68
Podstawowe słowa kluczowe w kwerendach SELECT	69
Ćwiczenie 1.06 — kwerenda SELECT z podstawowymi słowami kluczowymi dotycząca tabeli salespeople	76
Zadanie 1.03 — kwerenda SELECT z podstawowymi słowami kluczowymi dotycząca tabeli customers	77
Tworzenie tabel	78
Tworzenie pustych tabel	78
Ćwiczenie 1.07 — tworzenie tabeli w SQL-u	79
Tworzenie tabel za pomocą kwerendy SELECT	80
Aktualizowanie tabel	81
Dodawanie i usuwanie kolumn	81
Dodawanie nowych danych	81
Aktualizowanie istniejących wierszy	83
Ćwiczenie 1.08 — aktualizowanie tabeli w celu podniesienia ceny pojazdu	84
Usuwanie danych i tabel	85
Usuwanie wartości z wiersza	85
Usuwanie wierszy z tabeli	86
Usuwanie tabel	86
Ćwiczenie 1.09 — usuwanie niepotrzebnej tabeli	87
Zadanie 1.04 — tworzenie i modyfikowanie tabel na potrzeby działań marketingowych	87
SQL i analityka	88
Podsumowanie	89
Rozdział 2. Przygotowywanie danych za pomocą SQL-a	90
Wprowadzenie	90
Łączenie danych	91
Łączenie tabel za pomocą słowa kluczowego JOIN	91
Rodzaje złączeń	93
Ćwiczenie 2.01 — używanie złączeń do analizy sprzedaży w salonach	102
Podkwerendy	103
Sumy	104
Ćwiczenie 2.02 — generowanie listy gości na przyjęcie dla klientów VIP za pomocą klauzuli UNION	105
Wyrażenia WITH	107
Przekształcanie danych	108
Funkcja CASE WHEN	108
Ćwiczenie 2.03 — używanie funkcji CASE WHEN do pobierania list klientów z danego regionu	109
Funkcja COALESCE	111
Funkcja NULLIF	112
Funkcje LEAST i GREATEST	113
Funkcja CASTING	114
Funkcje DISTINCT i DISTINCT ON	115
Zadanie 2.01 — używanie SQL-a do tworzenia modelu wspomagającego sprzedaż	118
Podsumowanie	119

Rozdział 3. Agregacja i funkcje okna	120
Wprowadzenie	120
Funkcje agregujące	120
Ćwiczenie 3.01 — używanie funkcji agregujących do analizowania danych	123
Funkcje agregujące z klauzulą GROUP BY	124
Klauzula GROUP BY	124
Klauzula GROUP BY dla kilku kolumn	129
Ćwiczenie 3.02 — obliczanie cen dla typów produktów za pomocą klauzuli GROUP BY	130
Klauzula GROUPING SETS	131
Funkcje agregujące dla zbiorów uporządkowanych	132
Klauzula HAVING	133
Ćwiczenie 3.03 — obliczanie wyników i wyświetlanie danych z użyciem klauzuli HAVING	134
Stosowanie funkcji agregujących do oczyszczania danych i sprawdzania ich jakości	135
Znajdowanie brakujących wartości za pomocą klauzuli GROUP BY	135
Pomiar jakości danych za pomocą funkcji agregujących	137
Zadanie 3.01 — analizowanie danych sprzedażowych z użyciem funkcji agregujących	138
Funkcje okna	139
Podstawy funkcji okna	140
Ćwiczenie 3.04 — analizowanie zmian współczynnika podawania danych przez klientów w czasie	144
Słowo kluczowe WINDOW	146
Obliczanie statystyk z użyciem funkcji okna	147
Ćwiczenie 3.05 — określanie pozycji na podstawie daty zatrudnienia	148
Ramka okna	149
Ćwiczenie 3.06 — motywowanie pracowników lunchem	151
Zadanie 3.02 — analizowanie sprzedaży z wykorzystaniem ramek okna i funkcji okna	153
Podsumowanie	154
Rozdział 4. Importowanie i eksportowanie danych	155
Wprowadzenie	155
Polecenie COPY	156
Kopiowanie danych za pomocą narzędzia psql	157
Konfigurowanie poleceń COPY i \copy	159
Użycie poleceń COPY i \copy do masowego wczytywania danych do bazy	160
Ćwiczenie 4.01 — eksportowanie danych do pliku w celu dalszego przetwarzania ich w Excelu	161
Zastosowanie języka R do bazy danych	165
Po co korzystać z języka R?	165
Wprowadzenie do języka R	165
Zastosowanie języka Python do bazy danych	168
Po co korzystać z języka Python?	168
Wprowadzenie do języka Python	168
Ułatwianie dostępu do baz PostgreSQL w Pythonie za pomocą narzędzi SQLAlchemy i pandas	171
Czym jest SQLAlchemy?	172

Używanie Pythona w narzędziu Jupyter Notebook	172
Pobieranie danych z bazy i ich zapisywanie w bazie za pomocą pakietu pandas	174
Ćwiczenie 4.02 — wczytywanie i wizualizowanie danych w Pythonie	175
Zapisywanie danych w bazie za pomocą Pythona	177
Zwiększanie szybkości zapisu w Pythonie za pomocą polecenia COPY	178
Odczyt i zapis plików CSV w Pythonie	179
Najlepsze praktyki z obszaru importowania i eksportowania danych	181
Pomijanie podawania hasła	181
Zadanie 4.01 — używanie zewnętrznego zbioru danych do wykrywania trendów sprzedażowych	182
Podsumowanie	183
Rozdział 5. Analityka z wykorzystaniem złożonych typów danych	184
Wprowadzenie	184
Wykorzystywanie typów danych z datami i czasem do analiz	185
Wprowadzenie do typu date	185
Przekształcanie typów danych	188
Przedziały	189
Ćwiczenie 5.01 — analiza danych z szeregów czasowych	191
Przeprowadzanie analiz geoprzestrzennych w PostgreSQL	192
Długość i szerokość geograficzna	193
Reprezentowanie długości i szerokości geograficznej w PostgreSQL	193
Ćwiczenie 5.02 — analizy geoprzestrzenne	195
Stosowanie tablicowych typów danych w PostgreSQL	197
Wprowadzenie do tablic	197
Ćwiczenie 5.03 — analizowanie sekwencji z użyciem tablic	200
Stosowanie formatu JSON w PostgreSQL	201
JSONB — wstępnie przetworzone dane w formacie JSON	203
Dostęp do danych z pól w formacie JSON lub JSONB	204
Stosowanie języka JSONPath do pól w formacie JSONB	206
Tworzenie i modyfikowanie danych w polu w formacie JSONB	208
Ćwiczenie 5.04 — przeszukiwanie obiektów JSONB	209
Analiza tekstu za pomocą PostgreSQL	210
Tokenizacja tekstu	211
Ćwiczenie 5.05 — analizowanie tekstu	212
Wyszukiwanie tekstu	216
Optymalizowanie wyszukiwania tekstu w PostgreSQL	218
Zadanie 5.01 — wyszukiwanie i analiza transakcji sprzedaży	220
Podsumowanie	221
Rozdział 6. Wydajny SQL	222
Wprowadzenie	222
Metody skanowania baz danych	224
Plany wykonywania kwerend	224
Skanowanie sekwencyjne i inne metody skanowania	224
Ćwiczenie 6.01 — interpretowanie działania planera kwerend	226
Zadanie 6.01 — plany wykonywania kwerendy	229
Skanowanie indeksu	230
Indeks w postaci B-drzewa	231

Ćwiczenie 6.02 — kwerenda ze skanowaniem indeksu	233
Zadanie 6.02 — skanowanie indeksu	237
Indeks z haszowaniem	238
Ćwiczenie 6.03 — tworzenie kilku indeksów z haszowaniem, aby zbadać ich wydajność	239
Zadanie 6.03 — stosowanie indeksów z haszowaniem	243
Skuteczne korzystanie z indeksów	244
Wydajne złączenia	245
Ćwiczenie 6.04 — ocenianie zastosowania złączeń wewnętrznych	246
Zadanie 6.04 — stosowanie wydajnych złączeń	252
Funkcje i wyzwalacze	253
Definicje funkcji	253
Ćwiczenie 6.05 — tworzenie funkcji, które nie przyjmują argumentów	254
Zadanie 6.05 — definiowanie funkcji zwracającej maksymalną wartość sprzedaży	257
Ćwiczenie 6.06 — tworzenie funkcji przyjmujących argumenty	258
Polecenia \df i \sf	259
Zadanie 6.06 — tworzenie funkcji przyjmujących argumenty	260
Wyzwalacze	260
Ćwiczenie 6.07 — tworzenie wyzwalaczy do aktualizowania pól	263
Zadanie 6.07 — tworzenie wyzwalacza do śledzenia średniej liczby kupionych sztuk	267
Kończenie pracy kwerend	268
Ćwiczenie 6.08 — anulowanie długo działającej kwerendy	269
Zadanie 6.08 — kończenie długo działającej kwerendy	270
Podsumowanie	271
Rozdział 7. Metoda naukowa i rozwiązywanie problemów w praktyce	272
Wprowadzenie	272
Studium przypadku	273
Metoda naukowa	273
Ćwiczenie 7.01 — wstępne zbieranie danych za pomocą technik SQL-a	274
Ćwiczenie 7.02 — pobieranie informacji sprzedażowych	276
Zadanie 7.01 — ilościowa ocena spadku sprzedaży	280
Ćwiczenie 7.03 — analiza czasu rozpoczęcia sprzedaży	281
Zadanie 7.02 — analiza hipotezy dotyczącej różnicy w cenie sprzedaży	288
Ćwiczenie 7.04 — analiza zależności wzrostu sprzedaży od współczynnika otwarć e-maili	290
Ćwiczenie 7.05 — analiza skuteczności e-mailowej kampanii marketingowej	297
Wnioski	300
Badania terenowe	300
Podsumowanie	301
Dodatek	303
Skorowidz	345

Wprowadzenie

O książce

Firmy działają 24 godziny na dobę siedem dni w tygodniu i szybko generują przy tym olbrzymie ilości danych. W tych danych ukryte są istotne wzorce i zachowania, które mogą pomóc Tobie i Twojej organizacji dogłębnie zrozumieć klientów. Czy jesteś gotów wkroczyć w eksycytujący świat analityki danych i zapewnić sobie dostęp do takich przydatnych wniosków?

Książka SQL. Analiza danych za pomocą zapytań. Warsztaty praktyczne została napisana przez zespół ekspertów z dziedziny data science, którzy z wykorzystaniem umiejętności z obszaru analizy danych przekształcali firmy o różnej postaci i wielkości. Pozycja ta jest świetnym sposobem na rozpoczęcie przygody z analizą danych. Zobaczysz tu, jak skutecznie przesiewać i uzyskiwać informacje z surowych danych — nawet jeśli nie masz jeszcze doświadczenia w tej dziedzinie.

Książka rozpoczyna się od wyjaśnienia, jak formułować hipotezy i generować opisowe statystyki, które mogą zapewnić ważne informacje na temat istniejących danych. Z dalszych rozdziałów dowiesz się, jak pisać w SQL-u kwerendy, aby agregować, obliczać i łączyć dane SQL-owe ze źródeł spoza bieżącego zbioru danych. Zobaczysz też, jak pracować z danymi w różnych formatach, na przykład w formacie JSON. Dzięki zapoznaniu się z zaawansowanymi technikami takimi jak analiza geoprzestrzenna i analiza tekstu będziesz w stanie zrozumieć swoją firmę na głębszym poziomie. W tej książce poznasz też sekret szybszego i skuteczniejszego pozyskiwania informacji za pomocą zaawansowanych metod takich jak profilowanie i automatyzacja.

Gdy ukończysz lekturę książki *SQL. Analiza danych za pomocą zapytań. Warsztaty praktyczne*, będziesz posiadać umiejętności potrzebne do tego, aby zacząć wykrywać wzorce i wyciągać wnioski z własnych danych. Nauczysz się tu badać i oceniać dane krytycznym okiem wyszkolonego analityka danych.

Odbiorcy

Jeśli jesteś inżynierem baz danych, który chce zająć się analityką, lub znasz podstawy SQL-a, ale nie wiesz, jak wykorzystać ten język do generowania wniosków biznesowych, ta książka jest dla Ciebie odpowiednia.

O rozdziałach

Rozdział 1., „Wprowadzenie do SQL-a dla analityków”, pomoże Ci poznać podstawy analityki danych i SQL-a. Dowiesz się, jak za pomocą technik matematycznych i graficznych analizować dane w Excelu. Dalej omawiam rolę SQL-a w świecie danych i pokazuję, jak używać tego języka do prostego operowania danymi w relacyjnych bazach danych.

W rozdziale 2., „Przygotowywanie danych za pomocą SQL-a”, wyjaśniam, jak przy użyciu SQL-a oczyszczać dane i przygotowywać je do analiz. Najpierw nauczysz się łączyć tabele i zapytania w zbiory danych. Następnie przejdziesz do bardziej zaawansowanych zagadnień.

W rozdziale 3., „Agregacja i funkcje okna”, omawiam funkcje agregujące i funkcje okna z SQL-a. Są to wartościowe techniki podsumowywania danych. Nauczysz się stosować je do uzyskiwania wglądu w dane i poznawania cech zbiorów danych (na przykład jakości danych).

Rozdział 4., „Importowanie i eksportowanie danych”, zapewni Ci umiejętności potrzebne do interakcji z bazami danych za pomocą innych narzędzi, takich jak Excel oraz języki R i Python.

Rozdział 5., „Analityka z wykorzystaniem złożonych typów danych”, pomoże Ci dobrze poznać różne typy danych dostępne w SQL-u oraz dowiedzieć się, jak uzyskiwać informacje na podstawie danych obejmujących daty i czas, danych geoprzestrzennych, tablic, danych w formacie JSON oraz tekstu.

Rozdział 6., „Wydażny SQL”, pomoże Ci zoptymalizować kwerendy, aby działały szybciej. Dowiesz się, jak analizować wydajność kwerend, a także jak dodawać różne mechanizmy SQL-a, takie jak funkcje i wyzwalacze, które wzbogacają podstawowe możliwości tego języka.

W rozdziale 7., „Metoda naukowa i rozwiązywanie problemów w praktyce”, zobaczysz, jak wykorzystać zdobyte umiejętności do rozwiązywania praktycznych problemów wykraczających poza scenariusze opisane w tej książce. Posługując się metodą naukową i krytycznym myśleniem, będziesz analizować dane i przekształcać je w informacje umożliwiające podejmowanie działań.

Konwencje stosowane w książce

Fragmenty kodu w tekście, nazwy tabel baz danych i dane wprowadzane przez użytkownika są zapisywane czcionką o stałej szerokości. Oto przykład: „Trzy z tych kolumn, Rok Urodzenia, Wzrost i Liczba Wizyt Doktora, są ilościowe, ponieważ są reprezentowane za pomocą liczb”. Nazwy katalogów, nazwy plików, rozszerzenia plików, ścieżki, fikcyjne adresy URL i identyfikatory z Twittera są zapisywane kursywą, na przykład: „Otwórz plik *adresy.txt* z podkatalogu *dane*”.

Słowa widoczne na ekranie (na przykład w menu lub oknach dialogowych) też są wyróżnione kursywą. Oto przykład: „Wybierz opcję *Rozdzielany* w oknie dialogowym *Kreator importu tekstu* i upewnij się, że importowanie rozpoczyna się od wiersza *1*”.

Bloki kodu mają następującą postać:

```
SELECT *
FROM products
WHERE production_end_date IS NULL;
```

Nowe pojęcia i ważne słowa są wyróżnione pogrubieniem: „Statystykę można podzielić na dwie podkategorie: **statystykę opisową** i **wnioskowanie statystyczne**”.

Przygotowywanie środowiska

Przed rozpoczęciem dokładnej lektury tej książki należy przygotować określone oprogramowanie i narzędzia. W następnym punkcie zobaczysz, jak to zrobić.

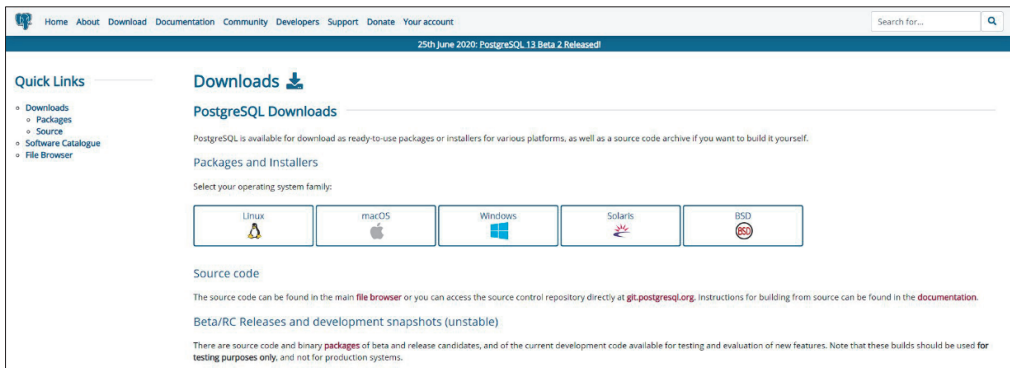
Instalowanie systemu PostgreSQL 12

W dalszych punktach znajdziesz instrukcje instalowania i konfigurowania systemu PostgreSQL 12 w systemach operacyjnych Windows, Linux i macOS.

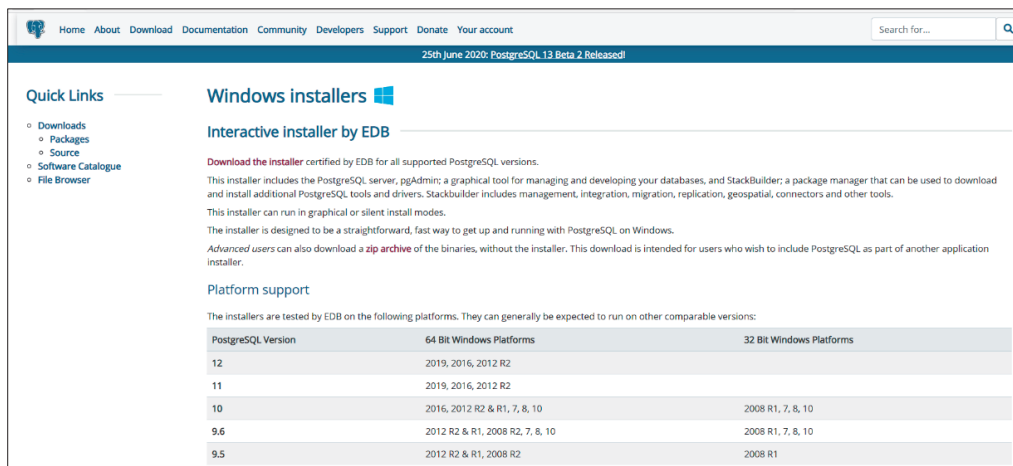
Pobieranie i instalowanie systemu PostgreSQL dla systemu Windows

Najpierw pobierz i zainstaluj system PostgreSQL dla systemu Windows. W tym celu wykonaj następujące kroki:

1. Wejdź na stronę <https://www.postgresql.org/download/> (rysunek 0.1). Wybierz opcję *Windows* z listy *Packages and Installers*.



Rysunek 0.1. Strona pobierania systemu PostgreSQL

2. Wybierz opcję *Download the Installer* (rysunek 0.2).


Home About Download Documentation Community Developers Support Donate Your account

25th June 2020: PostgreSQL 13 Beta 2 Released!

Quick Links

- Downloads
 - Packages
 - Source
 - Software Catalogue
 - File Browser

Windows installers

Interactive installer by EDB

Download the installer certified by EDB for all supported PostgreSQL versions.

This installer includes the PostgreSQL server, pgAdmin; a graphical tool for managing and developing your databases, and StackBuilder; a package manager that can be used to download and install additional PostgreSQL tools and drivers. Stackbuilder includes management, integration, migration, replication, geospatial, connectors and other tools.

This installer can run in graphical or silent install modes.

The installer is designed to be a straightforward, fast way to get up and running with PostgreSQL on Windows.

Advanced users can also download a **zip archive** of the binaries, without the installer. This download is intended for users who wish to include PostgreSQL as part of another application installer.

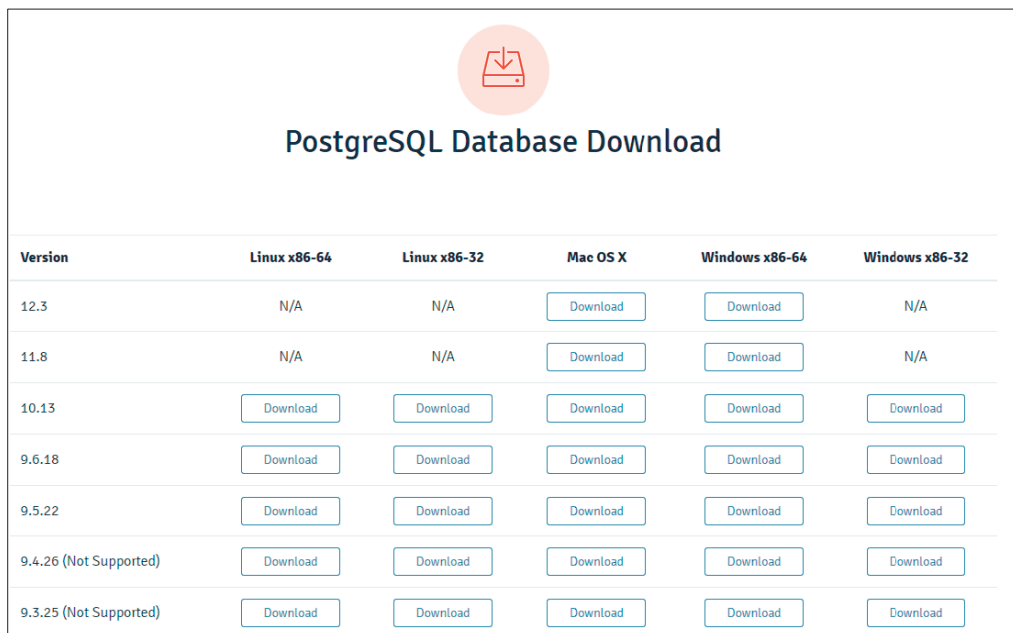
Platform support

The installers are tested by EDB on the following platforms. They can generally be expected to run on other comparable versions:

PostgreSQL Version	64 Bit Windows Platforms	32 Bit Windows Platforms
12	2019, 2016, 2012 R2	
11	2019, 2016, 2012 R2	
10	2016, 2012 R2 & R1, 7, 8, 10	2008 R1, 7, 8, 10
9.6	2012 R2 & R1, 2008 R2, 7, 8, 10	2008 R1, 7, 8, 10
9.5	2012 R2 & R1, 2008 R2	2008 R1

Rysunek 0.2. Pobieranie interaktywnego instalatora systemu PostgreSQL

3. Wybierz wersję 12.x (rysunek 0.3), ponieważ to ona będzie używana w tej książce.

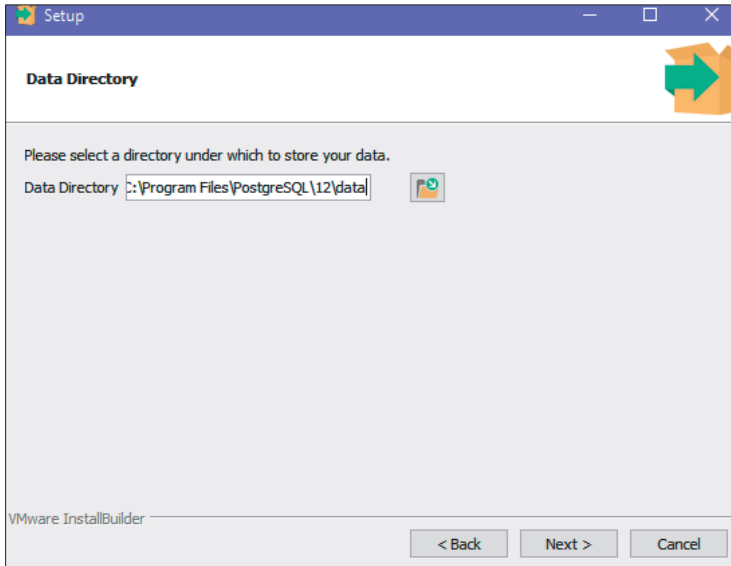


PostgreSQL Database Download

Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
12.3	N/A	N/A	Download	Download	N/A
11.8	N/A	N/A	Download	Download	N/A
10.13	Download	Download	Download	Download	Download
9.6.18	Download	Download	Download	Download	Download
9.5.22	Download	Download	Download	Download	Download
9.4.26 (Not Supported)	Download	Download	Download	Download	Download
9.3.25 (Not Supported)	Download	Download	Download	Download	Download

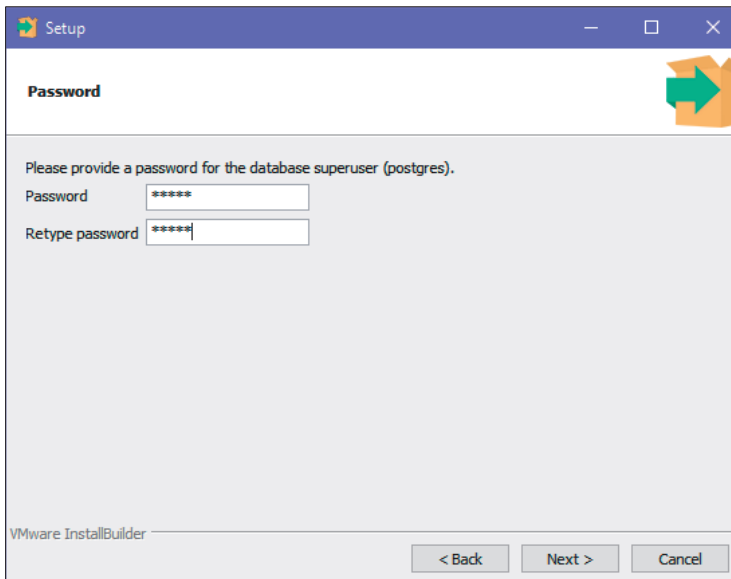
Rysunek 0.3. Strona pobierania systemu PostgreSQL

4. W większości kroków instalacji kliknij przycisk *Next*. Gdy pojawi się prośba o wskazanie katalogu na dane (rysunek 0.4), warto podać ścieżkę, którą można łatwo zapamiętać.



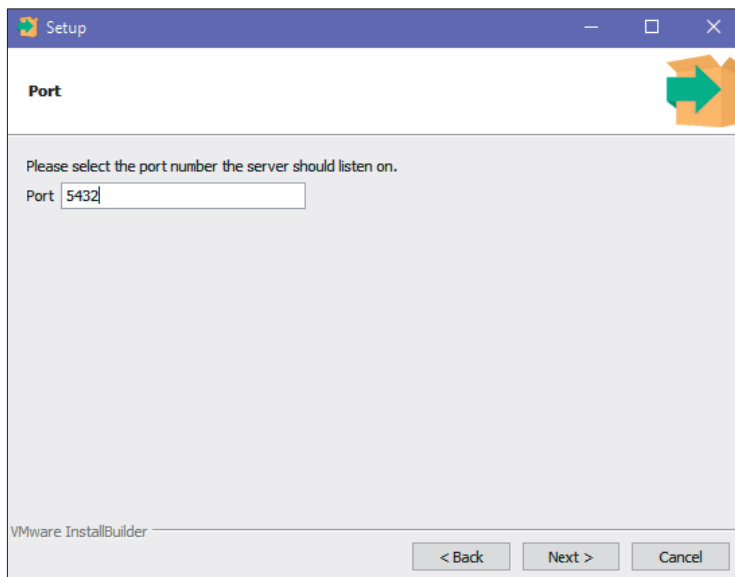
Rysunek 0.4. Instalowanie systemu PostgreSQL w systemie Windows

5. Podaj hasło dla administratora postgres (rysunek 0.5).



Rysunek 0.5. Ustawianie hasła administratora

6. Nie zmieniaj domyślnie używanego numeru portu (rysunek 0.6), chyba że powoduje on konflikt z aplikacją, która już jest zainstalowana w systemie.



Rysunek 0.6. Konfigurowanie portu systemu PostgreSQL

7. Kliknij przycisk *Next* w pozostałych krokach i poczekaj na ukończenie instalacji.

Konfigurowanie zmiennej Path

Aby się upewnić, że zmienna Path została poprawnie ustawiona, otwórz wiersz poleceń, wpisz lub wklej poniższą instrukcję, a następnie wciśnij klawisz *Enter*:

```
psql -U postgres
```

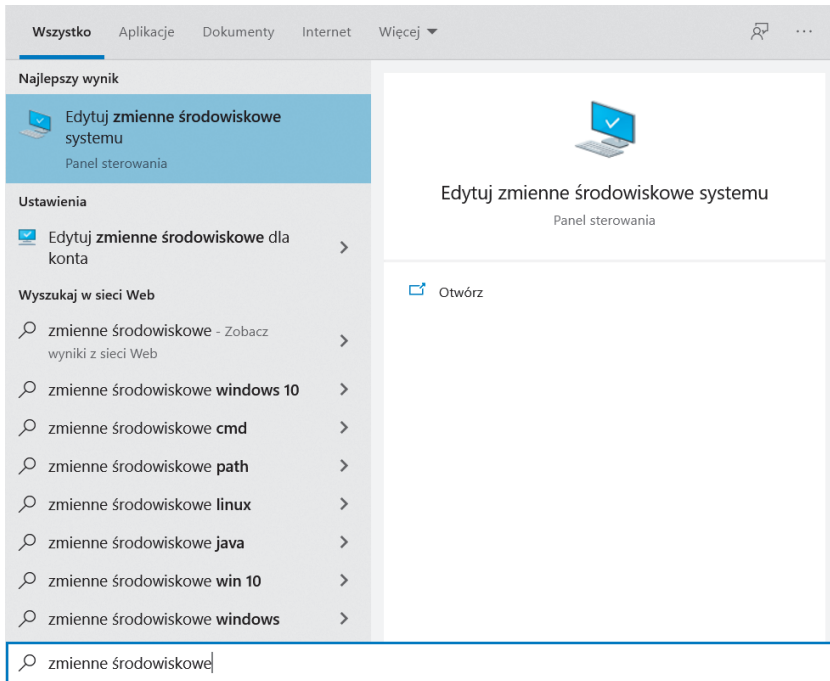
Jeśli pojawi się błąd z rysunku 0.7, musisz dodać katalog z binariami systemu PostgreSQL do zmiennej Path.

```
C:\Users\abhis>psql
'psql' is not recognized as an internal or external command,
operable program or batch file.
```

Rysunek 0.7. Błąd spowodowany nieskonfigurowaną zmienną Path

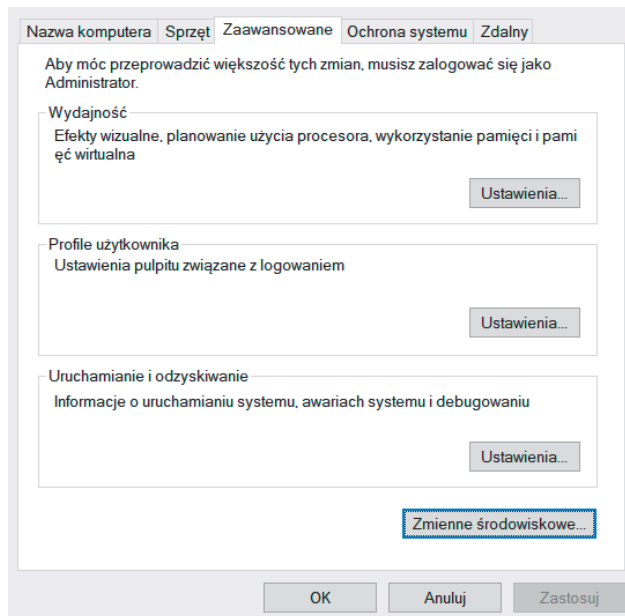
W tym celu wykonaj następujące kroki:

1. Wpisz nazwę zmiennej środowiskowej w polu wyszukiwania w systemie Windows (rysunek 0.8).



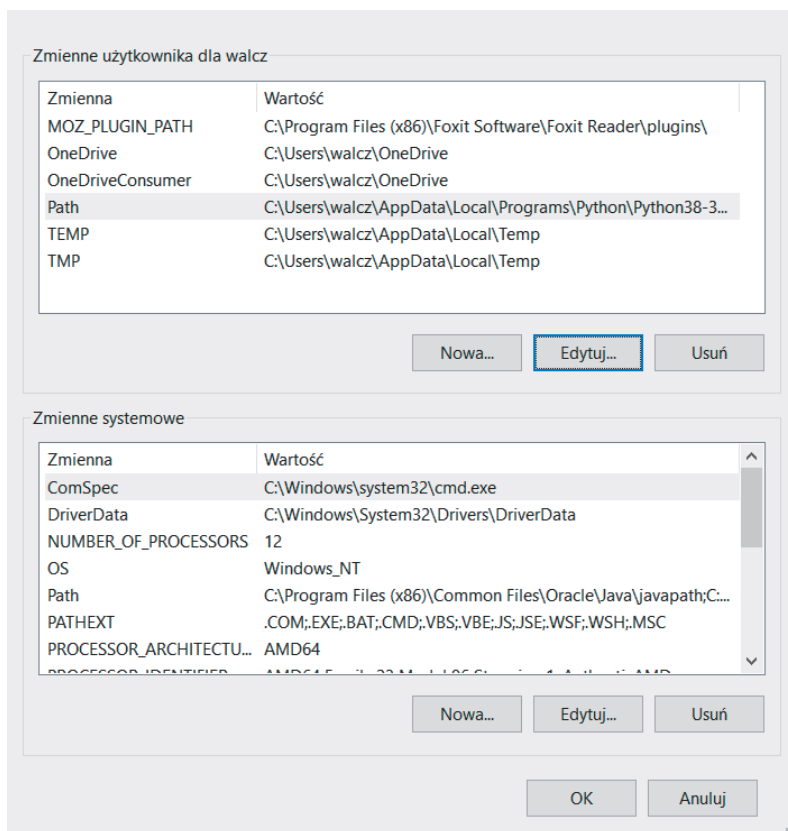
Rysunek 0.8. Wyszukiwanie nazwy zmienne środowiskowe w systemie Windows

2. Wybierz opcję *Zmienne środowiskowe* (rysunek 0.9).



Rysunek 0.9. Właściwości systemu Windows

3. Zaznacz zmienną Path i kliknij przycisk *Edycja* (rysunek 0.10).

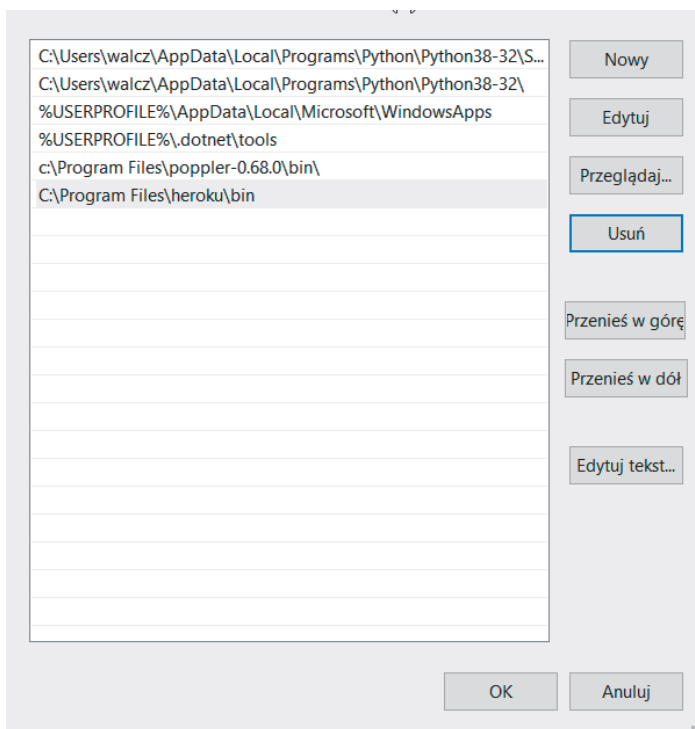


Rysunek 0.10. Konfigurowanie zmiennej Path

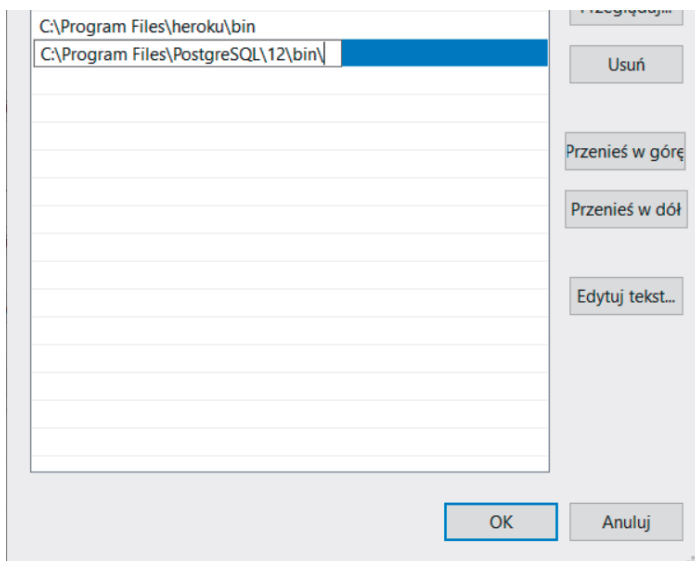
4. Kliknij przycisk *Nowy* (rysunek 0.11).
5. W eksploratorze plików znajdź katalog, w którym zainstalowany jest system PostgreSQL. Dodaj ścieżkę do katalogu *bin* z instalacji systemu PostgreSQL (rysunek 0.12).
- Kliknij *OK* i zrestartuj system.
6. Teraz otwórz wiersz poleceń i wpisz lub wklej poniższą instrukcję. Wciśnij klawisz *Enter*, aby ją wykonać:

```
psql -U postgres
```

Wpisz hasło, jakie skonfigurowałeś w kroku 5. w punkcie „Pobieranie i instalowanie systemu PostgreSQL dla systemu Windows”. Następnie wciśnij klawisz *Enter*. Powinieneś móc zalogować się do konsoli systemu PostgreSQL (rysunek 0.13).



Rysunek 0.11. Konfigurowanie zmiennej Path



Rysunek 0.12. Podawanie ścieżki

```
C:\Users\abhis>psql -U postgres
Password for user postgres:
psql (12.3)
WARNING: Console code page (850) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=#
```

Rysunek 0.13. Powłoka systemu PostgreSQL

7. Wpisz `\q` i wciśnij klawisz *Enter*, aby wyjść z powłoki (rysunek 0.14).

```
sql># \q
C:\Users\abhis>
```

Rysunek 0.14. Wychodzenie z powłoki systemu PostgreSQL

Instalowanie systemu PostgreSQL w systemie Linux

Poniższe kroki pomogą Ci zainstalować system PostgreSQL w systemie Ubuntu lub w systemach bazujących na dystrybucji Debian:

1. Otwórz terminal. Następnie wpisz lub wklej poniższe polecenie w nowym wierszu i wciśnij klawisz *Enter*:

```
sudo apt-get install postgresql-12
```

2. Po instalacji system PostgreSQL utworzy użytkownika `postgres`. Aby utworzyć powłokę systemu PostgreSQL, musisz się zalogować jako ten użytkownik:

```
sudo su postgres
```

Powinieneś zobaczyć zmianę wiersza zachęty w terminalu (rysunek 0.15).

```
abhishek@abhishek-VM: ~$ sudo su postgres
postgres@abhishek-VM: /home/abhishek$
```

Rysunek 0.15. Dostęp do powłoki systemu PostgreSQL w systemie Linux

3. Gdy wpiszesz poniższą instrukcję, przejdziesz do powłoki systemu PostgreSQL:

```
psql
```

Możesz wpisać `\l` (lewy ukośnik i mała litera L), aby wyświetlić listę wszystkich domyślnie wczytywanych baz danych (rysunek 0.16).

```

abhishek@abhishek-VM: ~
postgres@abhishek-VM:/home/abhishek$ psql
psql (12.2 (Ubuntu 12.2-4))
Type "help" for help.

postgres=# \l

               List of databases
  Name      | Owner   | Encoding | Collate | Ctype   | Access privileges
-----+-----+-----+-----+-----+-----
 postgres   | postgres | UTF8     | en_IN   | en_IN   | =c/postgres
 template0   | postgres | UTF8     | en_IN   | en_IN   | postgres=CtC/postgres
 template1   | postgres | UTF8     | en_IN   | en_IN   | =c/postgres
              postgres=CtC/postgres
(3 rows)

postgres=#

```

Rysunek 0.16. Lista baz danych w systemie Linux

Omówiliśmy tu instalację systemu PostgreSQL w Ubuntu i systemach bazujących na dystrybucji Debian. Instrukcje dotyczące instalacji w innych dystrybucjach znajdziesz w ich dokumentacjach. Strona pobierania systemu PostgreSQL dla systemów linuxowych to <https://www.postgresql.org/download/linux/>.

Instalowanie systemu PostgreSQL w systemie macOS

W tym punkcie dowiesz się, jak zainstalować system PostgreSQL w systemie macOS. Zanim zaczniesz instalować oprogramowanie, sprawdź, czy w systemie zainstalowany jest menedżer pakietów Homebrew. Jeśli nie, wejdź na stronę <https://brew.sh/>, podany tam skrypt wklej w terminalu systemu macOS i wciśnij klawisz *return*. Reaguj na wyświetlane instrukcje i poczekaj, aż skrypt zakończy instalację (rysunek 0.17).

Przedstawione instrukcje są dostosowane do systemu Catalina macOS w wersji 10.15.6 (w czasie, gdy powstaje ta książka, jest to najnowsza wersja). Więcej informacji o używaniu terminala znajdziesz na stronie <https://support.apple.com/en-in/guide/terminal/apd5265185d-f365-44cb-8b09-71a064a42125/mac>.

Po zainstalowaniu menedżera Homebrew wykonaj następujące kroki, aby zainstalować system PostgreSQL:

1. Otwórz nowe okno terminala. Aby zainstalować pakiet PostgreSQL, wpisz po kolei trzy poniższe instrukcje i po każdej z nich wciśnij klawisz *return*:

```

brew doctor
brew update
brew install postgres

```

Poczekaj na zakończenie instalacji. W zależności od konfiguracji i szybkości połączenia możesz zobaczyć komunikaty podobne do tych z rysunku 0.18 (zauważ, że tu widoczny jest tylko fragment zapisów dotyczących instalacji).



Rysunek 0.17. Instalowanie menedżera pakietów Homebrew

```
--> Updated Formulae
angular-cli      earthly          hasura-cli       oha
aspectj          elasticsearch@8 haxe              onscripter
aws-okta         ethereum        igv               openjdk@11
awscli@1         feedgnuplot     imgproxy         patchutils
buildifier       flyway          jsdoc3            prometheus
buildozer        fork-cleaner    kibana@8         pueue
check            fortio          libfabric         pulumi
citus            freerdp         libunp           pumba
clib             frugal          metabase          pyenv
cmake            gitlab-runner   mill              re2c
convoy           goreleaser      mpd               reorder-python-imports
copilot          grin            nifi-registry    ruby-install
doctl            groovy          node@10           slacknimate
dscanner         groovysdk       numpy             terraform

--> Deleted Formulae
sflowtool

--> Downloading https://homebrew.bintray.com/bottles/postgresql-12.3.4.catalina.bottle.tar.gz
Already downloaded: /Users/maahedev/Library/Caches/Homebrew/downloads/c1c03224d5c338848eae489487602745780257ded
ec241c55df4cd59dd08e731--postgresql-12.3.4.catalina.bottle.tar.gz
--> Pouring postgresql-12.3.4.catalina.bottle.tar.gz
--> Caveats

To migrate existing data from a previous major version of PostgreSQL run:
  brew postgresql-upgrade-database

To have launchd start postgresql now and restart at login:
  brew services start postgresql

Or, if you don't want/need a background service you can just run:
  pg_ctl -D /usr/local/var/postgres start
--> Summary
🍺 /usr/local/Cellar/postgresql/12.3.4: 3,220 files, 37.8MB

- took 36s
```

Rysunek 0.18. Informacje o postępie instalacji systemu PostgreSQL (fragment)

2. Po zakończeniu instalacji uruchom proces PostgreSQL. W tym celu wpisz następującą instrukcję w terminalu i wciśnij klawisz *return*:

```
pg_ctl -D /usr/local/var/postgres start
```

Powinieneś zobaczyć dane wyjściowe podobne do tych z rysunku 0.19.

```
➔ pg_ctl -D /usr/local/var/postgres start
waiting for server to start....2020-07-22 21:53:38.064 IST [35123] LOG:  starting PostgreSQL 12.3 on x86_64-apple-darwin19.4.0, compiled by Apple clang version 11.0.3 (clang-1103.0.32.59), 64-bit
2020-07-22 21:53:38.066 IST [35123] LOG:  listening on IPv6 address ":::1", port 5432
2020-07-22 21:53:38.066 IST [35123] LOG:  listening on IPv4 address "127.0.0.1", port 5432
2020-07-22 21:53:38.068 IST [35123] LOG:  listening on Unix socket "/tmp/.s.PGSQL.5432"
2020-07-22 21:53:38.070 IST [35124] LOG:  database system was shut down at 2020-07-22 21:25:45 IST
2020-07-22 21:53:38.074 IST [35123] LOG:  database system is ready to accept connections
done
server started
```

Rysunek 0.19. Uruchamianie procesu PostgreSQL

3. Po uruchomieniu procesu zaloguj się do powłoki systemu PostgreSQL. Użyj domyślnego konta administratora postgres (wciśnij klawisz *return*, aby wykonać instrukcję):

```
psql postgres
```

Możesz wpisać \l (lewy ukośnik i mała litera L) i wcisnąć klawisz *return*, aby wyświetlić listę wszystkich domyślnie wczytywanych baz danych (rysunek 0.20).

```
➔ psql postgres
psql (12.3)
Type "help" for help.

postgres=# \l

               List of databases
  Name      | Owner      | Encoding | Collate | Ctype | Access privileges
-----+-----+-----+-----+-----+-----
maahedev    | maahedev   | UTF8     | C       | C     | 
postgres    | maahedev   | UTF8     | C       | C     | 
template0   | maahedev   | UTF8     | C       | C     | =c/maahedev      +
            |            |          |         |       | maahedev=CtC/maahedev
template1   | maahedev   | UTF8     | C       | C     | =c/maahedev      +
            |            |          |         |       | maahedev=CtC/maahedev
(4 rows)

postgres=#
```

Rysunek 0.20. Lista domyślnie wczytywanych baz danych

Wpisz `\q` i wciśnij klawisz *return*, aby wyjść z powłoki systemu PostgreSQL.

Razem z systemem PostgreSQL 12 automatycznie instalowane jest narzędzie pgAdmin.

Instalowanie Pythona

Instalowanie Pythona w systemie Windows

1. Znajdź potrzebną wersję Pythona na oficjalnej stronie z pakietami instalacyjnymi <https://www.anaconda.com/distribution/#windows>.
2. Na stronie pobierania wybierz wersję Python 3.7.
3. Zainstaluj wersję odpowiednią dla architektury systemu komputerowego (32- lub 64-bitową). Informacje o architekturze znajdziesz w oknie *System i zabezpieczenia/System* w systemie operacyjnym.
4. Po pobraniu pliku z pakietem instalacyjnym kliknij go dwukrotnie i reaguj na instrukcje wyświetlane na ekranie.

Instalowanie Pythona w systemie Linux

Istnieje kilka wygodnych sposobów instalowania Pythona w systemie Linux:

1. Otwórz wiersz poleceń i sprawdź, czy Python 3 nie jest już zainstalowany w systemie. W tym celu uruchom instrukcję `python3 --version`.
2. Aby zainstalować Pythona, uruchom następujące polecenie:

```
sudo apt-get update
sudo apt-get install python3.7
```
3. Jeśli napotkasz problemy, w internecie dostępnych jest wiele źródeł, które pomogą Ci się z nimi uporać.
4. Możesz też zainstalować Pythona, pobierając pakiet instalacyjny Anaconda dla Linuksa ze strony <https://www.anaconda.com/distribution/#linux> i stosując się do instrukcji.

Instalowanie Pythona w systemie macOS

Podobnie jak w Linuksie, także w systemie macOS istnieje kilka dobrych metod instalowania Pythona. Wykonaj następujące kroki:

1. Otwórz terminal systemu macOS. W tym celu wciśnij kombinację *CMD + spacja*, wpisz `terminal` w oknie wyszukiwania i wciśnij *return*.
2. Zainstaluj Xcode z poziomu wiersza poleceń, uruchamiając polecenie `xcode-select --install`.

3. Najłatwiejszy sposób instalacji Pythona 3 polega na użyciu menedżera Homebrew. Narzędzie to można zainstalować z poziomu wiersza poleceń za pomocą instrukcji ruby `-e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"`.
4. Dodaj narzędzie Homebrew do zmiennej środowiskowej \$PATH. Otwórz profil użytkownika w wierszu poleceń za pomocą instrukcji `sudo nano ~/.profile`. Następnie wstaw `export PATH="/usr/local/opt/python/libexec/bin:$PATH"` na końcu.
5. Ostatni krok polega na zainstalowaniu Pythona. W wierszu poleceń uruchom instrukcję `brew install python`.
6. Możesz też zainstalować Pythona za pomocą instalatora Anaconda dostępnego na stronie <https://www.anaconda.com/distribution/#macos>.

Instalowanie systemu Git

Instalowanie systemu Git w systemach Windows i macOS X

Git dla systemów Windows i macOS można pobrać i zainstalować ze strony <https://git-scm.com/>. Jednak wygodniejsze jest zainstalowanie go za pomocą zaawansowanego klienta, takiego jak GitKraken (<https://www.gitkraken.com/>).

Instalowanie systemu Git w systemie Linux

System Git można łatwo zainstalować z poziomu wiersza poleceń:

```
sudo apt-get install git
```

Jeśli preferujesz graficzny interfejs użytkownika, narzędzie GitKraken (<https://www.gitkraken.com/>) jest dostępne także dla systemu Linux.

Pobieranie przykładowych zbiorów danych dla systemu Windows

W większości ćwiczeń z tej książki używana jest przykładowa baza danych sqllda, która zawiera sformatowane dane dotyczące fikcyjnej firmy ZoomZoom produkującej samochody elektryczne. Przygotuj tę bazę, wykonując następujące kroki:

1. Najpierw utwórz bazę danych sqllda. Otwórz wiersz poleceń i wpisz lub wklej poniższą instrukcję. Następnie wciśnij klawisz *Enter*, aby ją wykonać:

```
createdb -U postgres sqllda
```

Pojawi się prośba o podanie hasła, które skonfigurowałeś dla administratora postgres w trakcie instalacji (rysunek 0.21).

```
C:\Users\abhis\Documents\Packt Work\Applied SQL>createdb -U postgres sqlda
Password:
```

Rysunek 0.21. Prośba o podanie hasła w powłoce systemu PostgreSQL

Aby sprawdzić, czy baza danych została poprawnie utworzona, zaloguj się do powłoki. W tym celu wpisz lub wklej poniższą instrukcję i wciśnij klawisz *Enter*:

```
psql -U postgres
```

Gdy pojawi się prośba o podanie hasła, wpisz je. Wciśnij klawisz *Enter*, aby kontynuować.

Wpisz \l (lewy ukośnik i małą literę L), a następnie wciśnij klawisz *Enter*, by sprawdzić, czy baza danych została utworzona. Baza sqlda powinna się pojawić razem z listą domyślnych baz (rysunek 0.22).

```
postgres=# \l
```

Name	Owner	Encoding	Collate	Ctype	Access privileges
postgres	postgres	UTF8	English_India.1252	English_India.1252	
sqlda	postgres	UTF8	English_India.1252	English_India.1252	
template0	postgres	UTF8	English_India.1252	English_India.1252	=c/postgres +
					postgres=CTc/postgres
template1	postgres	UTF8	English_India.1252	English_India.1252	=c/postgres +
					postgres=CTc/postgres

(4 rows)

Rysunek 0.22. Lista baz z systemu PostgreSQL

Pobierz plik *data.dump* z katalogu *Datasets* z repozytorium tej książki w serwisie GitHub (<https://packt.live/30Uhcfl>); wszystkie pliki są dostępne także w witrynie wydawnictwa Helion. Zmodyfikuj wyróżniony fragment ścieżki w poniższym poleceniu zgodnie z lokalizacją pliku w systemie. Wpisz lub wklej tę instrukcję w wierszu poleceń i wciśnij klawisz *Enter*, aby ją wykonać:

```
psql -U postgres -d sqlda -f C:\<ścieżka>\data.dump
```

Inna możliwość to przejście w wierszu poleceń za pomocą instrukcji *cd* do lokalnego katalogu, do którego pobrałeś omawiany plik. Na przykład jeśli pobrałeś go do katalogu *Downloads*, możesz przejść do niego za pomocą instrukcji *cd C:\Users\<nazwa użytkownika>\Downloads*. Jeśli tak zrobisz, usuń wyróżniony przedrostek ze ścieżką z tego kroku. Polecenie powinno wyglądać wtedy tak: `psql -U postgres -d sqlda -f data.dump`.

Powinieneś zobaczyć dane wyjściowe podobne do tych z rysunku 0.23.

Sprawdź, czy baza danych została poprawnie wczytana. Zaloguj się do konsoli systemu PostgreSQL, wpisując lub wklejając poniższą instrukcję. Wciśnij klawisz *Enter*, aby ją wykonać.

```
psql -U postgres
```

```

C:\Users\abhis\Documents\Packt Work\Applied SQL>psql -U postgres -d sqlda -f data.dump
Password for user postgres:
SET
SET
SET
SET
SET
set_config
-----
(1 row)

SET
SET
SET
SET
CREATE EXTENSION
COMMENT
CREATE EXTENSION
COMMENT
CREATE TEXT SEARCH DICTIONARY
SET
SET
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE MATERIALIZED VIEW
CREATE TABLE
CREATE MATERIALIZED VIEW
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
COPY 44533
COPY 0
COPY 50000
COPY 32
COPY 50000
COPY 20
COPY 418158
COPY 12
COPY 15412
COPY 37711
COPY 300
COPY 20
ALTER TABLE
ALTER TABLE
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
REVOKE
GRANT
REFRESH MATERIALIZED VIEW
REFRESH MATERIALIZED VIEW

C:\Users\abhis\Documents\Packt Work\Applied SQL>

```

Rysunek 0.23. Importowanie baz danych systemu PostgreSQL

W powłoce wpisz następujące polecenie, aby nawiązać połączenie z bazą sql`da`:

```
\c sqlda
```

Teraz wpisz `\dt`. Ta instrukcja powinna wyświetlić wszystkie tabele z bazy danych (rysunek 0.24).

```
C:\Users\abhis>psql -U postgres
Password for user postgres:
psql (12.3)
WARNING: Console code page (850) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=# \c sqlda
You are now connected to database "sqlda" as user "postgres".
sqlda=# \dt
```

Schema	Name	Type	Owner
public	closest_dealerships	table	postgres
public	countries	table	postgres
public	customer_sales	table	postgres
public	customer_survey	table	postgres
public	customers	table	postgres
public	dealerships	table	postgres
public	emails	table	postgres
public	products	table	postgres
public	public_transportation_by_zip	table	postgres
public	sales	table	postgres
public	salespeople	table	postgres
public	top_cities_data	table	postgres

```
(12 rows)

sqlda=#
```

Rysunek 0.24. Sprawdzanie, czy baza danych została zaimportowana

Baza danych jest importowana z poziomu konta administratora postgres wyłącznie w celach demonstracyjnych. W środowisku produkcyjnym zalecane jest użycie odrębnego konta.

Pobieranie przykładowych zbiorów danych dla systemu Linux

W większości ćwiczeń z tej książki używana jest przykładowa baza danych sql`da`, która zawiera spreparowane dane dotyczące fikcyjnej firmy ZoomZoom produkującej samochody elektryczne. Przygotuj tę bazę, wykonując następujące kroki:

1. Przejdź do konta użytkownika postgres, wpisując poniższą instrukcję w terminalu. Wciśnij klawisz `Enter`, aby ją wykonać:

```
sudo su postgres
```

Powinieneś zobaczyć w powłoce zmianę widoczną na rysunku 0.25.

```
abhishek@abhishek-VM:~$ sudo su postgres
postgres@abhishek-VM:/home/abhishek$
```

Rysunek 0.25. Wczytywanie przykładowych zbiorów danych w systemie Linux

2. Wpisz lub wklej poniższe polecenie, aby utworzyć nową bazę sqla.
Wciśnij klawisz *Enter*, by je wykonać.

```
createdb sqla
```

Następnie możesz wpisać polecenie `psql`, aby przejść do powłoki systemu PostgreSQL, i polecenie `\l` (lewy ukośnik i małą literę L) w celu sprawdzenia, czy baza została poprawnie utworzona (rysunek 0.26).

```
postgres@abhishek-VM:/home/abhishek$ createdb sqla
postgres@abhishek-VM:/home/abhishek$ psql
psql (12.2 (Ubuntu 12.2-4))
Type "help" for help.

postgres=# \l

          List of databases
  Name      | Owner   | Encoding | Collate | Ctype   | Access privileges
-----+-----+-----+-----+-----+-----
 postgres  | postgres | UTF8     | en_IN   | en_IN   |
 sqla      | postgres | UTF8     | en_IN   | en_IN   |
 template0 | postgres | UTF8     | en_IN   | en_IN   | =c/postgres +
           |          |          |          |          | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_IN   | en_IN   | =c/postgres +
           |          |          |          |          | postgres=CTc/postgres
(4 rows)

postgres=#
```

Rysunek 0.26. Dostęp do powłoki systemu PostgreSQL w systemie Linux

Wpisz `\q`, a następnie wciśnij klawisz *Enter*, by wyjść z powłoki systemu PostgreSQL.

3. Pobierz plik `data.dump` z katalogu *Datasets* z repozytorium tej książki w serwisie GitHub (<https://packt.live/30Uhcfl>). Za pomocą instrukcji `cd` przejdź do katalogu z pobranym plikiem. Następnie wpisz poniższe polecenie:

```
psql -d sqla data.dump
```

4. Teraz poczekaj na zaimportowanie zbioru danych (rysunek 0.27).
5. Aby sprawdzić, czy zbiór danych został poprawnie zaimportowany, najpierw wpisz `psql`, a potem wciśnij klawisz *Enter*. Przejdiesz w ten sposób do powłoki systemu PostgreSQL. Teraz uruchom polecenie `\c sqla`, a następnie `\dt`, aby zobaczyć listę tabel z bazy danych (rysunek 0.28).

```

postgres@abhishek-VM:/home/abhishek/Downloads$ psql -U postgres -d sqlda < data.dump
SET
SET
SET
SET
SET
set_config
-----
(1 row)

SET
SET
SET
SET
CREATE EXTENSION
COMMENT
CREATE EXTENSION
COMMENT
CREATE TEXT SEARCH DICTIONARY
SET
SET
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE MATERIALIZED VIEW
CREATE TABLE
CREATE MATERIALIZED VIEW
CREATE TABLE

```

Rysunek 0.27. Importowanie zbioru danych w systemie Linux

```

postgres=# \c sqlda
You are now connected to database "sqlda" as user "postgres".
sqlda=# \dt

```

Schema	Name	Type	Owner
public	closest_dealerships	table	postgres
public	countries	table	postgres
public	customer_sales	table	postgres
public	customer_survey	table	postgres
public	customers	table	postgres
public	dealerships	table	postgres
public	emails	table	postgres
public	products	table	postgres
public	public_transportation_by_zip	table	postgres
public	sales	table	postgres
public	salespeople	table	postgres
public	top_cities_data	table	postgres

```

(12 rows)

sqlda=#

```

Rysunek 0.28. Sprawdzanie poprawności zaimportowanych danych w Linuksie

Baza danych jest importowana z poziomu konta administratora postgres wyłącznie w celach demonstracyjnych. W środowisku produkcyjnym zalecane jest użycie odrębnego konta.

Pobieranie przykładowych zbiorów danych dla systemu macOS

W większości ćwiczeń z tej książki używana jest przykładowa baza danych `sqlda`, która zawiera spreparowane dane dotyczące fikcyjnej firmy ZoomZoom produkującej samochody elektryczne. Przygotuj tę bazę, wykonując następujące kroki:

1. Przejdź do powłoki systemu PostgreSQL, wpisując poniższe polecenie w terminalu. Wciśnij klawisz *return*, by je wykonać:


```
psql postgres
```
2. Teraz utwórz nową bazę danych `sqlda`. Wpisz poniższą instrukcję (nie zapomnij o średniku na końcu) i wciśnij klawisz *return*:


```
create database sqlda;
```
3. Powinieneś zobaczyć dane wyjściowe pokazane na rysunku 0.29. Wpisz `\l` (lewy ukośnik z małą literą L) w terminalu i wciśnij klawisz *return*, by sprawdzić, czy baza danych została poprawnie utworzona (powinieneś zobaczyć na liście bazę `sqlda`).

```
postgres=# create database sqlda;
CREATE DATABASE
postgres=# \l
```

Name	Owner	Encoding	Collate	Ctype	Access privileges
maahedev	maahedev	UTF8	C	C	
postgres	maahedev	UTF8	C	C	
sqlda	maahedev	UTF8	C	C	
template0	maahedev	UTF8	C	C	=c/maahedev +
					maahedev=CTc/maahedev
template1	maahedev	UTF8	C	C	=c/maahedev +
					maahedev=CTc/maahedev

```
(5 rows)

postgres=#
```

Rysunek 0.29. Sprawdzanie, czy baza danych została poprawnie utworzona

4. Wpisz lub wklej instrukcję `\q` w powłoce systemu PostgreSQL i wciśnij klawisz *return*, aby z niej wyjść.
5. Pobierz plik `data.dump` z katalogu `Datasets` z repozytorium tej książki w serwisie GitHub (<https://packt.live/30Uhcfl>). Następnie wpisz poniższe polecenie:

```
psql sqlda < ~/Downloads/data.dump
```

To polecenie zadziała, jeśli używany plik jest zapisany w katalogu *Downloads*. Koniecznie dostosuj wyróżnioną ścieżkę do lokalizacji pliku *data.dump* w Twoim systemie.

Następnie poczekaj na zaimportowanie zbioru danych (rysunek 0.30).

```

→ psql sqlda < ~/Downloads/data.dump
SET
SET
SET
SET
SET
set_config
-----
(1 row)

SET
SET
SET
SET
CREATE EXTENSION
COMMENT
CREATE EXTENSION
COMMENT
CREATE TEXT SEARCH DICTIONARY
SET
SET
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE MATERIALIZED VIEW
CREATE TABLE
CREATE MATERIALIZED VIEW
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE

```

Rysunek 0.30. Importowanie zbioru danych

6. Aby sprawdzić, czy zbiór danych został poprawnie zaimportowany, wpisz `psql`, a potem wciśnij klawisz *return*. Ponownie przejdziesz wtedy do powłoki systemu PostgreSQL. Następnie uruchom polecenia `\c sqla` i `\dt`, aby wyświetlić listę tabel w bazie danych (rysunek 0.31).

```
postgres=# \c sqla
You are now connected to database "sqla" as user "maahedev".
sqla=# \dt

               List of relations
Schema |          Name          | Type | Owner
-----+-----+-----+-----
public | closest_dealerships    | table | maahedev
public | countries              | table | maahedev
public | customer_sales         | table | maahedev
public | customer_survey       | table | maahedev
public | customers              | table | maahedev
public | dealerships            | table | maahedev
public | emails                 | table | maahedev
public | products               | table | maahedev
public | public_transportation_by_zip | table | maahedev
public | sales                  | table | maahedev
public | salespeople            | table | maahedev
public | top_cities_data        | table | maahedev
(12 rows)

sqla=#
```

Rysunek 0.31. Lista tabel w bazie sqla

Uruchamianie plików SQL

Polecenia i instrukcje można uruchamiać z poziomu wiersza poleceń za pomocą plików `*.sql`, używając następującej składni:

```
psql -d nazwa_bazy_danych -U nazwa_użytkownika < polecenia.sql
```

Inna możliwość to użycie interpretera SQL-a:

```
baza_danych=#
```

Aby przejść do interaktywnego interpretera, wpisz następujące polecenie:

```
psql -d nazwa_bazy_danych -U nazwa_użytkownika
```

Instalowanie bibliotek

W środowisku Anaconda narzędzie `pip` jest instalowane automatycznie. Po zainstalowaniu w komputerze środowiska Anaconda możesz instalować wszystkie potrzebne biblioteki za pomocą narzędzia `pip`, na przykład `pip install numpy`. Inna możliwość to zainstalowanie wszystkich potrzebnych bibliotek za pomocą instrukcji `pip install -r requirements.txt`. Plik *requirements.txt* znajdziesz na stronie <https://packt.live/330I2FI>.

Ćwiczenia i operacje można wykonywać w notatnikach Jupytera. Jupyter jest biblioteką Pythona; można ją zainstalować w taki sam sposób jak inne biblioteki Pythona (`pip install jupyter`), jednak na szczęście w środowisku Anaconda Jupyter jest instalowany automatycznie. Aby otworzyć notatnik Jupytera, uruchom polecenie `jupyter notebook` w terminalu lub wierszu poleceń.

Pliki z kodem

Kompletny zestaw plików z kodem z tej książki jest dostępny na stronie <https://packt.live/2UCHVer> i w witrynie wydawnictwa Helion.

Oryginalne kolorowe grafiki w wysokiej jakości znajdziesz na stronie <https://packt.live/2HZVdLs>.

Wprowadzenie do SQL-a dla analityków

Gdy zakończysz lekturę tego rozdziału, będziesz umiał opisywać dane i ich typy oraz kategoryzować dane na podstawie ich cech. Obliczysz podstawowe statystyki jednoczynnikowe dotyczące danych i zidentyfikujesz obserwacje odstające. Wykorzystasz też analizy dwuczynnikowe, które pomagają zrozumieć relacje między dwiema zmiennymi. Dowiesz się, do czego służy SQL, i zobaczysz, jak korzystać z tego języka w analityce danych. Na koniec nauczysz się podstaw relacyjnych baz danych i wykonywania operacji **CRUD** (od ang. *create*, *read*, *update* i *delete*, czyli tworzenie, wczytywanie, aktualizowanie i usuwanie) na tabelach.

Wprowadzenie

Dane zmieniły rzeczywistość w XXI wieku. Dzięki łatwemu dostępowi do komputerów firmy i inne organizacje mogły zmodyfikować sposób pracy z większymi i bardziej złożonymi zbiorami danych. Na podstawie danych można obecnie za pomocą kilku wierszy kodu komputerowego uzyskać informacje, jakie 50 lat temu były praktycznie niemożliwe do zdobycia. Dwa najważniejsze narzędzia w tej rewolucji to relacyjne bazy danych i ich podstawowy język, **SQL** (ang. *Structured Query Language*).

Choć teoretycznie możliwe jest analizowanie wszystkich danych ręcznie, komputery radzą sobie z tym zadaniem znacznie lepiej i są preferowanym narzędziem do przechowywania, porządkowania i przetwarzania danych. Do najważniejszych narzędzi związanych z danymi należą relacyjne bazy danych i język umożliwiający dostęp do nich, SQL. Te dwie technologie były przełomem w przetwarzaniu danych i nadal stanowią podstawowe narzędzia w większości firm korzystających z dużych ilości danych.

Firmy używają SQL-a jako podstawowej metody przechowywania większości danych. Ponadto dużą część tych danych umieszczają w specjalnych bazach nazywanych **hurtowniami danych** i **jeziorami danych**, które umożliwiają wykonywanie zaawansowanych analiz. Dostęp do prawie wszystkich hurtowni i jezior danych uzyskuje się za pomocą SQL-a. Dalej zobaczysz, jak używać SQL-a razem z analitycznymi platformami takimi jak hurtownie danych.

Zakładam, że wszyscy Czytelnicy mieli jakąś styczność z SQL-em. Jednak dla osób, które miały bardzo ograniczone doświadczenia z tym językiem lub dawno z niego nie korzystały, w tym rozdziale zamieszczam proste przypomnienie tego, czym są relacyjne bazy danych i SQL, a także podstawowy przegląd operacji i składni SQL-a. Omawiam też praktyczne ćwiczenia, które pomogą utrwalić te informacje.

Następny podrozdział pomoże Ci zrozumieć dane i ich typy.

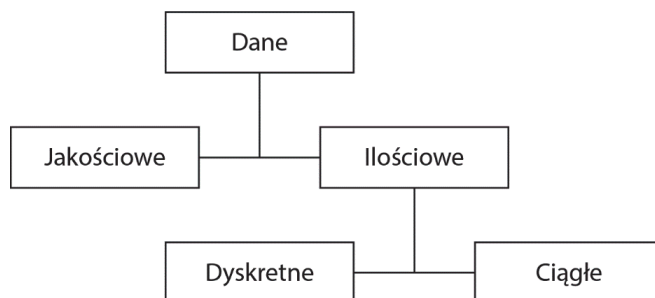
Świat danych

Zacznij od prostego pytania: czym są dane? **Dane** można traktować jako zapisane pomiary czegoś z rzeczywistego świata. Na przykład lista wzrostów to dane, ponieważ wzrost to pomiar odległości między głową a stopami danej osoby. Obiekt danych opisuje **jednostkę obserwacji**. W przypadku wzrostu jednostką obserwacji jest osoba.

Łatwo się domyślić, że istnieje dużo danych opisujących osobę: wiek, waga, czy pali papierosy itd. Jeden lub więcej pomiarów opisujących określoną jednostkę obserwacji to **punkt danych**, a każdy pomiar z punktu danych jest nazywany **zmienną** (lub **cechą**). Zestaw powiązanych punktów danych tworzy **zbiór danych**.

Rodzaje danych

Dane można podzielić na dwie podstawowe kategorie — **ilościowe** i **jakościowe** (rysunek 1.1).



Rysunek 1.1. Rodzaje danych

Dane ilościowe to pomiary, które można opisać za pomocą liczby. Dane jakościowe są reprezentowane za pomocą wartości nieliczbowych, na przykład tekstowo. Wzrost to dane ilościowe. Jednak opis kogoś jako „palącego” lub „niepalącego” to dane jakościowe.

Dane ilościowe można podzielić na dwie podkategorie: **dyskretne** i **ciągłe**. Dane ilościowe dyskretne to wartości mające określony poziom precyzji (zwykle mają postać liczb całkowitych). Na przykład liczba operacji w ciągu życia to dane dyskretne. Możesz mieć 0, 1 lub więcej operacji, ale nie możesz ich mieć 1,5. Zmienna ciągła teoretycznie może mieć dowolną precyzję. Na przykład masę ciała można opisać na dowolnym poziomie precyzji: 55 kg, 55,3 kg, 55,32 kg itd. W praktyce instrumenty pomiarowe oczywiście ograniczają precyzję, jeśli jednak wartość można przedstawić w bardziej precyzyjny sposób, zwykle zmienną uważa się za ciągłą.

Należy zauważyć, że dane jakościowe przeważnie można przekształcić na dane ilościowe, a dane ilościowe na dane jakościowe.

Rozważ to na przykładzie „palącego” i „niepalącego”. Choć możesz stwierdzić, że należysz do jednej z tych kategorii, możesz też przedstawić je w inny sposób, jako ocenę stwierdzenia „palę regularnie”, i użyć wartości logicznych 0 i 1 do reprezentowania odpowiedzi „prawda” i „fałsz”.

Możliwe są też przekształcenia w odwrotną stronę, z danych ilościowych (na przykład wzrostu) na jakościowe. Zamiast myśleć o wzroście dorosłej osoby w kategoriach liczby cali lub centymetrów, możesz przypisać takie wartości do grup. Osoby wyższe niż 180 cm można uznać za „wysokie”, osoby z przedziału od 160 do 180 cm za „średnie”, a osoby niższe niż 160 cm za „niskie”.

Analityka danych i statystyka

Surowe dane są jedynie zestawem wartości. W tej postaci nie są zbyt ciekawe. Dopiero gdy zaczniesz szukać wzorców w danych i je interpretować, możesz przejść do ciekawych czynności, takich jak prognozowanie przyszłości i identyfikowanie nieoczekiwanych zmian. Takie wzorce w danych to **informacje**. Duże uporządkowane kolekcje trwałych i rozbudowanych informacji oraz doświadczeń, które można wykorzystywać do opisywania i prognozowania zjawisk z rzeczywistego świata, zapewniają **wiedzę**. **Analiza danych** to proces przekształcania danych w informację, a dalej w wiedzę. Połączenie analizy danych z prognozowaniem to **analityka danych**.

Dostępnych jest wiele narzędzi, które pomagają zrozumieć dane. Jednym z najbardziej rozbudowanych narzędzi do analizy danych jest zastosowanie technik matematycznych do zbiorów danych. Jedną z tych technik matematycznych jest **statystyka**.

Rodzaje statystyki

Statystykę można podzielić na dwie kategorie: **statystykę opisową** i **wnioskowanie statystyczne**.

Statystyka opisowa służy do opisywania danych. Statystyki opisowe dotyczące jednej zmiennej ze zbioru danych to analizy **jednoczynnikowe**, a statystyki opisowe dotyczące jednocześnie dwóch lub więcej zmiennych to statystyki **wieloczynnikowe**.

We wnioskowaniu statystycznym zbiór danych jest traktowany jako **próbka**, czyli niewielka część pomiarów z większej grupy nazywanej **populacją**. Na przykład ankieta obejmująca 10 000 głosujących w wyborach krajowych daje próbkę opinii całej populacji głosującej w kraju. Wnioskowanie statystyczne służy do wyciągania wniosków na temat cech populacji na podstawie cech próbki.

W tej książce skupiamy się przede wszystkim na statystykach opisowych. Więcej o wnioskowaniu statystycznym dowiesz się z podręczników do statystyki takich jak *Statistics* Davida Freedmana, Roberta Pisaniego i Rogera Purvesa.

Wyobraź sobie, że jesteś analitykiem zajmującym się polityką zdrowotną i otrzymałeś zbiór danych z informacjami o pacjentach (rysunek 1.2).

Rok Urodzenia	Kraj Urodzenia	Wzrost (cm)	Kolor Oczu	Liczba Wizyt Lekarza w 2018
1997	Egipt	182	Niebieskie	1
1988	Chiny	196	Piżne	2
1986	USA	180	Brązowe	2
1990	USA	166	Brązowe	1
1975	Indie	181	Zielone	3
1951	Niemcy	184	Brązowe	1
2000	Australia	174	Szare	5
1995	Indie	183	Brązowe	1
1992	Chiny	187	Brązowe	2
1987	USA	169	Niebieskie	2

Rysunek 1.2. Dane dotyczące opieki zdrowotnej

Gdy dostępny jest zbiór danych, często warto je poklasyfikować. Tu jednostką obserwacji dla zbioru danych jest pacjent, ponieważ każdy wiersz reprezentuje pojedynczą obserwację, która odpowiada unikatowemu pacjentowi. Trzy kolumny, Data Urodzenia, Wzrost i Liczba Wizyt Lekarza, są ilościowe, ponieważ do ich reprezentowania służą liczby. Dwie kolumny, Kolor Oczu i Kraj Urodzenia, są jakościowe.

Zadanie 1.01 — klasyfikowanie nowego zbioru danych

W tym zadaniu poklasyfikujesz dane ze zbioru. Niedługo zaczynasz pracę w startupie w innym mieście. Jesteś podekscytowany, ale przed przeprowadzką decydujesz się sprzedać wszystkie swoje rzeczy, w tym samochód. Nie jesteś pewien, jakiej ceny zażądać, dlatego chcesz zebrać trochę danych. Pytasz znajomych i członków rodziny, którzy ostatnio sprzedali samochód, o markę i cenę ich pojazdów. Na tej podstawie otrzymujesz zbiór danych z rysunku 1.3.

Data	Marka	Wartość Sprzedaży (w tysiącach zł)
01.02.2018	Ford	12
02.02.2018	Honda	15
02.02.2018	Mazda	19
03.02.2018	Ford	20
04.02.2018	Toyota	10
04.02.2018	Toyota	10
04.02.2018	Mercedes	30
05.02.2018	Ford	11
06.02.2018	Chevrolet	12,5
06.02.2018	Chevrolet	19

Rysunek 1.3. Dane na temat sprzedaży używanych samochodów

Oto kroki, jakie należy wykonać:

1. Ustalić jednostkę obserwacji.
2. Ocenić trzy kolumny pod kątem tego, czy zawierają dane ilościowe, czy jakościowe.
3. Przekształcić kolumnę Marka na kolumnę z danymi ilościowymi.

Rozwiązanie tego zadania znajdziesz w „Dodatku”.

W tym zadaniu nauczyłeś się klasyfikować dane. W następnym podrozdziale poznasz różne metody z obszaru statystyki opisowej.

Metody z obszaru statystyki opisowej

Wcześniej wspomnieliśmy, że statystyka opisowa jest jednym ze sposobów na analizowanie danych w celu ich zrozumienia. Analiza jedno- i wieloczynnikowa mogą dać wgląd w dane zjawisko. W tym podrozdziale przyjrzyj się podstawowym technikom matematycznym, które możesz wykorzystać, aby lepiej zrozumieć i opisać zbiór danych.

Analiza jednoczynnikowa

Jedną z gałęzi statystyki jest analiza jednoczynnikowa. Jej metody pozwalają zrozumieć jedną zmienną ze zbioru danych. W tym punkcie omawiamy wybrane z najczęściej stosowanych technik analizy jednoczynnikowej.

Rozkład danych

Rozkład danych bazuje na liczbie określonych wartości w zbiorze danych. Przyjmijmy, że zbiór danych zawiera 1000 kart zdrowia, a jedną ze zmiennych w tych kartach jest kolor oczu. Jeśli po przejrzeniu tego zbioru stwierdzisz, że 700 osób ma brązowe oczy, 200 osób ma zielone oczy, a 100 — niebieskie, opiszesz rozkład danych, a dokładniej **rozkład bezwzględnej częstości występowania**. Gdybyśmy podali nie liczbę wystąpień wartości w zbiorze danych, lecz odsetek jej wystąpień w łącznej liczbie punktów danych, opisalibyśmy **rozkład względnej częstości występowania**. W przykładzie z kolorami oczu rozkład względnej częstości występowania to: oczy brązowe 70%, oczy zielone 20%, oczy niebieskie 10%.

Łatwo jest obliczyć rozkład, gdy zmienna przyjmuje niewielką liczbę stałych wartości takich jak kolor oczu. Co jednak ze zmiennymi ilościowymi, które mogą przyjmować wiele różnych wartości, na przykład ze wzrostem? Ogólna metoda obliczania rozkładu dla zmiennych tego rodzaju polega na tworzeniu „kubeków”, do których można przypisać poszczególne wartości, i obliczaniu rozkładów na podstawie tych kubeków. Na przykład wzrost można podzielić na kubeczki obejmujące po 5 cm, aby uzyskać rozkład bezwzględnej częstości występowania. Następnie można podzielić każdy wiersz tabeli przez łączną liczbę punktów danych (10 000) i otrzymać rozkład względnej częstości występowania.

Inną przydatną techniką jest tworzenie wykresów rozkładów. Teraz utworzysz **histogram**, który jest graficzną reprezentacją rozkładu ciągłego z użyciem kubeków.

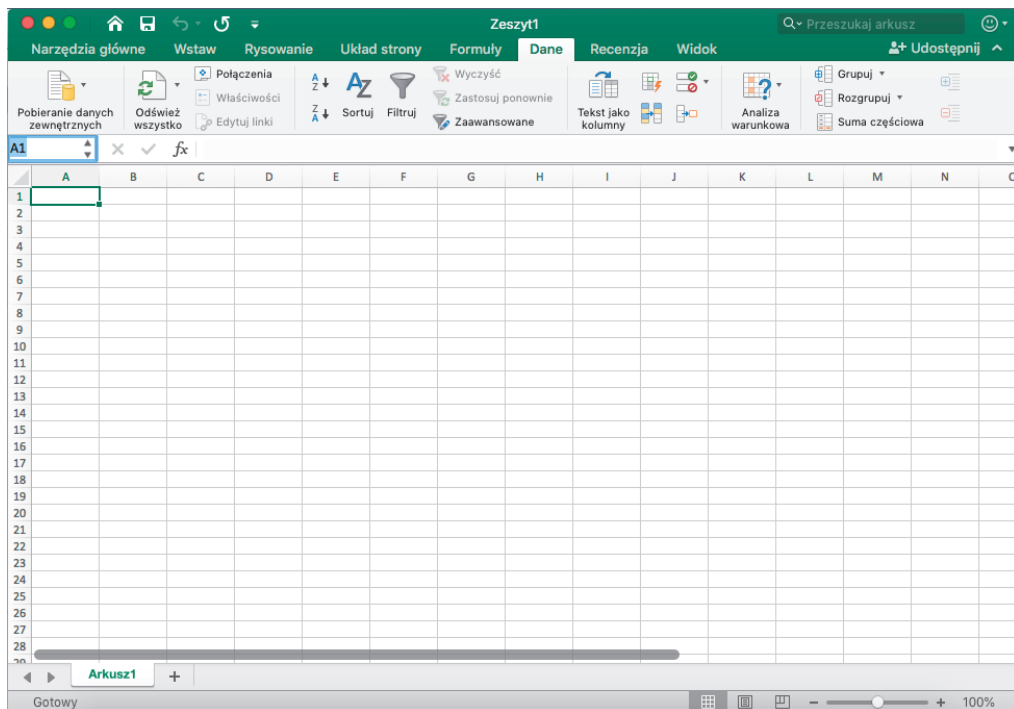
Ćwiczenie 1.01 — tworzenie histogramu

W tym ćwiczeniu użyjesz programu Microsoft Excel do utworzenia histogramu. Przyjmij, że jesteś analitykiem polityki zdrowotnej i chcesz zobaczyć rozkład wzrostu, aby dostrzec w nim wzorce. Aby wykonać to zadanie, musisz przygotować histogram.

Do tworzenia histogramów możesz użyć arkusza kalkulacyjnego, na przykład w programie Excel, albo języków takich jak Python lub R. Dla wygody tu posłużysz się Excelem. Zbiory danych z tego rozdziału znajdziesz w serwisie GitHub: <https://packt.live/2B1apb3>.

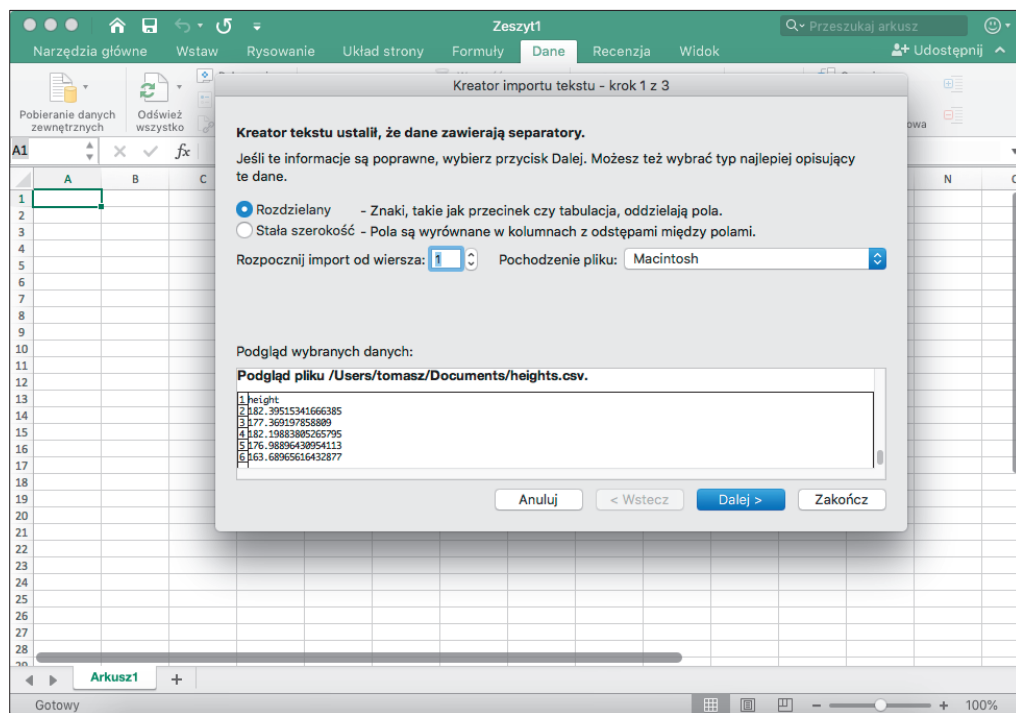
Wykonaj następujące kroki:

1. Otwórz pusty skoroszyt w programie Microsoft Excel (rysunek 1.4).
2. Przejdź do zakładki *Dane* i wybierz opcję *Pobieranie danych zewnętrznych/Z pliku tekstowego*.



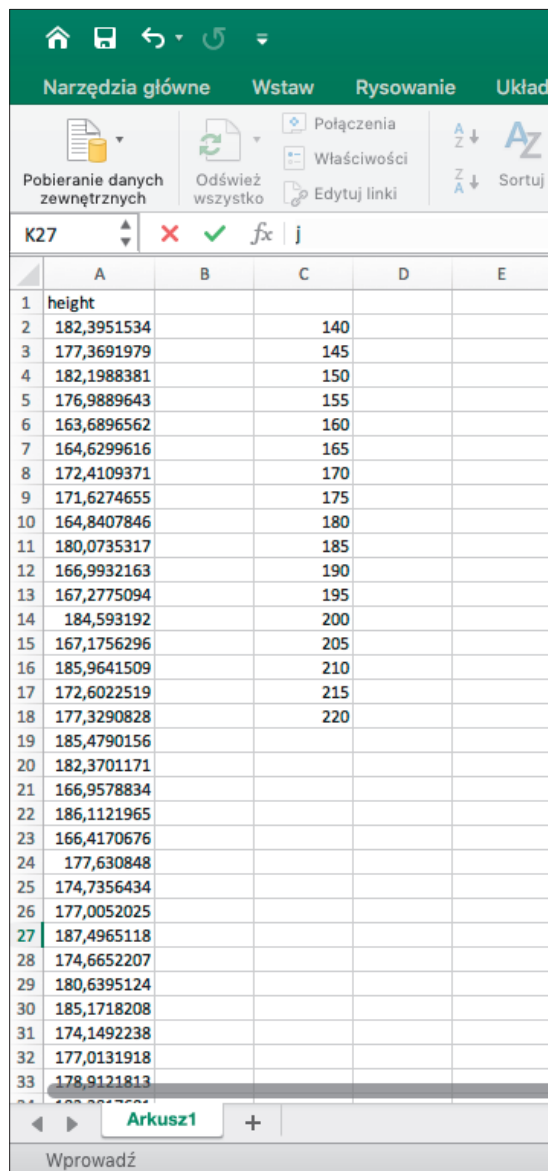
Rysunek 1.4. Pusty skoroszyt w Excelu

3. Znajdź plik ze zbiorem danych *heights.csv* w katalogu *Datasets* z repozytorium serwisu GitHub. Po wskazaniu tego pliku kliknij OK.
4. Wybierz opcję *Rozdzielany* w oknie dialogowym *Kreator importu tekstu*. Import należy rozpocząć od wiersza 1. Następnie kliknij *Dalej*.
5. Wybierz ogranicznik używany w pliku. Ponieważ ten plik ma tylko jedną kolumnę, nie występują w nim ograniczniki. W plikach CSV tradycyjnie ogranicznikami są przecinki; w przyszłości używaj ograniczników odpowiednich dla zbiorów danych. Teraz kliknij *Dalej*.
6. Wybierz *Ogólne* w sekcji *Format danych kolumny*. Kliknij przycisk *Zaawansowane* i w nowym oknie jako *Separator dziesiętny* wybierz kropkę, po czym kliknij OK. Następnie kliknij *Zakończ*.
7. W oknie dialogowym z pytaniem *Gdzie chcesz umieścić dane?* wybierz opcję *Istniejący arkusz* i nie zmieniaj wartości w polu tekstowym obok tej opcji. Kliknij przycisk OK.
8. W kolumnie C zapisz liczby 140, 145, 150 itd., zwiększając wartości o 5 aż do 220. Umieść je w komórkach od C2 do C18, tak jak na rysunku 1.6.



Rysunek 1.5. Zaznacz opcję Rozdzielany

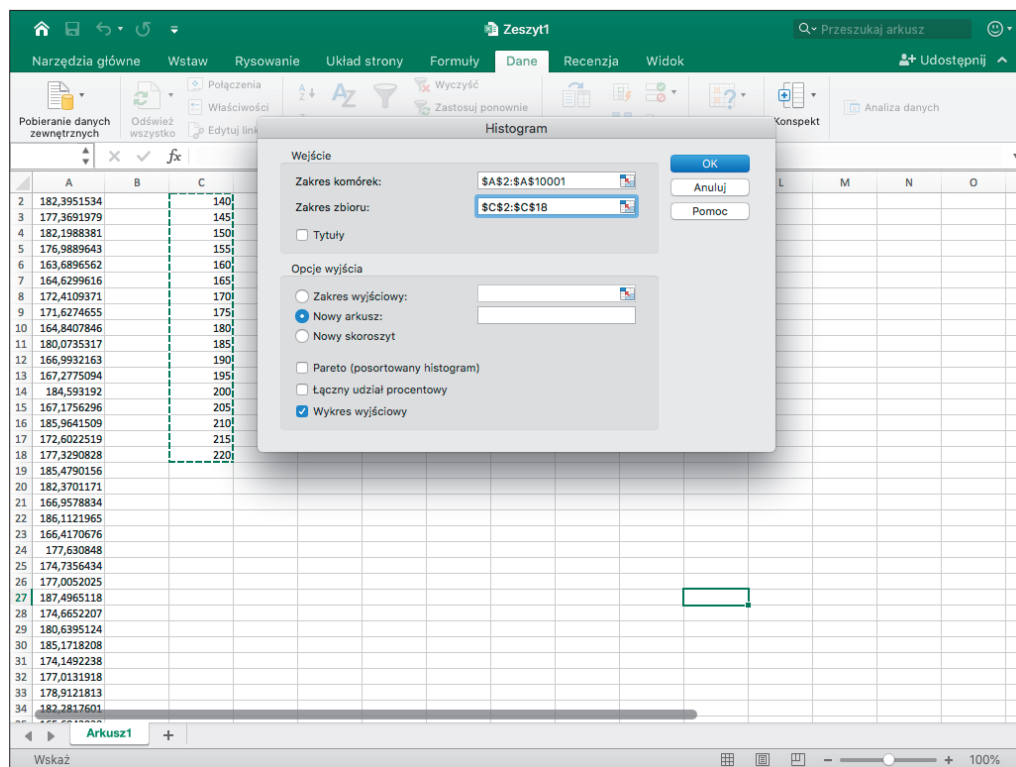
9. W zakładce *Dane* wybierz opcję *Analiza danych* (jeśli jej nie widzisz, zastosuj się do instrukcji ze strony <https://support.office.com/en-us/article/load-the-analysis-toolpak-in-excel-6a63e598-cd6d-42e3-9317-6b40ba1a66b4>, aby ją dodać).
10. W polu wyboru, które się pojawi, wybierz opcję *Histogram* i kliknij przycisk *OK*.
11. Aby podać opcję *Zakres komórek*, kliknij przycisk po prawej stronie pola tekstowego. Powinieneś wrócić do arkusza *Arkusz1*, gdzie widoczne będzie puste pole z przyciskiem z czerwoną strzałką. Przeciągnij kursor i zaznacz wszystkie dane w arkuszu od komórki A2 do A10001. Następnie kliknij przycisk z czerwoną strzałką.
12. Aby ustawić opcję *Zakres zbioru*, kliknij przycisk po prawej stronie pola tekstowego. Powinieneś wrócić do arkusza *Arkusz1*, gdzie widoczne będzie puste pole z przyciskiem z czerwoną strzałką. Przeciągnij kursor i zaznacz wszystkie dane w arkuszu od komórki C2 do C18. Następnie kliknij przycisk z czerwoną strzałką.
13. W sekcji *Opcje wyjścia* zaznacz pole *Nowy arkusz* i upewnij się, że zaznaczone jest pole *Wykres wyjściowy*, tak jak na rysunku 1.7. Następnie kliknij przycisk *OK*.



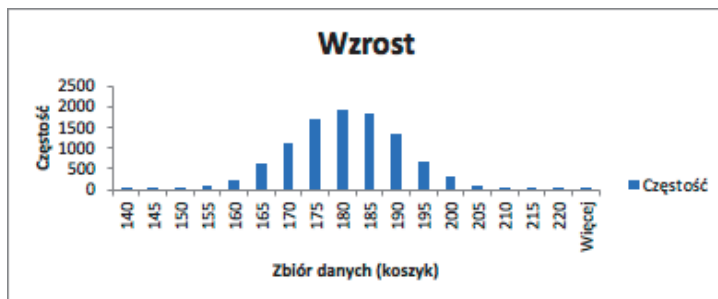
Rysunek 1.6. Wprowadzanie danych w arkuszu Excela

14. Kliknij *Arkusz2*. Znajdź wykres i kliknij dwukrotnie tytuł *Histogram*. Zastąp go słowem *Wzrost*. Powinieneś uzyskać wykres podobny do tego z rysunku 1.8.

Przyjrzenie się kształtowi rozkładu pozwala odkryć ciekawe wzorce. Zauważ, że ten rozkład ma symetryczny kształt dzwona. Jest to tak zwany *rozkład normalny*, występujący w wielu zbiorach danych. W tej książce nie omawiamy szczegółowo tego rozkładu, ale zwróć na niego uwagę w trakcie analiz danych; często będziesz na niego natrafiać.



Rysunek 1.7. Wybierz opcję Nowy arkusz



Rysunek 1.8. Rozkład wzrostu dorosłych mężczyzn

Kwantyle

Jednym ze sposobów na liczbowy opis rozkładu jest użycie kwantyli. Kwantyle rzędu N to zbiór $n - 1$ punktów dzielących zmienną na n grup. Te punkty czasem nazywa się **punktami podziału**. Na przykład kwantyle rzędu 4 (nazywane kwartylami) to grupa trzech punktów, które dzielą zmienną na cztery w przybliżeniu równe grupy wartości. Na rysunku 1.9 wymienione są nazwy kwantyli różnego rzędu.

N	Nazwa
3	Tercyle
4	Kwartyle
5	Kwintyle
10	Decyle
20	Vingtile
100	Percentyle

Rysunek 1.9. Nazwy kwantyli rzędu N

Istnieją różne procedury obliczania kwantyli. Tu użyjesz następującej techniki do obliczania kwantyli rzędu N dla d punktów danych dla jednej zmiennej:

1. Uporządkuj punkty danych od najmniejszego do największego.
2. Ustal liczbę n dla kwantyli rzędu N, jakie chcesz wyznaczyć, oraz liczbę punktów podziału ($n - 1$).
3. Ustal liczbę k -tego punktu podziału, jaki chcesz obliczyć, czyli liczbę z przedziału od 1 do $n - 1$. Jeśli rozpoczynasz obliczenia, użyj k równego 1.
4. Wyznacz indeks i dla k -tego punktu podziału. Użyj wzoru z rysunku 1.10.

$$i = \left\lceil \frac{k}{n} (d - 1) \right\rceil + 1$$

Rysunek 1.10. Indeks

5. Jeśli i jest liczbą całkowitą, wybierz element o tym numerze spośród uporządkowanych punktów danych. Jeżeli k -ty punkt podziału nie jest liczbą całkowitą, znajdź elementy bezpośrednio przed oraz po i . Oblicz różnicę między wartościami tych elementów i pomnóż ją przez część dziesiętną uzyskanego indeksu. Dodaj wynik do mniejszego z dwóch uwzględnianych elementów.
6. Powtarzaj kroki od 2. do 5. dla różnych wartości k do momentu obliczenia wszystkich punktów podziału.

Te kroki dość trudno jest zrozumieć bez kontekstu, dlatego warto wykonać ćwiczenie. Dzięki wielu współczesnym narzędziom, między innymi SQL-owi, komputery mogą szybko obliczać kwantyle za pomocą wbudowanych mechanizmów.

Ćwiczenie 1.02

— obliczanie kwartyli dla sprzedaży dodatków

W tym ćwiczeniu za pomocą Excela poklasyfikujesz dane i obliczysz kwantyle dotyczące sprzedaży samochodów. Twój nowy szef chce, abyś przejrzał dane, zanim zaczniesz pracę w poniedziałek, co pozwoli Ci lepiej zrozumieć jedno z zadań, jakimi będziesz się zajmować

— zwiększenie sprzedaży dodatkowego wyposażenia przy zakupie samochodów. Szef przesyła Ci listę 11 transakcji i kwot wydanych na dodatki do podstawowej wersji nowego modelu ZoomZoom Chi. Oto wartości sprzedaży dodatkowego wyposażenia: 5000, 1700, 8200, 1500, 3300, 9000, 2000, 0, 0, 2300, 4700.

Wszystkie zbiory danych używane w tym rozdziale znajdziesz w serwisie GitHub: <https://packt.live/2B1apb3>.

Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Otwórz pusty skoroszyt w programie Microsoft Excel.
2. Przejdź do zakładki *Dane* i wybierz opcję *Pobieranie danych zewnętrznych/ Z pliku tekstowego*.
3. Plik ze zbiorem danych *auto_upgrades.csv* znajdziesz w katalogu *Datasets* w repozytorium w serwisie GitHub. Przejdź do tego pliku i kliknij przycisk *Pobierz dane*.
4. Wybierz opcję *Rozdzielany* w oknie dialogowym *Kreator importu tekstu*. Import musi się rozpoczynać od wiersza 1. Następnie kliknij *Dalej*.
5. Wybierz ogranicznik używany w pliku. Ponieważ ten plik ma tylko jedną kolumnę, nie występują w nim ograniczniki. W plikach CSV tradycyjnie ogranicznikami są przecinki; w przyszłości używaj ograniczników odpowiednich dla zbiorów danych. Teraz kliknij *Dalej*.
6. Wybierz *Ogólne* w sekcji *Format danych kolumny*. Kliknij przycisk *Zaawansowane* i w nowym oknie jako *Separator dziesiętny* wybierz kropkę, po czym kliknij *OK*. Następnie kliknij *Zakończ*.
7. W oknie dialogowym z pytaniem *Gdzie chcesz umieścić dane?* wybierz opcję *Istniejący arkusz* i nie zmieniaj wartości w polu tekstowym obok tej opcji. Kliknij przycisk *OK*.
8. Kliknij komórkę *A1*, a następnie otwórz zakładkę *Dane* i wybierz opcję *Sortuj*.
9. Pojawi się okno dialogowe sortowania. Kliknij przycisk *OK*. Wartości zostaną posortowane od najmniejszych do największych. Lista z rysunku 1.11 przedstawia posortowane wartości.
10. Teraz ustal liczbę *kwantyli rzędu n* i punktów podziału, jakie musisz obliczyć. Kwartyle to kwantyle rzędu 4, zgodnie z rysunkiem 1.9. Ponieważ liczba punktów podziału jest o 1 mniejsza od liczby *kwantyli rzędu n*, wiadomo, że potrzebne są trzy punkty podziału.
11. Oblicz indeks pierwszego punktu podziału. Tu $k=1$, liczba wartości w populacji (d) jest równa 11, a liczba *kwantyli rzędu n* (n) to cztery. Po podstawieniu tych wartości w równaniu z rysunku 1.12 uzyskasz 3,5.

	A	B	C	D
1	Add-on Sales (\$)			
2	0			
3	0			
4	1500			
5	1700			
6	2000			
7	2300			
8	3300			
9	4700			
10	5000			
11	8200			
12	9000			
13				

Rysunek 1.11. Posortowane wartości sprzedaży dodatkowego wyposażenia

$$\begin{aligned}
 i &= \left\lceil \frac{k}{n} (d - 1) \right\rceil + 1 \\
 i &= \left\lceil \frac{1}{4} (11 - 1) \right\rceil + 1 \\
 i &= \frac{10}{4} + 1 \\
 i &= \frac{10}{4} + 1 \\
 i &= 2,5 + 1 = 3,5
 \end{aligned}$$

Rysunek 1.12. Obliczanie indeksu pierwszego punktu podziału

12. Ponieważ indeks 3,5 nie jest liczbą całkowitą, najpierw trzeba znaleźć elementy trzeci i czwarty (1500 i 1700). Należy ustalić różnicę między nimi (200), a następnie pomnożyć ją przez część dziesiętną indeksu, czyli 0,5, co daje w wyniku 100. Tę wartość należy dodać do trzeciego elementu (1500), uzyskasz więc wartość 1600.
13. Powtórz kroki od 2. do 5. procedury dla $k=2$ i $k=3$, aby obliczyć poziom drugiego i trzeciego kwartyla. Powinieneś otrzymać wartości 2300 i 4850.

W tym ćwiczeniu zobaczyłeś, jak klasyfikować dane i obliczać kwartyle za pomocą Excela.

Tendencja centralna

Jedno z podstawowych pytań na temat zmiennych ze zbioru danych dotyczy typowej wartości określonej zmiennej. Ta wartość jest często nazywana **tendencją centralną** zmiennej. Do opisywania tendencji centralnej można użyć wielu wartości obliczanych na podstawie zbioru danych; wszystkie one mają wady i zalety. Oto niektóre miary tendencji centralnej:

- **Wartość modalna** — jest to wartość, która najczęściej występuje w rozkładzie zmiennej. Na rysunku 1.2 dla przykładu dotyczącego koloru oczu wartość modalna to „oczy brązowe”, ponieważ występuje ona w tym zbiorze najczęściej. Jeśli istnieje kilka takich wartości, zmienna jest nazywana **wielomodalną** i należy podać wszystkie najczęściej występujące wartości. Jeżeli żadna wartość się nie powtarza, w danym zbiorze wartość modalna nie występuje. Wartość modalna jest przydatna, jeśli zmienna może przyjmować niewielką, stałą liczbę wartości. Trudno jest ją jednak wyznaczyć dla ciągłych zmiennych ilościowych, na przykład dla wzrostu. Wtedy tendencję centralną lepiej jest określać za pomocą innych miar.
- **Średnia** — średnia dla zmiennej to wartość obliczona przez zsumowanie wszystkich jej wartości i podzielenie wyniku przez liczbę punktów danych. Załóżmy, że używamy niewielkiego zbioru danych z wiekiem: 26, 25, 31, 35 i 29. Średnia dla tego zbioru wynosi 29,2, ponieważ ten właśnie wynik uzyskasz, gdy zsumujesz tych pięć liczb, a następnie podzielisz uzyskaną sumę przez 5 (czyli liczbę punktów danych). Średnią łatwo jest obliczyć i zwykle dobrze opisuje ona „typową” wartość zmiennej. Nie jest więc zaskoczeniem, że stanowi ona jedną z najczęściej podawanych statystyk opisowych w literaturze przedmiotu. Jednak średnia jako miara tendencji centralnej ma ważną wadę: jest wrażliwa na **wartości odstające**.
 Wartość odstająca to punkt danych, który znacznie różni się od pozostałych danych i występuje bardzo rzadko. Wartości odstające często można identyfikować za pomocą technik graficznych, na przykład na wykresach punktowych lub skrzynkowych, gdzie widoczne są wszystkie punkty danych bardzo oddalone od pozostałych. Gdy w zbiorze danych występuje wartość odstająca, można go nazwać **skośnym zbiorem danych**. Częste powody występowania wartości odstających to nieoczyszczone dane, niezwykle rzadkie zdarzenia i problemy z instrumentami pomiarowymi. Wartości odstające często zniekształcają średnią do tego stopnia, że przestaje ona reprezentować typowe wartości danych.
- **Mediana** — jest nazywana także drugim kwartylem lub percentylem 50% i stanowi dość dziwną miarę tendencji centralnej, jednak ma ważne zalety w porównaniu ze średnią. Aby obliczyć medianę, posortuj wartości zmiennej od najmniejszej do największej, a następnie wybierz wartość środkową. Dla nieparzystej liczby punktów danych jest to środkowa wartość w uporządkowanych danych. Gdy liczba punktów danych jest parzysta, użyj średniej dwóch środkowych wartości.

Choć obliczanie mediany jest trochę niewygodne, jest ona mniej wrażliwa na wartości odstające (w porównaniu ze średnią). Aby się o tym przekonać, oblicz medianę dla skośnego zbioru danych z wiekiem: 26, 25, 31, 35, 29 i 82. Mediana dla tego zbioru danych wynosi 30. Wartość ta jest dużo bliższa typowej wartości

tego zbioru danych niż średnia, która wynosi 38. Ta odporność na wartości odstające jest jednym z głównych powodów używania mediany.

Zgodnie z ogólną regułą warto obliczyć zarówno średnią, jak i medianę zmiennej. Jeśli te miary znacznie się od siebie różnią, w zbiorze danych mogą występować wartości odstające.

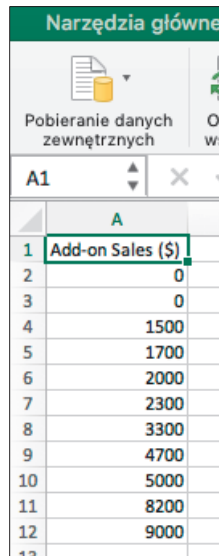
W następnym ćwiczeniu zobaczysz, jak obliczać miary tendencji centralnej.

Ćwiczenie 1.03 — obliczanie miar tendencji centralnej dla sprzedaży dodatków

W tym ćwiczeniu obliczysz miary tendencji centralnej dla danych w Excelu. Aby lepiej zrozumieć dane dotyczące sprzedaży dodatkowego wyposażenia (które można dokupić do podstawowego modelu), należy ustalić typową wartość zmiennej. Oblicz wartość modalną, średnią i medianę dla tych danych. Oto wartość dodatków w 11 transakcjach zakupu samochodów: 5000, 1700, 8200, 1500, 3300, 9000, 2000, 0, 0, 2300 i 4700.

W tym ćwiczeniu wykonaj następujące kroki:

1. Najpierw oblicz wartość modalną, aby wyznaczyć najczęściej występującą wartość. Ponieważ w tym zbiorze danych najczęściej pojawia się 0, wartość modalna to 0.
2. Teraz oblicz średnią. Zsumuj liczby z kolumny Add-on Sales. Wynik to 37 700. Podziel go przez liczbę wartości (11), a otrzymasz średnią 3427,27.
3. W ostatnim kroku oblicz medianę. W tym celu posortuj dane tak jak na rysunku 1.13.



Narzędzia główne	
Pobieranie danych zewnętrznych	
A1	
	A
1	Add-on Sales (\$)
2	0
3	0
4	1500
5	1700
6	2000
7	2300
8	3300
9	4700
10	5000
11	8200
12	9000
13	

Rysunek 1.13. Posortowane dane o sprzedaży dodatków

Ustal środkową wartość. Ponieważ jest 11 punktów danych, środkowa jest szósta wartość na liście. Sprawdź więc szósty element w posortowanych danych, a otrzymasz medianę równą 2300.

Po zapoznaniu się z tendencją centralną możesz przejść do innej cechy danych — dyspersji.

Gdy porównasz średnią z medianą, zobaczysz, że znacznie się od siebie różnią. Wcześniej wspomnieliśmy, że jest to oznaka występowania wartości odstających w zbiorze danych. Dalej dowiesz się, jak stwierdzić, które wartości są odstające.

Dyspersja

Inną ciekawą cechą zbioru danych jest to, jak blisko siebie znajdują się wartości zmiennej w punktach danych. Na przykład zbiory liczb [100, 100, 100] i [50, 100, 150] oba mają średnią 100, ale wartości w drugim są bardziej rozproszone niż w pierwszym. Cecha opisująca rozproszenie danych jest nazywana **dyspersją**.

Istnieje wiele sposobów pomiaru dyspersji zmiennej. Oto kilka popularnych technik:

- **Rozstęp** — jest to różnica między największą a najmniejszą wartością zmiennej. Można go bardzo łatwo obliczyć, ale jest niezwykle wrażliwy na wartości odstające. Ponadto nie daje informacji na temat rozproszenia wartości pośrodku zbioru danych.
- **Odchylenie standardowe i wariancja** — odchylenie standardowe to pierwiastek kwadratowy ze średniej kwadratów różnic wartości punktu danych od średniej. Odchylenie standardowe przyjmuje wartości od zera do dodatniej nieskończoności. Im bliżej odchylenie standardowe jest zera, tym mniejsze są różnice między liczbami w zbiorze danych. Gdy odchylenie standardowe wynosi zero, wszystkie wartości w zbiorze danych są takie same.

