

Paul DuBois



Wydanie V

MySQL[®]

Vademecum profesjonalisty

Kompendium wiedzy o MySQL!



Helion

Tłumaczenie: Robert Górczyński

ISBN: 978-83-246-8149-5

Authorized translation from the English language edition, entitled: MYSQL, Fifth Edition; ISBN 0321833872; by Paul DuBois; published by Pearson Education, Inc, publishing as Addison Wesley. Copyright © 2013 Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc. Polish language edition published by HELION S.A. Copyright © 2014.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

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

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

Wydawnictwo HELION

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

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

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
<ftp://ftp.helion.pl/przyklady/mysvp5.zip>

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

http://helion.pl/user/opinie/mysvp5_ebook

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

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

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

Spis treści

O autorze	17
Wprowadzenie	19
 Część I Ogólne użycie MySQL	
Rozdział 1. Rozpoczęcie pracy z MySQL	33
1.1. W jaki sposób MySQL może Ci pomóc?	33
1.2. Przykładowa baza danych	37
1.2.1. Projekt „Liga Historyczna USA”	37
1.2.2. Projekt „Oceny uczniów”	40
1.2.3. Jak przykładowa baza danych może Ci pomóc?	41
1.3. Podstawowa terminologia bazy danych	41
1.3.1. Terminologia strukturalna	42
1.3.2. Terminologia języka zapytań	44
1.3.3. Terminologia architekuralna MySQL	45
1.4. Przewodnik po MySQL	46
1.4.1. Pobranie przykładowej bazy danych	47
1.4.2. Wymagania wstępne	47
1.4.3. Nawiązywanie i zrywanie połączenia z serwerem MySQL	49
1.4.4. Wykonywanie zapytań SQL	51
1.4.5. Tworzenie bazy danych	54
1.4.6. Tworzenie tabel	55
1.4.7. Dodawanie nowych rekordów	75
1.4.8. Przywrócenie bazy danych sampdb do znanego stanu	79
1.4.9. Pobieranie informacji	80
1.4.10. Usuwanie lub uaktualnianie istniejących rekordów	110
1.5. Wskazówki przydatne podczas pracy z klientem mysql	113
1.5.1. Uproszczenie procesu nawiązywania połączenia	113
1.5.2. Wpisywanie mniejszej ilości tekstu przy wykonywaniu zapytań	115
1.6. Co dalej?	120

Rozdział 2. Użycie MySQL do zarządzania danymi	123
2.1. Tryby SQL serwera	124
2.2. Identyfikatory składni MySQL i reguły nadawania nazw	126
2.3. Wielkość liter w zapytaniach SQL	128
2.4. Obsługa kodowania znaków	130
2.4.1. Określenie kodowania znaków	131
2.4.2. Określenie dostępności kodowania znaków i ustawień bieżących	132
2.4.3. Obsługa Unicode	133
2.5. Wybór, utworzenie, usunięcie i zmiana bazy danych	134
2.5.1. Wybór bazy danych	134
2.5.2. Utworzenie bazy danych	135
2.5.3. Usunięcie bazy danych	136
2.5.4. Zmiana bazy danych	136
2.6. Tworzenie, usuwanie, indeksowanie i modyfikowanie tabel	137
2.6.1. Cechy charakterystyczne silników bazy danych	137
2.6.2. Tworzenie tabel	143
2.6.3. Usuwanie tabel	152
2.6.4. Indeksowanie tabel	153
2.6.5. Zmiana struktury tabeli	158
2.7. Pobieranie metadanych bazy danych	161
2.7.1. Pobieranie metadanych za pomocą zapytania SHOW	162
2.7.2. Pobieranie metadanych z bazy danych INFORMATION_SCHEMA	164
2.7.3. Pobieranie metadanych z poziomu wiersza poleceń	166
2.8. Pobieranie danych z wielu tabel za pomocą złączeń	167
2.8.1. Złączenia wewnętrzne	169
2.8.2. Kwalifikowane odwołania do kolumn z poziomu złączonych tabel	170
2.8.3. Złączenia typu LEFT i RIGHT (OUTER)	171
2.9. Pobieranie informacji z wielu tabel za pomocą podzapytań	175
2.9.1. Podzapytania używające względnych operatorów porównania	176
2.9.2. Podzapytania IN i NOT IN	177
2.9.3. Podzapytania ALL, ANY i SOME	178
2.9.4. Podzapytania EXISTS i NOT EXISTS	179
2.9.5. Podzapytania skorelowane	180
2.9.6. Podzapytania w klauzuli FROM	181
2.9.7. Przepisywanie podzapytań na postać złączeń	181
2.10. Pobieranie informacji z wielu tabel za pomocą zapytań UNION	183
2.11. Usuwanie i uaktualnianie rekordów w wielu tabelach	186

2.12. Przeprowadzanie transakcji	188
2.12.1. Użycie transakcji do zapewnienia bezpiecznego środowiska wykonywania	190
2.12.2. Użycie punktów pośrednich transakcji	194
2.12.3. Izolacja transakcji	194
2.13. Klucze zewnętrzne i integralność odwołań	197
2.14. Wyszukiwanie pełnego tekstu	204
2.14.1. Wyszukiwanie pełnego tekstu w języku naturalnym	207
2.14.2. Wyszukiwanie pełnego tekstu w trybie boolowskim	208
2.14.3. Wyszukiwanie rozszerzone o wynik zapytania	210
2.14.4. Konfiguracja silnika wyszukiwania pełnego tekstu	211

Rozdział 3. Typy danych213

3.1. Kategorie wartości danych	215
3.1.1. Wartości liczbowe	215
3.1.2. Wartości ciągu tekstowego	217
3.1.3. Wartości daty i godziny	226
3.1.4. Wartości przestrzenne	226
3.1.5. Wartości boolowskie	227
3.1.6. Wartość NULL	227
3.2. Typy danych w MySQL	227
3.2.1. Ogólny opis typów danych	228
3.2.2. Określenie typów kolumn w definicji tabeli	230
3.2.3. Definiowanie wartości domyślnych kolumn	231
3.2.4. Liczbowe typy danych	232
3.2.5. Typy danych w postaci ciągów tekstowych	240
3.2.6. Typy danych dla wartości daty i czasu	256
3.3. Jak MySQL obsługuje nieprawidłowe wartości danych?	267
3.4. Praca z sekwencjami	270
3.4.1. Ogólne właściwości AUTO_INCREMENT	270
3.4.2. Właściwości AUTO_INCREMENT charakterystyczne dla silnika bazy danych	272
3.4.3. Kwestie dotyczące kolumn AUTO_INCREMENT	275
3.4.4. Wskazówki pomocne podczas pracy z kolumnami AUTO_INCREMENT	276
3.4.5. Generowanie sekwencji bez AUTO_INCREMENT	278
3.5. Obliczanie wyrażeń i konwersja typu	280
3.5.1. Tworzenie wyrażeń	281
3.5.2. Konwersja typu	289
3.6. Wybór typu danych	297
3.6.1. Jakie wartości będą przechowywane w kolumnie?	299
3.6.2. Czy przechowywane wartości pochodzą z określonego zakresu?	302

Rozdział 4. Widoki i programy składowane	305
4.1. Używanie widoków	306
4.2. Używanie programów składowanych	309
4.2.1. Zapytania złożone i ograniczniki zapytań	310
4.2.2. Procedury i funkcje składowane	311
4.2.3. Wyzwalacze	316
4.2.4. Zdarzenia	318
4.3. Zapewnienie bezpieczeństwa widokom i programom składowanym	320