Warto zwrócić uwagę na to, że są dwa różne wzory na obliczanie odchylenia standardowego, co ilustruje rysunek 1.14. Gdy zbiór danych reprezentuje całą populację, należy obliczyć odchylenie standardowe dla populacji, używając wzoru A z tego rysunku. Jeśli próbka reprezentuje tylko część obserwacji, należy posłużyć się wzorem B do obliczenia odchylenia standardowego dla próbki. Gdy masz wątpliwości, zastosuj wzór na odchylenie standardowe dla próbki, ponieważ jest on bardziej zachowawczy. W praktyce gdy liczba punktów danych jest duża, różnica między tymi dwoma wzorami jest bardzo niewielka.

$$\text{A)} \quad \sqrt{\frac{\sum_{i=1}^n (x_i - u_x)^2}{n}} \quad \text{B)} \quad \sqrt{\frac{\sum_{i=1}^n (x_i - u_x)^2}{n-1}}$$

Rysunek 1.14. Wzory na odchylenie standardowe dla populacji (A) i próbki (B)

Najczęściej używaną miarą do opisywania dyspersji jest właśnie odchylenie standardowe. Jednak, podobnie jak rozstęp, jest ona wrażliwa na wartości odstające, choć w mniejszym stopniu. Obliczanie odchylenia standardowego jest też dość skomplikowane, lecz współczesne narzędzia zwykle umożliwiają łatwe uzyskanie tej miary.

Na koniec ostatnia uwaga: czasem możesz zetknąć się z powiązaną wartością, wariancją. Równa się ona odchyleniu standardowemu podniesionemu do kwadratu.

- **Rozstęp ćwiartkowy** — jest to różnica między pierwszym kwartyłem (Q1, nazywany też dolnym) i trzecim kwartyłem (Q3, nazywany też górnym).

Więcej informacji o obliczaniu kwantyli i kwartyli znajdziesz w punkcie „Rozkład danych” w tym rozdziale.

Rozstęp ćwiartkowy, w odróżnieniu od rozstępu i odchylenia standardowego, jest odporny na wartości odstające. Dlatego choć jest to jedna z najtrudniejszych do obliczenia miar, dobrze nadaje się do pomiaru dyspersji zbioru danych. Często jest też używana do definiowania wartości odstających. Jeśli wartość w zbiorze danych jest mniejsza niż $(Q1 - 1,5 \times \text{rozstęp ćwiartkowy})$ lub większa niż $(Q1 + 1,5 \times \text{rozstęp ćwiartkowy})$, jest uznawana za odstającą.

Aby lepiej zrozumieć dyspersję, wykonaj następne ćwiczenie.

Ćwiczenie 1.04

— obliczanie dyspersji dla sprzedaży dodatków

W tym ćwiczeniu obliczysz rozstęp, odchylenie standardowe i rozstęp ćwiartkowy. Aby lepiej zrozumieć sprzedaż dodatków i opcjonalnych urządzeń, dokładnie przyjrzyj się dyspersji danych. Oto dane dla 11 transakcji zakupu dodatkowego wyposażenia: 5000, 1700, 8200, 1500, 3300, 9000, 2000, 0, 0, 2300 i 4700.

Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Oblicz zakres. W tym celu znajdź najmniejszą wartość w danych (0) i odejmij ją od wartości maksymalnej (9000). Uzyskasz wynik 9000.
2. Obliczenie odchylenia standardowego wymaga, aby najpierw ustalić, czy ma ono dotyczyć próbki czy populacji. Ponieważ 11 analizowanych punktów danych reprezentuje niewielką próbkę wszystkich transakcji, obliczysz odchylenie standardowe dla próbki.
3. Następnie znajdź średnią zbioru danych. Obliczyłeś ją już w „Ćwiczeniu 1.02 — obliczanie kwartyli dla sprzedaży dodatków”; wynik to 3427,27.
4. Teraz odejmij wszystkie punkty danych od średniej i podnieś wynik do kwadratu. Wyniki są pokazane na rysunku 1.15.

Add-on Sales (\$)	Różnica względem średniej	Kwadrat różnicy względem średniej
5000	1572,727273	2473471,074
1700	-1727,272727	2983471,074
8200	4772,727273	22778925,62
1500	-1927,272727	3714380,165
3300	-127,2727273	16198,34711
9000	5572,727273	31055289,26
2000	-1427,272727	2037107,438
0	-3427,272727	11746198,35
0	-3427,272727	11746198,35
2300	-1127,272727	1270743,802
4700	1272,727273	1619834,711

Rysunek 1.15. Obliczanie sumy kwadratów różnic

5. Zsumuj wartości z kolumny *Kwadrat różnicy względem średniej*; wynik to 91 441 818.
6. Podziel tę sumę przez liczbę punktów danych minus 1 (czyli przez 10) i wyciągnij pierwiastek kwadratowy. Te obliczenia powinny dać odchylenie standardowe dla próbki równe 3023,93.
7. Aby obliczyć rozstęp ćwiartkowy, wyznacz pierwszy i trzeci kwartył. Obliczenia znajdziesz w „Ćwiczeniu 1.02 — obliczanie kwartyli dla sprzedaży dodatków”; wyniki to 1600 i 4850. Odejmij te dwie wartości, a uzyskasz wynik 3250.

W tym ćwiczeniu obliczyłeś rozstęp, odchylenie standardowe i rozstęp ćwiartkowy. W następnym ćwiczeniu zobaczysz, jak posłużyć się analizą dwuczynnikową do znajdowania wzorców.

Analiza dwuczynnikowa

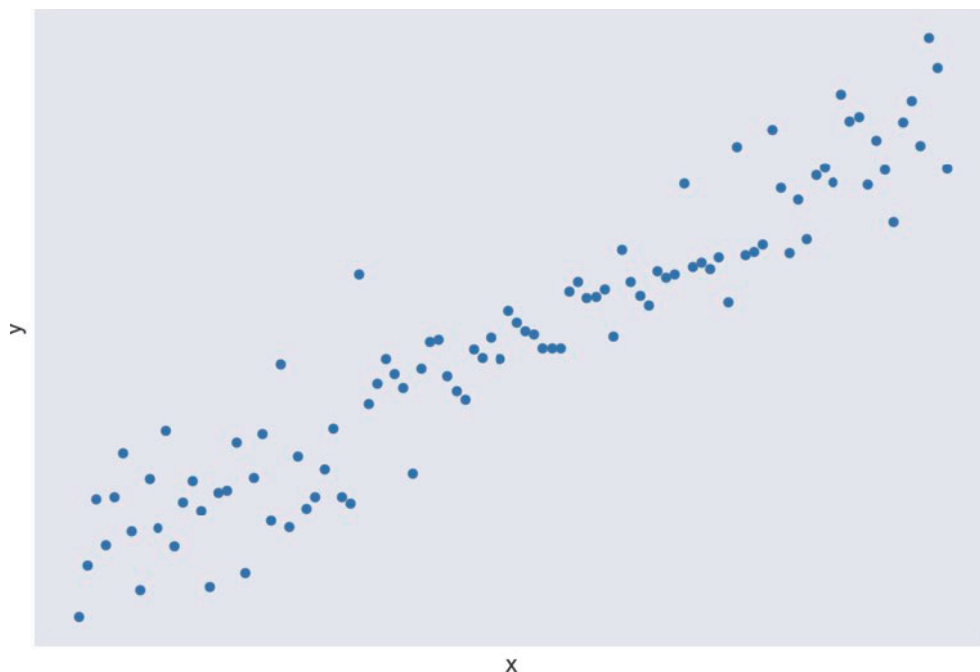
Do tej pory omawialiśmy metody opisu jednej zmiennej. Teraz zobaczysz, jak za pomocą analizy dwuczynnikowej wykrywać wzorce dotyczące dwóch zmiennych.

Wykresy punktowe

Ogólną regułą, jaką odkryjesz w analityce, jest to, że wykresy są niezwykle pomocne w wyszukiwaniu wzorców. Podobnie jak histogramy pomagają zrozumieć jedną zmienną, wykresy punktowe ułatwiają zapoznanie się z dwiema zmiennymi. Wykresy punktowe można łatwo przygotować za pomocą wybranego arkusza kalkulacyjnego.

Wykresy punktowe są pomocne przede wszystkim w sytuacji, gdy liczba punktów jest niewielka (zwykle od 30 do 500). Jeśli liczba punktów jest duża i naniesienie ich na wykres kończy się uzyskaniem jednej wielkiej plamy, wybierz losową próbkę 200 punktów i utwórz wykres na ich podstawie, co może pomóc Ci w wykryciu ciekawych trendów.

Na wykresie punktowym możesz znaleźć wiele różnych wzorców. Najczęściej wyszukiwane są trendy wzrostowe i malejące dla dwóch zmiennych. Pozwala to stwierdzić, czy wraz ze wzrostem wartości jednej zmiennej druga zmienna też rośnie, czy maleje. Trend wskazuje na to, że między dwiema zmiennymi może występować przewidywalna zależność matematyczna. Istnieje na przykład trend rosnący dla zależności między wiekiem i zarobkami. Rysunek 1.16 ilustruje przykładowy trend liniowy.

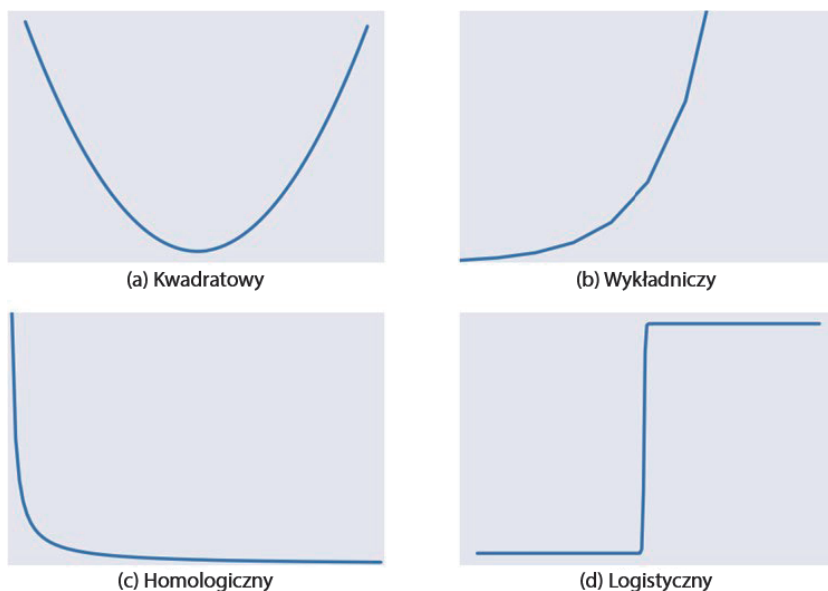


Rysunek 1.16. Rosnący trend liniowy dla dwóch zmiennych — wieku i zarobków osób

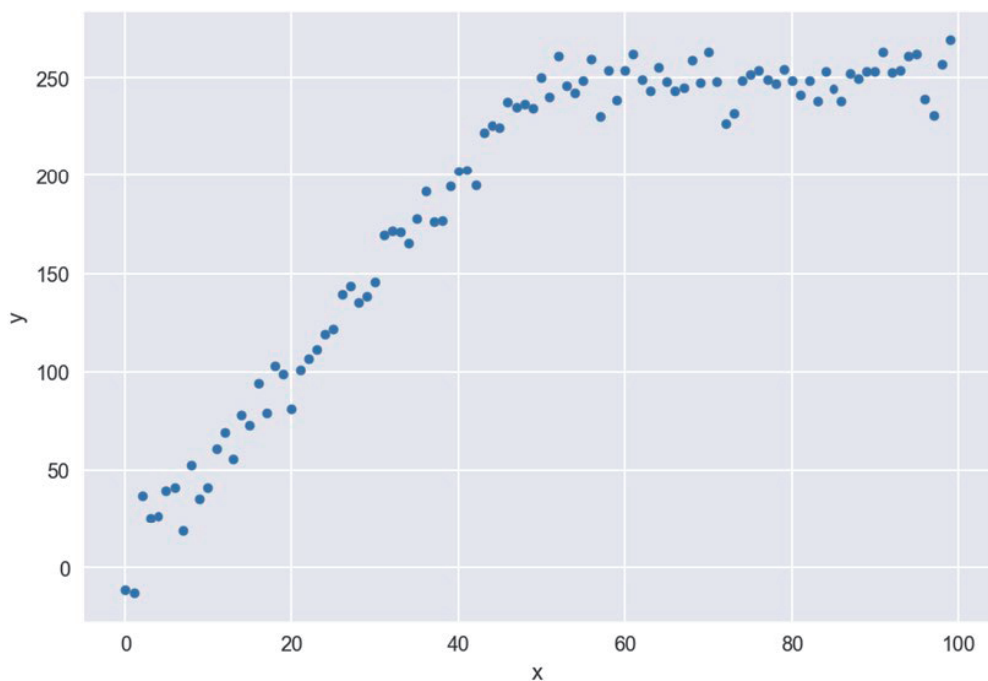
Istnieje też wiele wartych uwagi trendów nieliniowych, w tym kwadratowe, wykładnicze, homologiczne i logistyczne. Na rysunku 1.17 pokazane są niektóre z nich.

Proces przybliżonego opisu trendu za pomocą funkcji matematycznej jest nazywany **analizą regresji**. Jest ona bardzo ważna w analityce, ale jej omawianie wykracza poza zakres tej książki. Więcej informacji o analizie regresji znajdziesz w zaawansowanych książkach, takich jak *Regression Modeling Strategies* Franka E. Harrela Jr.

Choć trendy pomagają w zrozumieniu i przewidywaniu wzorców, często ważniejsze jest wykrywanie zmian w trendach. Zwykle wskazują one na ważną zmianę w mierzonym zjawisku, którą warto dodatkowo zbadać, aby ją wyjaśnić. W praktyce taką zmianą może być zmiana trendu cen akcji spółki na spadkowy po długich wzrostach. Rysunek 1.18 przedstawia przykładową zmianę trendu. Tu trend liniowy załamuje się po punkcie $x=50$.

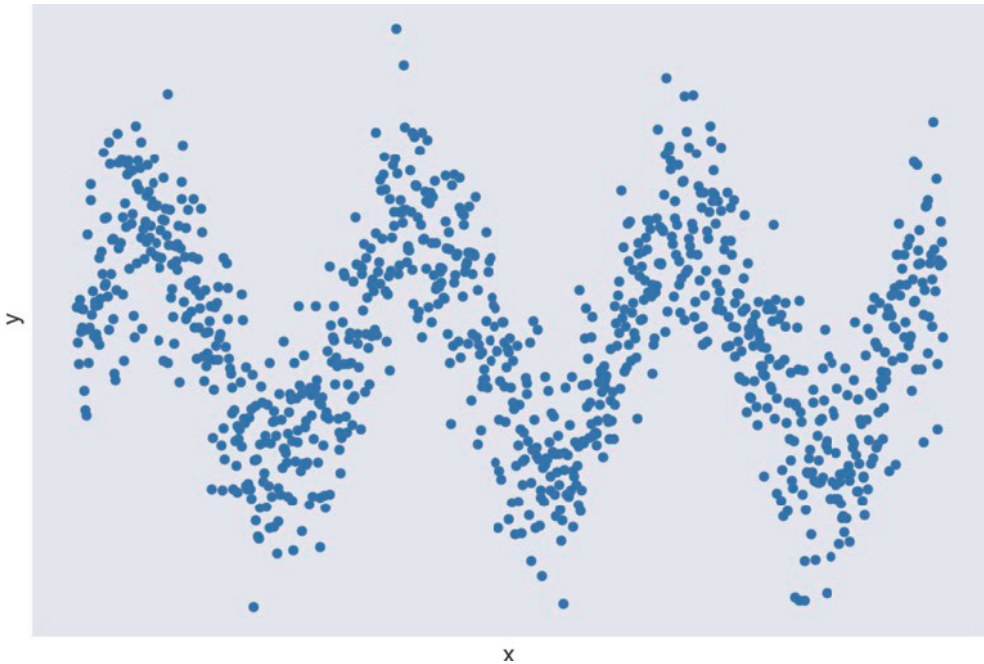


Rysunek 1.17. Inne często spotykane trendy



Rysunek 1.18. Przykład zmiany trendu

Innym wzorcem, na który często zwraca się uwagę, jest cykliczność, czyli powtarzające się wzorce w danych. Takie wzorce mogą wskazywać na to, że dwie zmienne zmieniają się cyklicznie, co może być przydatne w prognozowaniu. Rysunek 1.19 pokazuje przykład zmian cyklicznych.



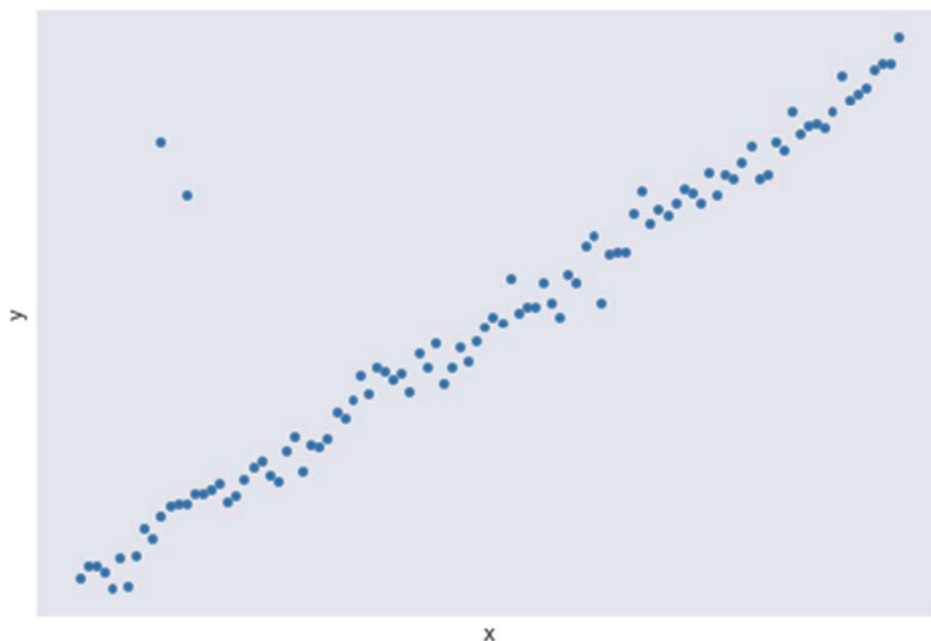
Rysunek 1.19. Przykład zmian cyklicznych

Wykresy punktowe umożliwiają też wykrywanie wartości odstających. Gdy większość punktów na wykresie znajduje się w określonym obszarze, ale niektóre są od niego znacznie oddalone, może to wskazywać, że te odległe punkty są wartościami odstającymi dla dwóch analizowanych zmiennych. W trakcie dalszych analiz dwuczynnikowych czasem warto pominąć takie punkty, aby ograniczyć szum w danych i uzyskać lepsze wnioski. Na rysunku 1.20 widoczne są punkty, które można uznać za wartości odstające.

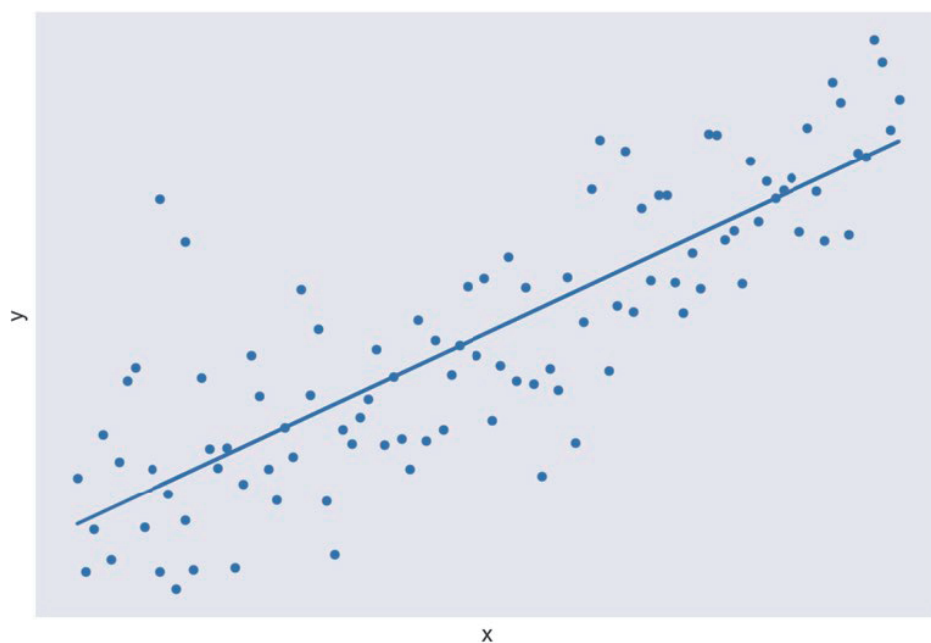
Techniki bazujące na wykresach punktowych umożliwiają profesjonalnym analitykom danych zrozumienie ogólnych trendów w danych i wykonanie pierwszych kroków na drodze do przekształcenia danych w informacje.

Współczynnik korelacji Pearsona

Jednym z najczęściej występujących trendów w trakcie analizy danych dwuczynnikowych jest trend liniowy. Często się zdarza jednak, że niektóre trendy liniowe pasują do danych lepiej, a inne gorzej. Na rysunkach 1.21 i 1.22 znajdziesz przykładowe wykresy punktowe z linią najlepszego dopasowania. Ta linia jest obliczana metodą **najmniejszych kwadratów**. Choć jej omawianie wykracza poza zakres tej książki, wiedza o tym, jak dobrze dane dwuczynnikowe pasują do trendu liniowego, pomaga zrozumieć zależność między dwiema zmiennymi.

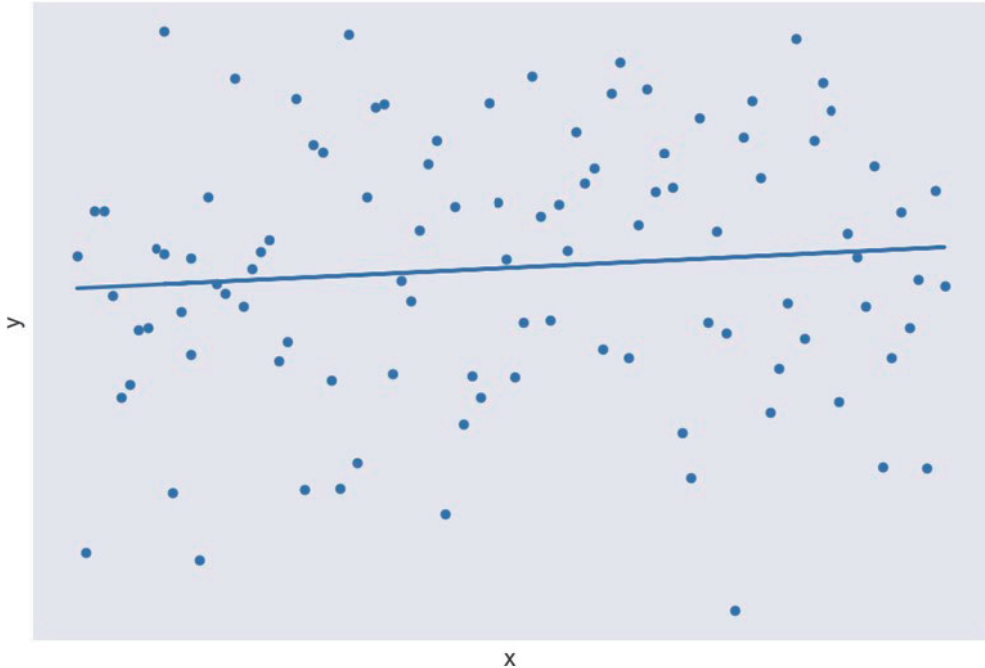


Rysunek 1.20. Wykres punktowy z dwiema wartościami odstającymi



Rysunek 1.21. Wykres punktowy z wyraźnym trendem liniowym

Kolejny rysunek ilustruje wykres punktowy ze słabym trendem liniowym.



Rysunek 1.22. Wykres punktowy ze słabym trendem liniowym

Więcej o metodzie najmniejszych kwadratów dowiesz się z podręcznika do statystyki, na przykład z książki *Statistics* Davida Freedmana, Roberta Pisaniego i Rogera Purvesa.

Jedną z metod ilościowego reprezentowania korelacji liniowej jest użycie współczynnika korelacji Pearsona. Ten współczynnik, często zapisywany za pomocą litery r , to liczba z przedziału od -1 do 1 oznaczająca, jak dobrze wykres punktowy pasuje do trendu liniowego. Do obliczania współczynnika korelacji Pearsona (r) służy wzór z rysunku 1.23.

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$$

Rysunek 1.23. Wzór na obliczanie współczynnika korelacji Pearsona

Te obliczenia są dość skomplikowane, dlatego prześledź przykład, aby przekształcić wzór na konkretne kroki.

Ćwiczenie 1.05 — obliczanie współczynnika korelacji Pearsona dla dwóch zmiennych

W tym ćwiczeniu obliczysz współczynnik korelacji Pearsona dla relacji między godzinami przepracowanymi w tygodniu (Hours Worked Per Week) a sprzedażą tygodniową (Sales Per Week (\$)). Na rysunku 1.24 widoczne są dane na temat 10 sprzedawców z salonu samochodowego firmy ZoomZoom z Houston, między innymi przychody z analizowanego tygodnia.

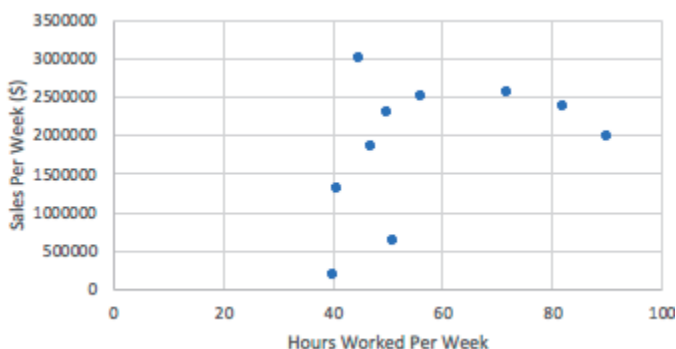
Hours Worked Per Week	Sales Per Week (\$)
40	179480,58
56	2495037,37
50	2285369,51
82	2367896,33
41	1309745,16
51	623013,69
45	2989943,37
90	1970316,24
47	1845840,39
72	2553231,33

Rysunek 1.24. Dane 10 sprzedawców z salonu firmy ZoomZoom

Zbiór danych *salesman.csv* potrzebny w tym ćwiczeniu możesz bezpośrednio pobrać z serwisu GitHub. Oto odsyłacz do katalogu *Datasets* — <https://packt.live/2B1apb3>.

Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Najpierw utwórz wykres dwóch zmiennych w Excelu, używając danych z tego scenariusza. Pomoże Ci to na ogólnym poziomie ocenić, jakiego współczynnika korelacji Pearsona możesz oczekiwać.



Rysunek 1.25. Wykres punktowy godzin przepracowanych w tygodniu i wartości tygodniowej sprzedaży

Nie widać tu silnej liniowej zależności, ale wygląda na to, że tygodniowa sprzedaż rośnie wraz z liczbą przepracowanych godzin.

2. Teraz oblicz średnią każdej zmiennej. Powinieneś uzyskać 57,40 dla zmiennej Hours Worked Per Week i 1 861 987,3 dla zmiennej Sales Per Week (\$). Jeśli nie masz pewności, jak obliczyć średnią, zajrzyj do punktu „Tendencja centralna”.
3. Teraz dla każdego wiersza oblicz cztery wartości: różnicę między wartością a średnią dla obu zmiennych oraz kwadrat tej różnicy dla obu zmiennych. Następnie oblicz iloczyn różnic. Powinieneś uzyskać tabelę wartości widoczną na rysunku 1.26.

Hours Worked Per Week	Sales Per Week (\$)	x-mean(x)	(x-mean(x))^2	y-mean(y)	(y-mean(y))^2	[x-mean(x)][y-mean(y)]
40	179480,58	-17,4	302,76	-1682506,8	2830829189251,47	29275618,62
56	2495037,37	-1,4	1,96	633049,973	400752268315,30	-886269,96
50	2285369,51	-7,4	54,76	423382,113	179252413608,34	-3133027,64
82	2367896,33	24,6	605,16	505908,933	255943848489,20	12445359,75
41	1309745,16	-16,4	268,96	-552242,24	304971488326,76	9056772,69
51	623013,69	-6,4	40,96	-1238973,7	1535055846637,32	7929431,72
45	2989943,37	-12,4	153,76	1127955,97	1272284677026,38	-13986654,07
90	1970316,24	32,6	1062,76	108328,843	11735138225,72	3531520,28
47	1845840,39	-10,4	108,16	-16147,007	260725835,06	167928,87
72	2553231,33	14,6	213,16	691243,933	477818174909,31	10092161,42

Rysunek 1.26. Obliczenia współczynnika korelacji Pearsona

4. Oblicz sumy kwadratów i sumę iloczynów różnic. Powinieneś otrzymać 2812,40 dla zmiennej Hours Worked Per Week (x), 7 268 904 222 394,36 dla zmiennej Sales Per Week (\$) (y) i 54 492 841,32 dla iloczynu różnic.
5. Oblicz pierwiastki kwadratowe sum różnic. Powinieneś uzyskać 53,03 dla zmiennej Hours Worked Per Week (x) i 2 696 090,54 dla zmiennej Sales Per Week (\$) (y).
6. Podstaw te wartości do wzoru z rysunku 1.27. Otrzymasz wynik 0,38. Obliczenia są pokazane na rysunku 1.27.

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} = \frac{54492841,32}{(53,03) * (2696090,54)} \approx 0,38$$

Rysunek 1.27. Gotowe obliczenia współczynnika korelacji Pearsona

W tym ćwiczeniu zobaczyłeś, jak obliczyć współczynnik korelacji Pearsona dwóch zmiennych. Po zastosowaniu wzoru otrzymałeś końcowy wynik 0,38.

Interpretowanie i analizowanie współczynnika korelacji

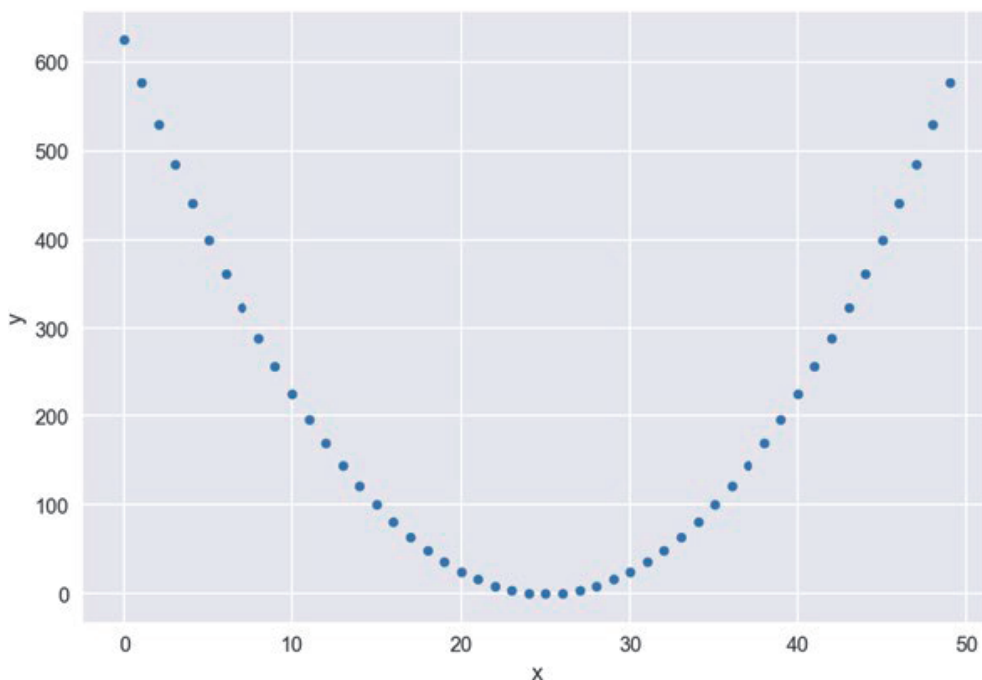
Ręczne obliczanie współczynnika korelacji może być bardzo skomplikowane. Zwykle lepiej jest obliczać go z użyciem komputera. W rozdziale 2., „Przygotowywanie danych za pomocą SQL-a”, zobaczysz, że współczynnik korelacji Pearsona można obliczyć za pomocą SQL-a.

Aby zinterpretować współczynnik korelacji Pearsona, porównaj uzyskaną wartość z tabelą z rysunku 1.28. Im wynik jest bliższy zeru, tym korelacja jest słabsza. Im wyższa wartość bezwzględna współczynnika korelacji Pearsona, tym bardziej prawdopodobne jest, że punkty pasują do linii prostej.

Wartość korelacji	Interpretacja
$-1,0 \leq r \leq -0,7$	Bardzo mocna korelacja ujemna
$-0,7 \leq r \leq -0,4$	Mocna korelacja ujemna
$-0,4 \leq r \leq -0,2$	Umiarkowana korelacja ujemna
$-0,2 \leq r \leq 0,2$	Słaba lub nieistniejąca korelacja
$0,2 \leq r \leq 0,4$	Umiarkowana korelacja dodatnia
$0,4 \leq r \leq 0,7$	Mocna korelacja dodatnia
$0,7 \leq r \leq 1,0$	Bardzo mocna korelacja dodatnia

Rysunek 1.28. Interpretowanie współczynnika korelacji Pearsona

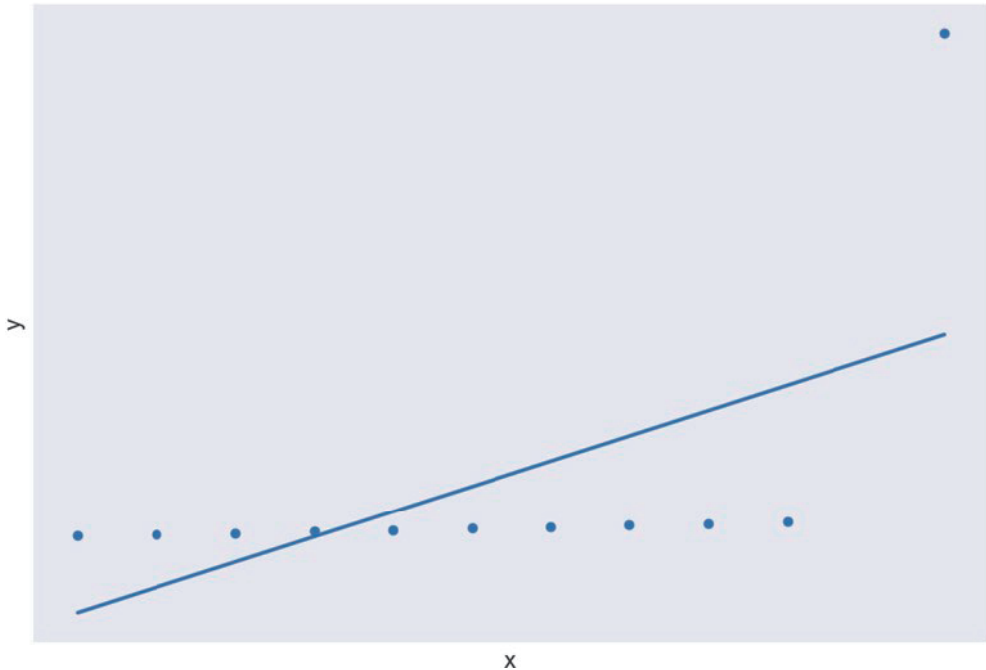
W trakcie analizowania współczynnika korelacji trzeba uwzględnić kilka kwestii. Pierwsza z nich dotyczy tego, że współczynnik korelacji mierzy, jak dobrze dwie zmienne pasują do trendu liniowego. Dwie zmienne mogą być ściśle powiązane, ale mieć stosunkowo niski współczynnik korelacji Pearsona. Przyjrzyj się na przykład punktom z rysunku 1.29. Jeśli obliczysz współczynnik dla tych dwóch zmiennych, otrzymasz wynik $-0,08$. Jednak krzywa wskazuje na bardzo silną zależność kwadratową. Dlatego gdy sprawdzasz współczynniki korelacji danych dwuczynnikowych, pamiętaj, że zależność między dwiema zmiennymi może być nieliniowa.



Rysunek 1.29. Silna nieliniowa zależność o niskim współczynniku korelacji

Innym ważnym aspektem jest liczba punktów używanych do obliczania korelacji. Wystarczą dwa punkty do zdefiniowania linii prostej. Dlatego mniejsza liczba punktów może skutkować otrzymaniem wysokiego współczynnika korelacji, który jednak nie zawsze zostanie utrzymany po dodaniu większej liczby danych. Zgodnie z ogólną regułą współczynniki korelacji obliczone dla mniej niż 30 punktów danych nie są wiarygodne. Do obliczania korelacji należy używać jak największej liczby dobrych punktów danych.

Zwróć uwagę na wyrażenie „dobre punkty danych”. Jednym z powtarzających się motywów w tym rozdziale jest negatywny wpływ wartości odstających na różne statystyki. W danych dwuczynnikowych wartości odstające mogą wpływać na współczynnik korelacji. Przyjrzyj się wykresowi z rysunku 1.30. Widocznych jest tam 11 punktów, z których jeden to wartość odstająca. Powoduje on, że współczynnik korelacji Pearsona (r) dla tych danych spada do 0,59. Jednak bez tego punktu współczynnik jest równy 1,0. Dlatego należy starannie usunąć wartości odstające, zwłaszcza wtedy, gdy ich liczba jest mała.



Rysunek 1.30. Obliczanie r dla wykresu punktowego z wartością odstającą

Ważnym problemem związanym z obliczaniem korelacji jest błędne przyjmowanie, że korelacja oznacza związek przyczynowo-skutkowy. Występowanie wysokiej korelacji między x i y nie oznacza, że x powoduje y . Przyjrzyj się zależności między liczbą przepracowanych godzin a wartością sprzedanych dodatków. Przyjmij, że po dodaniu punktów danych okazuje się, że korelacja między tymi dwiema zmiennymi wynosi 0,5. Wielu początkujących analityków danych i doświadczonych menedżerów założy, że większa liczba przepracowanych godzin skutkuje wyższą sprzedażą, po czym zaczną zmuszać sprzedawców do nieustannej pracy.

Wprawdzie możliwe jest, że większa liczba godzin pracy skutkuje wyższą sprzedażą, jednak wysoki współczynnik korelacji nie wystarczy jako dowód tej tezy.

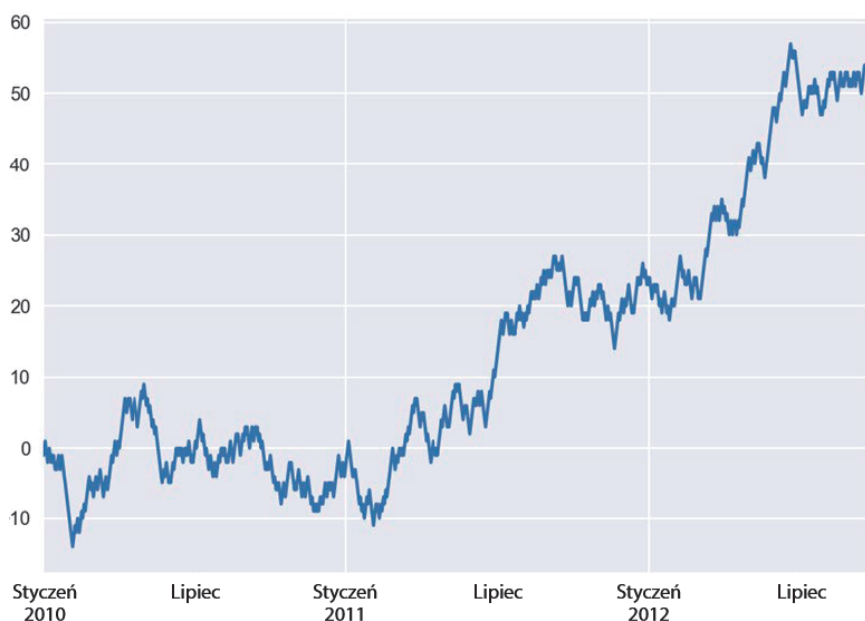
Możliwe nawet, że związek przyczynowy działa w drugą stronę: większa liczba transakcji wymaga więcej pracy papierkowej, co przekłada się na większą liczbę godzin w biurze. W tym scenariuszu większa liczba godzin nie powoduje wyższej sprzedaży.

Jeszcze inna możliwość to występowanie trzeciej zmiennej wpływającej na zależność między dwiema zmiennymi. Możliwe, że doświadczeni sprzedawcy pracują więcej godzin i lepiej radzą sobie ze sprzedażą. Dlatego prawdziwym powodem wyższej sprzedaży jest doświadczenie, z czego wynika zalecenie zatrudnienia większej grupy doświadczonych sprzedawców.

Profesjonalny analityk powinien unikać pułapek takich jak mylenie korelacji ze związkiem przyczynowym. Musisz krytycznie zastanowić się nad wszystkimi możliwościami, jakie wpływają na uzyskane wyniki.

Dane w postaci szeregów czasowych

Jednym z najważniejszych rodzajów analiz dwuczynnikowych jest analiza szeregów czasowych. **Szereg czasowy** reprezentuje relację dwuczynnikową, w której na osi x przedstawiony jest czas. Przykład szeregu czasowego jest pokazany na rysunku 1.31. Widoczny jest tam szereg czasowy obejmujący okres od stycznia 2010 roku do końca 2012 roku. Choć początkowo nie jest to oczywiste, daty i czas mają charakter ilościowy. Zrozumienie zmian zachodzących w czasie jest jednym z najważniejszych rodzajów analiz przeprowadzanych w firmach i zapewnia cenne informacje na temat kontekstu prowadzenia działalności.



Rysunek 1.31. Przykładowy szereg czasowy

Wszystkie wzorce opisane w poprzednim podrozdziale występują także w szeregach czasowych. Szeregi czasowe są ważne w firmach, ponieważ mogą wskazywać na czas wystąpienia zmian. Punkty w czasie mogą pomóc w ustaleniu przyczyn tych zmian.

Teraz przyjrzyj się niewielkiemu zbiorowi danych. Posłuży on do pokazania, jak przeprowadzać proste analizy statystyczne.

Zadanie 1.02 — eksplorowanie danych sprzedażowych z salonu samochodowego

W tym zadaniu przyjrzyj się dokładnie zbiorowi danych, wykorzystując statystykę. Wyobraź sobie, że jesteś analitykiem w ZoomZoom, firmie specjalizującej się w sprzedaży samochodów elektrycznych, i przeprowadzasz wysokopoziomowe analizy rocznej sprzedaży w salonach z całego kraju. Dane znajdują się w pliku *.csv*.

1. Otwórz dokument *dealerships.csv* w arkuszu kalkulacyjnym lub edytorze tekstu. Plik ten znajdziesz w katalogu *Datasets* w repozytorium w serwisie GitHub.
2. Przygotuj rozkład liczby kobiet zatrudnionych w poszczególnych salonach.
3. Ustal średnią i medianę dla rocznej sprzedaży salonu.
4. Oblicz odchylenie standardowe sprzedaży.
5. Czy dane z któregoś salonu są odstające? Wyjaśnij, dlaczego tak uważasz.
6. Oblicz kwantyle na podstawie rocznej sprzedaży.
7. Oblicz współczynnik korelacji rocznej sprzedaży z liczbą zatrudnionych kobiet i zinterpretuj wyniki.

Rozwiązanie tego zadania znajdziesz w „Dodatku”.

8. W tym zadaniu dane są kompletne. Co jednak zrobić, jeśli jest inaczej? Jak radzić sobie z brakiem danych? Następny punkt pomoże Ci zrozumieć, co robić w takich sytuacjach.

Praca z niepełnymi danymi

We wszystkich dotychczasowych przykładach zbiory danych były świetnie oczyszczone. Jednak w praktyce zbiory danych prawie nigdy nie są tak prawidłowe. Jednym z wielu problemów, z jakimi trzeba sobie radzić w trakcie pracy z danymi, są brakujące wartości. Szczegóły przygotowywania danych omówiliśmy w rozdziale 2., „Przygotowywanie danych za pomocą SQL-a”. Tu omawiamy kilka strategii, które możesz zastosować do radzenia sobie z niepełnymi danymi. Oto kilka możliwości:

- **Usuwanie wierszy** — jeśli danych brakuje w bardzo niewielkiej części wierszy (w mniej niż 5% zbioru danych), najprostszym rozwiązaniem może być usunięcie niepełnych punktów danych. Nie powinno to mieć istotnego wpływu na wyniki.

- **Wykorzystanie średniej, mediany lub wartości modalnej** — jeżeli wartość zmiennej jest nieobecna w od 5% do 25% punktów danych, możesz obliczyć średnią, medianę lub wartość modalną dla danej kolumny i uzupełnić luki otrzymanym wynikiem. Może to wprowadzać niewielką tendencyjność w obliczeniach, ale pozwala przeprowadzić więcej analiz bez usuwania cennych danych.
- **Wykorzystanie regresji** — jeśli jest to możliwe, przygotuj i zastosuj model do oszacowania brakujących wartości. Może to przekraczać możliwości większości analityków danych, jeśli jednak pracujesz ze specjalistą od data science, rozwiązanie to może być wykonalne.
- **Usuwanie zmiennej** — nie da się analizować nieistniejących danych. Jeśli danych jest niewiele, a w większości obserwacji brakuje wartości określonych zmiennej, lepszym rozwiązaniem może być usunięcie tej zmiennej niż przyjmowanie zbyt wielu założeń i dochodzenie do błędnych wniosków.

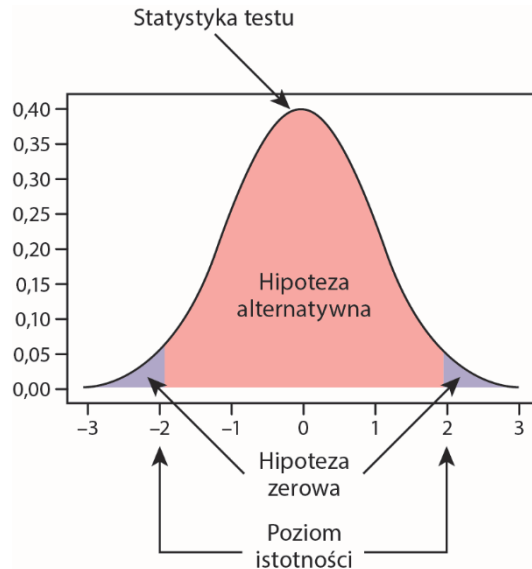
Przekonasz się, że analiza danych często jest bardziej sztuką niż nauką. Dotyczy to między innymi pracy z niekompletnymi danymi. Gdy nabierzesz doświadczenia, będziesz znać kombinacje strategii, które sprawdzają się w różnych scenariuszach.

Testy istotności statystycznej

Następną czynnością przydatną w trakcie analiz danych jest sprawdzanie istotności statystycznej. Analitycy często porównują cechy statystyczne dwóch grup (lub tej samej grupy przed wprowadzeniem zmiany i po niej). Różnice między grupami mogą oczywiście wynikać z przypadku.

Ta technika jest stosowana na przykład w marketingowych testach A/B. Firmy często testują dwa rodzaje stron wejściowych produktu i mierzą **współczynnik klikalności** (ang. *click-through rate* — CTR). Może się okazać, że współczynnik klikalności dla wersji A strony wejściowej wynosi 10%, a dla wersji B — 11%. Czy to oznacza, że wersja B jest o 10% lepsza od wersji A? A może różnica wynika wyłącznie z przypadku i każdego dnia może być inna? Testy statystyczne pomagają odpowiedzieć na takie pytania.

W testach istotności statystycznej trzeba uwzględnić kilka ważnych elementów (rysunek 1.32). Przede wszystkim jest to **statystyka testu**. Może to być stosunek, średnia, różnica między grupami lub rozkład. Następnym niezbędnym elementem jest **hipoteza zerowa**, która zakłada, że zaobserwowane wyniki uzyskano przypadkowo. Potrzebna jest też **hipoteza alternatywna**, zgodnie z którą uzyskane wyniki nie są dziełem przypadku. Należy też ustalić **poziom istotności**, czyli wartość, jaką musi mieć statystyka testu, aby można było uznać, że hipoteza zerowa nie wyjaśnia różnic między grupami. Te cztery elementy występują we wszystkich testach istotności statystycznej, a różnice między testami wynikają ze sposobów obliczania tych komponentów.



Rysunek 1.32. Elementy testów istotności statystycznej

Często używane testy istotności statystycznej

Oto kilka często stosowanych testów istotności statystycznej:

- **Test Z dla dwóch próbek** — jest to test określający, czy średnie dwóch próbek różnią się od siebie. Ten test bazuje na założeniu, że obie próbki pochodzą z rozkładu normalnego o znanym odchyleniu standardowym dla populacji.
- **Test T dla dwóch próbek** — jest to test określający, czy średnie dwóch próbek różnią się od siebie. Stosuje się go, gdy albo próbki są zbyt małe (poniżej 30 punktów danych na próbkę), albo nieznane jest odchylenie standardowe dla populacji. Także tu przyjmuje się, że obie próbki pochodzą z populacji o rozkładzie normalnym.
- **Test zgodności chi-kwadrat (inaczej test Pearsona)** — ten test określa, czy rozkład punktów danych między kategorie różni się od oczekiwanego przypadkowego rozkładu. Służy przede wszystkim do sprawdzania, czy proporcje w testach (na przykład w testach A/B) różnią się od proporcji, jakie można uzyskać przypadkowo.

Więcej o istotności statystycznej dowiesz się z podręcznika do statystyki, na przykład z książki *Statistics* Davida Freedmana, Roberta Pisaniego i Rogera Purvesa.

W następnym podrozdziale poznasz podstawy relacyjnych baz danych i SQL-a. Dalej omawiamy typy danych, polecenia i kwerendy.

Relacyjne bazy danych i SQL

Relacyjna baza danych to baza, w której używany jest **relacyjny model** danych. Model relacyjny został wymyślony w 1970 roku przez Edgara F. Codd'a i cechuje się tym, że dane są uporządkowane za pomocą relacji jako zbiory krotek. Każda krotka obejmuje zestaw atrybutów, które ją opisują. Wyobraź sobie relację, w której każda krotka reprezentuje klienta. Krotki mają tu atrybuty opisujące jednego klienta — na przykład jego nazwisko, imię i wiek w formacie (Sawicki, Jan, 27). Do unikatowego identyfikowania krotki w relacji służy **klucz** w postaci jednego atrybutu lub zestawu atrybutów. W modelu relacyjnym możliwe jest wykonywanie operacji logicznych na relacjach.

W bazie relacyjnej relacje mają zwykle postać tabel, podobnych jak w arkuszach kalkulacyjnych Excela. Każdy wiersz tabeli to krotka, a atrybuty są reprezentowane w kolumnach tabeli. Choć technicznie nie jest to wymagane, większość tabel w relacyjnych bazach danych ma kolumnę nazywaną **kluczem głównym**, która w unikatowy sposób identyfikuje wiersz bazy. Każda kolumna ma też **typ danych**, opisujący rodzaj danych przechowywanych w tej kolumnie. Tabele są zwykle łączone w kolekcje w bazach nazywane **schematami**. Wczytywanie tabel odbywa się w ramach **procesu ETL** (od ang. *extract, transform, load*, czyli pobieranie, przetwarzanie i wczytywanie).

W kwerendach tabele przeważnie podaje się w formacie `[schemat].[tabela]`. Na przykład tabelę `products` ze schematu `analytics` zwykle będziesz podawać jako `analytics.products`. Istnieje też specjalny schemat o nazwie **public**. Jest to schemat domyślny, używany, gdy programista nie poda bezpośrednio schematu. Na przykład zapisy `public.products` i `products` oznaczają tę samą tabelę.

Oprogramowanie służące do zarządzania relacyjnymi bazami danych na komputerze to **system zarządzania relacyjnymi bazami danych (SZRBD)**. SQL to język stosowany przez użytkowników SZRBD do dostępu do relacyjnych baz danych i interakcji z nimi.

Prawie wszystkie relacyjne bazy danych oparte na SQL-u na jakimś poziomie są niezgodne z modelem relacyjnym. Na przykład nie każda tabela ma klucz. Ponadto model relacyjny nie dopuszcza powtarzających się wierszy, ale w relacyjnych bazach mogą one występować. Te rozbieżności są jednak niewielkie i dla zdecydowanej większości Czytelników tej książki nie mają żadnego znaczenia.

Wady i zalety baz SQL-owych

Od czasu wprowadzenia bazy danych Oracle w 1979 roku SQL stał się standardowym językiem do pracy z danymi w prawie wszystkich zastosowaniach informatycznych. Jest to uzasadnione. Bazy SQL-owe mają mnóstwo zalet, dlatego w wielu zastosowaniach wybiera się je prawie automatycznie. Oto te zalety:

- **Intuicyjność** — relacje w postaci tabel to często używana struktura danych zrozumiała prawie dla każdego. Dlatego praca z bazami relacyjnymi i myślenie o nich są znacznie prostsze niż w innych modelach.
- **Wydajność** — dzięki technice normalizacji bazy relacyjne umożliwiają reprezentowanie danych bez niepotrzebnego ich duplikowania. Dlatego pozwalają reprezentować dużą ilość informacji za pomocą niewielkiej ilości pamięci. To ograniczone zużycie pamięci zmniejsza też koszty operacyjne, dlatego przetwarzanie dobrze zaprojektowanych baz relacyjnych jest szybkie.
- **Deklaratywność** — SQL jest językiem deklaratywnym, co oznacza, że gdy piszesz kod, wystarczy poinformować komputer, jakie dane są potrzebne, a SZRBD zadba o ustalenie, jak wykonać kod w SQL-u. Nigdy nie musisz martwić się o przekazywanie komputerowi, jak ma uzyskać dostęp do danych z tabeli i je pobrać.
- **Niezawodność** — większość popularnych SQL-owych baz danych jest zgodna z właściwościami ACID (ang. *atomicity, consistency, isolation, durability*, czyli atomowość, spójność, izolacja i trwałość), co gwarantuje poprawność danych nawet po awariach sprzętu.

Bazy SQL-owe mają też jednak kilka wad. Oto one:

- **Stosunkowo niska specyficzność** — choć SQL jest deklaratywny, jego możliwości często są ograniczone do mechanizmów, jakie już zostały w nim zaprogramowane. Mimo że większość popularnych SZRBD jest nieustannie rozbudowywana o nowe możliwości, trudno jest pracować ze strukturami danych i algorytmami, które nie są zaprojektowane w używanym SZRBD.
- **Ograniczona skalowalność** — bazy SQL-owe są wysoce niezawodne, ma to jednak swoje koszty. Gdy przechowywana ilość informacji się podwaja, podwaja się też koszt zasobów. W przypadku przechowywania bardzo dużych zbiorów informacji lepszym wyborem mogą być inne magazyny danych, na przykład bazy typu NoSQL.
- **Niedopasowanie impedancji** — chociaż tabele to bardzo intuicyjna struktura danych, nie zawsze są najlepszym formatem do reprezentowania obiektów w komputerze. Wynika to przede wszystkim z tego, że obiekty często mają atrybuty z relacjami wiele do wielu. Na przykład klient firmy może kupić wiele produktów, a każdy produkt może zostać zakupiony przez wielu klientów. W obiektach w komputerze produkty można łatwo zapisać jako atrybut `lista` w obiekcie `klient`. Jednak w znormalizowanej bazie danych produkty klienta zapewne trzeba będzie zapisać za pomocą trzech różnych tabel, z których każda będzie wymagać aktualizacji po nowym zakupie, zwrocie lub wycofaniu ze sprzedaży.

Podstawowe typy danych w SQL-u

Wcześniej wspomnieliśmy, że każda kolumna w tabeli ma typ danych. Tu opisujemy podstawowe typy danych.

Typy liczbowe

Liczbowe typy danych reprezentują liczby. Na rysunku 1.33 pokazane są wybrane spośród popularnych typów liczbowych.

Nazwa	Zajmowana pamięć	Opis	Zakres
smallint	2 bajty	Małe liczby całkowite	Od -32 768 do +32 767
integer	4 bajty	Typowy wybór dla liczb całkowitych	Od -2 147 483 648 do +2 147 483 647
bigint	8 bajtów	Duże liczby całkowite	Od -9 223 372 036 854 775 808 do +9 223 372 036 854 775 807
decimal	Zmienna	Precyzja określona przez użytkownika, dokładne wartości	Do 131 072 cyfr przed przecinkiem; do 16 383 cyfr po przecinku
numeric	Zmienna	Precyzja określona przez użytkownika, dokładne wartości	Do 131 072 cyfr przed przecinkiem; do 16 383 cyfr po przecinku
real	4 bajty	Zmienna precyzja, niedokładne wartości	Precyzja do 6 cyfr po przecinku
double	8 bajtów	Zmienna precyzja, niedokładne wartości	Precyzja do 15 cyfr po przecinku
smallserial	2 bajty	Małe automatycznie zwiększane liczby całkowite	Od 1 do 32 767
serial	4 bajty	Automatycznie zwiększane liczby całkowite	Od 1 do 2 147 483 647
bigserial	8 bajtów	Duże automatycznie zwiększane liczby całkowite	Od 1 do 9 223 372 036 854 775 807

Rysunek 1.33. Podstawowe liczbowe typy danych

Typy znakowe

Znakowe typy danych przechowują informacje tekstowe. Na rysunku 1.34 znajdziesz opis znakowych typów danych.

Nazwa	Opis
character varying(n), varchar(n)	Zmienna ograniczona długość
character(n), char(n)	Stała długość, dopełnianie spacjami
text	Zmienna nieograniczona długość

Rysunek 1.34. Podstawowe znakowe typy danych

W systemie PostgreSQL i wielu innych bazach SQL-owych na zapleczu dla wszystkich znakowych typów danych używana jest ta sama struktura danych. Większość programistów nie stosuje obecnie typu `char(n)`.

Typ logiczny

Logiczny typ danych służy do reprezentowania wartości `True` lub `False`. W tabeli na rysunku 1.35 opisane są wartości traktowane jako logiczne w kwerendach z kolumną typu logicznego.

Wartość logiczna	Akceptowane wartości
True	t, true, y, yes, on, 1
False	f, false, n, no, off, 0

Rysunek 1.35. Akceptowane wartości logiczne

Choć akceptowane są wszystkie wymienione wartości, zgodnie z najlepszymi praktykami należy używać wartości `True` i `False`. W kolumnach logicznych dopuszczalne mogą być także wartości `NULL`.

Daty i godziny

Typy danych z datą i godziną służą do zapisywania informacji związanych z czasem, na przykład dat i godzin. Na rysunku 1.36 pokazane są przykładowe typy danych z datą i godziną.

Nazwa	Wielkość	Opis
Timestamp without timezone	8 bajtów	Data i czas bez strefy czasowej
Timestamp with timezone	8 bajtów	Data i czas ze strefą czasową
date	4 bajty	Data (bez czasu)
Time without timezone	8 bajtów	Czas (bez daty)
Time with timezone	12 bajtów	Sam czas ze strefą czasową
interval	16 bajtów	Przedział czasu

Rysunek 1.36. Popularne typy danych z datą i godziną

Ten typ danych omawiamy dokładniej w rozdziale 5., „Analizy z wykorzystaniem złożonych typów danych”.

Struktury danych — format JSON i tablice

Wiele wersji SQL-a udostępnia też struktury danych takie jak obiekty w formacie **JSON** (ang. *JavaScript Object Notation*) i tablice. Tablice to listy danych zwykle zapisywane jako elementy w nawiasie klamrowym. Oto przykładowa tablica: ['kot', 'pies', 'koń']. Obiekt w formacie JSON to zbiór par klucz – wartość rozdzielonych przecinkami i umieszczonych w nawiasie klamrowym. Na przykład {'imię': 'Borys', 'wiek': 27, 'miasto': 'Nowy Sącz'} to poprawny obiekt w formacie JSON. Tego rodzaju struktury danych często występują w aplikacjach, a możliwość stosowania ich w bazach danych ułatwia wykonywanie wielu analiz.

Struktury danych opisujemy szczegółowo w rozdziale 5., „Analizy z wykorzystaniem złożonych typów danych”.

Przyjrzyj się teraz podstawowym operacjom z użyciem SQL-a w SZRBD.

Wczytywanie tabel — kwerenda SELECT

Najczęściej wykonywaną operacją w bazach danych jest wczytywanie danych z bazy. Prawie zawsze używane jest do tego słowo kluczowe **SELECT**.

Podstawowa budowa i działanie kwerendy SELECT

Na ogólnym poziomie kwerendę można podzielić na pięć części:

- **Operacja** — pierwsza część kwerendy opisuje, jaka operacja zostanie wykonana. Tu używane jest słowo **SELECT**, po którym podawane są nazwy kolumn i funkcji.
- **Dane** — następną część kwerendy stanowią dane. Podaje się je jako słowo kluczowe **FROM**, po którym następuje nazwa tabeli (lub kilku tabel) wraz z zarezerwowanymi słowami kluczowymi określającymi, jakie dane należy uwzględnić na potrzeby wykonywania obliczeń, filtrowania i pobierania.
- **Warunek** — ta część kwerendy filtruje dane, aby uwzględniane były tylko te wiersze, które spełniają podany warunek, zwykle określony w klauzuli **WHERE**.
- **Grupowanie** — ten etap bazuje na specjalnej klauzuli, która pobiera wiersze ze źródła danych i łączy je ze sobą na podstawie klucza z klauzuli **GROUP BY**, a następnie oblicza wynik na podstawie wartości z wszystkich wierszy o tym samym kluczu. Ten krok jest opisany szczegółowo w rozdziale 3., „Agregacja i funkcje okna”.
- **Przetwarzanie końcowe** — na tym etapie kwerendy wynikowe dane są pobierane i przetwarzane (sortowanie i uwzględnianie ograniczeń), często z użyciem słów kluczowych takich jak **ORDER BY** i **LIMIT**.

Oto etapy działania kwerendy SELECT:

1. Tworzenie źródła danych. Podaj jedną tabelę lub wskaż kilka tabel, które zostaną połączone w jedną dużą tabelę.
2. Filtrowanie tabeli z dużego źródła danych utworzonego w kroku 1.. W tym celu sprawdzane jest, które wiersze są zgodne z klauzulą WHERE.
3. Obliczanie wartości na podstawie kolumn ze źródła danych z kroku 1. Jeśli używana jest klauzula GROUP BY, wiersze są dzielone na grupy, po czym obliczane są statystyki zbiorcze dla każdej grupy. W przeciwnym razie zwracana jest kolumna lub wartość obliczona przez wykonanie funkcji na jednej kolumnie lub kilku kolumnach.
4. Pobranie zwróconych wierszy i uporządkowanie ich zgodnie z kwerendą.

Aby przeanalizować te kroki, przyjrzyj się typowej kwerendzie i prześledź opisany proces:

```
SELECT
  first_name
FROM
  customers
WHERE
  state='AZ'
ORDER BY
  first_name;
```

Ta kwerenda działa w następujący sposób:

1. Punktem wyjścia jest tabela customers.
2. Ta tabela jest filtrowana; uwzględniane są tylko te wiersze, w których kolumna state ma wartość 'AZ'.
3. Z przefiltrowanej tabeli pobierana jest kolumna first_name.
4. Wartości z kolumny first_name są sortowane alfabetycznie.

Pokazaliśmy tu, jak rozbić kwerendę na serię kroków do wykonania przez bazę danych. Teraz poznasz słowa kluczowe używane w kwerendach SELECT i wzorce takich kwerend.

Podstawowe słowa kluczowe w kwerendach SELECT

W trakcie pisania kwerendy SELECT możesz stosować wiele słów kluczowych. Na początek poznaj słowa kluczowe SELECT i FROM.

Instrukcje SELECT i FROM

Najbardziej podstawowa wersja kwerendy SELECT ma postać SELECT...FROM <nazwa_tabeli>;. Ta kwerenda pozwala pobrać dane z jednej tabeli. Na przykład jeśli chcesz pobrać wszystkie dane z tabeli products z przykładowej bazy, użyj następującej kwerendy:

```
SELECT
  *
FROM
  products;
```

Ta kwerenda pobiera wszystkie dane z bazy. Użyty tu symbol * to skrót pozwalający pobrać z bazy wszystkie kolumny. Średnik (;) informuje komputer, że dotarł do końca kwerendy, podobnie jak kropka kończy zwykle zdania. Należy zauważyć, że wiersze są tu zwracane w nieokreślonej kolejności. Jeśli chcesz pobrać w kwerendzie tylko określone kolumny, zastąp gwiazdkę (*) nazwami potrzebnych kolumn. Kolumny należy rozdzielić przecinkami i podać w kolejności, w jakiej kwerenda ma zwracać dane. Na przykład jeśli chcesz otrzymać kolumnę `product_id`, po której następuje kolumna `model` (obie z tabeli `products`), użyj tej kwerendy:

```
SELECT product_id, model
FROM products;
```

Jeśli chcesz, aby najpierw zwracana była kolumna `model`, a następnie kolumna `product_id`, zapisz kwerendę tak:

```
SELECT model, product_id
FROM products;
```

W następnym punkcie poznasz klauzulę `WHERE`.

Klauzula WHERE

Klauzula `WHERE` pozwala dodać warunek ograniczający ilość zwracanych danych. Wszystkie wiersze zwracane przez kwerendę `SELECT` z klauzulą `WHERE` spełniają warunek z tej klauzuli. W kwerendzie `SELECT` klauzula `WHERE` zwykle jest umieszczana po klauzuli `FROM`.

Warunkiem w klauzuli `WHERE` zazwyczaj jest wyrażenie logiczne, które dla każdego wiersza przyjmuje wynik `True` lub `False`. Gdy używane są kolumny liczbowe, w wyrażeniu logicznym można stosować operatory równości, większości lub mniejszości, aby porównywać wartość z kolumny z podaną wartością.

Oto ilustrujący to przykład. Załóżmy, że chcesz sprawdzić nazwy modeli z 2014 roku z przykładowego zbioru danych. W tym celu możesz napisać następującą kwerendę:

```
SELECT
    model
FROM
    products
WHERE
    year=2014;
```

Z następnego punktu dowiesz się, jak stosować w kwerendach klauzule `AND` i `OR`.

Klauzule AND i OR

W ostatniej kwerendzie używany był tylko jeden warunek. Często analityka interesują dane spełniające kilka warunków. W tym celu można połączyć zestaw wyrażeń za pomocą klauzul `AND` i `OR`.

Przyjrzyj się ilustrującemu to przykładowi. Wyobraź sobie, że chcesz zwrócić modele, które nie tylko zostały zbudowane w 2014 roku, ale dodatkowo mają sugerowaną cenę detaliczną producenta niższą niż 1000 dolarów. Możesz napisać następującą kwerendę:

```
SELECT
  model
FROM
  products
WHERE
  year=2014
  AND msrp<=1000;
```

Teraz założmy, że chcesz zwrócić dowolne modele, które zostały wypuszczone w 2014 roku lub których typ produktu to automobile. Umożliwia to poniższa kwerenda:

```
SELECT
  model
FROM
  products
WHERE
  year=2014
  OR product_type='automobile';
```

Gdy używasz więcej niż jednej klauzuli AND lub OR, dodaj nawiasy, aby odpowiednio oddzielić i rozmieścić wyrażenia logiczne. To gwarantuje, że kwerenda będzie działać zgodnie z oczekiwaniami i że będzie czytelna. Jeśli na przykład chcesz pobrać wszystkie produkty z lat od 2014 do 2016, a także produkty typu 'scooter', możesz użyć następującej kwerendy:

```
SELECT
  *
FROM
  products
WHERE
  year>2014
  AND year<2016
  OR product_type='scooter';
```

Jednak aby klauzula WHERE była bardziej czytelna, lepiej jest użyć następującego zapisu:

```
SELECT
  *
FROM
  products
WHERE
  (year>2014 AND year<2016)
  OR product_type='scooter';
```

W następnym punkcie poznasz klauzule IN i NOT IN.

Klauzule IN i NOT IN

Wcześniej wspomnieliśmy, że w wyrażeniach logicznych można używać znaku równości, aby określić, że kolumna musi się równać określonej wartości. Co jednak zrobić, jeśli chcesz zwrócić wiersze, w których kolumna równa się jednej z grup wartości? Załóżmy, że interesują Cię wszystkie modele z lat 2014, 2016 i 2019. Możesz zapisać taką kwerendę tak:

```
SELECT
    model
FROM
    products
WHERE
    year = 2014
    OR year = 2016
    OR year = 2019;
```

Jednak ten kod jest długi i żmudny w pisaniu. Za pomocą klauzuli IN możesz zapisać go tak:

```
SELECT
    model
FROM
    products
WHERE
    year IN (2014, 2016, 2019);
```

Tę wersję znacznie łatwiej jest zapisać, a także zrozumieć.

Możesz też zastosować klauzulę NOT IN, aby zwrócić wszystkie wartości poza podanymi na liście. Jeśli chcesz otrzymać wszystkie towary, które nie zostały wyprodukowane w latach 2014, 2016 lub 2019, możesz użyć tej kwerendy:

```
SELECT
    model
FROM
    products
WHERE
    year NOT IN (2014, 2016, 2019);
```

W następnym punkcie dowiesz się, jak używać w kwerendach klauzuli ORDER BY.

Klauzula ORDER BY

Wcześniej wspomnieliśmy, że jeśli kwerenda SQL-owa nie otrzyma konkretnych instrukcji, uporządkuje wiersze tak, jak zostały one znalezione przez bazę danych. W wielu scenariuszach jest to akceptowalne. Jednak często pożądane jest zwracanie wierszy w określonym porządku. Wyobraź sobie, że chcesz pobrać wszystkie produkty uporządkowane od najstarszych do najnowszych według daty rozpoczęcia produkcji. W SQL-u umożliwia to klauzula ORDER BY:

```
SELECT
    model
FROM
    products
ORDER BY
    production_start_date;
```

Jeśli kolejność sortowania nie jest bezpośrednio określona, zwracane wiersze będą uporządkowane rosnąco. Oznacza to, że wiersze są porządkowane od najmniejszej do największej wartości z wybranej kolumny (lub ze zbioru kolumn). W przypadku tekstu uwzględniana jest

kolejność alfabetyczna. Aby jawnie zażądać kolejności rosnącej, użyj słowa kluczowego ASC. W poprzedniej kwerendzie można to zrobić tak:

```
SELECT
    model
FROM
    products
ORDER BY
    production_start_date ASC;
```

Jeśli chcesz pobrać dane w porządku malejącym, użyj słowa kluczowego DESC. Jeżeli zamierzasz wczytać modele uporządkowane od najnowszych do najstarszych, zastosuj następujący kod:

```
SELECT
    model
FROM
    products
ORDER BY
    production_start_date DESC;
```

Ponadto zamiast podawać nazwę kolumny, według której chcesz sortować dane, możesz podać numer tej kolumny w naturalnym układzie tabeli. Załóżmy, że chcesz zwrócić wszystkie modele z tabeli products uporządkowane według identyfikatorów produktów. Możesz zapisać kwerendę tak:

```
SELECT
    model
FROM
    products
ORDER BY
    product_id;
```

Ponieważ jednak product_id to pierwsza kolumna w tej tabeli, możesz też zastosować taki zapis:

```
SELECT
    model
FROM
    products
ORDER BY
    1;
```

Możesz także sortować dane według kilku kolumn, podając po klauzuli ORDER BY dodatkowe kolumny rozdzielone przecinkami. Załóżmy, że chcesz uporządkować wszystkie wiersze tabeli najpierw według roku produkcji modelu (od najnowszych do najstarszych), a następnie według sugerowanej ceny detalicznej (od najniższej do najwyższej). Możesz zapisać kod tak:

```
SELECT
    *
FROM
    products
ORDER BY
    year DESC,
    base_msrp ASC;
```

Na rysunku 1.37 widoczne są dane wyjściowe tego kodu.

product_id bigint	model text	year bigint	product_type text	base_msrp numeric	production_start_date timestamp without time zone	production_end_date timestamp without time zone
12	Lemon ...	2019	scooter	349.99	2019-02-04 00:00:00	[null]
11	Model ...	2019	automobile	95000.00	2019-02-04 00:00:00	[null]
8	Bat Limi...	2017	scooter	699.99	2017-02-15 00:00:00	[null]
9	Model E...	2017	automobile	35000.00	2017-02-15 00:00:00	[null]
10	Model ...	2017	automobile	85750.00	2017-02-15 00:00:00	[null]
7	Bat	2016	scooter	599.99	2016-10-10 00:00:00	[null]
6	Model S...	2015	automobile	65500.00	2015-04-15 00:00:00	2018-10-01 00:00:00
5	Blade	2014	scooter	699.99	2014-06-23 00:00:00	2015-01-27 00:00:00
4	Model ...	2014	automobile	115000.00	2014-06-23 00:00:00	2018-12-28 00:00:00
3	Lemon	2013	scooter	499.99	2013-05-01 00:00:00	2018-12-28 00:00:00
2	Lemon ...	2011	scooter	799.99	2011-01-03 00:00:00	2011-03-30 00:00:00

Rysunek 1.37. Porządkowanie według wielu kolumn za pomocą klauzuli ORDER BY

W następnym punkcie poznasz słowo kluczowe LIMIT z SQL-a.

Klauzula LIMIT

Większość tabel w bazach SQL-owych jest dość duża, dlatego nie trzeba zwracać wszystkich wierszy. Czasem potrzebnych jest tylko kilka pierwszych wierszy. W takim scenariuszu przydatne jest słowo kluczowe LIMIT. Wyobraź sobie, że chcesz pobrać tylko pięć pierwszych produktów firmy. Możesz to zrobić za pomocą następującej kwerendy:

```
SELECT
  model
FROM
  products
ORDER BY
  production_start_date
LIMIT
  5;
```

Na rysunku 1.38 pokazane są dane wyjściowe tego kodu.

model text
Lemon
Lemon Limited Edition
Lemon
Blade
Model Chi

Rysunek 1.38. Kwerenda z klauzulą LIMIT

Zgodnie z ogólną regułą zwykle warto używać słowa kluczowego LIMIT w tabelach i kwerendach, z których wcześniej nie korzystałeś.

Klauzule IS NULL i IS NOT NULL

Często w kolumnie brakuje niektórych wartości. Może to wynikać z wielu powodów. Możliwe, że dane nie zostały pobrane lub były niedostępne w momencie ich wprowadzania. Możliwe, że proces ETL nie pobrał danych i nie wczytał ich do kolumny. Możliwe też, że brak wartości reprezentuje określony stan wiersza i stanowi cenną informację. Niezależnie od powodu analityk często chce znaleźć wiersze, w których w określonej kolumnie brakuje wartości. W SQL-u brak wartości nieraz jest reprezentowany za pomocą NULL. Na przykład w tabeli products wartość NULL w kolumnie `production_end_date` oznacza, że produkt wciąż jest wytwarzany. Dlatego jeśli chcesz wyświetlić wszystkie nadal produkowane towary, możesz użyć następującej kwerendy:

```
SELECT
  *
FROM
  products
WHERE
  production_end_date IS NULL;
```

Na rysunku 1.39 pokazane są dane wyjściowe.

product_id bigint	model text	year bigint	product_type text	base_msrp text	production_start_date timestamp without time zone	production_end_date timestamp without time zone
7	Bat	2016	scooter	599.99	2016-10-10 00:00:00	[null]
8	Bat Limi...	2017	scooter	699.99	2017-02-15 00:00:00	[null]
9	Model E...	2017	automobile	35,000.00	2017-02-15 00:00:00	[null]
10	Model ...	2017	automobile	85,750.00	2017-02-15 00:00:00	[null]
11	Model ...	2019	automobile	95,000.00	2019-02-04 00:00:00	[null]
12	Lemon ...	2019	scooter	349.99	2019-02-04 00:00:00	[null]

Rysunek 1.39. Produkty z wartością NULL w kolumnie `production_end_date`

Jeśli interesują Cię tylko produkty, które już nie są wytwarzane, możesz użyć klauzuli `IS NOT NULL`, tak jak w tej kwerendzie:

```
SELECT *
FROM products
WHERE production_end_date IS NOT NULL;
```

Dane wyjściowe są pokazane na rysunku 1.40.

product_id bigint	model text	year bigint	product_type text	base_msrp text	production_start_date timestamp without time zone	production_end_date timestamp without time zone
1	Lemon	2010	scooter	399.99	2010-03-03 00:00:00	2012-06-08 00:00:00
2	Lemon ...	2011	scooter	799.99	2011-01-03 00:00:00	2011-03-30 00:00:00
3	Lemon	2013	scooter	499.99	2013-05-01 00:00:00	2018-12-28 00:00:00
4	Model ...	2014	automobile	115,000.00	2014-06-23 00:00:00	2018-12-28 00:00:00
5	Blade	2014	scooter	699.99	2014-06-23 00:00:00	2015-01-27 00:00:00
6	Model S...	2015	automobile	65,500.00	2015-04-15 00:00:00	2018-10-01 00:00:00

Rysunek 1.40. Produkty z kolumną `production_end_date` o wartości różnej od NULL

W następnym ćwiczeniu zobaczysz, jak wykorzystać nowo poznane słowa kluczowe.

Ćwiczenie 1.06 — kwerenda SELECT z podstawowymi słowami kluczowymi dotycząca tabeli salespeople

W tym ćwiczeniu utworzysz kilka kwerend SELECT, używając podstawowych słów kluczowych. Wyobraź sobie, że po kilku dniach w nowej pracy wreszcie uzyskujesz dostęp do firmowej bazy danych. Dziś Twój szef prosi Cię, abyś pomógł menedżerowi ds. sprzedaży, który nie zna dobrze SQL-a. Menedżer chce uzyskać kilka różnych list z danymi o sprzedawcach. Najpierw przygotuj listę internetowych nazw użytkownika dziesięciu pierwszych kobiet pracujących na stanowisku sprzedawcy; dane uporządkuj od pierwszej do ostatniej zatrudnionej.

We wszystkich kolejnych ćwiczeniach z tej książki używane jest narzędzie pgAdmin 4.

Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Otwórz wybranego klienta SQL-a i nawiąż połączenie z bazą sql da.
2. Zbadaj schemat tabeli salespeople za pomocą listy rozwijanej. Zwróć uwagę na nazwy kolumn z rysunku 1.41.



Rysunek 1.41. Schemat tabeli salespeople

3. Wykonaj poniższą kwerendę, aby uzyskać nazwy użytkownika kobiet na stanowisku sprzedawcy posortowane według wartości hire_date; w klauzuli LIMIT użyj wartości 10:

```
SELECT
    username
FROM
    salespeople
WHERE
    gender= 'Female'
ORDER BY
    hire_date
LIMIT 10;
```


Rysunek 1.42 przedstawia dane wyjściowe.

	username text
1	nlie2l
2	adufaire3r
3	bgrimoldby4q
4	jmedgewick...
5	bhain3y
6	kclyburn54
7	adobbing4g
8	skinner1h
9	alimon7j

Rysunek 1.42. Nazwy użytkownika kobiet pracujących jako sprzedawcy posortowane według daty zatrudnienia

W ten sposób otrzymujesz listę nazw użytkownika kobiet pracujących jako sprzedawca uporządkowanych od najwcześniej do najpóźniej zatrudnionych.

Kod źródłowy do tego punktu znajdziesz na stronie <https://packt.live/2B4qMUK>.

W tym ćwiczeniu użyłeś różnych podstawowych słów kluczowych w kwerendzie SELECT, aby pomóc menedżerowi ds. sprzedaży w pobraniu listy sprzedawców zgodnie z podanymi kryteriami.

Zadanie 1.03 — kwerenda SELECT z podstawowymi słowami kluczowymi dotycząca tabeli customers

Dział marketingu uznał, że chce przeprowadzić serię kampanii marketingowych, aby pomóc w zwiększeniu sprzedaży. W tym celu potrzebuje szczegółowych informacji o wszystkich klientach z Nowego Jorku. Oto kroki potrzebne do wykonania tego zadania:

1. Otwórz wybranego klienta SQL-a i nawiąż połączenie z bazą sql da.
Zbadaj schemat z tabeli customers za pomocą listy rozwijanej schematu.
2. Napisz kwerendę, która zwraca w porządku alfabetycznym e-maile wszystkich klientów firmy ZoomZoom ze stanu Floryda.
3. Napisz kwerendę, która zwraca imię, nazwisko i e-mail klientów firmy ZoomZoom z miasta Nowy Jork w stanie Nowy Jork. Dane mają być uporządkowane alfabetycznie najpierw według nazwiska, a następnie według imienia.

4. Napisz kwerendę, która zwraca dane wszystkich klientów wraz z numerem telefonu. Dane mają być uporządkowane według daty dodania klienta do bazy.

Rozwiązanie tego zadania znajduje się w „Dodatku”.

W tym zadaniu użyłeś różnych podstawowych słów kluczowych w kwerendzie SELECT, aby pomóc menedżerowi z działu marketingu w pobraniu danych potrzebnych do kampanii marketingowej.

Tworzenie tabel

Wiesz już, jak wczytywać dane z tabel. Teraz zobacz, jak tworzyć nowe tabele. Można to robić na dwa sposoby — tworząc puste tabele lub używając kwerend SELECT.

Tworzenie pustych tabel

Aby utworzyć nową pustą tabelę, należy użyć instrukcji CREATE TABLE. Ta instrukcja ma następującą strukturę:

```
CREATE TABLE {nazwa_tabeli} (
  {nazwa_kolumny_1} {typ_danych_1} {ograniczenia_kolumny_1},
  {nazwa_kolumny_2} {typ_danych_2} {ograniczenia_kolumny_2},
  {nazwa_kolumny_3} {typ_danych_3} {ograniczenia_kolumny_3},
  ...
  {nazwa_kolumny_ostatnia} {typ_danych_ostatni} {ograniczenia_kolumny_ostatnie},
);
```

Tu {nazwa_tabeli} to nazwa tabeli, {nazwa_kolumny} to nazwa kolumny, {typ_danych} to typ danych kolumny, a {ograniczenia_kolumny} to jedno lub kilka opcjonalnych słów kluczowych określających specjalne cechy kolumny. Przed omówieniem użycia kwerendy CREATE TABLE warto najpierw omówić ograniczenia kolumn.

Ograniczenia kolumn

Ograniczenia kolumn to słowa kluczowe określające specjalne cechy kolumn. Oto wybrane ważne ograniczenia:

- NOT NULL — gwarantuje, że żadna wartość w danej kolumnie nie będzie równa NULL.
- UNIQUE — gwarantuje, że każdy wiersz w kolumnie ma unikatową wartość (żadna wartość się nie powtarza).
- PRIMARY KEY — jest to specjalne ograniczenie wyznaczające kolumnę klucza głównego. Oznacza, że wartość w danej kolumnie jest unikatowa dla każdego wiersza i pomaga szybciej wyszukiwać wiersze. Tylko jedna kolumna tabeli może być kluczem głównym.

Załóżmy, że chcesz utworzyć tabelę `state_populations` z kolumnami ze skróconą nazwą i liczbą mieszkańców każdego stanu USA. Potrzebna kwerenda wygląda tak:

```
CREATE TABLE state_populations (
    state VARCHAR(2) PRIMARY KEY,
    population NUMERIC
);
```

Ta kwerenda daje następujący wynik:

Query returned successfully in 122 msec.

Czasem po uruchomieniu kwerendy `CREATE TABLE` możesz zobaczyć błąd `relation {nazwa_tabeli} already exists`. Oznacza to, że istnieje już tabela o danej nazwie. Musisz wtedy albo usunąć istniejącą tabelę o tej nazwie, ale zmienić nazwę nowej tabeli.

Teraz poznasz drugi sposób tworzenia tabel — za pomocą kwerend w SQL-u. Najpierw jednak wykonaj ćwiczenie, w którym utworzysz tabelę za pomocą SQL-a.

Ćwiczenie 1.07 — tworzenie tabeli w SQL-u

W tym ćwiczeniu utworzysz tabelę za pomocą instrukcji `CREATE TABLE`. Dział marketingu w firmie ZoomZoom chce utworzyć tabelę `countries`, aby przeanalizować dane dla różnych państw. Ta tabela ma zawierać cztery kolumny: całkowitoliczbową kolumnę z kluczem, kolumnę z unikatową nazwą, kolumnę z rokiem powstania i kolumnę ze stolicą.

Oto kroki niezbędne do wykonania tego zadania:

1. Otwórz wybranego klienta SQL-a i nawiąż połączenie z bazą sql da.
2. Wykonaj poniższą kwerendę, aby usunąć tabelę `countries`, jeśli w bazie istnieje już tabela o tej nazwie:

```
DROP TABLE IF EXISTS countries;
```

3. Uruchom następującą kwerendę, aby utworzyć tabelę `countries`:

```
CREATE TABLE countries (
    key INT PRIMARY KEY,
    name text UNIQUE,
    founding_year INT,
    capital text
);
```

Powinieneś otrzymać pustą tabelę z rysunku 1.43.

Kod źródłowy z tego punktu znajdziesz na stronie <https://packt.live/3cWfoSE>.

	key integer	name text	founding_year integer	capital text

Rysunek 1.43. Pusta tabela countries z nazwami kolumn

W tym ćwiczeniu pokazaliśmy, jak utworzyć tabelę z różnymi ograniczeniami kolumn za pomocą instrukcji `CREATE TABLE`. W następnym punkcie utworzysz tabelę przy użyciu kwerendy `SELECT`.

Tworzenie tabel za pomocą kwerendy `SELECT`

Wiesz już, jak utworzyć tabelę. Załóżmy jednak, że chcesz utworzyć tabelę na podstawie danych z istniejącej tabeli. Można to zrobić za pomocą innej wersji instrukcji `CREATE TABLE`:

```
CREATE TABLE {nazwa_tabeli} AS (
    {kwerenda_select}
);
```

Tu `{kwerenda_select}` to dowolna kwerenda `SELECT`, jaką można uruchomić w bazie. Załóżmy, że chcesz utworzyć tabelę opartą na tabeli `products`, ale zawierającą tylko produkty z 2014 roku. Nazwij tę tabelę `products_2014`. Potrzebna kwerenda wygląda tak:

```
CREATE TABLE products_2014 AS (
    SELECT
        *
    FROM
        products
    WHERE
        year=2014
);
```

W taki sposób można użyć dowolnej kwerendy, a nowa tabela odziedziczy wszystkie wartości z pierwotnej tabeli.

Aktualizowanie tabel

W przyszłości konieczne może być zmodyfikowanie tabeli przez dodanie kolumn, dodanie danych lub aktualizację istniejących wierszy. W tym podrozdziale zobaczysz, jak to zrobić.

Dodawanie i usuwanie kolumn

Aby dodać nowe kolumny do istniejącej tabeli, użyj instrukcji `ADD COLUMN`:

```
ALTER TABLE {nazwa_tabeli}
ADD COLUMN {nazwa_kolumny} {typ_danych};
```

Załóżmy, że chcesz dodać do tabeli `products` nową kolumnę, `weight`, która posłuży do zapisywania wagi produktów w kilogramach. Możesz to zrobić za pomocą następującej kwerendy:

```
ALTER TABLE products
ADD COLUMN weight INT;
```

Ta kwerenda tworzy w tabeli `products` nową kolumnę, `weight`, i przypisuje jej całkowitoliczbowy typ danych, aby można w niej było zapisywać tylko liczby.

Jeśli chcesz usunąć kolumnę z tabeli, możesz użyć instrukcji `DROP COLUMN`:

```
ALTER TABLE {nazwa_tabeli}
DROP COLUMN {nazwa_kolumny};
```

Tu `{nazwa_tabeli}` to nazwa tabeli, którą chcesz zmodyfikować, a `{nazwa_kolumny}` to nazwa usuwanej kolumny.

Jeśli chcesz usunąć wcześniej utworzoną kolumnę `weight`, możesz to zrobić za pomocą następującej kwerendy:

```
ALTER TABLE products
DROP COLUMN weight;
```

Dodawanie nowych danych

W SQL-u nowe dane możesz dodać do tabeli za pomocą kilku metod.

Jedna z nich polega na bezpośrednim wstawieniu wartości do tabeli za pomocą instrukcji `INSERT INTO...VALUES`. Oto struktura tej instrukcji:

```
INSERT INTO {table_name} (
    {kolumna_1}, {kolumna_2}, ... {kolumna_ostatnia}
)
VALUES (
    {wartość_kolumny_1}, {wartość_kolumny_2},
    ... {wartość_kolumny_ostatniej}
);
```

Tu {nazwa_tabeli} to nazwa tabeli, do której chcesz wstawić dane, {kolumna_1}, {kolumna_2}, ..., {kolumna_ostatnia} to lista kolumn, do których mają trafić wartości, a {wartość_kolumny_1}, {wartość_kolumny_2}, ..., {wartość_kolumny_ostatniej} to lista wartości wiersza, jakie kwerenda ma zapisać w tabeli. Jeśli w instrukcji INSERT jakieś istniejące kolumny tabeli zostaną pominięte, domyślnie umieszczone zostaną w nich wartości NULL.

Założmy, że chcesz wstawić do tabeli products nowy skuter. Możesz użyć następującej kwerendy:

```
INSERT INTO products (
    product_id, model, year,
    product_type, base_msrp,
    production_start_date, production_end_date
)
VALUES (
    13, 'Nimbus 5000', 2019,
    'scooter', 500.00,
    '2019-03-03', '2020-03-03'
);
```

Ta kwerenda odpowiednio modyfikuje tabelę products, co widać na rysunku 1.44.

product_id bigint	model text	year bigint	product_type text	base_msrp text	production_start_date timestamp without time zone	production_end_date timestamp without time zone
1	Lemon	2010	scooter	399.99	2010-03-03 00:00:00	2012-06-08 00:00:00
2	Lemon ...	2011	scooter	799.99	2011-01-03 00:00:00	2011-03-30 00:00:00
3	Lemon	2013	scooter	499.99	2013-05-01 00:00:00	2018-12-28 00:00:00
4	Model ...	2014	automobile	115,000.00	2014-06-23 00:00:00	2018-12-28 00:00:00
5	Blade	2014	scooter	699.99	2014-06-23 00:00:00	2015-01-27 00:00:00
6	Model S...	2015	automobile	65,500.00	2015-04-15 00:00:00	2018-10-01 00:00:00
7	Bat	2016	scooter	599.99	2016-10-10 00:00:00	[null]
8	Bat Limi...	2017	scooter	699.99	2017-02-15 00:00:00	[null]
9	Model E...	2017	automobile	35,000.00	2017-02-15 00:00:00	[null]
10	Model ...	2017	automobile	85,750.00	2017-02-15 00:00:00	[null]
11	Model ...	2019	automobile	95,000.00	2019-02-04 00:00:00	[null]
12	Lemon ...	2019	scooter	349.99	2019-02-04 00:00:00	[null]
13	Nimbus...	2019	scooter	500.00	2019-03-03 00:00:00	2020-03-03 00:00:00

Rysunek 1.44. Tabela products po udanym wstawieniu kwerendy INSERT

Innym sposobem na wstawianie danych do tabeli jest użycie instrukcji INSERT razem z kwerendą SELECT. Oto składnia tego rozwiązania:

```
INSERT INTO {nazwa_tabeli} ({kolumna_1}, {kolumna_2}, ... {kolumna_ostatnia})
{kwerenda_select};
```

Tu {nazwa_tabeli} to nazwa tabeli, w której chcesz wstawić dane, {kolumna_1}, {kolumna_2}, ..., {kolumna_ostatnia} to lista kolumn, w których chcesz umieścić wartości, a {kwerenda_select} to kwerenda o strukturze zgodnej ze strukturą wartości wstawianych do tabeli.

Przyjrzyj się opisanej wcześniej tabeli `products_2014`. Załóżmy, że zamiast tworzyć ją za pomocą kwerendy `SELECT` przygotowałeś pustą tabelę o tej samej strukturze co tabela `products`. Jeśli chcesz wstawić te same dane co wcześniej, możesz użyć następującej kwerendy:

```
INSERT INTO products_2014(
    product_id, model, year,
    product_type, base_msrp,
    production_start_date, production_end_date
)
SELECT
    *
FROM
    products
WHERE
    year=2014;
```

Ta kwerenda daje efekt widoczny na rysunku 1.45.

product_id bigint	model text	year bigint	product_type text	base_msrp text	production_start_date timestamp without time zone	production_end_date timestamp without time zone
4	Model ...	2014	automobile	115,000.00	2014-06-23 00:00:00	2018-12-28 00:00:00
5	Blade	2014	scooter	699.99	2014-06-23 00:00:00	2015-01-27 00:00:00

Rysunek 1.45. Tabela `products_2014` po udanym wykonaniu kwerendy `INSERT INTO`

Teraz dowiesz się, jak zaktualizować zawartość wiersza.

Aktualizowanie istniejących wierszy

Czasem trzeba zaktualizować wartości danych już znajdujących się w tabeli. Możesz użyć do tego instrukcji `UPDATE`:

```
UPDATE {nazwa_tabeli}
SET {kolumna_1} = {wartość_kolumny_1},
    {kolumna_2} = {wartość_kolumny_2},
    ...
    {kolumna_ostatnia} = {{wartość_kolumny_ostatniej}}
WHERE
    {warunek};
```

Tu `{nazwa_tabeli}` to nazwa tabeli z modyfikowanymi danymi, `{kolumna_1}`, `{kolumna_2}`, ..., `{kolumna_ostatnia}` to lista kolumn, których wartości chcesz zmodyfikować, a `{wartość_kolumny_1}`, `{wartość_kolumny_2}`, ..., `{wartość_kolumny_ostatniej}` to lista nowych wartości, które chcesz wstawić w tych kolumnach. Z kolei `{warunek}` to instrukcja warunkowa, taka, jakie stosuje się w kwerendach `SQL`.

Założmy, że przez resztę roku firma zamierza sprzedawać wszystkie skutery z roczników starszych niż 2018 po 299,99 dolara. Możesz zmodyfikować odpowiednie dane w tabeli `products`, używając następującej kwerendy:

```
UPDATE
    products
SET
```

```

base_msrp = 299.99
WHERE
  product_type = 'scooter'
  AND year<2018;

```

Ta kwerenda zwraca dane wyjściowe widoczne na rysunku 1.46.

product_id bigint	model text	year bigint	product_type text	base_msrp text	production_start_date timestamp without time zone	production_end_date timestamp without time zone
1	Lemon	2010	scooter	299.99	2010-03-03 00:00:00	2012-06-08 00:00:00
2	Lemon ...	2011	scooter	299.99	2011-01-03 00:00:00	2011-03-30 00:00:00
3	Lemon	2013	scooter	299.99	2013-05-01 00:00:00	2018-12-28 00:00:00
4	Model ...	2014	automobile	115,000.00	2014-06-23 00:00:00	2018-12-28 00:00:00
5	Blade	2014	scooter	299.99	2014-06-23 00:00:00	2015-01-27 00:00:00
6	Model S...	2015	automobile	65,500.00	2015-04-15 00:00:00	2018-10-01 00:00:00
7	Bat	2016	scooter	299.99	2016-10-10 00:00:00	[null]
8	Bat Limi...	2017	scooter	299.99	2017-02-15 00:00:00	[null]
9	Model E...	2017	automobile	35,000.00	2017-02-15 00:00:00	[null]
10	Model ...	2017	automobile	85,750.00	2017-02-15 00:00:00	[null]
11	Model ...	2019	automobile	95,000.00	2019-02-04 00:00:00	[null]
12	Lemon ...	2019	scooter	349.99	2019-02-04 00:00:00	[null]
13	Nimbus...	2019	scooter	500.00	2019-03-03 00:00:00	2020-03-03 00:00:00

Rysunek 1.46. Udana aktualizacja tabeli products

W następnym ćwiczeniu przyjrzyj się dokładnie działaniu instrukcji UPDATE w bazie SQL-owej.

Ćwiczenie 1.08 — aktualizowanie tabeli w celu podniesienia ceny pojazdu

W tym ćwiczeniu za pomocą instrukcji UPDATE zaktualizujesz dane w tabeli. Z powodu wyższych kosztów rzadkich metali potrzebnych do produkcji samochodów elektrycznych cena nowego modelu Chi z 2019 roku musi zostać podniesiona o 10% względem aktualnej ceny 95 000 dolarów. Zaktualizuj tabelę products, aby podwyższyć cenę produktu.

Oto kroki niezbędne do wykonania tego zadania:

1. Otwórz wybrany klienta SQL-a, aby nawiązać połączenie z bazą sql da.
2. Uruchom poniższą kwerendę, aby zaktualizować cenę samochodu Model Chi w tabeli products:

```

UPDATE
  products
SET
  base_msrp = base_msrp*1.10
WHERE
  model='Model Chi '
  AND year=2019;

```


3. Teraz napisz kwerendę SELECT, aby sprawdzić, czy cena samochodu Model Chi z 2019 roku została zaktualizowana:

```
SELECT
  *
FROM
  products
WHERE
  model='Model Chi '
  AND year=2019;
```

Na rysunku 1.47 pokazane są dane wyjściowe tego kodu.

product_id	model	year	product_type	base_msrp	production_start_date	production_end_date
11	Model Chi	2019	automobile	104500	2019-02-04 00:00:00	NULL

Rysunek 1.47. Zaktualizowana cena pojazdu Model Chi z 2019 roku.

W tych danych wyjściowych widać, że cena pojazdu Model Chi wynosi obecnie 104 500, choć wcześniej była równa 95 000.

Kod źródłowy z tego punktu jest dostępny na stronie <https://packt.live/2XRJV17>.

W tym ćwiczeniu pokazaliśmy, jak zaktualizować tabelę za pomocą instrukcji UPDATE. Teraz dowiesz się, jak usuwać tabele i dane.

Usuwanie danych i tabel

Często okazuje się, że dane w tabeli są nieprawidłowe, dlatego nie mogą być już używane. Wtedy trzeba usunąć dane z tabeli.

Usuwanie wartości z wiersza

Niekiedy trzeba usunąć wartość z wiersza. Najłatwiej użyć do tego opisanej już instrukcji UPDATE i przypisać do kolumny wartość NULL:

```
UPDATE {nazwa_tabeli}
SET {kolumna_1} = NULL,
    {kolumna_2} = NULL,
    ...
    {kolumna_ostatnia} = NULL
WHERE {warunek};
```

W tej instrukcji {nazwa_tabeli} to nazwa tabeli, w której dane są modyfikowane, {kolumna_1}, {kolumna_2}, ..., {kolumna_ostatnia} to lista kolumn, z których chcesz usunąć wartości, a {warunek} to instrukcja warunkowa, taka, jakie stosuje się w kwerendach SQL.

Załóżmy, że w pliku zapisany jest błędny adres e-mail klienta o identyfikatorze równym 3. Aby to zmienić, możesz użyć następującej kwerendy:

```
UPDATE
  customers
SET
  email = NULL
WHERE
  customer_id=3;
```

W następnym punkcie nauczysz się usuwać wiersze z tabeli.

Usuwanie wierszy z tabeli

Aby usunąć wiersz z tabeli, możesz użyć instrukcji DELETE. Wygląda ona tak:

```
DELETE FROM {nazwa_tabeli}
WHERE {warunek};
```

Załóżmy, że musisz usunąć szczegółowe informacje o kliencie z adresem e-mail bjordan2@geocities.com. Aby to zrobić, możesz się posłużyć następującą kwerendą:

```
DELETE FROM
  customers
WHERE
  email='bjordan2@geocities.com';
```

Jeśli chcesz usunąć wszystkie dane z tabeli customers bez kasowania jej samej, możesz zastosować taką kwerendę:

```
DELETE FROM customers;
```

Inny sposób na usunięcie w kwerendzie wszystkich danych bez kasowania tabeli polega na użyciu słowa kluczowego TRUNCATE:

```
TRUNCATE TABLE customers;
```

Usuwanie tabel

Aby usunąć wszystkie dane z tabeli i samą tabelę, możesz wywołać instrukcję DROP TABLE. Oto jej składnia:

```
DROP TABLE {nazwa_tabeli};
```

Tu {nazwa_tabeli} to nazwa tabeli, którą chcesz usunąć. Jeśli zamierzasz skasować wszystkie dane z tabeli customers wraz z samą tabelą, użyj takiej kwerendy:

```
DROP TABLE customers;
```

Teraz wykonaj ćwiczenie, aby usunąć tabelę za pomocą instrukcji DROP TABLE.

Ćwiczenie 1.09 — usuwanie niepotrzebnej tabeli

W tym ćwiczeniu nauczysz się usuwać tabelę za pomocą SQL-a. Dział marketingu zakończył analizowanie potencjalnej liczby klientów w każdym stanie i nie potrzebuje już tabeli `state_populations`. Aby zmniejszyć ilość miejsca zajmowanego przez bazę danych, usuń tę tabelę.

Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Otwórz wybranego klienta SQL-a i nawiąż połączenie z bazą `sql` da.
2. Uruchom poniższą kwerendę, aby usunąć tabelę `state_populations`:

```
DROP TABLE state_populations;
```

Tabela `state_populations` powinna zostać usunięta z bazy danych.

3. Ponieważ tabela została skasowana, dotycząca jej kwerenda `SELECT` zwróci — zgodnie z oczekiwaniami — błąd:

```
SELECT
  *
FROM
  state_populations;
```

Ten błąd jest pokazany na rysunku 1.48.

```
ERROR: BŁĄD: relacja "state_populations" nie istnieje
LINE 4:  state_populations;
        ^
```

Rysunek 1.48. Błąd po usunięciu tabeli `state_populations`

Kod źródłowy z tego punktu jest dostępny pod adresem <https://packt.live/2XWLVZA>.

W tym ćwiczeniu pokazaliśmy, jak usunąć tabelę za pomocą instrukcji `DROP TABLE`. W następnym zadaniu utworzysz i zmodyfikujesz tabele za pomocą SQL-a.

Zadanie 1.04 — tworzenie i modyfikowanie tabel na potrzeby działań marketingowych

W tym zadaniu przetestujesz swoje umiejętności z zakresie tworzenia i modyfikowania tabel za pomocą SQL-a. Wykonałeś dobrą robotę w pobieraniu danych dla działu marketingu. Jednak menedżer ds. marketingu, któremu pomogłeś, zdał sobie sprawę, że o czymś zapomniał. Okazuje się, że oprócz pobrania danych chce też utworzyć nową tabelę w analitycznej bazie danych firmy. Ponadto musi wprowadzić zmiany w danych z tabeli `customers`. Twoje zadanie polega na tym, by pomóc menedżerowi w tworzeniu tabeli:

1. Utwórz nową tabelę `customers_nyc`, która pobiera z tabeli `customers` wszystkie wiersze dotyczące klientów mieszkających w mieście Nowy Jork w stanie o tej samej nazwie.
2. Usuń z nowej tabeli wszystkich klientów, którzy mają adres z kodem pocztowym 10014. Z powodu lokalnych przepisów nie można kierować do nich akcji marketingowej.
3. Dodaj nową kolumnę tekstową `event`.
4. Ustaw wartość w kolumnie `event` na `Impreza w ramach podziękowań`.
Na rysunku 1.49 pokazane są oczekiwane dane wyjściowe.

	customer_id	title	first_name	last_name	suffix	email	gender	ip_address	phone	street_address	city	state	postal_code	latitude	longitude	data_added	event
	bigint	text	text	text	text	text	text	text	text	text	text	text	text	double precision	double precision	timestamp	text
1	52	[null]	Giusto	Backe	[null]	gback...	M	26.56.68.189	212-9...	6 Onsgard Terrace	New York City	NY	10131	40.7808	-73.9772	2010-07-06	Impreza w ramach podziękowań
2	162	[null]	Artair	Betchley	[null]	abec...	M	108.147.128.250	[null]	7 Boyd Road	New York City	NY	10090	40.7808	-73.9772	2014-06-25	Impreza w ramach podziękowań
3	406	[null]	Rozina	Jeal	[null]	rjealb...	F	50.235.32.29	917-6...	64653 Homewood T...	New York City	NY	10105	40.7628	-73.9785	2010-09-15	Impreza w ramach podziękowań
4	456	Rev	Cybil	Noke	[null]	cnoke...	F	5.31.139.106	212-3...	88 Sycamore Parkw...	New York City	NY	10260	40.7808	-73.9772	2017-01-21	Impreza w ramach podziękowań
5	472	[null]	Rawley	Yegorov	[null]	ryego...	M	183.199.243.74	212-5...	872 Old Shore Park...	New York City	NY	10034	40.8662	-73.9221	2014-11-24	Impreza w ramach podziękowań
6	496	[null]	Layton	Spolton	[null]	lspolt...	M	108.112.8.165	646-9...	7 Old Gate Drive	New York City	NY	10024	40.7864	-73.9764	2010-12-20	Impreza w ramach podziękowań
7	1028	[null]	Issy	Andrieux	[null]	landri...	F	199.50.5.37	212-2...	33337 Dahle Way	New York City	NY	10115	40.8111	-73.9642	2017-11-27	Impreza w ramach podziękowań
8	1037	[null]	Magdalene	Veryard	[null]	mvery...	F	93.201.129.213	[null]	41028 Katie Junction	New York City	NY	10039	40.8265	-73.9383	2014-03-04	Impreza w ramach podziękowań
9	1063	[null]	Juliet	Beedles	[null]	jbeadl...	F	47.96.88.226	212-4...	34984 Goodland Poi...	New York City	NY	10120	40.7506	-73.9894	2014-08-17	Impreza w ramach podziękowań
10	1211	[null]	Gwyneth	McCobb	[null]	gmcc...	F	38.182.151.212	[null]	4 Jana Park	New York City	NY	10160	40.7808	-73.9772	2014-01-08	Impreza w ramach podziękowań
11	1262	[null]	Conrado	Escoffier	[null]	cesco...	M	23.120.12.44	646-5...	2 Atwood Court	New York City	NY	10060	40.7808	-73.9772	2015-02-17	Impreza w ramach podziękowań
12	1333	[null]	Franny	Tipping	[null]	ftippi...	F	125.115.139.118	212-5...	93444 Drewry Trail	New York City	NY	10292	40.7808	-73.9772	2014-09-17	Impreza w ramach podziękowań
13	1403	[null]	Terza	Dericut	[null]	tderic...	F	243.18.169.116	646-7...	3037 Linden Center	New York City	NY	10060	40.7808	-73.9772	2015-07-10	Impreza w ramach podziękowań
14	1587	[null]	Gregoire	Marciek	[null]	gmar...	M	75.31.7.169	[null]	9 Lunder Place	New York City	NY	10039	40.8265	-73.9383	2011-01-14	Impreza w ramach podziękowań
15	1675	[null]	Ram	Acheson	[null]	rach...	M	34.129.78.67	718-4...	5 Rutledge Point	New York City	NY	10019	40.7651	-73.9858	2016-06-15	Impreza w ramach podziękowań
16	1860	[null]	Fernanda	Haney	[null]	fhane...	F	148.198.91.104	[null]	17 Hanover Point	New York City	NY	10150	40.7808	-73.9772	2012-03-27	Impreza w ramach podziękowań

Rysunek 1.49. Tabela `customers_nyc` z kolumną `event` o wartości `Impreza w ramach podziękowań`

5. Informujesz menedżera, że wykonałeś opisane kroki. Menedżer powiadamia o tym zespół ds. działań marketingowych, który wykorzystuje dane do przeprowadzenia kampanii. Menedżer dziękuje Ci, po czym prosi, abyś usunął tabelę `customers_nyc`.

Rozwiązanie tego zadania znajdziesz w „Dodatku”.

W tym zadaniu wykorzystałeś różne operacje CRUD, aby zmodyfikować tabelę zgodnie z prośbami menedżera ds. marketingu. Po tym wstępie pora wrócić do powiązań SQL-a z analizą danych.

SQL i analityka

Możliwe, że w tym rozdziale dostrzegłeś związki między tabelami z SQL-a a zbiorami danych. Powinno być już zrozumiałe, że tabele z SQL-a należy traktować jak zbiory danych, wiersze to odpowiednik jednostek obserwacji, a kolumny przechowują wartości cech. Jeśli spojrzysz na tabele z SQL-a w ten sposób, zobaczysz, że SQL to naturalne narzędzie do przechowywania zbiorów danych w komputerze.

Jednak SQL nie tylko zapewnia wygodny sposób przechowywania zbiorów danych. Nowe implementacje SQL-a udostępniają też narzędzia do przetwarzania i analizowania danych z użyciem różnych funkcji. Za pomocą SQL-a możesz oczyszczać dane, przetwarzać je na bardziej przydatne formaty, a także analizować z wykorzystaniem statystyk, by wykrywać ciekawe wzorce. Reszta książki pozwoli Ci zrozumieć, jak korzystać z SQL-a do wykonywania takich operacji w produktywny i wydajny sposób.

Podsumowanie

Analiza danych to rozbudowana dziedzina pozwalająca zrozumieć świat. Ostatecznym celem analiz jest przekształcanie danych w informacje i wiedzę. Aby osiągnąć te cele, można posłużyć się statystyką (przede wszystkim statystyką opisową i testami istotności statystycznej), by lepiej zrozumieć dane.

Gałąź statystyki opisowej zwana analizą jednoczynnikową pomaga zrozumieć jedną zmienną z danych. Analiza jednoczynnikowa pozwala znajdować wartości odstające, badać rozkład danych za pomocą rozkładu częstości występowania i kwantyli, sprawdzać tendencję centralną na podstawie obliczeń średniej, mediany i wartości modalnej, a także analizować dyspersję danych z użyciem rozstępu, odchylenia standardowego i rozstępu ćwiartkowego.

Do zrozumienia relacji między danymi można też wykorzystać analizę dwuczynnikową. Za pomocą wykresów punktowych można ocenić trendy, zmiany w trendach, zjawiska cykliczne i anomalie dotyczące dwóch zmiennych. Można także za pomocą współczynnika korelacji Pearsona zmierzyć siłę trendu liniowego dla dwóch zmiennych. Współczynnik ten należy jednak dokładnie sprawdzić pod kątem wartości odstających i liczby punktów danych użytych w obliczeniach. Ponadto wysoka korelacja między dwiema zmiennymi nie oznacza, że jedna zmienna przyczynowo wpływa na drugą.

Także testy istotności statystycznej mogą zapewniać ważne informacje na temat danych. Te testy pozwalają ocenić, jak prawdopodobne jest przypadkowe uzyskanie określonych wyników. Pomagają też zrozumieć, czy zmiany i różnice między grupami są istotne statystycznie.

Analizę danych można wzbogacić dzięki możliwościom, jakie dają relacyjne bazy danych. Bazy relacyjne to dojrzała i wszechobecna technologia przechowywania i pobierania danych. Takie bazy przechowują dane w formie relacji (nazywanych tabelami) i zapewniają doskonałe połączenie szybkości działania, wydajności i łatwości użytkowania. SQL to język używany do dostępu do relacyjnych baz danych. Jest językiem deklaratywnym, dlatego umożliwia użytkownikom skupienie się na tym, co chcą uzyskać, a nie na procesie tworzenia wyniku. SQL udostępnia wiele typów danych, w tym typy liczbowe i tekstowe, a nawet struktury danych.

W procesie pobierania danych SQL umożliwia użytkownikowi wskazanie pól, z których dane należy wczytać, a także sposobu filtrowania danych. Dane można też sortować. Ponadto SQL umożliwia pobranie dowolnie dużej lub małej ilości danych. Również operacje tworzenia, wczytywania, aktualizowania i usuwania danych są dość proste i można je precyzyjnie wykonywać.

Po objaśnieniu podstaw analizy danych i SQL-a w następnym rozdziale przejdziemy do omówienia tego, jak używać SQL-a do wykonywania pierwszego kroku w analizie danych — oczyszczania i przekształcania danych.

Przygotowywanie danych za pomocą SQL-a

W tym rozdziale nauczysz się oczyszczać i przygotowywać dane za pomocą SQL-a na potrzeby analiz. Najpierw zobaczysz, jak na podstawie kilku tabel i kwerend tworzyć zbiory danych z zastosowaniem złączeń, sum, podkwerend i funkcji, aby przekształcić dane. Później przejdziesz do bardziej zaawansowanych zagadnień. Dzięki lekturze tego rozdziału nauczysz się przekształcać i oczyszczać dane za pomocą funkcji SQL-a oraz usuwać powtarzające się dane z użyciem instrukcji `DISTINCT` i `DISTINCT ON`.

Wprowadzenie

W poprzednim rozdziale omówiliśmy podstawy analizy danych i SQL-a. Używaliśmy też operacji CRUD (ang. *create, read, update, delete*, czyli tworzenie, wczytywanie, aktualizowanie i usuwanie) do tabel. Te techniki są podstawą wszystkich zadań wykonywanych w trakcie analiz. Jednym z takich zadań jest tworzenie oczyszczonych zbiorów danych.

Według magazynu „Forbes” profesjonalni analitycy poświęcają prawie 80% czasu na przygotowanie danych wykorzystywanych do analiz. Budowanie modeli z użyciem nieoczyszczonych danych szkodzi analizom i prowadzi do wniosków o niskiej przydatności. SQL może pomóc w żmudnym, ale ważnym zadaniu przygotowywania danych, ponieważ zapewnia wydajne sposoby tworzenia oczyszczonych zbiorów.

Zacniemy od omówienia łączenia danych za pomocą klauzul `JOIN` i `UNION`. Dalej zobaczysz, jak używać różnych funkcji (takich jak `CASE WHEN`, `COALESCE`, `NULLIF`, `LEAST` i `GREATEST`) do oczyszczania danych. Następnie wyjaśniamy, jak przekształcać dane i usuwać powtórzenia z wyników kwerend za pomocą polecenia `DISTINCT`.

Łączenie danych

Wcześniej wyjaśniliśmy, jak wykonywać operacje na jednej tabeli. Co jednak zrobić, jeśli potrzebne są dane z dwóch lub więcej tabel? W tym podrozdziale połączysz dane z kilku tabel, używając złączeń i sum.

Łączenie tabel za pomocą słowa kluczowego JOIN

W poprzednim rozdziale omówiliśmy jak pobierać dane z tabeli. Jednak potrzebne dane zwykle znajdują się w kilku tabelach. Na szczęście SQL udostępnia metody łączenia powiązanych tabel za pomocą słowa kluczowego JOIN.

Aby zrozumieć to zagadnienie, najpierw przyjrzyj się dwóm tabelom z bazy danych — dealerships i salespeople (rysunki 2.1 i 2.2).

Column	Type
dealership_id	bigint
street_address	text
city	text
state	text
postal_code	text
latitude	double precision
longitude	double precision
date_opened	timestamp without time zone
date_closed	timestamp without time zone

Rysunek 2.1. Struktura tabeli dealerships

Column	Type
salesperson_id	bigint
dealership_id	bigint
title	text
first_name	text
last_name	text
suffix	text
username	text
gender	text
hire_date	timestamp without time zone
termination_date	timestamp without time zone

Rysunek 2.2. Struktura tabeli salespeople

W tabeli salespeople znajduje się kolumna dealership_id, która bezpośrednio wskazuje wartości z kolumny dealership_id tabeli dealerships. Gdy tabela A zawiera kolumnę wskazującą wartości klucza głównego z tabeli B, taka kolumna jest kluczem obcym tabeli A. Tu kolumna dealership_id z tabeli salespeople jest kluczem obcym tej tabeli wskazującym wartości z tabeli dealerships.

Klucze obce można też dodawać do tabeli jako ograniczenia kolumny, aby zwiększyć integralność danych poprzez zagwarantowanie, że klucz obcy nigdy nie będzie zawierał wartości nieobecnej we wskazywanej tabeli. Ta cecha danych to **integralność referencyjna**. Dodanie ograniczenia klucza obcego pomaga też zwiększyć wydajność niektórych baz. W większości analitycznych baz danych klucze obce nie są używane, a ich omawianie wykracza poza zakres tej książki. Więcej o ograniczeniach klucza obcego dowiesz się z dokumentacji systemu PostgreSQL na stronie <https://www.postgresql.org/docs/9.4/tutorialfk.html>.

Ponieważ dwie wspomniane tabele są powiązane, możesz wykonywać na nich interesujące analizy. Na przykład może ciekawić Cię, które osoby pracują w salonie w Kalifornii. Jednym ze sposobów na uzyskanie tych informacji jest pobranie najpierw salonów zlokalizowanych w Kalifornii. Możesz to zrobić za pomocą następującej kwerendy:

```
SELECT
*
FROM
dealerships
WHERE
state='CA';
```

Ta kwerenda daje wyniki pokazane na rysunku 2.3.

	dealership_id bigint	street_address text	city text	state text	postal_code text	latitude double precision	longitude double precision	date_opened timestamp without time zone	date_closed timestamp without time zone
1	2	808 South Hobart...	Los ...	CA	90005	34.057754	-118.305423	2014-06-01 00:00:00	[null]
2	5	2210 Bunker Hill ...	San ...	CA	94402	37.524487	-122.343609	2014-06-01 00:00:00	[null]

Rysunek 2.3. Salony z Kalifornii

Teraz, gdy już wiesz, że jedyne dwa salony z Kalifornii mają identyfikatory 2 i 5, możesz pobrać dane z tabeli salespeople w następujący sposób:

```
SELECT
*
FROM
salespeople
WHERE
dealership_id in (2, 5)
ORDER BY
1;
```

Rysunek 2.4 przedstawia dane wyjściowe tego kodu.

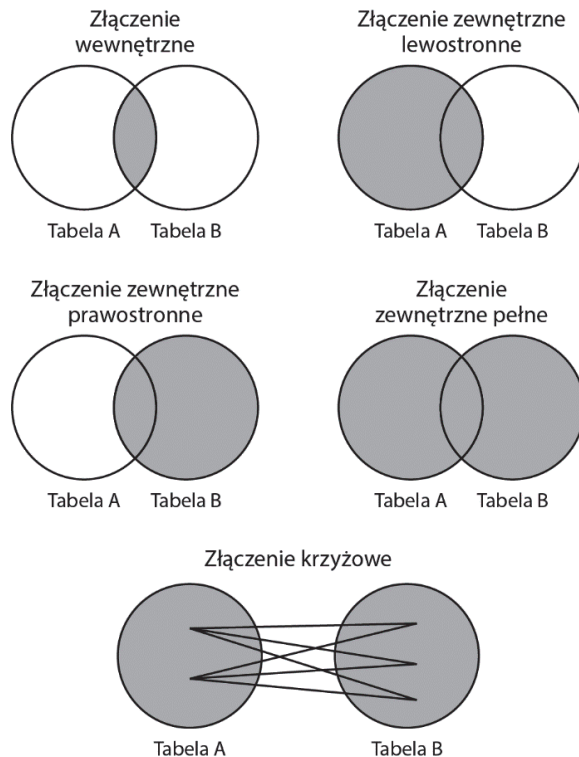
salesperson_id bigint	dealership_id bigint	title text	first_name text	last_name text	suffix text	username text	gender text	hire_date timestamp without time zone	termination_date timestamp without time zone
23	2	[null]	Beauregard	Peschke	[null]	bpeschkem	Male	2018-09-12 00:00:00	[null]
51	5	[null]	Lanette	Gerriessen	[null]	lgerriessen1e	Female	2018-06-24 00:00:00	[null]
57	5	[null]	Spense	Pithcock	[null]	spithcock1k	Male	2017-12-15 00:00:00	[null]
61	5	[null]	Ludvig	Baynam	[null]	lbaynam1o	Male	2016-08-25 00:00:00	[null]
62	2	[null]	Carroll	Pudan	[null]	cpudan1p	Female	2016-05-17 00:00:00	[null]
63	2	[null]	Adrianne	Otham	[null]	aotham1q	Female	2014-12-20 00:00:00	[null]
71	2	[null]	Georgianna	Bastian	[null]	gbastian1y	Female	2018-12-23 00:00:00	[null]
75	2	[null]	Saundra	Shoebottom	[null]	sshoebotto...	Female	2018-03-18 00:00:00	[null]
108	2	[null]	Hale	Brigshaw	[null]	hbrigshaw2z	Male	2015-07-30 00:00:00	[null]

Rysunek 2.4. Sprzedawcy z Kalifornii

Choć ta technika daje oczekiwane wyniki, wykonywanie dwóch kwerend w celu ich otrzymania jest żmudne. Łatwiej byłoby w jakiś sposób dodać informacje z tabeli `dealerships` do tabeli `salespeople` i przefiltrować wyniki, aby uzyskać dane o sprzedawcach z Kalifornii. SQL udostępnia potrzebne do tego narzędzie — klauzulę `JOIN`. Umożliwia ona użytkownikowi złączanie tabel na podstawie określonych warunków.

Rodzaje złączeń

W tym rozdziale omawiamy trzy podstawowe rodzaje złączeń przedstawione na rysunku 2.5. Są to złączenia wewnętrzne, złączenia zewnętrzne i złączenia krzyżowe.



Rysunek 2.5. Główne typy złączeń

Złączenia wewnętrzne

Złączenie wewnętrzne łączy ze sobą różne tabele na podstawie warunku nazywanego **predykatem złączenia**. Takim predykatem często jest warunek logiczny w postaci równości. Każdy wiersz pierwszej tabeli jest porównywany z każdym wierszem drugiej tabeli. Jeśli dana kombinacja wierszy spełnia predykat złączenia, odpowiedni wiersz jest dodawany do wyników kwerendy. W przeciwnym razie dana kombinacja jest pomijana w wynikach.

Złączenia wewnętrzne są zwykle zapisywane w następujący sposób:

```
SELECT {kolumny}
FROM {tabela1}
INNER JOIN {tabela2} ON {tabela1}.{wspólny_klucz_1}={tabela2}.{wspólny_klucz_2};
```

Tu {kolumny} to kolumny, jakie mają się znajdować w złączonej tabeli, {tabela1} to pierwsza tabela, {tabela2} to druga tabela, {wspólny_klucz_1} to kolumna tabeli {tabela1} sprawdzana w warunku, a {wspólny_klucz_2} to kolumna tabeli {tabela2} sprawdzana w warunku.

Wróć teraz do dwóch opisanych wcześniej tabel — dealerships i salespeople. Wcześniej wspomnieliśmy, że byłoby dobrze, gdyby udało się dodać informacje z tabeli dealerships do tabeli salespeople, aby umożliwić ustalenie, w którym stanie pracuje każdy sprzedawca. Na razie przyjmij, że w wierszach z danymi wszystkich sprzedawców znajduje się poprawna wartość dealership_id.

Nie masz jeszcze umiejętności potrzebnych do sprawdzenia, czy wszystkie identyfikatory salonów w tabeli salespeople są poprawne, dlatego trzeba to założyć. Jednak w praktyce ważne jest, aby samodzielnie sprawdzać takie informacje. Istnieją bardzo nieliczne zbiory danych i systemy gwarantujące, że dane są oczyszczone.

Możesz złączyć dwie tabele za pomocą warunku równości w predykcji złączenia:

```
SELECT
*
FROM
salespeople
INNER JOIN
dealerships
ON salespeople.dealership_id = dealerships.dealership_id
ORDER BY
1;
```

Ta kwerenda daje wyniki widoczne na rysunku 2.6.

salesperson_id bigint	dealership_id bigint	title text	first_name text	last_name text	suffix text	username text	gender text	hire_date timestamp without time zone	termination_date timestamp without time zone	dealership_id bigint	street_address text	city text	state text	postal_code text
1	17	[null]	Electra	Elleyne	[null]	eelleyne0	Female	2017-05-31 00:00:00	[null]		17 2120 Walnut Street	Phila...	PA	19092
2	6	[null]	Montague	Alcoran	[null]	malcoran1	Male	2018-12-31 00:00:00	[null]		6 7315 California A.	Seatt...	WA	98136
3	17	[null]	Ethyl	Sloss	IV	esloss2	Female	2016-08-10 00:00:00	[null]		17 2120 Walnut Street	Phila...	PA	19092
4	10	[null]	Nester	Dugood	[null]	ndugood3	Male	2017-06-03 00:00:00	[null]		10 7425 Wilson Aven.	Chic...	IL	60706
5	17	[null]	Comall	Swanger	[null]	cswanger4	Male	2018-05-17 00:00:00	[null]		17 2120 Walnut Street	Phila...	PA	19092
6	8	[null]	Ellary	Hend	[null]	ehend5	Male	2016-05-07 00:00:00	[null]		8 5938 Comfort Ro...	Portl...	OR	97218
7	1	[null]	Granville	Fidel	[null]	gfidel6	Male	2017-06-17 00:00:00	[null]		1 52 Hillside Terrace	Millb...	NJ	07039
8	18	[null]	Lanie	Tisun	[null]	ltisun7	Male	2017-12-12 00:00:00	[null]		18 1447 Hardesty Av...	Kans...	MO	64195
9	14	[null]	Lamar	Treleven	[null]	ltreleven8	Male	2018-05-08 00:00:00	[null]		14 800 North Mays S...	Roun...	TX	78664

Rysunek 2.6. Tabela salespeople złączona z tabelą dealerships

W danych wyjściowych z rysunku 2.6 widać, że otrzymana tabela to wynik złączenia tabel salespeople i dealerships. Zauważ, że pierwsza wynikowa tabela kwerendy, salespeople, to lewa strona wyników, a tabela dealerships to strona prawa. Będzie to istotne do zrozumienia następnego punktu, poświęconego złączeniom zewnętrznym.

Tu pole `dealership_id` z tabeli `salespeople` musi pasować do pola `dealership_id` z tabeli `dealerships`. To pokazuje, jak można spełnić predykat złączenia. Za pomocą kwerendy z klauzulą `join` utworzyliśmy nowy „superzbiór danych”, który składa się z wierszy z dwóch tabel łączonych, gdy wartości z kolumn `dealership_id` są takie same.

Teraz można tworzyć kwerendy dotyczące „superzbioru danych” w taki sam sposób jak kwerendy dotyczące jednej dużej tabeli, używając klauzul i słów kluczowych z rozdziału 1., „Wprowadzenie do SQL-a dla analityków”. Wróć do problemu kilku kwerend potrzebnych do ustalenia, którzy sprzedawcy pracują w Kalifornii. Teraz zadanie można wykonać za pomocą jednej prostej kwerendy:

```
SELECT
    *
FROM
    salespeople
INNER JOIN
    dealerships
    ON salespeople.dealership_id = dealerships.dealership_id
WHERE
    dealerships.state = 'CA'
ORDER BY
    1;
```

Dane wyjściowe są pokazane na rysunku 2.7.

	salesperson_id bigint	dealership_id bigint	title text	first_name text	last_name text	suffix text	username text	gender text	hire_date timestamp without time zone	termination_date timestamp without time zone	dealership_id bigint	street_address text	city text
1		23	2 [null]	Beauregard	Peschke	[null]	bpeschkem	Male	2018-09-12 00:00:00	[null]		2 808 South Hobart...	Los ...
2		51	5 [null]	Lanette	Gerriessen	[null]	lgerriessen1e	Female	2018-06-24 00:00:00	[null]		5 2210 Bunker Hill ...	San ...
3		57	5 [null]	Spense	Pithcock	[null]	spithcock1k	Male	2017-12-15 00:00:00	[null]		5 2210 Bunker Hill ...	San ...
4		61	5 [null]	Ludvig	Baynam	[null]	lbyaynam1o	Male	2016-08-25 00:00:00	[null]		5 2210 Bunker Hill ...	San ...
5		62	2 [null]	Carroll	Pudan	[null]	cpudan1p	Female	2016-05-17 00:00:00	[null]		2 808 South Hobart...	Los ...
6		63	2 [null]	Adrianne	Otham	[null]	aotham1q	Female	2014-12-20 00:00:00	[null]		2 808 South Hobart...	Los ...
7		71	2 [null]	Georgianna	Bastian	[null]	gbastian1y	Female	2018-12-23 00:00:00	[null]		2 808 South Hobart...	Los ...
8		75	2 [null]	Saundra	Shoebottom	[null]	sshoebotto...	Female	2018-03-18 00:00:00	[null]		2 808 South Hobart...	Los ...
9		108	2 [null]	Hale	Brigshaw	[null]	hbrigshaw2z	Male	2015-07-30 00:00:00	[null]		2 808 South Hobart...	Los ...

Rysunek 2.7. Sprzedawcy z Kalifornii pobrani za pomocą jednej kwerendy

Zauważ, że dane wyjściowe z rysunków 2.2 i 2.5 są prawie identyczne. Różnica polega na tym, że w tabeli z rysunku 2.5 dołączone są także dane z tabeli `dealerships`. Jeśli chcesz pobrać z nich tylko dane z tabeli `salespeople`, możesz wskazać kolumny z tej tabeli, używając gwiazdki:

```
SELECT
    salespeople.*
FROM
    salespeople
INNER JOIN
    dealerships
    ON dealerships.dealership_id = salespeople.dealership_id
WHERE
    dealerships.state = 'CA'
ORDER BY
    1;
```

Istnieje jeszcze jeden skrót pomocny przy pisaniu instrukcji z kilkoma klauzulami join. Otóż możesz utworzyć alias nazwy tabeli, aby nie musieć za każdym razem wpisywać jej całej. Wystarczy podać alias po pierwszym wystąpieniu nazwy tabeli w klauzuli join, aby zaoszczędzić sobie sporo pisania. Jeśli w omawianej kwerendzie chcesz zastosować alias s dla tabeli salespeople i d dla tabeli dealerships, możesz użyć następującej instrukcji:

```
SELECT
  s.*
FROM
  salespeople s
  INNER JOIN
    dealerships d
      ON d.dealership_id = s.dealership_id
WHERE
  d.state = 'CA'
ORDER BY
  1;
```

Możesz też dodać słowo kluczowe AS między nazwą tabeli a aliasem, aby operacja tworzenia aliasu była bardziej widoczna:

```
SELECT
  s.*
FROM
  salespeople AS s
  INNER JOIN
    dealerships AS d
      ON d.dealership_id = s.dealership_id
WHERE
  d.state = 'CA'
ORDER BY
  1;
```

Po zapoznaniu się z podstawami złączeń wewnętrznych pora przejść do złączeń zewnętrznych.

Złączenia zewnętrzne

Napisaaliśmy już, że złączenia wewnętrzne zwracają wiersze z obu tabel, jeśli predykat złączenia jest spełniony dla danej kombinacji wierszy. W przeciwnym razie nie są zwracane wiersze z żadnej tabeli. Jednak czasem trzeba zwrócić wszystkie wiersze z jednej z tabel niezależnie od tego, czy spełniony jest predykat złączenia. W takim scenariuszu, jeśli predykat nie jest spełniony, wiersz z drugiej tabeli zostanie zastąpiony wartością NULL. Złączenia tego rodzaju, w których po operacji join zwracane są wszystkie wiersze z przynajmniej jednej tabeli, to **złączenia zewnętrzne**.

Można je podzielić na trzy kategorie: lewostronne, prawostronne i pełne.

W **złączeniu zewnętrznym lewostronnym** zwracane są wszystkie wiersze tabeli podanej po lewej stronie (pierwszej w klauzuli join). Jeśli instrukcja nie znajdzie pasującego wiersza z drugiej tabeli, zwraca wiersz NULL. Złączenia zewnętrzne lewostronne są tworzone za pomocą

słów kluczowych LEFT OUTER JOIN, po których należy podać predykat złączenia. Można też użyć skróconego zapisu LEFT JOIN.

Aby zrozumieć działanie złączeń zewnętrznych lewostronnych, przyjrzyj się dwóm tabelom: customers i emails. Na razie przyjmij, że nie każdy klient otrzymał e-mail, a celem jest przesłanie e-maili do wszystkich klientów, którzy ich jeszcze nie dostali. Można użyć do tego złączenia zewnętrznego lewostronnego z tabelą customers po lewej stronie. Aby ułatwić sobie zarządzanie danymi wyjściowymi, ogranicz je do pierwszego 1000 wierszy. Oto potrzebny fragment kodu:

```
SELECT
    *
FROM
    customers c
LEFT OUTER JOIN
    emails e ON e.customer_id=c.customer_id
ORDER BY
    c.customer_id
LIMIT
    1000;
```

Rysunek 2.8 przedstawia dane wyjściowe.

customer_id bigint	title text	first_name text	last_name text	suffix text	email text	gender text	ip_address text	phone text	street_address text	city text	state text	postal_code text	latitude double precision	longitude double precision	date_added timestamp without time zone	email_id bigint
1	[null]	Arlena	Riveles	[null]	arivele..	F	98.36.172.246	[null]	[null]	[null]	[null]	[null]	[null]	[null]	2017-04-23 00:00:00	282584
1	[null]	Arlena	Riveles	[null]	arivele..	F	98.36.172.246	[null]	[null]	[null]	[null]	[null]	[null]	[null]	2017-04-23 00:00:00	370722
1	[null]	Arlena	Riveles	[null]	arivele..	F	98.36.172.246	[null]	[null]	[null]	[null]	[null]	[null]	[null]	2017-04-23 00:00:00	323983
2	Dr	Ode	Stovin	[null]	ostovin..	M	16.97.59.186	314-534..	2573 Fordem Par...	Saint..	MO	63116	38.5814	-90.2625	2014-10-02 00:00:00	323984
2	Dr	Ode	Stovin	[null]	ostovin..	M	16.97.59.186	314-534..	2573 Fordem Par...	Saint..	MO	63116	38.5814	-90.2625	2014-10-02 00:00:00	245816
2	Dr	Ode	Stovin	[null]	ostovin..	M	16.97.59.186	314-534..	2573 Fordem Par...	Saint..	MO	63116	38.5814	-90.2625	2014-10-02 00:00:00	144183
2	Dr	Ode	Stovin	[null]	ostovin..	M	16.97.59.186	314-534..	2573 Fordem Par...	Saint..	MO	63116	38.5814	-90.2625	2014-10-02 00:00:00	370723
2	Dr	Ode	Stovin	[null]	ostovin..	M	16.97.59.186	314-534..	2573 Fordem Par...	Saint..	MO	63116	38.5814	-90.2625	2014-10-02 00:00:00	282585
2	Dr	Ode	Stovin	[null]	ostovin..	M	16.97.59.186	314-534..	2573 Fordem Par...	Saint..	MO	63116	38.5814	-90.2625	2014-10-02 00:00:00	117146
2	Dr	Ode	Stovin	[null]	ostovin..	M	16.97.59.186	314-534..	2573 Fordem Par...	Saint..	MO	63116	38.5814	-90.2625	2014-10-02 00:00:00	209804
2	Dr	Ode	Stovin	[null]	ostovin..	M	16.97.59.186	314-534..	2573 Fordem Par...	Saint..	MO	63116	38.5814	-90.2625	2014-10-02 00:00:00	174737
2	Dr	Ode	Stovin	[null]	ostovin..	M	16.97.59.186	314-534..	2573 Fordem Par...	Saint..	MO	63116	38.5814	-90.2625	2014-10-02 00:00:00	91913
3	[null]	Braden	Jordan	[null]	bjorda..	M	192.86.248.59	[null]	5651 Kennedy Park	Pens..	FL	32590	30.6143	-87.2758	2018-10-27 00:00:00	323985

Rysunek 2.8. Złączenie lewostronne tabel customers i emails

Gdy przyjrzyj się danym wyjściowym kwerendy, zobaczysz, że znajdują się w nich dane z tabeli customers. Jednak w niektórych wierszach, na przykład dotyczących klienta o identyfikatorze 27 z rysunku 2.9, kolumny należące do tabeli emails zawierają same wartości NULL. To pokazuje, dlaczego złączenie zewnętrzne różni się od wewnętrznego. Gdybyś zastosował złączenie wewnętrzne, kolumna customer_id nie byłaby pusta.

Mimo to kwerenda jest przydatna, ponieważ pozwala znaleźć osoby, które nie otrzymały e-maila. Klienci, do których nie wysłano wiadomości, mają wartość NULL w kolumnie customer_id tabeli emails. Można więc znaleźć wszystkie takie osoby, sprawdzając wartość kolumny customer_id w tabeli emails:

```
SELECT
    c.customer_id,
    c.title,
    c.first_name,
    c.last_name,
```

```

        c.suffix,
        c.email,
        c.gender,
        c.ip_address,
        c.phone,
        c.street_address,
        c.city,
        c.state,
        c.postal_code,
        c.latitude,
        c.longitude,
        c.date_added,
        e.email_id,
        e.email_subject,
        e.opened,
        e.clicked,
        e.bounced,
        e.sent_date,
        e.opened_date,
        e.clicked_date
FROM
    customers c
LEFT OUTER JOIN
    emails e ON c.customer_id = e.customer_id
WHERE
    e.customer_id IS NULL
ORDER BY
    c.customer_id
LIMIT
    1000;

```

Dane wyjściowe tej kwerendy są pokazane na rysunku 2.9.

customer_id bigint	title text	first_name text	last_name text	suffix text	email text	gender text	ip_address text	phone text	street_address text	city text	state text	postal_code text	latitude double prec	longitude double prec	date_added timestamp without time zone	email_id bigint	customer_id bigint	email text
27	[null]	Anson	Fellbrand	[null]	afellbr...	M	64.80.85.50	203-107...	65 Shelley Road	New ...	CT	06505	41.3057	-72.7799	2019-04-07 00:00:00	[null]	[null]	[null]
32	[null]	Hammet	Purselowe	[null]	hpurse...	M	225.215.209...	239-462...	5 Johnson Way	Napli...	FL	34102	26.134	-81.7953	2019-03-07 00:00:00	[null]	[null]	[null]
70	[null]	Caty	Woolveridge	[null]	cwoolv...	F	104.21.118.34	757-238...	[null]	[null]	[null]	[null]	[null]	[null]	2019-04-09 00:00:00	[null]	[null]	[null]
77	[null]	Donal	Lathey	[null]	dlathey...	M	5.51.114.103	804-575...	48889 Laurel Pass	Charl...	WV	25326	38.2968	-81.5547	2019-05-25 00:00:00	[null]	[null]	[null]
112	[null]	Harcourt	Cripps	[null]	hcripp...	M	219.20.188.2...	951-922...	9 Hoard Place	San ...	CA	92410	34.1069	-117.2975	2019-02-21 00:00:00	[null]	[null]	[null]
113	[null]	Gifty	Bernington	Jr	gbenni...	M	181.117.182...	202-767...	7861 Michigan Po...	Was...	DC	20231	38.8933	-77.0146	2019-02-13 00:00:00	[null]	[null]	[null]
125	[null]	Bernard	Jirka	[null]	bjirka...	M	124.68.237.8	[null]	112 Lunder Hill	Pitts...	PA	15215	40.5048	-79.9138	2019-03-17 00:00:00	[null]	[null]	[null]
192	[null]	Selina	Hearl	[null]	shearl...	F	174.136.106...	585-208...	842 Moulton Court	Roch...	NY	14646	43.286	-77.6843	2019-04-09 00:00:00	[null]	[null]	[null]
199	[null]	Mercy	Martschik	[null]	mmart...	F	63.73.23.98	352-750...	66667 Stone Corn...	Broo...	FL	34605	28.5059	-82.4226	2019-04-01 00:00:00	[null]	[null]	[null]
212	[null]	Norma	Goldis	[null]	ngoldi...	F	90.182.242.61	215-737...	4865 Sauthoff Cir...	Phila...	PA	19125	39.9788	-75.1262	2019-03-26 00:00:00	[null]	[null]	[null]

Rysunek 2.9. Klienci, do których nie wysłano e-maili

Widać tu, że w kolumnie `customer_id` tabeli `emails` wszystkie wiersze mają wartość `NULL`, co oznacza, że te osoby nie otrzymały wiadomości. Można więc pobrać dane z tego złączenia, aby uzyskać wszystkich klientów, którzy nie dostali e-maila.

Złączenie zewnętrzne prawostronne jest bardzo podobne do lewostronnego, ale obejmuje wszystkie wiersze z „prawej” tabeli (podanej jako druga). Wiersze z „lewiej” tabeli, dla których warunek złączenia nie jest spełniony, zawierają wartości `NULL`. Aby się o tym przekonać, „odwróć” ostatnią kwerendę i zastosuj złączenie prawostronne tabeli `emails` z tabelą `customers`:

```

SELECT c.customer_id,
       c.title,
       c.first_name,
       c.last_name,
       c.suffix,
       c.email,
       c.gender,
       c.ip_address,
       c.phone,
       c.street_address,
       c.city,
       c.state,
       c.postal_code,
       c.latitude,
       c.longitude,
       c.date_added,
       e.email_id,
       e.email_subject,
       e.opened,
       e.clicked,
       e.bounced,
       e.sent_date,
       e.opened_date,
       e.clicked_date
FROM
  emails e
RIGHT OUTER JOIN
  customers c ON e.customer_id=c.customer_id
ORDER BY
  c.customer_id
LIMIT
  1000;

```

Gdy uruchomisz tę kwerendę, otrzymasz wyniki pokazane na rysunku 2.10.

email_id bigint	customer_id bigint	email_subject text	opened text	clicked text	bounced text	sent_date timestamp without time zone	opened_date timestamp without time zone	clicked_date timestamp without time zone	customer_id bigint	title text	first_name text	last_name text	suffix text
282584	1	Black Friday, Gre...	t	f	f	2017-11-24 15:00:00	2017-11-26 01:12:32	[null]	1	[null]	Arlena	Riveles	[null]
370722	1	A New Year, And ...	f	f	f	2019-01-07 15:00:00	[null]	[null]	1	[null]	Arlena	Riveles	[null]
323983	1	Save the Planet ...	f	f	f	2018-11-23 15:00:00	[null]	[null]	1	[null]	Arlena	Riveles	[null]
323984	2	Save the Planet ...	f	f	f	2018-11-23 15:00:00	[null]	[null]	2	Dr	Ode	Stovin	[null]
245816	2	We Really Outrid...	t	f	f	2017-01-15 15:00:00	2017-01-16 09:22:16	[null]	2	Dr	Ode	Stovin	[null]
144183	2	Tis' the Season f...	f	f	f	2015-11-26 15:00:00	[null]	[null]	2	Dr	Ode	Stovin	[null]
370723	2	A New Year, And ...	f	f	f	2019-01-07 15:00:00	[null]	[null]	2	Dr	Ode	Stovin	[null]
282585	2	Black Friday, Gre...	f	f	f	2017-11-24 15:00:00	[null]	[null]	2	Dr	Ode	Stovin	[null]
117146	2	An Electric Car f...	f	f	f	2015-04-01 15:00:00	[null]	[null]	2	Dr	Ode	Stovin	[null]
209804	2	25% off all EVs. I...	f	f	f	2016-11-25 15:00:00	[null]	[null]	2	Dr	Ode	Stovin	[null]
174737	2	Like a Bat out of ...	f	f	f	2016-09-21 15:00:00	[null]	[null]	2	Dr	Ode	Stovin	[null]
91913	2	Zoom Zoom Bla...	f	f	f	2014-11-28 15:00:00	[null]	[null]	2	Dr	Ode	Stovin	[null]
323985	3	Save the Planet ...	f	f	f	2018-11-23 15:00:00	[null]	[null]	3	[null]	Braden	Jordan	[null]
370724	3	A New Year, And ...	f	f	f	2019-01-07 15:00:00	[null]	[null]	3	[null]	Braden	Jordan	[null]
323986	4	Save the Planet ...	f	f	f	2018-11-23 15:00:00	[null]	[null]	4	[null]	Jessika	Nussen	[null]
282586	4	Black Friday, Gre...	f	f	f	2017-11-24 15:00:00	[null]	[null]	4	[null]	Jessika	Nussen	[null]
370725	4	A New Year, And ...	f	f	f	2019-01-07 15:00:00	[null]	[null]	4	[null]	Jessika	Nussen	[null]
323987	5	Save the Planet ...	t	f	f	2018-11-23 15:00:00	2018-11-25 04:31:57	[null]	5	[null]	Lonnie	Rembaud	[null]
174738	5	Like a Bat out of ...	t	f	f	2016-09-21 15:00:00	2016-09-22 10:12:21	[null]	5	[null]	Lonnie	Rembaud	[null]

Rysunek 2.10. Złączenie prawostronne tabeli emails z tabelą customers

Zauważ, że dane wyjściowe są podobne jak na rysunku 2.8, jednak dane z tabeli `emails` znajdują się po lewej stronie, a dane z tabeli `customers` po prawej. Także teraz dla klienta z polem `customer_id` równym 27 kolumny z tabeli `emails` mają wartości równe `NULL`. To pokazuje symetrię między złączeniami prawo- i lewostronnymi.

Istnieje też **pełne złączenie zewnętrzne**. Zwraca ono wszystkie wiersze z tabel lewej i prawej niezależnie od tego, czy predykat złączenia jest spełniony. W przypadku wierszy, dla których predykat jest spełniony, wiersze z obu tabel są łączone w grupę. Wiersze, dla których predykat nie jest spełniony, są uzupełniane wartością `NULL`. Pełne złączenie zewnętrzne jest wykonywane za pomocą klauzuli `FULL OUTER JOIN`, po której należy podać predykat złączenia. Oto składnia:

```
SELECT
    *
FROM
    emails e
FULL OUTER JOIN
    customers c
ON e.customer_id=c.customer_id;
```

Rysunek 2.11 przedstawia dane wyjściowe tej kwerendy.

email_id bigint	customer_id bigint	email_subject text	opened text	clicked text	bounced text	sent_date timestamp without time zone	opened_date timestamp without time zone	clicked_date timestamp without time zone	customer_id bigint
1	18	Introducing A Li...	f	f	f	2011-01-03 15:00:00	[null]	[null]	18
2	30	Introducing A Li...	f	f	f	2011-01-03 15:00:00	[null]	[null]	30
4	52	Introducing A Li...	f	f	f	2011-01-03 15:00:00	[null]	[null]	52
9	103	Introducing A Li...	f	f	f	2011-01-03 15:00:00	[null]	[null]	103
14	137	Introducing A Li...	f	f	f	2011-01-03 15:00:00	[null]	[null]	137
20	215	Introducing A Li...	f	f	f	2011-01-03 15:00:00	[null]	[null]	215
25	311	Introducing A Li...	f	f	f	2011-01-03 15:00:00	[null]	[null]	311
26	315	Introducing A Li...	f	f	f	2011-01-03 15:00:00	[null]	[null]	315
27	338	Introducing A Li...	t	f	f	2011-01-03 15:00:00	2011-01-04 21:59:51	[null]	338
32	380	Introducing A Li...	f	f	f	2011-01-03 15:00:00	[null]	[null]	380
37	422	Introducing A Li...	f	f	f	2011-01-03 15:00:00	[null]	[null]	422
49	596	Introducing A Li...	t	f	f	2011-01-03 15:00:00	2011-01-04 23:29:26	[null]	596
56	673	Introducing A Li...	t	t	f	2011-01-03 15:00:00	2011-01-04 12:03:22	2011-01-04 12:06:54	673

Rysunek 2.11. Pełne złączenie zewnętrzne tabeli `emails` z tabelą `customers`

W tym fragmencie pokazaliśmy, jak stosować trzy różne rodzaje złączeń zewnętrznych. W następnym punkcie użyjesz złączenia krzyżowego.

Złączenia krzyżowe

Ostatnim rodzajem złączeń omawianym w tej książce są **złączenia krzyżowe**. Nazywa się je także iloczynem kartezjańskim. Zwracają one wszystkie możliwe kombinacje wierszy z tabel „lewej” i „prawej”. Przeprowadzić takie złączenie można za pomocą klauzuli `CROSS JOIN`, po której należy podać nazwę drugiej tabeli. Przyjrzyj się przykładowi bazującemu na tabeli `products`.

Żałujemy, że chcesz poznać wszystkie kombinacje dwóch produktów, jakie możesz utworzyć na podstawie danego zbioru (na przykład produktów z tabeli `products`). Celem jest opracowanie dwumiesięcznej kampanii marketingowej z gratisami. Aby uzyskać taką listę kombinacji, możesz użyć złączenia krzyżowego w następującej kwerendzie:


```

SELECT
    p1.product_id, p1.model,
    p2.product_id, p2.model
FROM
    products p1
CROSS JOIN
    products p2;

```

Rysunek 2.12 przedstawia dane wyjściowe tej kwerendy.

product_id bigint	model text	product_id bigint	model text
1	Lemon	1	Lemon
1	Lemon	2	Lemon Li...
1	Lemon	3	Lemon
1	Lemon	4	Model Chi
1	Lemon	5	Blade
1	Lemon	6	Model Sig...
1	Lemon	7	Bat
1	Lemon	8	Bat Limite...
1	Lemon	9	Model Ep...
1	Lemon	10	Model Ga...
1	Lemon	11	Model Chi
1	Lemon	12	Lemon Ze...
2	Lemon ...	1	Lemon

Rysunek 2.12. Złączenie krzyżowe tabeli products z nią samą

Zauważ, że w tym scenariuszu instrukcja zwróciła wszystkie możliwe kombinacje dwóch wartości z tej samej tabeli. Wyniki kwerendy obejmują 144 pozycje (12 produktów na pierwszej pozycji razy 12 produktów na drugiej pozycji). Predykat złączenia nie jest tu potrzebny. Złączenie krzyżowe można potraktować jak złączenie zewnętrzne bez warunków.

W praktyce złączenia krzyżowe nie są stosowane, a jeśli nie zachowasz ostrożności, mogą się okazać bardzo niebezpieczne. Złączenie krzyżowe dwóch dużych tabel może doprowadzić do powstania setek miliardów wierszy, co może zablokować bazę danych i doprowadzić do jej awarii.

Aby dowiedzieć się więcej o złączeniach, zapoznaj się z dokumentacją systemu PostgreSQL na stronie <https://www.postgresql.org/docs/9.1/queries-table-expressions.html>.

Omówiliśmy już podstawy stosowania złączeń do łączenia tabel. Teraz pora przejść do technik złączania wyników kwerend w celu uzyskania zbioru danych.

Ćwiczenie 2.01

— używanie złączeń do analizy sprzedaży w salonach

W tym ćwiczeniu użyjesz złączeń do połączenia powiązanych ze sobą tabel. Dyrektor ds. sprzedaży w Twojej firmie chce otrzymać listę wszystkich klientów, którzy kupili samochód. Trzeba utworzyć kwerendę, która zwraca identyfikator, imię, nazwisko i numer telefonu wszystkich klientów, którzy zakupili taki pojazd.

We wszystkich ćwiczeniach z tej książki używane jest narzędzie **pgAdmin4**. Wszystkie pliki z kodem ćwiczeń i zadań z tego rozdziału są dostępne w serwisie GitHub na stronie <https://packt.live/3hf91Ch>.

Aby rozwiązać problem, wykonaj następujące kroki:

1. Otwórz wybranego klienta SQL-a i nawiąż połączenie z bazą sql da.
2. Użyj złączenia wewnętrznego, aby połączyć tabele sales i customers. Kwerenda ma zwracać następujące dane: identyfikator, nazwisko, imię i numer telefonu klientów.

```
SELECT
    c.customer_id, c.first_name,
    c.last_name, c.phone
FROM
    sales s
INNER JOIN
    customers c ON c.customer_id=s.customer_id
INNER JOIN
    products p ON p.product_id=s.product_id
WHERE
    p.product_type='automobile'
    AND c.phone IS NOT NULL;
```

Powinieneś otrzymać dane wyjściowe podobne do tych z rysunku 2.13.

customer_id bigint	first_name text	last_name text	phone text
35824	Wyatan	Dickie	405-786...
13206	Stace	Tuison	810-769...
2958	Kirstyn	Draysay	208-534...
32636	Kile	Fishlee	937-207...
26730	Raina	Titterell	304-871...
23832	Harrietta	Leverette	803-298...
35844	Maura	Clyne	904-169...
43229	Field	Lopes	757-409...
6038	Carey	Swadling	727-426...

Rysunek 2.13. Klienci, którzy kupili samochód

Widać, że wykonanie kwerendy umożliwiło złączenie danych z tabel sales i customers oraz uzyskanie listy klientów, którzy zakupili samochód.

Kod źródłowy z tego fragmentu znajdziesz na stronie <https://packt.live/2XTzNbr>.

W tym ćwiczeniu za pomocą złączeń udało Ci się połączyć powiązane dane w łatwy i wydajny sposób.

Podkwerendy

Do tej pory pobieraliśmy dane z tabel. Jednak może zauważyłeś już, że kwerendy SELECT zwracają tabele jako dane wyjściowe. Dlatego możesz się zastanawiać, czy istnieje sposób na wykorzystanie tabel zwróconych przez kwerendy SELECT zamiast wskazywania istniejących tabel z bazy danych. Tak, jest to możliwe. Wystarczy umieścić kwerendę w nawiasie i przypisać do niej alias. Jeśli na przykład chcesz znaleźć wszystkich sprzedawców pracujących w Kalifornii i uzyskać takie same wyniki jak na rysunku 2.5, możesz zapisać kwerendę w następujący sposób:

```
SELECT
  *
FROM
  salespeople
INNER JOIN (
  SELECT
    *
  FROM
    dealerships
  WHERE
    dealerships.state = 'CA'
) d
ON d.dealership_id = salespeople.dealership_id
ORDER BY
  1;
```

Tu zamiast łączyć dwie tabele i za pomocą filtrowania pobierać wiersze, w których kod stanu to 'CA', najpierw znajdowane są w tabeli dealerships salony z kodem stanu równym 'CA', po czym stosowane jest złączenie wewnętrzne wierszy z tej kwerendy z tabelą salespeople.

Jeśli kwerenda zwraca tylko jedną kolumnę, można użyć podkwerendy ze słowem kluczowym IN w klauzuli WHERE. Dlatego istnieje też inny sposób na pobranie informacji z tabeli salespeople na podstawie identyfikatora salonów ze stanu Kalifornia:

```
SELECT
  *
FROM
  salespeople
WHERE dealership_id IN (
  SELECT dealership_id FROM dealerships
  WHERE dealerships.state = 'CA'
)
ORDER BY
  1;
```

Wszystkie te przykłady pokazują, że łatwo można zapisać tę samą kwerendę, używając różnych technik. W następnym punkcie omawiamy sumy.

Sumy

Do tej pory opisywaliśmy złączanie danych poziomo. Oznacza to, że złączenia powodują dodawanie nowych kolumn w poziomie. Jednak przydatne może być też połączenie wyników kilku kwerend w pionie, aby zachowana została ta sama liczba kolumn, ale z większą liczbą wierszy. Przykład pomoże to zilustrować.

Załóżmy, że chcesz wyświetlić adresy salonów (z tabeli `dealerships`) i klientów (z tabeli `customers`) za pomocą map Google. W tym celu potrzebne są adresy z tabel `dealerships` i `customers`. Możesz utworzyć kwerendę zwracającą adresy wszystkich klientów:

```
SELECT
    street_address, city, state, postal_code
FROM
    customers
WHERE
    street_address IS NOT NULL;
```

Możesz też pobrać adresy salonów, używając następującej kwerendy:

```
SELECT
    street_address, city, state, postal_code
FROM
    dealerships
WHERE
    street_address IS NOT NULL;
```

Byłoby jednak wygodnie, gdyby można było połączyć wyniki obu tych kwerend w jedną listę w jednej kwerendzie. W takich sytuacjach przydatne jest słowo kluczowe `UNION`. Używając dwóch poprzednich kwerend, możesz utworzyć następującą kwerendę:

```
(
SELECT
    street_address, city, state, postal_code
FROM
    customers
WHERE
    street_address IS NOT NULL
)
UNION
(
SELECT
    street_address, city, state, postal_code
FROM
    dealerships
WHERE
    street_address IS NOT NULL
)
ORDER BY
    1;
```

Uzyskasz dane wyjściowe widoczne na rysunku 2.14.

street_address text	city text	state text	postal_code text
00003 Continenta...	Suff...	VA	23436
00003 Sullivan Ro...	Des ...	IA	50981
00006 Birchwood ...	Lake...	FL	33805
00006 Roth Plaza	Fort ...	AR	72916
00006 Vidon Place	Dallas	TX	75358
00027 Judy Place	Hou...	TX	77293
00031 Redwing D...	Minn...	MN	55446
0003 Novick Trail	Mont...	VT	05609
0004 Northport Al...	Boise	ID	83705
0004 Superior Alley	New ...	NJ	08922
0005 Eagle Crest ...	Ralei...	NC	27626

Rysunek 2.14. Suma adresów

Ze stosowaniem klauzuli UNION związane są pewne zastrzeżenia. Po pierwsze, klauzula UNION wymaga, aby w podkwerendach kolumny miały te same nazwy i typy danych. W przeciwnym razie kwerenda nie zostanie wykonana. Po drugie, klauzula UNION może nie zwrócić wszystkich wierszy z wyników podkwerend. Domyślnie wszystkie powtarzające się wiersze są usuwane z danych wyjściowych. Jeśli chcesz zachować duplikaty, należy zastosować słowo kluczowe UNION ALL. W następnym ćwiczeniu utworzysz sumę.

Ćwiczenie 2.02 — generowanie listy gości na przyjęcie dla klientów VIP za pomocą klauzuli UNION

W tym ćwiczeniu połączysz wyniki dwóch kwerend za pomocą klauzuli UNION. W ramach zaznajamiania odbiorców z nowym pojazdem Model Chi dział marketingu chce urządzić przyjęcie dla wybranych najbogatszych klientów firmy ZoomZoom z Los Angeles w stanie Kalifornia. Aby ułatwić sobie organizację imprezy, dział marketingu prosi Cię o przygotowanie listy gości — klientów firmy ZoomZoom mieszkających w Los Angeles i sprzedawców pracujących w salonie w tym mieście. Na liście należy umieścić imiona i nazwiska oraz informację o tym, czy dana osoba jest klientem, czy pracownikiem.

Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Otwórz wybranego klienta SQL-a i nawiąż połączenie z bazą sql da.
2. Napisz kwerendę, która wygeneruje listę klientów i pracowników firmy ZoomZoom z Los Angeles. Na liście gości należy umieścić imiona i nazwiska oraz informację o tym, czy dana osoba jest klientem, czy pracownikiem:

```

(
SELECT
    first_name, last_name, 'Klient' as guest_type
FROM
    customers
WHERE
    city='Los Angeles'
    AND state='CA'
)
UNION
(
SELECT
    first_name, last_name,
    'Pracownik' as guest_type
FROM
    salespeople s
INNER JOIN
    dealerships d ON d.dealership_id=s.dealership_id
WHERE
    d.city='Los Angeles'
    AND d.state='CA'
)

```

Powinieneś otrzymać dane wyjściowe widoczne na rysunku 2.15.

first_name 	last_name 	guest_type 
Euell	MacWhirter	Klient
Cordy	MacGinney	Klient
Kele	Liepins	Klient
Erma	Tante	Klient
Cheston	Iddon	Klient
Lee	Snalum	Klient
Sly	Thombleson	Klient
Bea	Wrate	Klient
Adelle	Brooksbie	Klient
Tori	Broschek	Klient
Lindy	Weller	Klient
Rock	Shakspeare	Pracownik
Ivor	Charleston	Klient
Court	Britt	Klient
Dick	Steward	Klient
Kristel	Cash	Klient
Abelard	Clayborn	Klient
Ashley	Hartly	Klient
Melisa	Ingleton	Klient

Rysunek 2.15. Lista gości (klientów i pracowników) z Los Angeles w stanie Kalifornia

Możesz wyświetlić listę klientów i pracowników z Los Angeles, uruchamiając pokazaną kwerendę UNION.

Kod źródłowy z tego fragmentu jest dostępny na stronie <https://packt.live/3ffQq79>.

W tym ćwiczeniu użyliśmy słowa kluczowego UNION do łatwego połączenia wierszy z różnych kwerend. W następnym punkcie poznasz wyrażenia WITH.

Wyrażenia WITH

Wyrażenia WITH (ang. *common table expressions*) w pewnym sensie są innym rodzajem podkwerend. Takie wyrażenia pozwalają tworzyć tabele tymczasowe z użyciem klauzuli WITH. Aby lepiej zrozumieć tę klauzulę, przyjrzyj się poniższej kwerendzie, użytej wcześniej do znalezienia sprzedawców z Kalifornii:

```
SELECT
  *
FROM
  salespeople
INNER JOIN (
  SELECT
    *
  FROM
    dealerships
  WHERE
    dealerships.state = 'CA'
) d
ON d.dealership_id = salespeople.dealership_id
ORDER BY
  1;
```

Tę kwerendę za pomocą klauzuli WITH można zapisać tak:

```
WITH d as (
  SELECT
    *
  FROM
    dealerships
  WHERE
    dealerships.state = 'CA'
)
SELECT
  *
FROM
  salespeople
INNER JOIN
  d ON d.dealership_id = salespeople.dealership_id
ORDER BY
  1;
```

Jedną z zalet wyrażeń **WITH** jest rekurencyjność. W **rekurencyjnych wyrażeniach WITH** można używać ich samych. Z tego powodu stosuje się je do rozwiązywania problemów, z którymi trudno jest poradzić sobie za pomocą kwerend. Jednak omawianie rekurencyjnych wyrażeń **WITH** wykracza poza zakres tej książki.

Po zapoznaniu się z kilkoma technikami złączania danych w bazie pora przejść do przekształcania danych wyjściowych.

Przekształcanie danych

Surowe dane wyjściowe kwerend często nie mają pożądanej postaci. Możliwe, że zechcesz usunąć niektóre wartości, zastąpić je lub odwzorować na inne. Do wykonywania tych zadań SQL udostępnia rozmaite instrukcje i funkcje. Funkcje to słowa kluczowe, które przyjmują dane wejściowe (na przykład kolumny lub wartości skalarne) i przekształcają je w dane wyjściowe określonego rodzaju. W następnych punktach omawiamy kilka bardzo przydatnych funkcji do oczyszczania danych.

Funkcja CASE WHEN

Funkcja **CASE WHEN** umożliwia odwzorowanie w kwerendzie określonych wartości z kolumny na inne wartości. Ogólny format instrukcji **CASE WHEN** wygląda tak:

```
CASE WHEN warunek1 THEN wartość1
      WHEN warunek2 THEN wartość2
      ...
      WHEN warunekX THEN wartośćX
      ELSE wartość_dla_else END;
```

Tu wyrażenia od **warunek1** i **warunek2** do **warunekX** to warunki logiczne, wyrażenia od **wartość1** i **wartość2** do **wartośćX** to wartości, na które odwzorowywane są określone warunki logiczne, a **wartość_dla_else** to wartość używana, jeśli żaden z warunków logicznych nie jest spełniony. Dla każdego wiersza program rozpoczyna sprawdzanie od początku instrukcji **CASE WHEN** i przetwarza pierwszy warunek logiczny. Następnie program sprawdza każdy warunek logiczny, poczynawszy od pierwszego. Po wykryciu pierwszego warunku o wartości **true** instrukcja zwraca wartość powiązaną z tym warunkiem. Jeśli żaden z warunków nie ma wartości **true**, zwracana jest wartość powiązana z blokiem **ELSE**.

Załóżmy, że chcesz zwrócić wszystkie wiersze z klientami z tabeli **customers**. Ponadto chcesz dodać kolumnę, aby oznaczyć klienta jako **VIP**, jeśli jego kod pocztowy to **33111**, lub jako klienta **premium**, jeżeli jego kod to **33124**. Pozostali klienci są oznaczani jako **standardowi**. Nową kolumnę nazwij **customer_type**. Tę tabelę możesz utworzyć za pomocą instrukcji **CASE WHEN**:

```
SELECT
    *,
    CASE WHEN postal_code='33111' THEN 'Klient VIP'
    WHEN postal_code='33124' THEN 'Klient premium'
```



```

ELSE 'Zwykły klient' END
AS customer_type
FROM customers;

```

Ta kwerenda powinna zwrócić dane wyjściowe widoczne na rysunku 2.16.

customer_id bigint	title text	first_name text	last_name text	suffix text	email text	gender text	ip_address text	phone text	street_address text	city text	state text	postal_code text	latitude double precision	longitude double precision	date_added timestamp with time zone	customer_type text	
1	[null]	Arlena	Riveles	[null]	ariveles0...	F	98.36.172.246	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	2017-04-23 00:...	Zwykly klient
2	Dr	Ode	Stovin	[null]	ostovin1...	M	16.97.59.186	314-534-4...	2573 Fordem Parkw...	Saint L...	MO	63116	38.5814	-90.2625	2014-10-02 00:...	Zwykly klient	
3	[null]	Braden	Jordan	[null]	bjordan2...	M	192.86.248.59	[null]	5651 Kennedy Park	Pensac...	FL	32590	30.6143	-87.2758	2018-10-27 00:...	Zwykly klient	
4	[null]	Jessika	Nussen	[null]	jnussen3...	F	159.165.138.166	615-824-2...	224 Village Circle	Nashvil...	TN	37215	36.0986	-86.8219	2017-09-03 00:...	Zwykly klient	
5	[null]	Lonnie	Krembaud	[null]	lkrembaud...	F	18.131.58.65	786-499-3...	38 Lindbergh Way	Miami	FL	33124	25.5584	-80.4582	2014-03-06 00:...	Klient premium	
6	[null]	Cortie	Locksley	[null]	clocksley...	M	140.194.59.82	[null]	6537 Delladonna Dri...	Miami	FL	33158	25.6364	-80.3187	2013-03-31 00:...	Zwykly klient	
7	[null]	Wood	Kennham	[null]	wkennha...	M	191.190.135.172	407-552-6...	001 Onsgard Park	Orlando	FL	32891	28.5663	-81.2608	2011-08-25 00:...	Zwykly klient	
8	[null]	Rutger	Humblestone	[null]	rhumbles...	M	77.10.235.191	203-551-6...	21376 Esker Center	New H...	CT	06510	41.3087	-72.9271	2013-12-15 00:...	Zwykly klient	
9	[null]	Melantha	Tibb	[null]	mtibb8@...	F	155.176.37.197	913-590-8...	05915 Havey Hill	Shawn...	KS	66225	38.8999	2000000000001	2016-02-11 00:...	Zwykly klient	
10	Ms	Barbara-anne	Gowlett	Jr	bgowlett...	F	67.110.62.119	915-714-5...	9 Kim Point	El Paso	TX	79940	31.6948	-106.3	2012-06-28 00:...	Zwykly klient	
11	Mrs	Urbano	Middlehurst	[null]	umiddleh...	M	185.118.6.23	918-339-5...	5203 7th Trail	Tulsa	OK	74156	36.3024	-95.9605	2011-10-22 00:...	Zwykly klient	
12	Mr	Tyne	Duggan	[null]	tdugganb...	F	13.29.231.228	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	2017-10-25 00:...	Zwykly klient

Rysunek 2.16. Kwerenda określająca kategorie klientów

Widać tu, że w tabeli znajduje się kolumna `customer_type`, informująca o kategorii klienta. Instrukcja `CASE WHEN` odwzorowuje kod pocztowy na łańcuch znaków opisujący typ klienta. Za pomocą instrukcji `CASE WHEN` możesz odwzorowywać wartości w dowolny sposób.

Ćwiczenie 2.03 — używanie funkcji `CASE WHEN` do pobierania list klientów z danego regionu

Aby wykonać to ćwiczenie, należy przygotować listę wszystkich klientów odwzorowanych na regiony. Do klientów ze stanów MA, NH, VT, ME, CT i RI należy przypisać etykietę Nowa Anglia. Do klientów ze stanów GA, FL, MS, AL, LA, KY, VA, NC, SC, TN, VI, WV i AR przypisz etykietę Południowy wschód. Klienci z pozostałych stanów powinni mieć przypisaną etykietę Inne.

Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Otwórz wybranego klienta SQL-a i nawiąż połączenie z bazą `sqla`.
2. Utwórz kwerendę, która zwraca kolumnę `customer_id` oraz kolumnę `region`. Stany należy podzielić na kategorie w następujący sposób:

```

SELECT
  c.customer_id,
  CASE WHEN c.state in (
    'MA', 'NH', 'VT', 'ME',
    'CT', 'RI')
  THEN 'Nowa Anglia'
  WHEN c.state in (
    'GA', 'FL', 'MS',
    'AL', 'LA', 'KY', 'VA',
    'NC', 'SC', 'TN', 'VI',
    'WV', 'AR')

```

```

        THEN 'Południowy wschód'
        ELSE 'Inne' END as region
FROM
    customers c
ORDER BY
    1;

```

Ta kwerenda odwzorowuje stany na jeden z regionów na podstawie tego, czy dany stan znajduje się w warunku instrukcji CASE WHEN z określonego wiersza. Powinieneś uzyskać dane wyjściowe widoczne na rysunku 2.17.

customer_id bigint	region text
1	Inne
2	Inne
3	Południowy wschód
4	Południowy wschód
5	Południowy wschód
6	Południowy wschód
7	Południowy wschód
8	Nowa Anglia
9	Inne
10	Inne
11	Inne
12	Inne

Rysunek 2.17. Dane wyjściowe kwerendy określającej regiony

W tych danych wyjściowych do każdego klienta przypisany jest region na podstawie stanu, w którym ta osoba mieszka.

Kod źródłowy z tego fragmentu znajdziesz na stronie <https://packt.live/3dW1ciN>.

W tym ćwiczeniu pokazaliśmy, jak odwzorowywać różne wartości z kolumny na inne wartości z użyciem funkcji CASE WHEN. W następnym punkcie omawiamy przydatną funkcję COALESCE, która pomaga zastępować wartości NULL.

Funkcja COALESCE

Inna przydatna technika polega na zastępowaniu wartości NULL wybraną wartością standardową. Służy do tego funkcja COALESCE. Można w niej podać dowolną liczbę kolumn i wartości skalarnych, a jeśli pierwsza wartość na liście to NULL, funkcja spróbuje ją zastąpić drugą wartością. Funkcja kontynuuje sprawdzanie elementów listy do czasu natrafienia na wartość różną od NULL. Jeżeli wszystkie wartości w funkcji COALESCE to NULL, funkcja zwraca NULL.

Aby przyjrzeć się prostemu zastosowaniu funkcji COALESCE, wróć do tabeli customers. W niektórych rekordach wartość pola phone nie jest podana, co widać na rysunku 2.18.

first_name text	last_name text	phone text
Aaren	Norrey	[null]
Aarika	Mawhinney	205-355-4...
Aarika	Danaher	904-175-3...
Aarika	Guerin	501-121-5...
Aarika	Emmanuel	[null]
Aarika	Chadwell	915-856-7...
Aaron	Glozman	716-972-4...
Aaron	MacAleese	[null]
Aaron	Glazer	814-167-9...
Aaron	Farress	[null]
Ab	Ferraraccio	727-889-1...

Rysunek 2.18. Brakujące dane w polu phone

Załóżmy, że dział marketingu chce uzyskać listę imion, nazwisk i numerów telefonu wszystkich klientów, którzy są mężczyznami. Jeśli numer telefonu klienta jest nieznany, w tabeli należy zapisać wartość 'BRAK NUMERU'. Efekt ten można uzyskać za pomocą funkcji COALESCE:

```
SELECT
    first_name, last_name,
    COALESCE(phone, 'BRAK NUMERU') as phone
FROM
    customers
ORDER BY
    1;
```

Ta kwerenda zwraca wyniki widoczne na rysunku 2.19.

first_name text	last_name text	phone text
Aaren	Whelpdale	607-761-2568
Aaren	Norrey	BRAK NUMERU
Aaren	Sadat	504-559-3464
Aaren	Deeman	BRAK NUMERU
Aaren	Lamlin	414-937-4628
Aarika	Mawhinney	205-355-4381
Aarika	Chadwell	915-856-7492
Aarika	Danaher	904-175-3112
Aarika	Guerin	501-121-5841
Aarika	Emmanuel	BRAK NUMERU
Aaron	Glozman	716-972-4466
Aaron	MacAleese	BRAK NUMERU

Rysunek 2.19. Działanie kwerendy z funkcją COALESCE

Jeśli chcesz tworzyć wartości domyślne i uniknąć wartości NULL, funkcja COALESCE zawsze będzie pomocna.

Funkcja NULLIF

Funkcja NULLIF jest w pewnym sensie przeciwieństwem funkcji COALESCE. NULLIF przyjmuje dwie wartości i zwraca NULL, jeśli pierwsza wartość jest równa drugiej.

Wyobraź sobie, że dział marketingu przygotował nowy tekst przesyłek reklamowych rozsyłanych do klientów. Jednym z dziwnych aspektów nowego tekstu reklamowego jest to, że nie są w nim akceptowane tytuły (Pan, Mgr, Dr itd.) dłuższe niż trzyliterowe. Jednak niektóre tytuły mają więcej liter. Skoro system ich nie akceptuje, należy je usunąć w trakcie pobierania wyników.

W przykładowej bazie jedyny tytuł mający więcej niż trzy znaki to 'Honorable'. Dział marketingu oczekuje, że przygotujesz listę wysyłkową zawierającą poprawne adresy i zastąpisz na niej wszystkie wystąpienia tytułu 'Honorable' wartością NULL. Można to zrobić za pomocą następującej kwerendy:

```
SELECT customer_id,
       NULLIF(title, 'Honorable') as title,
       first_name,
       last_name,
       suffix,
       email,
       gender,
```

```

        ip_address,
        phone,
        street_address,
        city,
        state,
        postal_code,
        latitude,
        longitude,
        date_added
FROM
    customers c
ORDER BY
    1;

```

Ten kod usuwa wszystkie wystąpienia słowa 'Honorable' z kolumny title, co pokazano na rysunku 2.20.

Explain	Data Output	Notifications	Messages	Query History	Query Editor							
	customer_id bigint	title text	first_name text	last_name text	suffix text	email text	gender text	ip_address text	phone text	street_address text		
1		1	[null]	Arlena	Riveles	[null]	ariveles0...	F	98.36.172.246	[null]	[null]	
2		2	Dr	Ode	Stovin	[null]	ostovin1...	M	16.97.59.186	314-534-4...	2573 Fordem Parkw...	
3		3	[null]	Braden	Jordan	[null]	bjordan2...	M	192.86.248.59	[null]	5651 Kennedy Park	
4		4	[null]	Jessika	Nussen	[null]	jnussen3...	F	159.165.138.166	615-824-2...	224 Village Circle	
5		5	[null]	Lonnie	Rembaud	[null]	lrembaud...	F	18.131.58.65	786-499-3...	38 Lindbergh Way	
6		6	[null]	Cortie	Locksley	[null]	clocksley...	M	140.194.59.82	[null]	6537 Delladonna Dri...	
7		7	[null]	Wood	Kennham	[null]	wkenna...	M	191.190.135.172	407-552-6...	001 Onsgard Park	
8		8	[null]	Rutger	Humblestone	[null]	rhumbles...	M	77.10.235.191	203-551-6...	21376 Esker Center	
9		9	[null]	Melantha	Tibb	[null]	mtibb8@...	F	155.176.37.197	913-590-8...	05915 Havey Hill	
10		10	Ms	Barbara-anne	Gowlett	Jr	bgowlett...	F	67.110.62.119	915-714-5...	9 Kim Point	
11		11	Mrs	Urbano	Middlehurst	[null]	umiddleh...	M	185.118.6.23	918-339-5...	5203 7th Trail	
12		12	Mr	Tyne	Duggan	[null]	tdugganb...	F	13.29.231.228	[null]	[null]	
13		13	[null]	Gannon	Braker	[null]	gbrakerc...	M	69.199.173.60	619-666-7...	1 Columbus Drive	
14		14	[null]	Derry	Lyburn	[null]	dlyburnd...	M	230.59.185.87	501-457-5...	8507 Garrison Junct...	
15		15	[null]	Nichols	Espinay	[null]	nespinay...	M	243.147.74.203	818-658-6...	43 Anthes Road	

Rysunek 2.20. Działanie kwerendy NULLIF

Dalej omawiamy funkcje LEAST i GREATEST.

Funkcje LEAST i GREATEST

Dwie funkcje przydatne w trakcie przygotowywania danych to LEAST i GREATEST. Każda z nich przyjmuje dowolną liczbę wartości i zwraca najmniejszą lub największą z nich.

Przykładowo jeśli użyjesz funkcji LEAST z dwoma parametrami, na przykład 600 i 900, zwrócona zostanie wartość 600. Funkcja GREATEST działa odwrotnie. Parametrami tych funkcji mogą być literały liczbowe lub wartości zapisane w polach liczbowych.

Prostym zastosowaniem tych funkcji jest zastępowanie wartości, jeśli są zbyt wysokie lub za niskie. Załóżmy, że zespół sprzedażowy chce przygotować listę transakcji dotyczących skuterów z ceną 600 dolarów lub niższą. Możesz utworzyć taką listę za pomocą następującej kwerendy:

```
SELECT
    product_id, model,
    year, product_type,
    LEAST(600.00, base_msrp) as base_msrp,
    production_start_date,
    production_end_date
FROM
    products
WHERE
    product_type='scooter'
ORDER BY
    1;
```

Ta kwerenda powinna zwrócić dane wyjściowe widoczne na rysunku 2.21.

	product_id bigint	model text	year bigint	product_type text	base_msrp numeric	production_start_date timestamp without time zone	production_end_date timestamp without time zone
1	1	Lemon	2010	scooter	399.99	2010-03-03 00:00:00	2012-06-08 00:00:00
2	2	Lemon ...	2011	scooter	600.00	2011-01-03 00:00:00	2011-03-30 00:00:00
3	3	Lemon	2013	scooter	499.99	2013-05-01 00:00:00	2018-12-28 00:00:00
4	5	Blade	2014	scooter	600.00	2014-06-23 00:00:00	2015-01-27 00:00:00
5	7	Bat	2016	scooter	599.99	2016-10-10 00:00:00	[null]
6	8	Bat Limi...	2017	scooter	600.00	2017-02-15 00:00:00	[null]
7	12	Lemon ...	2019	scooter	349.99	2019-02-04 00:00:00	[null]

Rysunek 2.21. Tanie skutery

Funkcja CASTING

Inną przydatną transformacją danych jest zmiana typu danych kolumny w kwerendzie. Zwykle robi się to po to, aby móc użyć funkcji dostępnej tylko dla jednego typu danych, na przykład dla danych tekstowych, a pierwotnie kolumna jest innego typu danych (na przykład jest kolumną liczbową). W celu zmiany typu danych kolumny wystarczy zastosować składnię kolumna::typ_danych, gdzie kolumna to nazwa kolumny, a typ_danych to typ danych, który ma mieć ta kolumna. Na przykład aby zmienić typ danych kolumny year w tabeli products na typ TEXT, użyj następującej kwerendy:

```
SELECT
    product_id, model,
    year::TEXT, product_type,
    base_msrp, production_start_date,
    production_end_date
FROM
    products;
```

Ta kwerenda zwraca dane wyjściowe widoczne na rysunku 2.22.

product_id bigint	model text	year text	product_type text	base_msrp text	production_start_date timestamp without time zone	production_end_date timestamp without time zone
4	Model ...	2014	automobile	115,000.00	2014-06-23 00:00:00	2018-12-28 00:00:00
6	Model S...	2015	automobile	65,500.00	2015-04-15 00:00:00	2018-10-01 00:00:00
9	Model E...	2017	automobile	35,000.00	2017-02-15 00:00:00	[null]
10	Model ...	2017	automobile	85,750.00	2017-02-15 00:00:00	[null]
11	Model ...	2019	automobile	95,000.00	2019-02-04 00:00:00	[null]
12	Lemon ...	2019	scooter	349.99	2019-02-04 00:00:00	[null]
13	Nimbus...	2019	scooter	500.00	2019-03-03 00:00:00	2020-03-03 00:00:00
1	Lemon	2010	scooter	299.99	2010-03-03 00:00:00	2012-06-08 00:00:00
2	Lemon ...	2011	scooter	299.99	2011-01-03 00:00:00	2011-03-30 00:00:00
3	Lemon	2013	scooter	299.99	2013-05-01 00:00:00	2018-12-28 00:00:00
5	Blade	2014	scooter	299.99	2014-06-23 00:00:00	2015-01-27 00:00:00
7	Bat	2016	scooter	299.99	2016-10-10 00:00:00	[null]

Rysunek 2.22. Kolumna year jako kolumna typu text

To powoduje przekształcenie zawartości kolumny year na tekst. Teraz możesz stosować do tak zmodyfikowanej kolumny funkcje tekstowe. Jest jednak pewien haczyk: nie wszystkie typy danych umożliwiają konwersję na określony typ. Na przykład typ datetime nie umożliwia konwersji na typy z rodziny float. Próba wykonania nieoczekiwanej, dziwnej konwersji spowoduje, że klient SQL-a zgłosi błąd.

Funkcje DISTINCT i DISTINCT ON

W trakcie badania zbioru danych często przydatne jest ustalenie unikatowych wartości w kolumnie lub w grupie kolumn. Jest to główne zastosowanie słowa kluczowego DISTINCT. Jeśli chcesz poznać wszystkie unikatowe lata produkcji modeli z tabeli products, możesz użyć następującej kwerendy:

```
SELECT DISTINCT year
FROM products
ORDER BY 1;
```

Powinieneś uzyskać wyniki widoczne na rysunku 2.23.

Możesz zastosować te funkcje do wielu kolumn, aby uzyskać wszystkie unikatowe kombinacje wartości z tych kolumn. Na przykład aby znaleźć wszystkie unikatowe lata i rodzaje produktów wprowadzonych na rynek w tych latach, możesz zastosować taki kod:

```
SELECT DISTINCT year, product_type
FROM products
ORDER BY 1, 2;
```

Powinieneś otrzymać dane wyjściowe widoczne na rysunku 2.24.

year bigint
2010
2011
2013
2014
2015
2016
2017
2019

Rysunek 2.23. Unikatowe lata produkcji modeli

year bigint	product_type text
2010	scooter
2011	scooter
2013	scooter
2014	automobile
2014	scooter
2015	automobile
2016	scooter
2017	automobile
2017	scooter
2019	automobile
2019	scooter

Rysunek 2.24. Unikatowe lata produkcji modeli i typy wytwarzanych w nich produktów

Z `DISTINCT` powiązane jest słowo kluczowe `DISTINCT ON`, które pozwala zagwarantować zwrócenie tylko po jednym wierszu z poszczególnymi wartościami ze wskazanych kolumn. Oto ogólna składnia kwerend `DISTINCT ON`:

```
SELECT DISTINCT ON (unikatowa_kolumna)
    kolumna_1,
    kolumna_2,
    ...
    kolumna_n
FROM tabela
ORDER BY kolumna_sortowania;
```


Tu unikatowa_kolumna to kolumna (lub zbiór kolumn), z której wartości mają być unikatowe, kolumny od kolumna_1 do kolumna_n to kolumny, które mają znaleźć się w wynikach kwerendy, a kolumna_sortowania to kolumna służąca do określenia pierwszego wiersza zwróconego przez kwerendę DISTINCT ON, jeśli w kilku wierszach w kolumnie unikatowa_kolumna znajduje się ta sama wartość.

Jako kolumna_sortowania pierwsza powinna być wymieniona unikatowa_kolumna. Jeśli kwerenda nie zawiera klauzuli ORDER BY, pierwszy wiersz zostanie wybrany losowo. Załóżmy, że chcesz uzyskać listę sprzedawców z tabeli salespeople, na której to liście każda osoba ma unikatowe imię. Wtedy jeśli dwóch sprzedawców ma to samo imię, zwrócony zostanie ten, który zaczął pracę w firmie wcześniej. Oto potrzebna kwerenda:

```
SELECT DISTINCT ON (first_name)
*
FROM
  salespeople
ORDER BY
  first_name, hire_date;
```

Powinna ona zwrócić dane wyjściowe pokazane na rysunku 2.25.

salesperson_id bigint	dealership_id bigint	title text	first_name text	last_name text	suffix text	username text	gender text	hire_date timestamp without time zone	termination_date timestamp without time zone
189	17	[null]	Abby	Drewery	[null]	adrewery58	Male	2015-09-01 00:00:00	[null]
137	4	[null]	Abie	Brydell	[null]	abrydell3s	Male	2016-11-04 00:00:00	[null]
27	4	[null]	Ad	Loding	[null]	alodingq	Male	2017-06-27 00:00:00	[null]
63	2	[null]	Adrianne	Otham	[null]	aotham1q	Female	2014-12-20 00:00:00	[null]
272	7	[null]	Afton	Limon	[null]	alimon7j	Female	2014-09-01 00:00:00	[null]
35	17	[null]	Agnella	Linke	[null]	alinkey	Female	2018-10-23 00:00:00	[null]
161	18	[null]	Aile	Dobbing	[null]	adobbing4g	Female	2014-08-14 00:00:00	2016-10-03 00:00:00
136	3	[null]	Alanna	Dufaire	[null]	adufaire3r	Female	2014-06-27 00:00:00	[null]
147	6	[null]	Alaric	Sterrick	[null]	asterrick42	Male	2014-06-17 00:00:00	[null]
221	19	[null]	Alberik	Polglase	[null]	apolglase64	Male	2015-11-19 00:00:00	[null]
139	18	[null]	Alexina	Coatsworth	[null]	acoatswort...	Female	2015-07-27 00:00:00	[null]
100	18	[null]	Alie	Bellfield	[null]	abellfield2r	Female	2017-12-11 00:00:00	[null]
287	7	[null]	Allayne	Billingham	[null]	abillingham7y	Male	2014-08-06 00:00:00	[null]

Rysunek 2.25. Kwerenda DISTINCT ON dla kolumny first_name

Teraz wiadomo, że w każdym wierszu znajduje się inne imię. Jeśli jest kilku sprzedawców o tym samym imieniu, kwerenda pobiera osobę, która została zatrudniona w firmie najwcześniej. Na przykład jeśli w tabeli salespeople znajduje się kilka wierszy z imieniem 'Abby', wiersz z rysunku 2.25 z imieniem 'Abby' (pierwszy wiersz w danych wyjściowych) zawiera dane pierwszej osoby o tym imieniu zatrudnionej w firmie. Gdy jest dwóch pracowników o tym samym imieniu, kwerenda porządkuje ich według daty rozpoczęcia pracy. Na przykład jeśli w bazie znajdują się pracownicy Andrey Haack z datą zatrudnienia 2016-01-10 i Andrey Kures z datą zatrudnienia 2016-05-17, Andrey Haack zostanie wyświetlony jako pierwszy, ponieważ ma wcześniejszą datę rozpoczęcia pracy.

W tym punkcie znajdziesz zadanie ilustrujące, jak użyć SQL-a do utworzenia zbioru danych dla modelu.

Zadanie 2.01 — używanie SQL-a do tworzenia modelu wspomagającego sprzedaż

W tym zadaniu oczyścisz i przygotujesz dane na potrzeby analiz za pomocą SQL-a. Zespół odpowiedzialny za data science chce zbudować nowy model do pomocy w prognozowaniu, którzy klienci z największym prawdopodobieństwem ponownie dokonają zakupu. Do zespołu dołączył nowy specjalista. Ty odpowiadasz za to, by pomóc mu w przygotowaniu i zbudowaniu zbioru danych na potrzeby treningu modelu. Napisz kwerendę, która utworzy zbiór danych. Oto kroki, jakie należy wykonać:

1. Otwórz klienta SQL-a i nawiąż połączenie z bazą danych.
2. Użyj złączenia INNER JOIN, aby złączyć tabelę customers z tabelą sales.
3. Użyj złączenia INNER JOIN, aby złączyć tabelę products z tabelą sales.
4. Użyj złączenia LEFT JOIN, aby złączyć tabelę dealerships z tabelą sales.
5. Zwróć wszystkie kolumny tabel customers i products.
6. Zwróć kolumnę dealership_id tabeli sales. Jeśli ta kolumna ma wartość NULL, zastąp ją wartością -1.
7. Dodaj kolumnę high_savings, która zwraca 1, jeśli wartość sprzedaży była o co najmniej 500 niższa od ceny bazowej (base_msrp). W przeciwnym razie należy zwrócić 0. Kwerenda musi być wykonywana na złączonej tabeli.

Oczekiwane dane wyjściowe są pokazane na rysunku 2.26.

customer_id bigint	title text	first_name text	last_name text	suffix text	email text	gender text	ip_address text	phone text	street_address text	city text	state text	postal_code text	latitude double precision	longitude double precision	date_added timestamp without time zone
1	[null]	Arlena	Riveles	[null]	arivele...	F	98.36.172.246	[null]	[null]	[null]	[null]	[null]	[null]	[null]	2017-04-23 00:00:00
4	[null]	Jessika	Nussen	[null]	jnusse...	F	159.165.138...	615-824...	224 Village Circle	Nash...	TN	37215	36.0986	-86.8219	2017-09-03 00:00:00
5	[null]	Lonnie	Rembaud	[null]	lremba...	F	18.131.58.65	786-499...	38 Lindbergh Way	Miami	FL	33124	25.5584	-80.4582	2014-03-06 00:00:00
6	[null]	Cortie	Locksley	[null]	clocks...	M	140.194.59.82	[null]	6537 Delladonna ...	Miami	FL	33158	25.6364	-80.3187	2013-03-31 00:00:00
7	[null]	Wood	Kennham	[null]	wkenn...	M	191.190.135...	407-552...	001 Onsgard Park	Orla...	FL	32891	28.5663	-81.2608	2011-08-25 00:00:00
7	[null]	Wood	Kennham	[null]	wkenn...	M	191.190.135...	407-552...	001 Onsgard Park	Orla...	FL	32891	28.5663	-81.2608	2011-08-25 00:00:00
7	[null]	Wood	Kennham	[null]	wkenn...	M	191.190.135...	407-552...	001 Onsgard Park	Orla...	FL	32891	28.5663	-81.2608	2011-08-25 00:00:00
11	Mrs	Urbano	Middlehurst	[null]	umiddl...	M	185.118.6.23	918-339...	5203 7th Trail	Tulsa	OK	74156	36.3024	-95.9605	2011-10-22 00:00:00
12	Mr	Tyne	Duggan	[null]	tdugga...	F	13.29.231.228	[null]	[null]	[null]	[null]	[null]	[null]	[null]	2017-10-25 00:00:00

Rysunek 2.26. Tworzenie kwerendy do budowania modelu wspomagającego sprzedaż

Rozwiązanie tego zadania znajdziesz w „Dodatku”.

Teraz już wiesz, jak używać SQL-a do oczyszczania i porządkowania danych na potrzeby analiz.

Podsumowanie

SQL udostępnia wiele narzędzi do łączenia i oczyszczania danych. Wiesz już, że złączenia umożliwiają użytkownikom łączenie wielu tabel, a słowo kluczowe `UNION` i podkwerendy pozwalają łączyć wyniki wielu kwerend. Nauczyłeś się też, że SQL udostępnia wiele funkcji i słów kluczowych umożliwiających użytkownikom odwzorowywanie nowych danych, uzupełnianie brakujących wartości oraz usuwanie duplikatów. Słowa kluczowe takie jak `CASE WHEN`, `COALESCE`, `NULLIF` i `DISTINCT` umożliwiają szybkie i proste wprowadzanie zmian w danych.

Teraz, gdy wiesz już, jak przygotować zbiór danych, w następnym rozdziale zaczniesz wyciągać wnioski analityczne, używając agregacji i funkcji okna.

Agregacja i funkcje okna

W tym rozdziale poznasz logikę agregacji i funkcji okna, napiszesz kod w SQL-u z wykorzystaniem tych mechanizmów oraz zmodyfikujesz ich działanie, używając słów kluczowych `HAVING` i `GROUP BY`.

Dzięki lekturze tego rozdziału nauczysz się stosować opisane mechanizmy do wyciągania wniosków z danych i poznawania cech zbiorów danych (na przykład ich jakości).

Wprowadzenie

W poprzednim rozdziale opisaliśmy, jak używać SQL-a do przygotowywania zbiorów danych na potrzeby analiz. Po przygotowaniu danych następnym krokiem jest ich analiza. Specjaliści od data science i analizy danych zwykle starają się zrozumieć dane przez ich podsumowanie oraz znaleźć wysokopoziomowe wzorce. W SQL-u pomocne do wykonywania tych zadań są przede wszystkim agregacje i funkcje okna. Przyjmują one dane wejściowe w postaci wielu wierszy i na ich podstawie zwracają nowe informacje. Najpierw przyjrzyj się funkcjom agregującym.

Funkcje agregujące

Często celem jest zrozumienie cech całej kolumny lub tabeli, a nie samo przejrzanie poszczególnych wierszy danych. Oto prosty przykład: założmy, że zastanawiasz się, ilu klientów ma firma ZoomZoom. Możesz pobrać wszystkie dane z tabeli, a następnie sprawdzić, ile wierszy zostało zwróconych. Jest to jednak bardzo żmudne rozwiązanie. Na szczęście SQL udostępnia funkcje, które można wykorzystać do wykonywania obliczeń na dużych grupach wierszy. Są to **funkcje agregujące**. Przyjmują one kolumny z wieloma wierszami i na ich podstawie

zwracają liczbę. W ramach przykładu użyj funkcji COUNT do zliczenia klientów firmy ZoomZoom. Tu funkcja zlicza wiersze z tabeli customers:

```
SELECT COUNT(customer_id) FROM customers;
```

Funkcja COUNT zwraca liczbę wierszy z pominięciem tych, które w danej kolumnie mają wartość NULL. Ponieważ kolumna customer_id jest kluczem głównym i nie może być równa NULL, funkcja COUNT zwraca tu liczbę wierszy tabeli. W tym przykładzie kwerenda zwróci następujące dane wyjściowe:

```
count
bigint
-----
50000
```

Funkcja COUNT działa tutaj dla jednej kolumny i zlicza wartości różne od NULL. Jeśli jednak w każdej kolumnie znajduje się choć jedna wartość NULL, nie da się określić w ten sposób liczby wierszy. Aby ustalić liczbę wierszy w takim scenariuszu, możesz użyć funkcji COUNT z gwiazdką, (*), która pozwoli otrzymać łączną liczbę wierszy:

```
SELECT
  COUNT(*)
FROM
  customers;
```

Także ta kwerenda zwróci wartość 50000.

Jeśli interesuje Cię liczba unikatowych stanów z listy klientów, możesz ją uzyskać za pomocą kwerendy z kodem COUNT (DISTINCT wyrażenie):

```
SELECT
  COUNT(DISTINCT state)
FROM
  customers;
```

Ta kwerenda zwróci następujące dane wyjściowe:

```
count
bigint
-----
51
```

W tabeli z rysunku 3.1 opisane są podstawowe funkcje agregujące używane w SQL-u.

Funkcji agregujących można też używać w klauzuli WHERE do obliczania zagregowanych wartości dla określonych podzbiorów danych. Jeśli na przykład chcesz sprawdzić, ilu klientów firma ZoomZoom ma w Kalifornii, możesz posłużyć się następującą kwerendą:

```
SELECT
  COUNT(*)
FROM
  customers
WHERE
  state='CA';
```

Funkcja	Objaśnienie
COUNT(kolumnaX)	Zlicza wiersze o wartości różnej od NULL w kolumnaX.
COUNT(*)	Zlicza wiersze tabeli wyjściowej.
MIN(kolumnaX)	Zwraca minimalną wartość w kolumnaX. W kolumnach tekstowych jest to wartość występująca jako pierwsza w porządku alfabetycznym.
MAX(kolumnaX)	Zwraca wartość maksymalną z kolumnaX.
SUM(kolumnaX)	Zwraca sumę wszystkich wartości z kolumnaX.
AVG(kolumnaX)	Zwraca średnią wszystkich wartości z kolumnaX.
STDDEV(kolumnaX)	Sprawdza odchylenie standardowe dla próbki na podstawie wszystkich wartości kolumny.
VAR(kolumnaX)	Zwraca wariancję dla próbki na podstawie wszystkich wartości z kolumnaX.
REGR_SLOPE(kolumnaX, kolumnaY)	Zwraca nachylenie linii regresji dla zmiennej zależnej (kolumnaX) i zmiennej niezależnej (kolumnaY).
REGR_INTERCEPT(kolumnaX, kolumnaY)	Zwraca punkt przecięcia linii regresji z osią dla zmiennej zależnej (kolumnaX) i zmiennej niezależnej (kolumnaY).
CORR(kolumnaX, kolumnaY)	Oblicza współczynnik korelacji Pearsona dla danych z kolumnaX i kolumnaY.

Rysunek 3.1. Podstawowe funkcje agregujące

Uzyskasz następujące dane wyjściowe:

```
count
bigint
-----
5038
```

Na wynikach funkcji agregujących możesz też wykonywać operacje arytmetyczne. W tej kwerendzie liczba wierszy tabeli customers jest dzielona przez 2:

```
SELECT
  COUNT(*)/2
FROM
  customers;
```

Ta kwerenda zwróci wartość 25000.

Możesz także łączyć funkcje agregujące za pomocą operacji matematycznych. W poniższej kwerendzie zamiast używać funkcji AVG do obliczenia średniej sugerowanej ceny detalicznej produktów firmy ZoomZoom możesz „zbudować” funkcję AVG, używając funkcji SUM i COUNT:

```
SELECT SUM(base_msrp)::FLOAT/COUNT(*) AS avg_base_msrp FROM products
```

Powinieneś otrzymać następujący wynik:

```
Avg_base_msrp
Double precision
-----
50000
```

Sumę trzeba rzutować, ponieważ PostgreSQL traktuje dzielenie liczb całkowitych inaczej niż dzielenie liczb zmiennoprzecinkowych. Na przykład dzielenie całkowitoliczbowe 7 przez 2 daje w PostgreSQL wynik 3. Aby uzyskać dokładniejszy wynik, 3,5, należy rzutować jedną z liczb na typ FLOAT.

W następnym punkcie zobaczysz, jak używać funkcji agregujących do analizy danych.

We wszystkich ćwiczeniach z tej książki używane jest narzędzie pgAdmin 4. Wszystkie ćwiczenia i zadania są dostępne w serwisie GitHub na stronie <https://packt.live/2XUYdla>.

Ćwiczenie 3.01 — używanie funkcji agregujących do analizowania danych

W tym ćwiczeniu przeanalizujesz ceny produktów i wykonasz na nich obliczenia z zastosowaniem różnych funkcji agregujących. Ponieważ ciekawią Cię dane dotyczące Twojej firmy, chcesz poznać podstawowe statystyki na temat cen produktów ZoomZoom. Celem jest ustalenie dla wszystkich produktów sprzedanych kiedykolwiek przez firmę najniższej, najwyższej i średniej ceny, a także odchylenia standardowego.

Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Otwórz wybranego klienta SQL-a i nawiąż połączenie z bazą sql da.
2. Oblicz najniższą, najwyższą i średnią cenę oraz odchylenie standardowe na podstawie tabeli products. Użyj do tego funkcji agregujących MIN, MAX, AVG i STDDEV:

```
SELECT
    MIN(base_msrp), MAX(base_msrp),
    AVG(base_msrp), STDDEV(base_msrp)
FROM products;
```

Ten kod powinien zwrócić dane wyjściowe podobne do tych z rysunku 3.2.

min numeric	max numeric	avg numeric	stddev numeric
349.99	115000.00	33358.327500000000	44484.40866379

Rysunek 3.2. Statystyki dotyczące cen produktów

U Ciebie wyniki mogą wyglądać inaczej niż w tych danych wyjściowych. Powodem są możliwe różnice w konfiguracji liczby wyświetlanych cyfr po przecinku w systemie PostgreSQL, a także modyfikacje danych względem pierwotnej bazy wygenerowanej na podstawie pliku *dump*. Jednak głównym celem jest tu pokazanie, jak stosować funkcje agregujące do analizowania danych.

W tych danych wyjściowych widać, że cena minimalna to 349,99, cena maksymalna jest równa 115 000,00, cena średnia wynosi 33 358,32750, a odchylenie standardowe dla cen to 44 484,408.

Kod źródłowy z tego fragmentu znajdziesz na stronie <https://packt.live/30GGU8W>.

W tym ćwiczeniu zastosowaliśmy funkcje agregujące, aby przedstawić podstawowe statystyki dotyczące cen. Dalej użyjesz funkcji agregujących z klauzulą GROUP BY.

Funkcje agregujące z klauzulą GROUP BY

Do tej pory wykorzystywaliśmy funkcje agregujące do obliczania statystyk dla całych kolumn. Jednak często potrzebne są wartości zagregowane nie dla całej tabeli, lecz dla mniejszych jej fragmentów. Aby się o tym przekonać, wróć do tabeli customers. Wiesz, że łączna liczba klientów wynosi 50 000. Możliwe, że chcesz ustalić, ilu klientów pochodzi z poszczególnych stanów. Jak uzyskać takie dane?

Listę stanów można otrzymać za pomocą następującej kwerendy:

```
SELECT DISTINCT
  state
FROM
  customers;
```

Ta kwerenda powinna zwrócić listę 52 stanów.

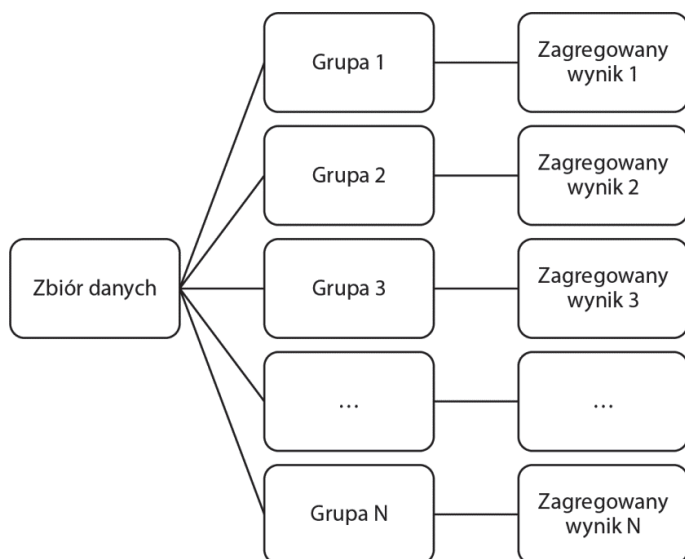
Po uzyskaniu listy stanów możesz uruchomić dla każdego z nich następującą kwerendę:

```
SELECT
  COUNT(*)
FROM
  customers
WHERE
  state='{state}'
```

Choć jest to wykonalne, ta technika jest niezwykle żmudna, a przy dużej liczbie stanów wymaga sporo czasu. Klauzula GROUP BY pozwala utworzyć dużo wygodniejsze rozwiązanie.

Klauzula GROUP BY

Klauzula GROUP BY dzieli wiersze zbioru danych na wiele grup na podstawie podanego w niej klucza. Następnie do wszystkich wierszy z każdej grupy stosowana jest funkcja agregująca, co pozwala uzyskać jedną wartość dla każdej z grup. W efekcie w danych wyjściowych SQL-a dla każdej grupy wyświetlane są klucz z klauzuli GROUP BY i zagregowana wartość. Ogólny schemat tego procesu jest pokazany na rysunku 3.3



Rysunek 3.3. Ogólny model obliczeń z użyciem klauzuli GROUP BY

Na tym rysunku widać, że zbiór danych obejmuje kilka grup (Grupa 1, Grupa 2, ..., Grupa N). Funkcja agregująca zastosowana do wszystkich wierszy z Grupa 1 daje Zagregowany wynik 1, funkcja agregująca zastosowana do wszystkich wierszy z Grupa 2 daje Zagregowany wynik 2 itd.

Instrukcje GROUP BY zwykle mają następującą strukturę:

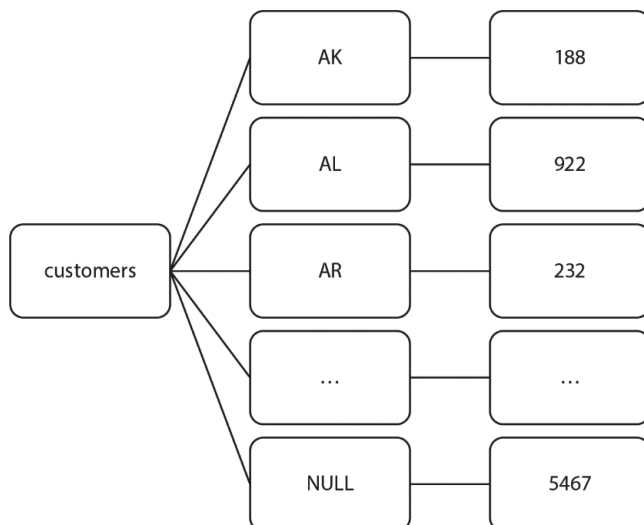
```
SELECT {KLUCZ}, {FUNK_AGREG(kolumna1)} FROM {tabela1} GROUP BY {KLUCZ}
```

Tu {KLUCZ} to kolumna (lub wynik funkcji) używana do tworzenia grup, {FUNK_AGREG(kolumna1)} to funkcja agregująca wykonywana dla danej kolumny dla wszystkich wierszy z każdej grupy, a {tabela1} to tabela lub zbiór złączonych tabel, z których pochodzą wiersze dzielone na grupy.

Aby zrozumieć tę technikę, zlicz klientów z każdego stanu USA, korzystając z kwerendy GROUP BY. Przy jej użyciu można za pomocą SQL-a zliczyć klientów z poszczególnych stanów, uruchamiając taką kwerendę:

```
SELECT state, COUNT(*)
FROM
  customers
GROUP BY
  state
```

Model przetwarzania jest pokazany na rysunku 3.4.



Rysunek 3.4. Model przetwarzania zastosowany do ustalenia liczby klientów w każdym stanie

Tu AK, AL, AR i inne klucze to skróty nazw stanów USA.

Powinieneś otrzymać dane wyjściowe widoczne na rysunku 3.5.

state text	count bigint
KS	619
[null]	5467
CA	5038
NH	77
OR	386
ND	93
TX	4865
NV	643
KY	598

Rysunek 3.5. Dane wyjściowe kwerendy zwracającej liczbę klientów w poszczególnych stanach

W celu wykonania operacji GROUP BY możesz też użyć numeru kolumny:

```
SELECT
    state,
    COUNT(*)
FROM
    customers
GROUP BY
    1
```

Jeśli chcesz zwrócić dane wyjściowe w kolejności alfabetycznej, użyj następującej kwerendy:

```
SELECT
    state,
    COUNT(*)
FROM
    customers
GROUP BY
    state
ORDER BY
    state
```

A oto inne rozwiązanie:

```
SELECT state, COUNT(*) FROM customers GROUP BY 1 ORDER BY 1
```

Obie te kwerendy dają wyniki widoczne na rysunku 3.6.

state text	count bigint
AK	188
AL	922
AR	232
AZ	931
CA	5038
CO	1042
CT	576
DC	1447
DE	149

Rysunek 3.6. Dane wyjściowe kwerendy zwracającej liczbę klientów z poszczególnych stanów, gdzie stany są uporządkowane alfabetycznie

Często potrzebna jest kolejność zagregowanych wartości. Można je uporządkować za pomocą klauzuli `ORDER BY`:

```
SELECT
    state, COUNT(*)
FROM
    customers
GROUP BY
    state
ORDER BY
    COUNT(*)
```

Ta kwerenda zwraca dane wyjściowe z rysunku 3.7.

state text	count bigint
VT	16
WY	23
ME	25
RI	47
NH	77
ND	93
MT	122
SD	124
DE	149

Rysunek 3.7. Dane wyjściowe kwerendy zwracającej uporządkowaną rosnąco liczbę klientów z poszczególnych stanów

Możesz też zliczyć tylko podzbiór danych, na przykład łączną liczbę klientów, którzy są mężczyznami. Aby ją uzyskać, użyj następującej kwerendy:

```
SELECT
    state, COUNT(*)
FROM
    customers
WHERE
    gender='M'
GROUP BY
    state
ORDER BY
    state
```

Otrzymasz dane wyjściowe z rysunku 3.8.

state text	count bigint
AK	87
AL	489
AR	120
AZ	415
CA	2572
CO	526
CT	301
DC	713
DE	74

Rysunek 3.8. Dane wyjściowe kwerendy zwracającej liczbę klientów będących mężczyznami z poszczególnych stanów; dane są uporządkowane alfabetycznie

Widać tu, że grupowanie danych według jednej kolumny pozwala uzyskać cenne informacje. W następnym punkcie zobaczysz, że klauzulę `GROUP BY` można zastosować do wielu kolumn, aby otrzymać bardziej precyzyjne informacje.

Klauzula `GROUP BY` dla kilku kolumn

Choć klauzula `GROUP BY` bazująca na jednej kolumnie daje duże możliwości, możesz pójść o krok dalej i pogrupować dane według kilku kolumn. Załóżmy, że chcesz ustalić nie tylko liczbę klientów firmy ZoomZoom z każdego stanu, ale też liczbę klientów według płci z poszczególnych stanów. Klauzula `GROUP BY` z kilkoma kolumnami pozwala uzyskać potrzebne wyniki:

```
SELECT
    state, gender, COUNT(*)
FROM
    customers
GROUP BY
    state, gender
ORDER BY
    state, gender
```

Otrzymasz wyniki widoczne na rysunku 3.9.

state text	gender text	count bigint
AK	F	101
AK	M	87
AL	F	433
AL	M	489
AR	F	112
AR	M	120
AZ	F	516
AZ	M	415
CA	F	2466

Rysunek 3.9. Dane wyjściowe kwerendy zwracającej liczbę klientów z podziałem na stany i płcie; dane są uporządkowane alfabetycznie

W klauzuli `GROUP BY` można podać w ten sposób dowolną liczbę kolumn. Teraz zastosujesz tę klauzulę w ćwiczeniu.

Ćwiczenie 3.02 — obliczanie cen dla typów produktów za pomocą klauzuli GROUP BY

W tym ćwiczeniu przeanalizujesz i obliczysz ceny produktów za pomocą funkcji agregujących i klauzuli GROUP BY. Menedżer ds. marketingu chce poznać minimum, maksimum, średnią i odchylenie standardowe dla cen poszczególnych typów produktów sprzedawanych przez firmę ZoomZoom. Dane będą potrzebne w kampanii marketingowej. Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Otwórz wybranego klienta SQL-a i nawiąż połączenie z przykładową bazą sql da.
2. Oblicz najniższą, najwyższą i średnią cenę oraz odchylenie standardowe, używając funkcji agregujących MIN, MAX, AVG i STDDEV do kolumn tabeli products; posłuż się klauzulą GROUP BY do sprawdzenia ceny produktów różnego typu:

```
SELECT
    product_type, MIN(base_msrp),
    MAX(base_msrp), AVG(base_msrp),
    STDDEV(base_msrp)
FROM
    products
GROUP BY
    1
ORDER BY
    1;
```

Powinieneś uzyskać wyniki z rysunku 3.10.

product_type text	min numeric	max numeric	avg numeric	stddev numeric
automobile	35000.00	115000.00	79250.000000000000	30477.45068079
scooter	349.99	799.99	578.5614285714285714	167.971085947212

Rysunek 3.10. Podstawowe statystyki dotyczące cen z podziałem na typy produktów

Na podstawie tych danych wyjściowych menedżer ds. marketingu może sprawdzić i porównać na potrzeby kampanii marketingowej ceny produktów sprzedawanych przez firmę ZoomZoom.

Kod źródłowy dotyczący tego fragmentu znajdziesz na stronie <https://packt.live/2Yv6JpM>.

W tym ćwiczeniu obliczyliśmy podstawowe statystyki dla typów produktów, używając funkcji agregujących i klauzuli GROUP BY. Teraz nauczysz się stosować klauzulę GROUPING SETS.

Klauzula GROUPING SETS

Załóżmy, że chcesz zliczyć klientów z poszczególnych stanów, a jednocześnie w tej samej funkcji agregującej określić łączną liczbę kobiet i mężczyzn z każdego stanu. Możesz w tym celu zastosować słowo kluczowe `UNION ALL` opisane w rozdziale 1., „Wprowadzenie do SQL-a dla analityków”:

```
(
SELECT
    state,
    NULL as gender,
    COUNT(*)
FROM
    customers
GROUP BY 1, 2
ORDER BY 1, 2
)
UNION ALL
(
(
SELECT
    state,
    gender,
    COUNT(*)
FROM
    customers
GROUP BY 1, 2
ORDER BY 1, 2
)
)
ORDER BY 1, 2
```

Ta kwerenda zwróci wyniki widoczne na rysunku 3.11.

state text	gender text	count bigint
AK	F	101
AK	M	87
AK	[null]	188
AL	F	433
AL	M	489
AL	[null]	922
AR	F	112
AR	M	120
AR	[null]	232

Rysunek 3.11. Dane wyjściowe kwerendy zwracającej liczbę klientów z podziałem na stany i płeć; dane są uporządkowane alfabetycznie

Jednak używanie `UNION ALL` jest żmudne i może wymagać pisania bardzo długich kwerend. Inna technika to zastosowanie klauzuli `GROUPING SETS`. Pozwala ona utworzyć kilka kategorii, na podstawie których grupowane są wartości — podobnie jak opisana wcześniej instrukcja `UNION ALL`. Za pomocą słowa kluczowego `GROUPING SETS` możesz zmodyfikować poprzednią kwerendę z instrukcją `UNION ALL` w następujący sposób:

```
SELECT
    state,
    gender,
    COUNT(*)
FROM
    customers
GROUP BY GROUPING SETS (
    (state),
    (gender),
    (state, gender)
)
ORDER BY 1, 2
```

Ten kod zwraca te same dane wyjściowe co poprzednia kwerenda z instrukcją `UNION ALL`. W następnym punkcie zobaczysz, jak działają funkcje agregujące dla zbiorów uporządkowanych.

Funkcje agregujące dla zbiorów uporządkowanych

Do tego miejsca żadna z omawianych funkcji agregujących nie była zależna od kolejności danych. Dane można uporządkować za pomocą klauzuli `ORDER BY`, przy czym opisane funkcje jej nie wymagają. Istnieją jednak zagregowane statystyki, których obliczanie jest zależne od kolejności wartości. Na przykład wyznaczenie mediany kolumny wymaga uporządkowania danych. Na potrzeby takich sytuacji SQL udostępnia zestaw **funkcji agregujących dla zbiorów uporządkowanych**. W tabeli z rysunku 3.12 podane są podstawowe funkcje tego rodzaju.

Funkcja	Objaśnienie
<code>mode()</code>	Zwraca wartość, która występuje najczęściej. Jeśli takich wartości jest kilka, zwracana jest ta, która występuje najwcześniej.
<code>Percentile_cont(pozycja_jako_ułamek)</code>	Zwraca wartość z uporządkowanego zbioru danych o pozycji podanej jako ułamek. W razie potrzeby wartość jest interpolowana na podstawie sąsiednich danych wejściowych.
<code>Percentile_disc(pozycja_jako_ułamek)</code>	Zwraca pierwszą wartość danych wyjściowych, której pozycja w uporządkowanym zbiorze danych jest równa lub większa względem podanego ułamka.

Rysunek 3.12. Podstawowe funkcje agregujące dla zbiorów uporządkowanych

Oto składnia takich funkcji:

```
SELECT
    {funkcja_dla_zbioru_uporzadkowanego} WITHIN GROUP (ORDER BY
    {kolumna_porzadkujaca})
FROM {tabela};
```


Tu {funkcja_dla_zbioru_uporzadkowanego} to funkcja agregująca dla zbioru uporządkowanego, {kolumna_poradkujaca} to kolumna z uporządkowanymi danymi dla tej funkcji, a {tabela} to tabela, w której znajduje się ta kolumna.

Aby zrozumieć działanie takich funkcji, oblicz medianę cen z tabeli products. Możesz użyć następującej kwerendy:

```
SELECT
  PERCENTILE_CONT(0.5)
  WITHIN GROUP (ORDER BY base_msrp)
  AS mediana
FROM
  products;
```

Użycie wartości 0.5 wynika z tego, że mediana to odpowiednik percentyla 50%, czyli 0.5 w zapisie ułamkowym. To daje następujące wyniki:

```
mediana
double precision
-----
749.99
```

Funkcje agregujące dla zbiorów uporządkowanych to narzędzia do obliczania niemal dowolnych zagregowanych statystyk dla zbioru danych. W następnym podrozdziale zobaczysz, jak wykorzystać funkcje agregujące do sprawdzania jakości danych.

Klauzula HAVING

Umiesz już wykonywać rozmaite operacje agregujące z użyciem klauzuli GROUP BY. Jednak czasem niektóre wiersze nie są przydatne w zagregowanych wynikach i należy je usunąć z danych wyjściowych kwerendy. Na przykład gdy zliczasz klientów, istotne mogą być tylko miejsca z przynajmniej 1000 klientów. Pierwsza próba może wyglądać tak:

```
SELECT
  state, COUNT(*)
FROM
  customers
WHERE
  COUNT(*) >= 1000
GROUP BY
  state
ORDER BY
  state;
```

Okazuje się jednak, że ta kwerenda nie zadziała — zwróci błąd widoczny na rysunku 3.13.

```
ERROR: BŁĄD:  funkcje agregujące są niedopuszczalne w WHERE
LINE 6:  COUNT(*)>=1000
          ^
```

```
SQL state: 42803
```

```
Character: 51
```

Rysunek 3.13. Błąd z informacją, że kwerenda nie działa

Do filtrowania wyników funkcji agregujących potrzebna jest nowa klauzula, **HAVING**. Przypomina ona klauzulę **WHERE**, ale jest specjalnie dostosowana do kwerend **GROUP BY**. Ogólna struktura operacji **GROUP BY** z klauzulą **HAVING** wygląda tak:

```
SELECT {KLUCZ}, {FUNK_AGREG(kolumna1)}
FROM {tabela1}
GROUP BY {KLUCZ}
HAVING {INNA_FUNK_AGREG(kolumna2)_WARUNEK}
```

W tej kwerendzie **{KLUCZ}** to kolumna lub funkcja służąca do tworzenia grup, **{FUNK_AGREG(kolumna1)}** to funkcja agregująca dla danej kolumny obliczana dla wszystkich wierszy z każdej grupy, **{tabela1}** to tabela (lub zestaw złączonych tabel), z której pochodzą wiersze dzielone na grupy, a **{INNA_FUNK_AGREG(kolumna2)_WARUNEK}** to warunek podobny jak w klauzuli **WHERE**, obejmujący funkcję agregującą. Teraz wykonaj ćwiczenie, używając klauzuli **HAVING**.