Rozdział 5. Optymalizacja zapytań	323
5.1. Użycie indeksowania	324
5.1.1. Zalety wynikające z indeksowania	324
5.1.2. Koszt indeksowania	327
5.1.3. Wybór indeksów	328
5.2. Optymalizator zapytań MySQL	333
5.2.1. Jak działa optymalizator zapytań?	334
5.2.2. Użycie zapytania EXPLAIN do sprawdzenia operacji optymalizatora	338
5.3. Wybór typu danych zapewniającego efektywne wykonywanie zapytań	344
5.4. Wybór formatu tabeli dla efektywnych zapytań	347
5.5. Efektywne wczytywanie danych	349
5.6. Harmonogram, blokady i współbieżność	352

Część II Użycie interfejsu programowania MySQL

Rozdział 6. Wprowadzenie do programowania MySQL	355
6.1. Dlaczego tworzyć własne aplikacje MySQL?	355
6.2. API dostępne dla MySQL	359
6.2.1. API C	360
6.2.2. API Perl DBI	360
6.2.3. API PHP	362
6.3. Wybór API	363
6.3.1. Przewidywane środowisko działania	364
6.3.2. Wydajność	365
6.3.3. Czas potrzebny na opracowanie aplikacji	366
6.4.3. Przenośność	367

Rozdział 7. Tworzenie programów MySQL przy użyciu języka C	369
7.1. Kompilacja i linkowanie programów klienckich	370
7.2. Nawiązanie połączenia z serwerem	373
7.3. Obsługa błędów i przetwarzanie opcji polecenia	377
7.3.1. Sprawdzanie pod kątem błędów	377

7.3.2. Pobieranie parametrów połączenia w trakcie działania programu	382
7.3.3. Implementacja przetwarzania opcji w programie klienta	395
7.4. Przetwarzanie zapytań SQL	399
7.4.1. Obsługa zapytań modyfikujących rekordy	401
7.4.2. Obsługa zapytań zwracających zbiór wynikowy	402
7.4.3. Procedury obsługi zapytań ogólnego przeznaczenia	405
7.4.4. Alternatywne podejścia do przetwarzania zapytań	407
7.4.5. Funkcja <code>mysql_store_result()</code> kontra <code>mysql_use_result()</code>	408
7.4.6. Używanie metadanych zbioru wynikowego	411
7.4.7. Kodowanie znaków specjalnych i danych binarnych	416
7.5. Program do interaktywnego wykonywania zapytań	420
7.6. Utworzenie klienta z obsługą SSL	421
7.7. Jednoczesne wykonywanie wielu zapytań	426
7.8. Używanie zapytań preinterpretowanych	428
7.9. Użycie preinterpretowanego zapytania CALL	440
Rozdział 8. Tworzenie programów MySQL przy użyciu Perl DBI	445
8.1. Cechy charakterystyczne skryptu Perl	446
8.2. Ogólny opis Perl DBI	447
8.2.1. Typy danych DBI	447
8.2.2. Prosty skrypt DBI	448
8.2.3. Obsługa błędów	453
8.2.4. Obsługa zapytań modyfikujących rekordy	456
8.2.5. Obsługa zapytań zwracających zbiór wynikowy	458
8.2.6. Cytowanie znaków specjalnych w ciągach tekstowych zapytań	467
8.2.7. Miejsca zarezerwowane i zapytania preinterpretowane	470
8.2.8. Dołączanie wyniku zapytania do zmiennych skryptu	474
8.2.9. Określenie parametrów połączenia	475
8.2.10. Usuwanie błędów	478
8.2.11. Używanie metadanych zbioru wynikowego	482
8.2.12. Przeprowadzanie transakcji	486
8.3. Praca z DBI	488
8.3.1. Generowanie katalogu Ligi Historycznej	488
8.3.2. Wysyłanie członkom Ligi przypomnień o konieczności przedłużenia członkostwa	495
8.3.3. Edycja rekordu członka Ligi Historycznej	500
8.3.4. Wyszukiwanie członków Ligi Historycznej o takich samych zainteresowaniach	506
8.3.5. Udostępnienie w internecie katalogu Ligi Historycznej	507
8.4. Użycie DBI w aplikacjach sieciowych	510
8.4.1. Konfiguracja Apache w celu obsługi skryptów CGI	512
8.4.2. Krótki opis modułu <code>CGI.pm</code>	513

8.4.3. Nawiązanie połączenia z serwerem MySQL z poziomu skryptu sieciowego	521
8.4.4. Przeglądarka bazy danych	523
8.4.5. Przeglądarka tabel projektu ocen uczniów	528
8.4.6. Wyszukiwanie wspólnych zainteresowań w projekcie Ligi Historycznej	531
Rozdział 9. Tworzenie programów MySQL przy użyciu języka PHP	537
9.1. Ogólny opis PHP	539
9.1.1. Prosty skrypt PHP	541
9.1.2. Użycie biblioteki plików PHP w celu hermetyzacji kodu ...	545
9.1.3. Prosta strona pobierająca dane	550
9.1.4. Przetwarzanie wyników zapytania	554
9.1.5. Sprawdzanie pod kątem wartości NULL w wynikach zapytania	558
9.1.6. Używanie zapytań preinterpretowanych	559
9.1.7. Użycie miejsc zarezerwowanych do obsługi kwestii związanych z cytowaniem danych	559
9.1.8. Obsługa błędów	561
9.2. Praca z PHP	564
9.2.1. Aplikacja pozwalająca na wprowadzenie wyników uzyskanych przez uczniów	564
9.2.2. Tworzenie interaktywnego quizu w internecie	577
9.2.3. Edycja informacji o członkach Ligi Historycznej	582
Część III Administracja MySQL	
Rozdział 10. Wprowadzenie do administracji bazą danych MySQL	591
10.1. Komponenty MySQL	592
10.2. Ogólna administracja MySQL	593
10.3. Kontrola dostępu i zapewnianie bezpieczeństwa	594
10.4. Obsługa bazy danych, tworzenie kopii zapasowych i replikacja	595
Rozdział 11. Katalog danych w MySQL	597
11.1. Położenie katalogu danych	598
11.2. Struktura katalogu danych	599
11.2.1. W jaki sposób serwer MySQL zapewnia dostęp do danych?	600
11.2.2. Przedstawienie baz danych w systemie plików	602
11.2.3. Przedstawienie tabel w systemie plików	603
11.2.4. Przedstawienie widoków i wyzwalaczy w systemie plików	604
11.2.5. Jak zapytania SQL są mapowane na operacje na pliku tabeli?	604
11.2.6. Ograniczenia systemu operacyjnego w zakresie nazw obiektów bazy danych	605

11.2.7. Czynniki wpływające na maksymalną wielkość tabeli	607
11.2.8. Wpływ struktury katalogu danych na wydajność systemu	609
11.2.9. Pliki dzienników zdarzeń i stanu MySQL	611
11.3. Przeniesienie zawartości katalogu danych	613
11.3.1. Metody przenoszenia katalogu danych lub jego elementów	614
11.3.2. Przygotowania do operacji przeniesienia	615
11.3.3. Uzyskanie dostępu do wyniku przeniesienia	615
11.3.4. Przeniesienie całego katalogu danych	617
11.3.5. Przeniesienie poszczególnych baz danych	617
11.3.6. Przeniesienie poszczególnych tabel	618
11.3.7. Przeniesienie systemowej przestrzeni tabel InnoDB	618
11.3.8. Przeniesienie plików dzienników zdarzeń i stanu	619
Rozdział 12. Ogólna administracja bazą danych MySQL	621
12.1. Zabezpieczenie nowej instalacji MySQL	622
12.1.1. Definiowanie haseł dla początkowych kont MySQL	622
12.1.2. Konfiguracja haseł dla serwerów dodatkowych	627
12.2. Konfiguracja uruchamiania i zamykania serwera MySQL	628
12.2.1. Uruchamianie serwera MySQL w systemach UNIX	628
12.2.2. Uruchamianie serwera MySQL w systemach Windows	634
12.2.3. Określanie opcji startowych serwera	637
12.2.4. W jaki sposób serwer nasłuchuje połączeń?	639
12.2.5. Zatrzymanie serwera	640
12.2.6. Odzyskanie kontroli nad serwerem, gdy nie można nawiązać z nim połączenia	641
12.3. Używanie zmiennych systemowych i stanu	644
12.3.1. Sprawdzanie i ustawienie wartości zmiennych systemowych	645
12.3.2. Sprawdzenie wartości zmiennej stanu	649
12.4. Interfejs wtyczek	651
12.5. Konfiguracja silnika bazy danych	654
12.5.1. Wybór silnika bazy danych	655
12.5.2. Wybór domyślnego silnika bazy danych	656
12.5.3. Konfiguracja silnika InnoDB	656
12.6. Kwestie związane z globalizacją	664
12.6.1. Konfiguracja obsługi stref czasowych	664
12.6.2. Ustawienie domyślnego kodowania znaków i kolejności sortowania	667
12.6.3. Wybór języka wyświetlania komunikatów błędów	667
12.6.4. Wybór ustawień językowych	668

12.7. Dostrajanie serwera	668
12.7.1. Zmienne systemowe ogólnego przeznaczenia do dostrajania serwera	670
12.7.2. Dostrajanie silnika bazy danych	673
12.7.3. Używanie bufora zapytań	680
12.7.4. Optymalizacje sprzętowe	682
12.8. Dzienniki zdarzeń serwera	684
12.8.1. Dziennik błędów	687
12.8.2. Ogólny dziennik zapytań	688
12.8.3. Dziennik wolno wykonywanych zapytań	689
12.8.4. Binarny dziennik zdarzeń	690
12.8.5. Dziennik przekazywania	692
12.8.6. Rejestracja zdarzeń w tabelach	692
12.8.7. Zarządzanie dziennikami zdarzeń	693
12.9. Uruchamianie wielu serwerów MySQL	701
12.9.1. Ogólne kwestie związane z uruchamianiem wielu serwerów	702
12.9.2. Konfiguracja i kompilacja różnych serwerów	705
12.9.3. Strategie podawania opcji startowych	707
12.9.4. Użycie skryptu mysqld_multi do zarządzania serwerem	708
12.9.5. Uruchamianie wielu serwerów w Windows	710
12.9.6. Uruchamianie klientów wielu serwerów	712
12.10. Uaktualnianie MySQL	712

Rozdział 13. Bezpieczeństwo i kontrola dostępu715

13.1. Zabezpieczenie dostępu do MySQL przez system plików	716
13.1.1. Jak ukraść dane?	717
13.1.2. Zabezpieczenie instalacji MySQL	718
13.2. Zarządzanie kontami użytkowników w MySQL	725
13.2.1. Zarządzanie kontem MySQL na wysokim poziomie	727
13.2.2. Nadawanie uprawnień	732
13.2.3. Wyświetlanie uprawnień użytkownika	744
13.2.4. Odbieranie uprawnień	744
13.2.5. Zmiana hasła lub zerowanie zapomnianego	745
13.2.6. Unikanie ryzyka związanego z kontrolą dostępu	746
13.2.7. Wtyczki metod uwierzytelniania i użytkownicy proxy ...	750
13.3. Struktura i zawartość tabel uprawnień	754
13.3.1. Istniejące w tabelach uprawnień kolumny dotyczące zasięgu udzielanego dostępu	757
13.3.2. Istniejące w tabelach uprawnień kolumny uprawnień	758
13.3.3. Istniejące w tabelach uprawnień kolumny uwierzytelnienia	759

13.3.4. Istniejące w tabelach uprawnień kolumny dotyczące SSL	759
13.3.5. Istniejące w tabelach uprawnień kolumny zarządzania zasobami	760
13.4. W jaki sposób serwer kontroluje dostęp uzyskiwany przez klientów?	761
13.4.1. Zawartość kolumn zasięgu	762
13.4.2. Weryfikacja uprawnień zapytania	764
13.4.3. Kolejność dopasowania kolumn zasięgu	766
13.4.4. Puzzle uprawnień	767
13.5. Konfiguracja bezpiecznych połączeń za pomocą SSL	770
Rozdział 14. Obsługa bazy danych, kopie zapasowe i replikacja	775
14.1. Zasady obsługi profilaktycznej	775
14.2. Obsługa bazy danych w działającym serwerze	777
14.2.1. Blokowanie poszczególnych tabel w celu uzyskania dostępu tylko do odczytu lub odczytu i zapisu	779
14.2.2. Nałożenie na wszystkie bazy danych blokady pozwalającej jedynie na ich odczyt	782
14.3. Ogólne działania profilaktyczne	783
14.3.1. Używanie możliwości serwera w zakresie automatycznej naprawy	783
14.3.2. Harmonogram działań profilaktycznych	784
14.4. Tworzenie kopii zapasowej bazy danych	785
14.4.1. Cechy charakterystyczne przenośności silników bazy danych	788
14.4.2. Tworzenie kopii zapasowej za pomocą narzędzia mysqldump	790
14.4.3. Tworzenie binarnej kopii zapasowej	793
14.4.4. Tworzenie kopii zapasowej tabel InnoDB	795
14.5. Kopiowanie baz danych do innego serwera	796
14.5.1. Kopiowanie bazy danych za pomocą pliku kopii zapasowej	797
14.5.2. Kopiowanie baz danych z jednego serwera do innego	798
14.6. Sprawdzanie i naprawianie tabel bazy danych	798
14.6.1. Sprawdzanie tabel za pomocą zapytania CHECK TABLE	800
14.6.2. Naprawa tabel za pomocą zapytania REPAIR TABLE	801
14.6.3. Użycie narzędzia mysqlcheck do sprawdzania i naprawy tabel	801
14.7. Użycie kopii zapasowej do przywrócenia danych	803
14.7.1. Przywracanie całych baz danych	803
14.7.2. Przywracanie poszczególnych tabel	804

14.7.3. Ponowne wykonanie zapytań zapisanych w plikach binarnego dziennika zdarzeń	805
14.7.4. Rozwiązywanie problemów związanych z automatyczną naprawą w InnoDB	807
14.8. Konfiguracja serwerów replikacji	809
14.8.1. Jak działa replikacja?	809
14.8.2. Utworzenie relacji typu główny – podległy	811
14.8.3. Formaty rejestracji zdarzeń w dzienniku binarnym	815
14.8.4. Użycie serwera podległego replikacji do tworzenia kopii zapasowych	815

Dodatki

Dodatek A Oprogramowanie wymagane do użycia tej książki819

A.1. Pobranie dystrybucji sampdb zawierającej przykładową bazę danych	819
A.2. Pobieranie serwera MySQL i powiązanego z nim oprogramowania	820
A.3. Instalacja MySQL	822
A.3.1. Tworzenie konta logowania dla użytkownika MySQL	823
A.3.2. Instalacja MySQL	823
A.3.3. Konfiguracja zmiennej środowiskowej PATH	824
A.3.4. Inicjalizacja katalogu danych i tabel uprawnień	825
A.3.5. Uruchamianie serwera	826
A.3.6. Inicjalizacja innych tabel systemowych	827
A.4. Informacje dotyczące instalacji Perl DBI	828
A.5. Informacje dotyczące instalacji PHP i PDO	829