Ćwiczenie 3.03 — obliczanie wyników i wyświetlanie danych z użyciem klauzuli **HAVING**

W tym ćwiczeniu obliczysz wyniki i wyświetlisz dane z użyciem klauzuli **HAVING**. Menedżer ds. sprzedaży z ZoomZoom chce znać liczbę klientów w stanach, w których przynajmniej 1000 klientów kupiło jakiś produkt tej firmy. Pomóż menedżerowi pobrać te dane. Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Otwórz wybranego klienta SQL-a i nawiąż połączenie z bazą sql da.
2. Użyj klauzuli **HAVING**, aby zliczyć klientów z poszczególnych stanów; uwzględnij tylko stany, dla których liczba klientów wynosi przynajmniej 1000:

```
SELECT
    state, COUNT(*)
FROM
    customers
GROUP BY
    state
HAVING
    COUNT(*)>=1000
ORDER BY
    state;
```

Ta kwerenda zwraca dane wyjściowe widoczne na rysunku 3.14.

state text	count bigint
CA	5038
CO	1042
DC	1447
FL	3748
GA	1251
IL	1094
NC	1070
NY	2395
OH	1656

Rysunek 3.14. Liczba klientów ze stanów, dla których ta liczba wynosi przynajmniej 1000

Widać tu stany, w których liczba klientów firmy ZoomZoom wynosi przynajmniej 1000. W tej grupie najwięcej klientów, 5038, jest w stanie CA, a najmniej, 1042, w stanie CO.

Kod źródłowy z tego fragmentu jest dostępny na stronie <https://packt.live/3hmF7fq>.

W tym ćwiczeniu zastosowaliśmy klauzulę `HAVING` do wydajnego obliczania wyników i wyświetlania danych.

Stosowanie funkcji agregujących do oczyszczania danych i sprawdzania ich jakości

W rozdziale 1., „Wprowadzenie do SQL-a dla analityków”, wyjaśniliśmy, że SQL może służyć do oczyszczania danych. Choć metody wymienione w tamtym rozdziale świetnie sprawdzają się przy oczyszczaniu danych, funkcje agregujące zapewniają dodatkowe techniki, dzięki którym oczyszczanie danych jest jeszcze łatwiejsze i bardziej kompletne. W tym podrozdziale poznasz niektóre z tych technik.

Znajdowanie brakujących wartości za pomocą klauzuli `GROUP BY`

W rozdziale 1., „Wprowadzenie do SQL-a dla analityków”, wspomnieliśmy, że jednym z największych problemów z oczyszczaniem danych jest radzenie sobie z brakującymi wartościami. Choć wyjaśniliśmy, jak znajdować brakujące wartości i jak je eliminować, nie opisaliśmy,

jak ustalić ilość brakujących danych w zbiorze. Wynikało to stąd, że wówczas nie były Ci jeszcze znane narzędzia do podsumowywania informacji o zbiorze danych.

Za pomocą funkcji agregujących można ustalić nie tylko to, w których kolumnach brakuje wartości, ale też to, czy duża ilość brakujących danych nie powoduje, że kolumna nie nadaje się do użytku. W zależności od ilości brakujących danych trzeba określić, czy najbardziej sensowne jest usunięcie wierszy z lukami w danych, uzupełnienie braków czy usunięcie kolumny (jeśli ilość danych nie wystarcza do wyciągania sensownych wniosków).

Najłatwiejszym sposobem na stwierdzenie, czy w kolumnie brakuje wartości, jest użycie zmodyfikowanej instrukcji CASE WHEN z funkcjami SUM i COUNT. W ten sposób można określić procent brakujących danych. Potrzebna kwerenda wygląda tak:

```
SELECT
    SUM(CASE WHEN {kolumna1}
        IS NULL OR {kolumna1}
        IN ({brakujące_wartości})
        THEN 1
        ELSE 0 END)::FLOAT/COUNT(*)
FROM
    {tabela1}
```

Tu {kolumna1} to kolumna, w której chcesz sprawdzać brakujące wartości, {brakujące_wartości} to lista rozdzielonych przecinkami wartości uznawanych za brakujące, a {tabela1} to tabela lub podkwerenda z lukami w danych.

Na podstawie wyników tej kwerendy można zastosować różne strategie radzenia sobie z brakami. Jeśli procent brakujących danych jest bardzo niski (<1%), możesz rozważyć odfiltrowanie lub usunięcie niepełnych danych z analiz. Jeżeli brakuje więcej danych (<20%), możesz uzupełnić luki typową wartością, na przykład średnią lub wartością modalną, aby przeprowadzić dokładne analizy. Jednak gdy brakuje ponad 20% danych, konieczne może być usunięcie kolumny z analiz, ponieważ danych jest za mało, aby wyciągać wiarygodne wnioski na podstawie wartości z takiej kolumny.

Przyjrzyj się lukom w danych z tabeli customers, a dokładniej — z kolumny state. W tym celu podziel liczbę rekordów niemających wartości w kolumnie state przez łączną liczbę rekordów:

```
SELECT
    SUM(CASE WHEN state IS NULL OR state IN ('')
        THEN 1
        ELSE 0 END)::FLOAT/COUNT(*)
    AS missing_state
FROM
    customers;
```

Otrzymasz następujące dane wyjściowe:

```
missing_state
double precision
-----
0.10934
```

Widać tu, że brakuje prawie 11% danych o stanie. Na potrzeby analiz możesz przyjąć, że brakujący klienci pochodzą z Kalifornii (CA), ponieważ jest to stan najczęściej występujący w danych. Jednak dużo dokładniejsze byłoby znalezienie i uzupełnienie brakujących danych.

Pomiar jakości danych za pomocą funkcji agregujących

Jednym z ważnych motywów w analizie danych jest to, że jest ona przydatna tylko wtedy, jeśli w danych występuje duża zmienność. Kolumna, w której każda wartość jest identyczna, nie jest przydatna. Dlatego często warto ustalić, ile różnych wartości znajduje się w kolumnie. Do pomiaru liczby różnych wartości w kolumnie możesz użyć wyrażenia `COUNT DISTINCT`. Oto struktura kwerendy z takim wyrażeniem:

```
SELECT COUNT (DISTINCT {kolumna1})
FROM {tabela1}
```

Tu `{kolumna1}` to kolumna, w której należy zliczyć wartości, a `{tabela1}` to tabela zawierająca tę kolumnę.

Inne często wykonywane zadanie to sprawdzanie, czy każda wartość w kolumnie jest unikatowa. Choć w wielu sytuacjach można to zapewnić, dodając do kolumny ograniczenie `PRIMARY KEY`, nie zawsze jest to możliwe. Aby rozwiązać problem, możesz napisać następującą kwerendę:

```
SELECT COUNT (DISTINCT {kolumna1})=COUNT(*)
FROM {tabela1}
```

Tu `{kolumna1}` to kolumna, w której należy zliczyć wartości, a `{tabela1}` to tabela zawierająca tę kolumnę. Jeśli kwerenda zwraca wartość `True`, kolumna w każdym wierszu zawiera unikatową wartość. W przeciwnym razie przynajmniej jedna z wartości się powtarza. Jeżeli wartości powtarzają się w kolumnie, która powinna przechowywać unikatowe dane, może to wskazywać na problemy z procesami ETL lub używane jest złączenie skutkujące powtarzaniem się wierszy.

W ramach prostego przykładu sprawdź, czy kolumna `customer_id` w tabeli `customers` zawiera unikatowe wartości:

```
SELECT
  COUNT (DISTINCT customer_id)=COUNT(*)
  AS equal_ids
FROM
  customers;
```

Ta kwerenda zwraca następujące dane:

```
equal_ids
boolean
-----
true
```

Znasz już różne sposoby tworzenia kwerend z funkcjami agregującymi. W następnym zadaniu zastosujesz takie funkcje do danych sprzedażowych.

Zadanie 3.01 — analizowanie danych sprzedażowych z użyciem funkcji agregujących

W tym zadaniu przeanalizujesz dane z użyciem funkcji agregujących. Dyrektor generalny, dyrektor ds. operacyjnych i dyrektor finansowy firmy ZoomZoom chcą się dowiedzieć, co może wpływać na poziom sprzedaży; uważają, że dzięki Twojemu przybyciu zespół analityków jest wystarczająco kompetentny, aby podołać temu zadaniu. Zostało ono zlecone Tobie, a Twój szef delikatnie informuje Cię, że jest to najważniejszy projekt, nad jakim kiedykolwiek pracował firmowy zespół analityków. Oto kroki niezbędne do wykonania tego zadania:

1. Otwórz wybranego klienta SQL-a i nawiąż połączenie z bazą sql da.
2. Ustal łączną liczbę produktów sprzedanych przez firmę.
3. Dla każdego stanu ustal łączną kwotę transakcji w dolarach.
4. Znajdź pięć najlepszych salonów pod względem liczby sprzedanych produktów (pomiń sprzedaż internetową).
5. Oblicz średnią wartość sprzedaży każdego produktu w każdym kanale. Dane są zapisane w tabeli `sales`. Sprawdź średnią wartość sprzedaży dla wszystkich kombinacji kanału sprzedaży i produktu (kolumny `channel` i `product_id`).

Oczekiwane dane wyjściowe są widoczne na rysunku 3.15.

channel text	product_id bigint	avg_sales_amount double precision
dealership	3	477.253737607644
dealership	4	109822.274881517
dealership	5	664.330132075472
dealership	6	62563.3763837638
dealership	7	573.744146637002
dealership	8	668.850500463391
dealership	9	33402.6845637584
dealership	10	81270.1121794872
dealership	11	91589.7435897436

Rysunek 3.15. Średnia sprzedaż obliczona za pomocą klauzuli `GROUPING SETS` zastosowanej do kolumn `channel` i `product_id`

Rozwiązanie tego zadania znajdziesz w „Dodatku”.

Za pomocą funkcji agregujących wykryliśmy wzorce, które pomogą firmie zrozumieć, jak zwiększyć przychody i uzyskać lepsze wyniki.

Funkcje okna

Funkcje agregujące umożliwiają pobranie wielu wierszy i przekształcenie ich w jedną liczbę. Na przykład funkcja COUNT przyjmuje wiersze tabeli i zwraca ich liczbę. Jednak niekiedy celem jest otrzymanie i zachowanie zbioru wierszy. Załóżmy, że chcesz przypisać wszystkim klientom pozycję na podstawie tego, kiedy po raz pierwszy kupili coś od firmy. Najdawniejszy klient otrzymuje pozycję 1., drugi najstarszy klient pozycję 2. itd. Możesz pobrać wszystkich klientów, używając następującej kwerendy:

```
SELECT
  *
FROM
  customers
ORDER BY
  date_added;
```

Możesz uporządkować klientów od najstarszego do najnowszego na kilka sposobów. Dalej w rozdziale dowiesz się, jak za pomocą funkcji RANK przypisywać pozycje do uporządkowanych rekordów. Możesz też użyć funkcji agregującej przy pobieraniu dat i je uporządkować:

```
SELECT
  date_added, COUNT(*)
FROM
  customers
GROUP BY
  date_added
ORDER BY
  date_added
```

Dane wyjściowe tego kodu są pokazane na rysunku 3.16.

date_added timestamp without time zone	count bigint
2010-03-15 00:00:00	11
2010-03-16 00:00:00	13
2010-03-17 00:00:00	12
2010-03-18 00:00:00	19
2010-03-19 00:00:00	23
2010-03-20 00:00:00	16
2010-03-21 00:00:00	20
2010-03-22 00:00:00	14
2010-03-23 00:00:00	11
2010-03-24 00:00:00	21
2010-03-25 00:00:00	15

Rysunek 3.16. Zastosowanie funkcji agregującej do porządkowania dat i czasu

Choć ta kwerenda zwraca daty, pomija pozostałe kolumny i nie zapewnia informacji o pozycjach. W takiej sytuacji przydadzą się **funkcje okna**. Takie funkcje przyjmują i przetwarzają wiele wierszy danych, przy czym zachowują wszystkie informacje. Właśnie to jest potrzebne do wyświetlenia pozycji.

Aby lepiej zrozumieć tę technikę, w następnym punkcie zobaczysz, jak wygląda kwerenda z funkcją okna.

Podstawy funkcji okna

Oto podstawowa składnia funkcji okna:

```
SELECT {kolumny},
       {funkcja_okna} OVER (PARTITION BY {klucz_podziału} ORDER BY {klucz_porządkowania})
FROM tabela1;
```

Tu {kolumny} to kolumny do pobrania z tabel w tej kwerendzie, {funkcja_okna} to stosowana funkcja okna, {klucz_podziału} to kolumna lub kolumny wykorzystywane do podziału danych (więcej na ten temat dowiesz się dalej), {klucz_porządkowania} to kolumna lub kolumny służące do porządkowania danych, a tabela1 to tabela lub połączone tabele, z których kwerenda ma pobrać dane. Słowo kluczowe OVER wskazuje początek definicji okna.

Aby zrozumieć to podejście, przyjrzyj się przykładowi. Możliwe, że uważasz, iż nie znasz żadnych funkcji okna. Jednak jako funkcje okna możesz stosować dowolne funkcje agregujące. W tej kwerendzie będzie to funkcja COUNT(*):

```
SELECT
    customer_id,
    title,
    first_name,
    last_name,
    gender,
    COUNT(*) OVER () as total_customers
FROM
    customers
ORDER BY
    customer_id;
```

Dane wyjściowe tej kwerendy są pokazane na rysunku 3.17.

Na rysunku widać, że ta kwerenda dotycząca tabeli customers zwróciła kolumny title, first_name i last_name, tak jak typowa kwerenda SELECT. Jednak teraz pojawiła się też nowa kolumna, total_customers. Zawiera ona liczbę użytkowników, jaką zwróciłaby następująca kwerenda:

```
SELECT COUNT(*)
FROM customers;
```

Ta kwerenda zwraca wartość 50000. Wcześniej wspomnieliśmy, że poprzednia kwerenda zwraca wszystkie wiersze i wartość COUNT(*) zamiast samej wartości COUNT zwracanej przez zwykłą funkcję agregującą.

customer_id bigint	title text	first_name text	last_name text	gender text	total_customers bigint
1	[null]	Arlena	Riveles	F	50000
2	Dr	Ode	Stovin	M	50000
3	[null]	Braden	Jordan	M	50000
4	[null]	Jessika	Nussen	F	50000
5	[null]	Lonnie	Rembaud	F	50000
6	[null]	Cortie	Locksley	M	50000
7	[null]	Wood	Kennham	M	50000
8	[null]	Rutger	Humblestone	M	50000
9	[null]	Melantha	Tibb	F	50000
10	Ms	Barbara-anne	Gowlett	F	50000
11	Mrs	Urbano	Middlehurst	M	50000

Rysunek 3.17. Klienci wyświetleni za pomocą kwerendy z funkcją okna COUNT(*)

Teraz przyjrzyj się innym parametrom kwerendy. Co się stanie, jeśli użyjesz słowa kluczowego PARTITION BY, tak jak w poniższej kwerendzie?

```
SELECT
  customer_id, title, first_name, last_name, gender,
  COUNT(*) OVER (PARTITION BY gender) as total_customers
FROM
  customers
ORDER BY
  customer_id;
```

Dane wyjściowe tego kodu są pokazane na rysunku 3.18.

customer_id bigint	title text	first_name text	last_name text	gender text	total_customers bigint
1	[null]	Arlena	Riveles	F	25044
2	Dr	Ode	Stovin	M	24956
3	[null]	Braden	Jordan	M	24956
4	[null]	Jessika	Nussen	F	25044
5	[null]	Lonnie	Rembaud	F	25044
6	[null]	Cortie	Locksley	M	24956
7	[null]	Wood	Kennham	M	24956
8	[null]	Rutger	Humblestone	M	24956
9	[null]	Melantha	Tibb	F	25044
10	Ms	Barbara-anne	Gowlett	F	25044
11	Mrs	Urbano	Middlehurst	M	24956

Rysunek 3.18. Kwerenda z funkcją okna zwracająca listę klientów z wartością COUNT(*) z podziałem na płeć

Tu widać, że w kolumnie `total_customers` jako liczba elementów podawane są teraz dwie wartości — 24956 lub 25044. Są to liczby klientów poszczególnych płci. Można się o tym przekonać za pomocą następującej kwerendy:

```
SELECT
  gender, COUNT(*)
FROM
  customers
GROUP BY
  1
```

Dla kobiet wyświetlana wartość równa się liczbie kobiet, a dla mężczyzn — liczbie mężczyzn. Co się stanie, jeśli użyjesz klauzuli `ORDER BY` w nawiasie po słowie kluczowym `OVER`, tak jak w poniższej kwerendzie?

```
SELECT
  customer_id, title,
  first_name, last_name, gender,
  COUNT(*) OVER (ORDER BY customer_id) as total_customers
FROM
  customers
ORDER BY
  customer_id;
```

Rysunek 3.19 przedstawia dane wyjściowe tego kodu.

customer_id bigint	title text	first_name text	last_name text	gender text	total_customers bigint
1	[null]	Arlena	Riveles	F	1
2	Dr	Ode	Stovin	M	2
3	[null]	Braden	Jordan	M	3
4	[null]	Jessika	Nussen	F	4
5	[null]	Lonnie	Rembaud	F	5
6	[null]	Cortie	Locksley	M	6
7	[null]	Wood	Kennham	M	7
8	[null]	Rutger	Humblestone	M	8
9	[null]	Melantha	Tibb	F	9
10	Ms	Barbara-anne	Gowlett	F	10
11	Mrs	Urbano	Middlehurst	M	11

Rysunek 3.19. Kwerenda z funkcją okna; wyświetla listę klientów z wykorzystaniem funkcji `COUNT(*)` i porządkuje ich według pola `customer_id`

Widać tu coś na kształt sumy narastającej, która określa łączną liczbę klientów. Co się dzieje w tej kwerendzie? Ilustruje ona, skąd pochodzi człon „okna” w nazwie „funkcje okna”. Gdy używasz takich funkcji, kwerenda tworzy „okno” obejmujące wycinek tabeli, dla którego wykonywane są obliczenia. Słowo kluczowe `PARTITION BY` działa podobnie jak `GROUP BY` i dzieli zbiór danych na kilka grup. Dla każdej grupy tworzone jest okno. Jeśli pominiesz klauzulę `ORDER BY`, okno obejmuje całą grupę.

Jeżeli jednak użyjesz klauzuli `ORDER BY`, wiersze w grupie są porządkowane zgodnie z nią i dla każdego wiersza tworzone jest okno, do którego stosowana jest funkcja. Gdy okno nie jest określone, domyślnie tworzone jest okno obejmujące wszystkie wiersze od pierwszego (ustalonego zgodnie z klauzulą `ORDER BY`) do bieżącego wiersza przetwarzanego przez funkcję, co ilustruje rysunek 3.20. Funkcja jest stosowana do tego właśnie okna.

customer_id bigint	title text	first_name text	last_name text	gender text	total_customers bigint
1	[null]	Arlena	Riveles	F	1
2	Dr	Ode	Stovin	M	2
3	[null]	Braden	Jordan	M	3

Okno dla pierwszego wiersza
Okno dla drugiego wiersza
Okno dla trzeciego wiersza

Rysunek 3.20. Okna obejmujące klientów w kwerendzie z funkcją `COUNT(*)` i wierszami porządkowanymi według kolumny `customer_id`

Na rysunku okno dla pierwszego wiersza zawiera jeden wiersz i zwracana liczba elementów to 1. Dla drugiego wiersza okno obejmuje dwa wiersze i zwracana liczba elementów jest równa 2. Dla trzeciego wiersza okno obejmuje trzy wiersze, dlatego w kolumnie `total_customers` zwracana jest liczba 3.

Co się stanie po połączeniu `PARTITION BY` i `ORDER BY`? Przyjrzyj się tej kwerendzie:

```
SELECT customer_id, title, first_name, last_name, gender,
COUNT(*) OVER (PARTITION BY gender ORDER BY customer_id)
as total_customers
FROM customers
ORDER BY customer_id;
```

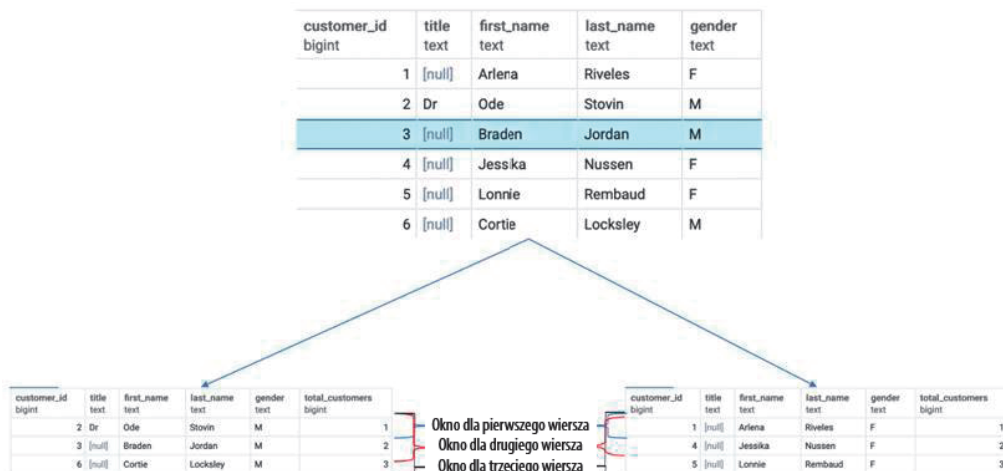
Gdy wykonasz tę kwerendę, otrzymasz wyniki widoczne na rysunku 3.21.

customer_id bigint	title text	first_name text	last_name text	gender text	total_customers bigint
1	[null]	Arlena	Riveles	F	1
2	Dr	Ode	Stovin	M	1
3	[null]	Braden	Jordan	M	2
4	[null]	Jessika	Nussen	F	2
5	[null]	Lonnie	Rembaud	F	3
6	[null]	Cortie	Locksley	M	3
7	[null]	Wood	Kennham	M	4
8	[null]	Rutger	Humblestone	M	5
9	[null]	Melantha	Tibb	F	4
10	Ms	Barbara-anne	Gowlett	F	5
11	Mrs	Urbano	Middlehurst	M	6

Rysunek 3.21. Kwerenda z funkcją okna wyświetlająca listę klientów z użyciem funkcji `COUNT(*)` i z podziałem na płcie; dane są uporządkowane według kolumny `customer_id`

Podobnie jak w poprzedniej kwerendzie otrzymaliśmy coś w rodzaju rankingu. Wygląda jednak na to, że wartości różnią się w zależności od płci. Jak działa ta kwerenda? Gdy wykonywana była jedna z poprzednich kwerend, wspomnieliśmy, że kod najpierw dzieli tabelę na dwa podzbiory zgodnie z klauzulą `PARTITION BY`, a potem każda grupa jest używana jako osobny zbiór zliczanych elementów, z własnymi oknami.

Ten proces jest pokazany na rysunku 3.22. W ten sposób otrzymywane są widoczne liczby. Możliwości funkcji okna wynikają z zastosowania trzech słów kluczowych: `OVER()`, `PARTITION BY` i `ORDER BY`.



Rysunek 3.22. Okna na liście klientów w kwerendzie z funkcjami okna z funkcją `COUNT(*)` i z podziałem na płcie; dane są uporządkowane według kolumny `customer_id`

Teraz, gdy wiesz już, jak działają funkcje okna, zastosujesz je w następnym ćwiczeniu.

Ćwiczenie 3.04 — analizowanie zmian współczynnika podawania danych przez klientów w czasie

W tym ćwiczeniu zastosujesz funkcje okna do zbioru danych i przeanalizujesz dane. Przez ostatnich sześć miesięcy firma ZoomZoom eksperymentowała z różnymi mechanizmami, aby zachęcić użytkowników do uzupełniania wszystkich pól — a przede wszystkim pól z adresem — w formularzu na dane klienta. W celu przeanalizowania danych firma chce obliczać sumę narastającą, która określa, ilu użytkowników podało swój adres do danego czasu. Napisz kwerendę, aby uzyskać potrzebne wyniki.

We wszystkich ćwiczeniach z tego rozdziału używane jest narzędzie pgAdmin 4.

Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Otwórz wybranego klienta SQL-a i nawiąż połączenie z bazą sql da.
2. Użyj funkcji okna i napisz kwerendę, która zwraca informacje o klientach oraz liczbę osób, które podały swój adres. Ponadto uporządkuj listę według dat. Kwerenda powinna wyglądać tak:

```
SELECT
    customer_id, street_address, date_added::DATE,
    COUNT(CASE WHEN street_address IS NOT NULL
    THEN customer_id
    ELSE NULL END)
    OVER (ORDER BY date_added::DATE) as total_customers_filled_street
FROM
    customers
ORDER BY
    date_added;
```

Powinieneś otrzymać wyniki widoczne na rysunku 3.23.

customer_id bigint	street_address text	date_added date	total_customers_filled_street bigint
2625	0353 Iowa Road	2010-03-15	10
17099	130 Marcy Crossing	2010-03-15	10
18685	86 Michigan Junction	2010-03-15	10
35683	1 Cordelia Crossing	2010-03-15	10
6173	79865 Hagan Terrace	2010-03-15	10
12484	[null]	2010-03-15	10
13390	38463 Forest Dale Way	2010-03-15	10
7486	61 Village Crossing	2010-03-15	10
30046	13961 Steensland Trail	2010-03-15	10
30555	294 Quincy Hill	2010-03-15	10
48307	8487 Warbler Plaza	2010-03-15	10
48229	943 Cody Trail	2010-03-16	22
42776	6010 Carey Drive	2010-03-16	22
46277	5799 Thackeray Crossing	2010-03-16	22
34189	0 Park Meadow Street	2010-03-16	22
8571	39223 Lunder Street	2010-03-16	22
17626	086 East Hill	2010-03-16	22
17832	62 Delladonna Road	2010-03-16	22

Rysunek 3.23. Kwerenda z funkcją okna zwracająca adresy; dane są uporządkowane według kolumny date_added

Teraz wszyscy klienci są uporządkowani według daty rejestracji i można sprawdzić, jak liczba osób, które podały adres, zmienia się z czasem.

Kod źródłowy z tego fragmentu znajdziesz na stronie <https://packt.live/30xeLBm>.

W tym ćwiczeniu pokazaliśmy, jak korzystać z funkcji okna do analizowania danych. W następnym punkcie zobaczysz, jak używać w kwerendach słowa kluczowego WINDOW.

Słowo kluczowe WINDOW

Rozumiesz już podstawy działania funkcji okna. Pora zapoznać się ze składnią, która ułatwia stosowanie takich funkcji. W niektórych kwerendach przydatne jest obliczanie różnych funkcji dla tego samego okna. Możliwe, że chcesz obliczyć sumę narastającą liczby klientów oraz sumę narastającą liczby klientów z tytułem dla każdej płci. Oto potrzebna kwerenda:

```
SELECT customer_id, title, first_name, last_name, gender,
COUNT(*) OVER (PARTITION BY gender ORDER BY customer_id) as total_customers,
SUM(CASE WHEN title IS NOT NULL THEN 1 ELSE 0 END)
OVER (PARTITION BY gender ORDER BY customer_id) as total_customers_title
FROM customers
ORDER BY customer_id;
```

Dane wyjściowe tego kodu są pokazane na rysunku 3.24.

customer_id bigint	title text	first_name text	last_name text	gender text	total_customers bigint	total_customers_title bigint
1	[null]	Arlena	Riveles	F	1	0
2	Dr	Ode	Stovin	M	1	1
3	[null]	Braden	Jordan	M	2	1
4	[null]	Jessika	Nussen	F	2	0
5	[null]	Lonnie	Rembaud	F	3	0
6	[null]	Cortie	Locksley	M	3	1
7	[null]	Wood	Kennham	M	4	1
8	[null]	Rutger	Humblestone	M	5	1
9	[null]	Melantha	Tibb	F	4	0
10	Ms	Barbara-anne	Gowlett	F	5	1
11	Mrs	Urbano	Middlehurst	M	6	2
12	Mr	Tyne	Duggan	F	6	2
13	[null]	Gannon	Braker	M	7	2
14	[null]	Derry	Lyburn	M	8	2
15	[null]	Nichols	Espinay	M	9	2

Rysunek 3.24. Kwerenda z funkcją okna zwracająca sumę narastającą liczby klientów ogółem oraz liczby klientów z tytułem z podziałem na płeć

Choć ta kwerenda zwraca pożądany wynik, jej zapisywanie może być żmudne — zwłaszcza odpowiednika klauzuli WINDOW. Na szczęście możesz uprościć kod, używając innej klauzuli WINDOW, która upraszcza tworzenie aliasów okien.

Poprzednią kwerendę możesz uprościć, stosując następujący zapis:

```
SELECT
  customer_id, title,
  first_name, last_name, gender,
  COUNT(*) OVER w as total_customers,
  SUM(CASE WHEN title IS NOT NULL THEN 1 ELSE 0 END)
  OVER w as total_customers_title
FROM
  customers
WINDOW w AS (PARTITION BY gender ORDER BY customer_id)
ORDER BY customer_id;
```

Ta kwerenda powinna dać ten sam wynik co na rysunku 3.24. Jednak nie musieliśmy pisać długich wyrażeń `PARTITION BY` i `ORDER BY` dla każdej funkcji okna. Wystarczyło utworzyć alias dla zdefiniowanego okna — `WINDOW w`.

Obliczanie statystyk z użyciem funkcji okna

Wiesz już, jak działają funkcje okna. Możesz więc zacząć używać ich do obliczania przydatnych statystyk, na przykład pozycji, percentyli i statystyk dla danych narastających.

W tabeli z rysunku 3.25 znajdziesz podsumowanie różnych przydatnych funkcji statystycznych. Należy jeszcze raz podkreślić, że jako funkcje okna można stosować wszystkie funkcje agregujące (`AVG`, `SUM`, `COUNT` itd.).

Funkcja	Objaśnienie
<code>ROW_NUMBER()</code>	Określa numer bieżącego wiersza w grupie.
<code>RANK()</code>	Zwraca pozycję wiersza w grupie zgodnie z klauzulą <code>ORDER BY</code> ; tworzy luki, jeśli kilka wierszy zajmuje tę samą pozycję (na przykład jeśli wiersze pierwszy i drugi zajmują pozycję pierwszą, wiersz trzeci będzie zajmował pozycję trzecią).
<code>DENSE_RANK()</code>	Zwraca pozycję wiersza w grupie zgodnie z klauzulą <code>ORDER BY</code> ; nie tworzy luk, jeśli kilka wierszy zajmuje tę samą pozycję (na przykład jeśli wiersze pierwszy i drugi zajmują pozycję pierwszą, wiersz trzeci będzie zajmował pozycję drugą).
<code>NTILE(liczba_przedziałów)</code>	Przypisuje do wierszy numery <code>n</code> przedziałów na podstawie klauzuli <code>ORDER BY</code> ; wartość <code>n</code> jest podawana za pomocą liczby całkowitej <code>liczba_przedziałów</code> .
<code>LAG(kolumna1, przesunięcie)</code>	Zwraca wartość kolumny <code>kolumna1</code> z wiersza poprzedzającego bieżący o przesunięcie zgodnie z uporządkowaniem na podstawie klauzuli <code>ORDER BY</code> .
<code>LEAD(kolumna1, przesunięcie)</code>	Zwraca wartość kolumny <code>kolumna1</code> z wiersza następującego po bieżącym o przesunięcie zgodnie z uporządkowaniem na podstawie klauzuli <code>ORDER BY</code> .

Rysunek 3.25. Statystyczne funkcje okna

Zwykle po wywołaniu dowolnej z tych funkcji w instrukcji SQL-owej znajduje się słowo kluczowe `OVER`. Po tym słowie znajduje się nawias z instrukcjami `PARTITION BY` i `ORDER BY`. Obie te instrukcje mogą być opcjonalne (zależy to od używanej funkcji).

Na przykład funkcja `ROW_NUMBER()` jest stosowana tak:

```
ROW_NUMBER() OVER(
    PARTITION BY kolumna_1, kolumna_2
    ORDER BY kolumna_3, kolumna_4
)
```

W następnym ćwiczeniu zobaczysz, jak się posługiwać wymienionymi funkcjami statystycznymi.

Ćwiczenie 3.05

— określanie pozycji na podstawie daty zatrudnienia

W tym ćwiczeniu zobaczysz, jak wykorzystać statystyczne funkcje okna do zrozumienia zbioru danych. Firma ZoomZoom chce awansować sprzedawcę z regionalnego salonu na menedżera. Przy podejmowaniu decyzji ważny jest staż pracy. Twoje zadanie polega na napisaniu kwerendy, która porządkuje sprzedawców z poszczególnych salonów zgodnie z datą zatrudnienia. Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Otwórz wybranego klienta SQL-a i nawiąż połączenie z bazą `sql` da.
2. Określ pozycję każdego sprzedawcy. Pozycję pierwszą przypisz pierwszej zatrudnionej osobie, drugą drugiemu zatrudnionemu pracownikowi itd. Użyj funkcji `RANK()`:

```
SELECT *,
    RANK() OVER (
        PARTITION BY dealership_id ORDER BY hire_date
    )
FROM
    salespeople
WHERE
    termination_date IS NULL;
```

Rysunek 3.26 przedstawia dane wyjściowe tego kodu.

Widać tu wszystkich sprzedawców z danymi na ich temat oraz pozycją w kolumnie `rank` określoną na podstawie daty zatrudnienia w salonie.

Kod źródłowy z tego fragmentu książki jest dostępny na stronie <https://packt.live/2B5OyyX>.

W tym ćwiczeniu skorzystaliśmy z funkcji `RANK()` do uporządkowania danych w zbiorze w określony sposób. W następnym punkcie nauczysz się używać ramek okna.

Stosowanie funkcji `DENSE_RANK()` jest również łatwe jak funkcji `RANK()`.

salesperson_id bigint	dealership_id bigint	title text	first_name text	last_name text	suffix text	username text	gender text	hire_date timestamp without time zone	termination_date timestamp without time zone	rank bigint
65	1	[null]	Dukie	Oxteby	[null]	doxteby1s	Male	2015-01-24 00:00:00	[null]	1
74	1	[null]	Marcos	Spong	[null]	mspong21	Male	2015-03-18 00:00:00	[null]	2
60	1	[null]	Eveleen	Mace	[null]	emace1n	Female	2015-07-15 00:00:00	[null]	3
87	1	[null]	Quent	Wogden	[null]	qwogden2e	Male	2015-08-17 00:00:00	[null]	4
98	1	[null]	Englebert	Loraine	[null]	elorraine2p	Male	2016-01-23 00:00:00	[null]	5
31	1	[null]	Lelia	Sheriff	[null]	lsheriffu	Female	2016-06-18 00:00:00	[null]	6
168	1	[null]	Sheff	McCoughan	[null]	smccougha...	Male	2016-07-22 00:00:00	[null]	7
49	1	[null]	Nadia	Rennick	[null]	nrerrick1c	Female	2016-07-24 00:00:00	[null]	8
10	1	[null]	Jereme	Onele	[null]	jonele9	Male	2016-08-15 00:00:00	[null]	9
7	1	[null]	Granville	Fidell	[null]	gfidell6	Male	2017-06-17 00:00:00	[null]	10
155	1	[null]	Ira	Meere	[null]	imeere4a	Male	2017-09-11 00:00:00	[null]	11
297	1	[null]	Shay	Nafziger	Sr	snafziger88	Male	2017-12-03 00:00:00	[null]	12
183	1	[null]	Eleen	McAndie	[null]	emcandie52	Female	2018-07-08 00:00:00	[null]	13
170	1	[null]	Giselbert	Schule	[null]	gschule4p	Male	2018-08-01 00:00:00	[null]	14
162	1	[null]	Cristine	Gibbens	[null]	cgibbens4h	Female	2018-10-07 00:00:00	[null]	15
258	1	[null]	Dorie	Dosedale	[null]	ddosedale75	Male	2018-10-15 00:00:00	[null]	16
92	1	Rev	Sandye	Duny	[null]	sduny2	Female	2019-01-03 00:00:00	[null]	17
39	1	[null]	Massimiliano	McSpiron	[null]	mmcspiron...	Male	2019-02-12 00:00:00	[null]	18

Rysunek 3.26. Sprzedawcy uporządkowani według stażu pracy

Ramka okna

Omawiając podstawy dotyczące funkcji okna, wspomnieliśmy, że domyślnie okno jest ustawiane w taki sposób, aby obejmowało wszystkie wiersze od pierwszego do bieżącego wiersza grupy (rysunek 3.20). Jednak to domyślne ustawienie można zmienić za pomocą klauzuli **ramki okna**. Kwerenda z funkcją okna i klauzulą ramki okna ma następującą postać:

```
SELECT {kolumny},
       {funkcja_okna} OVER (PARTITION BY {klucz_podziału}
                           ORDER BY {klucz_porządkowania} {RANGE_lub_ROWS}
                           BETWEEN {początek_ramki} AND {koniec_ramki})
FROM {tabela1};
```

W tej kwerendzie {kolumny} to kolumny pobierane z tabel w kwerendzie, {funkcja_okna} to stosowana funkcja okna, {klucz_podziału} to kolumna lub kolumny używane do podziału (więcej na ten temat dowiesz się dalej), {klucz_porządkowania} to kolumna lub kolumny służące do porządkowania danych, {RANGE_lub_ROWS} to słowo kluczowe RANGE lub ROWS, {początek_ramki} to słowo kluczowe określające początek ramki okna, {koniec_ramki} to słowo kluczowe określające koniec ramki okna, a {tabela1} to tabela lub złączone tabele, z których pobierane będą dane.

Warto zwrócić uwagę na różnicę między słowami RANGE i ROWS w klauzuli ramki. Słowo ROWS dotyczy wierszy i powoduje, że do obliczania wartości wykorzystywane są wiersze znajdujące się przed bieżącym wierszem i po nim. Słowo RANGE działa inaczej, gdy przynajmniej dwa wiersze mają te same wartości na podstawie klauzuli ORDER BY stosowanej dla danego okna. Jeśli bieżący wiersz używany w obliczeniach dla okna ma zgodnie z klauzulą ORDER BY tę samą wartość co inne wiersze, wszystkie te wiersze są dodawane do ramki okna.

Inna kwestia warta uwagi to wartości przyjmowane przez {początek_ramki} i {koniec_ramki}. Te elementy mogą mieć jedną z następujących wartości:

- UNBOUNDED PRECEDING — to słowo kluczowe użyte jako {początek_ramki} oznacza pierwszy rekord grupy, a zastosowane jako {koniec_ramki} oznacza ostatni rekord grupy.
- {liczba} PRECEDING — {liczba} (liczba całkowita) wierszy lub przedziałów przed bieżącym wierszem.
- CURRENT ROW — bieżący wiersz.
- {liczba} FOLLOWING — {liczba} (liczba całkowita) wierszy lub przedziałów po bieżącym wierszu.

Dla określonego okna można obliczyć wiele przydatnych statystyk. Jedną z nich jest **średnia krocząca**. Jest to średnia dla danej statystyki z określonego okna czasowego. Załóżmy, że chcesz obliczyć siedmiodniową średnią krocząca sprzedaży w firmie ZoomZoom. Te obliczenia można wykonać za pomocą następującej kwerendy:

```
WITH
  daily_sales as (
    SELECT sales_transaction_date::DATE,
           SUM(sales_amount) as total_sales
    FROM sales
    GROUP BY 1
  ),
moving_average_calculation_7 AS (
  SELECT
    sales_transaction_date, total_sales,
    AVG(total_sales)
    OVER (ORDER BY
           sales_transaction_date
           ROWS BETWEEN 7 PRECEDING
           and CURRENT ROW) AS sales_moving_average_7,
    ROW_NUMBER() OVER (ORDER BY sales_transaction_date) as row_number
  FROM
    daily_sales
  ORDER BY 1)
SELECT
  sales_transaction_date,
  CASE WHEN row_number>=7
  THEN sales_moving_average_7
  ELSE NULL END
  AS sales_moving_average_7
FROM
  moving_average_calculation_7;
```

Rysunek 3.27 przedstawia dane wyjściowe tego kodu.

W pierwszych siedmiu wierszach średnia to null, ponieważ siedmiodniowa średnia krocząca jest zdefiniowana tylko wtedy, jeśli dostępne są informacje z siedmiu dni, a w obliczeniach dla pierwszych wierszy ta liczba jest mniejsza.

sales_transaction_date date	sales_moving_average_7 double precision
2010-03-10	[null]
2010-03-12	[null]
2010-03-15	[null]
2010-03-17	[null]
2010-03-18	[null]
2010-03-19	[null]
2010-03-21	394.275857142857
2010-03-23	394.990125
2010-03-24	399.99
2010-03-25	399.99
2010-03-29	449.98875
2010-04-01	544.986375
2010-04-02	594.985125
2010-04-03	594.985125
2010-04-04	589.98525
2010-04-05	589.98525
2010-04-06	639.984
2010-04-07	689.98275

Rysunek 3.27. Siedmiodniowa średnia krocząca sprzedaży

W następnym ćwiczeniu zobaczysz, jak wykorzystać ruchome okno do obliczenia statystyk dotyczących uporządkowanych danych.

Ćwiczenie 3.06 — motywowanie pracowników lunchem

W tym ćwiczeniu użyjesz ramki okna do znalezienia ważnych informacji w danych. Aby pomóc zwiększyć przychody, zespół sprzedaży zdecydował, aby kupować wszystkim sprzedawcom w firmie lunch za każdym razem, gdy przekroczą najwyższe dzienne przychody z ostatnich 30 dni. Napisz kwerendę, która zwraca łączną sprzedaż w dolarach z danego dnia i poziom, jaki sprzedawcy muszą przekroczyć w danym dniu. Zaczynij od 1 stycznia 2019 roku. Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Otwórz wybranego klienta SQL-a i nawiąż połączenia z bazą sql da.
2. Oblicz łączną sprzedaż z danego dnia i docelowy poziom. Użyj do tego następującej kwerendy:

```
WITH daily_sales as (
  SELECT sales_transaction_date::DATE,
         SUM(sales_amount) as total_sales
  FROM sales
  GROUP BY 1
```

```

),

sales_stats_30 AS (
SELECT
    sales_transaction_date, total_sales,
    MAX(total_sales)
    OVER (ORDER BY sales_transaction_date ROWS BETWEEN 30 PRECEDING and
          1 PRECEDING)
    AS max_sales_30
FROM
    daily_sales
ORDER BY 1)

SELECT
    sales_transaction_date, total_sales,
    max_sales_30
FROM
    sales_stats_30
WHERE
    sales_transaction_date>='2019-01-01';

```

Powinieneś otrzymać wyniki widoczne na rysunku 3.28.

sales_transaction_date date	total_sales double precision	max_sales_30 double precision
2019-01-01	87694.844	316464.847
2019-01-02	76149.854	316464.847
2019-01-03	161269.809	316464.847
2019-01-04	193209.912	316464.847
2019-01-05	49469.77	316464.847
2019-01-06	96319.835	316464.847
2019-01-07	42239.837	316464.847
2019-01-08	101729.748	316464.847
2019-01-09	118634.902	316464.847
2019-01-10	100089.78	316464.847
2019-01-11	183209.871	283849.84

Rysunek 3.28. Najwyższa sprzedaż z ostatnich 30 dni

Zwróć uwagę na ramkę okna od 30 PRECEDING do 1 PRECEDING, która pozwala usunąć bieżący wiersz z obliczeń.

Kod źródłowy z tego fragmentu znajdziesz na stronie <https://packt.live/3cWKbDC>.

Widać tu, że dzięki ramkom okna obliczanie statystyk kroczących jest proste, a nawet zabawne.

Na zakończenie rozdziału wykonasz zadanie, które pozwoli Ci sprawdzić umiejętności korzystania z funkcji okna.

Zadanie 3.02 — analizowanie sprzedaży z wykorzystaniem ramek okna i funkcji okna

W tym zadaniu użyjesz w różny sposób funkcji okna i ramek okna, aby zrozumieć dane sprzedażowe. Zbliżają się święta i pora przyznać gwiazdkowe premie pracownikom firmy ZoomZoom. Zespół sprzedażowy chce sprawdzić, jak poradziła sobie cała firma i jakie wyniki miały poszczególne salony. W tym celu dyrektor działu sprzedaży poprosił Cię o przeprowadzenie analiz. Oto kroki niezbędne do wykonania tego zadania:

1. Otwórz wybranego klienta SQL-a i nawiąż połączenie z bazą sql da.
2. Oblicz łączny poziom dziennej sprzedaży dla wszystkich dni w 2018 roku (sprzed dnia 1 stycznia 2019 roku).
3. Oblicz 30-dniową średnią kroczącą dziennej liczby transakcji.
4. Na podstawie łącznego poziomu sprzedaży oblicz decyl dla każdego salonu.

Rysunek 3.29 przedstawia oczekiwane dane wyjściowe.

dealership_id double precision	total_sales_amount double precision	ntile integer
13	538079.414	1
9	618263.995	1
8	671619.251	2
4	905158.609	2
17	907058.842	3
20	949849.053	3
12	1086033.376	4
15	1197118.234	4
6	1316253.465	5
14	1551108.481	5
3	1622872.801	6
16	1981062.341	6

Rysunek 3.29. Decyle obliczone na podstawie wartości sprzedaży w salonach

Rozwiązanie tego zadania znajdziesz w „Dodatku”.

Podsumowanie

W tym rozdziale przedstawiliśmy niezwykle możliwości funkcji agregujących. Omówiliśmy tu kilka najczęściej stosowanych funkcji tego typu i pokazaliśmy, jak z nich korzystać. Użyliśmy klauzuli `GROUP BY` do podziału zbiorów danych na grupy i obliczania statystyk zbiorczych dla każdej z nich. Dalej pokazaliśmy, jak używać klauzuli `HAVING` do dodatkowego filtrowania kwerend. Funkcje agregujące pomogły Ci też oczyszczać dane i analizować ich jakość.

Ponadto omówiliśmy funkcje okna. Pokazaliśmy, jak tworzyć podstawowe funkcje okna z użyciem słów kluczowych `OVER`, `PARTITION BY` i `ORDER BY`. Dalej dowiedziałeś się, jak obliczać statystyki z zastosowaniem funkcji okna, a także jak przesuwac ramkę okna, by obliczać statystyki kroczące.

W następnym rozdziale zobaczysz, jak importować i eksportować dane, aby wykorzystać SQL razem z innymi programami. Użyjesz polecenia `COPY` do masowego wczytywania danych do bazy. Ponadto posłużysz się Excelem do przetwarzania danych z bazy i uprościsz kod za pomocą narzędzia `SQLAlchemy`.

Importowanie i eksportowanie danych

W tym rozdziale poznasz sposoby przenoszenia danych między bazą a narzędziami analitycznymi. Pierwszym narzędziem, któremu się przyjrysz, jest uruchamianie w wierszu poleceń `psql`. Umożliwia ono szybkie pobieranie danych z bazy. Za pomocą `psql` możesz też wykorzystać polecenie `COPY`, służące do wydajnego importowania i eksportowania danych. Stosując te proste narzędzia, możesz komunikować się z bazą oraz skutecznie przenosić dane tam i z powrotem. Dzięki lekturze tego rozdziału nauczysz się importować i eksportować dane. Będziesz przetwarzać i analizować je za pomocą Excela, Pythona i R. Dalej w tym rozdziale poznasz zaawansowane funkcje narzędzia `SQLAlchemy`, umożliwiające komunikację z bazą za pomocą Pythona.

Wprowadzenie

Aby wyciągać wnioski na podstawie bazy, potrzebujesz danych. Choć wiele firm przechowuje i aktualizuje dane w centralnej bazie, zdarza się, że potrzebne są dodatkowe dane oprócz tych z bazy. W tym rozdziale zobaczysz, jak wydajnie wczytywać dane do centralnej bazy na potrzeby dalszych analiz.

Jednak pobieranie danych do bazy w celu późniejszych analiz nie jest jedyną potrzebną operacją. Jeśli przeprowadzasz zaawansowane analizy, zdarzają się sytuacje, gdy trzeba wyeksportować dane z bazy (na przykład w celu przeprowadzenia analiz statystycznych niedostępnych w `SQL-u`). Dlatego poznasz też proces pobierania danych z własnych baz. Pozwoli Ci to używać innego oprogramowania do analizowania danych.

W tym rozdziale zobaczysz, jak zintegrować proces pracy z dwoma językami programowania często wykorzystywanymi do analizy danych — Pythonem i R. Te języki dają duże możliwości, ponieważ są łatwe w użyciu, udostępniają zaawansowane funkcje, są oprogramowaniem o otwartym dostępie do kodu źródłowego, a dzięki ich popularności rozwój tych języków wspomaga duża społeczność. Zobaczysz, w jaki sposób wydajnie przekazywać duże zbiory danych między językami programowania i bazami, co pozwoli tworzyć procesy pracy z wykorzystaniem dostępnych narzędzi analitycznych.

Zaczniesz od zapoznania się z masowym wczytywaniem i pobieraniem danych za pomocą polecenia `COPY` z systemu PostgreSQL oraz uruchamianego w wierszu poleceń klienta `psql`. Następnie przejdziesz do importowania i eksportowania danych z wykorzystaniem języków Python i R. Najpierw zapoznaj się z działaniem polecenia `COPY`.

Polecenie COPY

Na tym etapie zapewne znasz już instrukcję `SELECT` (opisaną w rozdziale 1., „Wprowadzenie do SQL-a dla analityków”), która umożliwia pobieranie danych z bazy. Choć jest ona przydatna dla małych zbiorów danych, które można szybko przejrzeć, duże zbiory danych często warto zapisać w pliku. Taki plik umożliwia dalsze lokalne przetwarzanie i analizowanie danych z Excela, Pythona i R. Do pobierania dużych zbiorów danych możesz użyć polecenia `COPY` z systemu PostgreSQL. To polecenie w wydajny sposób przenosi dane z bazy do pliku lub z pliku do bazy.

Instrukcja `COPY` pobiera dane z bazy i zapisuje je w pliku o wskazanym formacie. Przyjrzyj się tej instrukcji:

```
COPY (SELECT * FROM customers LIMIT 5) TO STDOUT WITH CSV HEADER;
```

Rysunek 4.1 przedstawia dane wyjściowe tego kodu.

```
customer_id,title,first_name,last_name,suffix,email,gender,ip_address,phone,street_address,city,state,postal_code,latitude,longitude,date_added
1,,Arlena,Riveles,,ariveles0@stumbleupon.com,F,98.36.172.246,,,,,,2017-04-23 00:00:00
2,,Dr,Ode,Stovin,,ostovin1@npr.org,M,16.97.59.186,314-534-4361,2573 Fordem Parkway,Saint Louis,MO,63116,38.5814,-90.2625,2014-10-02 00:00:00
3,,Braden,Jordan,,bjordan2@geocities.com,M,192.86.248.59,,5651 Kennedy Park,Pensacola,FL,32590,30.6143,-87.2758,2018-10-27 00:00:00
4,,Jessica,Nussen,,jnussen3@salon.com,F,159.165.138.166,615-824-2506,224 Village Circle,Nashville,TN,37215,36.0986,-86.8219,2017-09-03 00:00:00
5,,Lonnie,Rembaud,,lrembaud4@discovery.com,F,18.131.58.65,786-499-3431,38 Lindbergh Way,Miami,FL,33124,25.5584,-80.4582,2014-03-06 00:00:00
```

Rysunek 4.1. Użycie polecenia `COPY` do wyświetlania wyników w standardowym wyjściu w formacie CSV

Ta instrukcja zwraca pięć wierszy z tabeli `customers`. Każdy rekord znajduje się w nowym wierszu, a wartości są rozdzielone przecinkami. Jest to format typowy dla plików `.csv`. Na początku dodany jest nagłówek.

- Ponieważ jako docelową lokalizację polecenia `COPY` podano standardowe wyjście (`STDOUT`), wyniki są kopiowane do wiersza poleceń, a nie do pliku.

Oto omówienie tego polecenia i przekazanych do niego parametrów:

- COPY to polecenie służące do przenoszenia danych do pliku.
- (SELECT * FROM customers LIMIT 5) to kwerenda, której wyniki należy skopiować.
- Człon TO STDOUT określa, że wyniki należy wyświetlić zamiast zapisywać je w pliku na dysku twardym. Wyświetlanie w „standardowym wyjściu” oznacza pokazywanie danych wyjściowych w wierszu poleceń lub terminalu.
- WITH to opcjonalne słowo kluczowe rozdzielające parametry używane przy przenoszeniu danych z bazy do pliku.
- CSV oznacza, że stosowany jest format pliku .csv. Można też użyć wartości BINARY lub pominąć ten parametr, aby otrzymać dane wyjściowe w formacie tekstowym.
- HEADER oznacza, że wyświetlony ma zostać także nagłówek.

Więcej o parametrach polecenia COPY dowiesz się z dokumentacji systemu PostgreSQL dostępnej na stronie <https://www.postgresql.org/docs/current/sql-copy.html>.

Choć opcja STDOUT jest przydatna, często dane warto zapisać w pliku. Polecenie COPY to umożliwia, przy czym dane są zapisywane **lokalnie** na serwerze PostgreSQL. Musisz podać pełną ścieżkę do pliku; ścieżki względne nie są dozwolone. Jeśli na Twoim komputerze działa baza PostgreSQL, możesz to sprawdzić, uruchamiając następujące polecenie:

```
COPY (SELECT * FROM customers LIMIT 5) TO '/ścieżka/do/plik.csv' WITH CSV
HEADER;
```

Z następnego punktu dowiesz się, jak kopiować dane za pomocą narzędzia psql.

Wartość w apostrofach po słowie kluczowym TO to pełna ścieżka do pliku wyjściowego. Format ścieżki zależy od używanego systemu operacyjnego. W systemach Linux i macOS separatorem katalogów jest prawy ukośnik (/), a katalog główny podstawowego dysku to /. Jednak w systemie Windows separatorem katalogów jest lewy ukośnik (\), a ścieżka rozpoczyna się od litery dysku.

Kopiowanie danych za pomocą narzędzia psql

Choć zapewne używałeś bazy z systemu PostgreSQL za pomocą klienta z interfejsem graficznym, możesz nie wiedzieć, że jednym z pierwszych klientów tego systemu był uruchamiany w wierszu poleceń program psql. Ten interfejs nadal jest dostępny, a psql umożliwia uruchamianie skryptów systemu PostgreSQL komunikujących się ze środowiskiem lokalnym. Pozwala to zdalnie wywoływać polecenie COPY za pomocą specjalnej instrukcji narzędzia psql, \copy.

Aby uruchomić psql, wywołaj w terminalu następujące polecenie:

```
psql -h Twój_host -p 5432 -d Twoja_baza_danych -U Twoja_nazwa_użytkownika
```

W tym poleceniu przekazywane są flagi, które zapewniają informacje potrzebne do nawiązania połączenia z bazą:

- Flaga `-h` służy do podawania nazwy hosta. Po flagdzie znajduje się spacja, a dalej łańcuch znaków z nazwą hosta używanej bazy. Możesz podać adres IP, nazwę domeny lub, jeśli używasz maszyny lokalnej, łańcuch znaków `'localhost'`.
- Flaga `-p` pozwala podać port bazy danych. Dla baz systemu PostgreSQL jest to zwykle 5432.
- Flaga `-d` określa nazwę bazy danych. Po tej flagdzie i spacji należy wpisać nazwę bazy.
- Flaga `-U` umożliwia podanie nazwy użytkownika. Po tej flagdzie i spacji wpisz nazwę użytkownika.

Po nawiązaniu połączenia z bazą za pomocą narzędzia `psql` możesz przetestować instrukcję `\copy`, korzystając z następującej instrukcji:

```
\copy (SELECT * FROM customers LIMIT 5) TO 'plik.csv' WITH CSV HEADER;
```

Tak wyglądają dane wyjściowe tego kodu:

```
COPY 5
Time: 22.208 ms
```

Oto omówienie tego polecenia i przekazanych parametrów:

- Instrukcja `\copy` uruchamia polecenie `COPY ... TO STDOUT ...` z systemu PostgreSQL, aby zwrócić dane.
- `(SELECT * FROM customers LIMIT 5)` to kwerenda, której wyniki chcesz skopiować.
- Człon `TO 'plik.csv'` oznacza, że `psql` ma zapisać dane wyjściowe ze standardowego wyjścia w pliku *plik.csv*.
- Parametry `WITH CSV HEADER` działają tak samo jak w poprzednim poleceniu.

Możesz podejrzeć zawartość pliku *plik.csv* w dowolnym edytorze tekstu. Przedstawia ją rysunek 4.2.

```
customer_id,title,first_name,last_name,suffix,email,gender,ip_address,phone,street_address,city,state,postal_code,latitude,longitude,date_added
1,,Arlene,Riveles,,ariveles0@stumbleupon.com,F,98.36.172.246,,,,,,2017-04-23 00:00:00
2,,Dr,Ode,Stovin,,ostovini1@npr.org,M,16.97.59.186,314-534-4361,2573 Fordem Parkway,Saint Louis,MO,63116,38.5814,-90.2625,2014-10-02 00:00:00
3,,Braden,Jordan,,bjordan2@geocities.com,M,192.86.248.59,,5651 Kennedy Park,Pensacola,FL,32590,30.6143,-87.2758,2018-10-27 00:00:00
4,,Jessica,Nussen,,jnussen3@salon.com,F,159.165.138.166,615-824-2506,224 Village Circle,Nashville,TN,37215,36.0986,-86.8219,2017-09-03 00:00:00
5,,Lannie,Rembaud,,lrembaud4@discovery.com,F,18.131.58.65,786-499-3431,38 Lindbergh Way,Miami,FL,33124,25.5584,-80.4582,2014-03-06 00:00:00
[sql>] #
```

Rysunek 4.2. Plik CSV utworzony za pomocą polecenia `\copy`

Warto zauważyć, że polecenie `\copy` nie umożliwia umieszczania znaków nowego wiersza w kwerendach. Prosty sposób na wykonywanie kwerend wielowierszowych jest utworzenie widoku z potrzebnymi danymi przed uruchomieniem polecenia `\copy` i usunięcie tego widoku po wykonaniu polecenia.

Możesz utworzyć widok `customers_sample`, posługując się następującą składnią:

```
CREATE TEMP VIEW customers_sample AS (
  SELECT *
  FROM customers
  LIMIT 12
);
```

W tym przykładzie dane wyjściowe kwerendy są zapisywane w widoku tymczasowym. Dane można z niego pobierać za pomocą składni podobnej do tej, która jest stosowana dla tabel. Przyjrzyj się na przykład temu kodowi:

```
SELECT COUNT(1) FROM customers_sample;
```

Zwróci on wartość 12.

Widok przypomina tabelę, przy czym sam nie zawiera danych. Zamiast tego przy każdym użyciu widoku uruchamiana jest powiązana z nim kwerenda. Słowo kluczowe `TEMP` oznacza, że widok może zostać automatycznie usunięty po zakończeniu sesji.

Możesz też ręcznie usunąć widok, używając prostego polecenia:

```
DROP VIEW customers_sample;
```

Przyjrzyj się następującym poleceniom:

```
CREATE TEMP VIEW customers_sample AS (
  SELECT *
  FROM customers
  LIMIT 5
);
\copy customers_sample TO 'plik.csv' WITH CSV HEADER
DROP VIEW customers_sample;
```

Dane wyjściowe będą tu identyczne jak w pierwszym przykładzie ilustrującym eksportowanie. Choć możesz wykonywać opisaną operację na oba sposoby, to z uwagi na większą czytelność dłuższe kwerendy będziemy w tej książce zapisywać z użyciem drugiej składni.

Konfigurowanie poleceń `COPY` i `\copy`

Dostępnych jest kilka opcji, które umożliwiają skonfigurowanie poleceń `COPY` i `\copy`:

- Opcję `FORMAT nazwa_formatu` można zastosować do określenia formatu danych. Jako `nazwa_formatu` można podać `csv`, `text` lub `binary`. Możesz też podać format `CSV` lub `BINARY` bez słowa kluczowego `FORMAT`, a także zupełnie pominąć format (wtedy dane wyjściowe będą miały domyślny format pliku tekstowego).
- Opcja `DELIMITER 'ogranicznika'` pozwala podać ogranicznik z plików `CSV` i tekstowych (na przykład `,` w plikach `CSV` lub `|` w plikach rozdzielonych znakiem potoku).

- Opcja NULL 'tekstowa_reprezentacja_null' pozwala określić, jak mają być reprezentowane wartości NULL (na przykład ' ', jeśli spacja reprezentuje NULL, lub 'NULL', jeżeli to słowo ma oznaczać brakujące wartości w danych).
- Słowo kluczowe HEADER oznacza, że należy wyświetlić nagłówki w danych wyjściowych.
- Opcja QUOTE 'symbol_cudzysłowu' pozwala określić, między jakimi symbolami umieszczać pola ze znakami specjalnymi (na przykład z przecinkiem w wartości tekstowej w plikach CSV), aby instrukcja COPY ignorowała znaczenie tych znaków.
- Opcja ESCAPE 'znak_ucieczki' służy do podawania znaku do tworzenia sekwencji ucieczki.
- Opcja ENCODING 'nazwa_kodowania' umożliwia podanie kodowania. Jest to przydatne zwłaszcza wtedy, gdy przetwarzasz zawierające znaki specjalne dane w językach obcych lub dane wejściowe od użytkownika.

Na przykład poniższa instrukcja tworzy plik rozdzielony znakami potoku z nagłówkiem, gdzie brakujące wartości (NULL) są reprezentowane za pomocą pustych łańcuchów znaków (o zerowej długości), a znakiem cudzysłowu jest cudzysłów ("):

```
\copy customers TO 'plik.csv' WITH CSV HEADER DELIMITER '|' NULL ''
QUOTE ''
```

Oto dane wyjściowe tego kodu:

```
COPY 50000
```

W następnym punkcie użyjesz poleceń COPY i \copy do wczytania dużych zbiorów danych do bazy.

Użycie poleceń COPY i \copy do masowego wczytywania danych do bazy

Pokazaliśmy już, że polecenia COPY i \copy możesz wykorzystać do wydajnego pobierania danych z bazy. Jednak umożliwiają one także wczytywanie danych do bazy.

Polecenia COPY i \copy wczytują dane dużo wydajniej niż instrukcja INSERT. Wynika to z kilku powodów:

- Gdy używasz polecenia COPY, zatwierdzanie danych odbywa się tylko raz, po wstawieniu wszystkich wierszy.
- Komunikacja między bazą danych i klientem jest mniej intensywna, dlatego latencja sieci jest mniej odczuwalna.
- PostgreSQL stosuje optymalizacje polecenia COPY, które są niedostępne za pomocą instrukcji INSERT.

Oto przykład pokazujący, jak użyć polecenia \copy do skopiowania wierszy z pliku do tabeli:

```
\copy customers FROM 'plik.csv' CSV HEADER DELIMITER '|'
```

To polecenie zwraca następujące dane wyjściowe:

```
COPY 50000
```

Oto omówienie tego polecenia i przekazanych parametrów:

- Polecenie `\copy` wywołuje instrukcję `COPY ... FROM STDOUT...` z systemu PostgreSQL, aby wczytać dane do bazy.
- `customers` to nazwa tabeli, do której chcesz dołączyć dane.
- Człon `FROM 'plik.csv'` określa, że wczytywane są rekordy z pliku *plik.csv*. Słowo kluczowe `FROM` informuje, że rekordy są wczytywane z pliku; wcześniej używane słowo kluczowe `TO` służy do pobierania rekordów do pliku.
- Parametry `WITH CSV HEADER` działają tak samo jak wcześniej.
- Fragment `DELIMITER ','` określa ogranicznik stosowany w pliku. W plikach CSV domyślnie jest to przecinek, dlatego ten parametr nie jest konieczny. Jednak użycie go zwiększa czytelność, dlatego może być przydatne — choćby po to, aby przypomnieć Ci o formacie pliku.

Choć polecenia `COPY` i `\copy` świetnie nadają się do eksportowania danych do innych narzędzi, PostgreSQL udostępnia dodatkowy mechanizm do eksportowania kopii zapasowej bazy. Do wykonywania tego zadania administracyjnego możesz używać instrukcji `pg_dump` (eksportowanie określonej tabeli) lub `pg_dumpall` (eksportowanie całej bazy bądź całego schematu). Te instrukcje pozwalają nawet zapisać dane w formie skompresowanej (w formacie `tar`), aby zaoszczędzić miejsce. Niestety format wyjściowy tych poleceń to zwykle instrukcje w SQL-u, których nie da się łatwo wykorzystać poza systemem PostgreSQL. Dlatego instrukcje te nie są pomocne, gdy chcesz importować dane z innych narzędzi analitycznych, takich jak języki Python i R, lub je do nich eksportować.

Pokazaliśmy już, jak importować i eksportować dane. Teraz wykonasz ćwiczenie, w którym wyeksportujesz dane do pliku i będziesz przetwarzać je w Excelu.

W ćwiczeniach i zadaniach z tego rozdziału będziesz potrzebować dostępu do bazy za pomocą narzędzia `psql`. Pliki z tego rozdziału możesz znaleźć w serwisie GitHub (<https://packt.live/2B5PGTd>).

Ćwiczenie 4.01 — eksportowanie danych do pliku w celu dalszego przetwarzania ich w Excelu

W tym ćwiczeniu celem jest wykorzystanie nowej wiedzy o eksportowaniu danych za pomocą narzędzia `psql` i polecenia `\copy` do zwizualizowania danych na komputerze. Zapiszesz plik z miastami o największej liczbie klientów firmy ZoomZoom. Te analizy pomogą zarządowi firmy zdecydować, gdzie warto otworzyć nowy salon.

1. Aby wykonać to ćwiczenie, otwórz wiersz poleceń (na przykład CMD w systemie Windows lub terminal w systemie macOS) i nawiąż połączenie z bazą za pomocą narzędzia `psql`.
2. Utwórz widok `top_cities`. Skopiuj tabelę `customers` z bazy firmy ZoomZoom do lokalnego pliku w formacie `.csv`. W tym celu możesz utworzyć widok tymczasowy przy użyciu pokazanego dalej polecenia. Zauważ, że przed nazwą pliku, `'top_cities.csv'`, trzeba dodać ścieżkę w formacie dostosowanym do systemu operacyjnego, aby określić miejsce zapisu pliku.

```
CREATE TEMP VIEW top_cities AS (
    SELECT city,
           count(1) AS number_of_customers
    FROM customers
    WHERE city IS NOT NULL
    GROUP BY 1
    ORDER BY 2 DESC
    LIMIT 10
);
\copy (SELECT * FROM top_cities) TO 'top_cities.csv' WITH CSV HEADER
DELIMITER ','
```

3. Utwórz widok `top_cities`. Skopiuj tabelę `customers` z bazy firmy ZoomZoom do lokalnego pliku w formacie `.csv`:

```
\copy top_cities TO 'top_cities.csv' WITH CSV HEADER DELIMITER ','
```

4. Usuń widok:

```
DROP VIEW top_cities;
```

Oto omówienie użytych instrukcji:

`CREATE TEMP VIEW top_cities AS (...)` tworzy nowy widok tymczasowy.
`SELECT city, count(1) AS number_of_customers ...` to kwerenda, która zwraca liczbę klientów z każdego miasta. Ponieważ używana jest klauzula `LIMIT 10`, kwerenda pobiera tylko 10 pierwszych miast uporządkowanych według drugiej kolumny (z liczbą klientów). Ponadto kod odfiltrowuje klientów, którzy nie podali miasta.

Polecenie `\copy ...` kopiuje dane z widoku do pliku `top_cities.csv` na Twoim komputerze.

Instrukcja `DROP VIEW top_cities;` usuwa widok, ponieważ nie jest już potrzebny.

Jeśli otworzysz plik tekstowy `top_cities.csv`, zobaczysz dane wyjściowe widoczne na rysunku 4.3.

Tu plik wyjściowy to `top_cities.csv`. Użyjesz go w dalszych ćwiczeniach z tego rozdziału.

Po zapisaniu danych wyjściowych z bazy w pliku CSV możesz otworzyć go w arkuszu kalkulacyjnym, na przykład w Excelu.

```

city,number_of_customers
Washington,1447
Houston,904
New York City,731
El Paso,713
Dallas,607
Atlanta,571
Sacramento,506
Los Angeles,466
San Antonio,426
Miami,426

```

Rysunek 4.3. Dane wyjściowe polecenia \copy

- Otwórz plik *top_cities.csv* w Excelu lub w innym arkuszu kalkulacyjnym albo edytorze tekstu (rysunek 4.4).

	A	B
1	city	number_of_customers
2	Washington	1447
3	Houston	904
4	New York City	731
5	El Paso	713
6	Dallas	607
7	Atlanta	571
8	Sacramento	506
9	Los Angeles	466
10	San Antonio	426
11	Miami	426

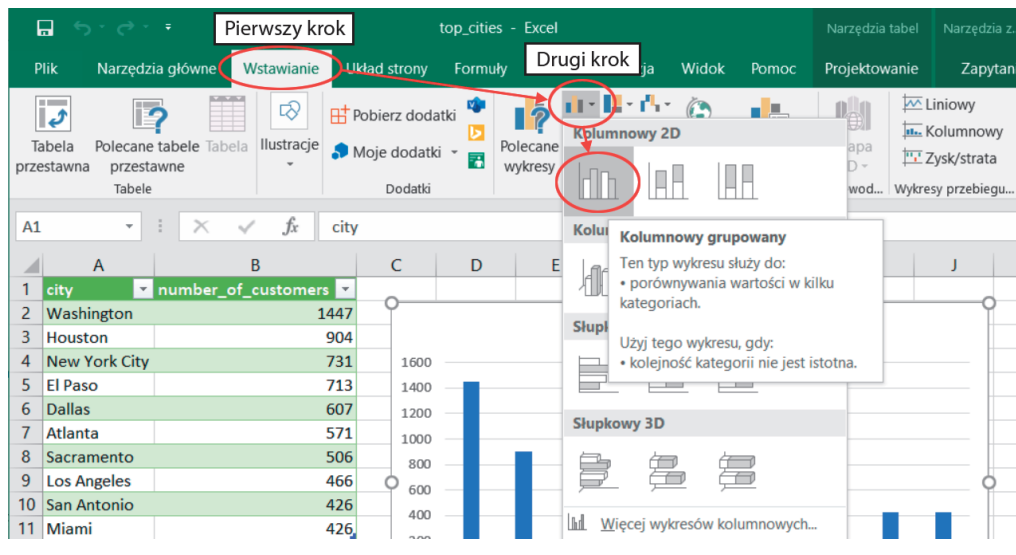
Rysunek 4.4. Plik *top_cities.csv* otwarty w Excelu

- Teraz zaznacz wszystkie dane tak jak na rysunku 4.5. Tu obejmują one komórki od A1 do B11.

	A	B
1	city	number_of_customers
2	Washington	1447
3	Houston	904
4	New York City	731
5	El Paso	713
6	Dallas	607
7	Atlanta	571
8	Sacramento	506
9	Los Angeles	466
10	San Antonio	426
11	Miami	426

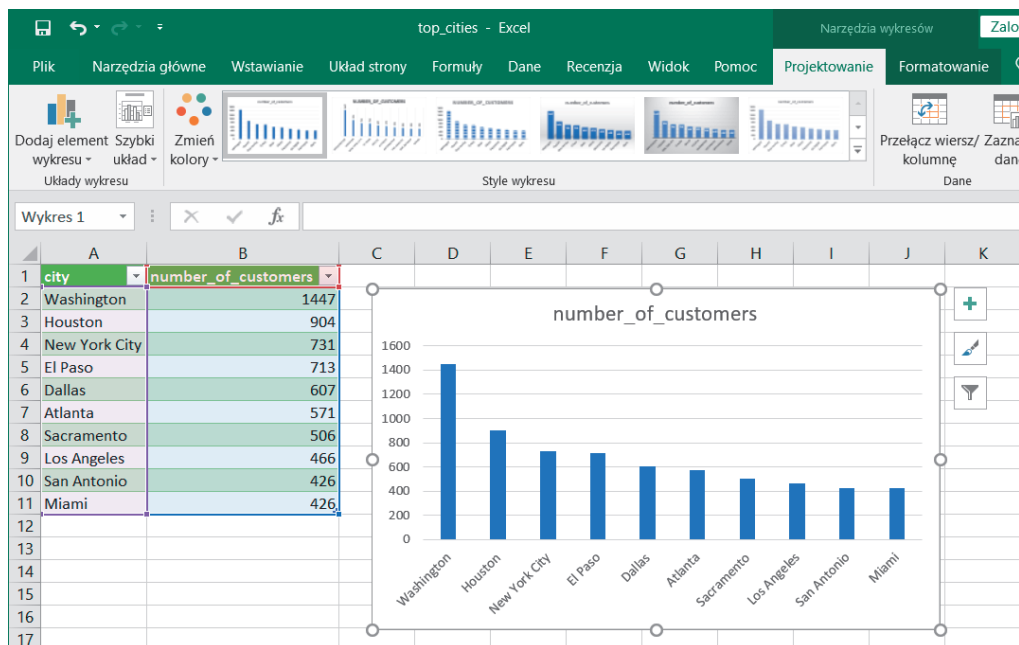
Rysunek 4.5. Zaznacz cały zbiór danych. W tym celu kliknij komórkę A1 i przeciągnij kursor do komórki B11

7. Teraz w górnym menu wybierz zakładkę *Wstawianie*, po czym kliknij ikonę wykresu kolumnowego, aby utworzyć wykres *Kolumnowy 2D* (rysunek 4.6).



Rysunek 4.6. Wstaw wykres kolumnowy, aby zwizualizować zaznaczone dane

8. Na koniec powinieneś otrzymać dane wyjściowe widoczne na rysunku 4.7.



Rysunek 4.7. Końcowe dane wyjściowe wizualizacji

Na wykresie widać, że bardzo duża liczba klientów pochodzi z Waszyngtonu. Na podstawie tych prostych analiz można stwierdzić, że oczywistym następnym celem ekspansji firmy ZoomZoom jest właśnie Waszyngton.

Kod z tego fragmentu książki znajdziesz na stronie <https://packt.live/2AsBNP9>.

W tym ćwiczeniu wykorzystaliśmy dane w narzędziu analitycznym. Wyeksportowaliśmy dane za pomocą programu `psql` i polecenia `\copy`, aby zwizualizować je w Excelu. Takie analizy mogą pomóc zarządowi w podjęciu na podstawie danych decyzji o tym, gdzie warto otworzyć następny salon. Teraz zobaczysz, jak wykorzystać zaawansowane programistyczne narzędzia analityczne.

Zastosowanie języka R do bazy danych

Potrąfisz już kopiować dane do bazy i je z niej pobierać. Dzięki temu możesz wyjść poza SQL i zastosować inne narzędzia do analizy danych (takie jak Excel), a także wykorzystać dowolny program, który potrafi wczytać dane wejściowe w postaci pliku CSV. Choć prawie wszystkie narzędzia analityczne potrafią wczytywać pliki CSV, nadal musisz pobrać dane. Dodatkowe kroki w procesie analiz mogą zwiększyć jego złożoność. Jest ona niepożądana, ponieważ wymaga dodatkowych prac administracyjnych i zwiększa liczbę punktów, w których mogą wystąpić błędy.

Inna technika polega na bezpośrednim powiązaniu bazy danych z kodem do przeprowadzania analiz. W tej części rozdziału zobaczysz, jak wykorzystać do tego R — język programowania zaprojektowany specjalnie na potrzeby obliczeń statystycznych. Z dalszej części rozdziału dowiesz się, jak zintegrować proces przetwarzania danych z Pythonem.

Po co korzystać z języka R?

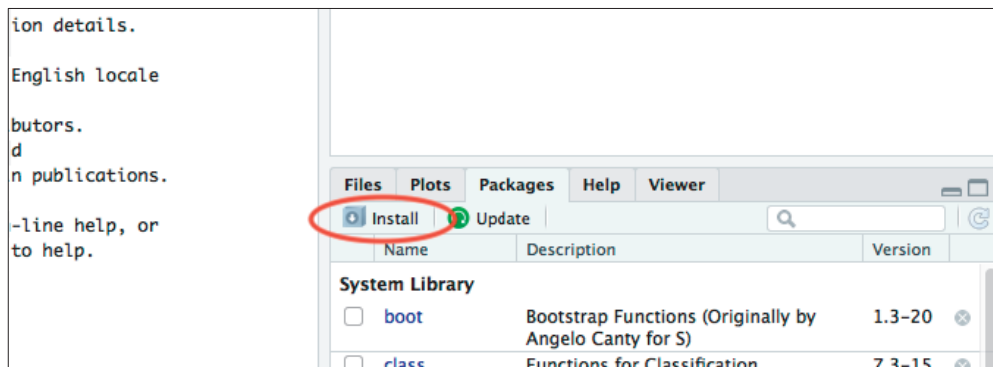
Choć udało Ci się obliczyć zbiorcze statystyki opisowe za pomocą samego SQL-a, R umożliwia przeprowadzanie innych analiz statystycznych, w tym obsługuje uczenie maszynowe, analizę regresji i testy istotności. Pozwala też generować wizualizacje danych, które uwidaczniają trendy i ułatwiają ich interpretowanie. R udostępnia prawdopodobnie więcej mechanizmów statystycznych niż jakiegokolwiek inne oprogramowanie analityczne.

Wprowadzenie do języka R

Ponieważ R jest językiem o otwartym dostępie do kodu źródłowego i działa w systemach Windows, macOS i Linux, bardzo łatwo jest rozpocząć korzystanie z niego. Oto kroki, dzięki którym szybko przygotujesz środowisko języka R:

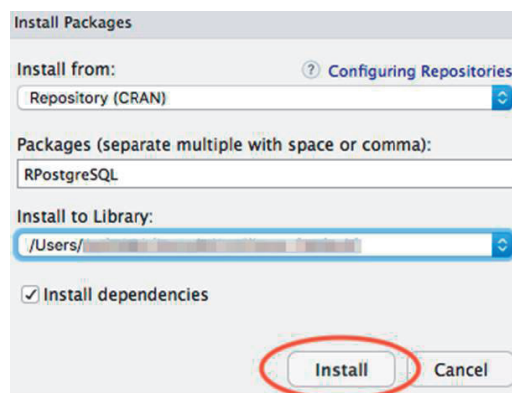
1. Pobierz najnowszą wersję języka R ze strony <https://cran.r-project.org/>.

- Po zainstalowaniu R możesz pobrać ze strony <http://rstudio.org/download/desktop> i zainstalować RStudio, **środowisko IDE** służące do programowania w R.
- Następnie zainstaluj w R pakiet RPostgreSQL. W tym celu otwórz w RStudio zakładkę *Packages* i kliknij ikonę *Install* (rysunek 4.8).



Rysunek 4.8. Instalowanie pakietów języka R w zakładce Packages w RStudio

- Teraz poszukaj pakietu RPostgreSQL w oknie *Install Packages* (rysunek 4.9) i go zainstaluj.

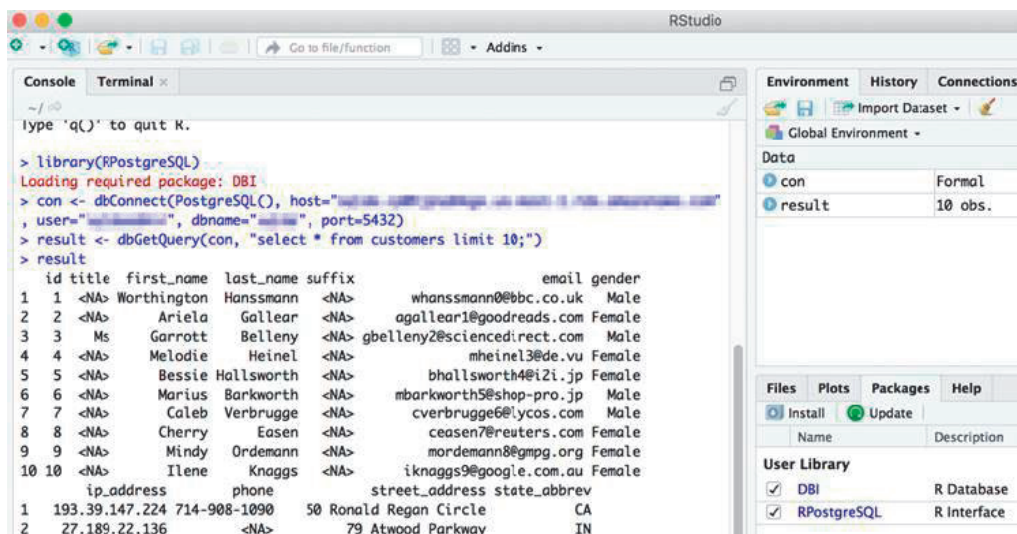


Rysunek 4.9. Okno Install Packages w RStudio umożliwia wyszukanie pakietu

- Teraz możesz użyć pakietu RPostgreSQL, aby wczytać dane do R. Posłuż się następującymi poleceniami:

```
library(RPostgreSQL)
con <- dbConnect(PostgreSQL(), host="my_host", user="my_username",
password="my password", dbname="zoomzoom", port=5432)
result <- dbGetQuery(con, "select * from customers limit 10;")
result
```

Dane wyjściowe są pokazane na rysunku 4.10.



Rysunek 4.10. Dane wyjściowe uzyskane po połączeniu się w R z bazą danych

Oto omówienie użytych instrukcji:

`library(RPostgreSQL)` to składnia służąca do wczytywania bibliotek w R.

Polecenie `con <- dbConnect(PostgreSQL(), host="Twój_host", user="Twoja_nazwa_użytkownika", password="Twoje_hasło", dbname="zoomzoom", port=5432)` nawiązuje połączenie z bazą. Wprowadzane są tu wszystkie parametry bazy danych, dlatego zastąp użyte tu wartości informacjami z Twojej konfiguracji. Jeśli przygotowałeś plik `.pgpass`, możesz pominąć parametr `password`.

Polecenie `result <- dbGetQuery(con, "select * from customers limit 10;")` uruchamia prostą kwerendę w celu przetestowania połączenia i sprawdzenia wyniku. Dane są zapisywane w zmiennej `result` jako ramka danych języka R.

W ostatnim wierszu `result` to nazwa zmiennej przechowującej ramkę danych. Jeśli w poleceniu nie ma przypisania, terminal języka R wyświetla zawartość zmiennej lub wyrażenia.

Na tym etapie z powodzeniem wyeksportowaliśmy dane z bazy do języka R. W ten sposób przygotowaliśmy podstawy do dowolnych analiz, jakie możesz chcieć przeprowadzić. Po wczytaniu danych do R możesz kontynuować przetwarzanie danych, poznając inne pakiety języka R i związane z nimi techniki. Na przykład pakiet `dplyr` umożliwia operowanie danymi i ich przekształcanie, a `ggplot2` służy do wizualizowania danych.

Teraz poznasz inny język programowania często używany do analizy danych — Pythona.

Zastosowanie języka Python do bazy danych

Choć R ma duże możliwości, wielu specjalistów od data science i analityków danych zaczyna używać Pythona. Dlaczego tak się dzieje? Ponieważ Python to wysokopoziomowy język, który można łatwo wykorzystać do przetwarzania danych. Liczba pakietów statystycznych i mechanizmów w R nadal jest większa niż w Pythonie, jednak Python jest szybko rozwijany i w większości najnowszych ankiet zwykle okazywał się popularniejszy od R. Ponadto wiele funkcji w Pythonie działa szybciej niż w R, co po części wynika z tego, że wiele mechanizmów w Pythonie jest napisanych w języku niższego poziomu — C.

Inną ważną zaletą Pythona jest jego wszechstronność. R jest używany prawie wyłącznie do badań naukowych i analiz statystycznych, natomiast Pythona można stosować do niemal dowolnych zadań — od analiz statystycznych do tworzenia aplikacji internetowych. Dlatego społeczność użytkowników Pythona jest znacznie większa niż społeczność użytkowników języka R. Większa społeczność jest ważną zaletą, ponieważ gwarantuje więcej pomocy (na przykład w serwisie Stack Overflow), a każdego dnia pojawiają się nowe pakiety i moduły Pythona. Ostatnia zaleta Pythona związana jest z tym, że jest to język programowania do użytku ogólnego, dlatego łatwiej jest napisany w tym języku kod wdrażać w środowisku produkcyjnym, a niektóre mechanizmy kontrolne (na przykład przestrzenie nazw) sprawiają, że jest on mniej podatny na błędy.

Z tych powodów nauka Pythona może się okazać lepszym pomysłem, chyba że potrzebne mechanizmy są dostępne tylko w R lub reszta zespołu korzysta z tego języka.

Po co korzystać z języka Python?

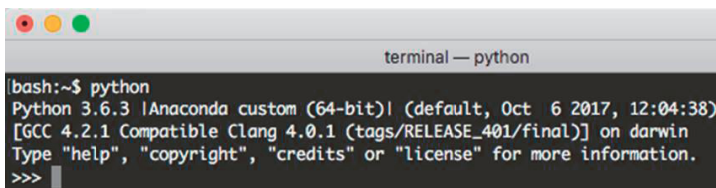
Mimo że SQL umożliwia obliczanie zbiorczych statystyk opisowych, Python (podobnie jak R) pozwala wykonywać inne analizy statystyczne i wizualizować dane. Ponadto Python pozwala tworzyć powtarzalne procesy, które można wdrożyć w środowisku produkcyjnym, a także umożliwia budowanie interaktywnych analitycznych serwerów WWW. R jest wyspecjalizowanym językiem programowania, a Python to język ogólny, „spec od wszystkiego”. Niezależnie od tego, jakie masz wymagania z obszaru analiz, prawie zawsze możesz wykonać zadanie za pomocą narzędzi dostępnych w Pythonie.

Wprowadzenie do języka Python

Chociaż istnieje wiele sposobów na rozpoczęcie pracy z Pythonem, dystrybucja Anaconda umożliwia wyjątkowo łatwe pobranie i zainstalowanie tego języka oraz innych narzędzi analitycznych, a także automatycznie instaluje liczne popularne pakiety analityczne wraz ze światowym menedżerem pakietów. Dlatego tu skorzystamy właśnie z tej dystrybucji.

Wykonaj następujące kroki, aby zainstalować dystrybucję Anaconda i nawiązać połączenie z systemem PostgreSQL:

1. Pobierz i zainstaluj Anacondę (<https://www.anaconda.com/distribution/>).
W trakcie instalacji pamiętaj, aby zaznaczyć opcję 'Add Anaconda to PATH'.
2. Po zakończeniu instalacji otwórz terminal Anacondy dla systemu macOS lub Windows. Wpisz `python` w wierszu poleceń i sprawdź, czy masz dostęp do interpretera Pythona (rysunek 4.11).



Rysunek 4.11. Interpreter Pythona jest dostępny i gotowy do przyjmowania danych wejściowych

Jeśli wystąpił błąd, możliwe, że musisz podać ścieżkę do Pythona. Możesz wpisać `quit()`, aby wyjść z interpretera.

3. Teraz pobierz i zainstaluj klienta systemu PostgreSQL dla Pythona, `psycopg2`, używając narzędzia `conda` — menedżera pakietów z Anacondy. Aby zainstalować klienta systemu PostgreSQL, wpisz następującą instrukcję w wierszu poleceń:

```
pip install psycopg2
```

4. Otwórz interpreter Pythona i wczytaj dane z bazy. Aby uruchomić interpreter, wpisz `python` w wierszu poleceń.
5. Aby wczytać dane, zacznij wpisywać poniższy skrypt w Pythonie:

```
import psycopg2
with psycopg2.connect(host="Twój_host", user="Twoja_nazwa_użytkownika",
password="Twoje_hasło", dbname="zoomzoom", port=5432) as conn:
    with conn.cursor() as cur:
        cur.execute("SELECT * FROM customers LIMIT 5")
        records = cur.fetchall()
```

```
records
```

Na rysunku 4.12 widoczne są kod i dane wyjściowe.

Oto omówienie zastosowanych instrukcji:

Najpierw kod importuje pakiet `psycopg2`, używając polecenia `import psycopg2`. Następnie konfigurowany jest obiekt połączenia — `psycopg2.connect(host="Twój_host", user="Twoja_nazwa_użytkownika", password="Twoje_hasło", dbname="sqlda", port=5432)`.

```

bash:~$ python
Python 3.7.4 (default, Aug 13 2019, 15:17:50)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import psycopg2
>>> with psycopg2.connect(host="0.0.0.0", dbname="sqlda", port=5432)
as conn:
...     with conn.cursor() as cur:
...         cur.execute("SELECT * FROM customers LIMIT 5")
...         records = cur.fetchall()
...
>>> records
[(1, None, 'Arlena', 'Riveles', None, 'ariveles0@stumbleupon.com',
'F', '98.36.172.246', None, None, None, None, None, None, None,
datetime.datetime(2017, 4, 23, 0, 0)), (2, 'Dr', 'Ode', 'Stovin',
None, 'ostovin1@npr.org', 'M', '16.97.59.186', '314-534-4361', '2573
Fordem Parkway', 'Saint Louis', 'MO', '63116', 38.5814, -90.2625,
datetime.datetime(2014, 10, 2, 0, 0)), (3, None, 'Braden', 'Jordan',
None, 'bjordan2@geocities.com', 'M', '192.86.248.59', None, '5651
Kennedy Park', 'Pensacola', 'FL', '32590', 30.6143, -87.2758,
datetime.datetime(2018, 10, 27, 0, 0)), (4, None, 'Jessika', 'Nussen',
None, 'jnussen3@salon.com', 'F', '159.165.138.166', '615-824-2506',
'224 Village Circle', 'Nashville', 'TN', '37215', 36.0986, -86.8219,
datetime.datetime(2017, 9, 3, 0, 0)), (5, None, 'Lonnie', 'Rembaud',
None, 'lrembaud4@discovery.com', 'F', '18.131.58.65', '786-499-3431',
'38 Lindbergh Way', 'Miami', 'FL', '33124', 25.5584, -80.4582,
datetime.datetime(2014, 3, 6, 0, 0))]

```

Rysunek 4.12. Kod i dane wyjściowe uzyskane po nawiązaniu w Pythonie połączenia z bazą danych

Podane są tu wszystkie parametry bazy danych. Użyte wartości zastąp danymi ze swojej konfiguracji. Jeśli przygotowałeś plik *.pgpass*, możesz pominąć parametr *password*. Ta instrukcja w Pythonie jest opakowana w blok `with ... as conn`. Instrukcja `with` powoduje automatyczne usunięcie obiektu (tu jest to obiekt połączenia) po zresetowaniu wcięcia. Jest to bardzo przydatne w przypadku połączeń bazodanowych, ponieważ nieaktywne połączenie niepotrzebnie zużywa zasoby bazy danych. Obiekt połączenia możesz zapisać w zmiennej `conn`, korzystając z instrukcji `as conn`.

Gdy masz już połączenie, należy utworzyć obiekt `cursor`, który umożliwia wczytywanie danych z bazy. Instrukcja `conn.cursor()` tworzy obiekt `cursor` dla bazy danych, który pozwala wykonywać kwerendy SQL-a za pomocą połączenia z bazą. Instrukcja `with` powoduje automatyczne usunięcie kursora, gdy nie będzie już potrzebny.

Polecenie `cur.execute("SELECT * FROM customers LIMIT 5")` przekazuje kwerendę "SELECT * FROM customers LIMIT 5" do bazy danych, gdzie jest wykonywana.

Polecenie `records = cur.fetchall()` pobiera wszystkie wiersze z wyników kwerendy i przypisuje je do zmiennej `records`.

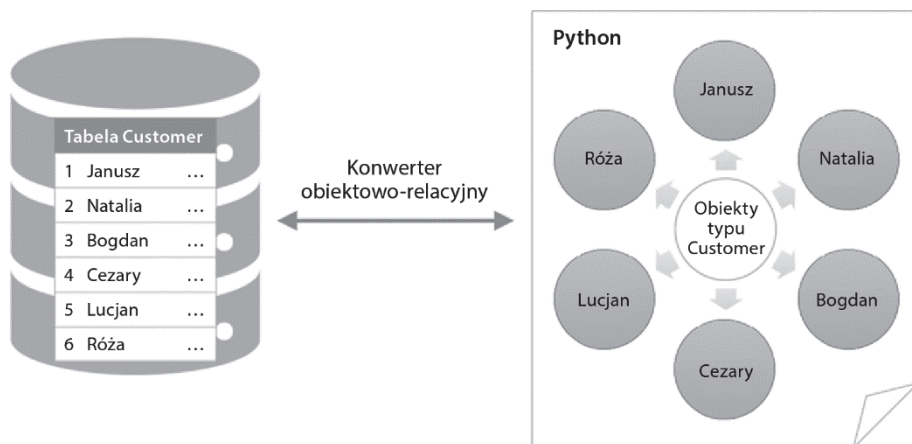
Po przesłaniu kwerendy do bazy i otrzymaniu rekordów można wyzerować poziom wcięcia. Aby wyświetlić wyniki, wpisz wyrażenie (tu jest to tylko nazwa zmiennej, `records`) i wciśnij *Enter*. W tym przykładzie dane wyjściowe to pięć pobranych rekordów z danymi klientów.

Choć udało Ci się nawiązać połączenie z bazą i wczytać dane, wymagało to kilku kroków, a składnia była bardziej skomplikowana niż w niektórych poprzednich rozwiązaniach. Narzędzie `psycopg2` ma duże możliwości, jednak pomocne może być użycie innych pakietów Pythona ułatwiających komunikację z bazą.

Ułatwianie dostępu do baz PostgreSQL w Pythonie za pomocą narzędzi SQLAlchemy i pandas

Chociaż `psycopg2` jest rozbudowanym klientem bazodanowym umożliwiającym dostęp do baz PostgreSQL w Pythonie, można uprościć kod, korzystając z kilku innych pakietów — `pandas` i `SQLAlchemy`.

Najpierw zapoznasz się z `SQLAlchemy`, pakietem SQL-owych narzędzi dla Pythona i **konwerterem obiektowo-relacyjnym** (ang. *Object-Relational Mapper*; rysunek 4.13), który odwzorowuje obiekty na tabele bazy danych i w drugą stronę. Przede wszystkim przyjrzyś się bibliotece bazodanowej `SQLAlchemy` i jej zaletom. Za pomocą tego narzędzia będziesz łatwo łączyć się z bazą danych, nie martwiąc się o połączenia i kursory.



Rysunek 4.13. Konwerter obiektowo-relacyjny odwzorowuje wiersze bazy danych na obiekty w pamięci

Następnie przyjrzyś się pakietowi Pythona `pandas`, który umożliwia operowanie danymi i ułatwia ich analizowanie. Ten pakiet pozwala zapisać strukturę tabeli z danymi (nazywaną **ramką danych**) w pamięci. Pakiet `pandas` udostępnia wysokopoziomowe API, które umożliwiają wczytanie danych z bazy za pomocą kilku wierszy kodu.

Choć oba te pakiety mają duże możliwości, warto zauważyć, że korzystają z pakietu `psycopg2` do łączenia się z bazą i wykonywania kwerend. Ważną zaletą tych pakietów jest to, że ukrywają niektóre złożone aspekty nawiązywania połączeń z bazą. Dzięki temu można łączyć się z bazą bez obaw, że zapomni się zamknąć połączenie lub usunąć kursor.

Czym jest SQLAlchemy?

Jest to pakiet narzędzi SQL-owych i konwerter obiektowo-relacyjny. Udostępnia wartościowe mechanizmy, lecz jego najważniejszą zaletą, na której skupisz się w tym miejscu, jest obiekt `Engine`.

Obiekt `Engine` z SQLAlchemy zawiera informacje na temat rodzaju bazy danych (tu będzie to baza PostgreSQL) i puli połączeń. Pula połączeń umożliwia utrzymywanie zestawu jednocześnie działających połączeń z bazą. Jest przydatna, nie trzeba bowiem tworzyć połączenia do momentu wysłania kwerendy do wykonania. Ponieważ połączenia są nawiązywane dopiero w momencie wykonywania kwerendy, w obiekcie `Engine` używana jest *leniwa inicjalizacja*. Słowo „leniwa” oznacza tu, że do czasu przesłania zapytania nic się nie dzieje (połączenie nie jest tworzone). Jest to korzystne, gdyż minimalizuje czas utrzymywania połączenia i ogranicza obciążenie bazy.

Inną zaletą obiektu `Engine` z SQLAlchemy jest to, że automatycznie zatwierdza zmiany w bazie danych spowodowane instrukcjami `CREATE TABLE`, `UPDATE` i `INSERT`, a także innymi poleceniami modyfikującymi bazę.

W omawianym przykładzie narzędzie SQLAlchemy jest przydatne, ponieważ zapewnia niezawodny obiekt `Engine`, który umożliwia korzystanie z bazy danych. Po zerwaniu połączenia ten obiekt potrafi je ponownie nawiązać, gdyż korzysta z puli połączeń. Udostępnia też wygodny interfejs dobrze współdziałający z innymi pakietami (na przykład z pakietem `pandas`).

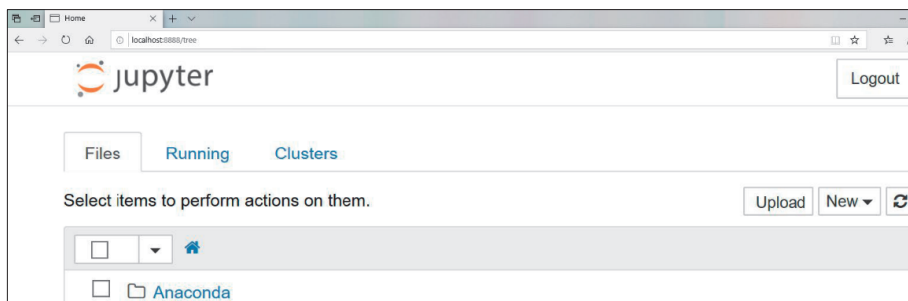
Używanie Pythona w narzędziu Jupyter Notebook

Oprócz interaktywnego używania Pythona w wierszu poleceń można korzystać z niego w notatniku w przeglądarce internetowej. Jest to przydatne do wyświetlania wizualizacji i przeprowadzania analiz eksploracyjnych.

W tym punkcie użyjesz narzędzia Jupyter Notebook instalowanego razem z systemem Anaconda. W wierszu poleceń uruchom następującą instrukcję:

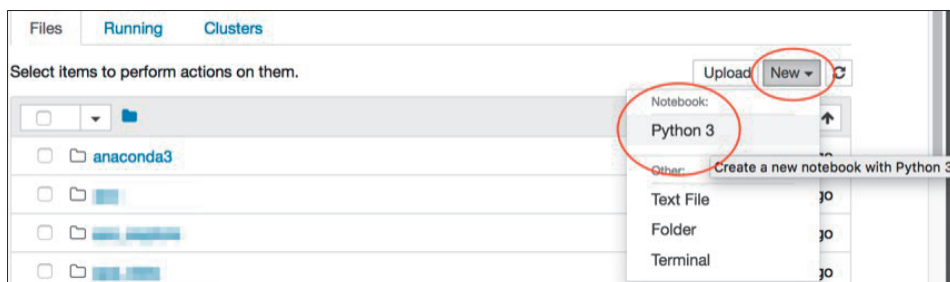
```
jupyter notebook
```

W domyślnej przeglądarce powinieneś zobaczyć ekran podobny do tego z rysunku 4.14.



Rysunek 4.14. Ekran narzędzia Jupyter Notebook w przeglądarce

Następnie utwórz nowy notatnik (rysunek 4.15).



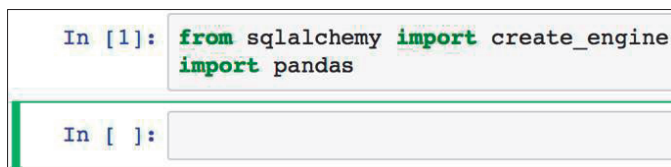
Rysunek 4.15. Otwieranie nowego notatnika Jupytera z kodem w Pythonie 3

W notatniku wpisz następujące instrukcje import:

```
from sqlalchemy import create_engine
import pandas as pd
```

Zauważ, że importowane są tu dwa pakiety. Pierwszy to moduł `create_engine` z pakietu `sqlalchemy`, a drugi to `pandas`, dla którego zwyczajowo tworzony jest alias `pd` (o mniejszej liczbie znaków). Za pomocą tych dwóch pakietów można pobierać dane z bazy, zapisywać je w niej lub wizualizować dane wyjściowe.

Wciśnij kombinację *Shift+Enter*, aby uruchomić te polecenia. Pojawi się nowa aktywna komórka, tak jak na rysunku 4.16.



Rysunek 4.16. Uruchamianie pierwszej komórki w notatniku Jupytera

Następnie skonfiguruj notatnik, aby wewnątrzwierszowo wyświetlał wykresy i wizualizacje. W tym celu uruchom to polecenie:

```
%matplotlib inline
```

Jest to informacja dla pakietu `matplotlib` (wymaganego przez pakiet `pandas`), aby w danym notatniku tworzył wykresy i wizualizacje wewnątrzwierszowo. Ponownie wciśnij kombinację *Shift+Enter*, aby przejść do następnej komórki.

W nowej komórce zdefiniuj łańcuch znaków połączenia:

```
cnxn_string = ("postgresql+psycopg2://{username}:{pswd}"
               "@{host}:{port}/{database}")
print(cnxn_string)
```

Ponownie wciśnij kombinację *Shift+Enter*. Powinieneś zobaczyć wyświetlony łańcuch znaków połączenia. Uzupełnij parametry i utwórz obiekt Engine dla bazy danych. Zastąp łańcuchy znaków rozpoczynające się od *Twoja*, *Twoje* lub *Twój* parametrami odpowiednimi dla połączenia:

```
engine = create_engine(cnxn_string.format(
    username="Twoja_nazwa_użytkownika",
    pswd="Twoje_hasło",
    host="Twój_host",
    port=5432,
    database="Twoja_nazwa_bazy"))
```

W tym poleceniu uruchamiana jest funkcja `create_engine`, aby utworzyć obiekt Engine dla bazy danych. Należy przekazać łańcuch znaków połączenia i dostosować go do połączenia z używaną bazą, podając wartości parametrów {username}, {pswd}, {host}, {port} i {database}. Jako host możesz podać adres IP, nazwę domeny lub słowo 'localhost' (jeśli baza jest przechowywana lokalnie).

Ponieważ narzędzie SQLAlchemy działa „leniwie”, o tym, czy nawiązywanie połączenia zakończyło się powodzeniem, dowiesz się dopiero przy próbie przesłania polecenia. Możesz sprawdzić, czy obiekt Engine dla bazy danych działa, wpisując poniższe polecenie i wciskając kombinację *Shift+Enter*:

```
engine.execute("SELECT * FROM customers LIMIT 2;").fetchall()
```

Powinieneś zobaczyć dane wyjściowe z rysunku 4.17.

```
[(1, None, 'Arlena', 'Riveles', None, 'ariveles0@stumbleupon.com', 'F', '98.36.172.246', None, None, None, None, None, None, None, None, datetime.datetime(2017, 4, 23, 0, 0)),
 (2, 'Dr', 'Ode', 'Stovin', None, 'ostovin1@npr.org', 'M', '16.97.59.186', '314-534-4361', '2573 Fordem Parkway', 'Saint Louis', 'MO', '63116', 38.5814, -90.2625, datetime.datetime(2014, 10, 2, 0, 0))]
```

Rysunek 4.17. Wykonywanie kwerendy w Pythonie

Dane wyjściowe tego polecenia to lista Pythona zawierająca wiersze z bazy zapisane w formie krotek. Choć udało Ci się wczytać dane z bazy, prawdopodobnie bardziej praktyczne będzie dla Ciebie wczytywanie danych do ramek pakietu pandas, co opisujemy w następnym punkcie.

Pobieranie danych z bazy i ich zapisywanie w bazie za pomocą pakietu pandas

Python udostępnia bardzo przydatne struktury danych, w tym listy, słowniki i krotki. Chociaż są one pomocne, dane często są zapisane w formie tabeli, z wierszami i kolumnami, podobnie jak dane w bazie. Wtedy przydatny jest obiekt ramki danych (DataFrame) pakietu pandas.

Pakiet pandas oprócz pomocnych struktur danych udostępnia też:

- funkcje do wczytywania danych bezpośrednio z bazy;
- mechanizmy do wizualizowania danych;
- narzędzia do analizy danych.

Jeśli kontynuujesz pracę w utworzonym wcześniej notatniku Jupytera, możesz użyć obiektu Engine z SQLAlchemy do wczytania danych do ramki pakietu pandas:

```
customers_data = pd.read_sql_table('customers', engine)
```

Teraz cała tabela customers jest zapisana jako ramka danych pakietu pandas w zmiennej customers_data. Funkcja read_sql_table pakietu pandas wymaga dwóch parametrów: nazwy tabeli i bazy, z którą można nawiązać połączenie (tu jako baza przekazywany jest obiekt Engine z SQLAlchemy). Można też użyć funkcji read_sql_query, która zamiast nazwy tabeli przyjmuje łańcuch znaków z kwerendą.

Na rysunku 4.18 pokazane jest, jak Twój notatnik może wyglądać na tym etapie.

```
In [1]: from sqlalchemy import create_engine
import pandas as pd
%matplotlib inline

In [2]: cnxn_string = ("postgresql+psycopg2://{username}:{pswd}"
                     "%(host)s:{port}/{database}")
print(cnxn_string)

postgresql+psycopg2://{username}:{pswd}%(host)s:{port}/{database}

In [3]: engine = create_engine(cnxn_string.format(
    username="ariveles0@stumbleupon.com",
    pswd="16.97.59.186",
    host="16.97.59.186",
    port=5432,
    database="customers"))

In [4]: engine.execute("SELECT * FROM customers LIMIT 2;").fetchall()
Out[4]: [(1, None, 'Arlena', 'Riveles', None, 'ariveles0@stumbleupon.com', 'F', '98.36.172.246', None, None, None, None, None,
None, None, datetime.datetime(2017, 4, 23, 0, 0)),
(2, 'Dr', 'Ode', 'Stovin', None, 'ostovin1@npr.org', 'M', '16.97.59.186', '314-534-4361', '2573 Fordem Parkway', 'Saint Louis', 'MO', '63116', 38.5814, -90.2625, datetime.datetime(2014, 10, 2, 0, 0))]

In [5]: customers_data = pd.read_sql_table('customers', engine)

In [ ]:
```

Rysunek 4.18. Cała zawartość notatnika Jupytera

Wiesz już, jak wczytywać dane z bazy, możesz więc zacząć przeprowadzać proste analizy i tworzyć wizualizacje. Wykonaj ćwiczenie, w którym wczytasz i zwizualizujesz dane za pomocą Pythona.

Ćwiczenie 4.02

— wczytywanie i wizualizowanie danych w Pythonie

W tym ćwiczeniu wczytasz dane zwracane przez bazę i zwizualizujesz wyniki. Użyjesz do tego języka Python, notatnika Jupytera, narzędzia SQLAlchemy i pakietu pandas. Przeanalizujesz informacje demograficzne na temat klientów z podziałem na miasta, aby lepiej zrozumieć docelowych odbiorców. Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Otwórz notatnik Jupytera z poprzedniego punktu i kliknij ostatnią, pustą komórkę.

2. Wprowadź poniższą kwerendę ujętą w potrójne cudzysłowy (taka sekwencja pozwala tworzyć w Pythonie łańcuchy znaków obejmujące kilka wierszy):

```
query = """
SELECT city,
COUNT(1) AS number_of_customers,
COUNT(NULLIF(gender, 'M')) AS kobiety,
COUNT(NULLIF(gender, 'F')) AS mężczyźni
FROM customers
WHERE city IS NOT NULL
GROUP BY 1
ORDER BY 2 DESC
LIMIT 10
"""
```

Dla każdego miasta ta kwerenda zlicza klientów z podziałem na płeć. Pomija przy tym klientów, dla których brakuje informacji o mieście, i agreguje dane o klientach według pierwszej kolumny (miasto). Oprócz tego sortuje malejąco (od największych wartości do najmniejszych) dane według drugiej kolumny (liczba klientów). Następnie ogranicza dane wyjściowe do pierwszych 10, czyli do 10 miast o największej liczbie klientów.

3. Wczytaj wynik kwerendy do ramki danych pakietu pandas. W tym celu wpisz poniższe polecenie i wciśnij kombinację *Shift+Enter*:

```
top_cities_data = pd.read_sql_query(query, engine)
```

Wyświetl dane ze zmiennej `top_cities_data`. W tym celu wpisz ją w nowej komórce i wciśnij kombinację *Shift+Enter*. Podobnie jak w interpreterze Pythona, tak i tu wpisanie zmiennej lub wyrażenia skutkuje wyświetleniem wartości wyjściowej (rysunek 4.19).

Out[14]:

	city	number_of_customers	kobiety	mężczyźni
0	Washington	1447	734	713
1	Houston	904	446	458
2	New York City	731	369	362
3	El Paso	713	369	344
4	Dallas	607	309	298
5	Atlanta	571	292	279
6	Sacramento	506	244	262
7	Los Angeles	466	241	225
8	San Antonio	426	207	219
9	Miami	426	195	231

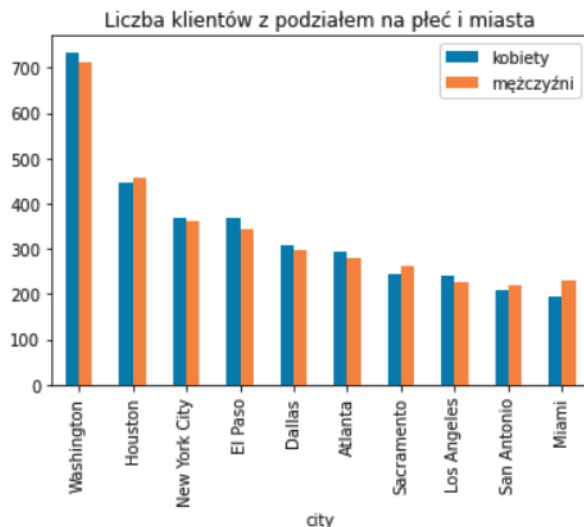
Rysunek 4.19. Zapisywanie wyniku kwerendy w ramce danych pakietu pandas

Zauważ, że pakiet pandas domyślnie numeruje wiersze, tworząc indeks.

4. Teraz narysuj wykres z liczbą mężczyzn i kobiet będących klientami w 10 pierwszych miastach. Aby wyświetlić osobno statystyki dla każdego miasta, użyj prostego wykresu słupkowego:

```
ax = top_cities_data.plot.bar('city', y=['kobiety', 'mężczyźni'],\
    title='Liczba klientów z podziałem na płeć i miasta')
```

Rysunek 4.20 przedstawia zrzut z wynikowymi danymi wyjściowymi w notatniku.



Rysunek 4.20. Wizualizowanie danych w notatniku Jupytera

Te wyniki pokazują, że nie ma istotnej różnicy w płci klientów w miastach, w których firma rozważa ekspansję.

Kod źródłowy z tego fragmentu znajdziesz na stronie <https://packt.live/3dWFw6b>.

W tym ćwiczeniu wczytaliśmy programowo dane z bazy i wykonaliśmy wizualizację wynikowych danych. Teraz zobaczysz, jak zapisywać dane (na przykład wyniki analiz statystycznych) w bazie.

Zapisywanie danych w bazie za pomocą Pythona

Często się zdarza, że za pomocą Pythona programista chce zapisywać dane w bazie. Na szczęście pakiet pandas i narzędzie SQLAlchemy sprawiają, że jest to stosunkowo łatwe zadanie.

Jeśli dane są zapisane w ramce danych pakietu pandas, można je zapisać w bazie za pomocą funkcji `to_sql(...)` z tego pakietu. Ta funkcja wymaga dwóch parametrów — nazwy docelowej tabeli i połączenia. Najlepsze jest to, że tworzy ona też docelową tabelę, wnioskując typy kolumn na podstawie typów danych z ramki.

Możesz sprawdzić ten mechanizm, używając utworzonej wcześniej ramki danych `top_cities_data`. Użyj poniższego polecenia `to_sql(...)` w istniejącym notatniku Jupytera:

```
top_cities_data.to_sql('top_cities_data', engine,\
    index=False, if_exists='replace')
```

Oprócz dwóch wymaganych parametrów podane są też dwa parametry opcjonalne. Parametr `index` określa, czy indeks ma zostać zachowany jako tabela w bazie danych. Wartość `False` oznacza, że indeks zostanie pominięty. Parametr `if_exists` pozwala określić, co robić w sytuacji, jeśli w bazie znajduje się już tabela z danymi o tej samej nazwie. Tu pożądane jest usunięcie tabeli i zastąpienie jej nowymi danymi, dlatego używana jest opcja `'replace'`. Zwykle powinieneś zachować ostrożność, gdy korzystasz z opcji `'replace'`, ponieważ może to skutkować przypadkową utratą istniejących danych.

Teraz możesz pobrać te dane za pomocą dowolnego klienta bazodanowego, w tym `psql`. Oto kwerenda dotycząca nowej tabeli w bazie danych:

```
select * from top_cities_data LIMIT 10;
```

Na rysunku 4.21 pokazane są dane wyjściowe tego kodu.

city	number_of_customers	kobiety	mężczyźni
Washington	1447	734	713
Houston	904	446	458
New York City	731	369	362
El Paso	713	369	344
Dallas	607	309	298
Atlanta	571	292	279
Sacramento	506	244	262
Los Angeles	466	241	225
San Antonio	426	207	219
Miami	426	195	231
(10 rows)			

Rysunek 4.21. Dane utworzone w Pythonie i zaimportowane do bazy danych

Choć ten mechanizm jest prosty i działa zgodnie z oczekiwaniami, wykorzystuje instrukcje `INSERT` do wstawiania danych do bazy. Dla małej tabeli z 10 wierszami jest to akceptowalne, jednak w przypadku większych tabel polecenie `\copy` z `psql` działa znacznie szybciej.

Zwiększanie szybkości zapisu w Pythonie za pomocą polecenia `COPY`

Możesz użyć polecenia `COPY` razem z Pythonem, `SQLAlchemy` i `pandas`, aby uzyskać tę samą szybkość co przy korzystaniu z instrukcji `\copy` w `psql`. Dana jest następująca funkcja:

```
import csv
from io import StringIO

def psql_insert_copy(table, conn, keys, data_iter):
    # Pobiera połączenie DBAPI, które udostępnia cursor
    dbapi_conn = conn.connection
```

```

with dbapi_conn.cursor() as cur:
    s_buf = StringIO()
    writer = csv.writer(s_buf)
    writer.writerows(data_iter)
    s_buf.seek(0)

    columns = ', '.join("{}".format(k) for k in keys)
    if table.schema:
        table_name = '{}.{}'.format(table.schema, table.name)
    else:
        table_name = table.name

    sql = 'COPY {} ({{}}) FROM STDIN WITH CSV'.format(table_name, columns)
    cur.copy_expert(sql=sql, file=s_buf)

```

Następnie możesz wykorzystać parametr `method` w wywołaniu funkcji `to_sql`:

```

top_cities_data.to_sql('top_cities_data', engine,
    index=False, if_exists='replace',
    method=psql_insert_copy)

```

Zdefiniowaną tu funkcję `psql_insert_copy` można zastosować bez żadnych zmian przy importowaniu danych z pakietu `pandas` do baz PostgreSQL. Oto omówienie działania tego kodu:

1. Po zaimportowaniu potrzebnych elementów najpierw definiowana jest funkcja. W tym celu należy podać słowo kluczowe `def`, nazwę funkcji (`psql_insert_copy`) oraz jej parametry (`table`, `conn`, `keys` i `data_iter`).
2. Dalej nawiązywane jest połączenie (`dbapi_conn`) i tworzony jest kursor (`cur`), który można wykorzystać do wykonywania instrukcji.
3. Następnie wszystkie dane z wierszy (reprezentowane za pomocą parametru `data_iter`) są zapisywane w buforze tekstowym (`s_buf`), który ma format taki jak plik CSV, ale znajduje się w pamięci, a nie w pliku na dysku twardym.
4. Dalej definiowane są nazwy kolumn (`columns`) i nazwa tabeli (`table_name`).
5. W ostatnim kroku wykonywana jest instrukcja `COPY`, która strumieniowo przekazuje dane w formacie CSV ze standardowego wejścia (`STDIN`).

Mimo że bezpośredni odczyt i zapis danych w bazach lub importowanie danych z bazy do pliku to przydatne operacje, czasem trzeba wczytać plik do Pythona na potrzeby wstępnego przetwarzania przed przesłaniem danych do bazy. Dzieje się tak na przykład wtedy, gdy plik zawiera błędy i nie można go bezpośrednio wczytać do bazy lub gdy plik wymaga dołączenia dodatkowych danych analitycznych. W takich scenariuszach można wykorzystać Pythona do odczytu i zapisu plików CSV.

Odczyt i zapis plików CSV w Pythonie

Omówiliśmy już użycie Pythona razem z SQL-em. Jednak Python udostępnia też inne techniki przetwarzania danych.

Oprócz odczytu i zapisu danych w bazach można wykorzystać Pythona do odczytu i zapisu danych w lokalnym systemie plików. Polecenia służące do odczytu i zapisu plików CSV za pomocą pakietu pandas są bardzo podobne do instrukcji używanych do odczytu i zapisu danych w bazach:

- Jeśli chodzi o zapis, metoda `pandas.DataFrame.to_csv(file_path, index=False)` zapisuje ramkę danych w lokalnym systemie plików zgodnie ze ścieżką `file_path`. Ramka danych (`DataFrame`) to struktura z pakietu pandas służąca do tymczasowego przechowywania danych. Metoda `to_csv()` ramki danych ma następujące parametry: `file_path` (jest to łańcuch znaków reprezentujący ścieżkę do pliku wyjściowego w formacie odpowiednim dla systemu operacyjnego) i `index` (jeśli jego wartość to `true`, w danych wyjściowych zapisywane są numery wierszy).
- Do odczytu służy metoda `pandas.read_csv(file_path, dtype={})`. Zwraca ona ramkę z danymi z pliku CSV, którego lokalizację określa ścieżka `file_path`.

Gdy wczytujesz plik CSV, pakiet pandas wnioskuje właściwy typ danych na podstawie wartości z tego pliku. Na przykład jeśli kolumna zawiera tylko liczby całkowite, tworzona jest kolumna typu `int64`.

Pakiet pandas potrafi też wywnioskować, czy kolumna zawiera liczby zmiennoprzecinkowe, znaczniki czasu lub łańcuchy znaków. Ponadto pakiet potrafi określić, czy w pliku występują nagłówki (mechanizm ten zwykle działa poprawnie). Jeśli któraś z kolumn nie jest poprawnie wczytywana (na przykład pięciocyfrowy kod pocztowy może zostać wczytany jako liczba całkowita, co powoduje pominięcie początkowego zera — „07123” stanie się wtedy wartością „7123”), można bezpośrednio określić jej typ, używając parametru `dtype`. Przykładowo jeśli masz w zbiorze danych kolumnę `kod_pocztowy`, za pomocą wyrażenia `dtype={'kod_pocztowy': str}` możesz określić, że zawiera ona łańcuchy znaków.

Plik CSV może być sformatowany na wiele sposobów. Choć pakiet pandas przeważnie potrafi wywnioskować nagłówki i typy danych, dostępnych jest wiele parametrów, które pozwalają dostosować proces odczytu i zapisu plików CSV do sytuacji.

Wykorzystując zbiór danych `top_cities_data` z notatnika, możesz przetestować opisany mechanizm:

```
top_cities_data.to_csv('top_cities_analysis.csv', index=False)
my_data = pd.read_csv('top_cities_analysis.csv')
my_data
```

Teraz zmienna `my_data` zawiera dane zapisane w pliku CSV i później wczytane ponownie. Tu nie trzeba podawać opcjonalnego parametru `dtype`, ponieważ pakiet pandas poprawnie wnioskuje typy kolumn. Powinieneś zobaczyć identyczne dane jak w `top_cities_data` (rysunek 4.22).

W tym przykładzie wczytaliśmy i zapisaliśmy plik CSV za pomocą Pythona, używając danych pobranych z bazy. Nabyte umiejętności pozwolą Ci importować i eksportować dane między plikiem a bazą danych, między Pythonem a bazą danych i między Pythonem a plikiem.

	A	B	C	D
1	city	number_of_customers	kobiety	mężczyźni
2	Washington	1447	734	713
3	Houston	904	446	458
4	New York City	731	369	362
5	El Paso	713	369	344
6	Dallas	607	309	298
7	Atlanta	571	292	279
8	Sacramento	506	244	262
9	Los Angeles	466	241	225
10	San Antonio	426	207	219
11	Miami	426	195	231

Rysunek 4.22. Sprawdzanie, czy za pomocą pakietu pandas można zapisywać i wczytywać pliki CSV

Najlepsze praktyki z obszaru importowania i eksportowania danych

Znasz już kilka różnych metod odczytu i zapisu danych w komputerze i bazie. Każda metoda ma określone zastosowania. Przy wyborze jednej z nich zwykle należy uwzględnić dwa podstawowe czynniki:

- Staraj się używać bazy za pomocą tego samego narzędzia, z którego korzystasz do analizowania danych. Dodawanie kolejnych kroków przy pobieraniu danych z bazy do narzędzia analitycznego zwiększa liczbę miejsc, w których mogą się pojawić nowe błędy. Jeśli nie możesz używać bazy za pomocą tego samego narzędzia, z którego korzystasz do przetwarzania danych, korzystaj z `psql` do odczytu i zapisu plików CSV w bazie.
- Gdy zapisujesz dane, możesz zaoszczędzić czas, korzystając z poleceń `COPY` i `\copy`.

Pomijanie podawania hasła

Dobrym pomysłem jest przygotowanie pliku `.pgpass`. Określa on parametry wykorzystywane do nawiązywania połączenia z bazą, w tym hasło. Wszystkie używane w tym rozdziale metody programowego dostępu do baz danych (za pomocą `psql`, `R` i `Pythona`) umożliwiają pominięcie parametru `password`, jeśli plik `.pgpass` zawiera hasło dla odpowiedniej kombinacji nazwy hosta, bazy danych i nazwy użytkownika. Pozwala to nie tylko przyspieszyć pracę, ale też zwiększa bezpieczeństwo bazy, ponieważ możesz swobodnie udostępniać kod bez obaw o zapisane w nim hasła. W systemach uniksowych i w `macOS` możesz utworzyć plik `.pgpass` w katalogu głównym. W systemie `Windows` ścieżka do tego pliku to `%APPDATA%\postgresql\pgpass.conf`. Plik powinien zawierać jeden wiersz dla każdego połączenia z bazą, jakie chcesz zapisać. Oto format tych wierszy (dostosowany do używanych parametrów bazy danych):

```
nazwa_hosta:port:baza_danych:nazwa_uzytkownika:haslo
```

Użytkownicy systemów uniksowych i macOS muszą zmienić uprawnienia do tego pliku, korzystając z następującej instrukcji w wierszu poleceń (w terminalu):

```
chmod 0600 ~/.pgpass
```

W systemie Windows przyjmuje się, że użytkownik odpowiednio skonfigurował uprawnienia do pliku, aby inni użytkownicy nie mieli do niego dostępu. Po utworzeniu pliku możesz sprawdzić, czy działa on poprawnie, wywołując poniższą instrukcję `psql` w terminalu:

```
psql -h Twój_host -p 5432 -d Twoja_baza_danych -U Twoja_nazwa_użytkownika
```

Jeśli plik `.pgpass` został poprawnie utworzony, prośba o podanie hasła się nie pojawi.

W ten sposób możesz nawiązać połączenie z bazą danych bez wprowadzania hasła. Przyspiesza to pracę i ogranicza ryzyko przypadkowego ujawnienia hasła.

W następnym zadaniu wykorzystasz całą wiedzę zdobytą w tym rozdziale, aby zobaczyć, jak wykrywać trendy sprzedażowe. W tym celu zaimportujesz nowe zbiory danych.

Zadanie 4.01 — używanie zewnętrznego zbioru danych do wykrywania trendów sprzedażowych

W tym zadaniu użyjesz danych ze spisu ludności Stanów Zjednoczonych na temat korzystania z transportu publicznego w zależności od kodu pocztowego. Zobaczysz, czy popularność transportu publicznego jest skorelowana ze sprzedażą produktów firmy ZoomZoom w danej lokalizacji. Pozwoli Ci to przećwiczyć następujące umiejętności:

- importowanie danych z bazy i ich eksportowanie do bazy;
- programowe komunikowanie się z bazą danych (na przykład za pomocą Pythona w połączeniu z SQLAlchemy i pandas).

Zanim zaczniesz, pobierz z serwisu GitHub (<https://packt.live/37ruT94>) statystyki dotyczące transportu publicznego w lokalizacjach o określonym kodzie pocztowym.

Pobrany zbiór danych zawiera trzy kolumny:

- `zip_code` — pięciocyfrowy kod pocztowy służący do identyfikowania różnych rejonów USA.
- `public_transportation_pct` — odsetek mieszkańców w rejonie o danym kodzie pocztowym, którzy korzystają z transportu publicznego do dojazdów do pracy.
- `public_transportation_population` — liczba mieszkańców w rejonie o danym kodzie pocztowym, którzy korzystają z transportu publicznego do dojazdów do pracy.

Oto kroki niezbędne do wykonania tego zadania:

1. Skopiuj dane ze zbioru danych dotyczącego transportu publicznego do bazy klientów firmy ZoomZoom. W tym celu zaimportuj te dane do nowej tabeli w bazie firmy.
2. Znajdź maksymalną i minimalną wartość kolumny `public_transportation_pct` w tych danych. Wartości mniejsze niż 0 zapewne oznaczają brakujące dane.

3. Oblicz średnią wartość sprzedaży dla klientów mieszkających w rejonach o dużej populacji transportu publicznego (ponad 10%) i w rejonach o niskiej populacji transportu publicznego (wartości do 10% włącznie).
4. Wczytaj dane za pomocą pakietu `pandas` i wyświetl histogram z rozkładem. (Wskazówka: jeśli wczytasz dane do ramki danych `my_data` pakietu `pandas`, możesz wygenerować histogram za pomocą wywołania `my_data.plot.hist(y='public_transportation_pct')`).
5. Wypróbuj funkcję `to_sql` z pakietu `pandas` z parametrem `method=psql_insert_copy` i bez tego parametru. Jakie są różnice w szybkości działania kodu? (Wskazówka: w notatniku Jupytera możesz dodać przed poleceniem człon `%time`, aby zobaczyć, jak długo trwa wykonywanie kodu).
6. Pogrupuj klientów na podstawie populacji transportu publicznego w lokalizacjach o określonych kodach pocztowych; wartości zaokrąglaj do 10%. Sprawdź średnią liczbę transakcji przypadających na jednego klienta w poszczególnych grupach. Wyeksportuj te dane do Excela i utwórz wykres punktowy, aby lepiej zrozumieć zależności między populacją transportu publicznego a poziomem sprzedaży.
7. Na podstawie tych analiz ustal zalecenia dotyczące ekspansji dla zarządu firmy ZoomZoom.

Rozwiązanie tego zadania znajdziesz w „Dodatku”.

Podsumowanie

W tym rozdziale pokazaliśmy, jak komunikować się z bazą danych za pomocą innych narzędzi analitycznych w celu dalszych analiz i tworzenia wizualizacji. Choć SQL ma duże możliwości, zdarzają się analizy, które trzeba wykonywać w innych systemach. Aby rozwiązać ten problem, SQL umożliwia przekazywanie danych do bazy i ich pobieranie w celu wykonania niezbędnych zadań.

Najpierw zobaczyłeś, jak używać działającego w wierszu poleceń narzędzia `psql` do pobierania danych z bazy. Następnie nauczyłeś się używać polecenia `COPY` i specyficznego dla `psql` polecenia `\copy`; umożliwiając one masowe importowanie danych do baz i eksportowanie danych z baz. Dalej zapoznałeś się z programowym dostępem do baz za pomocą oprogramowania analitycznego takiego jak języki R i Python, a także przyjrzałeś się niektórym zaawansowanym możliwościom Pythona, w tym narzędziu `SQLAlchemy` i pakietowi `pandas`, które pozwoliły przygotować wizualizacje danych.

W następnym rozdziale przyjrzyj się strukturom danych, których można używać do reprezentowania złożonych relacji. Nauczysz się wyciągać wnioski z danych tekstowych, a także poznasz format `JSON` i tablice, dzięki czemu będziesz w pełni korzystać ze wszystkich dostępnych informacji.

Analityka z wykorzystaniem złożonych typów danych

W tym rozdziale wyjaśniamy, jak w pełni wykorzystać dane poprzez posłużenie się w analizach złożonymi i rzadziej stosowanymi typami danych. Choć zwykle dane kojarzą się z liczbami, w praktyce często mają inny format — mogą być tekstem, datami i czasem, szerokością i długością geograficzną itd. Oprócz tych specjalnych typów danych istnieją też typy zapewniające kontekst związany z sekwencjami lub relacjami. Dzięki lekturze tego rozdziału nauczysz się, jak używać SQL-a i technik analitycznych do wyciągania wniosków na podstawie tych innych typów danych.

Nauczysz się też przeprowadzać analizy opisowe szeregów czasowych z użyciem typu `datetime`. Ponadto wykorzystasz dane geoprzestrzenne do wykrywania relacji, wydobędziesz informacje ze złożonych typów danych (tablic oraz danych w formatach JSON i JSONB), a także przeprowadzisz analizę tekstu.

Wprowadzenie

W poprzednim rozdziale pokazaliśmy, jak importować dane i eksportować je do innych narzędzi analitycznych, aby wykorzystać narzędzia dostępne poza bazą danych. Często najłatwiej jest analizować liczby. Jednak w praktyce dane nierzadko mają inną postać — słów, lokalizacji, dat, a czasem złożonych struktur danych. W tym rozdziale przyjrzyj się tym formatom i zobaczysz, jak wykorzystać tego rodzaju dane w analizach.

Najpierw zapoznasz się z często używanymi typami kolumn — *latitude* (szerokość geograficzna) i *longitude* (długość geograficzna). Te typy danych pomagają zrozumieć dane z perspektywy czasowej i przestrzennej. Następnie przyjrzyj się złożonym typom danych takim jak tablice i dane w formacie JSON. Dowiesz się, jak pobierać z nich punkty danych. Te struktury danych często są używane dla danych alternatywnych lub danych z dziennika (na przykład z dzienników witryn). Na zakończenie zobaczysz, jak ustalić znaczenie tekstu z bazy danych i wykorzystać dane tekstowe do wyciągania wniosków.

Dzięki lekturze tego rozdziału rozwiniesz umiejętności analityczne, co pozwoli Ci wykorzystać niemal dowolny z dostępnych typów danych.

Wykorzystywanie typów danych z datami i czasem do analiz

Wszyscy wiedzą, czym są daty i czas, ale ludzie rzadko zastanawiają się nad reprezentacją tych miar. To prawda, są one przedstawiane za pomocą liczb, lecz nie jest to jedna liczba. Są to zestawy wartości — po jednej liczbie dla roku, miesiąca, dnia, godziny, minuty itd.

Możesz sobie jednak nie zdawać sprawy, że jest to złożona reprezentacja składająca się z kilku różnych komponentów. Na przykład znajomość minuty bez znajomości godziny jest bezużyteczna. Ponadto istnieją skomplikowane sposoby interakcji z datami i czasem — różne punkty w czasie można na przykład odejmować od siebie. Aktualny czas może też być reprezentowany w inny sposób w zależności od miejsca, w którym się znajdujesz.

Z powodu tych zawiłości praca z takimi danymi wymaga szczególnej uwagi. PostgreSQL, podobnie jak większość systemów bazodanowych, udostępnia specjalne typy danych, które pozwalają reprezentować takie wartości. Zacznij od zapoznania się z typem *date*.

Wprowadzenie do typu *date*

Daty można reprezentować za pomocą łańcuchów znaków (na przykład 1 stycznia 2000, który jednoznacznie reprezentuje określoną datę), ale stanowią specjalny typ tekstu, w którym reprezentują wartość ilościową i sekwencyjną. Możesz na przykład dodać tydzień do aktualnej daty. Data może mieć wiele cech, które mogą być przydatne w analizach — na przykład rok lub dzień tygodnia. Praca z datami jest także nieunikniona w trakcie analiz szeregów czasowych. Jest to jeden z najczęściej przeprowadzanych rodzajów analiz.

W standardzie SQL-a opisany jest typ danych *DATE*, a PostgreSQL udostępnia przydatne mechanizmy do pracy z tym typem danych. Przede wszystkim możesz skonfigurować bazę tak, aby wyświetlała daty w najbardziej odpowiadającym Ci formacie. W PostgreSQL służy do tego parametr *DateStyle*. W celu sprawdzenia aktualnych ustawień użyj następującego polecenia:

```
SHOW DateStyle;
```

Oto dane wyjściowe tej kwerendy:

```
DateStyle
-----
ISO, DMY
(1 row)
```

Pierwszy parametr wskazuje format danych wyjściowych określony przez Międzynarodową Organizację Normalizacyjną (ISO), w którym data jest wyświetlana jako *rok, miesiąc, dzień*. Drugi parametr określa kolejność komponentów w danych wejściowych i wyjściowych (na przykład *miesiąc, dzień, rok* lub *dzień, miesiąc, rok*). Aby skonfigurować dane wyjściowe bazy danych, możesz wywołać następującą instrukcję:

```
SET DateStyle='ISO, MDY';
```

Na przykład jeśli chcesz używać europejskiego formatu *dzień, miesiąc, rok*, przypisz do `DateStyle` wartość `'GERMAN, DMY'`. W tym rozdziale do wyświetlania danych używany jest format ISO (*rok, miesiąc, dzień*), a dla danych wejściowych format *miesiąc, dzień, rok*. Format możesz skonfigurować, używając opisaną wcześniej instrukcji.

Zacznij od przetestowania formatu DATE:

```
# SELECT '1/8/1999'::DATE;
```

Oto dane wyjściowe tej kwerendy:

```
date
-----
1999-01-08
(1 row)
```

Widać tu, że po wprowadzeniu łańcucha znaków `'1/8/1999'` w formacie *miesiąc, dzień, rok* PostgreSQL traktuje tę datę jako 8 stycznia 1999 roku (a nie jako 1 sierpnia 1999 roku). Data jest wyświetlana w zdefiniowanym wcześniej formacie ISO (*RRRR-MM-DD*).

Możesz też użyć pokazanych niżej formatów, z dywizami i kropkami rozdzielającymi komponenty daty, a efekt będzie taki sam:

```
# SELECT '1-8-1999'::DATE;
```

Oto dane wyjściowe tej kwerendy:

```
date
-----
1999-01-08
(1 row)
```

W tej kwerendzie wyświetlana data jest zapisana w innym formacie:

```
# SELECT '1.8.1999'::DATE;
```

Oto dane wyjściowe:

```
date
-----
1999-01-08
(1 row)
```

W PostgreSQL oprócz wyświetlania dat podawanych jako łańcuchy znaków można w bardzo prosty sposób wyświetlić aktualną datę z użyciem słów kluczowych `current_date`:

```
# SELECT current_date;
```

Oto dane wyjściowe tej kwerendy:

```
current_date
-----
2020-02-04
(1 row)
```

Oprócz typu danych `DATE` w standardzie SQL-a zdefiniowany jest typ danych `TIMESTAMP`. Reprezentuje on datę i czas z dokładnością do mikrosekund.

Aktualny czas można wyświetlić, korzystając z funkcji `now()`. Możesz też podać strefę czasową za pomocą wyrażenia `AT TIME ZONE 'UTC'`. Oto przykład zastosowania funkcji `now()` ze strefą czasową `EST` (czas wschodni):

```
# SELECT now() AT TIME ZONE 'EST';
```

Oto dane wyjściowe tej kwerendy:

```
timezone
-----
2019-04-28 13:47:44.472096
(1 row)
```

Typu danych `timestamp` można też używać bez podawania strefy czasowej. Aby pobrać aktualny czas bez podawania strefy czasowej, zastosuj funkcję `now()`:

```
# SELECT now();
```

Oto dane wyjściowe tej kwerendy:

```
now
-----
2019-04-28 19:16:31.670096+00
(1 row)
```

Zaleca się, aby używać znaczników czasu z podaną strefą czasową. Jeśli pominiesz strefę czasową, wartość znacznika czasu będzie niepewna. Czas może być reprezentowany na przykład zgodnie ze strefą czasową, w której znajduje się siedziba firmy, zgodnie ze strefą UTC (ang. *Universal Time Coordinated*) lub zgodnie ze strefą czasową klienta.

Typy danych DATE i TIMESTAMP są przydatne nie tylko dlatego, że wyświetlają daty w czytelnym formacie, ale też dlatego, że przechowują je za pomocą mniejszej liczby bajtów niż analogiczna reprezentacja tekstowa (wartość typu DATE wymaga tylko 4 bajtów, natomiast reprezentacja tekstowa wymaga 8 bajtów przy ośmioznakowym zapisie takim jak '20160101'). Ponadto PostgreSQL udostępnia specjalne funkcje do operowania danymi i ich modyfikowania, co jest bardzo przydatne przy analizie danych.

Przekształcanie typów danych

Często datę trzeba rozdzielić na komponenty. Na przykład w analizie danych miesięcznych ważny może być tylko rok i miesiąc (bez dnia). Do uzyskania potrzebnych danych możesz użyć wywołania `EXTRACT(komponent FROM data)`. Oto przykład:

```
# SELECT current_date,
      EXTRACT(year FROM current_date) AS year,
      EXTRACT(month FROM current_date) AS month,
      EXTRACT(day FROM current_date) AS day;
```

Dane wyjściowe tego kodu wyglądają tak:

```
current_date | year | month | day
-----+-----+-----+-----
2019-04-28  | 2019 | 4     | 28
(1 row)
```

Możesz też podawać skrócone nazwy komponentów (y, mon i d), a PostgreSQL zrozumie, jakich danych potrzebujesz:

```
# SELECT current_date,
      EXTRACT(y FROM current_date) AS year,
      EXTRACT(mon FROM current_date) AS month,
      EXTRACT(d FROM current_date) AS day;
```

Oto dane wyjściowe:

```
current_date | year | month | day
-----+-----+-----+-----
2019-04-28  | 2019 | 4     | 28
(1 row)
```

Oprócz roku, miesiąca i dnia niekiedy potrzebne są dodatkowe komponenty, na przykład dzień tygodnia, tydzień roku lub kwartał. Te informacje możesz pobrać w następujący sposób:

```
# SELECT current_date,
      EXTRACT(dow FROM current_date) AS day_of_week,
      EXTRACT(week FROM current_date) AS week_of_year,
      EXTRACT(quarter FROM current_date) AS quarter;
```

Dane wyjściowe wyglądają tak:

```
current_date | day_of_week | week | quarter
-----+-----+-----+-----
2019-04-28  |            | 0    | 2
(1 row)
```


Zauważ, że wywołanie `EXTRACT` zawsze zwraca liczbę. Tu dzień tygodnia (`dow`) przyjmuje wartości od 0 (niedziela) do 6 (sobota). Zamiast `dow` można też użyć nazwy `isodow` — wtedy dni tygodnia przyjmują wartości od 1 (poniedziałek) do 7 (niedziela).

Oprócz pobierania komponentów możesz też chcieć przyciąć datę lub czas. Załóżmy, że chcesz wyświetlić tylko rok i miesiąc daty. Należy wtedy usunąć dzień i czas. Można w tym celu użyć funkcji `DATE_TRUNC()`:

```
# SELECT NOW(), DATE_TRUNC('month', NOW());
```

Oto dane wyjściowe:

now	date_trunc
2019-04-28 19:40:08.691618+00	2019-04-01 00:00:00+00

(1 row)

Zauważ, że funkcja `DATE_TRUNC(...)` nie zaokrągla wartości, lecz zwraca największą zaokrągloną wartość mniejszą lub równą względem podanej daty.

Funkcja `DATE_TRUNC(...)` działa podobnie jak funkcja „podłoga” (ang. *floor*) w matematyce, która zwraca największą liczbę całkowitą mniejszą lub równą względem wartości wejściowej (na przykład 5 dla 5,7).

Funkcja `DATE_TRUNC(...)` jest przydatna przede wszystkim w instrukcjach `GROUP BY`. Możesz na przykład użyć jej do pogrupowania sprzedaży według kwartałów i uzyskać łączną sprzedaż kwartalną:

```
SELECT DATE_TRUNC('quarter', NOW()) AS quarter,
       SUM(sales_amount) AS total_quarterly_sales
FROM sales
GROUP BY 1
ORDER BY 1 DESC;
```

Funkcja `DATE_TRUNC(...)` wymaga łańcucha znaków reprezentującego pole, które chcesz przyciąć, natomiast funkcja `EXTRACT(...)` przyjmuje reprezentację tekstową (z cudzysłowami) lub nazwę pola (bez cudzysłowów).

Przedziały

Obok reprezentowania dat można też zapisywać stałe przedziały czasu, używając typu danych `INTERVAL`. Jest to przydatne, jeśli chcesz przeanalizować czas, jaki coś zajmuje — na przykład ustalić, ile czasu potrzebuje klient na dokonanie zakupu.

Oto przykład:

```
# SELECT INTERVAL '5 days';
```

A oto dane wyjściowe kodu:

```
interval
-----
5 days
(1 row)
```

Przedziały przydają się do odejmowania znaczników czasu:

```
# SELECT TIMESTAMP '2016-03-01 00:00:00' - TIMESTAMP '2016-02-01
00:00:00' AS days_in_feb;
```

To dane wyjściowe tego kodu:

```
days_in_feb
-----
29 days
(1 row)
```

Przedziały umożliwiają też dodanie określonej liczby dni do znacznika czasu:

```
# SELECT TIMESTAMP '2016-03-01 00:00:00' + INTERVAL '7 days' AS new_date;
```

Dane wyjściowe tego kodu wyglądają tak:

```
new_date
-----
2016-03-08 00:00:00
(1 row)
```

Choć przedziały zapewniają precyzyjną metodę wykonywania obliczeń na znacznikach czasu, podobny efekt można uzyskać, używając typu danych DATE i liczb całkowitych. W poniższym przykładzie do daty dodawana jest liczba całkowita 7, aby obliczyć nową datę:

```
# SELECT DATE '2016-03-01' + 7 AS new_date;
```

Oto dane wyjściowe tego kodu:

```
new_date
-----
2016-03-08
(1 row)
```

Można też odjąć dwie daty od siebie i otrzymać wynik w postaci liczby całkowitej:

```
# SELECT DATE '2016-03-01' - DATE '2016-02-01' AS days_in_feb;
```

Oto dane wyjściowe:

```
days_in_feb
-----
29
(1 row)
```

Chociaż typ danych DATE jest łatwy w użyciu, znaczniki czasu ze **strefą czasową** zapewniają większą precyzję. Jeśli chcesz, aby pole z datą i czasem precyzyjnie określało czas wystąpienia zdarzenia, zastosuj znacznik czasu ze strefą czasową. W przeciwnym razie możesz użyć pola typu DATE.

Cały kod ćwiczeń i zadań z tego rozdziału znajdziesz w serwisie GitHub na stronie <https://packt.live/2B3doiY>.

Ćwiczenie 5.01 — analiza danych z szeregów czasowych

W tym ćwiczeniu wykonasz proste analizy danych z szeregów czasowych, aby zrozumieć efekty intensyfikacji działań firmy ZoomZoom w celu zwiększenia sprzedaży w 2018 roku.

Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Najpierw przyjrzyj się wysokości sprzedaży miesięcznej. W tym celu możesz użyć kwerendy agregującej z metodą `DATE_TRUNC`:

```
SELECT
    DATE_TRUNC('month', sales_transaction_date)
        AS month_date,
    COUNT(1) AS number_of_sales
FROM sales
WHERE EXTRACT(year FROM sales_transaction_date) = 2018
GROUP BY 1
ORDER BY 1;
```

Po wykonaniu tego kodu w SQL-u otrzymasz wynik widoczny na rysunku 5.1.

month_date	number_of_sales
2018-01-01 00:00:00	504
2018-02-01 00:00:00	487
2018-03-01 00:00:00	624
2018-04-01 00:00:00	755
2018-05-01 00:00:00	945
2018-06-01 00:00:00	993
2018-07-01 00:00:00	1119
2018-08-01 00:00:00	1046
2018-09-01 00:00:00	867
2018-10-01 00:00:00	780
2018-11-01 00:00:00	801
2018-12-01 00:00:00	596
(12 rows)	

Rysunek 5.1. Sprzedaż miesięczna

2. Porównaj wynik z liczbą nowych klientów pozyskanych w każdym miesiącu:

```
SELECT
    DATE_TRUNC('month', date_added)
        AS month_date,
    COUNT(1) AS number_of_new_customers
FROM customers
WHERE EXTRACT(year FROM date_added) = 2018
GROUP BY 1
ORDER BY 1;
```

Rysunek 5.2 przedstawia dane wyjściowe tej kwerendy.

month_date	number_of_new_customers
2018-01-01 00:00:00	430
2018-02-01 00:00:00	415
2018-03-01 00:00:00	450
2018-04-01 00:00:00	423
2018-05-01 00:00:00	454
2018-06-01 00:00:00	456
2018-07-01 00:00:00	478
2018-08-01 00:00:00	448
2018-09-01 00:00:00	440
2018-10-01 00:00:00	464
2018-11-01 00:00:00	460
2018-12-01 00:00:00	473
(12 rows)	

Rysunek 5.2. Liczba rejestracji nowych klientów w poszczególnych miesiącach

Można z tego wydedukować, że klienci nie są wprowadzani do bazy danych dopiero w momencie dokonania zakupu, lecz rejestrują się wcześniej. Napływ nowych potencjalnych klientów jest w miarę stały — w każdym miesiącu rejestruje się ok. 400–500 nowych osób. Jednak liczba sprzedaży (z kwerendy z kroku 1.) znacznie się zmienia. W lipcu liczba transakcji (1119) była 2,3 raza wyższa niż liczba nowych klientów (478).

Kod źródłowy z tego fragmentu jest dostępny na stronie <https://packt.live/30ArB1y>.

To ćwiczenie pozwoliło stwierdzić, że napływ nowych klientów rejestrujących się w bazie jest stały, ale liczba transakcji znacznie się zmienia z miesiąca na miesiąc.

Przeprowadzanie analiz geoprzestrzennych w PostgreSQL

Oprócz analizy danych z szeregów czasowych w celu lepszego zrozumienia trendów można też wykorzystać informacje geoprzestrzenne (miasto, kraj, długość i szerokość geograficzna), by lepiej zrozumieć klientów. Na przykład agencje rządowe wykorzystują takie analizy, aby dowiedzieć się więcej o różnicach ekonomicznych między regionami, a w systemach zarządzania wspólnymi przejazdami samochodowymi dane geoprzestrzenne pozwalają znaleźć kierowcę znajdującego się najbliżej danego klienta.

Lokalizację geoprzestrzenną można reprezentować za pomocą współrzędnych (długości i szerokości geograficznej). Jest to podstawowy element pomagający rozpocząć analizy geoprzestrzenne.

Długość i szerokość geograficzna

Gdy myślisz o lokalizacji, na myśl przychodzi Ci adres — miasto, województwo, państwo lub kod pocztowy miejsca, które Cię interesuje. W kontekście analiz czasem jest to odpowiednie podejście. Możesz na przykład sprawdzić poziom sprzedaży z podziałem na miasta i uzyskać przydatne wyniki określające, w których miastach sprzedaż jest wysoka.

Jednak często trzeba ocenić relacje geoprzestrzenne w ujęciu liczbowym, aby poznać odległość między dwoma punktami lub relacje zależne od miejsca na mapie. W końcu jeśli ktoś mieszka blisko granicy między dwoma miastami, jest mało prawdopodobne, aby jego zachowania nagle się zmieniły, gdy przeprowadzi się do drugiego z tych miast.

Długość i szerokość geograficzna pozwalają analizować lokalizację na skali ciągłej. Umożliwia to analizowanie relacji liczbowych między lokalizacją a innymi wartościami (na przykład poziomem sprzedaży). Długość i szerokość geograficzna pozwalają też uwzględniać odległości między dwoma miejscami.

Szerokość geograficzna informuje o tym, jak daleko na północ lub na południe znajduje się dane miejsce. Punkt o szerokości geograficznej $+90^\circ$ znajduje się na biegunie północnym, a punkt o szerokości geograficznej -90° jest umieszczony na biegunie południowym. Na mapie linie o stałej szerokości geograficznej biegną na wschód i na zachód.

Długość geograficzna oznacza, jak daleko na wschód lub na zachód znajduje się określone miejsce. Na mapie linie o stałej długości geograficznej biegną w kierunku północnym i południowym. Greenwich w Anglii ma długość geograficzną 0° . Punkty można definiować jako znajdujące się na zachód ($-$) lub na wschód ($+$) względem tego miejsca, a zakres przyjmowanych wartości to od -180° na zachód do $+180^\circ$ na wschód. Te skrajne wartości oznaczają to samo — pionową linię biegnącą przez Pacyfik po drugiej stronie świata względem Greenwich.

Reprezentowanie długości i szerokości geograficznej w PostgreSQL

W PostgreSQL długość i szerokość geograficzną można przedstawić za pomocą dwóch liczb zmiennoprzecinkowych. W ten sposób długość i szerokość geograficzna są zapisane w tabeli `customers` firmy ZoomZoom:

```
SELECT
    latitude,
    longitude
FROM customers
LIMIT 10;
```

Rysunek 5.3 przedstawia dane wyjściowe tej kwerendy.

latitude	longitude
NULL	NULL
38.5814	-90.2625
30.6143	-87.2758
36.0986	-86.8219
25.5584	-80.4582
25.6364	-80.3187
28.5663	-81.2608
41.3087	-72.9271
38.8999	-94.832
31.6948	-106.3
(10 rows)	

Rysunek 5.3. Długość i szerokość geograficzna lokalizacji klientów firmy ZoomZoom

Widać tu, że wszystkie szerokości geograficzne są dodatnie, ponieważ Stany Zjednoczone znajdują się na północ od równika. Ponadto wszystkie długości geograficzne są ujemne, ponieważ Stany Zjednoczone są zlokalizowane na zachód od Greenwich w Anglii. Widać też, że dla niektórych klientów długość i szerokość geograficzna nie są podane; lokalizacja tych osób jest nieznana.

Choć omawiane wartości pozwalają ustalić dokładną lokalizację klientów, trudno jest wykorzystać te dane, ponieważ obliczanie odległości wymaga trygonometrii i przyjęcia upraszczających założeń, że Ziemia jest idealnie okrągła.

Na szczęście PostgreSQL udostępnia narzędzia pozwalające rozwiązać ten problem. Aby obliczać odległości w PostgreSQL, zainstaluj następujące pakiety:

```
CREATE EXTENSION cube;
CREATE EXTENSION earthdistance;
```

Te dwa rozszerzenia wystarczy zainstalować raz, uruchamiając dwa podane polecenia. Moduł earthdistance wymaga do działania modułu cube. Po zainstalowaniu modułu earthdistance możesz zdefiniować typ danych point:

```
SELECT
    point(longitude, latitude)
FROM customers
LIMIT 10;
```

Rysunek 5.4 przedstawia dane wyjściowe tej kwerendy.

W typie danych point najpierw zdefiniowana jest długość, a następnie szerokość geograficzna. Jest to niezgodne ze zwyczajową kolejnością — najpierw szerokość, potem długość. Kolejność z typu danych point wynika z tego, że długość geograficzna odpowiada punktom na osi x, szerokość zaś punktom na osi y, a w matematyce punkty na wykresie są zwykle opisywane za pomocą współrzędnych (x, y).

point
NULL
(-90.2625,38.5814)
(-87.2758,30.6143)
(-86.8219,36.0986)
(-80.4582,25.5584)
(-80.3187,25.6364)
(-81.2608,28.5663)
(-72.9271,41.3087)
(-94.832,38.8999)
(-106.3,31.6948)
(10 rows)

Rysunek 5.4. Długości i szerokości geograficzne lokalizacji klientów reprezentowane jako punkty w PostgreSQL

Moduł earthdistance umożliwia też obliczanie odległości między punktami w milach:

```
SELECT
    point(-90, 38) <@> point(-91, 37) AS distance_in_miles;
```

Oto dane wyjściowe tej kwerendy:

```
distance_in_miles
-----
      88.1949338379752
(1 row)
```

W tym przykładzie zdefiniowane są dwa punkty, (38° N, 90° W) i (37° N, 91° W). Możesz obliczyć odległość między nimi, używając operatora <@>. Zwraca on odległość w milach (tu punkty są oddalone o ok. 88,2 mili).

W następnym ćwiczeniu zobaczysz, jak wykorzystać odległości w praktycznym scenariuszu biznesowym.

Ćwiczenie 5.02 — analizy geoprzestrzenne

W tym ćwiczeniu zidentyfikujesz salon znajdujący się najbliżej klienta. Dział marketingu firmy ZoomZoom próbuje zwiększyć aktywność klientów, pomagając im znaleźć najbliższy salon. Ponadto zespół ds. produktów chce poznać średnią odległość od klientów do najbliższego salonu.

Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Utwórz tabelę z długością i szerokością geograficzną lokalizacji poszczególnych klientów:

```
CREATE TEMP TABLE customer_points AS (
    SELECT
        customer_id,
        point(longitude, latitude) AS lng_lat_point
    FROM customers
```

```
WHERE longitude IS NOT NULL
AND latitude IS NOT NULL
);
```

2. Następnie utwórz podobną tabelę z wszystkimi salonami:

```
CREATE TEMP TABLE dealership_points AS (
  SELECT
    dealership_id,
    point(longitude, latitude) AS lng_lat_point
  FROM dealerships
);
```

3. Teraz użyj złączenia krzyżowego tych tabel, aby obliczyć odległość między każdym klientem a każdym salonem (w milach):

```
CREATE TEMP TABLE customer_dealership_distance AS (
  SELECT
    customer_id,
    dealership_id,
    c.lng_lat_point <@> d.lng_lat_point AS distance
  FROM customer_points c
  CROSS JOIN dealership_points d
);
```

4. W ostatnim kroku dla każdego identyfikatora klienta należy wybrać najbliższy salon:

```
CREATE TEMP TABLE closest_dealerships AS (
  SELECT DISTINCT ON (customer_id)
    customer_id,
    dealership_id,
    distance
  FROM customer_dealership_distance
  ORDER BY customer_id, distance
);
```

Pamiętaj, że klauzula `DISTINCT ON` gwarantuje zwrócenie tylko jednego rekordu dla każdej unikatowej wartości z kolumny podanej w nawiasie. Tu zwracany jest tylko jeden rekord dla każdej wartości `customer_id`, a ponieważ dane są posortowane według odległości, zwracany jest rekord z najmniejszą odległością.

5. Gdy masz już dane potrzebne do zrealizowania prośby działu marketingu, możesz obliczyć średnią odległość między każdym klientem a najbliższym salonem:

```
SELECT
  AVG(distance) AS avg_dist,
  PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY distance)
  AS median_dist
FROM closest_dealerships;
```

Rysunek 5.5 przedstawia dane wyjściowe tej kwerendy.

avg_dist	median_dist
146.778266080342	91.2395829323349
(1 row)	

Rysunek 5.5. Średnia i mediana odległości między klientami a najbliższymi salonami

Wyniki pokazują, że średnia odległość w milach to 147 mil, a mediana wynosi 91 mil.

Kod źródłowy z tego fragmentu książki znajdziesz na stronie <https://packt.live/3fkQlIL>.

W tym ćwiczeniu znaleźliśmy najbliższy salon dla każdego klienta. Najpierw obliczyliśmy odległość od każdego klienta do wszystkich możliwych salonów, ustaliliśmy najbliższy salon oraz obliczyliśmy średnią i medianę odległości między klientami a najbliższymi salonami.

Stosowanie tablicowych typów danych w PostgreSQL

Typy z PostgreSQL omawiane do tej pory umożliwiają przechowywanie wielu różnych rodzajów danych, zdarza się jednak, że w tabeli należy zapisać serię wartości. Możliwe, że chcesz zachować listę produktów kupionych przez klienta lub numery identyfikacyjne pracowników określonego salonu. Na potrzeby takich sytuacji PostgreSQL udostępnia typ danych ARRAY. Umożliwia on zapis listy wartości.

Wprowadzenie do tablic

Tablice w PostgreSQL umożliwiają zapisanie wielu wartości w polu tabeli. Przyjrzyj się pierwszemu rekordowi tabeli customers:

customer_id	1
title	NULL
first_name	Arlena
last_name	Riveles
suffix	NULL
email	ariveles0@stumbleupon.com
gender	F
ip_address	98.36.172.246
phone	NULL
street_address	NULL
city	NULL
state	NULL
postal_code	NULL
latitude	NULL
longitude	NULL
date_added	2017-04-23 00:00:00

Każde pole zawiera tu jedną wartość (NULL też jest wartością). Jednak niektóre atrybuty mogą zawierać listę wartości o nieokreślonej długości. Załóżmy, że chcesz dodać pole `purchased_products`. Może ono zawierać zero lub więcej wartości. Przykładowo klient kupił skutery Lemon i Bat Limited Edition. Można zapisać te dane tak:

```
purchased_products | {Lemon,"Bat Limited Edition"}
```

Tablicę można zdefiniować na kilka sposobów. Możesz po prostu utworzyć tablicę za pomocą następującej instrukcji:

```
SELECT ARRAY['Lemon', 'Bat Limited Edition'] AS example_purchased_
products;
```

Oto dane wyjściowe tego kodu:

```
example_purchased_products
-----
{Lemon,"Bat Limited Edition"}
```

PostgreSQL wie, że wartości 'Lemon' i 'Bat Limited Edition' są typu danych text, dlatego w celu ich zapisania tworzy tablicę wartości typu text.

Choć można utworzyć tablicę wartości dowolnego typu danych, tablica przechowuje wartości tylko jednego zdefiniowanego typu. Dlatego nie możesz zapisać w niej liczby całkowitej, a następnie wartości tekstowej (prawdopodobnie spowoduje to błąd).

Do tworzenia tablic możesz też wykorzystać funkcję agregującą ARRAY_AGG. Tworzy ona tablicę z wszystkimi wartościami z grupy. Na przykład poniższa kwerenda agreguje pojazdy poszczególnych typów produktów:

```
SELECT product_type, ARRAY_AGG(DISTINCT model) AS models FROM products
GROUP BY 1;
```

Rysunek 5.6 przedstawia dane wyjściowe tej kwerendy.

product_type	models
automobile	{ "Model Chi", "Model Epsilon", "Model Gamma", "Model Sigma" }
scooter	{ Bat, "Bat Limited Edition", Blade, Lemon, "Lemon Limited Edition", "Lemon Zester" }
(2 rows)	

Rysunek 5.6. Dane wyjściowe funkcji ARRAY_AGG

Możesz też określić uporządkowanie elementów, dodając klauzulę ORDER BY do funkcji ARRAY_AGG. Oto przykład:

```
SELECT product_type, ARRAY_AGG(model ORDER BY year) AS models FROM
products GROUP BY 1;
```

Ta kwerenda zwraca dane wyjściowe widoczne na rysunku 5.7.

product_type	models
automobile	{ "Model Chi", "Model Sigma", "Model Epsilon", "Model Gamma", "Model Chi" }
scooter	{ Lemon, "Lemon Limited Edition", Lemon, Blade, Bat, "Bat Limited Edition", "Lemon Zester" }
(2 rows)	

Rysunek 5.7. Dane wyjściowe funkcji ARRAY_AGG z klauzulą ORDER BY

Możesz również odwrócić tę operację, używając funkcji UNNEST. Tworzy ona jeden wiersz dla każdej wartości z tablicy:

```
SELECT UNNEST(ARRAY[123, 456, 789]) AS example_ids;
```

Oto dane wyjściowe tej kwerendy:

```
example_ids
-----
123
456
789
(3 rows)
```

Inny sposób tworzenia tablic to podział łańcucha znaków za pomocą funkcji STRING_TO_ARRAY. Oto przykład:

```
SELECT STRING_TO_ARRAY('Cześć, co u Ciebie słychać?', ' ');
```

W tym przykładzie zdanie jest dzielone na podstawie drugiego łańcucha znaków (' '), a otrzymany wynik to:

```
string_to_array
-----
{"Cześć","co,u,Ciebie,słychać?}
(1 row)
```

Możesz też wykonać odwrotną operację i złączyć łańcuchy znaków z tablicy w jeden łańcuch:

```
SELECT ARRAY_TO_STRING(ARRAY['Lemon', 'Bat Limited Edition'], ', ') AS
example_purchased_products;
```

W tym przykładzie jeden łańcuch znaków jest łączony z drugim sekwencją ', ':

```
example_purchased_products
-----
Lemon, Bat Limited Edition
```

Istnieją też inne funkcje umożliwiające pracę z tablicami. Rysunek 5.8 przedstawia kilka dodatkowych funkcji tablicowych dostępnych w PostgreSQL.

Wykonywana operacja	Funkcja z PostgreSQL	Przykładowe dane wyjściowe
Złączanie dwóch tablic	array_cat(ARRAY[1, 2], ARRAY[3, 4]) lub ARRAY[1, 2] ARRAY[3, 4]	{1, 2, 3, 4}
Dołączanie wartości do tablicy	array_append(ARRAY[1, 2], 3) lub ARRAY[1, 2] 3	{1, 2, 3}
Sprawdzanie, czy wartość znajduje się w tablicy	3 = ANY(ARRAY[1, 2])	false
Sprawdzanie, czy dwie tablice mają wspólne elementy	ARRAY[1, 2, 3] && ARRAY[3, 4]	true
Sprawdzanie, czy tablica zawiera inną tablicę	ARRAY[1, 2, 3] @> ARRAY[2, 1]	true

Rysunek 5.8. Przykładowe inne funkcje tablicowe

Teraz zastosujesz te operatory i funkcje tablicowe do analizy sekwencji wiadomości marketingowych.

Ćwiczenie 5.03 — analizowanie sekwencji z użyciem tablic

W tym ćwiczeniu użyjesz tablic do przeanalizowania sekwencji. Załóżmy, że dział marketingu w firmie ZoomZoom chce zidentyfikować trzy najczęściej występujące sekwencje e-maili. Następnie pomożesz im lepiej zrozumieć różnice między tymi sekwencjami, sprawdzając, czy któreś z nich są nadzbiorami innych:

1. Najpierw utwórz tabelę reprezentującą sekwencję e-maili każdego klienta:

```
CREATE TEMP TABLE customer_email_sequences AS (
  SELECT
    customer_id,
    ARRAY_AGG(email_subject ORDER BY sent_date) AS email_sequence
  FROM emails
  GROUP BY 1
);
```

2. Następnie ustal trzy najczęściej występujące sekwencje e-maili:

```
CREATE TEMP TABLE top_email_sequences AS (
  SELECT
    email_sequence,
    COUNT(1) AS occurrences
  FROM customer_email_sequences
  GROUP BY 1
  ORDER BY 2 DESC
  LIMIT 3
);
SELECT email_sequence FROM top_email_sequences;
```

Rysunek 5.9 przedstawia dane wyjściowe tego kodu.

```
{ "The 2013 Lemon Scooter is Here", "Shocking Holiday Savings On Electric Scooters", "A Brand New Scooter...and Car", "We cut yo
u a deal: 20% off a Blade", "Zoom Zoom Black Friday Sale", "An Electric Car for a New Age", "Tis' the Season for Savings", "Like
a Bat out of Heaven", "25% off all EVs. It's a Christmas Miracle!", "We Really Outdid Ourselves this Year", "Black Friday. Gree
n Cars.", "Save the Planet with some Holiday Savings.", "A New Year, And Some New EVs" }
{ "Save the Planet with some Holiday Savings.", "A New Year, And Some New EVs" }
{ "Black Friday. Green Cars.", "Save the Planet with some Holiday Savings.", "A New Year, And Some New EVs" }
(3 rows)
```

Rysunek 5.9. Trzy pierwsze wyniki z listy sekwencji e-maili

3. Teraz sprawdź, które z tych tablic są nadzbiorami innych tablic. W tym celu warto ponumerować wiersze:

```
ALTER TABLE top_email_sequences ADD COLUMN id SERIAL PRIMARY KEY;
```

4. Następnie wykonaj złączenie krzyżowe tabeli z nią samą. Użyj operatora @>, aby sprawdzić, czy tablica zawierająca sekwencję e-maili zawiera inną taką tablicę:

```
SELECT
  super_email_seq.id AS superset_id,
  sub_email_seq.id AS subset_id
```

```
FROM top_email_sequences AS super_email_seq
CROSS JOIN top_email_sequences AS sub_email_seq
WHERE super_email_seq.email_sequence @> sub_email_seq.email_sequence
AND super_email_seq.id != sub_email_seq.id;
```

Dane wyjściowe tego kodu są pokazane na rysunku 5.10.

superset_id	subset_id
1	2
1	3
3	2

(3 rows)

Rysunek 5.10. Te wyniki pokazują, które z czołowych sekwencji e-maili są nadzbiorami innych sekwencji

Z wyników można wywnioskować, że pierwsza sekwencja zawiera drugą i trzecią, a trzecia jest nadzbiorem drugiej. Takie analizy są pomocne, gdy sprawdzasz, jakie sposoby kontaktu z klientem mogą skutkować sprzedażą (ta technika jest nazywana modelowaniem atrybucji).

Kod źródłowy z tego fragmentu znajdziesz na stronie <https://packt.live/2MRT1rK>.

Tablice świetnie nadają się do tworzenia list wartości i sekwencji, natomiast format JSON umożliwia zarządzanie danymi w postaci par klucz – wartość.

Stosowanie formatu JSON w PostgreSQL

Choć tablice przydają się do zapisywania list wartości w jednym polu, czasem struktury danych są bardziej złożone. Możesz na przykład chcieć zapisać w jednym polu kilka wartości różnych typów i używać kluczy w postaci etykiet zamiast zapisywać dane sekwencyjnie. Jest to często potrzebne w danych z dzienników, a także danych alternatywnych.

JSON (ang. *JavaScript Object Notation*) to otwarty standardowy format tekstowy do przechowywania danych o różnym poziomie złożoności. Można go używać do reprezentowania niemal dowolnych danych. Podobnie jak w tabelach baz danych występują nazwy kolumn, tak w danych w formacie JSON stosowane są klucze. Za pomocą formatu JSON możesz łatwo zapisać rekord z bazy customers, zapisując nazwy kolumn jako klucze i wartości z wierszy jako wartości. Funkcja `row_to_json` przekształca wiersze na format JSON:

```
SELECT row_to_json(c) FROM customers c limit 1;
```

Na rysunku 5.11 widoczne są dane wyjściowe tej kwerendy.

```
{
  "customer_id": 1,
  "title": null,
  "first_name": "Arlena",
  "last_name": "Riveles",
  "suffix": null,
  "email": "ariveles@stumbleupon.com",
  "gender": "F",
  "ip_address": "98.36.172.246",
  "phone": null,
  "street_address": null,
  "city": null,
  "state": null,
  "postal_code": null,
  "latitude": null,
  "longitude": null,
  "date_added": "2017-04-23T00:00:00"
}
```

Rysunek 5.11. Wiersz przekształcony na format JSON

Dane w tej postaci są dość nieczytelne, jednak możesz użyć opcji `pretty_bool` w funkcji `row_to_json`, aby wygenerować dane w bardziej czytelnej formie:

```
SELECT row_to_json(c, TRUE) FROM customers c limit 1;
```

Dane wyjściowe tej kwerendy są pokazane na rysunku 5.12.

```
row_to_json
-----
{"customer_id":1,
 "title":null,
 "first_name":"Arlena",
 "last_name":"Riveles",
 "suffix":null,
 "email":"ariveles@stumbleupon.com",
 "gender":"F",
 "ip_address":"98.36.172.246",
 "phone":null,
 "street_address":null,
 "city":null,
 "state":null,
 "postal_code":null,
 "latitude":null,
 "longitude":null,
 "date_added":"2017-04-23T00:00:00"}
(1 row)
```

Rysunek 5.12. Dane wyjściowe funkcji `row_to_json` w formacie JSON

Widać tu, że po zmianie wyglądu danych wyjściowych z kwerendy w formacie JSON funkcja `row_to_json` wyświetla prostą i czytelną tekstową reprezentację wiersza. Format JSON obejmuje klucze i wartości. W tym przykładzie kluczami są nazwy kolumn, a wartościami to wartości z wiersza. W formacie JSON wartości mogą być liczbowe (liczby całkowite i zmiennoprzecinkowe), logiczne (`true` lub `false`) i tekstowe (w cudzysłowach). Używana jest też wartość `null`.

Dane w formacie JSON mogą też obejmować zagnieżdżone struktury danych. Rozważ hipotetyczny scenariusz, w którym w tabeli należy umieścić także listę zakupionych produktów:

```
{
  "customer_id": 1,
  "example_purchased_products": ["Lemon", "Bat Limited Edition"]
}
```

Możesz nawet pójść o krok dalej:

```
{
  "customer_id": 7,
  "sales": [
    {
      "product_id": 7,
```

```

        "sales_amount": 599.99,
        "sales_transaction_date": "2019-04-25T04:00:30"
    },
    {
        "product_id": 1,
        "sales_amount": 399.99,
        "sales_transaction_date": "2011-08-08T08:55:56"
    },
    {
        "product_id": 6,
        "sales_amount": 65500,
        "sales_transaction_date": "2016-09-04T12:43:12"
    }
],
}

```

W tym przykładzie używany jest obiekt JSON z dwoma kluczami — `customer_id` i `sales`. Widać tu, że klucz `sales` prowadzi do tablicy wartości, przy czym każda wartość to następny obiekt JSON reprezentujący transakcję sprzedaży. Obiekty JSON zapisane w innym obiekcie JSON nazywa się obiektami zagnieżdżonymi. Tu kod reprezentuje wszystkie transakcje klienta za pomocą zagnieżdżonej tablicy zawierającej zagnieżdżone obiekty JSON z danymi o każdej sprzedaży.

Choć JSON jest uniwersalnym formatem przechowywania danych, jest niewygodny, ponieważ wszystkie dane są zapisywane jako jeden długi łańcuch znaków. Aby pobrać wartość powiązaną z kluczem, najpierw trzeba przetworzyć tekst, co jest dość kosztowne obliczeniowo. Jeśli liczba obiektów JSON jest niewielka, koszty obliczeniowe nie są poważnym problemem. Mogą jednak stać się kłopotliwe, jeżeli na przykład zechcesz pobrać obiekt JSON z kluczem `"customer_id": 7` spośród milionów innych obiektów JSON z bazy danych.

W następnym punkcie poznasz format JSONB. Jest to binarny odpowiednik formatu JSON zoptymalizowany pod kątem PostgreSQL. JSONB pozwala uniknąć dużej części kosztów przetwarzania związanych ze standardowymi łańcuchami znaków z formatu JSON.

JSONB — wstępnie przetworzone dane w formacie JSON

Pola tekstowe z formatu JSON trzeba przetwarzać za każdym razem, gdy są używane. Jednak wartość w formacie JSONB jest wstępnie przetwarzana, a dane są zapisywane w formacie binarnym. Wymaga to wstępnego przetwarzania początkowych danych wejściowych, ale pobieranie kluczy i wartości odbywa się później znacznie szybciej. Wynika to z tego, że klucze i wartości nie wymagają ponownego przetwarzania — zostały już pobrane i zapisane w dostępnym formacie binarnym.

JSONB różni się od formatu JSON także pod kilkoma innymi względami. Po pierwsze, w JSONB nie można utworzyć więcej niż jednego klucza o tej samej nazwie. Po drugie, kolejność kluczy nie jest zachowywana. Po trzecie, semantycznie nieistotne szczegóły, na przykład spacje, też nie są zachowywane.

Dostęp do danych z pól w formacie JSON lub JSONB

Klucze w formacie JSON można wykorzystać, aby uzyskać dostęp do powiązanej wartości za pomocą operatora `->`. Oto przykładowy kod:

```
SELECT
  '{
    "a": 1,
    "b": 2,
    "c": 3
  }'::JSON -> 'b' AS data;
```

W tym przykładzie występują trzy klucze i wartości w formacie JSON, a kwerenda próbuje uzyskać dostęp do wartości klucza `b`. Dane wyjściowe to jedna wartość — 2. Wynika to z tego, że operacja `-> 'b'` pobiera wartość klucza `b` z następujących danych w formacie JSON: `{"a": 1, "b": 2, "c": 3}`.

PostgreSQL umożliwia też wykonywanie bardziej skomplikowanych operacji dostępu do zagnieżdżonych danych w formacie JSON. Służy do tego operator `#>`. Przyjrzyj się przykładowi:

```
SELECT
  '{
    "a": 1,
    "b": [
      {"d": 4},
      {"d": 6},
      {"d": 4}
    ],
    "c": 3
  }'::JSON #> ARRAY['b', '1', 'd'] AS data;
```

Po prawej stronie operatora `#>` tablica z tekstem definiuje ścieżkę dostępu do szukanej wartości. Tu kod najpierw pobiera wartość `'b'`, czyli listę zagnieżdżonych obiektów JSON. Następnie pobierany jest element tej listy odpowiadający indeksowi `'1'`, czyli element drugi, ponieważ indeksowanie tablicy rozpoczyna się od zera. W ostatnim kroku pobierana jest wartość powiązana z kluczem `'d'`, dlatego dane wyjściowe to 6.

Omawiane funkcje działają dla pól w formatach JSON i JSONB (pamiętaj, że dla pól w formacie JSONB kod działa szybciej). Format JSONB udostępnia też dodatkowe możliwości. Załóżmy, że chcesz przefiltrować wiersze na podstawie pary klucz – wartość. Możesz użyć do tego operatora `@>`, który sprawdza, czy obiekt JSONB podany po lewej stronie zawiera parę klucz – wartość podaną po stronie prawej. Oto przykład:

```
SELECT * FROM customer_sales WHERE customer_json @> '{"customer_id":20}'::JSONB;
```

Ta kwerenda zwróci następujący rekord JSONB:

```
{"email": "ihughillj@nationalgeographic.com", "phone": null, "sales": [],
"last_name": "Hughill", "date_added": "2012-08-08T00:00:00", "first_name":
"Itch", "customer_id": 20}
```


Gdy używasz formatu JSONB, możesz też zwiększyć czytelność danych wyjściowych, korzystając z funkcji `jsonb_pretty`:

```
SELECT JSONB_PRETTY(customer_json) FROM customer_sales WHERE customer_json @>
'{"customer_id":20}':::JSONB;
```

Dane wyjściowe tej kwerendy są pokazane na rysunku 5.13.



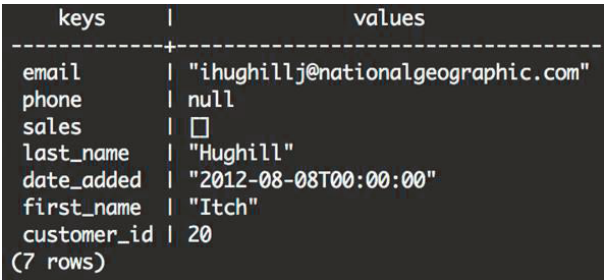
jsonb_pretty	
{	+
"email": "ihughillj@nationalgeographic.com",	+
"phone": null,	+
"sales": [+
],	+
"last_name": "Hughill",	+
"date_added": "2012-08-08T00:00:00",	+
"first_name": "Itch",	+
"customer_id": 20	+
}	+
(1 row)	

Rysunek 5.13. Dane wyjściowe funkcji `JSONB_PRETTY`

Z pola w formacie JSONB możesz też pobrać same klucze i rozdzielić je między wiersze. Służy do tego funkcja `JSONB_OBJECT_KEYS`. Za pomocą tej funkcji możesz też pobrać wartości powiązane z poszczególnymi kluczami z pierwotnego pola w formacie JSONB. Umożliwia to operator `->`. Oto przykład:

```
SELECT
  JSONB_OBJECT_KEYS(customer_json) AS keys,
  customer_json -> JSONB_OBJECT_KEYS(customer_json) AS values
FROM customer_sales
WHERE customer_json @> '{"customer_id":20}':::JSONB
;
```

Dane wyjściowe tej kwerendy są pokazane na rysunku 5.14.



keys	values
email	"ihughillj@nationalgeographic.com"
phone	null
sales	□
last_name	"Hughill"
date_added	"2012-08-08T00:00:00"
first_name	"Itch"
customer_id	20
(7 rows)	

Rysunek 5.14. Pary klucz – wartość rozdzielone między wiele wierszy za pomocą funkcji `JSONB_OBJECT_KEYS`

Stosowanie języka JSONPath do pól w formacie JSONB

Oprócz wcześniej wymienionych funkcji PostgreSQL udostępnia też specjalny język JSONPath, który można wykorzystać do pobierania danych z pól w formacie JSONB. Pierwsza z funkcji tego języka sprawdza, czy w obiekcie JSON istnieje podana ścieżka:

```
SELECT
    jsonb_path_exists(customer_json, '$.sales[0]')
FROM customer_sales
LIMIT 3;
```

Oto dane wyjściowe tej kwerendy:

```
jsonb_path_exists
-----
t
t
t
(3 rows)
```

Ta funkcja dla każdego wiersza zwraca wartość logiczną true lub false w zależności od tego, czy dana wartość w formacie JSONB zawiera transakcję sprzedaży. Funkcja `jsonb_path_exists` ma dwa parametry wymagane: wartość w formacie JSONB i wyrażenie JSONPath (czyli wyrażenie w języku JSONPath). Tu `$` reprezentuje element nadrzędny wartości w formacie JSONB, a notacja `.klucz` służy do dostępu do wartości danego klucza. Pokazany kod uzyskuje dostęp do transakcji sprzedaży za pomocą klucza `.sales`. Wartość `[0]` informuje, że żądana jest pierwsza wartość z tablicy `sales`. Można też użyć zapisu `[*]`, który reprezentuje wszystkie elementy z tablicy `sales`.

Do przedstawionej kwerendy można też dodać inne filtry. Możliwe, że chcesz sprawdzić, czy występują transakcje sprzedaży z wartością `sale_amount` przekraczającą 400 dolarów. W tym celu możesz dodać wyrażenie `filter`:

```
SELECT
    jsonb_path_exists(customer_json, '$.sales[*].sales_amount ? (@ > 400)')
FROM customer_sales
LIMIT 3;
```

Oto dane wyjściowe tej kwerendy:

```
jsonb_path_exists
-----
t
f
f
(3 rows)
```

W tej zmodyfikowanej kwerendzie do ścieżki został dodany kolejny element, `.sales_amount`, co pozwala sprawdzić wartość każdej transakcji z tablicy `sales`. Ponadto za pomocą operatora `?` zostało dodane wyrażenie filtrujące, `?(@ > 400)`. Oznacza ono, że akceptowane są tylko wartości większe niż 400.

Oprócz sprawdzania, czy w obiekcie JSON istnieje określona ścieżka (z dodatkowymi warunkami lub bez nich), można też pobierać wyniki:

```
SELECT
    jsonb_path_query(customer_json, '$.sales[0].sales_amount')
FROM customer_sales
LIMIT 3;
```

Oto dane wyjściowe:

```
jsonb_path_query
-----
479.992
314.991
319.992
(3 rows)
```

Tu funkcja `jsonb_path_query` pobiera pierwszą transakcję sprzedaży za pomocą indeksu pozycyjnego `[0]`, po czym pobiera wartość powiązaną z kluczem `sales_amount`. Funkcja `jsonb_path_query`, podobnie jak `UNNEST`, rozdziela wynik do wielu wierszy, jeśli znajdziesz więcej niż jedną pasującą wartość:

```
SELECT
    jsonb_path_query('{"test":[1, 2, 3]}', '$.test[*]')
;
```

Dane wyjściowe tego kodu wyglądają tak:

```
jsonb_path_query
-----
1
2
3
(3 rows)
```

Jeśli nie istnieje ścieżka spełniająca kryteria z filtra (jeżeli są one podane), funkcja `jsonb_path_query` pomija cały wiersz w danych wyjściowych. Jest to niezgodne z intuicją, ponieważ standardowo filtrowanie wierszy powinno się odbywać tylko na podstawie wyrażeń z klauzuli `WHERE`, dlatego opisywany mechanizm może zwracać nieoczekiwane wyniki.

Co jednak zrobić, jeśli chcesz pobrać tablicę z wartościami transakcji sprzedaży, gdy takich transakcji jest wiele lub gdy nie ma ich wcale? W takim scenariuszu możesz użyć funkcji `jsonb_path_query_array`. W następnym przykładzie zwracana jest cała tablica wartości transakcji wyższych niż 400 dolarów:

```
SELECT
    jsonb_path_query_array(customer_json, '$.sales[*].sales_amount ? (@ > 400)')
FROM customer_sales
LIMIT 3;
```

Oto dane wyjściowe tego kodu:

```
jsnob_path_query_array
-----
[479.992]
[]
[]
(3 rows)
```

W tym przykładzie pierwszy rekord zawiera jedną transakcję przekraczającą wartość progową, a w drugim i trzecim wierszu nie ma żadnych takich transakcji.

Tworzenie i modyfikowanie danych w polu w formacie JSONB

Format JSONB umożliwia też dodawanie i usuwanie elementów. Aby dodać parę klucz – wartość "c": 2, możesz wykonać następujące kroki:

```
select jsonb_insert('{"a":1,"b":"foo"}', ARRAY['c'], '2');
```

Oto dane wyjściowe tej kwerendy:

```
{"a": 1, "b": "foo", "c": 2}
```

Jeśli chcesz, możesz też wstawić wartości do zagnieżdżonego obiektu JSON:

```
select jsonb_insert('{"a":1,"b":"foo", "c":[1, 2, 3, 4]}', ARRAY['c', '1'], '10');
```

Ta kwerenda zwróci następujące dane wyjściowe:

```
{"a": 1, "b": "foo", "c": [1, 10, 2, 3, 4]}
```

W tym przykładzie `ARRAY['c', '1']` reprezentuje ścieżkę, gdzie należy wstawić nową wartość. Tu najpierw pobierane są klucz 'c' i powiązana z nim wartość w postaci tablicy, po czym na pozycji '1' wstawiana jest wartość ('10').

Aby usunąć klucz, wystarczy go odjąć. Oto przykład:

```
SELECT '{"a": 1, "b": 2}':::JSONB - 'b';
```

W tym przykładzie używany jest obiekt JSON z dwoma kluczami, a i b. Gdy odejmiesz b, pozostanie klucz a i powiązana z nim wartość:

```
{"a": 1}
```

Oprócz stosowania opisanych tu metod możesz też przeszukiwać wiele warstw zagnieżdżonych obiektów. W następnym ćwiczeniu dowiesz się, jak to zrobić.

Ćwiczenie 5.04 — przeszukiwanie obiektów JSONB

W tym ćwiczeniu będziesz szukać wartości w danych w formacie JSONB. Załóżmy, że celem jest zidentyfikowanie wszystkich klientów, którzy kupili skuter Blade. Możesz to zrobić, używając danych zapisanych w formacie JSONB.

1. Umieść każdą transakcję sprzedaży w odrębnym wierszu za pomocą funkcji `JSONB_ARRAY_ELEMENTS`:

```
CREATE TEMP TABLE customer_sales_single_sale_json AS (
  SELECT
    customer_json,
    JSONB_ARRAY_ELEMENTS(customer_json -> 'sales') AS sale_json
  FROM customer_sales LIMIT 10
);
```

2. Przefiltruj dane wyjściowe i pobierz rekordy, w których kolumna `product_name` ma wartość 'Blade':

```
SELECT DISTINCT customer_json FROM customer_sales_single_sale_json
WHERE sale_json ->> 'product_name' = 'Blade' ;
```

Operator `->>` działa podobnie jak operator `->`, ale zwraca tekstowe dane wyjściowe zamiast danych w formacie JSONB. Ta kwerenda zwróci wyniki pokazane na rysunku 5.15.

```
{"email": "nеспinaye@51.la", "phone": "818-658-6748", "sales":
[{"product_id": 5, "product_name": "Blade", "sales_amount":
559.992, "sales_transaction_date": "2014-07-19T06:33:44"}]},
"last_name": "Espinay", "date_added": "2014-07-05T00:00:00",
"first_name": "Nichols", "customer_id": 15}
```

Rysunek 5.15. Rekordy, w których kolumna `product_name` ma wartość 'Blade'

3. Użyj funkcji `JSONB_PRETTY()`, aby sformatować dane wyjściowe i poprawić czytelność wyników:

```
SELECT DISTINCT JSONB_PRETTY(customer_json) FROM
customer_sales_single_sale_json
WHERE sale_json ->> 'product_name' = 'Blade' ;
```

Dane wyjściowe tej kwerendy są pokazane na rysunku 5.16.

Teraz, po zastosowaniu funkcji `JSONB_PRETTY()`, możesz łatwo odczytać sformatowane wyniki.

4. Wykonaj te same operacje, korzystając z wyrażeń `JSONPath`:

```
CREATE TEMP TABLE blade_customer_sales AS (
  SELECT
    jsonb_path_query(
      customer_json,
      '$ ? (@.sales[*].product_name == "Blade")'
    ) AS customer_json
  FROM customer_sales
);
SELECT JSONB_PRETTY(customer_json) FROM blade_customer_sales;
```

```

                                jsonb_pretty
-----
{
  "email": "aabatev03@flickr.com",
  "phone": "269-168-7519",
  "sales": [
    {
      "product_id": 5,
      "product_name": "Blade",
      "sales_amount": 699.99,
      "sales_transaction_date": "2014-09-12T06:04:35"
    }
  ],
  "last_name": "Abate",
  "date_added": "2014-07-10T00:00:00",
  "first_name": "Adriana",
  "customer_id": 40180
}

```

Rysunek 5.16. Formatowanie danych wyjściowych za pomocą funkcji JSONB_PRETTY()

5. W ostatnim kroku zlicz klientów, którzy kupili model Blade:

```
SELECT COUNT(1) FROM blade_customer_sales;
```

Oto dane wyjściowe tego kodu:

```

Count
-----
986
(1 row)

```

W tym ćwiczeniu zidentyfikowaliśmy wartości, używając danych zapisanych w formacie JSONB. Do wykonania ćwiczenia zastosowaliśmy funkcje JSONB_PRETTY() i JSONB_ARRAY_ELEMENTS().

Kod źródłowy z tego fragmentu książki znajdziesz na stronie <https://packt.live/37kTwnN>.

Choć format JSONB umożliwia zapisywanie złożonych informacji za pomocą tekstu, dane często są przechowywane jako nieustrukturyzowany tekst. Gdy nie ma zdefiniowanej struktury, odkodowanie pól tekstowych może być trudne, jednak takie pola często mogą zapewniać cenne informacje. W następnym podrozdziale poznasz różne techniki interakcji z polami tekstowymi. Dalej zobaczysz, jak za pomocą analiz wydobywać informacje z tekstu.

Analiza tekstu za pomocą PostgreSQL

W PostgreSQL oprócz przeprowadzania analiz na złożonych strukturach danych można też wykorzystać dane nieliczbowe. W tekście często kryją się cenne informacje. Wyobraź sobie sprzedawcę, który robi notatki na temat obiecującego klienta: „Bardzo obiecująca rozmowa,

„klient chce jutro dokonać zakupu”. Ten tekst zawiera wartościowe dane, podobnie jak następna notatka: „Klient jest niezainteresowany. Nie potrzebuje już naszego produktu”. Choć ten tekst może być przydatny dla kogoś, kto go przeczyta, może też okazać się wartościowy w trakcie analiz. Słowa kluczowe w takich zdaniach, na przykład „obiecujący”, „zakup”, „jutro”, „niezainteresowany” i „nie”, można pobrać za pomocą odpowiednich technik, aby automatycznie identyfikować najbardziej obiecujących potencjalnych klientów.

W każdym bloku tekstu — na przykład w recenzjach klientów, e-mailach lub notatkach sprzedawców — mogą się znajdować słowa kluczowe, które można pobrać w celu sprawdzania trendów. W wielu sytuacjach dane tekstowe mogą się okazać najważniejszym rodzajem danych i trzeba je wykorzystać, aby uzyskać wartościowe informacje.

W tym podrozdziale zobaczysz, jak używać wybranych funkcji z PostgreSQL do pobierania słów kluczowych, które pomogą w wykrywaniu trendów. Wykorzystasz też dostępne w PostgreSQL mechanizmy przeszukiwania tekstu, które umożliwiają błyskawiczne wyszukiwanie danych.

Tokenizacja tekstu

Duże bloki tekstu (na przykład zdania i akapity) mogą zapewniać informacje przydatne dla czytelnika, jednak nieliczne narzędzia analityczne potrafią wydobywać informacje z nieprzetworzonego tekstu. W prawie wszystkich scenariuszach pomocne jest przetwarzanie tekstu na pojedyncze słowa. Tekst jest często dzielony na tokeny, czyli na sekwencje znaków tworzące jednostkę semantyczną. Zwykle tokenem jest słowo w zdaniu, choć w niektórych scenariuszach, na przykład dla angielskiego słowa „can't”, system może zwrócić dwa tokeny — „can” i „t”.

Nawet najnowsze techniki z obszaru **przetwarzania języka naturalnego** zazwyczaj obejmują tokenizację przed rozpoczęciem przetwarzania tekstu. Przetwarzanie języka naturalnego może być przydatne do przeprowadzania analiz wymagających dogłębnego zrozumienia tekstu.

Słowa i tokeny są pomocne, ponieważ można je dopasowywać do różnych dokumentów z danych. Pozwala to wyciągać ogólne wnioski na poziomie zagregowanych danych. Na przykład jeśli masz zbiór danych zawierający notatki sprzedawców i możesz wyróżnić w nim token „zainteresowany”, można przyjąć hipotezę, że notatki obejmujące to słowo dotyczą klientów, którzy są skłonni dokonać zakupu.

PostgreSQL obejmuje mechanizmy, dzięki którym tokenizacja jest dość łatwa. Zacznij od zastosowania funkcji `STRING_TO_ARRAY`, która dzieli łańcuch znaków na tablicę na podstawie ogranicznika, na przykład spacji:

```
SELECT STRING_TO_ARRAY('Damian i Mateusz są przyjaciółmi.', ' ');
```

Oto dane wyjściowe tej kwerendy:

```
{Damian,i,Mateusz,są,przyjaciółmi.}
```

W tym przykładzie zdanie Damian i Mateusz są przyjaciółmi. jest dzielone w miejscach wystąpienia spacji.

Ten tekst obejmuje znak przestankowy, który warto usunąć. Możesz to łatwo zrobić, używając funkcji `REGEXP_REPLACE`. Ta funkcja przyjmuje cztery argumenty: tekst, który chcesz zmodyfikować, wzorzec, który chcesz zastąpić, tekst używany jako zastępnik i dodatkowe opcje (najczęściej stosowana jest opcja 'g', oznaczająca, że zastępowanie powinno odbywać się globalnie, czyli przy każdym napotkaniu wzorca). Możesz usunąć kropkę, korzystając z wzorca pasującego do znaków przestankowych zdefiniowanych w łańcuchu znaków `!,.-` i zastępując podane znaki spacją:

```
SELECT REGEXP_REPLACE('Damian i Mateusz są przyjaciółmi.', '[!,.-]', ' ', 'g');
```

Oto dane wyjściowe tej kwerendy:

```
Damian i Mateusz są przyjaciółmi
```

Znak przestankowy został usunięty.

PostgreSQL udostępnia też narzędzia związane ze stemmingiem, przydatne do identyfikowania rdzenia tokenu. Na przykład tokeny „quick” i „quickly” lub „run” i „running” mają zbliżone znaczenie i ten sam rdzeń. Funkcja `ts_lexize` pomaga znaleźć standardową postać tekstu, zwracając rdzeń słowa. Ilustruje to następny przykład:

```
SELECT TS_LEXIZE('english_stem', 'running');
```

Ten kod zwróci następującą wartość:

```
{run}
```

Omawiane tu techniki możesz wykorzystać do identyfikowania tokenów w tekście, tak jak w następnym ćwiczeniu.

Ćwiczenie 5.05 — analizowanie tekstu

W tym ćwiczeniu na podstawie analizy tekstu określisz liczbę słów kluczowych odpowiadających ocenom wyższym niż średnia i niższym niż średnia. W bazie ZoomZoom dostępne są informacje zwrotne od klientów wraz z ocenami określającymi, jak prawdopodobne jest, że dany klient poleci firmę ZoomZoom znajomym. Badane słowa kluczowe mają pomóc ustalić najważniejsze mocne i słabe punkty firmy, jakie kierownictwo powinno uwzględnić w przyszłości.

Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Pobierz dane z tabeli `customer_survey`, aby zapoznać się ze zbiorem danych:

```
SELECT * FROM customer_survey limit 5;
```

Dane wyjściowe z tej kwerendy są pokazane na rysunku 5.17.

rating	feedback
9	I highly recommend the lemon scooter. It's so fast
10	I really enjoyed the sale - I was able to get the Bat for a 20% discount
4	Overall, the experience was ok. I don't think that the customer service rep was really understanding the issue.
9	The model epsilon has been a fantastic ride - one of the best cars I have ever driven.
9	I've been riding the scooter around town. It's been good in urban areas.

(5 rows)

Rysunek 5.17. Przykładowe odpowiedzi ankietowe klientów z bazy danych

Widać tu, że dostępne są oceny liczbowe z przedziału od 1 do 10 i opinie w formacie tekstowym (kolumna feedback).

- Przeanalizuj tekst, przekształcając go na pojedyncze słowa i powiązane z nimi oceny. Można to zrobić, używając funkcji `STRING_TO_ARRAY` i `UNNEST`:

```
SELECT UNNEST(STRING_TO_ARRAY(feedback, ' ')) AS word, rating FROM
customer_survey limit 10;
```

Dane wyjściowe tej kwerendy przedstawia rysunek 5.18.

word	rating
I	9
highly	9
recommend	9
the	9
lemon	9
scooter.	9
It's	9
so	9
fast	9
I	10

(10 rows)

Rysunek 5.18. Przetworzone tekstowe dane wyjściowe

W tych danych wyjściowych widać, że tokeny nie zostały ustandaryzowane, dlatego sprawiają problemy. Dotyczy to przede wszystkim znaków przestankowych (na przykład `It's`), wielkich liter (na przykład `I` i `It's`) oraz słów typu słowa nieinformatywne (na przykład `I`, `the` i `so`); można je przekształcić, aby wyniki były rzetelniejsze.

- Ustandaryzuj tekst, używając funkcji `ts_lexize` i stemmera dla języka angielskiego — `'english_stem'`. Następnie usuń z pierwotnego tekstu znaki, które nie są literami. Użyj do tego funkcji `REGEXP_REPLACE`. Gdy połączysz te dwie funkcje z pierwotną kwerendą, otrzymasz następujący kod:

```
SELECT
  (TS_LEXIZE('english_stem',
    UNNEST(STRING_TO_ARRAY(
      REGEXP_REPLACE(feedback, '^[^a-zA-Z]+' , ' ', 'g'),
      ' '))
    ))[1] AS token,
  rating
FROM customer_survey
LIMIT 10;
```

Dane wyjściowe tego kodu przedstawia rysunek 5.19.

token	rating
NULL	9
high	9
recommend	9
NULL	9
lemon	9
scooter	9
NULL	9
NULL	9
NULL	9
fast	9
(10 rows)	

Rysunek 5.19. Dane wyjściowe uzyskane za pomocą funkcji TS_LEXIZE i REGEX_REPLACE

Gdy stosowane są takie transformacje, dane wyjściowe są nazywane tokenami, a nie słowami. Tokeny oznaczają każdą jednostkę lingwistyczną.

Masz już najważniejsze tokeny i powiązane z nimi oceny. Zauważ, że w danych wyjściowych z tej operacji znajdują się wartości NULL, które trzeba odfiltrować z par z ocenami.

1. Znajdź średnią ocenę powiązaną z każdym tokenem. Użyj do tego klauzuli GROUP BY:

```
SELECT
    (TS_LEXIZE('english_stem',
        UNNEST(STRING_TO_ARRAY(
            REGEXP_REPLACE(feedback, '^[a-zA-Z]+' , ' ', 'g'),
            ' ')))[1] AS token,
    AVG(rating) AS avg_rating
FROM customer_survey
GROUP BY 1
HAVING COUNT(1) >= 3
ORDER BY 2
;
```

Ta kwerenda grupuje dane według pierwszego wyrażenia z instrukcji SELECT, która przeprowadza tokenizację. Następnie można obliczyć średnią ocenę powiązaną z każdym tokenem. Należy zadbać o to, aby używać tokenów o odpowiedniej liczbie wystąpień, co pozwoli odfiltrować szum. Tu, z powodu małej liczby odpowiedzi z opiniami, wymagane jest występowanie tokenów tylko przynajmniej trzy razy (HAVING COUNT(1) >= 3). Na koniec wyniki są posortowane według drugiego wyrażenia — średniej oceny. Rysunek 5.20 przedstawia dane wyjściowe.

word	avg_rating
pop	2.0000000000000000
batteri	2.3333333333333333
servic	2.3333333333333333
custom	2.3333333333333333
issu	2.5000000000000000
long	2.6666666666666667
ship	2.6666666666666667
email	3.5000000000000000
help	4.0000000000000000
one	4.3333333333333333
littl	4.6666666666666667
hook	5.0000000000000000
get	5.0000000000000000
work	5.0000000000000000
NULL	5.1872659176029963
realli	5.5000000000000000
scooter	5.9090909090909091
ride	6.7500000000000000
model	7.3333333333333333
lemon	7.6666666666666667
great	7.7500000000000000
fast	8.0000000000000000
dealership	9.0000000000000000
sale	9.5000000000000000
discount	9.6666666666666667
(25 rows)	

Rysunek 5.20. Średnie oceny powiązane z tokenami tekstowymi

Po jednej stronie spektrum znajduje się spora grupa negatywnych tokenów. Token **pop** (pękać) prawdopodobnie dotyczy pękających opon, a **batteri** zapewne związany jest z problemami z czasem pracy baterii. Jeśli chodzi o pozytywne wypowiedzi, widać, że klienci używają w nich słów **discount** (rabat), **sale** (wyprzedaż) i **dealership** (salon).

- 2. Zweryfikuj założenia. W tym celu użyj wyrażenia `ILIKE`, aby przefiltrować odpowiedzi ankietowe zawierające poszczególne tokeny:

```
SELECT * FROM customer_survey WHERE feedback ILIKE '%pop%';
```

Ten kod zwróci trzy odpowiedzi (rysunek 5.21).

rating	feedback
1	On my second trip one of the tires popped. I would have really expected it to get repaired under the warranty.
3	I was riding to work and one my wheels popped! It was going to cost \$200 to fix it - what a scam!
2	I popped a wheel, and can't seem to fix it.
(3 rows)	

Rysunek 5.21. Filtrowanie odpowiedzi ankietowych za pomocą wyrażenia `ILIKE`

Wyrażenie `ILIKE` umożliwia dopasowanie tekstu do wzorca. Tu szukany jest tekst zawierający człon `pop`, a wielkość liter nie ma znaczenia. Umieszczenie wzorca między symbolami `%` oznacza, że po lewej i po prawej stronie wzorca może się znajdować dowolna liczba znaków.

Kod źródłowy z tego fragmentu książki znajduje się na stronie <https://packt.live/3fimG9W>.

Po otrzymaniu wyników analiz możesz przekazać najważniejsze problemy do przejrzenia działowi rozwoju produktu. Możesz też poinformować o ogólnych wnioskach: po pierwsze, klienci lubią rabaty, a po drugie, po otwarciu salonów opinie były pozytywne.

ILIKE działa podobnie jak inne wyrażenie z SQL-a — LIKE. W wyrażeniu ILIKE wielkość liter nie ma znaczenia, natomiast są one ważne w wyrażeniu LIKE. Dlatego zwykle lepiej jest używać wyrażenia ILIKE. Jeśli wydajność jest bardzo istotna, pamiętaj, że wyrażenie LIKE może działać trochę szybciej.

Wyszukiwanie tekstu

W trakcie analizowania tekstu z użyciem funkcji agregujących, tak jak w poprzednich punktach, pomocne może być pobieranie z bazy istotnych wpisów, podobnie jak w wyszukiwarce internetowej.

Choć możesz w tym celu użyć wyrażenia ILIKE w klauzuli WHERE, takie rozwiązanie nie jest ani zbyt szybkie, ani rozszerzalne. Co na przykład zrobisz, jeśli zechcesz szukać w tekście wielu słów kluczowych, automatycznie poprawiać literówki w szukanych wyrażeniach lub uwzględniać sytuacje, w których jedno z szukanych słów nie występuje?

W takich scenariuszach możesz użyć mechanizmu przeszukiwania tekstu dostępnego w PostgreSQL. Jeśli ten mechanizm jest w pełni zoptymalizowany, może przeszukiwać miliony dokumentów.

„Dokumenty” oznaczają tu rekordy w przeszukiwanej bazie danych. Każdy dokument reprezentuje encję, której szukasz. Na przykład w prywatnej witrynie mogą to być wpisy na blogu, które obejmują tytuł, autora i artykuł. W ankietach mogą to być same odpowiedzi lub odpowiedź powiązana z pytaniem. Dokument może obejmować wiele pól, a nawet wiele tabel.

Zacznij od funkcji `to_tsvector`, która działa podobnie jak `ts_lexize`. Jednak zamiast tworzyć token na podstawie słowa, dzieli na tokeny cały dokument. Oto przykład:

```
SELECT
    feedback,
    to_tsvector('english', feedback) AS tsvectorized_feedback
FROM customer_survey
LIMIT 1;
```

Ta kwerenda zwraca wynik pokazany na rysunku 5.22.

feedback	tsvectorized_feedback
I highly recommend the lemon scooter. It's so fast	'fast':10 'high':2 'lemon':5 'recommend':3 'scooter':6

Rysunek 5.22. Reprezentacja pierwotnej opinii po jej podziale na tokeny za pomocą funkcji `to_tsvector`

Tu opinia **I highly recommend the lemon scooter. It's so fast** (Gorąco polecam skuter lemon. Jest bardzo szybki) została przekształcona po tokenizacji na wektor `'fast':10 'high':2 'lemon':5 'recommend':3 'scooter':6`. Podobnie jak w funkcji `ts_lexize` słowa nieinformatywne (na przykład `I`, `the`, `It's` i `so`) są pomijane. Inne słowa, na przykład `highly`, są sprowadzane do rdzenia (`high`). Kolejność słów nie jest zachowywana.

Funkcja `to_tsvector` może też przyjmować dane w formatach JSON lub JSONB i dzielić na tokeny wartości (bez kluczy), tworząc na ich podstawie obiekt `tsvector`.

Typ danych wyjściowych opisanej operacji to `tsvector`. Jest to specjalny typ danych zaprojektowany na potrzeby wyszukiwania tekstu. Obok typu `tsvector` przydatny jest też typ `tsquery`, pomocny przy przekształcaniu szukanych wyrażeń na typ danych akceptowalny w PostgreSQL. Załóżmy, że chcesz utworzyć szukane wyrażenie ze słowami kluczowymi `lemon scooter`. Możesz zapisać je tak:

```
SELECT to_tsquery('english', 'lemon & scooter');
```

Jeśli nie chcesz używać operatorów logicznych, możesz zapisać je w prostszy sposób:

```
SELECT plainto_tsquery('english', 'lemon scooter');
```

Obie te kwerendy zwracają ten sam wynik:

```
plainto_tsquery
-----
'lemon' & 'scooter'
(1 row)
```

Funkcja `to_tsquery` akceptuje operatory logiczne, na przykład `|` (or) i `&` (and). Przyjmuje też operator `!` (not).

Możesz też używać operatorów logicznych do łączenia obiektów `tsquery`. Na przykład operator `&&` tworzy szukane wyrażenie, które wymaga dopasowania lewego wyrażenia i prawego wyrażenia. Z kolei operator `||` tworzy wyrażenie, które wymaga dopasowania lewego lub prawego obiektu `tsquery`:

```
SELECT plainto_tsquery('english', 'lemon') && plainto_tsquery('english',
'bat') || plainto_tsquery('english', 'chi');
```

Ta kwerenda zwraca następujący wynik:

```
'lemon' & 'bat' | 'chi'
```

Możesz przesłać kwerendę dotyczącą obiektu `ts_vector`. W tym celu użyj obiektu `ts_query` i operatora `@@`. Możesz na przykład poszukać wszystkich opinii klientów na temat modelu 'lemon scooter':

```
SELECT *
FROM customer_survey
WHERE to_tsvector('english', feedback) @@ plainto_tsquery('english', 'lemon
scooter');
```

Ta kwerenda zwróci trzy wyniki widoczne na rysunku 5.23.

rating	feedback
9	I highly recommend the lemon scooter. It's so fast
8	The lemon scooter has been incredible! I love it!
6	The lemon scooter was a little too fast for me. I will be returning this item.
(3 rows)	

Rysunek 5.23. Dane wyjściowe kwerendy wykorzystującej mechanizmy wyszukiwania z PostgreSQL

Z następnego punktu dowiesz się, jak zoptymalizować wyszukiwanie tekstu w PostgreSQL.

Optymalizowanie wyszukiwania tekstu w PostgreSQL

Choć opisana w poprzednim przykładzie składnia wyszukiwania z PostgreSQL jest prosta, wymaga przekształcenia wszystkich dokumentów tekstowych na obiekty `tsvector` za każdym razem, gdy wykonywane jest nowe wyszukiwanie. Ponadto wyszukiwarka musi przeszukać każdy dokument, aby sprawdzić, czy zawiera tekst pasujący do szukanych słów.

To podejście można ulepszyć na dwa sposoby:

- zapisując obiekty `tsvector`, aby nie wymagały ponownego tworzenia;
- zachowując tokeny i powiązane z nimi dokumenty, podobnie jak indeks na końcu książki zawiera słowa lub wyrażenia oraz numery stron z ich wystąpieniami, dzięki czemu nie trzeba sprawdzać każdego dokumentu, aby stwierdzić, czy pasuje do szukanych słów.

Aby wprowadzić te dwa usprawnienia, trzeba wstępnie wygenerować i zapisać obiekty `tsvector` dla każdego dokumentu i obliczyć **indeks GIN** (ang. *generalized inverted index*).

Do wstępnego wygenerowania obiektów `tsvector` posłużą **widoki zmaterializowane**. Taki widok jest definiowany jako kwerenda, ale w odróżnieniu od zwykłego widoku, który wymaga uruchomienia kwerendy przy każdym użyciu, powoduje utrwalenie i zapisanie wyniku w formie tabeli.

Ponieważ wyniki widoku zmaterializowanego są zapisywane w tabeli, może on nie być zsynchronizowany z tabelami używanymi w kwerendzie.

Do utworzenia widoku zmaterializowanego z wynikami ankiety użyj następującej kwerendy:

```
CREATE MATERIALIZED VIEW customer_survey_search AS (
  SELECT
    rating,
    feedback,
    to_tsvector('english', feedback)
      || to_tsvector('english', rating::text) AS searchable
  FROM customer_survey
);
```

Widać tu, że kolumna `searchable` jest tworzona na podstawie dwóch kolumn — `rating` i `feedback`. W wielu sytuacjach wyszukiwane są dane z kilku pól. Pola można łatwo łączyć, używając kilku obiektów `tsvector` i operatora `||`.

Możesz sprawdzić, czy widok zadziałał, pobierając z niego wiersz:

```
SELECT * FROM customer_survey_search LIMIT 1;
```

Ta kwerenda zwróci dane wyjściowe z rysunku 5.24.

rating	feedback	searchable
9	I highly recommend the lemon scooter. It's so fast	'9':11 'fast':10 'high':2 'lemon':5 'recommend':3 'scooter':6

Rysunek 5.24. Rekord ze zmaterializowanego widoku z obiektem `tsvector`

Gdy musisz odświeżyć widok (na przykład po wstawieniu lub aktualizacji danych), możesz zastosować następującą składnię:

```
REFRESH MATERIALIZED VIEW customer_survey_search;
```

To powoduje ponowne wygenerowanie widoku. W czasie jego generowania starsza kopia widoku pozostaje dostępna i odblokowana.

Możesz też dodać indeks GIN, używając następującej składni:

```
CREATE INDEX idx_customer_survey_search_searchable ON customer_survey_search
USING GIN(searchable);
```

Dzięki tym dwóm operacjom (utworzeniu widoku zmaterializowanego i indeksu GIN) możesz łatwo pisać kwerendy do wyszukiwania słów w tabeli z opiniami klientów:

```
SELECT rating, feedback FROM customer_survey_search WHERE searchable @@
plainto_tsquery('dealership');
```

Ta kwerenda zwraca dane wyjściowe widoczne na rysunku 5.25.

rating	feedback
8	I really appreciated having a dealership so close to me - it made the transaction much easier!
9	The sales people at the dealership were so nice and helpful!
10	The millburn dealership is the best! Those folks are great!

Rysunek 5.25. Dane wyjściowe z widoku zmaterializowanego zoptymalizowanego na potrzeby wyszukiwania

Choć dla małej tabeli liczącej 32 wiersze skrócenie czasu wykonywania kwerendy może być niewielkie lub zerowe, opisane operacje znacznie przyspieszają przetwarzanie dużych tabel (zawierających na przykład miliony wierszy) i umożliwiają użytkownikom szybkie przeszukiwanie baz w czasie liczonym w sekundach.

W następnym zadaniu wykorzystasz opisane techniki w praktyce i utworzysz przeszukiwalną bazę z transakcjami sprzedaży, która umożliwi wyszukiwanie potrzebnych informacji za pomocą kwerend tekstowych.

Zadanie 5.01

— wyszukiwanie i analiza transakcji sprzedaży

W tym zadaniu przygotujesz widok zmaterializowany na potrzeby wyszukiwania danych i odpowiesz na kilka pytań biznesowych. Wykorzystasz do tego wiedzę zdobytą w poprzednich ćwiczeniach. Szef działu sprzedaży w ZoomZoom natrafił na problem: zespół sprzedażowy nie ma łatwego sposobu wyszukiwania klientów. Zgłosiłeś się na ochotnika, aby opracować nowatorską wewnętrzną wyszukiwarkę, która umożliwi wyszukiwanie klientów na podstawie danych kontaktowych i będzie wyświetlać zakupione przez nich w przeszłości produkty.

Oto kroki niezbędne do wykonania tego zadania:

1. Na podstawie tabeli `customer_sales` utwórz przeszukiwalny widok zmaterializowany zawierający jeden rekord dla każdego klienta. Ten widok ma mieć klucz w postaci kolumny `customer_id` i umożliwiać wyszukiwanie wszystkich danych dotyczących klientów: imion, nazwisk, e-maili, numerów telefonu i zakupionych produktów. Możesz też dodać inne pola.
Utwórz indeks wyszukiwania dla tego widoku zmaterializowanego.
2. Sprzedawca pyta Cię, czy można użyć nowej prototypowej wyszukiwarki do znalezienia klienta o imieniu Danny, który kupił skuter Bat. Utwórz kwerendę dotyczącą nowego przeszukiwalnego widoku. Jako szukanych słów użyj Danny Bat. Ile wierszy otrzymałeś?
3. Zespół sprzedażowy chce wiedzieć, jak często się zdarza, że klienci kupują pary: określony model skutera i określony model samochodu. Aby mu odpowiedzieć, złącz tabelę `products` z nią samą, aby otrzymać wszystkie unikatowe pary skuter – samochód.
4. Przyjmij, że edycje limitowane należy połączyć w wynikach ze standardowym modelem (na przykład modele Bat i Bat Limited Edition należy potraktować jak ten sam skuter). Odfiltruj z par produktów wiersze z tekstem `Limited Edition`.
5. Na podstawie wyników złączenia krzyżowego utwórz kwerendę, która zlicza klientów pasujących do poszczególnych par produktów.

Oczekiwane dane wyjściowe są pokazane na rysunku 5.26.

Rozwiązanie tego zadania znajdziesz w „Dodatku”.

query	count
'lemon' & 'model' & 'sigma'	340
'lemon' & 'model' & 'chi'	331
'bat' & 'model' & 'epsilon'	241
'bat' & 'model' & 'sigma'	226
'bat' & 'model' & 'chi'	221
'lemon' & 'model' & 'epsilon'	217
'bat' & 'model' & 'gamma'	153
'lemon' & 'model' & 'gamma'	133
'lemon' & 'zester' & 'model' & 'chi'	28
'lemon' & 'zester' & 'model' & 'epsilon'	22
'blade' & 'model' & 'chi'	21
'lemon' & 'zester' & 'model' & 'sigma'	17
'blade' & 'model' & 'sigma'	12
'lemon' & 'zester' & 'model' & 'gamma'	11
'blade' & 'model' & 'epsilon'	4
'blade' & 'model' & 'gamma'	4
(16 rows)	

Rysunek 5.26. Liczba klientów, którzy zakupili poszczególne kombinacje modeli skuterów i samochodów

W tym zadaniu znaleźliśmy i przeanalizowaliśmy dane z użyciem widoku zmaterializowanego. Następnie użyliśmy słów kluczowych `DISTINCT` i `JOIN` do zmodyfikowania kwerendy. Na koniec pokazaliśmy Ci, jak tworzyć kwerendy z zastosowaniem obiektów `tsquery`, aby uzyskać ostateczne dane wyjściowe.

Podsumowanie

W tym rozdziale omówiliśmy specjalne typy danych, w tym daty i czas, dane geoprzestrzenne, złożone struktury danych i tekstowe typy danych. Jeśli chodzi o daty i czas, pokazaliśmy, jak operować danymi z szeregów czasowych, jak pobierać komponenty i jak zapisywać takie dane w praktyczny sposób umożliwiający przeprowadzanie analiz. W kontekście geoprzestrzennych typów danych nauczyłeś się przekształcać długość i szerokość geograficzną na typ danych `point`, który umożliwia obliczanie odległości między lokalizacjami.

W obszarze złożonych typów danych opisaliśmy kilka przydatnych typów takich jak tablice, format `JSON` i format `JSONB`. Pokazaliśmy, jak tworzyć wartości tego rodzaju, a także jak pisać złożone kwerendy do poruszania się po strukturze takich wartości.

Na koniec dowiedziałeś się, że dane tekstowe mogą być przydatne w analityce — przy wykonywaniu analiz bazujących na słowach kluczowych, a także w kontekście wyszukiwania tekstu, co może być cennym narzędziem analitycznym.

Im większe są zbiory danych, tym więcej czasu zajmują złożone analizy. W następnym rozdziale dokładnie omawiamy, jak rozpocząć optymalizowanie kwerend z wykorzystaniem objaśnień i analiz planu wykonywania kwerendy, a także dodatkowych narzędzi takich jak indeksy, które pozwalają przyspieszyć działanie kwerend.

Wydajny SQL

W tym rozdziale nauczysz się optymalizować korzystanie z bazy danych, co pozwoli wykonywać kwerendy z użyciem mniejszej ilości zasobów. Najpierw zobaczysz, jak silnik bazodanowy wykonuje podstawowe kwerendy. Dzięki temu zrozumiesz przebieg skanowania sekwencyjnego. Następnie przyjrzyj się zoptymalizowanym kwerendom `SELECT`, tworząc dla tabel indeksy, które poprawiają wydajność. Zbadasz też korzyści używania złączeń zamiast innych technik. Zapoznasz się również z zaawansowanymi mechanizmami, tworząc niestandardowe funkcje do wykonywania specjalnych obliczeń. Ponadto zobaczysz, jak stosować w bazach danych niestandardowe ograniczenia oparte na wyzwalaczach. Na koniec poznasz narzędzia i techniki pozwalające przerywać niewydajne kwerendy, które zużywają zasoby bazy danych.

Wprowadzenie

W poprzednim rozdziale zdobyłeś umiejętności potrzebne do skutecznego analizowania danych w bazach SQL-owych. W tym rozdziale skupisz się na wydajności takich analiz i dowiesz się, jak przyspieszyć wykonywanie kwerend SQL-owych. Wydajność i szybkość to bardzo ważne aspekty w analityce, ponieważ jeśli ich nie uwzględniś, fizyczne ograniczenia związane z czasem i mocą obliczeniową mogą mieć istotny wpływ na wyniki analiz. Aby lepiej zrozumieć takie ograniczenia, przyjrzyjmy się dwóm scenariuszom.