Dodatek B Przewodnik po typach danych831

B.1. Typy liczbowe	833
B.1.1. Typy liczb całkowitych	834
B.1.2. Typy liczb o stałej ilości cyfr	836
B.1.3. Typy liczb zmiennoprzecinkowych	836
B.1.4. Typ BIT	837
B.2. Typy tekstowe	838
B.2.1. Typy binarnych ciągów tekstowych	840
B.2.2. Typy niebinarnych ciągów tekstowych	842
B.2.3. Typy ENUM i SET	845
B.3. Typy daty i godziny	845

Dodatek C Przewodnik po operatorach i funkcjach851

C.1. Operatory	852
C.1.1. Kolejność operatorów	852
C.1.2. Operatory grupowania	853
C.1.3. Operatory arytmetyczne	854

C.1.4. Operatory porównania	856
C.1.5. Operatory bitowe	861
C.1.6. Operatory logiczne	862
C.1.7. Operatory rzutowania	864
C.1.8. Operatory dopasowania wzorca	865
C.2. Funkcje	869
C.2.1. Funkcje porównań	870
C.2.2. Funkcje rzutowania	872
C.2.3. Funkcje liczbowe	873
C.2.4. Funkcje ciągu tekstowego	878
C.2.5. Funkcje daty i godziny	892
C.2.6. Funkcje podsumowań	907
C.2.7. Funkcje zapewnienia bezpieczeństwa oraz związane z kompresją	911
C.2.8. Funkcje nakładania blokad doradczych	915
C.2.9. Funkcje związane z adresem IP	917
C.2.10. Funkcje XML	919
C.2.11. Funkcje przestrzenne	920
C.2.12. Różne funkcje	920
Dodatek D Przewodnik po zmiennych systemowych, stanu i użytkownika	927
D.1. Zmienne systemowe	927
D.1.1. Zmienne systemowe InnoDB	972
D.2. Zmienne stanu	987
D.2.1. Zmienne stanu InnoDB	994
D.2.2. Zmienne stanu bufora zapytań	998
D.2.3. Zmienne stanu SSL	999
D.3. Zmienne zdefiniowane przez użytkownika	1000
Dodatek E Przewodnik po składni SQL	1003
E.1. Składnia zapytań SQL (zapytania niezłożone)	1004
E.2. Składnia zapytań SQL (zapytania złożone)	1103
E.2.1. Polecenia struktury kontrolnej	1103
E.2.2. Zapytania obsługi deklaracji	1105
E.2.3. Zapytania obsługi kursora	1107
E.2.4. Zapytania obsługi warunków	1108
E.3. Składnia komentarzy	1112
Dodatek F Przewodnik po programie SQL	1115
F.1. Wyświetlanie komunikatu pomocy programu	1116
F.2. Określanie opcji programu	1117
F.2.1. Opcje standardowe programu MySQL	1119
F.2.2. Pliki opcji	1124
F.2.3. Zmienne środowiskowe	1129

F.3. Narzędzie myisamchk	1130
F.3.1. Opcje standardowe obsługiwane przez myisamchk	1132
F.3.2. Opcje charakterystyczne dla myisamchk	1132
F.3.3. Zmienne narzędzia myisamchk	1136
F.4. mysql	1137
F.4.1. Opcje standardowe obsługiwane przez mysql	1139
F.4.2. Opcje charakterystyczne dla mysql	1140
F.4.3. Zmienne programu mysql	1145
F.4.4. Polecenia programu mysql	1145
F.4.5. Sekwencje definiujące znak zachęty mysql	1148
F.5. mysql.server	1150
F.5.1. Opcje obsługiwane przez mysql.server	1150
F.6. mysql_config	1150
F.6.1. Opcje charakterystyczne dla mysql_config	1151
F.7. Skrypt mysql_install_db	1152
F.7.1. Opcje standardowe obsługiwane przez mysql_install_db	1152
F.7.2. Opcje charakterystyczne dla mysql_install_db	1152
F.8. mysql_upgrade	1153
F.8.1. Opcje standardowe obsługiwane przez mysql_upgrade	1153
F.8.2. Opcje charakterystyczne dla mysql_upgrade	1154
F.9. mysqladmin	1154
F.9.1. Opcje standardowe obsługiwane przez mysqladmin	1154
F.9.2. Opcje charakterystyczne dla mysqladmin	1155
F.9.3. Zmienne dla mysqladmin	1155
F.9.4. Polecenia mysqladmin	1156
F.10. mysqlbinlog	1158
F.10.1. Opcje standardowe obsługiwane przez mysqlbinlog	1159
F.10.2. Opcje charakterystyczne dla mysqlbinlog	1159
F.10.3. Zmienne dla mysqlbinlog	1162
F.11. mysqlcheck	1163
F.11.1. Opcje standardowe obsługiwane przez mysqlcheck	1163
F.11.2. Opcje charakterystyczne dla mysqlcheck	1163
F.12. mysqld	1167
F.12.1. Opcje standardowe obsługiwane przez mysqld	1168
F.12.2. Opcje charakterystyczne dla mysqld	1168
F.12.3. Zmienne dla mysqld	1180
F.13. mysqld_multi	1180
F.13.1. Opcje standardowe obsługiwane przez mysqld_multi	1180
F.13.2. Opcje charakterystyczne dla mysqld_multi	1181

F.14. mysqld_safe	1181
F.14.1. Opcje standardowe obsługiwane przez mysqld_safe	1182
F.14.2. Opcje charakterystyczne dla mysqld_safe	1182
F.15. mysqldump	1184
F.15.1. Opcje standardowe obsługiwane przez mysqldump	1184
F.15.2. Opcje charakterystyczne dla mysqldump	1185
F.15.3. Oferowane przez mysqldump opcje formatu danych	1193
F.15.4. Zmienne dla mysqldump	1193
F.16. mysqlimport	1194
F.16.1. Opcje standardowe obsługiwane przez mysqlimport ...	1194
F.16.2. Opcje charakterystyczne dla mysqlimport	1194
F.16.3. Oferowane przez mysqlimport opcje formatu danych	1196
F.17. mysqlshow	1196
F.17.1. Opcje standardowe obsługiwane przez mysqlshow	1197
F.17.2. Opcje charakterystyczne dla mysqlshow	1197
F.18. perror	1198
F.18.1. Opcje standardowe obsługiwane przez perror	1198
Skorowidz	1199

O autorze

Paul DuBois jest administratorem baz danych oraz liderem w społecznościach open source i MySQL. Ma swój wkład w powstanie dokumentacji MySQL, a ponadto jest autorem wielu książek, między innymi *MySQL and Perl for the Web* (New Riders), *MySQL Cookbook*, *Using csh and tcsh* i *Software Portability with imake* (O'Reilly). Aktualnie zajmuje się tworzeniem dokumentacji technicznej w firmie Oracle.

Podziękowania

Mój recenzent techniczny Stephen Frein dostarczył wielu cennych wskazówek i podpowiedzi pozwalających na ulepszenie niniejszej książki. Ponadto, ponieważ powstanie tego wydania byłoby niemożliwe bez wcześniejszych, podziękowania kieruję również pod adresem wszystkich osób wymienionych w poprzednich wydaniach, a także tych, które udzielały się w charakterze recenzentów technicznych lub po prostu cierpliwie odpowiadały na moje pytania.

W wydawnictwie Pearson za przygotowanie niniejszego wydania odpowiadały następujące osoby: Mark Taber, redaktor, Tonya Simpson, redaktor projektu, Sarah Kearns, redaktor, Kim Scott, odpowiedzialna za skład tekstu, Jess DeGabriele, korektor, Heather McNeill, odpowiedzialna za przygotowanie indeksu, i Chuti Prasertsith, która przygotowała okładkę. Podziękowania składam wszystkim wymienionym osobom.