Założmy, że przeprowadzasz **analizy post hoc** (po fakcie lub zdarzeniu). W pierwszym scenariuszu ukończyłeś badanie i zebrałeś duży zbiór danych z obserwacjami różnych czynników lub cech. Dotyczy to na przykład bazy danych z transakcjami salonów samochodowych, która pozwala przeanalizować dane sprzedażowe powiązane z każdym klientem.

Po zebraniu danych są one analizowane pod kątem wzorców i wniosków związanych z opisem problemu. Gdy zbiór danych jest wystarczająco duży, możesz szybko natrafić na problemy, jeśli najpierw nie zoptymalizujesz kwerend. Najczęstszym problemem jest długi czas wykonywania kwerend. Choć może się wydawać, że nie jest to poważny kłopot, niepotrzebnie długi czas przetwarzania może powodować następujące skutki:

- Ograniczenie szczegółowości analiz. Gdy wykonywanie każdej kwerendy zajmuje dużo czasu, harmonogram prac nad projektem może ograniczyć liczbę wykonywanych kwerend, a tym samym szczegółowość i złożoność analiz.
- Ograniczenie zakresu analizowanych danych. Sztuczne ograniczenie zbioru danych przez pobranie z niego próbek może pozwolić na ukończenie analiz w akceptowalnym czasie, ale kosztem jest zmniejszenie liczby uwzględnianych obserwacji. To z kolei może prowadzić do przypadkowego wprowadzenia tendencji w analizach.
- Konieczność jednoczesnego wykorzystania znacznie większej liczby zasobów w celu ukończenia analiz w akceptowalnym czasie, co zwiększa koszty projektu.

Innym problemem związanym z nieoptymalnymi kwerendami jest zwiększenie ilości potrzebnej pamięci i mocy obliczeniowej. Może to prowadzić do dwóch sytuacji:

- niepowodzenie analiz spowodowane niewystarczającą ilością zasobów;
- znaczne zwiększenie kosztów projektu w celu pozyskania potrzebnych zasobów.

Analizy i kwerendy mogą też być elementem usługi lub produktu. Przyjrzyjmy się teraz drugiemu scenariuszowi, w którym analizy są wykonywane jako element dużej usługi lub rozbudowanego produktu, dlatego kwerendy bazodanowe muszą być przetwarzane w czasie rzeczywistym lub zbliżonym do rzeczywistego. W takich przypadkach optymalizacja i wydajność są niezbędne, aby produkt odniósł sukces. Przykładem jest system nawigacji GPS, który uwzględnia sytuację na drodze sygnalizowaną przez innych użytkowników.

Aby taki system był skuteczny i zapewniał aktualne informacje, bazę danych trzeba analizować z szybkością dostosowaną do prędkości samochodu i pokonanej trasy. Wszelkie opóźnienia w analizie, które uniemożliwiają aktualizację nawigacji w odpowiedzi na sytuację na drodze, mają poważny wpływ na komercyjną wartość aplikacji.

Ten przykład pokazuje, że wydajność jest ważna, jeśli chcesz przeprowadzać skuteczne i dokładne analizy post hoc, i bezwzględnie konieczna, jeżeli analizy danych są wykonywane w ramach działania produktu lub usługi.

Choć to nie specjalista od data science lub analityk danych odpowiada za to, by procesy w środowisku produkcyjnym i baza danych działały z optymalną wydajnością, bardzo ważne jest, aby kwerendy używane w analizach były jak najwydajniejsze. Jeśli nie masz wydajnej i aktualnej bazy danych, inne poprawki mogą nie pomóc w przyspieszeniu analiz. W następnym podrozdziale omawiamy metody zwiększania wydajności skanowania informacji w bazie danych.

Metody skanowania baz danych

SQL-owe bazy danych udostępniają wiele różnych metod skanowania, wyszukiwania i pobierania danych. Optymalna metoda skanowania jest w dużym stopniu zależna od zastosowania i stanu bazy danych. Ile rekordów znajduje się w bazie? Które pola są istotne? Ile rekordów będzie zwracanych? Jak często trzeba wykonywać daną kwerendę? To tylko niektóre z pytań, które warto zadać przy wybieraniu najodpowiedniejszej metody skanowania.

W tym podrozdziale omawiamy wybrane spośród dostępnych metod wyszukiwania, opisujemy, jak używać ich w SQL-u do skanowania danych, a także przedstawiamy kilka scenariuszy, w których należy stosować (lub nie) poszczególne techniki.

Plany wykonywania kwerend

Przed omówieniem różnych metod wykonywania kwerend lub skanowania baz danych w poszukiwaniu informacji warto wyjaśnić, w jaki sposób serwer SQL-a podejmuje decyzje dotyczące stosowanych typów kwerend. Bazy danych zgodne z SQL-em korzystają z rozbudowanego narzędzia, **planera kwerend**, który używa różnych mechanizmów działających na serwerze, aby przeanalizować żądanie i wybrać sposób jego wykonania.

Planer kwerend optymalizuje szereg różnych zmiennych w żądaniu, aby skrócić łączny czas jego wykonywania. Te zmienne są szczegółowo opisane w dokumentacji systemu PostgreSQL (<https://www.postgresql.org/docs/current/runtime-config-query.html>) i obejmują parametry reprezentujące koszt sekwencyjnego pobierania stron i operacji procesora oraz wielkość pamięci podręcznej.

W tym rozdziale nie omawiamy dokładnie analiz wykonywanych w planerze kwerend, ponieważ szczegóły techniczne tego procesu są dość skomplikowane. Ważne jest jednak, aby umieć zinterpretować plan wyświetlany przez planer kwerend. Umiejętność interpretacji planu jest niezwykle ważna, jeśli chcesz zapewnić wysoką wydajność bazy danych, pozwala bowiem zmodyfikować treść i strukturę kwerend w celu zoptymalizowania szybkości ich wykonywania. Dlatego zanim przejdziemy do omawiania różnych metod skanowania, nabierzesz praktyki w stosowaniu i interpretowaniu analiz planera kwerend.

Skanowanie sekwencyjne i inne metody skanowania

Gdy chcesz pobrać informacje z bazy danych, planer kwerend musi przeszukać dostępne rekordy, by uzyskać potrzebne dane. W bazie danych stosuje się różne strategie, aby uporządkować i rozdzielić informacje w celu szybkiego ich zwrócenia. Proces przeszukiwania bazy danych przez serwer SQL-a jest nazywany skanowaniem.

We wszystkich przykładach z tego rozdziału używane są wiersz poleceń lub powłoka.

Najpierw zapoznasz się ze **skanowaniem sekwencyjnym**, ponieważ najłatwiej je zrozumieć i działa w każdym scenariuszu. W niektórych sytuacjach skanowanie sekwencyjne nie jest najszybsze lub najwydajniejsze, jednak zawsze zwraca prawidłowe wyniki. Innym ciekawym aspektem skanowania sekwencyjnego jest to, że — choć może nie zdajesz sobie z tego sprawy — w poprzednich rozdziałach już nie raz przeprowadzaliśmy takie skanowanie. Przypominasz sobie poniższe polecenie z rozdziału 4., „Importowanie i eksportowanie danych”?

```
SELECT * FROM customers LIMIT 5
```

Rysunek 6.1 przedstawia dane wyjściowe tego kodu.

customer_id	title	first_name	last_name	suffix	email
1		Arlena	Riveles		ariveles0@stumbleupon.com
2	Dr	Ode	Stovin		ostovin1@npr.org
3		Braden	Jordan		bjordan2@geocities.com
4		Jessika	Nussen		jnussen3@salon.com
5		Lonnie	Rembaud		lrembaud4@discovery.com

(5 rows)

Rysunek 6.1. Sześć pierwszych kolumn danych wyjściowych instrukcji SELECT

Pobieranie danych bezpośrednio z bazy za pomocą polecenia SELECT skutkuje uruchomieniem skanowania sekwencyjnego. W tym podejściu serwer bazy danych sprawdza każdy rekord i porównuje go z kryteriami skanowania sekwencyjnego. W efekcie zwracane są wiersze pasujące do kryteriów. Jest to skanowanie przez atak siłowy, dlatego zawsze można go użyć, aby znaleźć dane. W wielu sytuacjach jest to także najwydajniejsza metoda automatycznie wybierana przez serwer SQL-a. Jest to prawdą zwłaszcza wtedy, gdy spełniony jest którykolwiek z następujących warunków:

- Tabela jest dość mała.
- Pole używane w poszukiwaniach zawiera wiele powtarzających się wartości.
- Planer stwierdza, że skanowanie sekwencyjne przy stosowanych kryteriach jest równie (lub bardziej) wydajne niż inne metody skanowania.

W tym ćwiczeniu wprowadzona zostanie instrukcja EXPLAIN, która wyświetla objaśnienie planu kwerendy przed jej wykonaniem. Gdy instrukcja EXPLAIN jest używana razem z poleceniem SQL-a, interpreter SQL-a nie wykonuje polecenia, lecz zwraca kroki (plan kwerendy), jakie wykona interpreter w celu zwrócenia oczekiwanych wyników.

Plan kwerendy zawiera wiele informacji, a umiejętność ich zrozumienia jest niezbędna, jeśli chcesz zwiększyć wydajność kwerend w bazie danych. Planowanie kwerend samo w sobie jest złożonym zagadnieniem, a opanowanie interpretowania planów wymaga praktyki. Nawet w oficjalnej dokumentacji systemu PostgreSQL jest napisane, że czytanie planów kwerend jest sztuką, która zasługuje na odpowiednią uwagę. Tu zaczniesz od prostego planu, a następnie przyjrzyj się bardziej skomplikowanym kwerendom i ich planom.

Wykonaj teraz ćwiczenie, aby zinterpretować działanie planera kwerend.

Wszystkie ćwiczenia i zadania z tego rozdziału są dostępne także w serwisie GitHub na stronie <https://packt.live/2XSiV56>.

We wszystkich ćwiczeniach i zadaniach z tego rozdziału miary używane w analizach kwerend zależą od konfiguracji systemu. Dlatego Twoje dane wyjściowe mogą się różnić od danych uzyskanych w ćwiczeniach i zadaniach w książce. Ważne jest to, że dane wyjściowe z tego rozdziału ilustrują działanie omawianych zasad.

Ćwiczenie 6.01

— interpretowanie działania planera kwerend

W tym ćwiczeniu zinterpretujesz działanie planera kwerend, używając polecenia EXPLAIN. Kwerenda będzie dotyczyła tabeli `emails` z bazy `sqllda`. Następnie przyjrzyś się bardziej skomplikowanej kwerendzie, wyszukującej w polu `clicked_date` daty pomiędzy dwiema określonymi wartościami. Bazę `sqllda` należy wczytać zgodnie z opisem z wprowadzenia.

Pobierz plik *Exercise6.01.sql* (*Ćwiczenie6.01.sql* z wersji polskojęzycznej z witryny wydawnictwa Helion) z kodu źródłowego do tej książki. Ten plik zawiera wszystkie kwerendy wykorzystywane w tym ćwiczeniu. Wprowadzaj je ręcznie w interpreterze SQL-a, aby lepiej zrozumieć działanie planera kwerend.

Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Otwórz interfejs wiersza poleceń (CMD lub terminal) i nawiąż połączenie z bazą `sqllda`:

```
C:\> psql sqllda
```

Po nawiązaniu połączenia zobaczysz interfejs wskazanej bazy z systemu PostgreSQL:

```
Type "help" for help
sqllda=#
```

2. Wprowadź poniższe polecenie, aby pobrać plan kwerendy dotyczącej tabeli `emails`:

```
sqllda=# EXPLAIN SELECT * FROM emails;
```

Pojawią się informacje podobne do tych z rysunku 6.2.

```

              QUERY PLAN
-----
Seq Scan on emails (cost=0.00..9606.58 rows=418158 width=79)
(1 row)

```

Rysunek 6.2. Plan kwerendy dotyczącej tabeli `emails`

Te informacje są zwracane przez planer kwerend. Choć jest to najprostszy możliwy przykład, w informacjach od planera znajduje się sporo elementów, które warto omówić. Przyjrzyj się więc danym wyjściowym krok po kroku. Najpierw spójrz na rysunek 6.3.

```

QUERY PLAN
-----
Seq Scan on emails (cost=0.00..9606.58 rows=418158 width=79)
(1 row)

```

Rysunek 6.3. Typ skanowania

Pierwszym podanym aspektem planu jest typ skanowania używany dla kwerendy. Więcej na temat typów skanowania dowiesz się z dalszej części rozdziału. Tu zapamiętaj, że **Seq Scan** (rysunek 6.3), czyli skanowanie sekwencyjne, to prosty, ale niezawodny sposób wykonywania kwerendy.

Pierwszą miarą podawaną przez planer, widoczną na rysunku 6.4, jest koszt operacji przygotowawczych, czyli czas potrzebny przed rozpoczęciem skanowania. Może on być niezbędny na posortowanie danych lub wykonanie innych wstępnych czynności. Warto zauważyć, że ta miara z rysunku 6.4 jest podawana w jednostkach kosztu, a nie w sekundach lub milisekundach. Te jednostki często wskazują na liczbę żądań kierowanych do dysku lub pobrań stron, nie stanowią więc bezwzględnej miary czasu.

```

QUERY PLAN
-----
Seq Scan on emails (cost=0.00..9606.58 rows=418158 width=79)
(1 row)

```

Rysunek 6.4. Koszt operacji przygotowawczych

Następna wartość (rysunek 6.5) określa łączny koszt wykonania kwerendy, jeśli pobrane zostaną wszystkie dostępne wiersze. W niektórych sytuacjach nie wszystkie dostępne wiersze są pobierane (to zagadnienie omawiamy dalej).

```

QUERY PLAN
-----
Seq Scan on emails (cost=0.00..9606.58 rows=418158 width=79)
(1 row)

```

Rysunek 6.5. Koszt łączny

Następna wartość w planie (rysunek 6.6) określa łączną liczbę wierszy dostępnych do zwrócenia, jeśli plan zostanie wykonany w całości.

```

QUERY PLAN
-----
Seq Scan on emails (cost=0.00..9606.58 rows=418158 width=79)
(1 row)

```

Rysunek 6.6. Liczba zwracanych wierszy

Ostatnia wartość (rysunek 6.7), zgodnie z podpisem rysunku, określa długość każdego wiersza w bajtach.

```

QUERY PLAN
-----
Seq Scan on emails (cost=0.00..9606.58 rows=418158 width=79)
(1 row)

```

Rysunek 6.7. Długość każdego wiersza

Gdy uruchomisz polecenie EXPLAIN, PostgreSQL nie wykona kwerendy i nie zwróci wartości. Zamiast tego zwróci opis wraz z kosztami przetwarzania poszczególnych etapów planu.

3. Pobierz plan kwerendy dotyczącej tabeli `emails` z limitem 5 wierszy. Wpisz poniższą instrukcję w interpreterze z systemu PostgreSQL:

```
sql>=# EXPLAIN SELECT * FROM emails LIMIT 5;
```

Ta kwerenda jest podobna do poprzedniej, ale tu występuje ograniczenie do pięciu pierwszych rekordów. Ta kwerenda powoduje wyświetlenie przez planer danych wyjściowych pokazanych na rysunku 6.8.

```

QUERY PLAN
-----
Limit (cost=0.00..0.11 rows=5 width=79)
-> Seq Scan on emails (cost=0.00..9606.58 rows=418158 width=79)
(2 rows)

```

Rysunek 6.8. Plan wykonania kwerendy z ograniczeniem liczby zwracanych wierszy

Na rysunku 6.8 widać, że plan obejmuje dwa wiersze. To oznacza, że plan składa się z dwóch kroków. Dolny wiersz planu (tu dotyczy on pierwszego wykonywanego kroku) to powtórzenie danych z rysunku 6.7. Górny wiersz planu dotyczy fragmentu ograniczającego wyniki do 5 wierszy. Klauzula `LIMIT` powoduje dodatkowe koszty, jednak są one bardzo niskie w porównaniu z kosztem planu z niższego poziomu, który wymaga pobrania 418158 wierszy kosztem 9606 żądań stron. Etap z klauzulą `LIMIT` powoduje zwrócenie tylko 5 wierszy kosztem 0,11 żądania stron.

Ogólny szacowany koszt obsługi żądania obejmuje czas na pobranie informacji z dysku i liczbę wierszy, jakie trzeba przeskanować. Wewnętrzne parametry `seq_page_cost` i `cpu_tuple_cost` definiują koszt odpowiednich operacji dla tabel z określonej bazy. Obie zmienne można zmodyfikować, aby wpłynąć na kroki przygotowywane przez planer, przy czym wymaga to dużej wiedzy.

Więcej informacji znajdziesz w dokumentacji systemu PostgreSQL na stronie <https://www.postgresql.org/docs/current/runtime-config-query.html>.

4. Teraz wykonaj bardziej skomplikowaną kwerendę, która wyszukuje w kolumnie `clicked_date` dane spomiędzy dwóch podanych wartości. Wpisz następującą instrukcję w interpreterze systemu PostgreSQL:

```
sql>=# EXPLAIN SELECT * FROM emails WHERE clicked_date BETWEEN
'2011-01-01' and '2011-02-01';
```


Otrzymasz plan wykonywania kwerendy podobny do tego z rysunku 6.9.

```
Gather (cost=1000.00..9051.49 rows=130 width=79)
  Workers Planned: 2
    -> Parallel Seq Scan on emails (cost=0.00..8038.49 rows=54 width=79)
        Filter: ((clicked_date >= '2011-01-01 00:00:00'::timestamp without time zone) AND
(clicked_date <= '2011-02-01 00:00:00'::timestamp without time zone))
(4 rows)
```

Rysunek 6.9. Skanowanie sekwencyjne przy wyszukiwaniu dat pomiędzy dwóch podanych wartości

Pierwszym wartym uwagi aspektem tej kwerendy jest to, że obejmuje ona kilka różnych kroków. Kwerenda z niższego poziomu jest podobna do poprzedniej, ponieważ też wykonuje skanowanie sekwencyjne. Jednak tu zamiast ograniczać liczbę wyników kwerenda filtruje dane na podstawie podanych łańcuchów znaków ze znacznikami czasu.

Zauważ, że skanowanie sekwencyjne jest tu przeprowadzane równolegle, o czym informuje tekst **Parallel Seq Scan** i to, że planowane jest użycie dwóch wątków roboczych. Każde skanowanie sekwencyjne powinno zwrócić 54 wiersze, a koszt tej operacji to 8038.49. Na wyższym poziomie planu widoczny jest krok **Gather**, wykonywany na początku kwerendy. W tej kwerendzie po raz pierwszy koszt operacji przygotowawczych jest niezerowy (1000). Łączny koszt wynosi 9051.49 (wartość ta dotyczy etapów łączenia i wyszukiwania danych).

Kod źródłowy z tego fragmentu książki znajdziesz na stronie <https://packt.live/30BwNIY>.

W tym ćwiczeniu przyjrzałeś się planerowi kwerend i danym wyjściowym polecenia EXPLAIN. Na podstawie stosunkowo prostych kwerend przedstawiliśmy szereg aspektów planera kwerend SQL-owych, a także szczegółowych informacji zwracanych przez ten planer. Dobre zrozumienie planera kwerend i zwracanych przez niego informacji przyda Ci się w trakcie nauki o danych. Pamiętaj jednak, że to zrozumienie rozwija się wraz z czasem i praktyką. Zawsze możesz zajrzeć do dokumentacji systemu PostgreSQL dostępnej na stronie <https://www.postgresql.org/docs/current/using-explain.html>.

W tym rozdziale nabierzesz wprawy w czytaniu planów wykonywania kwerend, przyglądając się różnym sposobom skanowania i technikom służącym poprawie wydajności.

Zadanie 6.01 — plany wykonywania kwerendy

W tym zadaniu pobierzesz plan, aby odczytać i zinterpretować informacje zwracane przez planer. Załóżmy, że nadal pracujesz z bazą sql_da zawierającą rekordy z danymi klientów. Dział finansów chce, abyś wdrożył system do regularnego generowania raportów na temat aktywności klientów mieszkających na określonym obszarze geograficznym. Aby umożliwić szybkie generowanie raportu, trzeba oszacować czas wykonywania kwerend w SQL-u. Użyj polecenia EXPLAIN, aby ustalić, ile potrwa wykonywanie niektórych kwerend potrzebnych do uzyskania raportu:

1. Otwórz PostgreSQL i nawiąż połączenie z bazą sql_da.
2. Użyj polecenia EXPLAIN, aby wyświetlić plan wykonywania kwerendy pobierającej wszystkie dostępne rekordy z tabeli customers.

3. Przeczytaj plan i ustal łączny koszt wykonywania kwerendy, koszt operacji przygotowawczych, liczbę zwracanych wierszy oraz długość każdego wiersza. Na podstawie analizy danych wyjściowych planera odpowiedz, jakie są jednostki wartości używanych w planie.
4. Ponownie użyj kwerendy z kroku 2., ale tym razem ogranicz liczbę zwracanych rekordów do 15.
Przyjrzawszy się zaktualizowanemu planowi wykonywania kwerendy, odpowiedz, ile kroków obejmuje nowy plan. Jaki jest koszt kroku ograniczającego liczbę rekordów?
5. Wygeneruj plan kwerendy pobierającej wszystkie wiersze z danymi klientów mieszkających w rejonie o szerokości geograficznej od 30 do 40 stopni. Jaki jest łączny koszt planu? Ile wierszy zwróci ta kwerenda?

Oczekiwane dane wyjściowe są pokazane na rysunku 6.10.

```

QUERY PLAN
-----
Seq Scan on customers (cost=0.00..1786.00 rows=26439 width=140)
  Filter: ((latitude > '30'::double precision) AND (latitude < '40'::double precision))
(2 rows)

```

Rysunek 6.10. Plan kwerendy pobierającej klientów mieszkających w rejonie o szerokości geograficznej od 30 do 40 stopni

Rozwiązanie tego zadania znajdziesz w „Dodatku”. Jako dodatkowe wyzwanie spróbuj wykonać to ćwiczenie w Pythonie z użyciem narzędzia `psycopg2`.

W tym zadaniu przećwiczyłeś czytanie planów zwracanych przez planer kwerend. Wcześniej wyjaśniliśmy, że opanowanie czytania planów wymaga sporo praktyki. To zadanie rozpoczyna ten proces. Gorąco zachęcamy do częstego stosowania polecenia `EXPLAIN`, gdyż pozwoli Ci to doskonalić się w czytaniu planów.

Z następnego punktu dowiesz się, jak zwiększyć wydajność kwerend za pomocą skanowania indeksu.

Skanowanie indeksu

Skanowanie indeksu jest jedną z technik zwiększania wydajności kwerend w bazach danych. Skanowanie indeksu różni się od skanowania sekwencyjnego tym, że przed przeszukiwaniem rekordów bazy danych potrzebny jest dodatkowy krok przygotowawczy.

Najprościej zrozumieć skanowanie indeksu, porównując je do przeglądania indeksu książki. W trakcie prac nad książkami niebeletrystycznymi wydawnictwo analizuje treść tekstu i zapisuje numery stron powiązane z alfabetycznie posortowanymi pojęciami. Wydawnictwo wkłada więc pracę w przygotowanie indeksu dla czytelnika. Podobny indeks można utworzyć w bazie w systemie PostgreSQL.

Indeks bazy danych zawiera wstępnie przygotowany i uporządkowany zbiór (lub podzbiór) odsyłaczy do danych zgodny z określonymi warunkami. Gdy podczas wykonywania kwerendy dostępny jest indeks zawierający adekwatne informacje, planer może wykorzystać wstępnie przygotowane i uporządkowane dane z tego indeksu. Jeśli indeks jest niedostępny, baza danych musi wielokrotnie skanować wszystkie rekordy, sprawdzając, czy zawierają potrzebne informacje.

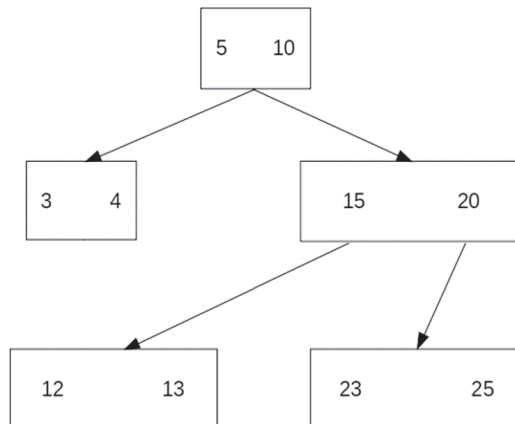
Nawet jeśli wszystkie potrzebne informacje znajdują się na początku bazy danych, to gdy indeks jest niedostępny, i tak trzeba przeskanować wszystkie rekordy. Oczywiście zajmuje to niepotrzebnie dużo czasu.

Istnieje wiele różnych strategii indeksowania, jakie PostgreSQL może stosować w celu zwiększenia wydajności wyszukiwania. Dostępne techniki obejmują **B-drzewa**, **indeksy z haszowaniem**, **indeksy GIN** (ang. *generalized inverted index*) i **indeksy GiST** (ang. *generalized search tree*).

Wszystkie wymienione rodzaje indeksów mają wady i zalety, dlatego są stosowane w różnych sytuacjach. Jednym z najczęściej używanych typów indeksów są B-drzewa. Jest to domyślny indeks wykorzystywany w PostgreSQL, dostępny w niemal każdym oprogramowaniu bazodanowym. Dlatego zaczynamy od omówienia indeksów w postaci B-drzew. Wyjaśniamy, dlaczego są one potrzebne, a także opisujemy niektóre ich ograniczenia.

Indeks w postaci B-drzewa

Indeks w postaci B-drzewa to binarne drzewo poszukiwań cechujące się tym, że automatycznie się równoważy, zachowując strukturę danych umożliwiającą wydajne wyszukiwanie. Ogólna struktura B-drzewa jest pokazana na rysunku 6.11. Widać na nim, że każdy węzeł takiego drzewa ma nie więcej niż dwa elementy (co pomaga zachować równowagę w drzewie), a pierwszy węzeł ma dwoje dzieci. Te cechy są typowe dla B-drzew, w których każdy węzeł może zawierać do n komponentów, dlatego w pewnym momencie wymaga podziału na węzły potomne. Gałęzie drzewa kończą się na liściach, które z definicji nie mają dzieci.



Rysunek 6.11. Ogólne B-drzewo

Posłuż się tym rysunkiem jako przykładem. Załóżmy, że w indeksie w postaci tego B-drzewa szukasz liczby 13. Należy zacząć od pierwszego węzła i ustalić, czy liczba jest mniejsza niż 5 lub większa niż 10. To prowadzi do prawej gałęzi drzewa, gdzie należy ustalić, czy szukana wartość jest mniejsza niż 15 lub większa niż 20. Po wybraniu gałęzi dla wartości mniejszych niż 15 dotrzesz do lokalizacji liczby 13 w indeksie.

Od razu widać, że ta operacja jest znacznie szybsza niż sprawdzanie wszystkich dostępnych wartości. Aby ta technika zwiększała wydajność, drzewo musi być zrównoważone i zapewniać wygodne ścieżki. Ponadto potrzebne są informacje wystarczające do podziału węzłów. Jeśli drzewo indeksu zawiera tylko kilka wartości używanych do podziału i dużą liczbę obserwacji, indeks spowoduje jedynie rozmieszczenie danych w kilku grupach.

W kontekście przeszukiwania baz danych za pomocą B-drzew potrzebny jest warunek do podziału danych, a także informacje niezbędne do tego, by podziały były dokonywane w sensowny sposób. Nie trzeba przejmować się logiką poruszania się po drzewie, ponieważ zarządza tym sama baza, a proces ten zależy od warunków przeszukiwania. Mimo to ważne jest, aby zrozumieć wady i zalety tej techniki, ponieważ pomoże to w podejmowaniu właściwych wyborów w trakcie tworzenia indeksu zapewniającego optymalną wydajność.

Aby utworzyć indeks dla zbioru danych, użyj następującej składni:

```
CREATE INDEX <nazwa indeksu> ON <nazwa tabeli>(kolumna tabeli);
```

Możesz też zastosować dodatkowe warunki i ograniczenia, aby indeks był bardziej selektywny:

```
CREATE INDEX <nazwa indeksu> ON <nazwa tabeli>(kolumna tabeli) WHERE  
[warunek];
```

Możesz również podać rodzaj indeksu:

```
CREATE INDEX <nazwa indeksu> ON <nazwa tabeli> USING TYPE(kolumna tabeli)
```

PostgreSQL obsługuje wiele rodzajów indeksów, w tym B-drzewa, indeksy z haszowaniem, indeksy GiST itd. Poniższa kwerenda tworzy dla podanej kolumny indeks w postaci B-drzewa:

```
CREATE INDEX ix_customers ON customers USING BTREE(customer_id);
```

Ta kwerenda powoduje wyświetlenie następującego prostego komunikatu:

```
CREATE INDEX
```

To oznacza, że indeks został z powodzeniem utworzony.

W następnym ćwiczeniu zaczniesz od prostego planu i przejdziesz do bardziej złożonych kwerend i planów, obejmujących skanowanie indeksu.

Ćwiczenie 6.02 — kwerenda ze skanowaniem indeksu

W tym ćwiczeniu zastosujesz kilka różnych operacji skanowania indeksu i zbadasz cechy wydajnościowe każdej z nich.

Wróć do scenariusza z poprzedniego zadania. Załóżmy, że ukończyłeś narzędzie do generowania raportów, ale chcesz, aby kwerendy były wykonywane szybciej. Postaraj się to osiągnąć, wykorzystując indeksy i ich skanowanie. Pamiętaj, że używana jest tabela z informacjami o klientach, które obejmują dane takie jak imię i nazwisko, adres e-mail, numer telefonu i adres, a także długość geograficzną i szerokość geograficzną powiązane z adresem. Wykonaj następujące kroki:

1. Wczytaj bazę danych `sql`da zgodnie z opisem w wprowadzenia. Pobierz plik *Exercise6.02.sql* z katalogu z kodem źródłowym. Ten plik zawiera wszystkie kwerendy używane w tym ćwiczeniu, jednak należy je wprowadzać ręcznie w interpreterze SQL-a, co pozwoli lepiej zrozumieć działanie planera kwerend.
2. Otwórz PostgreSQL i nawiąż połączenie z bazą `sql`da:

```
C:\> psql sqlda
```

Po nawiązaniu połączenia zobaczysz interfejs do bazy z systemu PostgreSQL:

```
Type "help" for help
sqlda=#
```

3. Zaczynij od tabeli `customers` i użyj polecenia `EXPLAIN`, aby ustalić koszt kwerendy i liczbę wierszy zwracanych, gdy pobierane są wszystkie rekordy z kolumną `state` o wartości `F0`:

```
sqlda=# EXPLAIN SELECT * FROM customers WHERE state='F0';
```

Dane wyjściowe powinny wyglądać podobnie jak na rysunku 6.12, choć mogą wystąpić pewne rozbieżności.

```

              QUERY PLAN
-----
Seq Scan on customers (cost=0.00..1661.00 rows=1 width=140)
    Filter: (state = 'F0'::text)
(2 rows)

```

Rysunek 6.12. Plan wykonywania kwerendy ze skanowaniem sekwencyjnym i ograniczeniami

Zauważ, że zwracany jest tylko 1 wiersz, a koszty operacji przygotowawczych to 0, jednak łączny koszt kwerendy wynosi 1661.

4. Ustal, ile unikatowych wartości występuje w kolumnie `state`. Ponownie użyj polecenia `EXPLAIN`:

```
sqlda=# EXPLAIN SELECT DISTINCT state FROM customers;
```

Dane wyjściowe powinny wyglądać podobnie jak na rysunku 6.13.

```

QUERY PLAN
-----
HashAggregate  (cost=1661.00..1661.51 rows=51 width=3)
  Group Key: state
    -> Seq Scan on customers  (cost=0.00..1536.00 rows=50000 width=3)

```

Rysunek 6.13. Unikatowe wartości z kolumny state

W kolumnie state występuje więc 51 unikatowych wartości.

5. Utwórz indeks ix_state dla kolumny state tabeli customers:

```
sql>=# CREATE INDEX ix_state ON customers(state);
```

6. Ponownie uruchom instrukcję EXPLAIN z kroku 5.:

```
sql>=# EXPLAIN SELECT * FROM customers WHERE state='FO';
```

Dane wyjściowe tego kodu powinny wyglądać podobnie do tych z rysunku 6.14.

```

QUERY PLAN
-----
Index Scan using ix_state on customers  (cost=0.29..8.31 rows=1 width=140)
  Index Cond: (state = 'FO'::text)

```

Rysunek 6.14. Plan wykonywania kwerendy dotyczącej tabeli customers; plan obejmuje skanowanie indeksu

Zauważ, że teraz planowane jest skanowanie indeksu utworzonego przed chwilą w kroku 5. Widać też, że koszt operacji przygotowawczych jest niezerowy (0.29), ale łączny koszt został znacznie zmniejszony z wcześniejszego poziomu 1661 do 8.31. W tym właśnie tkwi wartość skanowania indeksu.

Teraz przeanalizujmy nieco inny przykład i sprawdźmy, ile czasu zajmie przeszukiwanie kolumny gender.

7. Użyj polecenia EXPLAIN, aby wyświetlić plan wykonywania kwerendy wyszukiującej w bazie wszystkie rekordy z danymi mężczyzn:

```
sql>=# EXPLAIN SELECT * FROM customers WHERE gender='M';
```

Dane wyjściowe są pokazane na rysunku 6.15.

```

QUERY PLAN
-----
Seq Scan on customers  (cost=0.00..1661.00 rows=24960 width=140)
  Filter: (gender = 'M'::text)

```

Rysunek 6.15. Plan wykonywania kwerendy dotyczącej tabeli customers; przeprowadzane jest tu skanowanie sekwencyjne

8. Utwórz indeks ix_gender dla kolumny gender tabeli customers:

```
sql>=# CREATE INDEX ix_gender ON customers(gender);
```

9. Sprawdź, czy indeks jest dostępny. W tym celu użyj instrukcji \d:

```
\d customers;
```

Gdy przewiniesz listę w dół, zobaczysz indeksy z przedrostkiem `ix_`, a także kolumnę tabeli użytą do utworzenia poszczególnych indeksów (rysunek 6.16).

```

state      | text      |
postal_code | text      |
latitude   | double precision |
longitude  | double precision |
date_added | timestamp without time zone |
Indexes:
    "ix_gender" btree (gender)
    "ix_state" btree (state)

```

Rysunek 6.16. Struktura tabeli customers

10. Ponownie uruchom instrukcję EXPLAIN z kroku 7.:

```
sql>=# EXPLAIN SELECT * FROM customers WHERE gender='M';
```

Dane wyjściowe tego kodu są pokazane na rysunku 6.17.

```

-----
              QUERY PLAN
-----
Seq Scan on customers (cost=0.00..1661.00 rows=24960 width=140)
  Filter: (gender = 'M'::text)

```

Rysunek 6.17. Plan wykonywania kwerendy ze skanowaniem sekwencyjnym i warunkiem

Zauważ, że plan wykonywania kwerendy w ogóle się nie zmienił mimo dostępności indeksu. Wynika to z tego, że brakuje informacji do utworzenia przydatnego drzewa na podstawie kolumny `gender`. Ta kolumna zawiera tylko dwie wartości, M i F. Indeks dla tabeli `gender` dzieli informacje na dwie części — jedną gałąź z danymi mężczyzn i drugą gałąź z danymi kobiet. Indeks nie dzieli danych wystarczająco dobrze, aby dawał tu jakiegokolwiek korzyści. Planer i tak musi wykonać skanowanie sekwencyjne połowy danych, dlatego nie warto ponosić kosztów generowania indeksu. To dlatego planer kwerend nie korzysta z indeksu.

11. Użyj instrukcji EXPLAIN, aby wyświetlić plan wykonywania kwerendy wyszukującej dane klientów zamieszkujących obszar o szerokości geograficznej od 30 do 38 stopni:

```
sql>=# EXPLAIN SELECT * FROM customers WHERE (latitude < 38) AND
(latitude > 30);
```

Rysunek 6.18 przedstawia dane wyjściowe tego kodu.

```

-----
              QUERY PLAN
-----
Seq Scan on customers (cost=0.00..1786.00 rows=17788 width=140)
  Filter: ((latitude < '38'::double precision) AND (latitude > '30'::double precision))
(2 rows)

```

Rysunek 6.18. Plan wykonywania kwerendy dotyczącej tabeli customers ze skanowaniem sekwencyjnym i złożonym warunkiem

Zauważ, że w tej kwerendzie stosowane jest skanowanie sekwencyjne z filtrowaniem. Początkowe skanowanie sekwencyjne zwraca przed filtrowaniem 17788 rekordów, a koszt tej operacji to 1786 (przy koszcie operacji przygotowawczych równym 0).

12. Utwórz indeks `ix_latitude` dla kolumny `latitude` tabeli `customers`:

```
sql># CREATE INDEX ix_latitude ON customers(latitude);
```

13. Ponownie uruchom kwerendę z kroku 11. i przyjrzyj się planowi (rysunek 6.19).

QUERY PLAN

```
-----
Bitmap Heap Scan on customers  (cost=382.62..1685.44 rows=17788 width=140)
  Recheck Cond: ((latitude < '38'::double precision) AND (latitude > '30'::double precision))
-> Bitmap Index Scan on ix_latitude  (cost=0.00..378.17 rows=17788 width=0)
   Index Cond: ((latitude < '38'::double precision) AND (latitude > '30'::double precision))
(4 rows)
```

Rysunek 6.19. Plan po ponownym uruchomieniu kwerendy

Widać, że plan jest teraz dużo bardziej skomplikowany niż wcześniej. Stosowane są tu etapy skanowania stron na podstawie indeksu bitmapowego (ang. *bitmap heap scan*) i skanowania indeksu bitmapowego (ang. *bitmap index scan*). Skanowanie indeksu bitmapowego omawiamy dalej. Najpierw zapoznaj się z dodatkowymi informacjami. W tym celu dodaj człon `ANALYZE` do instrukcji `EXPLAIN`.

14. Użyj instrukcji `EXPLAIN ANALYZE` do pobrania planu wykonywania kwerendy zwracającej dane z tabeli `customers` dotyczące klientów, których miejsce zamieszkania znajduje się na szerokości geograficznej od 30 do 38 stopni:

```
sql># EXPLAIN ANALYZE SELECT * FROM customers WHERE (latitude < 38)
AND (latitude > 30);
```

Wyświetlone zostaną dane wyjściowe widoczne na rysunku 6.20.

QUERY PLAN

```
-----
Bitmap Heap Scan on customers  (cost=382.62..1685.44 rows=17788 width=140) (actual time=4.064..12.818 rows=17896 loops=1)
  Recheck Cond: ((latitude < '38'::double precision) AND (latitude > '30'::double precision))
  Heap Blocks: exact=1036
-> Bitmap Index Scan on ix_latitude  (cost=0.00..378.17 rows=17788 width=0) (actual time=3.700..3.701 rows=17896 loops=1)
   Index Cond: ((latitude < '38'::double precision) AND (latitude > '30'::double precision))
Planning Time: 0.301 ms
Execution Time: 14.582 ms
(7 rows)
```

Rysunek 6.20. Plan wykonywania kwerendy z dodatkowymi informacjami z polecenia `EXPLAIN ANALYZE`

Z tych dodatkowych informacji wynika, że czas planowania wynosi **0,301 ms**, a czas wykonywania **14,582 ms**. Skanowanie indeksu bitmapowego zajmuje prawie tyle samo czasu co operacje przygotowawcze z kroku skanowania stron na podstawie indeksu bitmapowego.

15. Utwórz następny indeks, z wartościami kolumny `latitude` tabeli `customers` większymi niż 30 i mniejszymi niż 38:

```
sql># CREATE INDEX ix_latitude_less ON customers(latitude) WHERE
(latitude < 38) and (latitude > 30);
```

16. Ponownie wykonaj kwerendę z kroku 14. i porównaj plany wykonywania kwerend (rysunek 6.21).

QUERY PLAN

```

Bitmap Heap Scan on customers  (cost=297.67..1600.49 rows=17788 width=140) (actual time=3.187..12.117 rows=17896 loops=1)
  Recheck Cond: ((latitude < '38'::double precision) AND (latitude > '30'::double precision))
  Heap Blocks: exact=1036
-> Bitmap Index Scan on ix_latitude_less  (cost=0.00..293.23 rows=17788 width=0) (actual time=2.726..2.727 rows=17896 loops=1)
    Planning Time: 0.681 ms
    Execution Time: 13.905 ms
(6 rows)

```

Rysunek 6.21. Plan wykonywania kwerendy ilustrujący zależności między czasem planowania i czasem wykonywania

Gdy używaliśmy ogólnego indeksu kolumny, czas planowania wynosił 0,301 ms, a czas wykonywania — 14,582 ms. Po utworzeniu bardziej precyzyjnego indeksu te wartości wynoszą 0,681 ms i 13,905 ms. Użycie precyzyjnego indeksu pozwoliło skrócić czas wykonywania o **0,677 ms** kosztem wydłużenia czasu planowania o **0,38 ms**.

Kod źródłowy z tego fragmentu książki znajdziesz na stronie <https://packt.live/2YuHeVI>.

W ten sposób udało Ci się zwiększyć wydajność kwerendy, ponieważ indeksy poprawiły proces wyszukiwania. Konieczne może być poniesienie dodatkowych kosztów na początku w celu utworzenia indeksu, ale po jego przygotowaniu ponowne kwerendy będą wykonywane szybciej.

Zadanie 6.02 — skanowanie indeksu

W tym zadaniu sprawdzisz, czy można wykorzystać skanowanie indeksu, aby skrócić czas wykonywania kwerendy. Po utworzeniu systemu generującego raporty o klientach dla działu marketingu („Zadanie 6.01 — planowanie kwerend”) poproszono Cię o umożliwienie identyfikowania rekordów także na podstawie adresów IP (oprócz identyfikowania na podstawie nazwisk klientów). Wiesz, że liczba adresów IP jest duża, a wyszukiwanie musi działać szybko. Zaplanuj kwerendy potrzebne do wyszukiwania rekordów na podstawie adresu IP, a także do wyszukiwania klientów ze skrótem Jr przy imieniu i nazwisku.

Oto kroki niezbędne do wykonania tego zadania:

1. Użyj poleceń EXPLAIN i ANALYZE, aby wyświetlić plan wykonywania kwerendy wyszukiwającej wszystkie rekordy o adresie IP 18.131.58.65. Jak długo trwa planowanie i wykonywanie tej kwerendy?
2. Utwórz ogólny indeks dla kolumny z adresami IP.
3. Ponownie uruchom kwerendę z kroku 1. Ile czasu zajmuje planowanie i wykonywanie kwerendy?
4. Utwórz bardziej precyzyjny indeks dla kolumny z adresami IP. Użyj warunku, zgodnie z którym adres IP ma być równy 18.131.58.65.
5. Ponownie uruchom kwerendę z kroku 1. Ile czasu zajmuje planowanie i wykonywanie kwerendy? Jakie są różnice między kolejnymi wersjami kwerendy?

6. Użyj poleceń EXPLAIN i ANALYZE, aby wyświetlić plan wykonywania kwerendy zwracającej wszystkie rekordy ze skrótem Jr. Jak długo trwa planowanie i wykonywanie tej kwerendy?
 7. Utwórz ogólny indeks dla kolumny ze skrótami (suffix).
 8. Ponownie uruchom kwerendę z kroku 6. Ile czasu zajmuje planowanie i wykonywanie kwerendy?
- Rysunek 6.22 przedstawia oczekiwane dane wyjściowe.

```

                                QUERY PLAN
-----
Bitmap Heap Scan on customers (cost=5.12..318.44 rows=107 width=140) (actual time=0.146..0.440 rows=102 loops=1)
  Recheck Cond: (suffix = 'Jr'::text)
  Heap Blocks: exact=100
    -> Bitmap Index Scan on ix_jr (cost=0.00..5.09 rows=107 width=0) (actual time=0.092..0.092 rows=102 loops=1)
        Index Cond: (suffix = 'Jr'::text)
Planning Time: 0.411 ms
Execution Time: 0.511 ms
(7 rows)

```

Rysunek 6.22. Plan wykonywania kwerendy ze skanowaniem po utworzeniu indeksu dla kolumny suffix

Rozwiązanie tego zadania znajdziesz w „Dodatku”.

W tym zadaniu zwiększyliśmy wydajność kwerendy, ponieważ indeksy przyspieszyły proces wyszukiwania. Z następnego punktu dowiesz się, jak działa indeks z haszowaniem.

Indeks z haszowaniem

Ostatni z omawianych rodzajów indeksów to indeks z haszowaniem. Stabilna wersja tego indeksu jest dostępna w PostgreSQL dopiero od niedawna. Wcześniej pojawiały się ostrzeżenia, że ten mechanizm nie gwarantuje bezpieczeństwa; ponadto wydajność indeksów z haszowaniem była zwykle niższa niż indeksów w postaci B-drzew. W czasie gdy powstaje ta książka, indeksy z haszowaniem umożliwiają porównywanie wartości wyłącznie za pomocą operatora równości (=).

Skoro więc ten mechanizm jest stabilny od niedawna i ma ograniczone możliwości, po co go używać? No cóż, indeksy z haszowaniem pozwalają opisywać duże zbiory danych (liczące dziesiątki tysięcy wierszy lub większe) za pomocą bardzo niewielkiej ilości danych. Dzięki temu więcej danych można zmieścić w pamięci, co skraca czas wyszukiwania w niektórych kwerendach. Jest to ważne przede wszystkim w bazach, które zajmują przynajmniej kilka gigabajtów.

W indeksach z haszowaniem wzrost wydajności jest uzyskiwany za pomocą funkcji haszującej. Jest to funkcja matematyczna, która przyjmuje dane lub zbiór danych i zwraca określoną liczbę znaków alfanumerycznych na podstawie przekazanych danych i unikatowego kodu.

Żałujemy, że masz klientkę "Josephine Marquez". Możesz przekazać te dane do funkcji haszującej, a ta zwróci skrót, na przykład *01f38e*. Ponadto dostępne są też rekordy dotyczące męża Josephine, który ma na imię Julio. Skrót dla Julia może mieć postać *43eb38a*. W tablicy z haszowaniem do wyszukiwania danych używane są pary klucz – wartość.

Można (i trzeba) użyć wyniku funkcji haszującej, aby podać klucz na podstawie danych z odpowiedniego wiersza bazy danych. Te dane są wartością powiązaną z kluczem. Jeśli klucz jest unikatowy dla danej wartości, można szybko uzyskać dostęp do potrzebnych informacji. Ta technika pozwala też zmniejszyć łączną wielkość indeksu w pamięci (jeśli zapisywane są tylko wygenerowane skróty), a także radykalnie skrócić wyszukiwanie w trakcie wykonywania kwerend.

Indeks z haszowaniem można utworzyć za pomocą składni podobnej jak dla indeksów w postaci B-drzew:

```
CREATE INDEX <nazwa indeksu> ON <nazwa tabeli> USING HASH(kolumna tabeli)
```

Następny przykład pokazuje, jak utworzyć indeks z haszowaniem dla kolumny `gender` z tabeli `customers`:

```
sql>=# CREATE INDEX ix_gender ON customers USING HASH(gender);
```

Pamiętaj, że planer kwerend może pominąć indeks, jeśli stwierdzi, że nie przyspiesza on wykonywania kwerendy lub nie jest dla niej odpowiedni. Ponieważ skanowanie indeksu z haszowaniem ma ograniczone zastosowania, w wielu sytuacjach takie indeksy mogą być ignorowane. Mimo to wykonaj teraz ćwiczenie, w którym zastosujesz indeks z haszowaniem. To ćwiczenie pozwoli Ci też zobaczyć różnice w wydajności poszczególnych typów indeksów.

Ćwiczenie 6.03 — tworzenie kilku indeksów z haszowaniem, aby zbadać ich wydajność

W tym ćwiczeniu utworzysz kilka indeksów z haszowaniem i zbadasz możliwy wzrost wydajności, jaki może wynikać z ich zastosowania. Zacznij od ponownego uruchomienia kilku kwerend z poprzednich ćwiczeń i porównania czasów wykonywania:

1. Usuń wszystkie indeksy, używając polecenia `DROP INDEX` dla każdego z utworzonych wcześniej indeksów (`ix_gender`, `ix_state` i `ix_latitude_less`):

```
DROP INDEX <nazwa indeksu>;
```

2. Użyj instrukcji `EXPLAIN` i `ANALYZE` do zbadania kwerendy dotyczącej tabeli `customer` bez indeksu z haszowaniem. Kwerenda ma pobierać klientów płci męskiej:

```
sql>=# EXPLAIN ANALYZE SELECT * FROM customers WHERE gender='M';
```

Wyświetlone zostaną dane wyjściowe podobne do tych z rysunku 6.23.

QUERY PLAN

```
Seq Scan on customers (cost=0.00..1661.00 rows=24960 width=140) (actual time=0.024..26.937 rows=24956 loops=1)
  Filter: (gender = 'M'::text)
  Rows Removed by Filter: 25044
  Planning Time: 0.107 ms
  Execution Time: 29.905 ms
(5 rows)
```

Rysunek 6.23. Standardowe skanowanie sekwencyjne

Widać tu, że szacowany czas planowania to **0,107 ms**, a czas wykonania wynosi **29,905 ms**. Jednak użycie tego samego planu nie zawsze gwarantuje uzyskanie tego samego czasu.

3. Utwórz dla kolumny gender indeks w postaci B-drzewa i ponów kwerendę, aby ustalić wydajność, gdy używany jest indeks domyślny:

```
sql>=# CREATE INDEX ix_gender ON customers USING btree(gender);
```

Rysunek 6.24 przedstawia dane wyjściowe tego kodu.

QUERY PLAN

```
Seq Scan on customers (cost=0.00..1661.00 rows=24960 width=140) (actual time=0.028..21.504 rows=24956 loops=1)
  Filter: (gender = 'M'::text)
  Rows Removed by Filter: 25044
  Planning Time: 0.444 ms
  Execution Time: 23.955 ms
(5 rows)
```

Rysunek 6.24. Planer kwerend ignoruje indeks w postaci B-drzewa

Widać tu, że planer kwerend nie użył indeksu w postaci B-drzewa, lecz przeprowadził skanowanie sekwencyjne. Koszty skanowania pozostają takie same, ale zmieniły się szacowane czasy planowania i wykonywania. Nie jest to zaskoczeniem, ponieważ są to tylko szacunki bazujące na różnych czynnikach, takich jak dane znajdujące się w pamięci i ograniczenia związane z operacjami wejścia – wyjścia.

4. Ręcznie uruchom poniższą kwerendę przynajmniej pięciokrotnie i zwróć uwagę na to, jak zmieniają się szacunki czasu:

```
sql>=# EXPLAIN ANALYZE SELECT * FROM customers WHERE gender='M';
```

Wyniki pięciu kolejnych kwerend są widoczne na rysunku 6.25. Zwróć uwagę na to, że czas planowania i czas wykonania są za każdym razem inne.

5. Usuń indeks:

```
sql>=# DROP INDEX ix_gender;
```

6. Utwórz indeks z haszowaniem dla kolumny gender:

```
sql>=# CREATE INDEX ix_gender ON customers USING HASH(gender);
```

7. Powtórz kwerendę z kroku 4., aby sprawdzić czas wykonania:

```
sql>=# EXPLAIN ANALYZE SELECT * FROM customers WHERE gender='M';
```

Wyświetlone zostaną dane wyjściowe widoczne na rysunku 6.26.

```

                                QUERY PLAN
-----
Seq Scan on customers (cost=0.00..1661.00 rows=24960 width=140) (actual time=0.020..21.596 rows=24956 loops=1)
  Filter: (gender = 'M'::text)
  Rows Removed by Filter: 25044
Planning Time: 0.521 ms
Execution Time: 24.105 ms
(5 rows)

sql>=# EXPLAIN ANALYZE SELECT * FROM customers WHERE gender='M';
                                QUERY PLAN
-----
Seq Scan on customers (cost=0.00..1661.00 rows=24960 width=140) (actual time=0.023..22.629 rows=24956 loops=1)
  Filter: (gender = 'M'::text)
  Rows Removed by Filter: 25044
Planning Time: 0.158 ms
Execution Time: 25.162 ms
(5 rows)

sql>=# EXPLAIN ANALYZE SELECT * FROM customers WHERE gender='M';
                                QUERY PLAN
-----
Seq Scan on customers (cost=0.00..1661.00 rows=24960 width=140) (actual time=0.023..21.765 rows=24956 loops=1)
  Filter: (gender = 'M'::text)
  Rows Removed by Filter: 25044
Planning Time: 0.153 ms
Execution Time: 24.198 ms
(5 rows)

sql>=# EXPLAIN ANALYZE SELECT * FROM customers WHERE gender='M';
                                QUERY PLAN
-----
Seq Scan on customers (cost=0.00..1661.00 rows=24960 width=140) (actual time=0.023..21.758 rows=24956 loops=1)
  Filter: (gender = 'M'::text)
  Rows Removed by Filter: 25044
Planning Time: 0.158 ms
Execution Time: 24.277 ms
(5 rows)

sql>=# EXPLAIN ANALYZE SELECT * FROM customers WHERE gender='M';
                                QUERY PLAN
-----
Seq Scan on customers (cost=0.00..1661.00 rows=24960 width=140) (actual time=0.023..21.967 rows=24956 loops=1)
  Filter: (gender = 'M'::text)
  Rows Removed by Filter: 25044
Planning Time: 0.163 ms
Execution Time: 24.444 ms
(5 rows)

```

Rysunek 6.25. Pięć powtórzeń z tym samym skanowaniem sekwencyjnym

```

                                QUERY PLAN
-----
Seq Scan on customers (cost=0.00..1661.00 rows=24960 width=140) (actual time=0.020..22.642 rows=24956 loops=1)
  Filter: (gender = 'M'::text)
  Rows Removed by Filter: 25044
Planning Time: 0.300 ms
Execution Time: 25.155 ms
(5 rows)

```

Rysunek 6.26. Planer kwerendy ignoruje indeks z haszowaniem

Indeks z haszowaniem dla kolumny gender (podobnie jak indeks w postaci B-drzewa) nie daje korzyści, dlatego planer go nie używa. Wynika to z tego, że kolumna gender przyjmuje tylko dwie wartości.

- Użyj polecenia EXPLAIN ANALYZE, aby ocenić wydajność kwerendy, która pobiera wszystkich klientów ze stanu F0:

```
sql>=# EXPLAIN ANALYZE SELECT * FROM customers WHERE state='F0';
```

Wyświetlone zostaną dane wyjściowe z rysunku 6.27.

```

QUERY PLAN
-----
Seq Scan on customers (cost=0.00..1661.00 rows=1 width=140) (actual time=22.293..22.293 rows=0 loops=1)
  Filter: (state = 'FO'::text)
  Rows Removed by Filter: 50000
Planning Time: 0.188 ms
Execution Time: 22.328 ms
(5 rows)

```

Rysunek 6.27. Skanowanie sekwencyjne z filtrowaniem według stanu

9. Utwórz indeks w postaci B-drzewa dla kolumny state tabeli customers i ponownie sprawdź wydajność kwerendy:

```

sql>=# CREATE INDEX ix_state ON customers USING BTREE(state);
sql>=# EXPLAIN ANALYZE SELECT * FROM customers WHERE state='FO';

```

Dane wyjściowe tego kodu są pokazane na rysunku 6.28.

```

QUERY PLAN
-----
Index Scan using ix_state on customers (cost=0.29..8.31 rows=1 width=140) (actual time=0.060..0.060 rows=0 loops=1)
  Index Cond: (state = 'FO'::text)
Planning Time: 0.374 ms
Execution Time: 0.103 ms
(4 rows)

```

Rysunek 6.28. Wzrost wydajności dzięki indeksowi w postaci B-drzewa

Tu widoczna jest znaczna poprawa wydajności dzięki indeksowi w postaci B-drzewa przy niewielkim koszcie operacji przygotowawczych. Jaka jest wydajność kwerendy ze skanowaniem indeksu? Widoczny jest spadek czasu wykonania z **22,3 ms** do **0,103 ms** przy wzroście czasu planowania na poziomie około 50%.

10. Usuń indeks w postaci B-drzewa ix_state i utwórz indeks z haszowaniem:

```

sql>=# DROP INDEX ix_state;
sql>=# CREATE INDEX ix_state ON customers USING HASH(state);

```

11. Użyj polecenia EXPLAIN ANALYZE, aby ocenić wydajność skanowania z haszowaniem:

```

sql>=# EXPLAIN ANALYZE SELECT * FROM customers WHERE state='FO';

```

Dane wyjściowe tego kodu są pokazane na rysunku 6.29.

```

QUERY PLAN
-----
Index Scan using ix_state on customers (cost=0.00..8.02 rows=1 width=140) (actual time=0.014..0.014 rows=0 loops=1)
  Index Cond: (state = 'FO'::text)
Planning Time: 0.271 ms
Execution Time: 0.048 ms
(4 rows)

```

Rysunek 6.29. Dodatkowy wzrost wydajności związany z użyciem indeksu z haszowaniem

Widać, że w tej konkretnej kwerendzie indeks z haszowaniem jest wyjątkowo skuteczny, ponieważ w porównaniu z indeksem w postaci B-drzewa pozwala skrócić czas planowania i operacji przygotowawczych, a także ogólny koszt. Ponadto czas wykonania spada z około **22 ms** do około **0,05 ms**.

Kod źródłowy z tego fragmentu książki jest dostępny na stronie <https://packt.live/37lcMS4>.

W tym ćwiczeniu zastosowaliśmy indeks z haszowaniem do sprawdzenia wydajności określonej kwerendy. Zobaczyłeś, że użycie takiego indeksu pozwoliło zmniejszyć czas wykonywania kwerendy.

Zadanie 6.03 — stosowanie indeksów z haszowaniem

W tym zadaniu zbadasz, czy użycie indeksów z haszowaniem zwiększa wydajność kwerendy dotyczącej tabeli `emails` z bazy `sql`da. Dział marketingu ma do Ciebie następną prośbę. Tym razem chce, abyś przeanalizował skuteczność e-mailowej kampanii marketingowej.

Ponieważ współczynnik powrodożenia w kompaniach e-mailowych jest niski, do wielu klientów jednocześnie wysyłanych jest dużo różnych e-maili. Użyj poleceń `EXPLAIN` i `ANALYZE`, aby ocenić czas i koszt planowania, a także czas i koszt wykonywania kwerendy pobierającej wszystkie wiersze z tematem e-maila `Shocking Holiday Savings On Electric Scooters` oraz kwerendy pobierającej wszystkie wiersze z tematem e-maila `Black Friday. Green Cars.`:

1. Użyj poleceń `EXPLAIN` i `ANALYZE`, aby ustalić czas i koszt planowania oraz czas i koszt wykonywania kwerendy pobierającej wszystkie wiersze z tematem e-maila `Shocking Holiday Savings On Electric Scooters` oraz kwerendy pobierającej wszystkie wiersze z tematem e-maila `Black Friday. Green Cars.`
2. Utwórz indeks z haszowaniem dla kolumny `email_subject`.
3. Powtórz krok 1. Porównaj dane wyjściowe planera kwerend dla wersji bez indeksu z haszowaniem i z takim indeksem. Jaki wpływ indeks z haszowaniem ma na wydajność obu kwerend?
4. Utwórz indeks z haszowaniem dla kolumny `customer_id`.
5. Użyj instrukcji `EXPLAIN` i `ANALYZE`, aby oszacować czas pobierania wszystkich wierszy z wartością pola `customer_id` większą niż 100. Jakiego rodzaju skanowanie zostało zastosowane i dlaczego?

Oczekiwane dane wyjściowe są pokazane na rysunku 6.30.

```

QUERY PLAN
-----
Seq Scan on emails (cost=0.00..10651.98 rows=417309 width=79) (actual time=0.024..121.483 rows=417315 loops=1)
  Filter: (customer_id > 100)
  Rows Removed by Filter: 843
Planning Time: 0.199 ms
Execution Time: 152.656 ms
(5 rows)

```

Rysunek 6.30. Planer kwerend ignoruje indeks z haszowaniem z powodu ograniczeń tej techniki

Rozwiązanie tego zadania znajdziesz w „Dodatku”.

W tym zadaniu stosowane jest skanowanie sekwencyjne zamiast skanowania z haszowaniem. Wynika to z ograniczeń skanowania z haszowaniem. W czasie gdy powstaje ta książka, w skanowaniu z haszowaniem można sprawdzać tylko równość, co oznacza możliwość wyszukiwania jedynie wartości równych podanej.

Skuteczne korzystanie z indeksów

Poznałeś już różne metody skanowania oraz używałeś indeksów w postaci B-drzew i indeksów z haszowaniem do skrócenia czasu wykonywania kwerend. Przyjrzałeś się też kilku różnym przykładom, w których indeks dla pola został utworzony, ale nie był używany przez planer kwerend, jeśli było to mniej wydajne od domyślnego rozwiązania.

W tym punkcie omawiamy właściwe korzystanie z indeksów do skracania czasu wykonywania kwerend, bo choć indeksy mogą wydawać się oczywistym sposobem na przyspieszenie działania kwerend, nie zawsze dają taki efekt.

Przyjrzyj się następującym sytuacjom:

- **Pole, dla którego tworzysz indeks, często jest modyfikowane.** W takim scenariuszu, gdy często wstawiasz lub usuwasz wiersze tabeli, utworzony indeks szybko może stać się niewydajny, ponieważ został utworzony dla danych, które już nie istnieją lub zmieniły wartość.
Pomyśl o indeksie na końcu książki. Jeśli zmienisz kolejność rozdziałów, indeks nie będzie już poprawny i trzeba będzie go zaktualizować. Wtedy konieczne może być okresowe ponowne indeksowanie danych, aby zagwarantować, że referencje do danych będą aktualne.
W SQL-u indeksy danych można odtworzyć za pomocą polecenia REINDEX. Jego stosowanie wymaga przeanalizowania kosztów, technik i strategii częstego odtwarzania indeksu oraz uwzględnienia ich razem z innymi czynnikami wpływającymi na wydajność, takimi jak przyspieszenie wykonywania kwerendy dzięki indeksowi, wielkość bazy danych, a nawet to, czy zmiana struktury bazy danych pozwoli całkowicie uniknąć problemu.
- **Indeks jest nieaktualny, a istniejące referencje są nieprawidłowe, albo występują segmenty niezindeksowanych danych, przez co planer kwerend nie może wykorzystać indeksu.** W takiej sytuacji indeks jest tak stary, że nie można go użyć, dlatego konieczna jest jego aktualizacja.
- **Często szukasz rekordów, używając tych samych kryteriów wyszukiwania dotyczących określonego pola.** Sytuacja wyglądała podobnie, gdy szukałeś w bazie klientów mieszkających w lokalizacji o szerokości geograficznej mniejszej niż 38 lub większej niż 30. Używałeś do tego kwerendy `SELECT * FROM customers WHERE (latitude < 38) and (latitude > 30)`.

W tym przykładzie wydajniejsze może być utworzenie częściowego indeksu z wykorzystaniem podzbioru danych, na przykład tak: `CREATE INDEX ix_latitude_less ON customers(latitude) WHERE (latitude < 38) and (latitude > 30)`. Dzięki temu indeks jest tworzony tylko z użyciem potrzebnych

danych, dlatego jest mniejszy, można go szybciej przeskanować, jest łatwiejszy w utrzymaniu, a także można go stosować w bardziej skomplikowanych kwerendach.

- **Baza danych nie jest duża.** W takiej sytuacji koszt tworzenia i stosowania indeksu może przewyższać korzyści. Skanowanie sekwencyjne jest dość szybkie (zwłaszcza jeśli dotyczy danych, które już znajdują się w pamięci RAM), dlatego jeśli utworzysz indeks dla małego zbioru danych, nie ma gwarancji, że planer kwerend go użyje lub że indeks zapewni istotne korzyści.

Z następnego podrozdziału dowiesz się, jak dzięki indeksom przyspieszyć złączanie tabel.

Wydajne złączenia

Mechanizm złączeń (polecenia JOIN) w bazach SQL-owych stanowi bardzo przydatną i skuteczną metodę łączenia danych z różnych źródeł bez konieczności stosowania skomplikowanych pętli lub serii odrębnych instrukcji w SQL-u. Złączenia i teorię złączeń omówiliśmy szczegółowo w rozdziale 2., „Przygotowywanie danych za pomocą SQL-a”.

Złączenie, jak wskazuje na to nazwa polecenia, przyjmuje dane z co najmniej dwóch tabel i wykorzystuje zawartość rekordów z każdej tabeli do połączenia dwóch zbiorów informacji. Ponieważ odbywa się to bez użycia pętli, operacja może być bardzo wydajna. W tym podrozdziale omawiamy złączenia jako wydajniejszą alternatywę pętli. Na rysunku 6.31 pokazana jest zawartość tabeli Customer Information.

Customer ID	First Name	Last Name	Address
1	Meat	Hook	Melee Island
2	Captain	Blondebeard	Puerto Pollo
3	Griswold	Goodsoup	Blood Island

Rysunek 6.31. Tabela Customer Information

Tabela Order Information jest przedstawiona na rysunku 6.32.

Order ID	Customer ID	Product Code	Qty
1618	3	GROG1	12
1619	2	POULET3	3

Rysunek 6.32. Tabela Order Information

Na podstawie tych informacji należy sprawdzić, czy występują jakieś zależności sprzedawanych produktów od adresów klientów. Możesz skorzystać z polecenia JOIN, aby złączyć oba zbiory informacji. Użyj kolumny Customer ID, aby połączyć oba zbiory danych i uzyskać informacje widoczne w tabeli z rysunku 6.33.

Customer ID	First Name	Last Name	Address	Order ID	Product Code	Qty
2	Captain	Blondebeard	Puerto Pollo	1619	POULET3	3
3	Griswald	Goodsoup	Blood Island	1618	GROG1	12

Rysunek 6.33. Złączanie na podstawie identyfikatora klienta

W tym przykładzie widać, że złączenie obejmuje wszystkie rekordy, w których dostępne są informacje o kliencie i o zamówieniu. Klient Meat Hook został pominięty w połączonych danych, ponieważ nie są dostępne dla niego informacje o zamówieniu. W tym przykładzie używane jest złączenie `INNER JOIN`. Dostępne są jednak różne złączenia. W tej książce omawiamy je wszystkie. Oto przykład ilustrujący zastosowanie wydajnego złączenia `INNER JOIN`:

```
smalljoins=# EXPLAIN ANALYZE SELECT customers.*, order_info.order_id,
order_info.product_code, order_info.qty FROM customers INNER JOIN order_
info ON customers.customer_id=order_info.customer_id;
```

Plan wykonywania tej kwerendy wygląda tak:

```
Hash Join (cost=24.18..172.89 rows=3560 width=140) (actual
time=0.100..0.103 rows=5 loops=1)
  Hash Cond: (order_info.customer_id = customers.customer_id)
    -> Seq Scan on order_info (cost=0.00..21.30 rows=1130 width=44)
    (actual time=0.027..0.027 rows=5 loops=1)
    -> Hash (cost=16.30..16.30 rows=630 width=100) (actual
time=0.032..0.032 rows=5 loops=1)
      Buckets: 1024 Batches: 1 Memory Usage: 9kB
      -> Seq Scan on customers (cost=0.00..16.30 rows=630 width=100)
      (actual time=0.008..0.009 rows=5 loops=1)
    Planning Time: 0.626 ms
    Execution Time: 0.181 ms
(8 rows)
```

Więcej informacji o złączeniach znajdziesz w rozdziale 2., „Przygotowywanie danych za pomocą SQL-a”. W następnym ćwiczeniu przyjrysz się zastosowaniu wydajnych złączeń wewnętrznych.

Ćwiczenie 6.04

— ocenianie zastosowania złączeń wewnętrznych

W tym ćwiczeniu przyjrysz się zastosowaniu złączeń wewnętrznych do wydajnego pobierania wielu wierszy danych z dwóch różnych tabel. Załóżmy, że Twój dobry znajomi z działu marketingu przekazali dwie odrębne bazy danych — z systemu Salesforce i z systemu Oracle. Możesz użyć instrukcji `JOIN`, aby scalić powiązane informacje z tych dwóch źródeł w jedno. Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Utwórz bazę danych `smalljoins` na serwerze PostgreSQL:

```
$ createdb -U postgres smalljoins
```

2. Wczytaj plik `smalljoins.dump` dostępny w kodzie źródłowym książki zamieszczonym w repozytorium w serwisie GitHub (<https://packt.live/3hl6G8Y>):

```
$psql smalljoins < smalljoins.dump
```

3. Otwórz bazę danych:

```
$ psql -U postgres smalljoins
```

4. Zbadaj informacje dostępne w tabeli `customers`:

```
SELECT * FROM customers;
```

Dane wyjściowe tego kodu są pokazane na rysunku 6.34.

customer_id	first_name	last_name	address
4	Guybrush	Threepwood	Melee Island
5	Murray	TheSkull	Plunder island
1	Meat	Hook	Melee Island
2	Captain	Blondebeard	Puerto Pollo
3	Griswold	Goodsoup	Blood Island

(5 rows)

Rysunek 6.34. Tabela `customers`

5. Sprawdź informacje dostępne w tabeli `order_info`:

```
smalljoins=# SELECT * FROM order_info;
```

Ten kod wyświetli dane wyjściowe pokazane na rysunku 6.35.

order_id	customer_id	product_code	qty
1620	4	MON123	1
1621	4	MON636	3
1622	5	MON666	1
1618	3	GROG1	12
1619	2	POULET3	3

(5 rows)

Rysunek 6.35. Tabela z informacjami o zamówieniach

6. Wykonaj złączenie `INNER JOIN`, aby pobrać z obu tabel wszystkie kolumny (ale bez powielania kolumny `customer_id`). Celem jest uzyskanie wyników przedstawionych na rysunku 6.33. Jako lewą tabelę podaj tabelę `customers`, a jako prawą — `order_info`. Aby było jasne: należy pobrać wszystkie kolumny tabeli `customers` oraz kolumny `order_id`, `product_code` i `qty` tabeli `order_info` powiązane z klientem, który złożył dane zamówienie. Zapisz to jako instrukcję SQL-ową:

```
smalljoins=# SELECT customers.*, order_info.order_id, order_info.  
product_code, order_info.qty FROM customers INNER JOIN order_info ON  
customers.customer_id=order_info.customer_id;
```

Rysunek 6.36 przedstawia dane wyjściowe tego kodu.

customer_id	first_name	last_name	address	order_id	product_code	qty
4	Guybrush	Threepwood	Melee Island	1620	MON123	1
4	Guybrush	Threepwood	Melee Island	1621	MON636	3
5	Murray	TheSkull	Plunder island	1622	MON666	1
3	Griswold	Goodsoup	Blood Island	1618	GROG1	12
2	Captain	Blondebeard	Puerto Pollo	1619	POULET3	3

(5 rows)

Rysunek 6.36. Złączanie informacji o klientach i zamówieniach

7. Zapisz wyniki tej kwerendy w odrębnej tabeli. W tym celu dodaj do kwerendy fragment INTO nazwa_tabeli:

```
smalljoins=# SELECT customers.*, order_info.order_id, order_info.
product_code, order_info.qty INTO join_results FROM customers INNER
JOIN order_info ON customers.customer_id=order_info.customer_id;
```

Na rysunku 6.37 pokazane są dane wyjściowe tego kodu.

```
smalljoins=# SELECT customers.*, order_info.order_id, order_info.product_code, order_info.qty INTO join_
results FROM customers INNER JOIN order_info ON customers.customer_id=order_info.customer_id;
```

```
SELECT 5
```

Rysunek 6.37. Zapisywanie wyników złączenia w nowej tabeli

8. Użyj instrukcji EXPLAIN ANALYZE, aby oszacować czas potrzebny na przeprowadzenie złączenia. Jak szybka jest ta operacja?

```
smalljoins=# EXPLAIN ANALYZE SELECT customers.*, order_info.order_id,
order_info.product_code, order_info.qty FROM customers INNER JOIN
order_info ON customers.customer_id=order_info.customer_id;
```

Zobaczysz dane wyjściowe pokazane na rysunku 6.38. Oznaczają one, że kwerenda jest mało wydajna.

```

QUERY PLAN
-----
Hash Join  (cost=24.18..172.89 rows=3560 width=140) (actual time=0.537..0.548 rows=10 loops=1)
  Hash Cond: (order_info.customer_id = customers.customer_id)
    -> Seq Scan on order_info  (cost=0.00..21.30 rows=1130 width=44) (actual time=0.238..0.240 rows=10 loops=1)
    -> Hash  (cost=16.30..16.30 rows=630 width=100) (actual time=0.225..0.226 rows=5 loops=1)
          Buckets: 1024  Batches: 1  Memory Usage: 9kB
          -> Seq Scan on customers  (cost=0.00..16.30 rows=630 width=100) (actual time=0.199..0.202 rows=5 loops=1)
Planning Time: 7.077 ms
Execution Time: 1.533 ms
(8 rows)
```

Rysunek 6.38. Bazowe wartości używane do pomiaru wydajności kwerendy JOIN

9. Pobierz wszystkie wartości kolumny customer_id tabeli order_info i użyj instrukcji EXPLAIN ANALYZE do ustalenia, ile czasu zajmuje wykonanie poszczególnych kwerend:

```
smalljoins=# EXPLAIN ANALYZE SELECT * FROM customers WHERE customer_id IN
(SELECT customer_id FROM order_info);
```

Rysunek 6.39 pokazuje dane wyjściowe tego kodu.

QUERY PLAN

```

Hash Join (cost=28.62..50.08 rows=315 width=100) (actual time=0.104..0.110 rows=4 loops=1)
  Hash Cond: (customers.customer_id = order_info.customer_id)
    -> Seq Scan on customers (cost=0.00..16.30 rows=630 width=100) (actual time=0.015..0.017 rows=5 loops=1)
    -> Hash (cost=26.12..26.12 rows=200 width=4) (actual time=0.057..0.057 rows=4 loops=1)
        Buckets: 1024 Batches: 1 Memory Usage: 9kB
        -> HashAggregate (cost=24.12..26.12 rows=200 width=4) (actual time=0.026..0.030 rows=4 loops=1)
            Group Key: order_info.customer_id
            -> Seq Scan on order_info (cost=0.00..21.30 rows=1130 width=4) (actual time=0.008..0.011 rows=5 loops=1)
Planning Time: 0.199 ms
Execution Time: 0.177 ms
(10 rows)

```

Rysunek 6.39. Wyższa wydajność złączenia dzięki użyciu indeksu z haszowaniem

Na podstawie wyników z dwóch planów wykonywania kwerendy widać, że złączenie wewnętrzne wymaga znacznie mniej czasu niż kwerenda sekwencyjna (**0,177 ms** w porównaniu z **1,533 ms**), a ponadto zwraca więcej informacji (zwracane są też kolumny `order_id`, `product_code` i `qty`).

10. Wykonaj złączenie lewostronne, używając tabeli `customers` jako lewej i tabeli `order_info` jako prawej:

```

smalljoins=# SELECT customers.*, order_info.order_id, order_info.
product_code, order_info.qty FROM customers LEFT JOIN order_info ON
customers.customer_id=order_info.customer_id;

```

Na zrzucie z rysunku 6.40 pokazane są dane wyjściowe tego kodu.

customer_id	first_name	last_name	address	order_id	product_code	qty
4	Guybrush	Threepwood	Melee Island	1620	MON123	1
4	Guybrush	Threepwood	Melee Island	1621	MON635	3
5	Murray	TheSkull	Plunder island	1622	MON666	1
3	Griswold	Goodsoup	Blood Island	1618	GROG1	12
2	Captain	Blondebeard	Puerto Pollo	1619	POULET3	3
1	Meat	Hook	Melee Island			

(6 rows)

Rysunek 6.40. Złączenie lewostronne tabel `customers` i `order_info`

Zwróć uwagę na różnice między złączeniami lewostronnym i wewnętrznym. Złączenie lewostronne dwukrotnie dodało wynik z wartością `customer_id` równą 4 i raz dodało wynik dla klienta `Meat Hook`, choć nie są z nim powiązane żadne informacje o zamówieniach. W wierszu z danymi tego klienta podane są informacje z lewej tabeli, a w prawej tabeli występują puste pola.

11. Użyj instrukcji `EXPLAIN ANALYZE`, aby określić czas i koszt wykonania złączenia:

```

smalljoins=# EXPLAIN ANALYZE SELECT customers.*, order_info.order_id,
order_info.product_code, order_info.qty FROM customers LEFT JOIN
order_info ON customers.customer_id=order_info.customer_id;

```

Ten kod wyświetli dane wyjściowe z rysunku 6.41.

12. Zastąp złączenie lewostronne z kroku 11. złączeniem prawostronnym i przyjrzyj się wynikom:

```

smalljoins=# EXPLAIN ANALYZE SELECT customers.*, order_info.order_id,
order_info.product_code, order_info.qty FROM customers RIGHT JOIN
order_info ON customers.customer_id=order_info.customer_id;

```

QUERY PLAN

```

Hash Right Join (cost=24.18..172.89 rows=3560 width=140) (actual time=0.068..0.089 rows=6 loops=1)
Hash Cond: (order_info.customer_id = customers.customer_id)
-> Seq Scan on order_info (cost=0.00..21.30 rows=1130 width=44) (actual time=0.007..0.009 rows=5 loops=1)
-> Hash (cost=16.30..16.30 rows=630 width=100) (actual time=0.034..0.034 rows=5 loops=1)
    Buckets: 1024 Batches: 1 Memory Usage: 9kB
    -> Seq Scan on customers (cost=0.00..16.30 rows=630 width=100) (actual time=0.020..0.024 rows=5 loops=1)
Planning Time: 0.219 ms
Execution Time: 0.188 ms
(8 rows)

```

Rysunek 6.41. Plan wykonywania kwerendy ze złączeniem lewostronnym

Rysunek 6.42 przedstawia dane wyjściowe tego kodu.

customer_id	first_name	last_name	address	order_id	product_code	qty
4	Guybrush	Threepwood	Melee Island	1620	MON123	1
4	Guybrush	Threepwood	Melee Island	1621	MON636	3
5	Murray	TheSkull	Plunder island	1622	MON666	1
3	Griswold	Goodsoup	Blood Island	1618	GROG1	12
2	Captain	Blondebeard	Puerto Pollo	1619	POULET3	3

(5 rows)

Rysunek 6.42. Wyniki złączenia prawostronnego

Także tu znajdują się dwa wiersze dotyczące klienta Guybrush Threepwood (kolumna `customer_id` równa 4), ale wiersz z danymi klienta Meat Hook (kolumna `customer_id` równa 1) nie jest już obecny, ponieważ złączenie odbyło się na podstawie zawartości tabeli `order_info`.

13. Użyj instrukcji `EXPLAIN ANALYZE`, aby ocenić czas i koszt przeprowadzenia złączenia prawostronnego:

```

smalljoins=# EXPLAIN ANALYZE SELECT customers.*, order_info.order_id,
order_info.product_code, order_info.qty FROM customers RIGHT JOIN
order_info ON customers.customer_id=order_info.customer_id;

```

Zrzut z rysunku 6.43 przedstawia dane wyjściowe tego kodu.

QUERY PLAN

```

Hash Left Join (cost=24.18..172.89 rows=3560 width=140) (actual time=0.066..0.075 rows=5 loops=1)
Hash Cond: (order_info.customer_id = customers.customer_id)
-> Seq Scan on order_info (cost=0.00..21.30 rows=1130 width=44) (actual time=0.022..0.024 rows=5 loops=1)
-> Hash (cost=16.30..16.30 rows=630 width=100) (actual time=0.021..0.022 rows=5 loops=1)
    Buckets: 1024 Batches: 1 Memory Usage: 9kB
    -> Seq Scan on customers (cost=0.00..16.30 rows=630 width=100) (actual time=0.007..0.012 rows=5 loops=1)
Planning Time: 0.220 ms
Execution Time: 0.141 ms
(8 rows)

```

Rysunek 6.43. Plan wykonywania kwerendy ze złączeniem prawostronnym

Widać tu, że złączenie prawostronne było minimalnie szybsze i bardziej efektywne kosztowo, co można przypisać temu, że zwrócony został o jeden wiersz mniej niż w złączeniu lewostronnym.

14. Wstaw do tabeli `order_info` dodatkowy wiersz z wartością pola `customer_id`, która nie występuje w tabeli `customers`:

```

smalljoins=# INSERT INTO order_info (order_id, customer_id, product_code, qty) VALUES
(1621, 6, 'MEL386', 1);

```

15. Zastąp złączenie lewostronne z kroku 11. pełnym złączeniem zewnętrznym i przyjrzyj się wynikom:

```
smalljoins=# SELECT customers.*, order_info.order_id, order_info.product_code,
order_info.qty FROM customers FULL OUTER JOIN order_info
ON customers.customer_id=order_info.customer_id;
```

Ten kod wyświetli następujące dane wyjściowe:

customer_id	first_name	last_name	address	order_id	product_code	qty
4	Guybrush	Threepwood	Melee Island	1620	MON123	1
4	Guybrush	Threepwood	Melee Island	1621	MON636	3
5	Murray	TheSkull	Plunder island	1622	MON666	1
3	Griswold	Goodsoup	Blood Island	1618	GROG1	12
2	Captain	Blondebeard	Puerto Pollo	1619	POULET3	3
				1621	MEL386	1
1	Meat	Hook	Melee Island			

(7 rows)

Rysunek 6.44. Wyniki pełnego złączenia zewnętrznego

Zwróć uwagę na wiersz zawierający pole `product_code` o wartości `MEL386`, ale bez informacji o kliencie. Jest to sytuacja podobna jak z wierszem z danymi klienta `Meat Hook` (pole `customer_id` o wartości 1). Pełne złączenie zewnętrzne zwraca wszystkie dostępne informacje, nawet jeśli w jednej z tabel brakuje niektórych informacji.

16. Użyj instrukcji `EXPLAIN ANALYZE`, aby ocenić wydajność kwerendy:

```
smalljoins=#
```

Zrzut z rysunku 6.45 przedstawia dane wyjściowe tego kodu.

```

QUERY PLAN
-----
Hash Full Join  (cost=24.18..172.89 rows=3560 width=140) (actual time=0.126..0.148 rows=7 loops=1)
  Hash Cond: (order_info.customer_id = customers.customer_id)
    -> Seq Scan on order_info  (cost=0.00..21.30 rows=1130 width=44) (actual time=0.009..0.012 rows=6 loops=1)
    -> Hash  (cost=16.30..16.30 rows=630 width=100) (actual time=0.064..0.065 rows=5 loops=1)
          Buckets: 1024  Batches: 1  Memory Usage: 9kB
          -> Seq Scan on customers  (cost=0.00..16.30 rows=630 width=100) (actual time=0.021..0.026 rows=5 loops=1)
Planning Time: 0.226 ms
Execution Time: 0.232 ms
(8 rows)

```

Rysunek 6.45. Plan wykonywania kwerendy z pełnym złączeniem zewnętrznym

Wydajność tej kwerendy jest bardzo zbliżona do pozostałych, jeśli uwzględnić dodatkowo wiersz widoczny w ostatecznych danych wyjściowych.

Kod źródłowy z tego fragmentu znajdziesz na stronie <https://packt.live/2WYKQPI>.

W tym ćwiczeniu pokazaliśmy stosowanie złączeń i dowiedliśmy, że zwiększają one wydajność kodu. Zobaczyłeś, że łączenie informacji z dwóch odrębnych tabel wymaga mniej zasobów niż odrębne operacje wyszukiwania. Przedstawiliśmy również zastosowanie złączenia `OUTER JOIN` do wydajnego połączenia wszystkich informacji. W następnym zadaniu wykorzystasz znajomość złączeń do dużo większego zbioru danych.

Zadanie 6.04 — stosowanie wydajnych złączeń

W tym zadaniu Twoim celem jest zastosowanie różnych wydajnych złączeń. Użyjesz ich do łączenia informacji z tabeli `customers` i ze zbioru danych dotyczącego e-maili marketingowych. Załóżmy, że zebrałeś różne rekordy z informacjami o e-mailach z rozmaitych baz danych. Chcesz zapisać informacje w jednej tabeli, aby umożliwić przeprowadzenie szczegółowych analiz. Oto kroki niezbędne do wykonania tego zadania:

1. Otwórz PostgreSQL i nawiąż połączenie z bazą `sql`da.
2. Ustal listę klientów (z polami `customer_id`, `first_name` i `last_name`), do których wysłano e-mail. Uwzględnij informacje o temacie e-maila oraz o tym, czy odbiorca otworzył i kliknął wiadomość. Wynikowa tabela powinna zawierać kolumny `customer_id`, `first_name`, `last_name`, `email_subject`, `opened` i `clicked`.
3. Zapisz wynikową tabelę w nowej tabeli, `customer_emails`.
4. Znajdź klientów, którzy otworzyli lub kliknęli e-mail.
5. Znajdź klientów, w których mieście znajduje się salon. Klienci, w których miejscu zamieszkania nie ma salonu, powinni mieć pustą wartość w kolumnie `city`.
6. Wyświetl klientów, którzy nie mają salonu w swoim mieście (wskazówka: puste pole ma wartość `NULL`).

Rysunek 6.46 przedstawia oczekiwane dane wyjściowe.

<code>customer_id</code>	<code>first_name</code>	<code>last_name</code>	<code>city</code>
1	Arlena	Riveles	
12	Tyne	Duggan	
21	Pryce	Geist	
24	Barbi	Lanegran	
30	Kath	Rivel	
38	Carter	Lagneaux	
44	Waldemar	Paroni	
49	Hannah	McGlew	
56	Riva	Cathesyed	
63	Gweneth	Maier	
70	Caty	Woolveridge	
72	Jodi	Fautly	

Rysunek 6.46. Klienci bez informacji o mieście

Dane wyjściowe przedstawiają końcową listę klientów z miast, w których nie ma salonów.

Rozwiązanie tego zadania znajdziesz w „Dodatku”.

W tym zadaniu zastosowaliśmy złączenia, aby powiązać informacje z tabeli `customers` z informacjami ze zbioru danych dotyczącego e-maili marketingowych i pomóc menedżerowi działu marketingu w opracowaniu kwerendy.

Z następnego podrozdziału dowiesz się, jak używać funkcji i wyzwalaczy w kwerendach SQL-owych oraz jak analizować dane.

Funkcje i wyzwalacze

Wiesz już, jak oceniać wydajność kwerend za pomocą planera kwerend, a także jakie są korzyści stosowania złączeń do scalania i pobierania informacji z wielu tabel bazy danych. W tym podrozdziale posłużysz się funkcjami do opracowania kwerend i instrukcji wielokrotnego użytku, a także nauczysz się automatycznie uruchamiać funkcje za pomocą wyzwalaczy. Korzystając z połączenia tych dwóch mechanizmów SQL-a, możesz nie tylko uruchamiać kwerendy i ponownie indeksować tabele po dodaniu, zaktualizowaniu lub usunięciu danych w bazie, ale też sprawdzać hipotezy i śledzić wyniki w różnych okresach użytkowania bazy.

Definicje funkcji

Funkcje w SQL-u, podobnie jak w prawie każdym innym języku programowania lub języku skryptowym, to wyodrębnione bloki kodu. Funkcje mają wiele zalet, na przykład umożliwiają wydajne ponowne wykorzystanie kodu i upraszczają proces rozwiązywania problemów. Za pomocą funkcji możesz powtarzać i modyfikować instrukcje i kwerendy bez konieczności każdorazowego wprowadzania poleceń lub wyszukiwania określonego bloku kodu w długich fragmentach. Jednym z najprzydatniejszych aspektów funkcji jest to, że pozwalają dzielić kod na mniejsze, możliwe do testowania porcje. W środowisku informatyków mówi się, że jeśli kod nie został przetestowany, nie można mu ufać.

Jak więc definiuje się funkcje w SQL-u? Służy do tego dość prosta składnia ze słowami kluczowymi SQL-a:

```
CREATE FUNCTION nazwa_funkcji (argumenty_funkcji)
RETURNS typ_zwracanej_wartości AS $nazwa_zwracanej_wartości$
DECLARE nazwa_zwracanej_wartości typ_zwracanej_wartości;
BEGIN
    <instrukcje_funkcji>;
RETURN <jakaś_wartość>;
END; $nazwa_zwracanej_wartości$
LANGUAGE PLPGSQL;
```

Oto krótkie objaśnienie elementów funkcji użytych w tym kodzie:

- `nazwa_funkcji` to nazwa przypisana funkcji i służąca do późniejszego jej uruchamiania.
- `argumenty_funkcji` to opcjonalna lista argumentów funkcji. Może być pusta (nie zawierać żadnych argumentów), jeśli funkcja nie wymaga podawania dodatkowych informacji. Aby przekazać dodatkowe informacje, możesz użyć albo listy wartości różnych typów danych (na przykład całkowitoliczbowych i innych typów liczbowych), albo listy argumentów z nazwami parametrów (na przykład `min_val` typu całkowitoliczbowego i `max_val` liczbowego typu danych).
- `typ_zwracanej_wartości` to typ danych zwracany przez funkcję.

- nazwa_zwracanej_wartości to (opcjonalna) nazwa zwracanej zmiennej. Instrukcja DECLARE nazwa_zwracanej_wartości typ_zwracanej_wartości jest potrzebna tylko wtedy, jeśli podana jest nazwa_zwracanej_wartości i funkcja zwraca zmienną. Jeżeli nazwa_zwracanej_wartości nie jest potrzebna, ten wiersz definicji funkcji można pominąć.
- instrukcje funkcji to SQL-owe instrukcje wykonywane w funkcji.
- jakaś_wartość to dane zwracane przez funkcję.
- Człon PLPGSQL określa język używany w funkcji. PostgreSQL umożliwia korzystanie także z innych języków, jednak omawianie ich stosowania w tym kontekście wykracza poza zakres tej książki.

Możesz na przykład utworzyć prostą funkcję do dodawania trzech liczb:

```
CREATE FUNCTION add_three(a integer, b integer, c integer)
RETURNS integer AS $$
BEGIN
    RETURN a + b + c;
END;
$$ LANGUAGE PLPGSQL;
```

Następnie możesz wywoływać ją w kwerendach w następujący sposób:

```
SELECT add_three(1, 2, 3);
```

Oto dane wyjściowe tego kodu:

```
add_three
-----
          6
(1 row)
```

Teraz wykonasz ćwiczenie, w którym utworzysz funkcję nieprzyjmującą argumentów.

Kompletną dokumentację funkcji w systemie PostgreSQL znajdziesz na stronie <https://www.postgresql.org/docs/current/extend.html>.

Ćwiczenie 6.05

— tworzenie funkcji, które nie przyjmują argumentów

W tym ćwiczeniu utworzysz najprostszą możliwą funkcję, która jedynie zwraca stałą wartość. Pozwoli Ci to zapoznać się ze składnią funkcji. Utworzysz swoją pierwszą funkcję w SQL-u. Nie będzie ona przyjmować żadnych dodatkowych informacji w postaci argumentów. Tego rodzaju funkcji możesz używać na przykład do wielokrotnego zwracania wyników kwerend SQL-a, które obliczają podstawowe statystyki na temat danych z tabel bazy sqlda. Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Nawiąż połączenie z bazą `sqla`, używając wiersza poleceń:

```
$ psql sqla
```

2. Utwórz funkcję `fixed_val`, która nie przyjmuje żadnych argumentów i zwraca liczbę całkowitą. Wymaga to wprowadzenia kilku wierszy kodu. Najpierw wpisz następujący wiersz:

```
sqla=# CREATE FUNCTION fixed_val() RETURNS integer AS $$
```

Ten wiersz rozpoczyna deklarację funkcji `fixed_val`. Widać tu, że nie przyjmuje ona żadnych argumentów (wskazują na to puste nawiasy otwierający i zamykający, `()`) oraz nie zwraca żadnych zmiennych.

3. Zauważ, że w następnym wierszu zmienia się znak zachęty; jest to informacja, że system oczekuje następnego wiersza funkcji:

```
sqla$#
```

4. Wprowadź słowo kluczowe `BEGIN`. Zauważ, że ponieważ ta funkcja nie zwraca zmiennej, wiersz zawierający instrukcję `DECLARE` został pominięty:

```
sqla$# BEGIN
```

5. Ta funkcja ma zwracać wartość 1, dlatego wpisz instrukcję `RETURN 1`:

```
sqla$# RETURN 1;
```

6. Zakończ definicję funkcji:

```
sqla$# END; $$
```

7. Dodaj instrukcję `LANGUAGE`, tak jak w tej definicji:

```
sqla-# LANGUAGE PLPGSQL;
```

To kończy definicję funkcji.

8. Teraz, gdy funkcja jest już zdefiniowana, można jej użyć. Podobnie jak w prawie wszystkich innych instrukcjach SQL-a, jakie wykonywaliśmy do tego miejsca, zastosujemy polecenie `SELECT`:

```
sqla=# SELECT * FROM fixed_val();
```

Spowoduje to wyświetlenie następujących danych wyjściowych:

```
fixed_val
-----
1
(1 row)
```

Zauważ, że funkcja jest wywoływana w instrukcji `SELECT` przez podanie nawiasów (otwierającego i zamykającego).

9. Użyj instrukcji `EXPLAIN` i `ANALYZE` do przedstawionego polecenia, aby sprawdzić wydajność funkcji:

```
sqla=# EXPLAIN ANALYZE SELECT * FROM fixed_val();
```

Na rzucie z rysunku 6.47 pokazane są dane wyjściowe tego kodu.

QUERY PLAN

```
Function Scan on fixed_val (cost=0.25..0.26 rows=1 width=4) (actual time=0.031..0.032 rows=1 loops=1)
Planning Time: 0.060 ms
Execution Time: 0.060 ms
(3 rows)
```

Rysunek 6.47. Wydajność wywołania funkcji

Zobaczyłeś już, jak utworzyć prostą funkcję. Jednak samo zwracanie stałej wartości nie jest zbyt przydatne. Teraz utworzysz funkcję, która zwraca liczbę obserwacji z tabeli sales. Zauważ, że trzy wiersze (3 rows) na rysunku 6.47 nie dotyczą wyniku kwerendy `SELECT * FROM fixed_val()`, lecz wyniku działania planera kwerend. Gdy przyjrzyj się pierwszemu wierszowi informacji zwróconych przez planer kwerend, zobaczysz, że wspomniana instrukcja `SELECT` zwraca tylko jeden wiersz informacji.

10. Utwórz funkcję `num_samples`, która nie przyjmuje żadnych argumentów, ale zwraca liczbę całkowitą `total` reprezentującą liczbę obserwacji z tabeli sales:

```
sql># CREATE FUNCTION num_samples() RETURNS integer AS $total$
```

11. Funkcja ma zwracać zmienną `total`, dlatego trzeba ją zadeklarować. Zadeklaruj zmienną `total` typu `integer`:

```
sql># DECLARE total integer;
```

12. Dodaj słowo kluczowe `BEGIN`:

```
sql># BEGIN
```

13. Wprowadź instrukcję, która określa liczbę obserwacji w tabeli i przypisuje wynik do zmiennej `total`:

```
sql># SELECT COUNT(*) INTO total FROM sales;
```

14. Zwróć wartość zmiennej `total`:

```
sql># RETURN total;
```

15. Zakończ funkcję, podając nazwę zmiennej:

```
sql># END; $total$
```

16. Dodaj instrukcję `LANGUAGE`, tak jak w tej definicji:

```
sql># LANGUAGE PLPGSQL;
```

To kończy definicję funkcji. Po utworzeniu funkcji zobaczysz informację `CREATE FUNCTION`.

17. Użyj funkcji do określenia, ile wierszy (obserwacji) znajduje się w tabeli sales:

```
sql># SELECT num_samples();
```

Oto dane wyjściowe tego kodu:

```
num_samples
-----
37711
(1 row)
```

Dzięki użyciu instrukcji `SELECT` razem z nową funkcją SQL-ową można stwierdzić, że tabela sales zawiera 37 711 rekordów.

Kod źródłowy z tego fragmentu książki jest dostępny na stronie <https://packt.live/2zqBi7H>.

W tym ćwiczeniu utworzyłeś swoją pierwszą funkcję SQL-ową definiowaną przez użytkownika i zobaczyłeś, jak generować i zwracać informacje ze zmiennych w takiej funkcji.

W następnym zadaniu utworzysz nową funkcję, którą można wywoływać w kwerendach.

Zadanie 6.05 — definiowanie funkcji zwracającej maksymalną wartość sprzedaży

W tym zadaniu utworzysz zdefiniowaną przez użytkownika funkcję, która umożliwi obliczanie największej wartości sprzedaży za pomocą jednego wywołania. Dzięki temu utrwalisz wiedzę na temat funkcji. Funkcja, którą dodasz, będzie podawać najwyższą wartość sprzedaży w bazie danych. Dział marketingu zaczyna przekazywać Ci wiele próśb związanych z analizą danych i musisz szybciej je realizować, ponieważ obecnie prace zajmują za dużo czasu. Oto kroki niezbędne do wykonania tego zadania:

1. Nawiąż połączenie z bazą `sql`da.
2. Utwórz funkcję `max_sale`, która nie przyjmuje argumentów wejściowych, ale zwraca wartość liczbową `big_sale`.
3. Zadeklaruj zmienną `big_sale` i rozpocznij funkcję.
4. Przypisz maksymalną wartość sprzedaży do zmiennej `big_sale`.
5. Zwróć wartość zmiennej `big_sale`.
6. Zakończ funkcję instrukcją `LANGUAGE`.
7. Wywołaj funkcję, aby stwierdzić, jaka jest najwyższa wartość sprzedaży w bazie.

Oto oczekiwane dane wyjściowe:

```
Max
-----
115000
(1 row)
```

Rozwiązanie tego zadania znajdziesz w „Dodatku”.

W tym zadaniu utworzyłeś definiowaną przez użytkownika funkcję do obliczania najwyższej wartości sprzedaży za pomocą jednego wywołania funkcji. Użyłeś do tego funkcji `MAX`. Teraz utworzysz funkcję, która przyjmuje argumenty.

Ćwiczenie 6.06

— tworzenie funkcji przyjmujących argumenty

W tym ćwiczeniu utworzysz jedną funkcję, która umożliwia wykonywanie obliczeń na podstawie informacji przekazywanych za pomocą parametrów. Utwórz funkcję do obliczania średniej wartości z kolumny `amount` na podstawie podanego kanału sprzedaży. Po utworzeniu poprzedniej funkcji zdefiniowanej przez użytkownika, zwracającej najwyższą wartość sprzedaży, zaobserwowałeś znaczny wzrost szybkości odpowiadania na prośby od działu marketingu.

Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Nawiąż połączenie z bazą `sql`da:

```
$ psql sqlda
```

2. Utwórz funkcję `avg_sales`, która przyjmuje argument tekstowy, `channel_type`, i zwraca wartość liczbową:

```
sqlda=# CREATE FUNCTION avg_sales(channel_type TEXT) RETURNS numeric
AS $channel_avg$
```

3. Zadeklaruj zmienną liczbową `channel_avg` i rozpocznij definicję funkcji:

```
sqlda$# DECLARE channel_avg numeric;
sqlda$# BEGIN
```

4. Określ średnią wartość z kolumny `sales_amount` dla wierszy, w których wartość kolumny `channel` jest równa `channel_type`:

```
sqlda$# SELECT AVG(sales_amount) INTO channel_avg FROM sales WHERE
channel=channel_type;
```

5. Zwróć wartość `channel_avg`:

```
sqlda$# RETURN channel_avg;
```

6. Zakończ definicję funkcji i dodaj instrukcję `LANGUAGE`:

```
sqlda$# END; $channel_avg$
sqlda=# LANGUAGE PLPGSQL;
```

7. Określ średnią wartość sprzedaży dla kanału `internet`:

```
sqlda=# SELECT avg_sales('internet');
```

Oto dane wyjściowe tego kodu:

```
avg_sales
-----
6413.11540412024
(1 row)
```

Teraz zrób to samo dla kanału `dealership`:

```
sqlda=# SELECT avg_sales('dealership');
```

Oto dane wyjściowe tego kodu:

```
avg_sales
-----
7939.33132075954
(1 row)
```

W tych danych wyjściowych widać średni poziom sprzedaży dla salonu. Wynosi on około 7939,331.

Kod źródłowy z tego fragmentu książki znajdziesz na stronie <https://packt.live/3hrZITf>.

W tym ćwiczeniu zapoznałeś się ze stosowaniem argumentów funkcji do modyfikowania działania funkcji i zwracanych danych wyjściowych. Dalej poznasz polecenia `\df` i `\sf`.

Polecenia `\df` i `\sf`

W PostgreSQL możesz używać polecenia `\df` do pobierania listy funkcji dostępnych w pamięci oraz zmiennych i typów danych argumentów (rysunek 6.48).

List of functions				
Schema	Name	Result data type	Argument data types	Type
public	avg_sales	numeric	channel_type text	func
public	avg_sales_since	numeric	since_date date	func
public	fixed_val	integer		func
public	max_sale	numeric		func
public	num_samples	integer		func
(5 rows)				

Rysunek 6.48. Wynik wywołania polecenia `\df` dla bazy danych `sqllda`

Polecenie `\sf nazwa_funkcji` w PostgreSQL pozwala wyświetlić definicję już istniejącej funkcji.

Przyjmij, że dostępna jest funkcja `num_samples`. Możesz wywołać następujące polecenie:

```
\sf num_samples
```

Dane wyjściowe (rysunek 6.49) zawierają definicję tej funkcji.

```
CREATE OR REPLACE FUNCTION public.num_samples()
  RETURNS integer
  LANGUAGE plpgsql
  AS $function$
  DECLARE total integer;
  BEGIN
    SELECT COUNT(*) INTO total FROM sales;
    RETURN total;
  END; $function$
```

Rysunek 6.49. Kod funkcji wyświetlony za pomocą polecenia `\sf`

Po wykonaniu kilku ćwiczeń, w których tworzyliśmy funkcje z argumentami i bez nich, możesz wykorzystać zdobytą wiedzę w praktycznych problemach. W następnym zadaniu przećwiczysz tworzenie funkcji przyjmujących argumenty.

Zadanie 6.06

— tworzenie funkcji przyjmujących argumenty

W tym zadaniu celem jest utworzenie funkcji z argumentami i obliczenie danych wyjściowych. Utworzysz funkcję, która oblicza średnią wartość sprzedaży dla transakcji z podanego okresu. Dаты początkową i końcową trzeba przekazać do funkcji jako tekst. Oto kroki niezbędne do wykonania tego zadania:

1. Utwórz definicję funkcji `avg_sales_window`, która zwraca wartość liczbową i przyjmuje dwie wartości DATE w formacie RRRR-MM-DD określające daty początkową i końcową.
2. Zadeklaruj zwracaną zmienną typu liczbowego i rozpocznij funkcję.
3. Pobierz średnią wartość sprzedaży i przypisz ją do zwracanej zmiennej; uwzględnij transakcje zawarte w określonym przedziale czasu.
4. Zwróć zmienną, zakończ funkcję i dodaj instrukcję LANGUAGE.
5. Użyj funkcji do obliczenia średniej wartości sprzedaży z okresu od 2013-04-12 do 2014-04-12.

Oto oczekiwane dane wyjściowe:

```
avg_sales_window
-----
477.6862463110066
(1 row)
```

Rozwiązanie tego zadania znajdziesz w „Dodatku”.

W tym zadaniu utworzyliśmy funkcję, która oblicza średnią wartość sprzedaży w bazie dla transakcji o datach z określonego przedziału.

W następnym punkcie nauczysz się tworzyć i uruchamiać wyzwalacze, aby zautomatyzować pracę bazy. Wykonasz też ćwiczenie i zadanie z wykorzystaniem wyzwalaczy.

Wyzwalacze

Wyzwalacze (w innych językach programowania nazywane zdarzeniami lub wywołaniami zwrotnymi) to przydatny mechanizm, który — jak wskazuje nazwa — wyzwalają uruchomienie instrukcji lub funkcji SQL-owych w reakcji na określone wydarzenie. Wyzwalacze mogą być aktywowane w wyniku następujących zdarzeń:

- wstawienie wiersza do tabeli;
- aktualizacja wiersza w tabeli;
- usunięcie wiersza z tabeli;
- wyzerowanie zawartości tabeli (szybkie usunięcie z tabeli wszystkich wierszy).

Można też podać moment uruchomienia wyzwalacza:

- przed operacją wstawienia, aktualizacji, usunięcia lub wyzerowania;
- po operacji wstawienia, aktualizacji, usunięcia lub wyzerowania;
- zamiast operacji wstawienia, aktualizacji, usunięcia lub wyzerowania.

W zależności od kontekstu i przeznaczenia bazy danych wyzwalacze mogą mieć rozmaite zastosowania. W środowisku produkcyjnym, gdzie baza służy do przechowywania informacji biznesowych i podejmowania decyzji (na przykład w aplikacji do obsługi wspólnych przejazdów lub w sklepie internetowym), wyzwalacze mogą być używane przed każdą operacją do zapisywania wpisów w dziennikach dostępu do bazy.

Te dzienniki można później wykorzystać do ustalenia, kto używał danych z bazy lub kto je modyfikował. Wyzwalacze pozwalają też przekierować operacje do innej bazy danych lub tabeli. Służą do tego wyzwalacze `INSTEAD OF`.

W analizie danych wyzwalaczy można używać na przykład do tworzenia zbiorów danych (na przykład w celu ustalenia zmian średniej wartości danych w czasie lub różnicy między próbkami), testowania hipotez dotyczących danych albo oznaczania wartości odstających wstawianych do bazy lub w niej modyfikowanych.

Ponieważ wyzwalacze są często stosowane do wykonywania instrukcji SQL-owych w reakcji na zdarzenia lub działania, łatwo jest zrozumieć, dlaczego funkcje są pisane specjalnie na potrzeby wyzwalaczy lub z nimi łączone. Niezależne, powtarzane bloki kodu w funkcjach można wykorzystać zarówno w testach i do debugowania, jak i do podawania kodu uruchamianego przez wyzwalacze. Jak odbywa się tworzenie wyzwalaczy? Podobnie jak w przypadku definicji funkcji służy do tego standardowa składnia ze słowami kluczowymi SQL-a:

```
CREATE TRIGGER nazwa_wyzwalacza { BEFORE | AFTER | INSTEAD OF } { INSERT
| DELETE | UPDATE | TRUNCATE } ON nazwa_tabeli
FOR EACH { ROW | STATEMENT }
EXECUTE PROCEDURE nazwa_funkcji ( argumenty_funkcji )
```

W tej ogólnej definicji wyzwalacza widocznych jest kilka elementów:

- Trzeba podać nazwę wyzwalacza zamiast `nazwa_wyzwalacza`.
- Należy określić, kiedy wyzwalacz jest uruchamiany: przed zdarzeniem (`BEFORE`), po nim (`AFTER`) czy zamiast niego (`INSTEAD OF`).
- Trzeba określić, jakie zdarzenie ma powodować uruchomienie wyzwalacza: wstawianie (`INSERT`), usuwanie (`DELETE`), aktualizowanie (`UPDATE`) czy zerowanie (`TRUNCATE`).
- Zamiast `nazwa_tabeli` należy podać tabelę, której dotyczą monitorowane zdarzenia.
- Instrukcja `FOR EACH` pozwala określić, w jaki sposób wyzwalacz ma być uruchamiany. Można go wykonywać dla każdego wiersza w zasięgu wyzwalacza (opcja `ROW`) lub tylko raz dla instrukcji niezależnie od liczby wierszy wstawianych do tabeli (opcja `STATEMENT`).
- Należy też podać nazwę funkcji (`nazwa_funkcji`) i odpowiednie (wymagane) argumenty funkcji (`argumenty_funkcji`), aby określić kod używany w każdym wyzwalaczu.

Oto kilka funkcji, z których będziesz tu korzystać:

- Funkcja `get_stock` przyjmuje kod produktu jako dane wejściowe typu `TEXT` i zwraca aktualnie dostępną liczbę sztuk danego towaru.
- Funkcja `insert_order` służy do dodawania nowych zamówień do tabeli `order_info`. Przyjmuje argumenty `customer_id` typu `INTEGER`, `product_code` typu `TEXT` i `qty` typu `INTEGER`, a zwraca wartość `order_id` wygenerowaną dla nowego rekordu.
- Funkcja `update_stock` pobiera informacje z najnowszego zamówienia i na podstawie wartości `product_code` aktualizuje liczbę sztuk odpowiednich produktów w tabeli `products`.

Przyjrzyj się następnemu przykładowi, w którym kod wykonuje test chroniący przed przypadkową sprzedażą produktu za cenę niższą niż połowa podstawowej ceny katalogowej. Przed utworzeniem wyzwalacza należy zdefiniować uruchamianą w nim funkcję:

```
CREATE OR REPLACE FUNCTION check_sale_amt_vs_msrp()
RETURNS TRIGGER AS $$
DECLARE min_allowed_price numeric;
BEGIN
    SELECT base_msrp * 0.5 INTO min_allowed_price FROM products WHERE
    product_id = NEW.product_id;
    IF NEW.sales_amount < min_allowed_price THEN
        RAISE EXCEPTION
            'Cena sprzedaży nie może być niższa niż połowa ceny katalogowej';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;
```

Następnie trzeba utworzyć wyzwalacz uruchamiany w momencie dodawania lub aktualizowania rekordu:

```
CREATE TRIGGER sales_product_sales_amount_msrp AFTER INSERT OR UPDATE ON
sales
FOR EACH ROW
EXECUTE PROCEDURE check_sale_amt_vs_msrp()
```

Możesz sprawdzić, czy wyzwalacz działa. W tym celu spróbuj wstawić do tabeli `sales` dane, które nie spełniają warunku minimalnej ceny:

```
INSERT INTO sales (SELECT customer_id, product_id,
sales_transaction_date, sales_amount/3.0, channel, dealership_id FROM sales LIMIT 1);
```

Otrzymasz następujące dane wyjściowe:

```
ERROR: BŁĄD: Cena sprzedaży nie może być niższa niż połowa ceny katalogowej
CONTEXT: funkcja PL/pgSQL check_sale_amt_vs_msrp(), wiersz 7 w RAISE
```

Teraz wykonaj ćwiczenie, aby utworzyć wyzwalacze do aktualizowania pól.

Dostępnych jest też wiele innych opcji związanych z wyzwalaczami SQL-a, ale ich omawianie wykracza poza zakres tej książki. Kompletną dokumentację wyzwalaczy znajdziesz na stronie <https://www.postgresql.org/docs/current/sql-createtrigger.html>.

Ćwiczenie 6.07

— tworzenie wyzwalaczy do aktualizowania pól

W tym ćwiczeniu utworzysz wyzwalacz aktualizujący pola po dodaniu danych. Użyjesz tu bazy danych `smalljoins` z punktu „Wydajne złączenia” tego rozdziału i utworzysz wyzwalacz, który aktualizuje liczbę sztuk produktu w tabeli `products` za każdym razem, gdy do tabeli `order_info` wstawiane jest zamówienie.

Za pomocą takiego wyzwalacza można aktualizować analizy w czasie rzeczywistym, gdy użytkownicy korzystają z bazy danych. Wyzwalacze sprawiają, że nie trzeba ręcznie przeprowadzać analiz dla działu marketingu, gdyż wygenerują potrzebne wyniki.

W tym ćwiczeniu utworzysz wyzwalacz do aktualizowania rekordów z liczbą sztuk produktów. Po zakupie produktów wyzwalacze zostaną aktywowane i zaktualizują liczbę dostępnych sztuk. Oto kroki, jakie należy wykonać:

1. Wczytaj przygotowane funkcje do bazy danych `smalljoins`, używając pliku *Functions.sql* (*Funkcje.sql* w wersji spolszczonej), który znajdziesz w kodzie źródłowym powiązanym z książką. Ten kod jest dostępny także w serwisie GitHub (<https://packt.live/2BS3Yr0>).

```
$ psql smalljoins < Functions.sql
```

2. Nawiąż połączenie z bazą danych `smalljoins`:

```
$ psql smalljoins postgres
```

3. Po wczytaniu definicji funkcji pobierz ich listę za pomocą polecenia `\df`:

```
smalljoins=# \df
```

To polecenie wyświetli dane wyjściowe widoczne na rysunku 6.50.

List of functions				
Schema	Name	Result data type	Argument data types	Type
public	get_stock	integer	text	func
public	insert_order	integer	integer, text, integer	func
public	update_stock	integer		func
(3 rows)				

Rysunek 6.50. Lista funkcji

4. Najpierw przyjrzyj się aktualnemu stanowi tabeli `products`:

```
smalljoins=# SELECT * FROM products;
```

Na rysunku 6.51 pokazane są dane wyjściowe tego kodu.

product_code	name	stock
MON636	Red Herring	99
GROG1	Grog	65
POULET3	El Pollo Diablo	2
MON123	Rubber Chicken + Pulley	7
MON666	Murray"s Arm	0
(5 rows)		

Rysunek 6.51. Lista produktów

Możesz teraz napisać kwerendę dotyczącą tabeli order_info:

```
smalljoins=# SELECT * FROM order_info;
```

Na rysunku 6.52 pokazane są dane wyjściowe tego kodu:

order_id	customer_id	product_code	qty
1618	3	GROG1	12
1619	2	POULET3	3
1620	4	MON123	1
1621	4	MON636	3
1622	5	MON666	1
(5 rows)			

Rysunek 6.52. Lista informacji o zamówieniach

5. Wstaw nowe zamówienie, używając funkcji insert_order z argumentami customer_id 4, product_code MON636 i qty 10:

```
smalljoins=# SELECT insert_order(4, 'MON636', 10);
```

Oto dane wyjściowe tego kodu:

```
insert_order
-----
1623
(1 row)
```

6. Przejrzyj dane z tabeli order_info:

```
smalljoins=# SELECT * FROM order_info;
```

Ten kod wyświetli dane wyjściowe pokazane na rysunku 6.53.

order_id	customer_id	product_code	qty
1618	3	GROG1	12
1619	2	POULET3	3
1620	4	MON123	1
1621	4	MON636	3
1622	5	MON666	1
1623	4	MON636	10
(6 rows)			

Rysunek 6.53. Lista zaktualizowanych informacji o zamówieniach

Zwróć uwagę na dodatkowy wiersz z wartością 1623 w kolumnie order_id.

7. Zaktualizuj tabelę `products`, aby uwzględnić ostatnią sprzedaż 10 sztuk produktu Red Herring. Użyj do tego funkcji `update_stock`:

```
smalljoins=# SELECT update_stock();
```

Oto dane wyjściowe tego kodu:

```
update_stock
-----
89
(1 row)
```

To wywołanie określa, ile produktów Red Herring pozostało w magazynie (po sprzedaży 10 sztuk), i odpowiednio aktualizuje tabelę.

8. Przyjrzyj się tabeli `products`. Zwróć uwagę na zaktualizowaną liczbę sztuk produktu Red Herring:

```
smalljoins=# SELECT * FROM products;
```

Na rysunku 6.54 pokazane są dane wyjściowe tego kodu.

product_code	name	stock
GROG1	Grog	65
POULET3	El Pollo Diablo	2
MON123	Rubber Chicken + Pulley	7
MON666	Murray's Arm	0
MON636	Red Herring	89
(5 rows)		

Rysunek 6.54. Lista ze zaktualizowaną liczbą sztuk produktu

Ręczne aktualizowanie liczby sztuk szybko staje się żmudne. Utwórz wyzwalacz, który będzie robił to automatycznie po złożeniu nowego zamówienia.

9. Usuń (za pomocą polecenia `DROP`) funkcję `update_stock`. Zanim utworzysz wyzwalacz, musisz zmodyfikować funkcję `update_stock`, by zwracała wyzwalacz. Pozwala to uprościć kod:

```
smalljoins=# DROP FUNCTION update_stock;
```

10. Utwórz nową funkcję `update_stock`, która zwraca wyzwalacz. Zauważ, że definicja tej funkcji znajduje się w pliku *Trigger.sql* (*Wyzwalacz.sql* w wersji spolszczonej), gdzie możesz ją przejrzeć lub skąd możesz ją wczytać to bazy:

```
smalljoins=# CREATE FUNCTION update_stock() RETURNS TRIGGER AS
$stock_trigger$
smalljoins## DECLARE stock_qty integer;
smalljoins## BEGIN
smalljoins## stock_qty := get_stock(NEW.product_code) - NEW.qty;
smalljoins## UPDATE products SET stock=stock_qty WHERE product_
code=NEW.product_code;
smalljoins## RETURN NEW;
smalljoins## END; $stock_trigger$
smalljoins=# LANGUAGE PLPGSQL;
```

W definicji tej funkcji używane jest słowo kluczowe NEW z operatorem kropki (.) i nazwami pól z tabeli order_info — product_code (NEW.product_code) oraz qty (NEW.qty). Słowo kluczowe NEW wskazuje rekord, który właśnie został wstawiony, zaktualizowany lub usunięty, dlatego zapewnia dostęp do informacji z tego rekordu.

W tym ćwiczeniu wyzwalacz ma być uruchamiany po wstawieniu rekordu do tabeli order_info, dlatego referencja NEW wskazuje informacje z tego rekordu. Możesz więc użyć funkcji get_stock z argumentem NEW.product_code, aby uzyskać aktualnie dostępną liczbę sztuk danego produktu, po czym odjąć od niej wartość NEW.qty.

11. Na koniec utwórz wyzwalacz. Ma on być uruchamiany po (AFTER) operacji INSERT w tabeli order_info. Dla każdego wiersza wyzwalacz ma wykonywać zmodyfikowaną przed chwilą funkcję update_stock, aby aktualizować liczbę sztuk w tabeli products:

```
smalljoins=# CREATE TRIGGER update_trigger
smalljoins=# AFTER INSERT ON order_info
smalljoins=# FOR EACH ROW
smalljoins=# EXECUTE PROCEDURE update_stock();
```

12. Po utworzeniu nowego wyzwalacza pora go przetestować. Wywołaj funkcję insert_order, aby wstawić nowy rekord do tabeli order_info:

```
smalljoins=# SELECT insert_order(4, 'MON123', 2);
```

Oto dane wyjściowe tego kodu:

```
insert_order
-----
1624
(1 row)
```

13. Przyjrzyj się rekordom z tabeli order_info:

```
smalljoins=# SELECT * FROM order_info;
```

Ten kod wyświetli dane wyjściowe z rysunku 6.55.

order_id	customer_id	product_code	qty
1618	3	GROG1	12
1619	2	POULET3	3
1620	4	MON123	1
1621	4	MON636	3
1622	5	MON666	1
1623	3	MON636	10
1624	4	MON123	2

(7 rows)

Rysunek 6.55. Informacje o zamówieniach z aktualizacją danych za pomocą wyzwalacza

14. Teraz wyświetl rekordy z tabeli products:

```
smalljoins=# SELECT * FROM products;
```

Na rysunku 6.56 pokazane są dane wyjściowe tego kodu.

product_code	name	stock
MON666	Murray's Arm	0
GROG1	Grog	65
POULET3	El Pollo Diablo	2
MON636	Red Herring	89
MON123	Rubber Chicken + Pulley	5
(5 rows)		

Rysunek 6.56. Informacje o produktach zaktualizowane przez wyzwalacz

Wyzwalacz zadziałał. Widać, że liczba sztuk produktu Rubber Chicken + Pulley (MON123) została zmniejszona z 7 do 5, zgodnie z liczbą sztuk we wstawionym zamówieniu.

Kod źródłowy z tego fragmentu książki jest dostępny na stronie <https://packt.live/2YqG51v>.

W tym ćwiczeniu utworzyliśmy wyzwalacz wykonujący funkcję pomocniczą po wstawieniu nowego rekordu do bazy. W następnym ćwiczeniu utworzysz wyzwalacz śledzący dane.

Zadanie 6.07 — tworzenie wyzwalacza do śledzenia średniej liczby kupionych sztuk

Celem jest utworzenie wyzwalacza do śledzenia aktualizowanych danych. Pracujesz jako specjalista od data science dla firmy Monkey Islands, czołowego dystrybutora przedmiotów o niejasnym pochodzeniu. Firma chce wypróbować kilka różnych strategii, aby zwiększyć liczbę produktów sprzedawanych w każdej transakcji. W celu uproszczenia analiz decydujesz się dodać prosty wyzwalacz, który po każdym nowym zamówieniu oblicza średnią liczbę sztuk we wszystkich zamówieniach i umieszcza wynik w nowej tabeli razem z identyfikatorem zamówienia (order_id). Oto kroki niezbędne do wykonania tego zadania:

1. Nawiąż połączenie z bazą `smalljoins`.
2. Utwórz nową tabelę, `avg_qty_log`, która zawiera pole `order_id` typu `integer` i pole `avg_qty` typu `numeric`.
3. Utwórz funkcję `avg_qty`, która nie przyjmuje żadnych argumentów, a zwraca wyzwalacz. Ta funkcja ma obliczać średnią liczbę sztuk z wszystkich zamówień (`order_info.qty`) i wstawiać tę wartość razem z najnowszym identyfikatorem `order_id` do tabeli `avg_qty_log`.
4. Utwórz wyzwalacz `avg_trigger`, który wywołuje funkcję `avg_qty` po (opcja `AFTER`) wstawieniu każdego wiersza do tabeli `order_info`.
5. Wstaw do tabeli `order_info` nowe wiersze z liczbą sztuk równą 6, 7 i 8.
6. Przejrzyj rekordy z tabeli `avg_qty_log`. Czy średnia liczba sztuk po każdym zamówieniu rośnie?

Oczekiwane dane wyjściowe są pokazane na rysunku 6.57.

order_id	avg_qty
1625	4.7500000000000000
1626	5.0000000000000000
1627	5.3000000000000000

(3 rows)

Rysunek 6.57. Zmiana średniej liczby sztuk w czasie

Rozwiązanie tego zadania znajduje się w „Dodatku”.

W tym zadaniu utworzyliśmy wyzwalacz do nieustannego śledzenia aktualizowanych danych, co pozwala analizować produkty w bazie danych.

Kończenie pracy kwerend

Czasem ilość danych jest duża lub ilość zasobów sprzętowych jest niewystarczająca, a kwerenda działa przez bardzo długi czas. Wtedy konieczne może być zakończenie pracy kwerendy — na przykład po to, aby napisać inną kwerendę, która szybciej zwraca potrzebne informacje. W tym punkcie zobaczysz, jak zakończy zablokowane lub bardzo długo działające kwerendy za pomocą dodatkowego interpretera systemu PostgreSQL. Oto polecenia, które służą do kończenia pracy kwerend:

- `pg_sleep` umożliwia poinformowanie interpretera SQL-a, że ma nic nie robić w najbliższym czasie, zdefiniowanym w sekundach w danych wejściowych funkcji.
- `pg_cancel_backend` powoduje, że interpreter kończy kwerendę wskazaną za pomocą identyfikatora procesu (`pid`). Proces jest wtedy zamykany w kontrolowany sposób, co umożliwia odpowiednie uporządkowanie zasobów. Kontrolowane kończenie pracy powinno być pierwszym wyborem programisty, ponieważ zmniejsza ryzyko uszkodzenia danych i bazy.
- `pg_terminate_backend` powoduje zatrzymanie istniejącego procesu, ale (w odróżnieniu od polecenia `pg_cancel_backend`) wymusza zakończenie procesu bez porządkowania zasobów wykorzystywanych przez kwerendę. Kwerenda jest kończona natychmiast, co może skutkować uszkodzeniem danych.

Aby uruchomić wymienione polecenia, trzeba je przetworzyć. Często używane jest do tego proste polecenie `SELECT`, tak jak w tym kodzie:

```
SELECT pg_terminate_backend(<PID>);
```

`PID` to identyfikator procesu z kwerendą, którą chcesz zakończyć. Po udanym wykonaniu tej instrukcji zobaczysz następujące dane wyjściowe:

```
pg_terminate_backend
-----
t
(1 row)
```


Teraz, gdy wiesz już, jak zamykać kwerendę zarówno w kontrolowany, jak i wymuszony sposób, wykonaj ćwiczenie, w którym zakończysz długo działającą kwerendę.

Ćwiczenie 6.08 — anulowanie długo działającej kwerendy

W tym ćwiczeniu anulujesz długo działającą kwerendę, aby zaoszczędzić czas, gdy kod utknie w trakcie jej wykonywania. Udało Ci się uzyskać dostęp do dużego zbioru danych i zdecydowałeś się uruchamiać kwerendę (którą początkowo uznałeś za dość prostą), aby uzyskać podstawowe statystyki opisowe danych. Jednak z jakichś powodów wykonywanie kwerendy zajmuje niezwykle dużo czasu i nie masz nawet pewności, że kod działa.

Uznałeś, że nadeszła pora, aby anulować kwerendę. To oznacza, że chcesz przesłać do kwerendy sygnał zatrzymania, ale też zapewnić jej wystarczająco dużo czasu, by umożliwić kontrolowane uporządkowanie zasobów. Ponieważ każdy Czytelnik używa innego sprzętu, a pobieranie danych potrzebnych do uruchomienia długo działającej kwerendy wymaga dużo czasu, zasymulujemy taką kwerendę za pomocą polecenia `pg_sleep`.

W tym ćwiczeniu będziesz potrzebować dwóch odrębnych sesji interpretera SQL-a działających w osobnych oknach, tak jak na rysunku 6.58.

1. Uruchom dwa odrębne interpretery za pomocą poleceń `psql` i `sqllda`:

```
C:\> psql sqllda postgres
```

Spowoduje to wyświetlenie dwóch osobnych okien z danymi wyjściowymi widocznymi na rysunku 6.58.

```
ben@hillValley:~$ psql sqllda
psql (11.4 (Ubuntu 11.4-0ubuntu0.19.04.1))
Type "help" for help.
```

```
sqllda=#
```

```
ben@hillValley:~$ psql sqllda
psql (11.4 (Ubuntu 11.4-0ubuntu0.19.04.1))
Type "help" for help.
```

```
sqllda=#
```

Rysunek 6.58. Uruchamianie kilku terminali

2. W pierwszym terminalu wykonaj polecenie `pg_sleep` z parametrem 1000 sekund:

```
sqllda=# SELECT pg_sleep(1000);
```

Po wciśnięciu klawisza *Enter* powinieneś zauważyć, że kursor w interpreterze nie wskazuje na zwrócenie sterowania (rysunek 6.59).

```
sqllda=# SELECT pg_sleep(1000);
```

Rysunek 6.59. „Uśpiony” interpreter

3. W drugim terminalu pobierz z tabeli `pg_stat_activity` kolumny `pid` i `query` z rekordów, w których kolumna `state` ma wartość `active`:

```
sqllda=# SELECT pid, query FROM pg_stat_activity WHERE state = 'active';
```

Na rysunku 6.60 pokazane są dane wyjściowe tego kodu.

```

pid |
-----+-----
14117 | SELECT pid, query FROM pg_stat_activity WHERE state = 'active';
14131 | SELECT pg_sleep(1000);
(2 rows)

```

Rysunek 6.60. Aktywne kwerendy

4. W drugim terminalu przekaż identyfikator procesu kwerendy z instrukcją `pg_sleep` do polecenia `pg_cancel_backend`, aby zakończyć tę kwerendę i przeprowadzić kontrolowane porządkowanie zasobów. Zauważ, że na Twoim komputerze identyfikator PID (14131) może być inny, użyj więc identyfikatora, który zobaczysz u siebie.

```
sql># SELECT pg_cancel_backend(14131);
```

Oto dane wyjściowe tego kodu.

```

pg_cancel_backend
-----
t
(1 row)

```

5. Sprawdź pierwszy terminal. Zauważ, że kwerenda z poleceniem `pg_sleep` nie jest już wykonywana, na co wskazuje zwrócony komunikat z rysunku 6.61.

```

BŁĄD: anulowano polecenie na skutek żądania użytkownika
sql>#

```

Rysunek 6.61. Komunikat informujący o anulowaniu kwerendy

Dane wyjściowe ze zrzutu informują o błędzie, ponieważ kwerenda została anulowana na żądanie użytkownika.

Kod źródłowy z tego fragmentu książki znajdziesz na stronie <https://packt.live/3dWLpQP>.

W tym ćwiczeniu pokazaliśmy, jak anulować kwerendę, której wykonywanie zajmuje dużo czasu. W następnym zadaniu spróbujesz zakończyć długo działającą kwerendę z wykorzystaniem poznanych tu informacji.

Zadanie 6.08 — kończenie długo działającej kwerendy

W tym zadaniu zakończysz długo działającą kwerendę za pomocą polecenia `pg_terminate_backend` (podobnie jak wcześniej użyliśmy polecenia `pg_cancel_backend` do zatrzymania procesu). Przyjmij, że w tym scenariuszu anulowanie kwerendy nie wystarcza do zatrzymania zbyt długo działającego procesu. Wymaga to bardziej zdecydowanych kroków niż zażądanie kontrolowanego

zakończenia procesu — konieczne jest wymuszenie zamknięcia procesu. Uruchom dwa odrębne interpretery SQL-a. Oto kroki niezbędne do wykonania tego zadania:

1. W pierwszym terminalu wywołaj polecenie `pg_sleep` z parametrem 1000 sekund.
2. W drugim terminalu ustal identyfikator procesu kwerendy z poleceniem `pg_sleep`.
3. Użyj wartości pid, aby wymusić zakończenie polecenia `pg_sleep` za pomocą instrukcji `pg_terminate_backend`.
4. W pierwszym terminalu upewnij się, że polecenie `pg_sleep` zostało zakończone. Zwróć uwagę na komunikat zwrócony przez interpreter.

Na rysunku 6.62 pokazane są oczekiwane dane wyjściowe.

```
sql>=# SELECT pg_sleep(1000);
KATASTROFALNY: zakończono połączenie na skutek polecenia administratora
server closed the connection unexpectedly
        This probably means the server terminated abnormally
        before or while processing the request.
The connection to the server was lost. Attempting reset: Succeeded.
sql>=#
```

Rysunek 6.62. Zakończony proces z wywołaniem `pg_sleep`

Rozwiązanie tego zadania znajdziesz na stronie 422.

W tym zadaniu zakończyliśmy długo działającą kwerendę za pomocą polecenia `pg_terminate_backend`.

Podsumowanie

W tym rozdziale omówiliśmy wiele zagadnień, które pomagają zrozumieć i zwiększyć wydajność kwerend SQL-owych. Na początku rozdziału zamieściliśmy rozbudowane omówienie planera kwerend (w tym instrukcji `EXPLAIN` i `ANALYZE`), a także różnych technik indeksowania. Opisaliśmy szereg kompromisów i czynników, które można uwzględnić, aby skrócić czas potrzebny na wykonywanie kwerend.

Przedstawiliśmy kilka scenariuszy, w których indeksy przyniosły korzyść, a także sytuacje, gdzie planer kwerend ignorował indeks, ponieważ nie zapewniał on wzrostu wydajności kwerendy. Następnie pokazaliśmy, jak używać złączeń do wydajnego łączenia informacji z różnych tabel, oraz omówiliśmy szczegółowo funkcje i automatyczne wywoływanie funkcji z wykorzystaniem wyzwalaczy. Opisaliśmy także polecenia `/df` i `/sf` oraz techniki zamykania długo działających kwerend.

W następnym rozdziale połączysz wszystkie poznane do tej pory zagadnienia w końcowym studium przypadku. Zastosujesz wiedzę z zakresu SQL-a i metodę naukową do rozwiązania praktycznego problemu.

Metoda naukowa i rozwiązywanie problemów w praktyce

Dzięki lekturze tego rozdziału nauczysz się rozwiązywać z wykorzystaniem zdobytych umiejętności problemy praktyczne inne niż opisane w tej książce. Posługując się metodą naukową i krytycznym myśleniem, będziesz analizować dane oraz przekształcać je w informacje i planować na ich podstawie działania. Aby się tego nauczyć, przyjrzyj się rozbudowanemu i szczegółowemu studium przypadku dotyczącemu danych sprzedażowych. Ilustruje ono procesy stosowane w analizach w SQL-u do znajdowania rozwiązań rzeczywistych problemów, ale też da Ci pewność i doświadczenie w radzeniu sobie z takimi problemami.

Wprowadzenie

Podręcznik *SQL. Analiza danych za pomocą zapytań* pozwolił Ci opanować zestaw nowych umiejętności (w tym korzystanie z podstawowych statystyk opisowych i poleceń SQL-owych oraz importowanie i eksportowanie danych w systemie PostgreSQL), a także zaawansowane metody optymalizowania i automatyzowania kodu w SQL-u (na przykład funkcje i wyzwalacze). W ostatnim rozdziale połączysz te nowe umiejętności z metodą naukową i krytycznym myśleniem, aby rozwiązać praktyczny problem i ustalić przyczynę nieoczekiwanego spadku sprzedaży.

Ten rozdział zawiera studium przypadku i pomoże Ci nabrać pewności siebie w stosowaniu nowych umiejętności z zakresu SQL-a do nurtujących Cię problemów. Aby rozwiązać problem opisany w tym studium przypadku, wykorzystasz cały zestaw zdobytych umiejętności

— od stosowania podstawowych operacji wyszukiwania w SQL-u w celu przefiltrowania dostępnych informacji po agregowanie i złączanie kilku zbiorów informacji oraz korzystanie z metod bazujących na oknie do logicznego grupowania danych. Dzięki pracy nad takimi studiami przypadku lepiej opanujesz najważniejsze narzędzia do analizy danych, co pomoże Ci w rozwoju kariery w dziedzinie data science.

Studium przypadku

W tym rozdziale zajmiesz się następującym studium przypadku: nowy skuter Bat firmy ZoomZoom jest sprzedawany wyłącznie za pośrednictwem witryny. Produkt sprzedawał się dobrze, ale po kilku tygodniach odnotowano spadek zamówień o 20%. Co się dzieje? Jesteś najlepszym analitykiem danych w ZoomZoom, dlatego przydzielono Ci do zbadania tej sytuacji.

Metoda naukowa

W tym studium przypadku w celu rozwiązania problemu posłużysz się metodą naukową. Jej istotą jest testowanie hipotez za pomocą obiektywnych danych. Metodę naukową możesz rozbić na następujące kluczowe kroki:

1. Zdefiniowanie pytania potrzebnego do ustalenia, co spowodowało spadek sprzedaży skutera Bat po dwóch tygodniach.
2. Zebranie wystarczających informacji w celu zaproponowania wstępnej hipotezy na temat określonego wydarzenia lub zjawiska.
3. Opracowanie hipotezy w celu wyjaśnienia zjawiska lub odpowiedzenia na pytanie.
4. Zdefiniowanie i przeprowadzenie obiektywnego eksperymentu, aby przetestować hipotezę. W idealnym scenariuszu wszystkie aspekty eksperymentu powinny być kontrolowane i stałe, z wyjątkiem zjawiska testowanego na podstawie hipotezy.
5. Przeanalizowanie danych zebranych w trakcie eksperymentu.
6. Opracowanie wyników analiz, które — jak można mieć nadzieję — wyjaśnią, z czego wynika spadek sprzedaży skuterów Bat.

Zauważ, że w tym rozdziale przeprowadzane są analizy post hoc. Oznacza to, że coś się już wydarzyło i wszystkie dostępne dane zostały zebrane. Analizy post hoc są przydatne zwłaszcza w sytuacji, kiedy zarejestrowanych zdarzeń nie można powtórzyć lub gdy nie da się kontrolować niektórych czynników zewnętrznych.

Na podstawie dostępnych danych można przeprowadzić analizy, które umożliwią uzyskanie informacji potrzebnych do potwierdzenia lub odrzucenia hipotezy. Nie da się jednak defini-tywnie potwierdzić lub odrzucić hipotezy bez praktycznych eksperymentów. Oto pytanie, które jest tematem tego rozdziału i wymaga odpowiedzi: dlaczego sprzedaż skuterów Bat firmy ZoomZoom spadła o 20% po dwóch tygodniach?

Zacznij od zupełnych podstaw.

Ćwiczenie 7.01

— wstępne zbieranie danych za pomocą technik SQL-a

W tym ćwiczeniu zbierzesz wstępne dane za pomocą technik SQL-a. Wiesz, że w czasie przedsprzedaży odnotowano wiele zamówień na skutery Bat firmy ZoomZoom, ale nagle ilość zainteresowanych spadła o 20%. Kiedy rozpoczęto produkcję i w jakiej cenie sprzedawano ten model? Jak skutery Bat wypadają cenowo w porównaniu z innymi skuterami? W tym ćwiczeniu celem jest uzyskanie odpowiedzi na te pytania. Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Wczytaj bazę sql da z kodu źródłowego ze strony <https://packt.live/2znKY2K>:

```
$ psql sql da
```

2. Wyświetl model (kolumna model), sugerowaną cenę detaliczną (kolumna base_msrp) i datę rozpoczęcia produkcji (kolumna production_start_date) z tabeli products dla produktów o typie scooter:

```
sql da=# SELECT model, base_msrp, production_start_date FROM products
WHERE product_type='scooter';
```

Tabela z rysunku 7.1 przedstawia informacje na temat wszystkich produktów typu scooter.

model	base_msrp	production_start_date
Lemon	399.99	2010-03-03 00:00:00
Lemon Limited Edition	799.99	2011-01-03 00:00:00
Lemon	499.99	2013-05-01 00:00:00
Blade	699.99	2014-06-23 00:00:00
Bat	599.99	2016-10-10 00:00:00
Bat Limited Edition	699.99	2017-02-15 00:00:00
Lemon Zester	349.99	2019-02-04 00:00:00

(7 rows)

Rysunek 7.1. Podstawowa lista skuterów z sugerowaną ceną detaliczną i datą rozpoczęcia produkcji

W wynikach wyszukiwania widać dwa produkty z członem Bat w nazwie — Bat i Bat Limited Edition. Produkcja skutera Bat rozpoczęła się 10 października 2016 roku, a sugerowana cena detaliczna tego modelu wynosiła 599,99 dolara. Skuter Bat Limited Edition trafił do produkcji mniej więcej cztery miesiące później, 15 lutego 2017 roku, a jego cena wynosiła 699,99 dolara.

Z informacji o produktach wynika, że skuter Bat ma wyjątkową cenę. Jako jedyny kosztuje 599,99 dolara. Dwa inne modele kosztują 699,99 dolara, a jeden — 499,99 dolara.

Również jeśli przyjrzyysz się samej dacie rozpoczęcia produkcji, zobaczysz, że skuter Bat jest wyjątkowy. Jest to jedyny model, którego produkcję rozpoczęto w ostatnim kwartale, a nawet półroczu (format dat to RRRR-MM-DD). Produkcję wszystkich pozostałych skuterów zaczęto w pierwszej połowie roku; tylko skuter Blade rozpoczęto wytwarzać w czerwcu.

Aby wykorzystać informacje sprzedażowe w połączeniu z dostępnymi informacjami o produktach, trzeba też pobrać identyfikator produktu dla każdego skutera.

3. Pobierz nazwę modelu i identyfikator produktu skuterów dostępnych w bazie danych. Te informacje będą potrzebne do połączenia informacji o produkcie z dostępnymi danymi sprzedażowymi:

```
sql># SELECT model, product_id FROM products WHERE product_type='scooter';
```

Ta kwerenda zwraca identyfikatory produktów widoczne w tabeli z rysunku 7.2.

model	product_id
Lemon	1
Lemon Limited Edition	2
Lemon	3
Blade	5
Bat	7
Bat Limited Edition	8
Lemon Zester	12
(7 rows)	

Rysunek 7.2. Identyfikatory produktów powiązane ze skuterami

4. Wstaw wyniki tej kwerendy do nowej tabeli, `product_names`, a następnie pobierz nowo wstawione dane:

```
SELECT model, product_id INTO product_names FROM products WHERE
product_type='scooter';
SELECT * FROM product_names;
```

Sprawdź zawartość tabeli `product_names`. Jest ona pokazana na rysunku 7.3.

model	product_id
Lemon	1
Lemon Limited Edition	2
Lemon	3
Blade	5
Bat	7
Bat Limited Edition	8
Lemon Zester	12
(7 rows)	

Rysunek 7.3. Zawartość nowej tabeli `product_names`

W tych danych wyjściowych widać, że skuter Bat cenowo znajduje się pomiędzy innymi modelami, a jego produkcję rozpoczęto w znacznie późniejszej porze roku niż pozostałych skuterów.

Kod źródłowy z tego fragmentu znajdziesz na stronie <https://packt.live/2MQ2QXd>.

Ten bardzo wstępny etap zbierania danych pozwolił Ci zdobyć informacje potrzebne do uzyskania danych sprzedażowych na temat skutera Bat, a także innych skuterów (na potrzeby porównań). Choć to ćwiczenie obejmowało zastosowanie najprostszych poleceń SQL-a, już zapewniło przydatne informacje.

Pokazało też, że nawet najprostsze polecenia SQL-a pozwala uzyskać przydatne informacje, dlatego nie należy ich lekceważyć. W następnym ćwiczeniu spróbujesz pobrać dane sprzedażowe związane ze spadkiem sprzedaży skuterów Bat.

Ćwiczenie 7.02 — pobieranie informacji sprzedażowych

W tym ćwiczeniu posłużysz się kombinacją prostych instrukcji SELECT, a także funkcjami agregującymi i funkcjami okna, aby zbadać dane sprzedażowe. Gdy masz już wstępne informacje, możesz je wykorzystać do pobrania rekordów dotyczących sprzedaży skuterów Bat i ustalić, co się dzieje. Masz tabelę, `product_names`, która zawiera nazwy modeli i identyfikatory produktów. Trzeba połączyć te informacje z rekordami z danymi sprzedażowymi i pobrać tylko wiersze dotyczące skuterów Bat:

1. Wczytaj bazę `sql`da:

```
$ psql sqlda
```

2. Wyświetl pola dostępne w tabeli `sales`:

```
sqlda=# \d sales
```

Ta kwerenda zwraca pola z tabeli `sales` (rysunek 7.4).

Column	Type	Collation	Nullable	Default
customer_id	bigint			
product_id	bigint			
sales_transaction_date	timestamp without time zone			
sales_amount	double precision			
channel	text			
dealership_id	double precision			

Rysunek 7.4. Struktura tabeli `sales`

Widać tu, że dostępne są identyfikatory klienta i produktu, a także data transakcji, informacje o transakcji, kanał, za pomocą którego dokonano zakupu, i identyfikator salonu.

3. Użyj złączenia wewnętrznego na podstawie kolumn `product_id` tabel `product_names` i `sales`. Z wyników tego złączenia wewnętrznego pobierz kolumny `model`, `customer_id`, `sales_transaction_date`, `sales_amount`, `channel` i `dealership_id` i zapisz je w osobnej tabeli `product_sales`:

```
DROP TABLE IF EXISTS products_sales;
SELECT model, customer_id, sales_transaction_date, sales_amount,
channel, dealership_id INTO products_sales FROM sales INNER JOIN
product_names ON sales.product_id=product_names.product_id;
```


4. Jeśli wystąpi błąd, usuń tabelę `products_sales` za pomocą kwerendy `DROP` (tak jak w poprzednim kodzie) i ponownie uruchom kod.

Dane wyjściowe kodu zobaczysz w następnym kroku.

W tym rozdziale wyniki kwerend i obliczeń są zapisywane w osobnych tabelach, ponieważ pozwala to sprawdzić wyniki poszczególnych etapów analiz w dowolnym momencie. W środowisku komercyjnym lub produkcyjnym w odrębnej tabeli zapisywane są zwykle tylko wyniki końcowe, choć zależy to od kontekstu rozwiązywanego problemu.

5. Przyjrzyj się pierwszym pięciu wierszom nowej tabeli, używając następującej kwerendy:

```
sql>=# SELECT * FROM products_sales LIMIT 5;
```

W tej tabeli wymienionych jest pięciu pierwszych klientów, którzy dokonali zakupu. Widoczne są tu kwota i inne informacje o transakcji (na przykład data i czas), co ilustruje rysunek 7.5.

model	customer_id	sales_transaction_date	sales_amount	channel	dealership_id
Lemon	41604	2012-03-30 22:45:29	399.99	internet	
Lemon	41531	2010-09-07 22:53:16	399.99	internet	
Lemon	41443	2011-05-24 02:19:11	399.99	internet	
Lemon	41291	2010-08-08 14:12:52	319.992	internet	
Lemon	41084	2012-01-09 03:34:52	319.992	internet	

(5 rows)

Rysunek 7.5. Tabela `products_sales` utworzona w wyniku złączenia

6. Pobierz z tabeli `products_sales` wszystkie informacje na temat skuterów `Bat` i uporządkuj je w tabeli `sales_transaction_date` rosnąco według dat. Dzięki pobraniu danych w ten sposób możesz przyjrzeć się informacjom z kilku pierwszych dni sprzedaży tego modelu:

```
sql>=# SELECT * FROM products_sales WHERE model='Bat' ORDER BY
sales_transaction_date;
```

Ta kwerenda generuje dane wyjściowe widoczne na rysunku 7.6.

7. Zlicz dostępne rekordy, używając następującej kwerendy:

```
sql>=# SELECT COUNT(model) FROM products_sales WHERE model='Bat';
```

Oto liczba rekordów dotyczących modelu `'Bat'`:

```
count
-----
7328
(1 row)
```

Od 10 października 2016 roku przeprowadzono więc 7328 transakcji. Sprawdź datę ostatniego rekordu, wykonując następny krok.

model text	customer_id bigint	sales_transaction_date timestamp without time zone	sales_amount double precision	channel text	dealership_id double precision
Bat	4319	2016-10-10 00:41:57	599.99	internet	[null]
Bat	40250	2016-10-10 02:47:28	599.99	dealership	4
Bat	35497	2016-10-10 04:21:08	599.99	dealership	2
Bat	4553	2016-10-10 07:42:59	599.99	dealership	11
Bat	11678	2016-10-10 09:21:08	599.99	internet	[null]
Bat	45868	2016-10-10 10:29:29	599.99	internet	[null]
Bat	24125	2016-10-10 18:57:25	599.99	dealership	1
Bat	31307	2016-10-10 21:22:38	599.99	internet	[null]
Bat	42213	2016-10-10 21:27:36	599.99	internet	[null]
Bat	47790	2016-10-11 01:28:58	599.99	dealership	20
Bat	6342	2016-10-11 03:04:57	599.99	internet	[null]
Bat	45880	2016-10-11 04:09:19	599.99	dealership	7
Bat	43477	2016-10-11 05:24:50	599.99	internet	[null]

Rysunek 7.6. Uporządkowane rekordy z danymi o sprzedaży

8. Określ ostatnią datę sprzedaży skutera Bat. W tym celu wybierz wartość maksymalną (za pomocą funkcji MAX) z pola sales_transaction_date:

```
sql>=# SELECT MAX(sales_transaction_date) FROM products_sales WHERE
model='Bat';
```

Oto ostatnia data sprzedaży:

```
max
-----
2019-05-31 22:15:30
```

Ostatnią sprzedaż zapisaną w bazie odnotowano 31 maja 2019 roku.

9. Pobierz dzienny poziom sprzedaży skuterów Bat i zapisz go w nowej tabeli, bat_sales, aby potwierdzić informacje uzyskane od działu sprzedażowego, od którego dowiedziałeś się, że po pierwszych dwóch tygodniach sprzedaż spadła o 20%:

```
sql>=# SELECT * INTO bat_sales FROM products_sales WHERE model='Bat'
ORDER BY sales_transaction_date;
```

10. Usuń informacje o godzinie, aby umożliwić analizowanie transakcji według dat (na tym etapie godzina sprzedaży nie jest istotna). W tym celu uruchom poniższą kwerendę, w której pole sales_transaction_date jest przekazywane do funkcji DATE, co powoduje pozostawienie w kolumnie tylko informacji o dniu, miesiącu i roku:

```
sql>=# UPDATE bat_sales SET
sales_transaction_date=DATE(sales_transaction_date);
```

Otrzymasz następujące dane wyjściowe:

```
UPDATE 7328
```

11. Wyświetl pięć pierwszych rekordów z tabeli bat_sales uporządkowanych według daty sprzedaży (sales_transaction_date):

```
sql>=# SELECT * FROM bat_sales ORDER BY sales_transaction_date LIMIT 5;
```

Na rysunku 7.7 pokazane są dane wyjściowe tego kodu.

model	customer_id	sales_transaction_date	sales_amount	channel	dealership_id
Bat	4553	2016-10-10 00:00:00	599.99	dealership	11
Bat	35497	2016-10-10 00:00:00	599.99	dealership	2
Bat	40250	2016-10-10 00:00:00	599.99	dealership	4
Bat	4319	2016-10-10 00:00:00	599.99	internet	
Bat	11678	2016-10-10 00:00:00	599.99	internet	

(5 rows)

Rysunek 7.7. Pięć pierwszych rekordów z danymi o sprzedaży skuterów Bat

- Utwórz nową tabelę, bat_sales_daily, zawierającą daty sprzedaży i dzienną liczbę transakcji:

```
sql>=# SELECT sales_transaction_date, COUNT(sales_transaction_date)
      INTO bat_sales_daily FROM bat_sales GROUP BY sales_transaction_date
      ORDER BY sales_transaction_date;
```

- Sprawdź pierwsze 22 rekordy (reprezentujące nieco ponad trzy tygodnie), ponieważ zgodnie z doniesieniami sprzedaż spadła po mniej więcej dwóch pierwszych tygodniach:

```
sql>=# SELECT * FROM bat_sales_daily LIMIT 22;
```

To spowoduje wyświetlenie danych wyjściowych z rysunku 7.8.

sales_transaction_date	count
2016-10-10 00:00:00	9
2016-10-11 00:00:00	6
2016-10-12 00:00:00	10
2016-10-13 00:00:00	10
2016-10-14 00:00:00	5
2016-10-15 00:00:00	10
2016-10-16 00:00:00	14
2016-10-17 00:00:00	9
2016-10-18 00:00:00	11
2016-10-19 00:00:00	12
2016-10-20 00:00:00	10
2016-10-21 00:00:00	6
2016-10-22 00:00:00	2
2016-10-23 00:00:00	5
2016-10-24 00:00:00	6
2016-10-25 00:00:00	9
2016-10-26 00:00:00	2
2016-10-27 00:00:00	4
2016-10-28 00:00:00	7
2016-10-29 00:00:00	5
2016-10-30 00:00:00	5
2016-10-31 00:00:00	3

(22 rows)

Rysunek 7.8. Trzy pierwsze tygodnie sprzedaży

Widoczny jest tu spadek sprzedaży po 20 października. W pierwszych 11 wierszach jest 7 dni z dwucyfrową sprzedażą, a w następnych 11 dniach nie ma ani jednego takiego dnia.

Kod źródłowy z tego fragmentu książki znajdziesz na stronie <https://packt.live/3cVQffy>.

Na tym etapie możesz potwierdzić, że wystąpił spadek sprzedaży. Musisz jeszcze precyzyjnie obliczyć zakres i powody tego spadku. Zajmiesz się tym w następnym zadaniu.

Zadanie 7.01 — ilościowa ocena spadku sprzedaży

W tym zadaniu wykorzystasz wiedzę o funkcjach okna, uzyskaną dzięki lekturze rozdziału 3., „Agregacja i funkcje okna”. W poprzednim ćwiczeniu wykryłeś spadek sprzedaży rozpoczynający się mniej więcej 10 dni po wprowadzeniu produktu na rynek. Teraz spróbujesz ilościowo ocenić spadek sprzedaży skuterów Bat.

Oto kroki niezbędne do wykonania tego zadania:

1. Ponownie wczytaj bazę danych `sql`da z kodu źródłowego ze strony <https://packt.live/2znKY2K>.
2. Za pomocą instrukcji `OVER` i `ORDER BY` oblicz dzienną skumulowaną sumę transakcji. Dzięki temu uzyskasz liczbę transakcji dla danego okresu po kolejnych dniach. Wstaw wyniki do nowej tabeli, `bat_sales_growth`.
3. Pobierz wartość kolumny `sum sprzed 7 dni` (za pomocą funkcji `lag`), a następnie wstaw wszystkie kolumny z tabeli `bat_sales_daily` i nową kolumnę `lag` do nowej tabeli, `bat_sales_daily_delay`. Kolumna `lag` informuje o poziomie sprzedaży sprzed tygodnia, co pozwala porównywać aktualną sprzedaż z poziomem z poprzedniego tygodnia.
4. Sprawdź pierwszych 15 wierszy tabeli `bat_sales_growth`.
5. Oblicz procentowy przyrost liczby transakcji, porównując aktualną liczbę sprzedaży z poziomem sprzed tygodnia. Wstaw wynikowe wartości do nowej tabeli, `bat_sales_delay_vol`.
6. Porównaj pierwsze 22 wartości z tabeli `bat_sales_delay_vol`, aby przekonać się o spadku sprzedaży.

Oczekiwane dane wyjściowe są pokazane na rysunku 7.9.

Rozwiązanie tego zadania znajdziesz w „Dodatku”.

Choć kolumny z liczbą transakcji i sumą skumulowaną są łatwe do zrozumienia, do czego służą kolumny `lag` i `volume`? Są potrzebne, ponieważ analizowany jest spadek wzrostu sprzedaży w kilku pierwszych tygodniach. Dlatego suma transakcji po określonym dniu jest porównywana z tą samą wartością sprzed 7 dni (wartość przesunięcia). Po odjęciu sumy sprzed 7 dni od aktualnej sumy i podzieleniu wyniku przez sumę sprzed 7 dni otrzymasz wartość kolumny `volume`, która pozwala ustalić względny przyrost liczby transakcji w porównaniu z poprzednim tygodniem.

sales_transaction_date	count	sum	lag	volume
2016-10-10 00:00:00	9	9		
2016-10-11 00:00:00	6	15		
2016-10-12 00:00:00	10	25		
2016-10-13 00:00:00	10	35		
2016-10-14 00:00:00	5	40		
2016-10-15 00:00:00	10	50		
2016-10-16 00:00:00	14	64		
2016-10-17 00:00:00	9	73	9	7.1111111111111111
2016-10-18 00:00:00	11	84	15	4.6000000000000000
2016-10-19 00:00:00	12	96	25	2.8400000000000000
2016-10-20 00:00:00	10	106	35	2.0285714285714286
2016-10-21 00:00:00	6	112	40	1.8000000000000000
2016-10-22 00:00:00	2	114	50	1.2800000000000000
2016-10-23 00:00:00	5	119	64	0.8593750000000000
2016-10-24 00:00:00	6	125	73	0.7123287671232876
2016-10-25 00:00:00	9	134	84	0.5952380952380952
2016-10-26 00:00:00	2	136	96	0.4166666666666666
2016-10-27 00:00:00	4	140	106	0.3207547169811320
2016-10-28 00:00:00	7	147	112	0.3125000000000000
2016-10-29 00:00:00	5	152	114	0.3333333333333333
2016-10-30 00:00:00	5	157	119	0.3193277310924369
2016-10-31 00:00:00	3	160	125	0.2800000000000000

(22 rows)

Rysunek 7.9. Względny przyrost liczby sprzedanych skuterów Bat z trzech tygodni

Zauważ, że 17 października wzrost liczby transakcji wynosi **700%** w porównaniu z dniem rozpoczęcia sprzedaży (10 października). W dniu 22 października liczba transakcji jest ponad dwukrotnie wyższa niż tydzień wcześniej. Wraz z upływem czasu względna różnica gwałtownie maleje. Pod koniec października liczba transakcji jest tylko o **28%** wyższa niż tydzień wcześniej. Na tym etapie zaobserwowałeś i potwierdziłeś spadek wzrostu sprzedaży po dwóch pierwszych tygodniach. W następnym kroku spróbujesz wyjaśnić przyczyny tego zjawiska.

Ćwiczenie 7.03 — analiza czasu rozpoczęcia sprzedaży

W tym ćwiczeniu spróbujesz zidentyfikować powody spadku sprzedaży. Po potwierdzeniu spadku szybkości przyrostu liczby transakcji spróbujesz wyjaśnić, z czego to wynika. Przetestujesz hipotezę, zgodnie z którą czas wprowadzenia skutera na rynek ma wpływ na spowolnienie sprzedaży. Z „Ćwiczenia 7.01 — wstępne zbieranie danych za pomocą technik SQL-a” wiesz, że skuter Bat firmy ZoomZoom trafił do sprzedaży 10 października 2016 roku. Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Wczytaj bazę danych `sql`da:

```
$ psql sql
```

2. Przeanalizuj inne produkty z bazy. Aby stwierdzić, czy data rozpoczęcia sprzedaży wpłynęła na spadek liczby transakcji, musisz porównać skuter Bat z innymi skuterami, uwzględniając datę wprowadzenia produktu na rynek. Wykonaj poniższą kwerendę, aby sprawdzić daty rozpoczęcia sprzedaży:

```
sql>=# SELECT * FROM products;
```

Zrzut z rysunku 7.10 pokazuje daty wprowadzenia na rynek wszystkich produktów.

product_id	model	year	product_type	base_msrp	production_start_date	production_end_date
1	Lemon	2010	scooter	399.99	2010-03-03 00:00:00	2012-06-08 00:00:00
2	Lemon Limited Edition	2011	scooter	799.99	2011-01-03 00:00:00	2011-03-30 00:00:00
3	Lemon	2013	scooter	499.99	2013-05-01 00:00:00	2018-12-28 00:00:00
4	Model Chi	2014	automobile	115,000.00	2014-06-23 00:00:00	2018-12-28 00:00:00
5	Blade	2014	scooter	699.99	2014-06-23 00:00:00	2015-01-27 00:00:00
6	Model Sigma	2015	automobile	65,500.00	2015-04-15 00:00:00	2018-10-01 00:00:00
7	Bat	2016	scooter	599.99	2016-10-10 00:00:00	
8	Bat Limited Edition	2017	scooter	699.99	2017-02-15 00:00:00	
9	Model Epsilon	2017	automobile	35,000.00	2017-02-15 00:00:00	
10	Model Gamma	2017	automobile	85,750.00	2017-02-15 00:00:00	
11	Model Chi	2019	automobile	95,000.00	2019-02-04 00:00:00	
12	Lemon Zester	2019	scooter	349.99	2019-02-04 00:00:00	

(12 rows)

Rysunek 7.10. Produkty i daty ich wprowadzenia na rynek

Sprzedaż wszystkich innych produktów rozpoczęła się przed lipcem, inaczej niż skutera Bat, który trafił na rynek w październiku.

- Wyświetl wszystkie skutery z tabeli products (w porównaniach istotne są tylko skutery):

```
sql># SELECT * FROM products WHERE product_type='scooter';
```

W tabeli z rysunku 7.11 pokazane są wszystkie informacje o produktach typu scooter.

product_id	model	year	product_type	base_msrp	production_start_date	production_end_date
1	Lemon	2010	scooter	399.99	2010-03-03 00:00:00	2012-06-08 00:00:00
2	Lemon Limited Edition	2011	scooter	799.99	2011-01-03 00:00:00	2011-03-30 00:00:00
3	Lemon	2013	scooter	499.99	2013-05-01 00:00:00	2018-12-28 00:00:00
5	Blade	2014	scooter	699.99	2014-06-23 00:00:00	2015-01-27 00:00:00
7	Bat	2016	scooter	599.99	2016-10-10 00:00:00	
8	Bat Limited Edition	2017	scooter	699.99	2017-02-15 00:00:00	
12	Lemon Zester	2019	scooter	349.99	2019-02-04 00:00:00	

(7 rows)

Rysunek 7.11. Daty wprowadzenia skuterów na rynek

Aby przetestować hipotezę, że pora roku wpływa na sprzedaż, potrzebny jest model skutera używany jako grupa kontrolna (grupa odniesienia). W idealnym świecie można byloby rozpocząć sprzedaż skuterów Bat w dwóch lokalizacjach w różnym czasie, a następnie porównać wyniki. Tu jednak jest to niemożliwe.

Zamiast tego posłużysz się danymi dotyczącymi podobnego skutera wprowadzonego na rynek w innym czasie. W bazie danych z produktami dostępne są różne skutery z określonymi podobieństwami i różnicami względem grupy eksperymentalnej (skuter Bat). Naszym zdaniem odpowiednim produktem do porównania będzie skuter Bat Limited Edition (grupa kontrolna). Jest on trochę droższy, ale trafił na rynek tylko cztery miesiące po modelu Bat.

Nazwa Bat Limited Edition wskazuje, że ten model ma podobną charakterystykę co Bat, ale ponieważ jest edycją limitowaną (Limited Edition), ma też kilka dodatków.

- Pobierz pięć pierwszych wierszy z tabeli sales:

```
sql># SELECT * FROM sales LIMIT 5;
```

Na rysunku 7.12 widoczne są informacje o sprzedaży produktu pięciu pierwszym klientom.

customer_id	product_id	sales_transaction_date	sales_amount	channel	dealership_id
1	7	2017-07-19 08:38:41	479.992	internet	
22	7	2017-08-14 09:59:02	599.99	dealership	20
145	7	2019-01-20 10:40:11	479.992	internet	
289	7	2017-05-09 14:20:04	539.991	dealership	7
331	7	2019-05-21 20:03:21	539.991	dealership	4

(5 rows)

Rysunek 7.12. Pięć pierwszych wierszy danych sprzedażowych

5. Z tabel `products` i `sales` pobierz kolumny `model` i `sales_transaction_date` dotyczące modelu `Bat Limited Edition`. Zapisz wyniki w tabeli `bat_ltd_sales`. Uporządkuj je według kolumny `sales_transaction_date` od najstarszych do najnowszych transakcji.

```
sql>=# SELECT products.model, sales.sales_transaction_date INTO
bat_ltd_sales FROM sales INNER JOIN products ON
sales.product_id=products.product_id
WHERE sales.product_id=8 ORDER BY sales.sales_transaction_date;
```

Otrzymasz następujące dane wyjściowe:

```
SELECT 5803
```

6. Pobierz pierwszych pięć wierszy tabeli `bat_ltd_sales`. Użyj następującej kwerendy:

```
sql>=# SELECT * FROM bat_ltd_sales LIMIT 5;
```

W tabeli z rysunku 7.13 pokazane są informacje o pięciu pierwszych transakcjach sprzedaży modelu `Bat Limited Edition`.

model	sales_transaction_date
Bat Limited Edition	2017-02-15 01:49:02
Bat Limited Edition	2017-02-15 09:42:37
Bat Limited Edition	2017-02-15 10:48:31
Bat Limited Edition	2017-02-15 12:22:41
Bat Limited Edition	2017-02-15 13:51:34

(5 rows)

Rysunek 7.13. Pięć pierwszych transakcji sprzedaży skutera `Bat Limited Edition`

7. Ustal łączną liczbę sprzedanych skuterów `Bat Limited Edition`. Możesz ją sprawdzić za pomocą funkcji `COUNT`:

```
sql>=# SELECT COUNT(model) FROM bat_ltd_sales;
```

Oto łączna liczba transakcji:

```
Count
-----
5803
(1 row)
```

Możesz porównać tę wartość z liczbą sprzedanych skuterów bazowego modelu `Bat`, wynoszącą 7328.

8. Sprawdź informacje o ostatniej transakcji modelu Bat Limited Edition.

Możesz w tym celu użyć funkcji MAX:

```
sql>=# SELECT MAX(sales_transaction_date) FROM bat_ltd_sales;
```

Oto informacje na temat ostatniej sprzedaży modelu Bat Limited Edition:

```
max
-----
2019-05-31 15:08:03
```

9. Dostosuj tabelę, aby uwzględnić tylko datę z kolumny (pomiń informacje o godzinie). Podobnie jak w analizach bazowego modelu Bat godzina sprzedaży nie jest istotna. Zapisz następującą kwerendę:

```
sql>=# ALTER TABLE bat_ltd_sales ALTER COLUMN sales_transaction_date
TYPE date;
```

10. Ponownie pobierz pięć pierwszych rekordów z tabeli bat_ltd_sales, aby sprawdzić, czy w kolumnie sales_transaction_date znajdują się teraz same daty:

```
sql>=# SELECT * FROM bat_ltd_sales LIMIT 5;
```

W tabeli z rysunku 7.14 widocznych jest pięć pierwszych rekordów z tabeli bat_ltd_sales.

model	sales_transaction_date
Bat Limited Edition	2017-02-15
Bat Limited Edition	2017-02-15
Bat Limited Edition	2017-02-15
Bat Limited Edition	2017-02-15
Bat Limited Edition	2017-02-15

(5 rows)

Rysunek 7.14. Pobieranie pierwszych pięciu transakcji (według daty) modelu Bat Limited Edition

11. Podobnie jak podczas analiz bazowego modelu Bat przygotuj liczbę transakcji z określonych dni. Wstaw wyniki do tabeli bat_ltd_sales_count za pomocą następującej kwerendy:

```
sql>=# SELECT sales_transaction_date, count(sales_transaction_date) INTO
bat_ltd_sales_count FROM bat_ltd_sales GROUP BY sales_transaction_date ORDER BY
sales_transaction_date;
```

12. Wyświetl liczbę transakcji wszystkich skuterów Bat Limited Edition, korzystając z poniższej kwerendy:

```
sql>=# SELECT * FROM bat_ltd_sales_count;
```

Liczba transakcji jest pokazana na zrzucie z rysunku 7.15.

13. Oblicz sumę skumulowaną dziennej sprzedaży i zapisz wynikową tabelę jako bat_ltd_sales_growth:

```
sql>=# SELECT *, sum(count) OVER (ORDER BY sales_transaction_date)
INTO bat_ltd_sales_growth FROM bat_ltd_sales_count;
```


sales_transaction_date	count
2017-02-15	6
2017-02-16	2
2017-02-17	1
2017-02-18	4
2017-02-19	5
2017-02-20	6
2017-02-21	5
2017-02-22	4
2017-02-23	6
2017-02-24	2
2017-02-25	2
2017-02-26	2
2017-02-27	4
2017-02-28	4
2017-03-01	5
2017-03-02	1

Rysunek 7.15. Dzienna sprzedaż modelu Bat Limited Edition

14. Pobierz dane sprzedażowe z pierwszych 22 dni z tabeli bat_ltd_sales_growth:

```
sql># SELECT * FROM bat_ltd_sales_growth LIMIT 22;
```

W tabeli z rysunku 7.16 wyświetlone są pierwsze 22 rekordy z danymi o wzroście sprzedaży.

sales_transaction_date	count	sum
2017-02-15	6	6
2017-02-16	2	8
2017-02-17	1	9
2017-02-18	4	13
2017-02-19	5	18
2017-02-20	6	24
2017-02-21	5	29
2017-02-22	4	33
2017-02-23	6	39
2017-02-24	2	41
2017-02-25	2	43
2017-02-26	2	45
2017-02-27	4	49
2017-02-28	4	53
2017-03-01	5	58
2017-03-02	1	59
2017-03-03	3	62
2017-03-04	8	70
2017-03-05	4	74
2017-03-06	7	81
2017-03-07	7	88
2017-03-08	8	96

(22 rows)

Rysunek 7.16. Sprzedaż modelu Bat Limited Edition — suma skumulowana

15. Porównaj te dane sprzedażowe z wartościami dla bazowego modelu Bat zwracanymi przez ten kod:

```
sql># SELECT * FROM bat_sales_growth LIMIT 22;
```

Tabela z rysunku 7.17 przedstawia informacje o sprzedaży z 22 pierwszych rekordów tabeli `bat_sales_growth`.

<code>sales_transaction_date</code>	<code>count</code>	<code>sum</code>
2016-10-10 00:00:00	9	9
2016-10-11 00:00:00	6	15
2016-10-12 00:00:00	10	25
2016-10-13 00:00:00	10	35
2016-10-14 00:00:00	5	40
2016-10-15 00:00:00	10	50
2016-10-16 00:00:00	14	64
2016-10-17 00:00:00	9	73
2016-10-18 00:00:00	11	84
2016-10-19 00:00:00	12	96
2016-10-20 00:00:00	10	106
2016-10-21 00:00:00	6	112
2016-10-22 00:00:00	2	114
2016-10-23 00:00:00	5	119
2016-10-24 00:00:00	6	125
2016-10-25 00:00:00	9	134
2016-10-26 00:00:00	2	136
2016-10-27 00:00:00	4	140
2016-10-28 00:00:00	7	147
2016-10-29 00:00:00	5	152
2016-10-30 00:00:00	5	157
2016-10-31 00:00:00	3	160

(22 rows)

Rysunek 7.17. Skumulowana sprzedaż skutera Bat — 22 wiersze

W pierwszych 22 dniach sprzedaż modelu Bat Limited Edition nie doszła do poziomu liczb dwucyfrowych i była mniej zmienna. Sumaryczna sprzedaż skuterów Bat Limited Edition była w pierwszych 22 dniach niższa o 64 sztuki w porównaniu z modelem Bat.

16. Za pomocą funkcji `lag` wyznacz wartości kolumny `sum` sprzed 7 dni i wstaw wyniki do tabeli `bat_ltd_sales_delay`:

```
sql>=# SELECT *, lag(sum, 7) OVER (ORDER BY sales_transaction_date)
      INTO bat_ltd_sales_delay FROM bat_ltd_sales_growth;
```

17. Oblicz wzrost liczby sprzedanych sztuk w tabeli `bat_ltd_sales_delay` w podobny sposób jak w „Zadaniu 7.01 — ilościowa ocena spadku sprzedaży”. Kolumnę z wynikami tych obliczeń nazwij `volume`, a wynikową tabelę zapisz jako `bat_ltd_sales_vol`:

```
sql>=# SELECT *, (sum-lag)/lag AS volume INTO bat_ltd_sales_vol FROM
      bat_ltd_sales_delay;
```

18. Przyjrzyj się pierwszym 22 rekordom danych sprzedażowych z tabeli `bat_ltd_sales_vol`:

```
sql>=# SELECT * FROM bat_ltd_sales_vol LIMIT 22;
```

Względny przyrost liczby transakcji jest widoczny na zrzucie z rysunku 7.18.

sales_transaction_date	count	sum	lag	volume
2017-02-15	6	6		
2017-02-16	2	8		
2017-02-17	1	9		
2017-02-18	4	13		
2017-02-19	5	18		
2017-02-20	6	24		
2017-02-21	5	29		
2017-02-22	4	33	6	4.5000000000000000
2017-02-23	6	39	8	3.8750000000000000
2017-02-24	2	41	9	3.5555555555555556
2017-02-25	2	43	13	2.3076923076923077
2017-02-26	2	45	18	1.5000000000000000
2017-02-27	4	49	24	1.0416666666666667
2017-02-28	4	53	29	0.8275862068965517
2017-03-01	5	58	33	0.7575757575757576
2017-03-02	1	59	39	0.5128205128205128
2017-03-03	3	62	41	0.5121951219512195
2017-03-04	8	70	43	0.6279069767441860
2017-03-05	4	74	45	0.6444444444444444
2017-03-06	7	81	49	0.6530612244897959
2017-03-07	7	88	53	0.6603773584905660
2017-03-08	8	96	58	0.6551724137931034

(22 rows)

Rysunek 7.18. Skumulowana sprzedaż modelu Bat Limited Edition ze względnym przyrostem liczby transakcji (kolumna volume)

Kolumna volume na rysunku pokazuje, że względny przyrost liczby transakcji jest bardziej stabilny niż dla bazowego modelu Bat. W pierwszym tygodniu ten przyrost jest niższy niż dla modelu Bat, ale zostaje utrzymany przez dłuższy czas. Po 22 dniach sprzedaży liczba sprzedanych sztuk skutera Bat Limited Edition jest o 65% wyższa niż tydzień wcześniej; możesz to porównać ze wzrostem 28% odnotowanym w „Zadaniu 7.01 — ilościowa ocena spadku sprzedaży”.

Kod źródłowy z tego fragmentu książki jest dostępny na stronie <https://packt.live/2YtuLS5>.

Zebrałeś dane na temat dwóch podobnych produktów wprowadzonych na rynek w innej porze roku i odkryłeś różnice we wzroście liczby sprzedanych sztuk w pierwszych trzech tygodniach sprzedaży. W kontekście profesjonalnym mógłbyś rozważyć zastosowanie bardziej zaawansowanych metod statystycznych, takich jak testy różnic średnich, wariancja, analiza przeżycia i inne techniki. Te metody wykraczają poza zakres niniejszej książki, dlatego posługujemy się tu prostszymi metodami porównawczymi.

Choć wykryłeś tu różnicę w sprzedaży skuterów z serii Bat, nie można wykluczyć, że wynikają one z różnicy w cenie sprzedaży. Wersja Limited Edition jest o 100 dolarów droższa. W następnym zadaniu porównasz sprzedaż skutera Bat z modelem Lemon z 2013 roku, który jest o 100 dolarów tańszy, trafił na rynek trzy lata wcześniej, nie jest już produkowany, a jego produkcję rozpoczęto w pierwszej połowie roku.

Zadanie 7.02

— analiza hipotezy dotyczącej różnicy w cenie sprzedaży

W tym zadaniu zbadasz hipotezę, zgodnie z którą spadek sprzedaży można przypisać cenie modelu Bat. Wcześniej analizowałeś datę wprowadzenia na rynek. Jednak istotny może być też inny czynnik — cena sprzedaży. Jeśli przejrzysz listę skuterów z tabeli z rysunku 7.19 i pominiesz model Bat, zobaczysz, że dostępne są produkty w dwóch cenach: 699,99 dolara i 499,99 dolara. Model Bat znajduje się dokładnie pośrodku. Możliwe, że spadek sprzedaży wynika z dostępności modeli w innych cenach. W tym zadaniu przetestujesz tę hipotezę, porównując sprzedaż skutera Bat ze sprzedażą modelu Lemon z 2013 roku.

product_id	model	year	product_type	base_msrp	production_start_date	production_end_date
12	Lemon Zester	2019	scooter	349.99	2019-02-04 00:00:00	
1	Lemon	2010	scooter	399.99	2010-03-03 00:00:00	2012-06-08 00:00:00
3	Lemon	2013	scooter	499.99	2013-05-01 00:00:00	2018-12-28 00:00:00
7	Bat	2016	scooter	599.99	2016-10-10 00:00:00	
5	Blade	2014	scooter	699.99	2014-06-23 00:00:00	2015-01-27 00:00:00
8	Bat Limited Edition	2017	scooter	699.99	2017-02-15 00:00:00	
2	Lemon Limited Edition	2011	scooter	799.99	2011-01-03 00:00:00	2011-03-30 00:00:00

(7 rows)

Rysunek 7.19. Lista modeli skuterów

Oto kroki niezbędne do wykonania tego zadania:

1. Wczytaj bazę sql da z katalogu z kodem źródłowym (<https://packt.live/2znKY2K>).
2. Pobierz kolumnę sales_transaction_date z transakcjami dotyczącymi modelu Lemon z 2013 roku. Wstaw tę kolumnę do tabeli lemon_sales.
3. Zlicz rekordy z transakcjami dotyczące modelu Lemon z 2013 roku.
4. Wyświetl najnowszą wartość z kolumny sales_transaction_date.
5. Przekształć typ kolumny sales_transaction_date na date.
6. Określ liczbę transakcji z poszczególnych dni z tabeli lemon_sales. Wstaw wynikowe dane do tabeli lemon_sales_count.
7. Oblicz sumę skumulowaną liczby transakcji i wstaw wyniki do nowej tabeli, lemon_sales_sum.
8. Za pomocą funkcji lag wyznacz wartości kolumny sum sprzed 7 dni i wstaw wyniki do tabeli lemon_sales_delay.
9. Oblicz tempo wzrostu liczby transakcji sprzedaży na podstawie danych z tabeli lemon_sales_delay. Zapisz wynikową tabelę pod nazwą lemon_sales_growth.
10. Zbadaj pierwsze 22 rekordy z tabeli lemon_sales_growth, analizując dane z kolumny volume.

Oczekiwane dane wyjściowe są pokazane na rysunku 7.20.

Rozwiązanie tego zadania znajdziesz w „Dodatku”.

sales_transaction_date	count	sum	lag	volume
2013-05-01	6	6		
2013-05-02	8	14		
2013-05-03	4	18		
2013-05-04	9	27		
2013-05-05	9	36		
2013-05-06	6	42		
2013-05-07	8	50		
2013-05-08	6	56	6	8.333333333333333
2013-05-09	6	62	14	3.4285714285714286
2013-05-10	9	71	18	2.9444444444444444
2013-05-11	3	74	27	1.7407407407407407
2013-05-12	4	78	36	1.1666666666666667
2013-05-13	7	85	42	1.0238095238095238
2013-05-14	3	88	50	0.7600000000000000
2013-05-15	3	91	56	0.6250000000000000
2013-05-16	4	95	62	0.5322580645161290
2013-05-17	6	101	71	0.4225352112676056
2013-05-18	9	110	74	0.4864864864864864
2013-05-19	6	116	78	0.4871794871794871
2013-05-20	6	122	85	0.4352941176470588
2013-05-21	11	133	88	0.5113636363636364
2013-05-22	8	141	91	0.5494505494505494

(22 rows)

Rysunek 7.20. Wzrost liczby sprzedanych sztuk skutera Lemon

Zebrałeś dane do przetestowania dwóch hipotez związanych z porą roku i ceną. Jakie obserwacje możesz poczynić i jakie wnioski wyciągnąć?

Pierwsza obserwacja związana jest z łączną liczbą sprzedanych sztuk trzech różnych skuterów. Skuter Lemon przez okres 4,5 roku został sprzedany w liczbie 16 558 sztuk. Dwa skutery z serii Bat, model bazowy i Limited Edition, sprzedano w liczbie 7328 i 5803 sztuk. Oba te skutery nadal są produkowane, a model Bat został wprowadzony na rynek 4 miesiące wcześniej i jest sprzedawany od około 2,5 roku.

Po przyjrzeniu się wzrostowi liczby sprzedanych sztuk można poczynić kilka różnych obserwacji:

- Sprzedaż bazowego modelu skutera Bat, wprowadzonego na rynek w październiku w cenie 599,99 dolara, w drugim tygodniu produkcji wzrosła o 700%, a po pierwszych 22 dniach odnotowano 28% wzrostu sprzedaży przy 160 sprzedanych sztukach.
- Sprzedaż skutera Bat Limited Edition, wprowadzonego na rynek w lutym w cenie 699,99 dolara, w drugim tygodniu produkcji wzrosła o 450%, a po pierwszych 22 dniach odnotowano 66% wzrostu sprzedaży przy 96 sprzedanych sztukach.
- Sprzedaż skutera Lemon z 2013 roku, wprowadzonego na rynek w maju w cenie 499,99 dolara, w drugim tygodniu produkcji wzrosła o 830%, a po pierwszych 22 dniach odnotowano 55% wzrostu sprzedaży przy 141 sprzedanych sztukach.

Na podstawie tych informacji można wyciągnąć kilka różnych wniosków:

- Początkowa szybkość wzrostu liczby transakcji w drugim tygodniu sprzedaży jest skorelowana z ceną skutera. Przy wyższej cenie 699,99 dolara początkowe tempo wzrostu spadło z 830% do 450%.

- Liczba sztuk sprzedanych w pierwszych 22 dniach nie jest bezpośrednio skorelowana z ceną. Mimo różnicy w cenie w pierwszym okresie dostępności skuter Bat w cenie 599,99 sprzedawał się lepiej niż model Lemon z 2013 roku.
- Dane częściowo dowodzą, że spadek sprzedaży można przypisać zmienności sezonowej. Wskazują na to znaczne zmniejszenie sprzedaży i fakt, że bazowy model Bat został wprowadzony na rynek dopiero w październiku. Dotychczasowe analizy wskazują, że spadek sprzedaży można przypisać momentowi wprowadzenia modelu na rynek.

Zanim dojdiesz do wniosku, że różnice można przypisać sezonowej zmienności i czasowi wprowadzenia produktu na rynek, upewnij się, że dobrze przetestowałeś wszystkie możliwości. Możliwe, że różnice wynikają z działań marketingowych takich jak kampanie e-mailowe (okresy, gdy firma wysyła e-maile) i częstotliwość otwierania wiadomości.

Po rozważeniu momentu rozpoczęcia sprzedaży i sugerowanej ceny detalicznej skuterów jako możliwych przyczyn spadku sprzedaży możesz skupić się na innych czynnikach, na przykład na współczynniku otwarć e-maili marketingowych. Czy ten współczynnik jest związany ze wzrostem sprzedaży w pierwszych trzech tygodniach? Przekonasz się o tym w następnym ćwiczeniu.

Ćwiczenie 7.04 — analiza zależności wzrostu sprzedaży od współczynnika otwarć e-maili

W tym ćwiczeniu przeanalizujesz wzrost liczby sprzedanych sztuk w zależności od współczynnika otwarć e-maili. Aby zbadać hipotezę, zgodnie z którą spadek współczynnika otwarć e-maili wpłynął na poziom sprzedaży skutera Bat, ponownie pobierzesz dane dotyczące modeli Bat i Lemon oraz porównasz współczynniki otwarć e-maili.

Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Wczytaj bazę danych `sql`da:

```
$ psql sql
```

2. Najpierw przyjrzyj się tabeli `emails`, aby zobaczyć, jakie informacje są dostępne. Pobierz pierwszych pięć wierszy tej tabeli:

```
sql>=# SELECT * FROM emails LIMIT 5;
```

W tabeli z rysunku 7.21 pokazane są informacje o e-mailach z pierwszych pięciu wierszy:

email_id	customer_id	email_subject	opened	clicked	bounced	sent_date	opened_date	clicked_date
1	18	Introducing A Limited Edition	f	f	f	2011-01-03 15:00:00		
2	30	Introducing A Limited Edition	f	f	f	2011-01-03 15:00:00		
3	41	Introducing A Limited Edition	t	f	f	2011-01-03 15:00:00	2011-01-04 10:41:11	
4	52	Introducing A Limited Edition	f	f	f	2011-01-03 15:00:00		
5	59	Introducing A Limited Edition	f	f	f	2011-01-03 15:00:00		

(5 rows)

Rysunek 7.21. E-maile od działu marketingu

Aby zbadać hipotezę, trzeba ustalić, czy e-mail został otwarty, kiedy to się stało, kto otworzył e-mail i czy klient ostatecznie kupił skuter. Jeśli e-mailowa kampania marketingowa skutecznie pomogła utrzymać wzrost liczby sprzedanych sztuk, można oczekiwać, że klient otworzył e-mail niedługo przed zakupem skutera.

Czas wysyłania e-maili, a także identyfikatory klientów, którzy otrzymali i otworzyli wiadomość, mogą pomóc w ustaleniu, czy e-mail zachęcił nabywcę do zakupu.

3. W celu sprawdzenia hipotezy musisz pobrać wartość kolumny `customer_id` nabywców skuterów Bat z tabel `emails` i `bat_sales`, a także kolumny `opened`, `sent_date`, `opened_date` i `email_subject` z tabeli `emails` oraz kolumnę `sales_transaction_date` z tabeli `bat_sales`. Ponieważ potrzebne są tylko rekordy z danymi o e-mailach do klientów, którzy kupili skuter Bat, złącz tabele na podstawie kolumny `customer_id`. Następnie wstaw wyniki do nowej tabeli — `bat_mails`:

```
sql># SELECT emails.email_subject, emails.customer_id, emails.opened,
      emails.sent_date, emails.opened_date, bat_sales.sales_transaction_date
      INTO bat_emails FROM emails INNER JOIN bat_sales ON
      bat_sales.customer_id=emails.customer_id ORDER BY
      bat_sales.sales_transaction_date;
```

Otrzymasz następujące dane wyjściowe:

```
SELECT 40190
```

4. Pobierz pierwszych 10 wierszy tabeli `bat_emails`. Uporządkuj wyniki według kolumny `sales_transaction_date`:

```
sql># SELECT * FROM bat_emails LIMIT 10;
```

Tabela z rysunku 7.22 przedstawia pierwszych 10 wierszy tabeli `bat_emails` uporządkowanych według kolumny `sales_transaction_date`.

email_subject	customer_id	opened	sent_date	opened_date	sales_transaction_date
A New Year, And Some New EVs	11678	f	2019-01-07 15:00:00		2016-10-10 00:00:00
A Brand New Scooter...and Car	40250	f	2014-05-06 15:00:00		2016-10-10 00:00:00
We Really Outdid Ourselves this Year	24125	f	2017-01-15 15:00:00		2016-10-10 00:00:00
Tis' the Season for Savings	31307	t	2015-11-26 15:00:00	2015-11-27 04:55:07	2016-10-10 00:00:00
25% off all EVs. It's a Christmas Miracle!	42213	f	2016-11-25 15:00:00		2016-10-10 00:00:00
Zoom Zoom Black Friday Sale	40250	f	2014-11-28 15:00:00		2016-10-10 00:00:00
Save the Planet with some Holiday Savings.	4553	f	2018-11-23 15:00:00		2016-10-10 00:00:00
The 2013 Lemon Scooter is Here	24125	t	2013-03-01 15:00:00	2013-03-02 14:43:34	2016-10-10 00:00:00
The 2013 Lemon Scooter is Here	40250	f	2013-03-01 15:00:00		2016-10-10 00:00:00
Save the Planet with some Holiday Savings.	40250	f	2018-11-23 15:00:00		2016-10-10 00:00:00

Rysunek 7.22. Informacje o e-mailach i transakcjach złączone na podstawie kolumny `customer_id`

Widać tu, że w podanym czasie niektóre e-maile nie zostały otwarte, a niektórzy klienci otrzymali wiele wiadomości. Gdy przyjrzyysz się tematowi e-maili, zobaczysz, że niektóre z nich w ogóle nie dotyczą skuterów firmy ZoomZoom.

5. Pobierz wszystkie wiersze, w których wartość z kolumny `sent_date` jest wcześniejsza niż wartość z kolumny `sales_transaction_date`. Uporządkuj dane według kolumny `customer_id` i ogranicz liczbę danych wyjściowych do 22 wierszy. To pomoże Ci ustalić, które e-maile zostały wysłane do poszczególnych klientów, zanim kupili skuter. Napisz następującą kwerendę:

```
sql># SELECT * FROM bat_emails WHERE sent_date < sales_transaction_date
      ORDER BY customer_id LIMIT 22;
```

W tabeli z rysunku 7.23 znajdziesz listę e-maili wysłanych do klientów przed datą z kolumny `sales_transaction_date`.

email_subject	customer_id	opened	sent_date	opened_date	sales_transaction_date
An Electric Car for a New Age	7	t	2015-04-01 15:00:00	2015-04-02 15:10:55	2019-04-25 00:00:00
The 2013 Lemon Scooter is Here	7	f	2013-03-01 15:00:00		2019-04-25 00:00:00
Tis' the Season for Savings	7	f	2015-11-26 15:00:00		2019-04-25 00:00:00
Black Friday. Green Cars.	7	f	2017-11-24 15:00:00		2019-04-25 00:00:00
We cut you a deal: 20% off a Blade	7	t	2014-09-18 15:00:00	2014-09-19 15:11:17	2019-04-25 00:00:00
Zoom Zoom Black Friday Sale	7	f	2014-11-28 15:00:00		2019-04-25 00:00:00
Like a Bat out of Heaven	7	f	2016-09-21 15:00:00		2019-04-25 00:00:00
Save the Planet with some Holiday Savings.	7	f	2018-11-23 15:00:00		2019-04-25 00:00:00
Shocking Holiday Savings On Electric Scooters	7	f	2013-11-29 15:00:00		2019-04-25 00:00:00
25% off all EVs. It's a Christmas Miracle!	7	t	2016-11-25 15:00:00	2016-11-26 03:55:30	2019-04-25 00:00:00
We Really Outdid Ourselves this Year	7	f	2017-01-15 15:00:00		2019-04-25 00:00:00
A Brand New Scooter...and Car	7	f	2014-05-06 15:00:00		2019-04-25 00:00:00
A New Year, And Some New EVs	7	f	2019-01-07 15:00:00		2019-04-25 00:00:00
Tis' the Season for Savings	22	f	2015-11-26 15:00:00		2017-08-14 00:00:00
Like a Bat out of Heaven	22	f	2016-09-21 15:00:00		2017-08-14 00:00:00
Zoom Zoom Black Friday Sale	22	t	2014-11-28 15:00:00	2014-11-29 11:31:03	2017-08-14 00:00:00
The 2013 Lemon Scooter is Here	22	f	2013-03-01 15:00:00		2017-08-14 00:00:00
25% off all EVs. It's a Christmas Miracle!	22	f	2016-11-25 15:00:00		2017-08-14 00:00:00
We Really Outdid Ourselves this Year	22	f	2017-01-15 15:00:00		2017-08-14 00:00:00
Shocking Holiday Savings On Electric Scooters	22	f	2013-11-29 15:00:00		2017-08-14 00:00:00
We cut you a deal: 20% off a Blade	22	f	2014-09-18 15:00:00		2017-08-14 00:00:00
An Electric Car for a New Age	22	f	2015-04-01 15:00:00		2017-08-14 00:00:00

Rysunek 7.23. E-maile wysłane do klientów przed datą transakcji

6. Usuń z tabeli `bat_emails` wiersze dotyczące e-maili wysłanych ponad sześć miesięcy przed rozpoczęciem produkcji. Widać, że niektóre e-maile zostały wysłane kilka lat przed datą transakcji. Możesz łatwo wyeliminować zbędne e-maile, usuwając te, które zostały wysłane przed rozpoczęciem produkcji modelu Bat. W tabeli `products` data rozpoczęcia produkcji skutera Bat to 10 października 2016 roku.

```
sql># DELETE FROM bat_emails WHERE sent_date < '2016-04-10';
```

W tym ćwiczeniu usuwamy niepotrzebne informacje z istniejącej tabeli. Jest to technika inna niż w poprzednich ćwiczeniach, gdzie tworzyliśmy wiele tabel, z których każda zawierała trochę inne informacje. Stosowana technika zależy od rozwiązywanego problemu — czy potrzebny jest zapis analiz do przesłedzenia, czy ważniejsza jest wydajność i mniejsza ilość zajmowanej pamięci?

7. Usuń też wiersze, w których data wysłania wiadomości jest późniejsza niż data zakupu, ponieważ takie e-maile nie są związane z transakcją:

```
sql># DELETE FROM bat_emails WHERE sent_date > sales_transaction_date;
```

8. Usuń wiersze, w których różnica między datą transakcji a datą wysłania wiadomości przekracza 30, ponieważ tu ważne są tylko e-maile przesłane niedługo przed zakupem skutera. E-mail sprzed roku prawdopodobnie nie wpłynął na decyzję o zakupie, jednak wiadomości nieodległe od daty transakcji mogły mieć z nią związek. Ustaw limit na miesiąc (30 dni) przed zakupem. Napisz następującą kwerendę:

```
sql># DELETE FROM bat_emails WHERE (sales_transaction_date-sent_date) > '30 days';
```

9. Ponownie sprawdź pierwsze 22 wiersze uporządkowane według kolumny `customer_id`. Uruchom w tym celu tę kwerendę:

```
sql># SELECT * FROM bat_emails ORDER BY customer_id LIMIT 22;
```


W tabeli z rysunku 7.24 pokazane są e-maile, w których różnica między datą transakcji a datą wysyłki jest mniejsza niż 30 dni.

email_subject	customer_id	opened	sent_date	opened_date	sales_transaction_date
25% off all EVs. It's a Christmas Miracle!	129	t	2016-11-25 15:00:00	2016-11-26 06:31:37	2016-11-28 00:00:00
A New Year, And Some New EVs	145	f	2019-01-07 15:00:00		2019-01-20 00:00:00
Black Friday. Green Cars.	150	f	2017-11-24 15:00:00		2017-12-19 00:00:00
Black Friday. Green Cars.	173	f	2017-11-24 15:00:00		2017-12-05 00:00:00
We Really Outdid Ourselves this Year	196	f	2017-01-15 15:00:00		2017-01-23 00:00:00
We Really Outdid Ourselves this Year	319	f	2017-01-15 15:00:00		2017-01-29 00:00:00
Like a Bat out of Heaven	369	f	2016-09-21 15:00:00		2016-10-13 00:00:00
Like a Bat out of Heaven	414	f	2016-09-21 15:00:00		2016-10-20 00:00:00
25% off all EVs. It's a Christmas Miracle!	418	f	2016-11-25 15:00:00		2016-12-21 00:00:00
A New Year, And Some New EVs	560	t	2019-01-07 15:00:00	2019-01-08 15:56:14	2019-01-29 00:00:00
We Really Outdid Ourselves this Year	600	f	2017-01-15 15:00:00		2017-01-18 00:00:00
A New Year, And Some New EVs	660	t	2019-01-07 15:00:00	2019-01-08 23:37:03	2019-01-08 00:00:00
A New Year, And Some New EVs	681	f	2019-01-07 15:00:00		2019-01-13 00:00:00
Black Friday. Green Cars.	806	t	2017-11-24 15:00:00	2017-11-25 16:59:40	2017-11-29 00:00:00
A New Year, And Some New EVs	881	t	2019-01-07 15:00:00	2019-01-08 21:07:28	2019-01-22 00:00:00
25% off all EVs. It's a Christmas Miracle!	934	t	2016-11-25 15:00:00	2016-11-26 09:22:45	2016-12-24 00:00:00
25% off all EVs. It's a Christmas Miracle!	983	f	2016-11-25 15:00:00		2016-11-29 00:00:00
A New Year, And Some New EVs	1060	f	2019-01-07 15:00:00		2019-01-27 00:00:00
25% off all EVs. It's a Christmas Miracle!	1288	f	2016-11-25 15:00:00		2016-12-11 00:00:00
25% off all EVs. It's a Christmas Miracle!	1317	f	2016-11-25 15:00:00		2016-12-13 00:00:00
A New Year, And Some New EVs	1400	t	2019-01-07 15:00:00	2019-01-08 15:01:00	2019-01-10 00:00:00
Save the Planet with some Holiday Savings.	1417	f	2018-11-23 15:00:00		2018-11-26 00:00:00

Rysunek 7.24. E-maile wysłane blisko daty sprzedaży

Na tym etapie przefiltrowaliśmy dostępne dane na podstawie dat wysłania i otwarcia e-maili. Gdy przyjrzyś się kolumnie email_subject, zobaczysz, że część wiadomości nie dotyczy skutera Bat (na przykład 25% of all EVs. It's a Christmas Miracle! i Black Friday. Green Cars). Te e-maile są związane z samochodami elektrycznymi, a nie ze skuterkami, dlatego można je pominąć w analizach.

10. Pobierz unikatowe wartości z kolumny email_subject, aby uzyskać listę różnych e-maili przesłanych do klientów:

```
sql># SELECT DISTINCT(email_subject) FROM bat_emails;
```

W tabeli z rysunku 7.25 pokazana jest lista unikatowych tematów e-maili.

```

email_subject
-----
Black Friday. Green Cars.
25% off all EVs. It's a Christmas Miracle!
A New Year, And Some New EVs
Like a Bat out of Heaven
Save the Planet with some Holiday Savings.
We Really Outdid Ourselves this Year
(6 rows)

```

Rysunek 7.25. Unikatowe tematy e-maili przesłanych do potencjalnych nabywców skutera Bat

11. Usuń wszystkie rekordy z tekstem Black Friday w temacie. Te e-maile nie są związane ze sprzedażą skuterów Bat:

```
sql># DELETE FROM bat_emails WHERE position('Black Friday' in
email_subject)>0;
```

Funkcja position w tym przykładzie służy do znajdowania wszystkich rekordów, w których łańcuch znaków Black Friday występuje w kolumnie email_subject, począwszy od pierwszego lub dowolnego dalszego znaku. W ten sposób usuwane są wszystkie wiersze ze słowami Black Friday w temacie wiadomości. Więcej informacji na temat funkcji do obsługi łańcuchów znaków w systemie PostgreSQL znajdziesz w dokumentacji dostępnej na stronie <https://www.postgresql.org/docs/current/functions-string.html>.

12. Usuń wszystkie wiersze z tekstem 25% off all EVs. It's a Christmas Miracle! lub A New Year, And Some New EVs w kolumnie email_subject:

```
sql># DELETE FROM bat_emails WHERE position('25% off all EV' in
email_subject)>0;
sql># DELETE FROM bat_emails WHERE position('Some New EV' in
email_subject)>0;
```

13. Na tym etapie masz ostateczny zbiór danych z e-mailami wysłanymi do klientów. Zlicz wiersze, które pozostały w tym zbiorze. Użyj do tego następującej kwerendy:

```
sql># SELECT count(sales_transaction_date) FROM bat_emails;
```

Informuje ona, że w zbiorze pozostało 401 wierszy:

```
count
-----
401
(1 row)
```

14. Teraz oblicz, jaki procent e-maili związanych ze sprzedażą został otwarty. Zlicz otwarte e-maile, używając poniższej kwerendy:

```
sql># SELECT count(opened) FROM bat_emails WHERE opened='t'
```

Informuje ona, że otwarto 98 e-maili:

```
count
-----
98
(1 row)
```

15. Zlicz klientów, którzy otrzymali e-maile i dokonali zakupu. W tym celu zlicz unikatowych klientów z tabeli bat_emails:

```
sql># SELECT COUNT(DISTINCT(customer_id)) FROM bat_emails;
```

Ta kwerenda informuje, że 396 klientów, którzy otrzymali e-mail, kupiło produkt:

```
count
-----
396
(1 row)
```

16. Zlicz unikatowych klientów, którzy dokonali zakupu:

```
sql># SELECT COUNT(DISTINCT(customer_id)) FROM bat_sales;
```

Oto dane wyjściowe tego kodu:

```
count
-----
6659
(1 row)
```

17. Oblicz procent klientów, którzy kupili skuter Bat po otrzymaniu e-maila:

```
sql># SELECT 396.0/6659.0 AS email_rate;
```

Oto dane wyjściowe tej kwerendy:

```
email_rate
-----
0.05946838864694398558
(1 row)
```

W tych obliczeniach w liczbach dodany jest separator części dziesiętnej (na przykład używana jest wartość 396.0 zamiast 396). Wynika to stąd, że wynik będzie równy mniej niż 1. Jeśli pominiesz separator części dziesiętnej, serwer SQL-a wykona dzielenie całkowitoliczbowe i zwróci wynik 0.

Tylko niecałe 6% klientów, którzy dokonali zakupu, otrzymało e-mail dotyczący skutera Bat. Jednocześnie 18% osób, które otrzymały e-mail, kupiło ten model. Jest to mocny argument na rzecz tego, że zwiększenie bazy klientów otrzymujących e-maile marketingowe może się przełożyć na większą sprzedaż skuterów Bat.

18. Ogranicz zakres dat do wszystkich transakcji sprzed 1 października 2016 roku. Umieść dane w nowej tabeli, `bat_emails_threewks`. Wcześniej zbadałeś już współczynnik otwarcia e-maili dla wszystkich dostępnych danych dotyczących skuterów Bat. Teraz sprawdź ten współczynnik dla pierwszych trzech tygodni, kiedy to odnotowano spadek sprzedaży:

```
sql>=# SELECT * INTO bat_emails_threewks FROM bat_emails WHERE
sales_transaction_date < '2016-11-01';
```

Otrzymasz następujące dane wyjściowe:

```
SELECT 82
```

19. Teraz zlicz otwarte e-maile z tego okresu:

```
sql>=# SELECT COUNT(opened) FROM bat_emails_threewks;
```

Widać, że w tym czasie otwarte zostały 82 e-maile:

```
count
-----
82
(1 row)
```

20. Teraz zlicz otwarte e-maile z pierwszych trzech tygodni:

```
sql>=# SELECT COUNT(opened) FROM bat_emails_threewks WHERE
opened='t';
```

Oto dane wyjściowe tego kodu:

```
count
-----
15
(1 row)
```

Tak więc w pierwszych trzech tygodniach otwartych zostało 15 e-maili.

21. Zlicz klientów, którzy otrzymali e-maile w pierwszych trzech tygodniach sprzedaży, a następnie dokonali zakupu:

```
sql>=# SELECT COUNT(DISTINCT(customer_id)) FROM bat_emails_threewks;
```

Informuje ona, że w pierwszych trzech tygodniach 82 klientów otrzymało e-maile:

```
count
-----
82
(1 row)
```

22. Oblicz procent klientów, którzy w pierwszych trzech tygodniach otworzyli e-mail dotyczący skutera Bat, a następnie dokonali zakupu:

```
sql>=# SELECT 15.0/82.0 AS sale_rate;
```

Oto obliczony procent:

```
sale_rate
-----
0.18292682926829268293
(1 row)
```

Około 18% klientów, którzy otrzymali e-mail dotyczący skutera Bat, dokonało zakupu w pierwszych trzech tygodniach sprzedaży. Jest to spójne ze współczynnikiem dla całego okresu sprzedaży tego modelu.

23. Oblicz, ilu unikatowych klientów odnotowano w pierwszych trzech tygodniach sprzedaży. Ta informacja zapewnia przydatny kontekst dla analizy obliczonych wcześniej procentów. Na przykład trzy sprzedaże na cztery e-maile oznaczałyby 75% skuteczności, jednak w tym scenariuszu korzystniejszy byłby niższy poziom skuteczności, ale przy znacznie większej bazie klientów. Informacje na temat dużej grupy klientów są bardziej przydatne, ponieważ lepiej odzwierciedlają całą bazę, a nie tylko niewielką próbkę. Wiesz już, że e-maile trafiły do 82 klientów:

```
sql>=# SELECT COUNT(DISTINCT(customer_id)) FROM bat_sales WHERE
sales_transaction_date < '2016-11-01';
```

Poniższe dane wyjściowe informują o 160 klientach, którzy dokonali zakupu przed 1 listopada 2016 roku:

```
count
-----
160
(1 row)
```

Masz więc 160 klientów z pierwszych trzech tygodni; 82 z tych osób otrzymały e-maile, co oznacza nieco ponad 50%. To znacznie więcej niż 6% z całego okresu sprzedaży skutera Bat.

Kod źródłowy z tego fragmentu jest dostępny na stronie <https://packt.live/3hkH3Vw>.

Po przeanalizowaniu skuteczności e-mailowej kampanii marketingowej skutera Bat potrzebna jest grupa kontrolna (porównawcza), aby ustalić, czy uzyskane wyniki są spójne z danymi dla innych produktów. Bez grupy porównawczej nie wiadomo, czy kampania e-mailowa dotycząca skutera Bat udała się, zakończyła się niepowodzeniem czy dała przeciętne wyniki. Skuteczność tej kampanii zbadasz w następnym ćwiczeniu.

Ćwiczenie 7.05 — analiza skuteczności e-mailowej kampanii marketingowej

W tym ćwiczeniu zbadasz skuteczność e-mailowej kampanii marketingowej dotyczącej skutera Lemon, aby porównać ją z kampanią dla modelu Bat. Hipoteza jest taka, że jeśli skuteczność kampanii dla skutera Bat jest porównywalna jak innych kampanii (na przykład dla modelu Lemon z 2013 roku), spadek sprzedaży nie wynika z różnic w kampaniach e-mailowych.

Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Wczytaj bazę danych sqlda:

```
$ psql sqlda
```

2. Usuń istniejącą tabelę lemon_sales:

```
sql=# DROP TABLE lemon_sales;
```

3. Dla skutera Lemon z 2013 roku kolumna product_id ma wartość 3. Pobierz z tabeli sales kolumny customer_id i sales_transaction_date dotyczące tego modelu.

Wstaw te informacje do tabeli lemon_sales:

```
sql=# SELECT customer_id, sales_transaction_date INTO lemon_sales
FROM sales WHERE product_id=3;
```

4. Pobierz z tabeli emails wszystkie informacje o klientach, którzy zakupili model Lemon. Umieść te informacje w nowej tabeli, lemon_emails:

```
sql=# SELECT emails.customer_id, emails.email_subject, emails.opened,
emails.sent_date, emails.opened_date, lemon_sales.sales_transaction_date INTO
lemon_emails FROM emails INNER JOIN lemon_sales
ON emails.customer_id=lemon_sales.customer_id;
```

5. Usuń wszystkie e-maile wysłane przed rozpoczęciem produkcji skutera Lemon. Potrzebna będzie do tego data początku produkcji:

```
sql=# SELECT production_start_date FROM products WHERE product_id=3;
```

W poniższych danych wyjściowych pokazana jest kolumna production_start_date:

```
production_start_date
-----
2013-05-01 00:00:00
(1 row)
```

6. Teraz usuń e-maile wysłane przed rozpoczęciem produkcji skutera Lemon:

```
sql=# DELETE FROM lemon_emails WHERE sent_date < '2013-05-01';
```

7. Usuń też wszystkie wiersze, w których data wysłania e-maila jest późniejsza niż data zakupu (sales_transaction_date):

```
sql=# DELETE FROM lemon_emails WHERE sent_date > sales_transaction_date;
```

8. Usuń wszystkie wiersze, w których data wysyłki jest wcześniejsza o ponad 30 dni od daty zakupu (sales_transaction_date):

```
sql=# DELETE FROM lemon_emails WHERE (sales_transaction_date -
sent_date) > '30 days';
```

9. Usuń z tabeli `lemon_emails` wszystkie wiersze, w których temat e-maila nie jest związany ze skuterami Lemon. Wcześniej jednak poszukaj wszystkich unikatowych e-maili:

```
sql>=# SELECT DISTINCT(email_subject) FROM lemon_emails;
```

Na rysunku 7.26 pokazane są unikatowe tematy e-maili.

```

----- email_subject -----
Tis' the Season for Savings
25% off all EVs. It's a Christmas Miracle!
A Brand New Scooter...and Car
Like a Bat out of Heaven
Save the Planet with some Holiday Savings.
Shocking Holiday Savings On Electric Scooters
We Really Outdid Ourselves this Year
An Electric Car for a New Age
We cut you a deal: 20% off a Blade
Black Friday. Green Cars.
Zoom Zoom Black Friday Sale
(11 rows)

```

Rysunek 7.26. E-maile wysłane w kampanii skutera Lemon

10. Teraz usuń tematy niepowiązane ze skuterem Lemon. Użyj polecenia `DELETE`:

```

sql>=# DELETE FROM lemon_emails WHERE POSITION('25% off all EVs.' in
email_subject)>0;
sql>=# DELETE FROM lemon_emails WHERE POSITION('Like a Bat out of Heaven'
in email_subject)>0;
sql>=# DELETE FROM lemon_emails WHERE POSITION('Save the Planet' in
email_subject)>0;
sql>=# DELETE FROM lemon_emails WHERE POSITION('An Electric Car' in
email_subject)>0;
sql>=# DELETE FROM lemon_emails WHERE POSITION('We cut you a deal'
in email_subject)>0;
sql>=# DELETE FROM lemon_emails WHERE POSITION('Black Friday. Green Cars.'
in email_subject)>0;
sql>=# DELETE FROM lemon_emails WHERE POSITION('Zoom' in email_subject)>0;

```

11. Teraz sprawdź, ile e-maili do klientów z tabeli `lemon_scooter` zostało otwartych:

```
sql>=# SELECT COUNT(opened) FROM lemon_emails WHERE opened='t';
```

Widać tu, że otwartych zostało 128 e-maili:

```

count
-----
128
(1 row)

```

12. Wyświetl listę klientów, którzy otrzymali e-mail i dokonali zakupu:

```
sql>=# SELECT COUNT(DISTINCT(customer_id)) FROM lemon_emails;
```

W poniższych danych wyjściowych widać, że 506 klientów zakupiło skuter po otrzymaniu e-maila:

```

count
-----
506
(1 row)

```

13. Oblicz procent klientów, którzy otworzyli otrzymane e-maile i dokonali zakupu:

```
sql># SELECT 128.0/506.0 AS email_rate;
```

Poniżej widać, że 25% klientów otworzyło e-maile i kupiło skuter:

```
email_rate
-----
0.25296442687747035573
(1 row)
```

14. Ustal liczbę unikatowych klientów, którzy kupili skuter:

```
sql># SELECT COUNT(DISTINCT(customer_id)) FROM lemon_sales;
```

Widać, że zakupu dokonało 13854 klientów:

```
count
-----
13854
(1 row)
```

15. Oblicz procent klientów, którzy dokonali zakupu po otrzymaniu e-maila.

Pozwoli Ci to porównać wynik z wartościami dla skutera Bat:

```
sql># SELECT 506.0/13854.0 AS email_sales;
```

Dane wyjściowe z tych obliczeń to 36%:

```
email_sales
-----
0.03652374765410711708
(1 row)
```

16. Pobierz z tabeli `lemon_emails` wszystkie rekordy dotyczące transakcji z pierwszych trzech tygodni od rozpoczęcia produkcji. Zapisz wyniki w nowej tabeli, `lemon_emails_threewks`:

```
sql># SELECT * INTO lemon_emails_threewks FROM lemon_emails WHERE
sales_transaction_date < '2013-06-01';
```

Oto dane wyjściowe:

```
SELECT 0;
```

17. Zlicz e-maile wysłane do nabywców skutera Lemon w pierwszych trzech tygodniach sprzedaży:

```
sql># SELECT COUNT(sales_transaction_date) FROM lemon_emails_threewks;
```

Oto dane wyjściowe tego kodu:

```
count
-----
0
(1 row)
```

Uzyskałeś wiele ciekawych informacji. Widać tu, że 25% klientów, którzy otworzyli e-mail, dokonało zakupu. Jest to wartość znacznie wyższa niż 18% dla skutera Bat. Obliczyłeś też, że nieco ponad 3,6% klientów, którzy kupili skuter Lemon, otrzymało e-mail. Jest to wartość znacznie niższa niż 6% dla nabywców skuterów Bat. Ostatnia ciekawa informacja jest taka, że żaden z nabywców

skutera Lemon nie otrzymał e-maila w pierwszych trzech tygodniach od wprowadzenia produktu na rynek. Można to porównać z 82 nabywcami skuterów Bat, którzy stanowią około 50% wszystkich klientów z pierwszych trzech tygodni sprzedaży tego modelu.

Kod źródłowy z tego fragmentu książki jest dostępny na stronie <https://packt.live/3cTCY7p>.

W tym ćwiczeniu zbadaliśmy skuteczność e-mailowej kampanii marketingowej skutera Lemon, aby porównać ją z kampanią skutera Bat. Użyliśmy do tego różnych technik SQL-a.

Wnioski

Po zebraniu różnych informacji na temat czasu wprowadzenia produktu na rynek, cen skuterów i kampanii marketingowych możesz wyciągnąć kilka wniosków dotyczących stawianych hipotez:

- W „Ćwiczeniu 7.03 — analiza czasu rozpoczęcia sprzedaży” zebraliśmy dane sugerujące, że czas wprowadzenia skutera na rynek może być przyczyną spadku sprzedaży po dwóch pierwszych tygodniach, choć nie da się tego udowodnić.
- Widoczna jest korelacja między początkowym tempem sprzedaży a ceną skuterów. Niższa cena skutkuje wyższą sprzedażą („Zadanie 7.02 — analiza hipotezy dotyczącej różnic w cenie sprzedaży”).
- Liczba sztuk sprzedanych w pierwszych trzech tygodniach nie jest bezpośrednio skorelowana z ceną sprzedaży produktu („Zadanie 7.02 — analiza hipotezy dotyczącej różnic w cenie sprzedaży”).
- Dane sugerują, że skuteczna kampania marketingowa może zwiększać początkową sprzedaż, a większy współczynnik otwarć e-maili skutkuje wyższą sprzedażą („Ćwiczenie 7.04 — analiza zależności wzrostu sprzedaży od współczynnika otwarć e-maili”). Występuje też zależność między liczbą klientów, którzy otrzymali e-maile, a wzrostem sprzedaży („Ćwiczenie 7.05 — analiza skuteczności e-mailowej kampanii marketingowej”).
- W ciągu pierwszych trzech tygodni sprzedano więcej skuterów Bat niż modeli Lemon lub Bat Limited („Zadanie 7.02 — analiza hipotezy dotyczącej różnic w cenie sprzedaży”).

Badania terenowe

Na tym etapie przeprowadziłeś analizy post hoc (czyli analizy danych wykonywane po zajściu zdarzenia) i uzyskałeś dowody na rzecz kilku teorii wyjaśniających, dlaczego sprzedaż skuterów Bat spadła po dwóch pierwszych tygodniach. Nie możesz jednak potwierdzić słuszności tych teorii, ponieważ nie możesz ich od siebie oddzielić.

W takiej sytuacji musisz posłużyć się następnym dostępnym narzędziem — badaniami terenowymi. Zgodnie z nazwą polegają one na testowaniu hipotez w praktyce, na przykład po wprowadzeniu na rynek nowego produktu lub przy badaniu aktualnej sprzedaży.

Jednym z najczęściej stosowanych badań terenowych są testy A/B. Polegają one na losowym podziale klientów na dwie grupy (A i B), udostępnieniu im nieco innych procesów lub środowisk i obserwowaniu wyników. W naszym przykładzie losowo przypisujesz klientów do grupy A lub B, do klientów z grupy A kierujesz nową kampanię marketingową, do klientów z grupy B zaś istniejącą kampanię, a następnie możesz obserwować sprzedaż i interakcje, aby zobaczyć, która kampania daje lepsze efekty.

Podobnie jeśli chcesz przetestować czas wprowadzenia produktu na rynek, możesz na przykład udostępnić go w Północnej Kalifornii na początku listopada, a w Południowej Kalifornii na początku grudnia i obserwować różnice.

Istotą badań terenowych jest to, że jeśli nie sprawdzisz analiz post hoc w praktyce, nie przekonasz się, czy hipotezy są prawdziwe. Przetestowanie hipotez wymaga tylko zmiany badanych warunków — na przykład daty wprowadzenia produktu na rynek. W celu potwierdzenia wyników analiz post hoc możesz zalecić, aby zespół sprzedażowy zastosował jeden lub kilka opisanych dalej scenariuszy i monitorował sprzedaż w czasie rzeczywistym w celu ustalenia przyczyn spadku liczby transakcji:

- Wprowadzenie następnego modelu skutera w różnych porach roku w dwóch regionach o podobnym klimacie i zbliżonych aktualnych danych sprzedażowych. To pomoże stwierdzić, czy pora rozpoczęcia sprzedaży wpływa na jej poziom.
- Udostępnienie następnego modelu skutera w tym samym czasie, ale w różnej cenie w regionach o zbliżonych aktualnych wynikach i obserwowanie różnic w poziomie sprzedaży.
- Rozpoczęcie sprzedaży następnego modelu skutera w tym samym czasie i w tej samej cenie w regionach o podobnych aktualnych wynikach i przeprowadzenie dwóch różnych e-mailowych kampanii marketingowych. Wymaga to obserwacji klientów, którzy uczestniczyli w obu kampaniach, i monitorowania poziomu sprzedaży.

Podsumowanie

Właśnie ukończyłeś swoje pierwsze praktyczne analizy danych z wykorzystaniem SQL-a. Dzięki lekturze tego rozdziału posiadasz umiejętności potrzebne do postawienia hipotez na temat problemu i systematycznego zbierania danych potrzebnych do ich potwierdzenia lub odrzucenia. Studium przypadku rozpoczęliśmy od stosunkowo trudnego problemu — wyjaśnienia zaobserwowanego spadku sprzedaży. Odkryliśmy dwa możliwe źródła odnotowanych wyników (czas wprowadzenia produktu na rynek i kampania marketingowa) oraz odrzuciliśmy jedno z wyjaśnień (cena sprzedaży).

Zrozumienie i umiejętność zastosowania metody naukowej do rozwiązywania problemów są niezbędne dla każdego analityka danych oraz zwiększają skuteczność analiz i pomagają wykryć ciekawe wątki badań. W tym rozdziale wykorzystales umiejętności z zakresu SQL-a zdobyte podczas lektury tej książki — od prostych instrukcji SELECT, przez agregowanie złożonych typów danych, po metody bazujące na oknie. Po zakończeniu tego rozdziału powinieneś umieć samodzielnie kontynuować i powtarzać analizy tego rodzaju we własnych projektach, aby wyciągać praktyczne wnioski.

Dotarłeś do końca książki. W kolejnych rozdziałach przedstawiliśmy Ci różne rodzaje danych i sposoby znajdowania w nich wzorców. Wiesz już też, jak wykorzystać rozbudowane możliwości SQL-a do porządkowania danych, ich przetwarzania oraz identyfikowania interesujących wzorców. Ponadto pokazaliśmy, że SQL można łączyć z innymi systemami i optymalizować w celu przeprowadzania analiz na dużą skalę. Na zakończenie wykorzystaliśmy SQL w studium przypadku, aby pomóc firmie w działalności.

Jednak opisane umiejętności to tylko początek Twojej pracy. Relacyjne bazy danych są stale modyfikowane i nieustannie rozwijane są nowe funkcje. Dostępnych jest też wiele zaawansowanych technik statystycznych, które nie zostały opisane w tej książce. Dlatego choć ta książka jest przewodnikiem po analityce danych i bezcennym narzędziu, jakim jest SQL, stanowi tylko pierwszy krok w — mamy nadzieję — satysfakcjonującej podróży.

Rozdział 1.

Wprowadzenie do SQL-a dla analityków

Zadanie 1.01 — klasyfikowanie nowego zbioru danych

Rozwiązanie

1. Jednostką obserwacji jest sprzedaż samochodu.
2. Kolumny Data i Wartość Sprzedaży zawierają dane ilościowe, natomiast kolumna Marka zawiera dane jakościowe.
3. Choć istnieje wiele sposobów na przekształcenie kolumny Marka na dane ilościowe, powszechnie przyjęta metoda polega na odwzorowaniu każdego z modeli na liczbę. Na przykład marce Ford można przypisać wartość 1, marce Honda wartość 2, marce Mazda wartość 3, marce Toyota wartość 4, marce Mercedes wartość 5, a marce Chevrolet wartość 6.