Dziękuję również mojej żonie Karen za wsparcie i słowa zachęty podczas prac nad przygotowaniem tej książki.

Wprowadzenie

System relacyjnych baz danych (ang. *Relational Database Management System*, RDBMS) to niezbędne narzędzie w wielu środowiskach; ma szerokie zastosowanie, począwszy od biznesu, przez badania, edukację aż do dostarczania treści w internecie. Jednak pomimo ogromnej wagi posiadania dobrego systemu bazy danych w celu zarządzania informacjami i uzyskiwania do nich dostępu, dla wielu organizacji nabycie tego rodzaju systemu pozostaje poza możliwościami finansowymi. Od początku istnienia systemy baz danych są bardzo drogimi produktami, a ich producenci pobierają ogromne opłaty zarówno za oprogramowanie, jak i pomoc techniczną. Ponadto, ponieważ silniki baz danych bardzo często mają duże wymagania sprzętowe, których spełnienie jest konieczne do zapewnienia sensownej wydajności działania, całkowity koszt wdrożenia rozwiązania rośnie jeszcze bardziej.

Czasy się jednak zmieniają — to dotyczy zarówno sprzętu, jak i oprogramowania. Niewielkie systemy biurkowe i serwery stały się znacząco tańsze, a jednocześnie oferują całkiem dużą wydajność działania. Obserwujemy także dobrze rozwijający się ruch, którego celem jest tworzenie systemów operacyjnych o wysokiej wydajności przeznaczonych dla niewielkich systemów biurkowych i serwerów. Wspomniane systemy operacyjne są bezpłatnie dostępne w internecie lub za niewielką opłatą na nośnikach fizycznych (CD/DVD). Powstały na bazie systemu BSD UNIX i wielu dystrybucji systemu Linux.

Opracowanie bezpłatnych systemów operacyjnych przebiega jednocześnie z powstaniem wielu narzędzi open source, a nawet do pewnego stopnia stało się możliwe dzięki powstaniu narzędzi takich jak gcc — kompilator GNU C, Apache — najbardziej rozpowszechniony w internecie serwer WWW, a także ogólnego przeznaczenia języków skryptowych, na przykład Perl, PHP, Python i Ruby. To zupełne przeciwieństwo rozwiązań własnościowych, które zamykają użytkowników w drogich produktach dostarczanych przez ich producentów bez kodu źródłowego.

Systemy baz danych również stały się łatwiej dostępne dla użytkowników, a udostępniane w ramach projektów typu open source są dostępne bezpłatnie. Jednym z najważniejszych systemów tego typu jest MySQL, czyli pochodzący ze Skandynawii system zarządzania relacyjną bazą danych SQL działającą na bazie architektury

klient-serwer. MySQL zawiera serwer SQL, programy klienckie pozwalające na uzyskanie dostępu do serwera, narzędzia administracyjne i interfejs programistyczny umożliwiający tworzenie własnych programów.

Korzenie MySQL sięgają roku 1979 i narzędzia UNIREG, utworzonego przez Michaela „Monty” Wideniusa dla szwedzkiej firmy TcX. W roku 1994 firma TcX rozpoczęła poszukiwania systemu RDBMS wraz z interfejsem SQL w celu jego użycia w opracowaniu aplikacji sieciowych. Przetestowane przez firmę TcX komercyjne serwery RDBMS okazały się zbyt wolne do obsługi ogromnych tabel TcX, a bezpłatnie dostępny produkt mSQL był pozbawiony funkcji wymaganych przez TcX. Dlatego też Monty przystąpił do prac nad nowym serwerem.

W roku 1995 David Axmark z Detron HB zaczął zachęcać firmę TcX do wydania bazy danych MySQL w internecie. David pracował nad dokumentacją produktu, a także nad przystosowaniem MySQL do kompilacji przy użyciu narzędzi konfiguracyjnych GNU. Baza danych MySQL 3.11.1 została wydana w roku 1996 w postaci dystrybucji binarnej dla systemów Linux i Solaris. Jednocześnie powstała firma MySQL AB w celu dostarczenia dystrybucji MySQL oraz związanych z nią usług komercyjnych. W roku 2008 firma Sun Microsystems przejęła MySQL AB, natomiast w roku 2010 sama została przejęta przez Oracle. Obecnie baza danych MySQL jest dostępna w postaci binarnej oraz kodu źródłowego i działa na wielu różnych platformach.

Początkowo ogromna popularność bazy danych MySQL wzięła się z powodu szybkości jej działania i prostoty. Nie brakowało także słów krytyki, głównie z powodu braku funkcji takich jak transakcje i obsługa kluczy zewnętrznych. Prace nad bazą danych MySQL były kontynuowane i dodano nie tylko wymienione funkcje, ale także i inne, takie jak replikacja, podzapytania, procedury składowane, wyzwalacze i widoki.

Dzięki wymienionym funkcjom baza danych MySQL zaczęła zaliczać się do sektora aplikacji przemysłowych. Na skutek tego awansu użytkownicy, którzy wcześniej stawiali jedynie na „wielkie nazwy” w kategorii rozwiązań systemów baz danych, zaczęli również rozważać użycie oprogramowania MySQL, które działa w szerokiej gamie sprzętu, począwszy od każdego w miarę nowoczesnego, aż po serwery przemysłowe. Oferowana przez MySQL wydajność może rywalizować z każdym systemem bazodanowym, który działa we wspomnianym sprzęcie, a ponadto MySQL bez problemów obsługuje ogromne bazy danych, zawierające miliardy rekordów. W świecie biznesu liczba instalacji MySQL nieustannie rośnie, ponieważ kolejne firmy przekonują się, że MySQL w pełni zaspokaja ich wymagania w zakresie obsługi baz danych, a koszt licencji komercyjnej i pomocy technicznej wynosi ułamek kwot, które dotąd płaciły za inne rozwiązania.

Podsumowując, bezpłatnie dostępne systemy operacyjne działające w oferującym potężne funkcje, choć nadal tanim sprzęcie komputerowym oddają ogromną moc obliczeniową i możliwości w ręce biznesmenów i zwykłych użytkowników w znacznie większym stopniu niż kiedykolwiek wcześniej. To znacznie obniża bariery ekonomiczne i pozwala, aby potężne i wydajne systemy RDBMS trafiły do dużo większej liczby firm niż wcześniej i na dodatek w znacznie niższych cenach. To samo dotyczy osób fizycznych.

Na przykład, osobiście używam bazy danych MySQL wraz z językami Perl i PHP oraz serwerem Apache w laptopie firmy Apple działającym pod kontrolą systemu operacyjnego OS X. W ten sposób moją pracę mogę zabrać wszędzie ze sobą. Całkowity koszt rozwiązania to koszt zakupu laptopa.

Dlaczego MySQL?

Na rynku dostępnych jest wiele tanich lub nawet bezpłatnych systemów zarządzania bazami danych, na przykład MySQL, PostgreSQL i SQLite. Podczas porównywania MySQL z innymi systemami baz danych pod uwagę musisz wziąć czynniki najważniejsze dla Ciebie. Wydajność, funkcje (na przykład zgodność z SQL lub rozszerzenia), pomoc techniczna, warunki licencji i cena — to wszystko trzeba brać pod uwagę. Uwzględniając wymienione czynniki, baza danych MySQL charakteryzuje się wieloma atrakcyjnymi cechami.