Zadanie 1.02 — eksplorowanie danych sprzedażowych z salonu samochodowego

Rozwiązanie

1. Otwórz pusty arkusz w programie Microsoft Excel.
2. Przejdź do zakładki *Dane* i wybierz opcję *Pobieranie danych zewnętrznych/ Z pliku tekstowego*.
3. Znajdź ścieżkę do pliku *dealerships.csv* i kliknij przycisk *OK*.

4. Wybierz opcję *Rozdzielany* w oknie dialogowym *Kreator importu tekstu*. Upewnij się, że import jest rozpoczynany od wiersza 1. Następnie kliknij *Dalej*.
5. Wybierz ogranicznik używany w pliku. W plikach CSV tradycyjnie ogranicznikami są przecinki; zawsze należy stosować ograniczniki odpowiednie dla zbiorów danych. Teraz kliknij *Dalej*.
6. Wybierz *Ogólne* w sekcji *Format danych kolumny*. Kliknij przycisk *Zaawansowane* i w nowym oknie jako *Separator dziesiętny* wybierz kropkę, po czym kliknij *OK*. Następnie kliknij *Zakończ*.
7. W oknie dialogowym z pytaniem *Gdzie chcesz umieścić dane?* wybierz opcję *Istniejący arkusz* i nie zmieniaj wartości w polu tekstowym obok tej opcji. Kliknij przycisk *OK*. Powinieneś zobaczyć dane podobne do tych z rysunku 1.50.

Lokalizacja	Roczna Sprzedaż Netto	Liczba Zatrudnionych Kobiet
Millburn, NJ	150803012	27
Los Angeles, CA	110872084	17
Houston, TX	183945873	22
Miami, FL	156355396	18
San Mateo, CA	143108603	17
Seattle, WA	142755480	33
Arlington VA	144772604	28
Portland, OR	179608438	32
Reno, NV	145101244	19
Chicago, IL	171491596	24
Atlanta, GA	198386988	27
Orlando, FL	180188054	24
Jacksonville, FL	158479693	32
Round Rock, TX	181820474	27
Phoenix, AZ	95512810,7	18
Charlotte, NC	199653776	32
Philadelphia, PA	193111679	31
Kansas City, MO	176816637	35
Dallas, TX	168769837	33
Boston, MA	350520724	20

Rysunek 1.50. Wczytany plik dealerships.csv

Wygląd histogramów zależy od wybranych parametrów. Tu histogram powinien wyglądać podobnie jak na rysunku 1.51.



Rysunek 1.51. Histogram przedstawiający liczbę zatrudnionych kobiet

8. Oblicz średnią i medianę zgodnie z krokami z „Ćwiczenia 1.03 — obliczanie miar tendencji centralnej dla sprzedaży dodatków”. Średnia sprzedaż powinna wynosić **171 603 750,13** dolara, a mediana — **170 130 716,50**.
9. Wykonując kroki podobne do tych z „Ćwiczenia 1.04 — obliczanie dyspersji dla sprzedaży dodatków”, oblicz odchylenie standardowe wartości sprzedaży. Powinno ono wynieść **50 152 290,42** dolara.
10. Wyniki salonu z Bostonu (MA) to wartość odstająca. Można to wykazać graficznie albo metodą rozstępu ćwiartkowego.
Powinieneś uzyskać cztery kwintyle widoczne na rysunku 1.52.

Kwintyle	Wartość
1	144439803,80
2	157629972,20
3	177933357,40
4	185779034,20

Rysunek 1.52. Kwintyle i ich wartości

11. Gdy usuniesz dane odstające (salon w Bostonie), powinieneś uzyskać współczynnik korelacji równy **0,55**. Ta wartość wskazuje na silną korelację liczby zatrudnionych kobiet i poziomu sprzedaży w salonie. Choć może to wskazywać, że większy żeński personel prowadzi do wyższych przychodów, takie wyniki mogą być spowodowane trzecim czynnikiem. W tym scenariuszu większe salony zatrudniają więcej osób, co oznacza, że pracuje w nich także więcej kobiet. Możliwe są też inne wyjaśnienia uzyskanej korelacji.

Zadanie 1.03 — kwerenda SELECT z podstawowymi słowami kluczowymi dotycząca tabeli customers

Rozwiązanie

1. Otwórz wybranego klienta SQL-a i nawiąż połączenie z bazą sql da. Zbadaj schemat tabeli customers, używając listy rozwijanej schematu. Zwróć uwagę na nazwy kolumn, podobnie jak wtedy, gdy w „Ćwiczeniu 1.06 — kwerenda SELECT z podstawowymi słowami kluczowymi dotycząca tabeli salespeople” badałeś tabelę salespeople.
2. Wykonaj poniższą kwerendę, aby pobrać e-maile klientów ze stanu Floryda w porządku alfabetycznym:

```
SELECT
    email
FROM
```

```

customers
WHERE
  state='FL'
ORDER BY
  email;

```

Dane wyjściowe tego kodu są pokazane na rysunku 1.53.

	email text
1	aachrameevu44@goo.gl
2	aambresinInt@walmart.com
3	aanstiss12af@eepurl.com
4	aantonove9l@last.fm
5	aarnaudet1v3@cisco.com
6	aarsmithxoe@dion.ne.jp
7	aastle11rg@slate.com
8	aaxelbey77x@cocolog-nifty.com
9	aazemary2f@washingtonpost.com
10	ababarm8m@ow.ly
11	abarkessi6f@wikimedia.org

Rysunek 1.53. Adresy e-mail klientów z Florydy w porządku alfabetycznym

- Wykonaj poniższą kwerendę, aby pobrać imiona, nazwiska i adresy e-mail klientów firmy ZoomZoom z Nowego Jorku. Klientów uporządkuj alfabetycznie najpierw według nazwisk, a następnie według imion:

```

SELECT
  first_name, last_name, email
FROM
  customers
WHERE
  city='New York City'
  AND state='NY'
ORDER BY
  last_name, first_name;

```

Na rysunku 1.54 pokazane są dane wyjściowe tego kodu.

- Wykonaj poniższą kwerendę, aby pobrać dane wszystkich klientów, którzy podali numer telefonu. Uporządkuj dane według daty dodania klienta do bazy:

```

SELECT
  *
FROM
  customers
WHERE
  phone IS NOT NULL
ORDER BY
  date_added;

```

Dane wyjściowe tego kodu są pokazane na rysunku 1.55.

	first_name text	last_name text	email text
1	Nell	Abdy	nabdyec4@fema.gov
2	Thomasine	Absolon	tabsolonomk@forbes.com
3	Ram	Acheson	racheson1ai@bloglovin.com
4	Pru	Achrameev	pachrameev2sr@example.com
5	Jandy	Adamowicz	jadamowiczb1w@clickbank.net
6	Kati	Adrian	kadrianeem@51.la
7	Orly	Aers	oaersx61@redcross.org
8	Bradney	Aglione	baglione5n@usgs.gov
9	Mellicent	Ainslee	mainsleir0@abc.net.au
10	Fergus	Aireton	fairetonq16@yellowpages.com
11	Ugo	Aldam	ualdamhnc@wikimedia.org

Rysunek 1.54. Uporządkowane alfabetycznie informacje o klientach z Nowego Jorku

customer_id bigint	title text	first_name text	last_name text	suffix text	email text	gender text	ip_address text	phone text	street_address text
2625	[null]	Binky	Dawtrey	[null]	bdawtr...	M	15.75.236.78	804-990...	0353 Iowa Road
6173	[null]	Danila	Gristwood	[null]	dgrist...	F	254.239.58.1...	832-157...	79865 Hagan Terr...
13390	[null]	Danika	Lough	[null]	dlough...	F	188.19.7.207	212-769...	38463 Forest Dal...
7486	[null]	Ciro	Ferencowicz	[null]	cferen...	M	8.151.167.184	786-458...	61 Village Crossing
17099	[null]	Pearla	Halksworth	[null]	phalks...	F	114.138.82.24	541-196...	130 Marcy Crossi...
18685	[null]	Ingram	Crossman	[null]	icross...	M	207.145.1.202	503-352...	86 Michigan Junc...
30046	[null]	Nanete	Hassur	[null]	nhassu...	F	232.115.170....	209-364...	13961 Steensland...
35683	[null]	Betteanne	Rulf	[null]	brulfrj6...	F	52.208.248.90	503-396...	1 Cordelia Crossing
22640	[null]	Shana	Nugent	[null]	snuge...	F	207.239.127....	202-378...	96725 Cordelia La...
34189	[null]	Devlin	Barhems	[null]	dbarhe...	M	180.175.21.2...	240-895...	0 Park Meadow St...
46277	Mr	Salomon	Rillatt	[null]	srillatt...	M	33.205.88.187	504-700...	5799 Thackeray C...

Rysunek 1.55. Klienci, którzy podali numer telefonu, uporządkowani według daty dodania do bazy

Dane wyjściowe widoczne na tym rysunku pomogą menedżerowi ds. marketingu w przeprowadzeniu kampanii i zwiększeniu sprzedaży.

Kod źródłowy z tego fragmentu książki znajdziesz na stronie <https://packt.live/3cVSBLE>.

Zadanie 1.04 — tworzenie i modyfikowanie tabel na potrzeby działań marketingowych

Rozwiązanie

1. Otwórz wybranego klienta SQL-a i nawiąż połączenie z bazą sql da.
2. Uruchom poniższą kwerendę, aby utworzyć tabelę z klientami z Nowego Jorku:

```
CREATE TABLE customers_nyc AS (
SELECT
*
FROM
customers
WHERE
city='New York City'
AND state='NY');
```

3. Uruchom poniższy kod, aby wyświetlić dane wyjściowe:

```
SELECT * FROM customers_nyc;
```

Dane wyjściowe są pokazane na rysunku 1.56.

customer_id bigint	title text	first_name text	last_name text	suffix text	email text	gender text	ip_address text	phone text	street_address text	city text	state text
52	[null]	Giusto	Backe	[null]	gbacke1f@diggg.com	M	26.56.68.189	212-959...	6 Onsgard Terrace	New ...	NY
162	[null]	Artair	Betchley	[null]	abetchley4h@dagondesign.com	M	108.147.128...	[null]	7 Boyd Road	New ...	NY
374	[null]	Verge	Esel	[null]	veselad@vistaprint.com	M	58.238.20.156	917-653...	6 Algoma Park	New ...	NY
406	[null]	Rozina	Jeal	[null]	rjealb9@howstuffworks.com	F	50.235.32.29	917-610...	64653 Homewoo...	New ...	NY
456	Rev	Cybil	Noke	[null]	cnokecn@jigsy.com	F	5.31.139.106	212-306...	88 Sycamore Park...	New ...	NY
472	[null]	Rawley	Yegorov	[null]	ryegorovd3@google.es	M	183.199.243...	212-560...	872 Old Shore Par...	New ...	NY
496	[null]	Layton	Spolton	[null]	lspoltondr@free.fr	M	108.112.8.165	646-900...	7 Old Gate Drive	New ...	NY
1028	[null]	Issy	Andrieux	[null]	iandrieuxs@deli.com	F	199.50.5.37	212-206...	33337 Dahle Way	New ...	NY
1037	[null]	Magdalene	Veryard	[null]	mveryardss@behance.net	F	93.201.129.2...	[null]	41028 Katie Junct...	New ...	NY
1063	[null]	Juliet	Beadles	[null]	jbeadlesi@time.com	F	47.96.88.226	212-645...	34984 Goodland ...	New ...	NY

Rysunek 1.56. Tabela z klientami z Nowego Jorku

4. Następnie uruchom poniższą kwerendę, aby usunąć użytkowników mających kod pocztowy 10014:

```
DELETE
FROM
customers_nyc
WHERE
postal_code='10014';
```

5. Wykonaj poniższą kwerendę, by dodać nową kolumnę event:

```
ALTER TABLE customers_nyc ADD COLUMN event text;
```


6. Zaktualizuj tabelę `customers_nyc` i zapisz w kolumnie `event` wartość `Impreza w ramach podziękowań`:

```
UPDATE
  customers_nyc
SET
  event = 'Impreza w ramach podziękowań';
```

7. Uruchom poniższy kod, aby wyświetlić dane wyjściowe:

```
SELECT
  *
FROM
  customers_nyc;
```

Dane wyjściowe tego kodu są pokazane na rysunku 1.57.

customer_id	title	first_name	last_name	suffix	email	gender	ip_address	phone	street_address	city	state	postal_code	latitude	longitude	date_added	event
1211	[null]	Gwyneth	McCobb	[null]	gmccobb...	F	38.182.15...	[null]	4 Jana Park	New Y...	NY	10160	40.7808	-73.9772	2014-01-08 00:00:00	Impreza w ramach podziękowań
4535	[null]	Alejandra	Jinda	[null]	ajinda3h...	F	232.155.2...	347-488-5...	3475 Kedzie PL	New Y...	NY	10280	40.7105	-74.0163	2011-06-05 00:00:00	Impreza w ramach podziękowań
7196	[null]	Patrice	Yarker	[null]	pyarker5...	F	96.131.45...	212-975-0...	2222 Killdeer P...	New Y...	NY	10175	40.7543	-73.9798	2016-09-02 00:00:00	Impreza w ramach podziękowań
10067	[null]	Etheline	Gledstane	[null]	egledsta...	F	22.161.22...	[null]	124 3rd Trail	New Y...	NY	10160	40.7808	-73.9772	2010-04-29 00:00:00	Impreza w ramach podziękowań
10119	[null]	Carolann	Hodgen	[null]	chodgen...	F	245.199.2...	212-865-8...	63 Waywood C...	New Y...	NY	10150	40.7808	-73.9772	2014-05-15 00:00:00	Impreza w ramach podziękowań
12951	[null]	Filberte	Jannequin	II	fjannequ...	M	41.156.14...	212-862-7...	6 Florence Point	New Y...	NY	10090	40.7808	-73.9772	2010-07-31 00:00:00	Impreza w ramach podziękowań
13212	[null]	Polly	Greenshiels	[null]	pgreensh...	F	251.209.1...	[null]	5 Mallory Road	New Y...	NY	10009	40.7262	-73.9796	2014-08-30 00:00:00	Impreza w ramach podziękowań
15883	[null]	Abey	Tapping	[null]	atapping...	M	18.67.193...	[null]	80 Kedzie Drive	New Y...	NY	10045	40.7086	-74.0087	2017-10-09 00:00:00	Impreza w ramach podziękowań
15907	[null]	Shaughn	Suckling	[null]	ssuckling...	M	157.85.97...	718-829-3...	59 Carey Park	New Y...	NY	10004	40.6964	-74.0253	2016-03-06 00:00:00	Impreza w ramach podziękowań
19303	[null]	Sheffield	Jowle	[null]	sjowlew...	M	115.22.37...	212-179-2...	5682 1st Trail	New Y...	NY	10160	40.7808	-73.9772	2011-03-24 00:00:00	Impreza w ramach podziękowań
21786	[null]	Morton	Wantling	[null]	mwantlin...	M	29.247.44...	212-323-3...	798 Reinke Hill	New Y...	NY	10034	40.8662	-73.9221	2014-03-12 00:00:00	Impreza w ramach podziękowań
21806	[null]	Erin	Praton	[null]	epraton...	F	245.131.1...	646-912-4...	51497 Crowley...	New Y...	NY	10004	40.6964	-74.0253	2016-05-23 00:00:00	Impreza w ramach podziękowań
25415	[null]	Dorrey	McGinnell	[null]	dmcginn...	F	223.150.3...	212-229-7...	77235 Knutson...	New Y...	NY	10270	40.7069	-74.0082	2010-08-17 00:00:00	Impreza w ramach podziękowań
28871	[null]	Hyacinthie	McElhargy	[null]	hmcelhar...	F	121.79.22...	212-903-9...	18 Stang Way	New Y...	NY	10009	40.7262	-73.9796	2016-07-25 00:00:00	Impreza w ramach podziękowań
28991	[null]	Cherlyn	Wooddisse	[null]	cwooddis...	F	185.105.0...	[null]	241 Doe Cross...	New Y...	NY	10165	40.7524	-73.9791	2016-07-24 00:00:00	Impreza w ramach podziękowań
29029	[null]	Andrey	Sussex	[null]	asussex...	M	80.249.54...	212-242-5...	15727 Emmet ...	New Y...	NY	10260	40.7808	-73.9772	2013-11-28 00:00:00	Impreza w ramach podziękowań

Rysunek 1.57. Tabela `customers_nyc` z kolumną `event` z wartością `Impreza w ramach podziękowań`

8. Teraz zgodnie z prośbą menedżera usuń tabelę `customers_nyc`. Użyj polecenia `DROP TABLE`:

```
DROP TABLE customers_nyc;
```

To spowoduje usunięcie tabeli `customers_nyc` z bazy danych.

Kod źródłowy z tego fragmentu książki jest dostępny pod adresem <https://packt.live/3dWtFVG>.

Rozdział 2. Przygotowywanie danych za pomocą SQL-a

Zadanie 2.01 — używanie SQL-a do tworzenia modelu wspomagającego sprzedaż

Rozwiązanie

1. Otwórz klienta SQL-a i nawiąż połączenie z bazą danych.
2. Użyj złączenia INNER JOIN, aby złączyć tabelę customers z tabelą sales.
Użyj złączenia INNER JOIN, aby złączyć tabelę products z tabelą sales.
Użyj złączenia LEFT JOIN, aby złączyć tabelę dealerships z tabelą sales.
3. Teraz zwróć wszystkie kolumny tabel customers i products, po czym zwróć kolumnę dealership_id tabeli sales, ale jeśli jej wartość to NULL, zastąp ją wartością -1.
4. Dodaj kolumnę high_savings. Zapisz w niej wartość 1, jeśli cena sprzedaży była o co najmniej 500 niższa niż cena z kolumny base_msrp. W przeciwnym razie zapisz w tej kolumnie 0. Tę kwerendę można wykonać na wiele sposobów. Oto jeden z nich:

```
SELECT
    c.*,
    p.*,
    COALESCE(s.dealership_id, -1),
    CASE WHEN p.base_msrp - s.sales_amount >500
        THEN 1
        ELSE 0
    END AS high_savings
FROM
    sales s
    INNER JOIN customers c
        ON c.customer_id=s.customer_id
    INNER JOIN products p
        ON p.product_id=s.product_id
    LEFT JOIN dealerships d
        ON s.dealership_id = d.dealership_id;
```

Dane wyjściowe tego kodu są pokazane na rysunku 2.27.

customer_id bigint	title text	first_name text	last_name text	su text	email text	gender text	ip_address text	phone text	street_address text	city text	state text	postal_code text	latitude double precision	longitude double precision	date_added timestamp without time zone	
1	[null]	Ariana	Rivetes	[null]	arivete...	F	98.36.172.246	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	2017-04-23 00:00:00
4	[null]	Jessika	Nussen	[null]	jnusse...	F	159.165.138...	615-824...	224 Village Circle	Nash...	TN	37215	36.0986	-86.8219	2017-09-03 00:00:00	
5	[null]	Lonnie	Rembaud	[null]	lremba...	F	18.131.58.65	786-499...	38 Lindbergh Way	Miami	FL	33124	25.5584	-80.4582	2014-03-06 00:00:00	
6	[null]	Cortie	Locksley	[null]	clocksl...	M	140.194.59.82	[null]	6537 Delladonna ...	Miami	FL	33158	25.6364	-80.3187	2013-03-31 00:00:00	
7	[null]	Wood	Kennham	[null]	wkenn...	M	191.190.135...	407-552...	001 Onsgard Park	Orla...	FL	32891	28.5663	-81.2608	2011-08-25 00:00:00	
7	[null]	Wood	Kennham	[null]	wkenn...	M	191.190.135...	407-552...	001 Onsgard Park	Orla...	FL	32891	28.5663	-81.2608	2011-08-25 00:00:00	
7	[null]	Wood	Kennham	[null]	wkenn...	M	191.190.135...	407-552...	001 Onsgard Park	Orla...	FL	32891	28.5663	-81.2608	2011-08-25 00:00:00	
11	Mrs	Urbano	Middlehurst	[null]	umiddl...	M	185.118.6.23	918-339...	5203 7th Trail	Tulsa	OK	74156	36.3024	-95.9605	2011-10-22 00:00:00	
12	Mr	Tyne	Duggan	[null]	tdugga...	F	13.29.231.228	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	2017-10-25 00:00:00

Rysunek 2.27. Kwerenda tworząca model wspomagający sprzedaż

W ten sposób uzyskałeś dane do zbudowania nowego modelu. Pomoże on zespołowi data science prognozować, do których klientów warto ponownie skierować przekaz marketingowy.

Kod źródłowy z tego fragmentu kodu znajdziesz na stronie <https://packt.live/2MQDusb>.

Rozdział 3. Agregacja i funkcje okna

Zadanie 3.01 — analizowanie danych sprzedażowych z użyciem funkcji agregujących

Rozwiązanie

1. Otwórz wybranego klienta SQL-a i nawiąż połączenie z bazą sql da.
2. Ustal łączną liczbę produktów sprzedanych przez firmę. Użyj do tego funkcji COUNT:

```
SELECT
  COUNT(*)
FROM
  sales;
```

Powinieneś uzyskać wynik **37 711** transakcji sprzedaży.

3. Dla każdego stanu ustal łączną kwotę transakcji w dolarach. Możesz użyć tu funkcji agregującej SUM:

```
SELECT
  c.state, SUM(sales_amount) as total_sales_amount
FROM
  sales s
INNER JOIN
  customers c
  ON c.customer_id=s.customer_id
GROUP BY
  1
ORDER BY
  1;
```

Otrzymasz dane wyjściowe widoczne na rysunku 3.30.

state text	sales_amount double precision
AK	1124268.776
AL	4820333.791
AR	1487923.589
AZ	4109364.447
CA	27942722.0350006
CO	5377388.30800006
CT	3038361.316
DC	7211615.1750001
DE	957264.298

Rysunek 3.30. Łączna sprzedaż w dolarach w poszczególnych stanach

4. Znajdź pięć najlepszych salonów pod względem liczby sprzedanych produktów. Użyj klauzuli GROUP BY oraz klauzuli LIMIT o wartości 5:

```
SELECT
    s.dealership_id,
    COUNT(*)
FROM
    sales s
WHERE
    channel='dealership'
GROUP BY
    1
ORDER BY
    2 DESC
LIMIT
    5
```

Powinieneś uzyskać dane wyjściowe widoczne na rysunku 3.31.

dealership_id double precision	count bigint
10	1781
7	1583
18	1465
11	1312
1	1297

Rysunek 3.31. Pięć pierwszych salonów według liczby sprzedanych sztuk produktów

5. Oblicz średnią wartość sprzedaży każdego produktu w każdym kanale. Dane są zapisane w tabeli sales. Sprawdź średnią wartość sprzedaży dla wszystkich kombinacji kanału sprzedaży i produktu (kolumny channel i product_id). Możesz do tego użyć instrukcji GROUPING SETS:

```
SELECT
    s.channel, s.product_id,
    AVG(sales_amount) as avg_sales_amount
FROM
    sales s
GROUP BY
    GROUPING SETS(
        (s.channel), (s.product_id),
        (s.channel, s.product_id)
    )
ORDER BY
    1, 2
```

Powinieneś uzyskać dane wyjściowe z rysunku 3.32.

channel text	product_id bigint	avg_sales_amount double precision
dealership	3	477.253737607644
dealership	4	109822.274881517
dealership	5	664.330132075472
dealership	6	62563.3763837638
dealership	7	573.744146637002
dealership	8	668.850500463391
dealership	9	33402.6845637584
dealership	10	81270.1121794872
dealership	11	91589.7435897436

Rysunek 3.32. Sprzedaż z klauzulą GROUPING SETS grupującą dane według pól channel i product_id

Na tym zrzucie pokazane są kanał i identyfikator produktu dla wszystkich produktów, a także wysokość sprzedaży dla każdej kombinacji wartości z tych kolumn.

Kod źródłowy z tego fragmentu jest dostępny na stronie <https://packt.live/2AXOXnc>.

Zadanie 3.02 — analizowanie sprzedaży z wykorzystaniem ramek okna i funkcji okna

Rozwiązanie

1. Otwórz wybranego klienta SQL-a i nawiąż połączenie z bazą sql da.
2. Oblicz łączną wartość sprzedaży z poszczególnych miesięcy 2018 roku.
Użyj funkcji SUM:

```
SELECT
    sales_transaction_date::DATE,
    SUM(sales_amount) as total_sales_amount
FROM
    sales
WHERE
    sales_transaction_date>='2018-01-01'
    AND sales_transaction_date<'2019-01-01'
GROUP BY
    1
ORDER BY
    1;
```

Dane wyjściowe tego kodu są pokazane na rysunku 3.33.

sales_transaction_date date	total_sales_amount double precision
2018-01-01	123689.951
2018-01-02	183859.79
2018-01-03	40029.854
2018-01-04	187119.878
2018-01-05	186459.904
2018-01-06	100479.888
2018-01-07	42989.864
2018-01-08	11089.815
2018-01-09	98119.878
2018-01-10	10449.823
2018-01-11	6449.891
2018-01-12	160659.838
2018-01-13	77789.869
2018-01-14	234429.898
2018-01-15	74429.847
2018-01-16	129189.854
2018-01-17	153839.873
2018-01-18	255529.864

Rysunek 3.33. Łączna wartość sprzedaży w poszczególnych miesiącach

3. Oblicz 30-dniową średnią kroczącą dziennej liczby transakcji. Użyj do tego ramki okna:

```
WITH daily_deals as (
  SELECT sales_transaction_date::DATE,
  COUNT(*) as total_deals
  FROM sales
  GROUP BY 1
),

moving_average_calculation_30 AS (
  SELECT sales_transaction_date, total_deals,
  AVG(total_deals) OVER (ORDER BY sales_transaction_date ROWS BETWEEN
  30 PRECEDING and CURRENT ROW) AS deals_moving_average,
  ROW_NUMBER() OVER (ORDER BY sales_transaction_date) as row_number
  FROM daily_deals
  ORDER BY 1)

SELECT sales_transaction_date,
CASE WHEN row_number>=30 THEN deals_moving_average ELSE NULL END
AS deals_moving_average_30
FROM moving_average_calculation_30
WHERE sales_transaction_date>='2018-01-01'
AND sales_transaction_date<'2019-01-01';
```

Dane wyjściowe są pokazane na rysunku 3.34.

sales_transaction_date date	deals_moving_average_30 numeric
2018-01-01	17.9354838709677419
2018-01-02	18.3548387096774194
2018-01-03	18.3548387096774194
2018-01-04	18.1290322580645161
2018-01-05	17.9354838709677419
2018-01-06	17.5806451612903226
2018-01-07	17.5161290322580645
2018-01-08	17.8064516129032258
2018-01-09	17.8709677419354839
2018-01-10	17.8387096774193548
2018-01-11	17.4193548387096774
2018-01-12	17.1935483870967742

Rysunek 3.34. 30-dniowa średnia krocząca liczby transakcji

4. Na podstawie łącznej wartości sprzedaży oblicz decyle, w jakich znajdują się poszczególne salony. Użyj funkcji okna:

```
WITH total_dealership_sales AS
(
  SELECT dealership_id,
         SUM(sales_amount) AS total_sales_amount
  FROM sales
  WHERE
    sales_transaction_date>='2018-01-01'
    AND sales_transaction_date<'2019-01-01'
    AND channel='dealership'
  GROUP BY 1
)
SELECT *,
       NTILE(10) OVER (ORDER BY total_sales_amount)
FROM
  total_dealership_sales;
```

Na rysunku 3.35 pokazane są dane wyjściowe tego kodu.

dealership_id double precision	total_sales_amount double precision	ntile integer
13	538079.414	1
9	618263.995	1
8	671619.251	2
4	905158.609	2
17	907058.842	3
20	949849.053	3
12	1086033.376	4
15	1197118.234	4
6	1316253.465	5
14	1551108.481	5
3	1622872.801	6
16	1981062.341	6

Rysunek 3.35. Decyle uzyskane na podstawie wartości sprzedaży w salonach

W ten sposób na podstawie łącznej wartości sprzedaży uzyskałeś listę decyli dla wszystkich salonów.

Kod źródłowy z tego fragmentu jest dostępny na stronie <https://packt.live/2UAI0ce>.

Rozdział 4.

Importowanie i eksportowanie danych

Zadanie 4.01 — używanie zewnętrznego zbioru danych do wykrywania trendów sprzedażowych

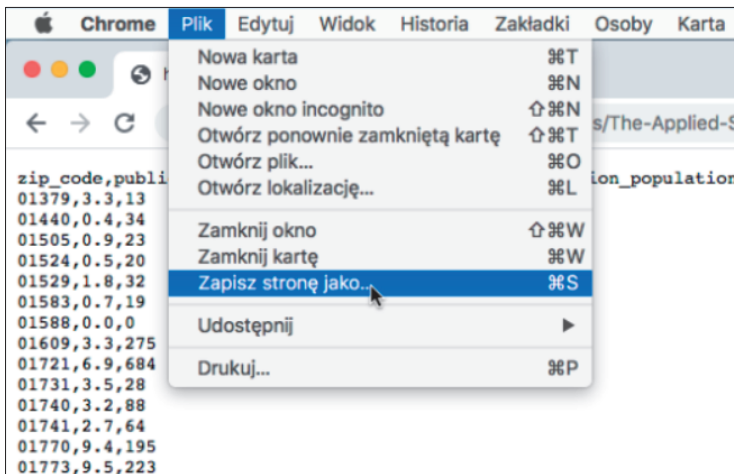
Rozwiązanie

1. Zanim zaczniesz dalsze analizy, wczytaj zbiór danych do Pythona i wyeksportuj go do bazy danych. Najpierw pobierz zbiór danych z serwisu GitHub, używając następującego odsyłacza: https://github.com/PacktWorkshops/The-Applied-SQL-Workshop/blob/master/Datasets/public_transportation_statistics_by_zip_code.csv.

Użytkownicy systemu Linux mogą skorzystać z polecenia `wget`:

```
wget https://github.com/PacktWorkshops/The-Applied-SQL-Workshop/blob/master/Datasets/public_transportation_statistics_by_zip_code.csv
```

2. Inną możliwość to wprowadzenie odsyłacza w przeglądarce. Po przejściu do docelowej strony wybierz opcję *Zapisz stronę jako* w menu przeglądarki (rysunek 4.23).



Rysunek 4.23. Zapisywanie pliku .csv z informacjami o transporcie publicznym

3. Następnie utwórz nowy notatnik Jupytera. W wierszu poleceń wpisz `jupyter notebook` (jeśli wcześniej nie uruchomiłeś serwera Jupytera). W oknie przeglądarki, które się pojawi, utwórz nowy notatnik dla Pythona 3. W pierwszej komórce wpisz standardowe instrukcje `import` i dane potrzebne do nawiązania połączenia. Zastąp człony `Twoje_x` odpowiednimi parametrami połączenia z bazą danych.

```

from sqlalchemy import create_engine
import pandas as pd
%matplotlib inline

cnxn_string = ("postgresql+psycopg2://{username}:{pswd}"
               "@{host}:{port}/{database}")
print(cnxn_string)

```

Oto dane wyjściowe tego kodu:

```
postgresql+psycopg2://{username}:{pswd}@{host}:{port}/{database}
```

Tworzenie obiektu engine:

```

engine = create_engine(cnxn_string.format(
    username="Twoja_nazwa_uzytkownika",
    pswd="Twoje_haslo",
    host="Twój_host",
    port=5432,
    database="sqlda"))

```

4. Wczytaj dane za pomocą polecenia podobnego do poniższego (zastąp ścieżkę z instrukcji ścieżką do pliku na Twoim komputerze):

```

data =
pd.read_csv("~/Downloads/public_transportation_statistics_by_zip_code.csv",
             dtype={'zip_code':str})

```

5. Sprawdź, czy dane są poprawne. W tym celu utwórz nową komórkę, wpisz `data`, a następnie wciśnij kombinację klawiszy *Shift + Enter*, aby wyświetlić zawartość zmiennej `data`. Możesz też wywołać funkcję `data.head()`, aby zobaczyć tylko kilka pierwszych wierszy danych.

```
data.head()
```

Na rysunku 4.24 widoczne są dane wyjściowe tego kodu.

	zip_code	public_transportation_pct	public_transportation_population
0	01379	3.3	13
1	01440	0.4	34
2	01505	0.9	23
3	01524	0.5	20
4	01529	1.8	32

Rysunek 4.24. Wczytywanie danych o transporcie publicznym do obiektów biblioteki pandas

6. Następnie przenieś dane do bazy, korzystając z funkcji `data.to_sql()`. Użycie funkcji `psql_insert_copy` pozwala znacznie szybciej wykonać tę operację, jednak nie jest to konieczne:

```

import csv
from io import StringIO

def psql_insert_copy(table, conn, keys, data_iter):
    # Tworzenie połączenia DBAPI, z którego można pobrać kursor

```

```

dbapi_conn = conn.connection
with dbapi_conn.cursor() as cur:
    s_buf = StringIO()
    writer = csv.writer(s_buf)
    writer.writerows(data_iter)
    s_buf.seek(0)

    columns = ', '.join("{} {}".format(k) for k in keys)
    if table.schema:
        table_name = '{}.{}'.format(table.schema, table.name)
    else:
        table_name = table.name
    sql = 'COPY {} ({} FROM STDIN WITH CSV'.format(
        table_name, columns)
    cur.copy_expert(sql=sql, file=s_buf)

data.to_sql('public_transportation_by_zip', engine, if_exists='replace',
method=psql_insert_copy)

```

Inna możliwość to użycie wolniejszej wersji kodu:

```
data.to_sql('public_transportation_by_zip', engine, if_exists='replace')
```

Na tym etapie masz już dane w bazie gotowe do obsługi kwerend.

- Wywołaj funkcję `max()`, aby wyświetlić wartość maksymalną w ramce danych:

```
data.max()
```

- Wywołaj funkcję `min()`, aby wyświetlić wartość minimalną w ramce danych:

```
data.min()
```

- Aby wyświetlić zakres wartości z kolumny `public_transportation_pct`, możesz pobrać go z bazy danych. Najpierw utwórz kwerendę:

```

engine.execute("""
SELECT
    MAX(public_transportation_pct) AS max_pct,
    MIN(public_transportation_pct) AS min_pct
FROM public_transportation_by_zip;
""").fetchall()

```

Rysunek 4.25 przedstawia wyniki tej kwerendy.

max_pct	min_pct
100	-666666666

Rysunek 4.25. Wyświetlanie wartości minimalnej i maksymalnej

Gdy przyjrzyś się wartościom maksymalnej i minimalnej, zauważysz coś dziwnego: wartość minimalna jest równa `-666666666`. Można przyjąć, że reprezentuje ona brakującą wartość, i usunąć ją ze zbioru danych.

10. Oblicz wartość sprzedaży, wykonując kwerendę na bazie danych. Zauważ, że trzeba odfiltrować błędne wartości procentowe mniejsze niż 0. Można to zrobić na kilka sposobów. Pokazane tu rozwiązanie to jedna zwężła kwerenda:

```
engine.execute("""
SELECT
    (public_transportation_pct > 10) AS is_high_public_transport,
    COUNT(s.customer_id) * 1.0 / COUNT(DISTINCT c.customer_id) AS
    sales_per_customer
FROM
    customers c
INNER JOIN public_transportation_by_zip t
    ON t.zip_code = c.postal_code
LEFT JOIN sales s
    ON s.customer_id = c.customer_id
WHERE
    public_transportation_pct >= 0
GROUP BY
    1
LIMIT
    10;
""").fetchall()
```

Oto omówienie tej kwerendy:

Możesz zidentyfikować klientów mieszkających w rejonach z transportem publicznym, sprawdzając dane o transporcie powiązane z kodem pocztowym. Jeśli `public_transportation_pct > 10`, klient mieszka w rejonie o wysokiej popularności transportu publicznego. Na podstawie tego wyrażenia można pogrupować dane, aby wykryć osoby mieszkające i niemieszkające w rejonie z dużą liczbą użytkowników transportu publicznego.

Liczbę sprzedaży na klienta można uzyskać, zliczając transakcje (na przykład za pomocą funkcji agregującej `COUNT(s.customer_id)`) i dzieląc wynik przez liczbę unikatowych klientów (otrzymaną na przykład za pomocą funkcji agregującej `COUNT(DISTINCT c.customer_id)`). Należy zachować część ułamkową, dlatego można pomnożyć wynik przez 1,0, aby zrzutować całe wyrażenie na typ `float` — `COUNT(s.customer_id) * 1.0 / COUNT(DISTINCT c.customer_id)`.

Musisz też złączyć dane o klientach z danymi o transporcie publicznym, a ostatecznie z danymi sprzedażowymi. Należy wyeliminować wszystkie kody pocztowe, w których wartość kolumny `public_transportation_pct` jest mniejsza od zera lub równa zero — pozwala to usunąć brakujące dane (reprezentowane przez wartość `-666666666`).

Ostatecznie otrzymasz dane wyjściowe widoczne na rysunku 4.26.

is_high_public_transport	sales_per_customer
f	0.71569054583929793988
t	0.83159379407616361072
(2 rows)	

Rysunek 4.26. Obliczanie żądanej liczby transakcji

Te dane pokazują, że klienci z rejonów o wysokiej popularności transportu publicznego dokonują 12% więcej zakupów produktów niż osoby mieszkające w rejonach, gdzie transport publiczny jest rzadziej używany.

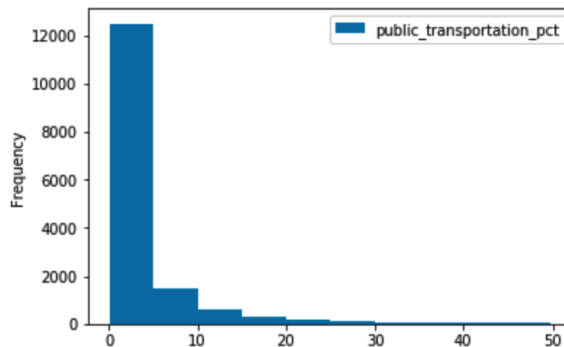
11. Wczytaj dane z bazy i dodaj klauzulę WHERE, aby wyeliminować wartości odstające. Następnie możesz wyświetlić wyniki kwerendy:

```
data = pd.read_sql_query("""
SELECT *
FROM public_transportation_by_zip
WHERE public_transportation_pct > 0
AND public_transportation_pct < 50""", engine)
data.plot.hist(y='public_transportation_pct')
```

Powinieneś otrzymać dane wyjściowe podobne do tych z rysunku 4.27.

```
In [13]: data = pd.read_sql_query("""
SELECT *
FROM public_transportation_by_zip
WHERE public_transportation_pct > 0
AND public_transportation_pct < 50""", engine)
data.plot.hist(y='public_transportation_pct')|

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1193ef160>
```



Rysunek 4.27. Notatnik Jupytera z analizą danych o transporcie publicznym

12. Uruchom instrukcje z kroku 4., wywołując polecenie %time, aby zmierzyć działanie kodu z instrukcją COPY i bez tej instrukcji:

```
data.to_sql('public_transportation_by_zip', engine, if_exists='replace',
method=psql_insert_copy)

%time data.to_sql('public_transportation_by_zip', engine, if_exists='replace')
```

Dane wyjściowe są pokazane na rysunku 4.28.

```
In [4]: import csv
        from io import StringIO

        def psql_insert_copy(table, conn, keys, data_iter):
            # gets a DBAPI connection that can provide a cursor
            dbapi_conn = conn.connection
            with dbapi_conn.cursor() as cur:
                s_buf = StringIO()
                writer = csv.writer(s_buf)
                writer.writerows(data_iter)
                s_buf.seek(0)

                columns = ', '.join("{} {}".format(k) for k in keys)
                if table.schema:
                    table_name = '{}.{}'.format(table.schema, table.name)
                else:
                    table_name = table.name

                sql = 'COPY {} ({} FROM STDIN WITH CSV'.format(
                    table_name, columns)
                cur.copy_expert(sql=s_buf, file=s_buf)

        %time data.to_sql('public_transportation_by_zip', engine, method=psql_insert_copy, if_exists='replace')
        CPU times: user 102 ms, sys: 21.1 ms, total: 123 ms
        Wall time: 1.2 s
```

Z instrukcją COPY – około 1 sekundy

```
In [5]: %time data.to_sql('public_transportation_by_zip', engine, if_exists='replace')
        CPU times: user 4.58 s, sys: 4.16 s, total: 8.75 s
        Wall time: 9min 15s
```

Bez instrukcji COPY – około 9 minut

Rysunek 4.28. Wstawianie rekordów z użyciem polecenia COPY (znacznie szybsze rozwiązanie) i bez niego

13. Pogrupuj klientów na podstawie popularności transportu publicznego dla kodu pocztowego (z zaokrągleniem do najbliższych 10%). Następnie sprawdź średnią liczbę transakcji na klienta. Wyeksportuj dane do Excela i utwórz wykres punktowy, aby lepiej zrozumieć zależność między popularnością transportu publicznego a poziomem sprzedaży. Na potrzeby tych analiz możesz zmodyfikować kwerendę z kroku 6.:

```
data = pd.read_sql_query("""
SELECT
10 * ROUND(public_transportation_pct/10)
    AS public_transport,
COUNT(s.customer_id) * 1.0 / COUNT(DISTINCT c.customer_id)
    AS sales_per_customer
FROM customers c
INNER JOIN public_transportation_by_zip t
    ON t.zip_code = c.postal_code
LEFT JOIN sales s ON s.customer_id = c.customer_id
WHERE public_transportation_pct >= 0
GROUP BY 1
""", engine)
data.to_csv('sales_vs_public_transport_pct.csv')
```

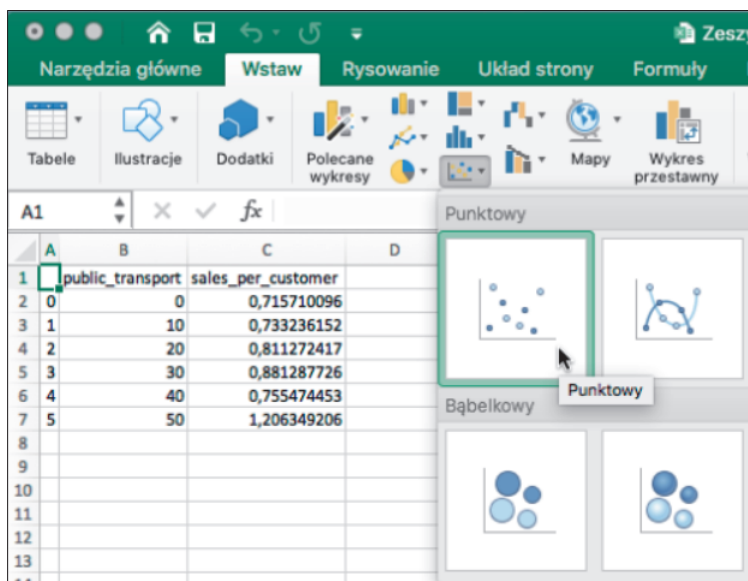
Najpierw należy zapisać wyniki tej kwerendy w zmiennej Pythona, co pozwoli później łatwo zapisać je w pliku CSV.

A teraz bardziej skomplikowana część zadania: chcesz zagregować statystyki dotyczące transportu publicznego. Możesz to zrobić, zaokrąglając wartości procentowe do najbliższych 10%. Dla 22% otrzymasz 20%, dla 39% otrzymasz

40% itd. Ten efekt możesz uzyskać, dzieląc wartość procentową (reprezentowaną jako liczba z przedziału od 0,0 do 100,0) przez 10, zaokrąglając wynik i mnożąc uzyskaną wartość przez 10: $10 * \text{ROUND}(\text{public_transportation_pct}/10)$.

Logika pozostałego kodu tej kwerendy jest opisana w kroku 6.

14. Następnie otwórz plik *sales_vs_public_transport_pct.csv* w Excelu (rysunek 4.29).



Rysunek 4.29. Arkusz Excela zawierający dane z wykonanej kwerendy

Po utworzeniu wykresu punktowego otrzymasz wyniki z rysunku 4.30. Widać w nich wyraźną pozytywną korelację między popularnością transportu publicznego a sprzedażą na danym obszarze.



Rysunek 4.30. Sprzedaż na klienta a procent użytkowników transportu publicznego

Na podstawie omówionych analiz można stwierdzić, że występuje pozytywna korelacja między „obszarami, gdzie popularny jest transport publiczny” a „zapotrzebowaniem na pojazdy elektryczne”. Intuicyjnie jest to sensowne, ponieważ pojazdy elektryczne mogą stanowić alternatywę transportu publicznego do poruszania się po miastach. Na podstawie tych analiz można zalecić zarządowi firmy ZoomZoom rozwijanie działalności w rejonach z wysokim odsetkiem użytkowników transportu publicznego i w rejonach miejskich.

Kod źródłowy z tego fragmentu książki jest dostępny na stronie <https://packt.live/3hniqYk>.

Rozdział 5. Analizy z wykorzystaniem złożonych typów danych

Zadanie 5.01 — wyszukiwanie i analiza transakcji sprzedaży

Rozwiązanie

1. Najpierw utwórz widok zmaterializowany bazujący na tabeli `customer_sales`. Na wypadek, gdyby istniał już widok o tej nazwie, przed poleceniem `CREATE` wywołaj instrukcję `DROP IF EXISTS`:

```
DROP MATERIALIZED VIEW IF EXISTS customer_search;

CREATE MATERIALIZED VIEW customer_search AS (
  SELECT
    customer_json -> 'customer_id' AS customer_id,
    customer_json,
    to_tsvector('english', customer_json) AS search_vector
  FROM customer_sales
);
```

W ten sposób uzyskasz tabelę o pokazanym dalej formacie (dane wyjściowe zostały skrócone dla większej czytelności):

```
SELECT * FROM customer_search LIMIT 1;
```

Na rysunku 5.27 pokazane są dane wyjściowe tego kodu.

```
customer_id | 1
customer_json | {"email": "ariveles@stumbleupon.com", "phone": null, "sales": [{"product_id": 7, "product_name": "Bat", "sales_amount": 479.992, "sales_transaction_date": "2017-07-19T08:38:41"}], "last_name": "Riveles", "date_added": "2017-04-23T00:00:00", "first_name": "Arlena", "customer_id": 1}
search_vector | '-04':15 '-07':6 '-19':7 '-23':16 '00':18,19 '2017':5,14 '38':9 '41':10 'ariveles@stumbleupon.com':1 'arlena':21 'bat':3 'rivel':12 't00':17 't08':8
```

Time: 1.678 ms

Rysunek 5.27. Przykładowy rekord z tabeli `customer_search`

2. Teraz możesz zgodnie z prośbą sprzedawcy poszukać w rekordach klienta o imieniu Danny, który kupił skuter Bat. Wymaga to tylko prostej kwerendy ze słowami kluczowymi Danny Bat:

```
SELECT
  customer_json
FROM
  customer_search
WHERE
  search_vector @@ plainto_tsquery('english', 'Danny Bat');
```

Otrzymasz osiem pasujących wierszy (rysunek 5.28).

```
{
  "email": "darundale87e@nytimes.com",
  "phone": null,
  "sales": [
    {
      "product_id": 8,
      "product_name": "Bat Limited Edition",
      "sales_amount": 699.99,
      "sales_transaction_date": "2017-05-16T08:41:03"
    }
  ],
  "last_name": "Arundale",
  "date_added": "2016-10-15T00:00:00",
  "first_name": "Danni",
  "customer_id": 10635
},
{
  "email": "dsinkins8vv@theatlantic.com",
  "phone": null,
  "sales": [
    {
      "product_id": 7,
      "product_name": "Bat",
      "sales_amount": 599.99,
      "sales_transaction_date": "2018-01-10T14:25:09"
    }
  ],
  "last_name": "Sinkins",
  "date_added": "2018-01-20T00:00:00",
  "first_name": "Danny",
  "customer_id": 11516
},
{
  "email": "dfalkusmrn@mysql.com",
  "phone": "360-138-1212",
  "sales": [
    {
      "product_id": 7,
      "product_name": "Bat",
      "sales_amount": 599.99,
      "sales_transaction_date": "2018-08-07T01:05:05"
    },
    {
      "product_id": 3,
      "product_name": "Lemon",
      "sales_amount": 399.992,
      "sales_transaction_date": "2018-07-05T09:51:51"
    }
  ],
  "last_name": "Falkus",
  "date_added": "2018-06-16T00:00:00",
  "first_name": "Dan",
  "customer_id": 35848
},
{
  "email": "dtyddwax@weebly.com",
  "phone": "626-781-3263",
  "sales": [
    {
      "product_id": 7,
      "product_name": "Bat",
      "sales_amount": 479.992,
      "sales_transaction_date": "2016-12-15T07:12:57"
    }
  ],
  "last_name": "Tydd",
  "date_added": "2016-12-08T00:00:00",
  "first_name": "Dannie",
  "customer_id": 41866
},
{
  "email": "dberthelmoxt5@jigsy.com",
  "phone": "559-535-5099",
  "sales": [
    {
      "product_id": 8,
      "product_name": "Bat Limited Edition",
      "sales_amount": 699.99,
      "sales_transaction_date": "2019-01-30T12:58:20"
    }
  ],
  "last_name": "Berthelmoxt",
  "date_added": "2018-01-09T00:00:00",
  "first_name": "Danni",
  "customer_id": 43818
},
{
  "email": "ddanev1b5@geocities.com",
  "phone": "415-491-7645",
  "sales": [
    {
      "product_id": 7,
      "product_name": "Bat",
      "sales_amount": 479.992,
      "sales_transaction_date": "2017-07-12T01:07:30"
    },
    {
      "product_id": 3,
      "product_name": "Lemon",
      "sales_amount": 499.99,
      "sales_transaction_date": "2016-12-09T08:02:24"
    },
    {
      "product_id": 3,
      "product_name": "Lemon",
      "sales_amount": 499.99,
      "sales_transaction_date": "2015-08-09T15:56:15"
    }
  ],
  "last_name": "Danev",
  "date_added": "2015-08-21T00:00:00",
  "first_name": "Danni",
  "customer_id": 1698
},
{
  "email": "dlamondpy@soundcloud.com",
  "phone": "585-779-9709",
  "sales": [
    {
      "product_id": 7,
      "product_name": "Bat",
      "sales_amount": 599.99,
      "sales_transaction_date": "2017-01-01T22:30:02"
    },
    {
      "product_id": 3,
      "product_name": "Lemon",
      "sales_amount": 499.99,
      "sales_transaction_date": "2016-12-19T03:55:45"
    }
  ],
  "last_name": "Lamond",
  "date_added": "2016-12-17T00:00:00",
  "first_name": "Danny",
  "customer_id": 33625
},
{
  "email": "dmagister113@canalblog.com",
  "phone": "860-336-0719",
  "sales": [
    {
      "product_id": 8,
      "product_name": "Bat Limited Edition",
      "sales_amount": 699.99,
      "sales_transaction_date": "2017-03-14T02:25:39"
    }
  ],
  "last_name": "Magister",
  "date_added": "2017-02-27T00:00:00",
  "first_name": "Danni",
  "customer_id": 48088
}
(8 rows)
```

Rysunek 5.28. Wynikowe pasujące wiersze z kwerendy ze słowami Danny Bat

3. W tym złożonym zadaniu celem jest znalezienie klientów, którzy zakupili określony skuter i określony samochód. To oznacza, że trzeba wykonać kwerendę dla każdej kombinacji produktów typu scooter i automobile. Aby pobrać każdą taką kombinację, wystarczy wykonać proste złączenie krzyżowe:

```
SELECT DISTINCT
  p1.model,
  p2.model
FROM
  products p1
CROSS JOIN products p2
WHERE p1.product_type = 'scooter'
AND p2.product_type = 'automobile'
AND p1.model NOT ILIKE '%Limited Edition%';
```

Ten kod zwraca dane wyjściowe z rysunku 5.29.

model		model
Bat		Model Chi
Bat		Model Epsilon
Bat		Model Gamma
Bat		Model Sigma
Blade		Model Chi
Blade		Model Epsilon
Blade		Model Gamma
Blade		Model Sigma
Lemon		Model Chi
Lemon		Model Epsilon
Lemon		Model Gamma
Lemon		Model Sigma
Lemon Zester		Model Chi
Lemon Zester		Model Epsilon
Lemon Zester		Model Gamma
Lemon Zester		Model Sigma

(16 rows)

Rysunek 5.29. Wszystkie kombinacje skutera i samochodu

4. Teraz wykorzystaj te dane wyjściowe w kwerendzie:

```
SELECT DISTINCT
  plainto_tsquery('english', p1.model) &&
  plainto_tsquery('english', p2.model)
FROM
  products p1
LEFT JOIN
  products p2 ON TRUE
WHERE p1.product_type = 'scooter'
AND p2.product_type = 'automobile'
AND p1.model NOT ILIKE '%Limited Edition%';
```

Otrzymasz wynik widoczny na rysunku 5.30.

```
'bat' & 'model' & 'chi'
'bat' & 'model' & 'sigma'
'blade' & 'model' & 'chi'
'lemon' & 'model' & 'chi'
'bat' & 'model' & 'gamma'
'blade' & 'model' & 'sigma'
'lemon' & 'model' & 'sigma'
'bat' & 'model' & 'epsilon'
'blade' & 'model' & 'gamma'
'lemon' & 'model' & 'gamma'
'blade' & 'model' & 'epsilon'
'lemon' & 'model' & 'epsilon'
'lemon' & 'zester' & 'model' & 'chi'
'lemon' & 'zester' & 'model' & 'sigma'
'lemon' & 'zester' & 'model' & 'gamma'
'lemon' & 'zester' & 'model' & 'epsilon'
```

(16 rows)

Rysunek 5.30. Kwerenda zwraca każdą kombinację skutera i samochodu

5. Uruchom kwerendę dla każdego z uzyskanych obiektów tsquery i zlicz wystąpienia każdego takiego obiektu:

```
SELECT
  sub.query,
  (
    SELECT COUNT(1)
    FROM customer_search
    WHERE customer_search.search_vector @@ sub.query)
FROM (
  SELECT DISTINCT
    plainto_tsquery('english', p1.model) &&
    plainto_tsquery('english', p2.model) AS query
  FROM products p1
  LEFT JOIN products p2 ON TRUE
  WHERE p1.product_type = 'scooter'
  AND p2.product_type = 'automobile'
  AND p1.model NOT ILIKE '%Limited Edition%'
) sub
ORDER BY 2 DESC;
```

Dane wyjściowe tej kwerendy są pokazane na rysunku 5.31.

query	count
'lemon' & 'model' & 'sigma'	340
'lemon' & 'model' & 'chi'	331
'bat' & 'model' & 'epsilon'	241
'bat' & 'model' & 'sigma'	226
'bat' & 'model' & 'chi'	221
'lemon' & 'model' & 'epsilon'	217
'bat' & 'model' & 'gamma'	153
'lemon' & 'model' & 'gamma'	133
'lemon' & 'zester' & 'model' & 'chi'	28
'lemon' & 'zester' & 'model' & 'epsilon'	22
'blade' & 'model' & 'chi'	21
'lemon' & 'zester' & 'model' & 'sigma'	17
'blade' & 'model' & 'sigma'	12
'lemon' & 'zester' & 'model' & 'gamma'	11
'blade' & 'model' & 'epsilon'	4
'blade' & 'model' & 'gamma'	4

(16 rows)

Rysunek 5.31. Liczba klientów dla każdej kombinacji skutera i samochodu

Choć wpływ na to może mieć wiele czynników, w danych widać, że najczęściej kupowaną kombinacją jest skuter lemon i samochód model sigma. Na drugiej pozycji występuje kombinacja skuter lemon i samochód model chi. Również skuter Bat jest dość często kupowany z tymi modelami, a także z modelem Epsilon. Pozostałe kombinacje są kupowane znacznie rzadziej. Wygląda na to, że klienci rzadko kupują skutery lemon zester i blade oraz samochód model gamma.

Kod źródłowy z tego fragmentu książki znajdziesz na stronie <https://packt.live/30HelCS>.

Rozdział 6. Wydajny SQL

Zadanie 6.01 — plany wykonywania kwerendy

Pamiętaj, że miary wydajności podawane w wyjściowym planie wykonywania kwerendy zależą od konfiguracji systemu.

Rozwiązanie

1. Otwórz PostgreSQL i nawiąż połączenie z bazą sql da:

```
C:\> psql sql da
```

2. Użyj polecenia EXPLAIN, aby zwrócić plan kwerendy pobierającej wszystkie dostępne rekordy z tabeli customers:

```
sql da=# EXPLAIN SELECT * FROM customers;
```

Planer na podstawie tej kwerendy zwróci dane wyjściowe pokazane na rysunku 6.63.

```

              QUERY PLAN
-----
Seq Scan on customers (cost=0.00..1536.00 rows=50000 width=140)
(1 row)

```

Rysunek 6.63. Plan kwerendy pobierającej wszystkie rekordy z tabeli customers

Koszt operacji przygotowawczych wynosi 0, łączny koszt wykonywania kwerendy to 1536, liczba wierszy jest równa 50000, a długość każdego wiersza to 140. Koszt jest podawany w jednostkach kosztu, liczba wierszy jest wyrażona w wierszach, a długość — w bajtach.

3. Zrób to samo dla kwerendy z kroku 2. zadania, ale tym razem ogranicz liczbę zwracanych rekordów do 15:

```
sql da=# EXPLAIN SELECT * FROM customers LIMIT 15;
```

Planer na podstawie tej kwerendy zwróci dane wyjściowe pokazane na rysunku 6.64.

```

              QUERY PLAN
-----
Limit (cost=0.00..0.46 rows=15 width=140)
-> Seq Scan on customers (cost=0.00..1536.00 rows=50000 width=140)
(2 rows)

```

Rysunek 6.64. Plan kwerendy pobierającej 15 rekordów z tabeli customers

Ta kwerenda jest wykonywana w dwóch etapach. Etap związany z ograniczeniem liczby rekordów kosztuje w planie 0,46 jednostki.

4. Wygeneruj plan wykonywania kwerendy, która pobiera wszystkie wiersze z danymi klientów mieszkających w rejonie o szerokości geograficznej większej niż 30 i mniejszej niż 40 stopni:

```
sql>da=# EXPLAIN SELECT *
        FROM customers
        WHERE latitude > 30 and latitude < 40;
```

Planer na podstawie tej kwerendy zwróci dane wyjściowe pokazane na rysunku 6.65.

```

                                QUERY PLAN
-----
Seq Scan on customers  (cost=0.00..1786.00 rows=26439 width=140)
  Filter: ((latitude > '30'::double precision) AND (latitude < '40'::double precision))
(2 rows)
```

Rysunek 6.65. Plan kwerendy pobierającej dane klientów mieszkających w rejonie o szerokości geograficznej większej niż 30 i mniejszej niż 40 stopni

Łączny koszt planu wynosi 1786 jednostek, a kwerenda zwraca 26439 wierszy.

Kod źródłowy z tego fragmentu książki jest dostępny na stronie <https://packt.live/3hxx5n3>.

Zadanie 6.02 — skanowanie indeksu

Rozwiązanie

1. Użyj poleceń EXPLAIN i ANALYZE do oceny planu wykonywania kwerendy, która wyszukuje wszystkie rekordy z adresem IP 18.131.58.65:

```
EXPLAIN ANALYZE SELECT *
        FROM customers
        WHERE ip_address = '18.131.58.65';
```

Zobaczysz dane wyjściowe widoczne na rysunku 6.66.

```

                                QUERY PLAN
-----
Seq Scan on customers  (cost=0.00..1661.00 rows=1 width=140) (actual time=0.019..15.592 rows=1 loops=1)
  Filter: (ip_address = '18.131.58.65'::text)
  Rows Removed by Filter: 49999
Planning Time: 0.191 ms
Execution Time: 15.625 ms
(5 rows)
```

Rysunek 6.66. Skanowanie sekwencyjne z filtrowaniem według kolumny ip_address

Utworzenie planu zajmuje 0.191 ms, a wykonanie kwerendy — 15.625 ms.

2. Utwórz ogólny indeks bazujący na kolumnie ip_address:

```
CREATE INDEX ON customers(ip_address);
```

3. Ponownie wykonaj kwerendę z kroku 1. i zwróć uwagę na czas wykonania:

```
EXPLAIN ANALYZE SELECT *
FROM customers
WHERE ip_address = '18.131.58.65';
```

Dane wyjściowe tego kodu są pokazane na rysunku 6.67.

```

QUERY PLAN
-----
Index Scan using customers_ip_address_idx on customers  (cost=0.29..8.31 rows=1 width=140) (actual time=0.072..0.075 rows=1 loops=1)
  Index Cond: (ip_address = '18.131.58.65'::text)
Planning Time: 0.467 ms
Execution Time: 0.123 ms
(4 rows)
```

Rysunek 6.67. Skanowanie indeksu z filtrowaniem według kolumny ip_address

Utworzenie planu zajmuje 0.467 ms, a wykonanie kwerendy — 0.123 ms.

4. Utwórz bardziej szczegółowy indeks bazujący na kolumnie ip_address. Tym razem użyj warunku, zgodnie z którym adres IP ma być równy 18.131.58.65:

```
CREATE INDEX ix_ip_where ON customers(ip_address)
WHERE ip_address = '18.131.58.65';
```

5. Jeszcze raz wykonaj kwerendę z kroku 1. i zwróć uwagę na czas wykonywania:

```
EXPLAIN ANALYZE SELECT *
FROM customers
WHERE ip_address = '18.131.58.65';
```

Dane wyjściowe tego kodu są pokazane na rysunku 6.68.

```

QUERY PLAN
-----
Index Scan using ix_ip_where on customers  (cost=0.12..8.14 rows=1 width=140) (actual time=0.021..0.023 rows=1 loops=1)
Planning Time: 0.458 ms
Execution Time: 0.056 ms
(3 rows)
```

Rysunek 6.68. Plan wykonywania kwerendy z czasem wykonania skróconym dzięki bardziej szczegółowemu indeksowi

Utworzenie planu zajmuje 0.458 ms, a wykonanie kwerendy — 0.056 ms.

Widać tu, że oba indeksy wymagają zbliżonego czasu planowania wykonywania kwerendy, ale indeks dla określonego adresu IP pozwala na znacznie szybsze wykonanie kwerendy (i nieco szybsze przygotowanie planu).

6. Użyj poleceń EXPLAIN i ANALYZE do zbadania planu wykonywania kwerendy, która wyszukuje wszystkie rekordy ze skrótem Jr:

```
EXPLAIN ANALYZE SELECT *
FROM customers
WHERE suffix = 'Jr';
```

Wyświetlone zostaną dane wyjściowe z rysunku 6.69.

QUERY PLAN

```
Seq Scan on customers (cost=0.00..1661.00 rows=107 width=140) (actual time=0.023..14.191 rows=102 loops=1)
  Filter: (suffix = 'Jr'::text)
  Rows Removed by Filter: 49898
  Planning Time: 0.153 ms
  Execution Time: 14.238 ms
(5 rows)
```

Rysunek 6.69. Plan wykonywania kwerendy ze skanowaniem sekwencyjnym i filtrowaniem na podstawie skrótu

Utworzenie planu zajmuje 0.153 ms, a wykonanie kwerendy — 14.238 ms.

7. Utwórz ogólny indeks bazujący na kolumnie suffix:

```
CREATE INDEX ix_jr ON customers(suffix);
```

8. Ponownie uruchom kwerendę z kroku 6. i zwróć uwagę na czas wykonania:

```
EXPLAIN ANALYZE SELECT *
  FROM customers
 WHERE suffix = 'Jr';
```

Wyświetlone zostaną dane wyjściowe z rysunku 6.70.

QUERY PLAN

```
Bitmap Heap Scan on customers (cost=5.12..318.44 rows=107 width=140) (actual time=0.146..0.440 rows=102 loops=1)
  Recheck Cond: (suffix = 'Jr'::text)
  Heap Blocks: exact=100
  -> Bitmap Index Scan on ix_jr (cost=0.00..5.09 rows=107 width=0) (actual time=0.092..0.092 rows=102 loops=1)
    Index Cond: (suffix = 'Jr'::text)
  Planning Time: 0.411 ms
  Execution Time: 0.511 ms
(7 rows)
```

Rysunek 6.70. Plan wykonywania kwerendy po utworzeniu indeksu dla kolumny suffix

Także tu czas planowania trochę się wydłużył, ale zostało to zrównoważone z nawiązką dzięki spadkowi czasu wykonania, który zmalał z 14.238 ms do 0.511 ms.

Kod źródłowy z tego fragmentu jest dostępny pod adresem <https://packt.live/3fkj72G>.

Zadanie 6.03 — stosowanie indeksów z haszowaniem

Rozwiązanie

1. Użyj poleceń EXPLAIN i ANALYZE, aby ustalić czas i koszt planowania oraz czas i koszt wykonania kwerendy pobierającej wszystkie wiersze, w których temat e-maila to Shocking Holiday Savings On Electric Scooters:

```
EXPLAIN ANALYZE SELECT *
  FROM emails
 WHERE
  email_subject='Shocking Holiday Savings On Electric Scooters';
```

Wyświetlone zostaną dane wyjściowe z rysunku 6.71.

QUERY PLAN

```
Seq Scan on emails (cost=0.00..10651.98 rows=19863 width=79) (actual time=7.843..117.840 rows=19873 loops=1)
  Filter: (email_subject = 'Shocking Holiday Savings On Electric Scooters'::text)
  Rows Removed by Filter: 398285
Planning Time: 0.117 ms
Execution Time: 119.801 ms
(5 rows)
```

Rysunek 6.71. Wydajność przy skanowaniu sekwencyjnym tabeli emails

Utworzenie planu zajmuje 0.117 ms, a wykonanie kwerendy — 119.801 ms. Koszt operacji przygotowawczych jest zerowy, ale koszt wykonania wynosi prawie 10 652 jednostki.

2. Użyj poleceń EXPLAIN i ANALYZE, aby ustalić czas i koszt planowania oraz czas i koszt wykonania kwerendy pobierającej wszystkie wiersze, w których temat e-maila to Black Friday. Green Cars.:

```
EXPLAIN ANALYZE SELECT *
FROM emails
WHERE email_subject='Black Friday. Green Cars.';
```

Wyświetlone zostaną dane wyjściowe z rysunku 6.72.

QUERY PLAN

```
Seq Scan on emails (cost=0.00..10651.98 rows=40645 width=79) (actual time=65.643..124.249 rows=41399 loops=1)
  Filter: (email_subject = 'Black Friday. Green Cars.'::text)
  Rows Removed by Filter: 376759
Planning Time: 0.097 ms
Execution Time: 127.736 ms
(5 rows)
```

Rysunek 6.72. Wydajność przy skanowaniu sekwencyjnym dla innego tematu e-maila

Utworzenie planu zajmuje 0.097 ms, a wykonanie kwerendy — 127.736 ms. Dłuższy czas wykonania można częściowo przypisać większej liczbie zwracanych wierszy. Także tu koszt operacji przygotowawczych wynosi zero; podobny jest też koszt wykonania — około 10 652 jednostek.

3. Utwórz indeks z haszowaniem dla pola email_subject:

```
CREATE INDEX ix_email_subject ON emails
USING HASH(email_subject);
```

4. Powtórz krok 1. z rozwiązania i porównaj dane wyjściowe:

```
EXPLAIN ANALYZE SELECT *
FROM emails
WHERE email_subject='Shocking Holiday Savings On Electric Scooters';
```

Wyświetlone zostaną dane wyjściowe z rysunku 6.73.

QUERY PLAN

```
Bitmap Heap Scan on emails (cost=641.94..6315.23 rows=19863 width=79) (actual time=2.096..15.061 rows=19873 loops=1)
  Recheck Cond: (email_subject = 'Shocking Holiday Savings On Electric Scooters'::text)
  Heap Blocks: exact=289
  -> Bitmap Index Scan on ix_email_subject (cost=0.00..636.97 rows=19863 width=0) (actual time=1.936..1.936 rows=19873 loops=1)
    Index Cond: (email_subject = 'Shocking Holiday Savings On Electric Scooters'::text)
Planning Time: 0.130 ms
Execution Time: 17.028 ms
(7 rows)
```

Rysunek 6.73. Dane wyjściowe planera dla kwerendy, w której używany jest indeks z haszowaniem

Ten plan wykonywania kwerendy pokazuje, że nowo utworzony indeks z haszowaniem jest używany i znacznie skraca czas (o ponad 100 ms), a także koszt wykonania. Widoczny jest niewielki wzrost czasu i kosztu planowania, który jest jednak z nawiązką rekompensowany przez skrócenie czasu wykonania.

5. Powtórz krok 2. rozwiązania i porównaj dane wyjściowe:

```
EXPLAIN ANALYZE SELECT *
  FROM emails
 WHERE email_subject='Black Friday. Green Cars.';
```

Dane wyjściowe tego kodu są pokazane na rysunku 6.74.

```

----- QUERY PLAN -----
Bitmap Heap Scan on emails (cost=1311.00..7244.06 rows=40645 width=79) (actual time=4.085..29.296 rows=41399 loops=1)
  Recheck Cond: (email_subject = 'Black Friday. Green Cars.'::text)
  Heap Blocks: exact=531
-> Bitmap Index Scan on ix_email_subject (cost=0.00..1300.84 rows=40645 width=0) (actual time=3.817..3.817 rows=41399 loops=1)
    Index Cond: (email_subject = 'Black Friday. Green Cars.'::text)
Planning Time: 0.403 ms
Execution Time: 33.216 ms
(7 rows)
```

Rysunek 6.74. Dane wyjściowe planera dla kwerendy z mniej wydajnym indeksem z haszowaniem

Także tu widoczny jest spadek kosztu i czasu wykonania. Jednak korzyści dla wyszukiwania tematu „Black Friday...” są mniejsze niż dla wyszukiwania tematu „Shocking Holiday Savings...”. Jeśli przyjrzyś się danym dokładniej, zobaczysz, że skanowanie indeksu zajmuje około dwóch razy więcej czasu, przy czym liczba zwracanych rekordów też jest jakieś dwa razy większa. Można na tej podstawie dojść do wniosku, że wzrost kosztów wynika ze wzrostu liczby rekordów zwracanych przez kwerendę.

6. Utwórz indeks z haszowaniem dla pola `customer_id`:

```
CREATE INDEX ix_customer_id ON emails
  USING HASH(customer_id);
```

7. Użyj poleceń `EXPLAIN` i `ANALYZE`, aby oszacować czas potrzebny do pobrania wszystkich wierszy z wartością pola `customer_id` większą niż 100. Jakie skanowanie zostanie przeprowadzone i dlaczego?

```
EXPLAIN ANALYZE SELECT *
  FROM emails
 WHERE customer_id > 100;
```

Wyświetlone zostaną dane wyjściowe z rysunku 6.75.

```

----- QUERY PLAN -----
Seq Scan on emails (cost=0.00..10651.98 rows=417309 width=79) (actual time=0.024..121.483 rows=417315 loops=1)
  Filter: (customer_id > 100)
  Rows Removed by Filter: 843
Planning Time: 0.199 ms
Execution Time: 152.656 ms
(5 rows)
```

Rysunek 6.75. Planer ignoruje indeks z haszowaniem z powodu jego ograniczeń

Ostateczny czas wykonania wynosi więc 152.656 ms, a czas planowania — 0.199 ms.

Kod źródłowy z tego fragmentu książki znajdziesz na stronie <https://packt.live/2YqkVVf>.

Zadanie 6.04 — stosowanie wydajnych złączeń

Rozwiązanie

1. Otwórz PostgreSQL i nawiąż połączenie z bazą sql`da`:

```
$ psql sqlda
```

2. Przygotuj listę klientów (z polami `customer_id`, `first_name` i `last_name`), którzy otrzymali e-mail. Uwzględnij temat e-maila i to, czy odbiorca otworzył i kliknął wiadomość. Wynikowa tabela powinna zawierać kolumny `customer_id`, `first_name`, `last_name`, `email_subject`, `opened` i `clicked`:

```
sqlda=# SELECT customers.customer_id, customers.first_name,
customers.last_name,
emails.opened, emails.clicked
FROM customers INNER JOIN emails ON customers.customer_id=emails.customer_id;
```

Na zrzucie z rysunku 6.76 pokazane są dane wyjściowe tego kodu.

customer_id	first_name	last_name	opened	clicked
18	Mareah	Edgell	f	f
30	Kath	Rivel	f	f
41	Rycca	Oakwell	t	f
52	Giusto	Backe	f	f
59	Laurene	Lobbe	f	f
78	West	Hampson	f	f
82	Claudie	Cancott	f	f
84	Nels	Beefon	f	f
103	Natalina	Dell 'Orto	f	f
119	Hugibert	Bullocke	f	f
132	Orrin	Evennett	f	f
134	Emmalyn	Hackney	f	f
135	Myrilla	Starcks	f	f
137	Cindee	Prandi	f	f

Rysunek 6.76. Złączenie tabel `customers` i `emails`

3. Zapisz wynikową tabelę pod nazwą `customers_emails`:

```
sqlda=# SELECT customers.customer_id, customers.first_name,
customers.last_name,
emails.opened, emails.clicked
INTO customer_emails FROM customers INNER JOIN emails ON
customers.customer_id=emails.customer_id;
```

4. Znajdź klientów, którzy otworzyli i kliknęli e-mail:

```
SELECT *
FROM customer_emails
WHERE clicked='t' and opened='t';
```

Na rysunku 6.77 pokazane są dane wyjściowe kodu.

customer_id	first_name	last_name	opened	clicked
554	Chet	Melchior	t	t
673	Hirsch	Kulver	t	t
1255	Randi	Benzing	t	t
1916	Gabrielle	Skeermer	t	t
2109	Augie	Rhymer	t	t
2308	Natale	Ruddiman	t	t
2909	Wilton	Silversmid	t	t
3367	Aidan	Hinzer	t	t
3718	Myrah	Capstack	t	t
4013	Dalton	Turrill	t	t
4303	Benson	Pruvost	t	t
4370	Krystle	Roiz	t	t
6405	Valaree	Wedmore	t	t

Rysunek 6.77. Klienci, którzy kliknęli i otworzyli e-mail

5. Znajdź klientów, którzy mają salon w swoim mieście. Klienci, w których mieście nie ma salonu, mają pustą wartość w kolumnie city:

```
sql># SELECT customers.customer_id, customers.first_name,
customers.last_name,
customers.city
FROM customers
LEFT JOIN dealerships on customers.city=dealerships.city;
```

Ten kod wyświetli dane wyjściowe z rysunku 6.78.

customer_id	first_name	last_name	city
1	Arlena	Riveles	
2	Ode	Stovin	Saint Louis
3	Braden	Jordan	Pensacola
4	Jessika	Nussen	Nashville
5	Lonnie	Rembaud	Miami
6	Cortie	Locksley	Miami
7	Wood	Kennham	Orlando
8	Rutger	Humblestone	New Haven
9	Melantha	Tibb	Shawnee Mission
10	Barbara-anne	Gowlett	El Paso
11	Urbano	Middlehurst	Tulsa

Rysunek 6.78. Złączenie lewostronne tabel customers i dealerships

6. Zapisz wyniki w tabeli customer_dealers:

```
sql># SELECT customers.customer_id, customers.first_name,
customers.last_name, customers.city INTO customer_dealers FROM customers
LEFT JOIN dealerships on customers.city=dealerships.city;
```

7. Wyświetl klientów, którzy nie mają salonu w swoim mieście (wskazówka: puste pole jest reprezentowane jako NULL):

```
sql># SELECT * from customer_dealers WHERE city is NULL;
```

Na rysunku 6.79 pokazane są dane wyjściowe tego kodu.

customer_id	first_name	last_name	city
1	Arlena	Riveles	
12	Tyne	Duggan	
21	Pryce	Geist	
24	Barbi	Lanegran	
30	Kath	Rivel	
38	Carter	Lagneaux	
44	Waldemar	Paroni	
49	Hannah	McGlew	
56	Riva	Cathesyed	
63	Gweneth	Maior	
70	Caty	Woolveridge	
72	Jodi	Fautly	

Rysunek 6.79. Klienci bez informacji o mieście

W danych wyjściowych widoczna jest ostateczna lista klientów z miast, w których firma nie ma salonu.

Kod źródłowy z tego fragmentu książki jest dostępny na stronie <https://packt.live/3ffcSZq>.

Zadanie 6.05 — definiowanie funkcji zwracającej maksymalną wartość sprzedaży

Rozwiązanie

1. Nawiąż połączenie z bazą sqllda. Poniższe polecenie zadziała tylko w wierszu poleceń SQL-a (nie zadziała w narzędziu pgadmin):

```
$ psql sqllda postgres
```

2. Utwórz funkcję `max_sale`, która nie przyjmuje żadnych argumentów, a zwraca wartość liczbową `big_sale`:

```
sqllda=# CREATE FUNCTION max_sale() RETURNS integer AS $big_sale$
```

3. Zadeklaruj zmienną `big_sale` i rozpocznij funkcję:

```
sqllda$# DECLARE big_sale numeric;
sqllda$# BEGIN
```

4. Przypisz maksymalną wartość sprzedaży do zmiennej `big_sale`:

```
sqllda$# SELECT MAX(sales_amount) INTO big_sale FROM sales;
```

5. Zwróć wartość zmiennej `big_sale`:

```
sqllda$# RETURN big_sale;
```

6. Zakończ funkcję instrukcją `LANGUAGE`:

```
sqllda$# END; $big_sale$
sqllda=# LANGUAGE PLPGSQL;
```

7. Wywołaj funkcję, aby stwierdzić, jaka jest najwyższa wartość sprzedaży w bazie:

```
sql># SELECT MAX(sales_amount) FROM sales;
```

Oto dane wyjściowe tego kodu:

```
Max
-----
115000
(1 row)
```

Dane wyjściowe są generowane w funkcji, która pobiera najwyższą wartość sprzedaży z bazy danych, czyli 115000.

Kod źródłowy z tego fragmentu książki znajdziesz na stronie <https://packt.live/2ztZshK>.

Zadanie 6.06

— tworzenie funkcji przyjmujących argumenty

Rozwiązanie

1. Utwórz definicję funkcji `avg_sales_window`, która zwraca wartość liczbową i przyjmuje dwie określające daty wartości typu `DATE` w formacie `RRRR-MM-DD`:

```
sql># CREATE FUNCTION avg_sales_window(from_date DATE, to_date DATE)
      RETURNS numeric AS $sales_avg$
```

2. Zadeklaruj zwracaną zmienną typu `numeric` i rozpocznij funkcję:

```
sql># DECLARE sales_avg numeric;
sql># BEGIN
```

3. Pobierz średnią wartość sprzedaży i przypisz ją do zwracanej zmiennej; uwzględnij transakcje zawarte w określonym przedziale czasu:

```
sql># SELECT AVG(sales_amount) FROM sales INTO sales_avg WHERE
      sales_transaction_date > from_date AND sales_transaction_date < to_date;
```

4. Zwróć zmienną, zakończ funkcję i dodaj instrukcję `LANGUAGE`:

```
sql># RETURN sales_avg;
sql># END; $sales_avg$
sql>-# LANGUAGE PLPGSQL;
```

5. Użyj funkcji do obliczenia średniej wartości sprzedaży z okresu od 2013-04-12 do 2014-04-12:

```
sql># SELECT avg_sales_window('2013-04-12', '2014-04-12');
```

Oto dane wyjściowe tego kodu:

```
avg_sales_window
-----
477.6862463110066
(1 row)
```

Końcowe dane wyjściowe to średnia sprzedaż z okresu między dwiema datami. Wynosi ona około 477,687.

Kod źródłowy z tego fragmentu książki jest dostępny na stronie <https://packt.live/2UCxY0A>.

Zadanie 6.07 — tworzenie wyzwalacza do śledzenia średniej liczby kupionych sztuk

Rozwiązanie

1. Nawiąż połączenie z bazą smalljoins:

```
$ psql smalljoins
```

2. Utwórz nową tabelę, avg_qty_log, która zawiera pole order_id typu integer i pole avg_qty typu numeric.

```
smalljoins=# CREATE TABLE avg_qty_log (order_id integer, avg_qty numeric);
```

3. Utwórz funkcję avg_qty, która nie przyjmuje żadnych argumentów, a zwraca wyzwalacz. Ta funkcja ma obliczać średnią liczbę sztuk z wszystkich zamówień (order_info.qty) i wstawiać tę wartość razem z najnowszym identyfikatorem order_id do tabeli avg_qty_log.

```
smalljoins=# CREATE FUNCTION avg_qty() RETURNS TRIGGER AS $_avg$
smalljoins$# DECLARE _avg numeric;
smalljoins$# BEGIN
smalljoins$# SELECT AVG(qty) INTO _avg FROM order_info;
smalljoins$# INSERT INTO avg_qty_log (order_id, avg_qty) VALUES (NEW.order_id, _avg);
smalljoins$# RETURN NEW;
smalljoins$# END; $_avg$
smalljoins=# LANGUAGE PLPGSQL;
```

4. Utwórz wyzwalacz avg_trigger, który wywołuje funkcję avg_qty po (opcja AFTER) wstawieniu każdego wiersza do tabeli order_info:

```
smalljoins=# CREATE TRIGGER avg_trigger
smalljoins=# AFTER INSERT ON order_info
smalljoins=# FOR EACH ROW
smalljoins=# EXECUTE PROCEDURE avg_qty();
```

5. Wstaw do tabeli order_info nowe wiersze z liczbą sztuk równą 6, 7 i 8:

```
smalljoins=# SELECT insert_order(3, 'GROG1', 6);
smalljoins=# SELECT insert_order(4, 'GROG1', 7);
smalljoins=# SELECT insert_order(1, 'GROG1', 8);
```

6. Przejrzyj rekordy z tabeli avg_qty_log, aby sprawdzić, czy średnia liczba sztuk po każdym zamówieniu rośnie:

```
smalljoins=# SELECT * FROM avg_qty_log;
```

Na rysunku 6.80 pokazane są dane wyjściowe tego kodu.

order_id	avg_qty
1625	4.7500000000000000
1626	5.0000000000000000
1627	5.3000000000000000
(3 rows)	

Rysunek 6.80. Zmiany średniej liczby sztuk w zamówieniu

Na podstawie tych zamówień i danych z tabeli widać, że średnia liczba sztuk w zamówieniu rośnie.

Kod źródłowy z tego fragmentu książki jest dostępny na stronie <https://packt.live/2MUJVdG>.

Zadanie 6.08 — kończenie długو działającej kwerendy

Rozwiązanie

1. Uruchom dwa odrębne interpretery SQL-a:

```
C:\> psql sqlda
```

2. W pierwszym terminalu wywołaj polecenie `pg_sleep` z parametrem 1000 sekund:

```
sqlda=# SELECT pg_sleep(1000);
```

3. W drugim terminalu ustal identyfikator procesu kwerendy z poleceniem `pg_sleep`:

```
sqlda=# SELECT pid, query FROM pg_stat_activity WHERE state = 'active';
```

Na rysunku 6.81 pokazane są dane wyjściowe tej kwerendy:

pid	query
14117	SELECT pid, query FROM pg_stat_activity WHERE state = 'active';
14131	SELECT pg_sleep(1000);
(2 rows)	

Rysunek 6.81. Sprawdzanie wartości pid procesu kwerendy z poleceniem `pg_sleep`

4. Użyj ustalonej wartości pid, aby wymusić zakończenie polecenia `pg_sleep` za pomocą instrukcji `pg_terminate_backend`:

```
sqlda=# SELECT pg_terminate_backend(14131);
```

Oto dane wyjściowe tego kodu:

```
pg_terminate_backend
-----
t
(1 row)
```

5. W pierwszym terminalu upewnij się, że polecenie `pg_sleep` zostało zakończone. Zwróć uwagę na komunikat zwrócony przez interpreter:

```
sql>=# SELECT pg_sleep(1000);
```

Zobaczysz dane wyjściowe widoczne na rysunku 6.82.

```
sql>=# SELECT pg_sleep(1000);
KATASTROFALNY: zakończono połączenie na skutek polecenia administratora
server closed the connection unexpectedly
        This probably means the server terminated abnormally
        before or while processing the request.
The connection to the server was lost. Attempting reset: Succeeded.
sql>=#
```

Rysunek 6.82. Zakończony proces z poleceniem `pg_sleep`

Na rzucie widać, że kwerenda z poleceniem `pg_sleep` została zakończona.

Kod źródłowy z tego fragmentu książki jest dostępny na stronie <https://packt.live/3hj8E9H>.

Rozdział 7. Metoda naukowa i rozwiązywanie problemów w praktyce

Zadanie 7.01 — ilościowa ocena spadku sprzedaży

Rozwiązanie

1. Wczytaj bazę danych `sql`:

```
$ psql sql
```

2. Za pomocą instrukcji `OVER` i `ORDER BY` oblicz dzienną skumulowaną sumę transakcji. Wstaw wyniki do nowej tabeli, `bat_sales_growth`:

```
sql>=# SELECT *, sum(count) OVER (ORDER BY sales_transaction_date)
INTO bat_sales_growth FROM bat_sales_daily;
```

Powinieneś uzyskać następujące dane wyjściowe:

```
SELECT 964
```

Pobierz wartość kolumny `sum` sprzed 7 dni (za pomocą funkcji `lag`), a następnie wstaw wszystkie kolumny tabeli `bat_sales_daily` i nową kolumnę `lag` do nowej tabeli, `bat_sales_daily_delay`. Kolumna `lag` informuje o poziomie sprzedaży tydzień przed danym rekordem:

```
sql>=# SELECT *, lag(sum, 7) OVER (ORDER BY sales_transaction_date)
INTO bat_sales_daily_delay FROM bat_sales_growth;
```


3. Sprawdź pierwszych 15 wierszy tabeli bat_sales_growth:

```
sql># SELECT * FROM bat_sales_daily_delay LIMIT 15;
```

Dane wyjściowe tego kodu są pokazane na rysunku 7.27.

sales_transaction_date	count	sum	lag
2016-10-10 00:00:00	9	9	
2016-10-11 00:00:00	6	15	
2016-10-12 00:00:00	10	25	
2016-10-13 00:00:00	10	35	
2016-10-14 00:00:00	5	40	
2016-10-15 00:00:00	10	50	
2016-10-16 00:00:00	14	64	
2016-10-17 00:00:00	9	73	9
2016-10-18 00:00:00	11	84	15
2016-10-19 00:00:00	12	96	25
2016-10-20 00:00:00	10	106	35
2016-10-21 00:00:00	6	112	40
2016-10-22 00:00:00	2	114	50
2016-10-23 00:00:00	5	119	64
2016-10-24 00:00:00	6	125	73

(15 rows)

Rysunek 7.27. Suma dziennej sprzedaży z przesunięciem czasowym

4. Oblicz procentowy przyrost liczby transakcji, porównując aktualną liczbę sprzedaży z poziomem sprzed tygodnia. Wstaw wynikowe wartości do nowej tabeli, bat_sales_delay_vol:

```
sql># SELECT *, (sum-lag)/lag AS volume INTO bat_sales_delay_vol
FROM bat_sales_daily_delay;
```

Procentowy wzrost sprzedaży można obliczyć za pomocą następującego wzoru:

$(\text{nowa_liczba_sztuk} - \text{poprzednia_liczba_sztuk}) / \text{poprzednia_liczba_sztuk}$

5. Porównaj pierwsze 22 wartości z tabeli bat_sales_delay_vol:

```
sql># SELECT * FROM bat_sales_delay_vol LIMIT 22;
```

Na rysunku 7.28 pokazane są względne zmiany w przyroście liczby transakcji dla pierwszych 22 dni.

W tabeli wyjściowej widoczne są cztery rodzaje informacji: dzienna liczba sprzedanych sztuk, skumulowana suma dziennej liczby sprzedanych sztuk, skumulowana suma sprzed tygodnia i względna zmiana poziomu sprzedaży.

Kod źródłowy z tego fragmentu książki jest dostępny na stronie <https://packt.live/2BXrV09>.

sales_transaction_date	count	sum	lag	volume
2016-10-10 00:00:00	9	9		
2016-10-11 00:00:00	6	15		
2016-10-12 00:00:00	10	25		
2016-10-13 00:00:00	10	35		
2016-10-14 00:00:00	5	40		
2016-10-15 00:00:00	10	50		
2016-10-16 00:00:00	14	64		
2016-10-17 00:00:00	9	73	9	7.1111111111111111
2016-10-18 00:00:00	11	84	15	4.6000000000000000
2016-10-19 00:00:00	12	96	25	2.8400000000000000
2016-10-20 00:00:00	10	106	35	2.0285714285714286
2016-10-21 00:00:00	6	112	40	1.8000000000000000
2016-10-22 00:00:00	2	114	50	1.2800000000000000
2016-10-23 00:00:00	5	119	64	0.8593750000000000
2016-10-24 00:00:00	6	125	73	0.7123287671232876
2016-10-25 00:00:00	9	134	84	0.5952380952380952
2016-10-26 00:00:00	2	136	96	0.4166666666666666
2016-10-27 00:00:00	4	140	106	0.3207547169811320
2016-10-28 00:00:00	7	147	112	0.3125000000000000
2016-10-29 00:00:00	5	152	114	0.3333333333333333
2016-10-30 00:00:00	5	157	119	0.3193277310924369
2016-10-31 00:00:00	3	160	125	0.2800000000000000

(22 rows)

Rysunki 7.28. Względne zmiany w przyroście liczby sprzedaży skutera z okresu trzech tygodni

Zadanie 7.02

— analiza hipotezy dotyczącej różnicy w cenie sprzedaży

Rozwiązanie

1. Wczytaj bazę danych sqllda:

```
$ psql sqllda
```

2. Pobierz kolumnę sales_transaction_date z transakcjami dotyczącymi modelu Lemon z 2013 roku. Wstaw tę kolumnę do tabeli lemon_sales:

```
sqllda=# SELECT sales_transaction_date INTO lemon_sales FROM sales
WHERE product_id=3;
```

3. Zlicz rekordy z transakcjami dotyczące modelu Lemon z 2013 roku:

```
sqllda=# SELECT count(sales_transaction_date) FROM lemon_sales;
```

Zobaczysz, że dostępnych rekordów jest 16558:

```
count
-----
16558
(1 row)
```

4. Użyj funkcji max, aby wyświetlić najnowszą wartość z kolumny sales_transaction_date:

```
sqllda=# SELECT max(sales_transaction_date) FROM lemon_sales;
```

Oto najnowsza wartość z kolumny sales_transaction_date:

```
max
-----
2018-12-17 19:12:10
(1 row)
```

5. Przekształć typ kolumny sales_transaction_date na date:

```
sql># ALTER TABLE lemon_sales ALTER COLUMN sales_transaction_date
TYPE DATE;
```

Typ danych jest zmieniany z DATE_TIME na DATE, aby usunąć z pola informacje o czasie. Ważne jest tu tylko akumulowanie wartości, ale na podstawie dat, a nie godzin. Dlatego łatwiej jest usunąć z pola informacje o czasie.

6. Określ liczbę transakcji z poszczególnych dni z tabeli lemon_sales.

Wstaw wynikowe dane do tabeli lemon_sales_count:

```
sql># SELECT *, COUNT(sales_transaction_date) INTO lemon_sales_count
FROM lemon_sales GROUP BY sales_transaction_date
ORDER BY sales_transaction_date;
```

7. Oblicz skumulowaną sumę liczby transakcji i wstaw wyniki do nowej tabeli, lemon_sales_sum:

```
sql># SELECT *, sum(count) OVER (ORDER BY sales_transaction_date)
INTO lemon_sales_sum FROM lemon_sales_count;
```

8. Za pomocą funkcji lag wyznacz wartości kolumny sum sprzed 7 dni i wstaw wyniki do tabeli lemon_sales_delay:

```
sql># SELECT *, lag(sum, 7) OVER (ORDER BY sales_transaction_date)
INTO lemon_sales_delay FROM lemon_sales_sum;
```

9. Oblicz tempo wzrostu liczby transakcji sprzedaży na podstawie danych z tabeli lemon_sales_delay. Zapisz wynikową tabelę pod nazwą lemon_sales_growth. Kolumnę z szybkością wzrostu nazwij volume:

```
sql># SELECT *, (sum-lag)/lag AS volume INTO lemon_sales_growth
FROM lemon_sales_delay;
```

10. Zbadaj pierwsze 22 rekordy z tabeli lemon_sales_growth, analizując dane z kolumny volume:

```
sql># SELECT * FROM lemon_sales_growth LIMIT 22;
```

Wzrost sprzedaży jest pokazany w tabeli z rysunku 7.29.

Podobnie jak w poprzednim zadaniu obliczyłeś sumę skumulowaną, sumę skumulowaną sprzed 7 dni i względny wzrost sprzedaży skutera Lemon. Widać tu, że początkowy względny wzrost sprzedaży był większy niż w przypadku innych skuterów (powyżej 800%), a na koniec badanego okresu też odnotowano wyższą wartość niż dla innych modeli (około 55%).

sales_transaction_date	count	sum	lag	volume
2013-05-01	6	6		
2013-05-02	8	14		
2013-05-03	4	18		
2013-05-04	9	27		
2013-05-05	9	36		
2013-05-06	6	42		
2013-05-07	8	50		
2013-05-08	6	56	6	8.333333333333333
2013-05-09	6	62	14	3.4285714285714286
2013-05-10	9	71	18	2.9444444444444444
2013-05-11	3	74	27	1.7407407407407407
2013-05-12	4	78	36	1.1666666666666667
2013-05-13	7	85	42	1.0238095238095238
2013-05-14	3	88	50	0.7600000000000000
2013-05-15	3	91	56	0.6250000000000000
2013-05-16	4	95	62	0.5322580645161290
2013-05-17	6	101	71	0.4225352112676056
2013-05-18	9	110	74	0.4864864864864864
2013-05-19	6	116	78	0.4871794871794871
2013-05-20	6	122	85	0.4352941176470588
2013-05-21	11	133	88	0.5113636363636364
2013-05-22	8	141	91	0.5494505494505495

(22 rows)

Rysunek 7.29. Wzrost liczby sprzedanych sztuk skutera Lemon

Kod źródłowy z tego fragmentu książki jest dostępny na stronie <https://packt.live/30CeOMm>.

Skorowidz

A

- agregacja, 120
- aktualizowanie
 - pól, 263
 - tabel, 81
- alias, 96
- Anaconda, 168, 169
- analitka danych, 88
- analiza
 - danych, 35, 123, 138
 - dwuczynnikowa, 50
 - jednoczynnikowa, 36, 38
 - przeżycia, 287
 - regresji, 51
 - współczynnika korelacji, 57
- analizowanie
 - sekwencji, 200
 - sprzedaży, 102
 - czas rozpoczęcia, 281
 - różnica cen, 288
 - skuteczność marketingu, 297
 - spadek, 280
 - wzrost, 290
 - tekstu, 210, 212
 - zmian współczynnika, 144
- analizy
 - geoprzestrzenne, 192, 195
 - post hoc, 222, 300
- anulowanie działającej kwerendy, 269

B

- badania terenowe, 300
- baza danych
 - metody skanowania, 224
 - zastosowanie języka Python, 168
 - zastosowanie języka R, 165
- B-drzewo, 231

C

- cecha, 34

D

- dane, 34
 - ilościowe
 - ciągłe, 35
 - dyskretne, 35
 - jakościowe, 35
 - niepełne, 61
- data i czas, 67
 - przedziały, 189
 - przekształcanie typów, 188
 - szeregi czasowe, 191
- definiowanie funkcji, 253
- długość geograficzna, 193
- dokumentacja systemu PostgreSQL, 92
- dyspersja, 48, 49

E

eksplorowanie danych sprzedażowych, 61
eksportowanie danych, 161, 181
e-mail

analiza skuteczności kampanii, 297
współczynnik otwarć, 290

Excel

Analiza dwuczynnikowa, 50
obliczanie
 dyspersji, 49
 kwartyli, 43
 miar tendencji centralnej, 47
 współczynnika korelacji Pearsona, 56
testy istotności statystycznej, 62
tworzenie histogramu, 38
wizualizacja danych, 164
wykresy punktowe, 50

F

filtrowanie, 206, 209, 215

format

DATE, 186
ISO, 186
JSON, 68, 201, 204
JSONB, 203
 dostęp do danych, 204
 modyfikowanie danych, 208
 tworzenie danych, 208
 wyszukiwanie wartości, 209

funkcja, 253

ARRAY, 199
ARRAY_AGG, 198
array_append, 199
array_cat, 199
AVG, 122
CASE WHEN, 108
CASTING, 114
COALESCE, 111
CORR, 122
COUNT (*), 140
COUNT, 122, 136, 139, 283
DATE, 278
DATE_TRUNC, 189, 191
DENSE_RANK, 147, 148
DISTINCT ON, 115
EXTRACT, 188
get_stock, 262

GREATEST, 113

insert_order, 262, 266
JSONB_ARRAY_ELEMENTS, 210
JSONB_OBJECT_KEYS, 205
jsonb_path_exists, 206
jsonb_path_query, 207
jsonb_pretty, 205
JSONB_PRETTY, 209, 210
LAG, 147, 286
LEAD, 147
LEAST, 113
MAX, 122, 278
MIN, 122
mode, 132
now, 187
NTILE, 147
NULLIF, 112
Percentile_cont, 132
Percentile_disc, 132
position, 293
RANK, 139, 147
REGEXP_REPLACE, 212, 213
REGR_INTERCEPT, 122
REGR_SLOPE, 122
ROW_NUMBER, 147
row_to_json, 202
STDDEV, 122
STRING_TO_ARRAY, 199, 211, 213
SUM, 122, 136
to_tsquery, 217
to_tsvector, 216, 217
ts_lexize, 212, 213
update_stock, 265
UNNEST, 213
update_stock, 262, 266
VAR, 122

funkcje

agregujące, 120
 analizowanie danych, 138
 dla zbiorów uporządkowanych, 132
 oczyszczanie danych, 135
 pomiar jakości danych, 137
 znajdowanie brakujących wartości, 135
okna, 139, 140
 obliczanie statystyk, 147
tablicowe, 199
tworzenie, 254, 258
wywoływanie, 255

G

generowanie listy, 105
 GIN, generalized inverted index, 218
 GiST, generalized search tree, 231
 Git, instalacja systemu, 23

H

hasło, 181
 hipoteza
 alternatywna, 62
 testowanie, 282, 288, 290
 zerowa, 62
 histogram, 38
 hurtownia danych, 34

I

ilościowa ocena spadku sprzedaży, 280
 importowanie danych, 181
 indeks
 bazy danych, 231
 bitmapowy, 236
 GIN, 218
 GiST, 231
 skuteczne korzystanie, 244
 w postaci B-drzewa, 231, 239, 242
 z haszowaniem, 231, 238–241
 informacje, 35
 instalowanie
 Anacondy, 169
 bibliotek, 32
 języka R, 165
 pakietu RPostgreSQL, 166
 Pythona
 Linux, 22
 macOS, 22
 Windows, 22
 systemu Git, 23
 systemu PostgreSQL
 Linux, 18
 macOS, 19
 Windows, 11
 instrukcja, *Patrz także*, klauzula, polecenie,
 słowo kluczowe
 ADD COLUMN, 81
 ANALYZE, 225, 236, 241, 248–251, 255
 CASE WHEN, 136
 COUNT DISTINCT, 137
 CREATE TABLE, 78

DECLARE, 254
 DELETE, 86, 294, 298
 DROP, 86, 265
 EXPLAIN, 225, 233, 241, 248–251
 FOR EACH, 261
 FROM, 69
 ILIKE, 215, 216
 INSERT, 160
 INSERT INTO...VALUES, 81
 LANGUAGE, 255
 OVER, 142, 280
 REINDEX, 244
 SELECT, 69, 77, 80, 225
 UNION, 105
 UNION ALL, 131, 132
 UPDATE, 83, 85
 integralność referencyjna, 92

J

jednostka obserwacji, 34
 jezioro danych, 34
 język
 JSONPath, 206
 Python, 168
 R, 165
 SQL, 64
 JSON, JavaScript Object Notation, 68, 201
 JSONB, 203–206
 JSONPath, 206, 209
 Jupyter Notebook, 172

K

klasyfikowanie zbioru danych, 37
 klauzula
 AND, 70
 AS, 96
 GROUP BY, 124, 130, 135
 GROUP BY dla kilku kolumn, 129
 GROUPING SETS, 131, 132, 138
 HAVING, 133, 134
 IN, 71
 IS NOT NULL, 75
 IS NULL, 75
 LIMIT, 74
 NOT IN, 71
 OR, 70
 ORDER BY, 72, 143, 280
 PARTITION BY, 141, 144
 WHERE, 70, 216

klauzula
 WINDOW, 146
 WITH, 107
 klient psychopg2, 169
 klucz
 .sales, 206
 główny, 64
 klucz_podziału, 140
 klucz_porządkowania, 140
 konfigurowanie zmiennej Path, 14
 konwerter obiektowo-relacyjny, 171
 kopiowanie danych, 157
 kwantyle, 42
 kwartyle, 43
 kwerendy
 aktywne, 270
 anulowanie, 269
 kończenie, 270

L

łączenie tabel, 91, *Patrz* łączenie

M

mediana, 46
 metoda najmniejszych kwadratów, 53
 miary tendencji centralnej, 47

N

narzędzie
 Jupyter Notebook, 172
 pandas, 171
 pip, 32
 psycopg, 157, 162
 SQLAlchemy, 171

O

obiekt JSON, 203
 obliczanie
 dyspersji, 49
 kwartyli, 43
 miar tendencji centralnej, 47
 statystyk, 147
 współczynnika korelacji Pearsona, 56
 ocenianie spadku sprzedaży, 280
 odchylenie standardowe, 48
 ograniczenie PRIMARY KEY, 137

operacje CRUD, 33
 operator
 #>, 204
 &&, 217
 @@, 218
 @>, 200, 204
 ||, 217
 ->, 204
 równości, 238
 operatory logiczne, 217

P

pakiet
 CREATE EXTENSION cube, 194
 CREATE EXTENSION earthdistance, 194
 RPostgreSQL, 166
 pandas, 171, 174
 pobieranie danych, 174
 zapisywanie w bazie, 174
 planer kwerend, 224, 225
 długość każdego wiersza, 228
 ignorowanie indeksu, 240
 koszt łączny, 227
 koszt operacji przygotowawczych, 227
 liczba zwracanych wierszy, 227
 nieaktualne indeksy, 244
 pełne złączenie zewnętrzne, 251
 skanowanie indeksu, 234
 typ skanowania, 227
 złączenie lewostronne, 250
 złączenie prawostronne, 250
 plik .pgpass, 182
 pliki CSV, 158, 162, 179
 pobieranie
 informacji, 276
 listy, 109
 podkwerendy, 103
 polecenie
 \copy, 158– 160
 \df, 259, 263
 \sf, 259
 COPY, 156, 160, 178
 masowe wczytywanie danych, 160
 opcje, 159
 pg_cancel_backend, 268
 pg_sleep, 268, 270
 pg_terminate_backend, 268, 270
 pomiar wydajności kwerendy JOIN, 248

populacja, 36
 PostgreSQL, 92
 analiza tekstu, 210
 analizy geoprzestrzenne, 192, 193
 dokumentacja systemu, 92
 format JSON, 201
 funkcje tablicowe, 199
 instalacja w systemie
 Linux, 18
 macOS, 19
 Windows, 11
 narzędzie psql, 157
 nawiązanie połączenia, 169, 226
 optymalizowanie wyszukiwania tekstu, 218
 polecenie COPY, 156, 160, 178
 tablice, 197
 ułatwianie dostępu, 171
 poziom istotności, 62
 predykat złączenia, 93
 proces ETL, 64
 próbka, 36
 punkt danych, 34
 punkty podziału, 42
 Python, 168
 instalacja w systemie
 Linux, 22
 macOS, 22
 Windows, 22
 odczyt plików CSV, 179
 wczytywanie danych, 175
 wizualizowanie danych, 175
 zapisywanie danych, 177
 pliki CSV, 179
 zwiększanie szybkości, 178

R

ramka okna, 149
 analizowanie sprzedaży, 153
 wyszukiwanie informacji, 151
 rekurencyjność, 108
 relacje geoprzestrzenne, 193
 relacyjna baza danych, 64
 rozkład
 bezwzględnej częstości występowania, 38
 względnej częstości występowania, 38
 rozstęp, 48
 ćwiartkowy, 49

S

schemat, 64
 sekwencje e-maili, 200
 serwis GitHub, 56
 skanowanie
 baz danych, 224
 indeksu, 230, 233, 234
 sekwencyjne, 224, 229
 skośny zbiór danych, 46
 słowo kluczowe
 BEGIN, 255, 256
 current_date, 187
 JOIN, 91, *Patrz* złączenie
 NEW, 266
 PRECEDING, 152
 TEMP, 159
 TRIGGER, 261
 UNBOUNDED PRECEDING, 150
 SQL, Structured Query Language, 33, 64
 pobieranie informacji, 276
 typy danych, 66
 wady i zalety, 64
 zbieranie danych, 274
 SQLAlchemy, 171, 172
 statystyka, 35
 opisowa, 36, 37
 testu, 62
 statystyki wieloczynnikowe, 36
 stemming, 212
 strefa UTC, 187
 struktury danych, 68
 suma, 104
 adresów, 105
 skumulowana, 285
 system bazodanowy PostgreSQL, 92
 szereg czasowy, 60, 191
 szerokość geograficzna, 193

Ś

średnia, 46
 krocząca, 150
 środowisko IDE, 166

T

tabele
 aktualizowanie, 81
 aktualizowanie wierszy, 83
 dodawanie danych, 81

tabele
 dodawanie kolumn, 81
 łączenie, 91
 modyfikowanie, 87
 rodzaje złączeń, 93
 tworzenie, 78
 usuwanie, 86
 danych, 85
 kolumn, 81
 wartości z wiersza, 85
 wierszy, 86

tablice, 68
 analizowanie sekwencji, 200
 tworzenie, 198

tekst
 analizowanie, 210, 212
 optymalizowanie wyszukiwania, 218
 tokenizacja, 211
 wyszukiwanie, 216

tendencja centralna, 46

test
 A/B, 301
 Pearsona, 63
 różnic średnich, 287
 T dla dwóch próbek, 63
 Z dla dwóch próbek, 63
 zgodności chi-kwadrat, 63

testowanie hipotezy, 282, 288, 290

tokenizacja tekstu, 211

tokeny, 214

trend, 52, 182
 liniowy, 54

tworzenie
 funkcji, 253, 254, 258
 histogramu, 38
 indeksu, 232
 indeksu z haszowaniem, 239
 tabel, 78
 tablicy, 198
 widoku, 159, 162
 wyzwalaczy, 263

typ danych, 64
 ARRAY, 197
 DATE, 185

INTERVAL, 189
 logiczny, 67
 point, 194
 TIMESTAMP, 187
 tsvector, 217

typy
 liczbowe, 66
 z datą i godziną, 67
 znakowe, 66

U

usuwanie
 tabel, 86
 widoku, 162

W

wariancja, 48, 287

wartość
 modalna, 46
 NULL, 85

wczytywanie tabel, 68

widok
 tymczasowy, 162
 zmaterializowany, 218

wnioskowanie statystyczne, 36

współczynnik
 klikalności, 62
 korelacji Pearsona, 53, 55, 56
 otwarć e-maili, 290

wydajność
 funkcji, 255
 indeksów z haszowaniem, 239
 kwerendy, 251
 złączenia, 249

wykreś punktowy, 50, 53, 54

wykrywanie trendu, 182

wyrażenie filtrujące, 206

wyszukiwanie tekstu, 216
 optymalizowane, 218

wyzwalacz, 260
 śledzący dane, 267
 tworzenie, 263

zdarzenie
 DELETE, 261
 INSERT, 261
 TRUNCATE, 261

Z

zbieranie danych, 274

zbiór danych, 34

 dla systemu Linux, 26

 dla systemu macOS, 29

 dla systemu Windows, 23

zdarzenie, 261

złączenie, JOIN, 91, 245

 krzyżowe, CROSS JOIN, 100

 lewostronne, LEFT JOIN, 249

 prawostronne, RIGHT JOIN, 250

 wewnętrzne, INNER JOIN, 93, 246, 247

 zewewnętrzne, OUTER JOIN, 96, 251

 lewostronne, 96

 pełne, FULL OUTER JOIN, 100, 251

 prawostronne, 98

złożone typy danych, 184

zmiana trendu, 52

zmienna, 34


 Path, 14

znaczniki czasu, 190, 229

znak zachęty, 255

PROGRAM PARTNERSKI

— GRUPY HELION —

- 
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

BLENDDED 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