- **Szybkość.** MySQL to szybka baza danych, która nieustannie staje się coraz szybsza; spójrz na stronę <http://www.mysql.com/why-mysql/benchmarks/>. W ostatnich wydaniach wprowadzono wiele istotnych usprawnień, przede wszystkim w InnoDB (który obecnie jest domyślnym silnikiem bazy danych) oraz w optymalizatorze zapytań.
- **Łatwość użycia.** Baza danych MySQL oferuje wysoką wydajność, ale jednocześnie to stosunkowo prosty system bazy danych i mniej skomplikowany w konfiguracji od większych, konkurencyjnych systemów.
- **Obsługa języka zapytań.** MySQL obsługuje SQL (ang. *Structured Query Language*), czyli standardowy język zapytań stosowany przez wszystkie nowoczesne systemy baz danych.
- **Możliwości.** Serwer MySQL jest wielowątkowy, a więc potrafi obsługiwać wiele klientów w tym samym czasie. Każdy klient może jednocześnie używać wielu baz danych. Dostęp do MySQL można uzyskać w sposób interaktywny przy użyciu interfejsu pozwalającego na wykonywanie zapytań i wyświetlanie wyników: klientów wiersza poleceń, przeglądarek internetowych lub klientów graficznych. Poza tym dostępne są interfejsy programowania dla wielu języków, między innymi C, Perl, Java, PHP, Python i Ruby. Istnieje również możliwość uzyskania dostępu do MySQL przy użyciu aplikacji obsługujących ODBC i .NET (protokoły opracowane przez Microsoft). W ten sposób masz wybór: użycie przygotowanych aplikacji klienckich bądź utworzenie własnych.
- **Połączenia i bezpieczeństwo.** MySQL to w pełni sieciowa baza danych, do której dostęp można uzyskać z dowolnego miejsca w internecie, a więc dane mogą być współdzielone z każdym i wszędzie. Jednak MySQL posiada mechanizm kontroli dostępu i jeśli zachodzi potrzeba, to użytkownik nie będzie miał dostępu do danych innych użytkowników. Aby zapewnić dodatkową ochronę, MySQL obsługuje połączenia szyfrowane przy użyciu protokołu SSL (ang. *Secure Sockets Layer*).

- **Przenośność.** MySQL działa w wielu różnych systemach UNIX i Linux, a także innych, na przykład Windows. Bazę danych MySQL można uruchamiać w szerokiej gamie sprzętu komputerowego, począwszy od małych urządzeń, takich jak routery i komputery osobiste, aż po wysokiej klasy serwery wyposażone w wiele procesorów i ogromne ilości pamięci.
- **Dostępność i koszt.** MySQL to projekt typu open source rozprowadzany na wielu licencjach. Przede wszystkim jest dostępny na licencji GNU GPL, co oznacza, że MySQL można używać bezpłatnie. Ponadto, dla firm preferujących formalne licencje lub tych, które chcą wykroczyć poza ograniczenia nakładane przez GPL, dostępne są licencje komercyjne.
- **Otwarta dystrybucja i dostępny kod źródłowy.** Bazę danych MySQL można bardzo łatwo uzyskać; wystarczy w tym celu użyć przeglądarki internetowej. Jeżeli nie rozumiesz sposobu działania komponentu, jesteś ciekaw konkretnego algorytmu lub chcesz przeprowadzić audyt bezpieczeństwa, wtedy możesz po prostu pobrać kod źródłowy MySQL i przeanalizować go. Jeśli sądzisz, że znalazłeś błąd w oprogramowaniu MySQL, proszę, zgłoś go — twórcy bazy danych powinni być informowani o znalezionych błędach.

Co z pomocą techniczną? Dobre pytanie — system bazy danych nie będzie użyteczny, jeśli nie ma możliwości uzyskania pomocy dotyczącej jego działania. Niniejsza książka to jedna z form pomocy i mam nadzieję, że uznasz ją za przydatną. (To jest już piąte wydanie książki, co oznacza, że spełnia ona swoją rolę.) Do dyspozycji masz również inne zasoby i przekonasz się o istnieniu dobrej pomocy technicznej dla MySQL.

- Podręcznik użytkownika jest dostarczany wraz z dystrybucją MySQL, a ponadto jest łatwo dostępny w internecie. Ten podręcznik jest regularnie dobrze oceniany przez użytkowników. To bardzo ważne, bo wartość dobrego produktu pozostaje niedoceniona, jeśli nikt nie potrafi z niego korzystać.
- Firma Oracle oferuje komercyjną pomoc techniczną oraz zasoby edukacyjne takie jak szkolenia.
- Listy dyskusyjne i fora poświęcone MySQL to nieocenione źródła pomocy technicznej dla każdego. Na wymienionych listach i forach można znaleźć wielu pomocnych użytkowników, między innymi programistów zaangażowanych w pracę nad MySQL.

Spółeczność MySQL, programiści i zwykli użytkownicy są niezwykle pomocni. Odpowiedzi na pytania zadawane na listach dyskusyjnych bardzo często pojawiają się w ciągu kilku minut od chwili zadania pytania. Po zgłoszeniu błędów programiści dość szybko przygotowują odpowiednie poprawki, a nowe wydania bazy danych pojawiają się regularnie.

Jeżeli właśnie szukasz systemu baz danych, MySQL to produkt, który warto wziąć pod uwagę. Możesz ją wypróbować bez żadnego ryzyka lub związanych z tym kosztów.

Czas konieczny na przeprowadzenie instalacji i konfiguracji MySQL jest znacznie krótszy niż w przypadku wielu innych systemów. Jeśli na jakimkolwiek etapie zabrniesz w ślepy zaułek, pomoc możesz otrzymać na listach dyskusyjnych.

Obecnie prawdopodobnie korzystasz z innego systemu baz danych, ale jesteś świadomy istnienia następujących ograniczeń: wydajność systemu pozostaje niezadowolająca, rozwiązanie jest własnościowe, a Ty nie chcesz być zamknięty w tym rozwiązaniu, Twój obecny sprzęt komputerowy nie jest obsługiwany przez używany system, oprogramowanie jest dostarczane jedynie w postaci binarnej, a Ty chciałbyś otrzymać również kod źródłowy lub po prostu koszt używania rozwiązania jest zbyt duży. Wszystkie wymienione powody są wystarczające, aby spojrzeć w kierunku MySQL. Wykorzystaj tę książkę do poznania możliwości oferowanych przez MySQL, skontaktuj się z działem sprzedaży MySQL, zadawaj pytania na listach dyskusyjnych, wyszukuj odpowiedzi na nurtujące Cię pytania i podejmij decyzję.

Czego możesz się spodziewać po tej książce?

Dowiesz się, jak efektywnie używać bazy danych MySQL, aby zachować maksymalną produktywność w trakcie wykonywania zadań. Przekonasz się, jak umieszczać informacje w bazie danych oraz jak je stamtąd pobierać przy użyciu zapytań odpowiadających na pytania, które chciałbyś zadać danym.

Nie trzeba być programistą, by rozumieć SQL lub go używać. Dzięki tej książce dowiesz się, jak SQL działa. Jednak prawidłowe używanie systemu baz danych to znacznie więcej niż znajomość składni języka SQL. W niniejszej książce położono nacisk na unikalne możliwości MySQL i pokazano, jak je wykorzystać.

Dowiesz się również, jak MySQL zintegrować z innymi narzędziami. Poznasz więc sposoby tworzenia własnych programów uzyskujących dostęp do bazy danych MySQL, a także nauczysz się używania MySQL wraz z językami Perl i PHP w celu wygenerowania dynamicznych stron internetowych na podstawie danych otrzymanych w wyniku zapytań do bazy danych.

Jeżeli będziesz odpowiedzialny za administrację bazą danych MySQL, materiał przedstawiony w książce pokaże, jakie są Twoje obowiązki i jak możesz je wykonywać. Dowiedz się, jak tworzyć konta użytkowników i kopie zapasowe baz danych, konfigurować replikację i zapewnić bezpieczeństwo witrynie internetowej.

Mapa drogowa książki

Ta książka została podzielona na cztery części. Pierwsza koncentruje się na ogólnych koncepcjach użycia bazy danych. Druga jest poświęcona tworzeniu własnych programów wykorzystujących MySQL. Z kolei część III jest przeznaczona dla czytelników zajmujących się administracją MySQL. Część IV książki zawiera dodatki.

Część I. Ogólne użycie MySQL

- Rozdział 1. „Rozpoczęcie pracy z MySQL”. Tutaj dowiesz się, jak baza danych MySQL może być użyteczna dla Ciebie. Ponadto przedstawiono przewodnik wprowadzający do interaktywnego programu klienta mysql, omówiono podstawy języka SQL i zaprezentowano ogólne możliwości MySQL.
- Rozdział 2. „Użycie MySQL do zarządzania danymi”. Każdy ważny system RDBMS dostępny obecnie na rynku oferuje obsługę języka SQL, ale poszczególne silniki baz danych w nieco odmienny sposób implementują różne dialekty SQL. W tym rozdziale omówiono SQL ze szczególnym naciskiem na funkcje, które wyróżniają MySQL spośród innych systemów.
- Rozdział 3. „Typy danych”. W tym rozdziale przedstawiono typy danych oferowane przez MySQL do przechowywania informacji oraz omówiono właściwości i ograniczenia poszczególnych typów. Ponadto dowiesz się, kiedy i jak używać typów danych, jak obliczać wyrażenia oraz przeprowadzać konwersje typów.
- Rozdział 4. „Widoki i programy składowane”. W rozdziale poruszono temat tworzenia i używania obiektów SQL przechowywanych po stronie serwera. To obejmuje widoki (tabele wirtualne) oraz programy składowane (funkcje i procedury, wyzwalacze oraz zdarzenia).
- Rozdział 5. „Optymalizacja zapytań”. Tutaj dowiesz się, jak zoptymalizować zapytania, aby zwiększyć szybkość ich wykonywania.

Część II. Użycie interfejsu programowania MySQL

- Rozdział 6. „Wprowadzenie do programowania MySQL”. W rozdziale omówiono interfejsy programowania aplikacji (API) dla bazy danych MySQL, a także przedstawiono ogólne porównanie API szczegółowo zaprezentowanych w tej książce.
- Rozdział 7. „Tworzenie programów MySQL przy użyciu języka C”. W tym rozdziale dowiesz się, jak tworzyć programy w języku C przy użyciu API dostarczanego przez bibliotekę klienta MySQL.
- Rozdział 8. „Tworzenie programów MySQL przy użyciu Perl DBI”. Omówiono temat tworzenia skryptów Perl przy użyciu modułu DBI. Zaprezentowane zostały skrypty zarówno wiersza poleceń, jak i stosowane w programowaniu witryn internetowych.
- Rozdział 9. „Tworzenie programów MySQL przy użyciu języka PHP”. W rozdziale dowiesz się, jak wykorzystać język skryptowy PHP i umożliwiające dostęp do bazy danych rozszerzenie PDO (PHP Data Objects) w celu tworzenia dynamicznych stron internetowych opartych na bazach danych MySQL.

Część III. Administracja MySQL

- Rozdział 10. „Wprowadzenie do administracji bazą danych MySQL”. Ten rozdział zawiera ogólny opis obowiązków administratora baz danych, a także dostarcza wiedzę niezbędną do zapewnienia prawidłowego działania witryny opartej na MySQL.
- Rozdział 11. „Katalog danych w MySQL”. Tutaj znajdziesz szczegółowe omówienie organizacji i zawartości katalogu danych, w którym MySQL przechowuje bazy danych, pliki dzienników zdarzeń oraz pliki z informacjami o stanie.
- Rozdział 12. „Ogólna administracja bazą danych MySQL”. W tym rozdziale dowiesz się, jak zagwarantować, aby używany system operacyjny prawidłowo uruchamiał i zatrzymywał serwer MySQL podczas uruchamiania i wyłączania systemu operacyjnego. Ponadto poruszono tematy konfiguracji silników baz danych, dostrajania serwera, obsługi dzienników zdarzeń oraz pracy z wieloma serwerami.
- Rozdział 13. „Bezpieczeństwo i kontrola dostępu”. Wszystko, co powinienś wiedzieć, aby chronić instalację MySQL przed intruzami, zarówno innymi użytkownikami serwera, jak również klientami nawiązującymi połączenia z sieci. Omówiono konfigurację kont użytkowników w MySQL, wyjaśniono strukturę tabel uprawnień kontrolujących uprawnienia dostępu klientów serwera MySQL. Ponadto poruszono temat konfiguracji serwera w celu zapewnienia obsługi bezpiecznych połączeń przez SSL.
- Rozdział 14. „Obsługa bazy danych, kopie zapasowe i replikacja”. W rozdziale dowiesz się, jak dzięki profilaktyce zmniejszyć ryzyko katastrofy. Omówione będą zagadnienia obsługi bazy danych, tworzenia kopii zapasowych, przywracania danych po awarii oraz konfiguracji serwerów replikacji.

Dodatki

- Dodatek A „Oprogramowanie wymagane do użycia tej książki”. W tym dodatku dowiesz się, skąd pobrać najważniejsze narzędzia i pliki przykładowej bazy danych używanej w książce.
- Dodatek B „Przewodnik po typach danych”. Ten dodatek przedstawia cechy charakterystyczne typów danych w MySQL.
- Dodatek C „Przewodnik po operatorach i funkcjach”. Tutaj znajdziesz omówienie operatorów i funkcji używanych do tworzenia wyrażeń w poleceniach SQL.
- Dodatek D „Przewodnik po zmiennych systemowych, stanu i użytkownika”. Dodatek prezentuje wszystkie zmienne obsługiwane przez serwer MySQL, a także sposoby ich używania we własnych poleceniach SQL.

- Dodatek E „Przewodnik po składni SQL”. W tym dodatku omówiono wszystkie polecenia SQL obsługiwane przez bazę danych MySQL.
- Dodatek F „Przewodnik po programie MySQL”. Tutaj znajdziesz informacje o programach dostarczanych wraz z dystrybucjami MySQL.

Jak czytać tę książkę?

Niezależnie od czytanego fragmentu książki, najlepszym rozwiązaniem jest od razu samodzielne wypróbowanie przykładów przedstawionych w tekście. Oznacza to dwie rzeczy:

1. Jeżeli baza danych MySQL nie jest zainstalowana w Twoim systemie, zainstaluj ją lub poproś kogoś innego o przeprowadzenie jej instalacji.
2. Pobierz pliki wymagane do instalacji przykładowej bazy danych sampdb, która jest używana w książce.

W dodatku A, zatytułowanym „Oprogramowanie wymagane do użycia tej książki”, znajdziesz informacje, jak i gdzie pobrać wszystkie wymagane komponenty.

Jeżeli jesteś początkującym użytkownikiem MySQL lub SQL, to lekturę rozpocznij od rozdziału 1., zatytułowanego „Rozpoczęcie pracy z MySQL”. Znajdziesz w nim przewodnik prezentujący podstawowe koncepcje dotyczące bazy danych MySQL i języka SQL, co ułatwi lekturę pozostałej części książki. Następnie przejdź do kolejnych rozdziałów części I, aby dowiedzieć się, jak opisać dane i nimi zarządzać, by maksymalnie wykorzystać we własnych aplikacjach możliwości oferowane przez MySQL.

Jeśli znasz już choć trochę SQL, to i tak nadal powinieneś przeczytać rozdziały 2., „Użycie MySQL do zarządzania danymi”, i 3., „Typy danych”. Implementacje SQL różnią się między sobą i dlatego warto wiedzieć, co wyróżnia MySQL spośród innych, być może znanych Ci systemów baz danych. Jeżeli masz doświadczenie w pracy z MySQL, ale potrzebujesz dokładniejszych informacji na temat wykonywania poszczególnych zadań, tę książkę potraktuj jako przewodnik i wyszukuj potrzebne informacje. Za szczególnie użyteczne prawdopodobnie uznasz dodatki.

Jeżeli jesteś zainteresowany tworzeniem własnych programów uzyskujących dostęp do baz danych MySQL, to zapoznaj się z rozdziałami poświęconymi API, począwszy od rozdziału 6., zatytułowanego „Wprowadzenie do programowania MySQL”. Aby przygotować oparty na przeglądarce internetowej interfejs ułatwiający dostęp do baz danych lub bazę danych dla witryny internetowej dostarczającej dynamicznie generowaną treść, zapoznaj się z rozdziałami 8., „Tworzenie programów MySQL przy użyciu Perl DBI”, i 9., „Tworzenie programów MySQL przy użyciu języka PHP”.

Gdy jesteś odpowiedzialny za administrację bazą danych MySQL, przeczytaj rozdziały umieszczone w części III książki, począwszy od rozdziału 10., zatytułowanego „Wprowadzenie do administracji bazą danych MySQL”.

Jeśli tylko sprawdzasz bazę danych MySQL i chcesz ją porównać z aktualnie używanym systemem RDBMS, użyteczne mogą być informacje przedstawione w wielu miejscach

książki. Zapoznaj się z rozdziałami poświęconymi składni SQL i typami danych (część I książki), aby porównać MySQL z używaną wersją SQL. Zwróć także uwagę na rozdziały w części II, o ile musisz tworzyć własne aplikacje. Gdy masz być również odpowiedzialny za administrację MySQL, zapoznaj się z rozdziałami w części III. Przedstawione w książce informacje są użyteczne zarówno dla użytkowników nieużywających obecnie żadnej bazy danych, jak również dla porównujących MySQL z innymi systemami baz danych.

Wersje oprogramowania omówionego w książce

W wydaniu pierwszym książki omówiono bazę danych MySQL 3.22 oraz początkowe wersje MySQL 3.23. Wydanie drugie zawierało informacje o MySQL 4.0 i pierwszej wersji MySQL 4.1. Wydanie trzecie było poświęcone MySQL 4.1 i pierwszej wersji MySQL 5.0. Z kolei wydanie czwarte książki prezentowało MySQL 5.0 i początkowe wersje MySQL 5.1.

W obecnym, piątym wydaniu książki jako wersję wyjściową przyjęto MySQL 5.5. Umieszczony tutaj materiał dotyczy więc MySQL 5.5 i wczesnych wersji MySQL 5.6. Większość przedstawionych informacji nadal ma zastosowanie w wersjach wcześniejszych niż 5.5, choć różnice charakterystyczne dla starszych wydań MySQL nie zostały wyraźnie zaznaczone.

Wersja 5.5 osiągnęła status GA (ang. *General Availability*), czyli dostępnej ogólnie i gotowej do stosowania w środowiskach produkcyjnych. W porównaniu do wczesnych wersji 5.5 wprowadzono wiele zmian, więc warto używać najnowszego dostępnego wydania (obecnie to 5.5.32). Warto dodać w tym miejscu, że MySQL 5.6 również jest już dostępna w wersji GA (obecnie to 5.6.12) gotowej do instalacji w środowiskach produkcyjnych.

Informacje dotyczące starszych wersji MySQL można znaleźć na stronie <http://dev.mysql.com/doc/>, na której znajdują się także podręczniki użytkownika dla poszczególnych wersji MySQL.

W trakcie uaktualniania każdego wydania nowym materiałem zawsze mam problem, aby zbyt nie zwiększyć objętości książki. Zdecydowałem się więc na usunięcie pewnego materiału znajdującego się w poprzednich wydaniach. Największą zmianą w nowej wersji MySQL jest użycie InnoDB (a nie MyISAM) jako domyślnego silnika bazy danych. Tym samym w książce większy nacisk położono na InnoDB, a mniejszy na MyISAM. Inne, rzadziej stosowane silniki bazy danych, takie jak FEDERATED i BLACKHOLE, zostały jedynie wspomniane. W wydaniu piątym usunięto informacje dotyczące `libmysql`d (serwer osadzony), `mysqlhotcopy`, `mysampack`, przestrzennych typów danych i funkcji oraz zastąpiono dokładny opis instalacji jedynie ogólnym materiałem. Więcej informacji na temat wymienionych zagadnień można znaleźć w podręczniku użytkownika MySQL.

Warto w tym miejscu zwrócić uwagę na tematy, które nie zostały poruszone w książce:

- Konektory MySQL, dzięki którym można zapewnić dostęp dla klientów Java, ODBC i programów .NET.
- Silnik NDB i klaster MySQL zapewniający przechowywanie danych w pamięci, wysoką dostępność i redundancję. Więcej informacji na ten temat znajduje się w podręczniku użytkownika MySQL.
- Narzędzie MySQL Workbench z graficznym interfejsem użytkownika (GUI), pomagające w używaniu bazy danych MySQL w środowisku graficznym.
- MySQL Enterprise, czyli komercyjna wersja MySQL, oferująca funkcje takie jak MySQL Enterprise Monitor (monitorowanie serwera i narzędzia diagnostyczne) i MySQL Enterprise Backup (tworzenie kopii zapasowej bazy danych w działającym serwerze).

Wymienione produkty i ich dokumentację znajdziesz na stronach <http://www.mysql.com/products/> i <http://dev.mysql.com/doc/>.

W przypadku innych pakietów najważniejszego oprogramowania omówionego w książce praktycznie każda z ostatnich wersji powinna być wystarczająca dla prezentowanych przykładów. W poniższej tabeli wymieniono najnowsze wersje pakietów dostępne w trakcie pisania książki.

Pakiet	Wersja
Moduł Perl DBI	1.623
Moduł Perl DBD::mysql	4.020
PHP	5.4.10
Apache	2.4.3
CGI.pm	3.63

Oprogramowanie omówione w książce można bez problemów znaleźć w internecie. W dodatku A, zatytułowanym „Oprogramowanie wymagane do użycia tej książki”, przedstawiono informacje dotyczące pobierania MySQL, Perl DBI, PHP wraz PDO, Apache i CGI.pm oraz instalacji wymienionych pakietów w systemie. W dodatku A znajdziesz także instrukcje pozwalające na pobranie używanej w przykładach bazy danych sampdb przedstawionej w książce i zawierającej programy opracowane w rozdziałach poświęconych programowaniu MySQL.

Jeżeli używasz systemu Windows, przyjęto założenie, że będzie to Windows w wersji 2000 lub nowszej. Niektóre funkcje omówione w książce, na przykład nazwane potoki i usługi Windows, są niedostępne w starszych wersjach systemu Windows.

Konwencje użyte w książce

W książce zastosowano następujące konwencje typograficzne:

- Czcionka o stałej szerokości znaków wskazuje nazwy komputerów, plików, katalogów, polecenia, opcje i witryny internetowe.
- **Pogrubiona czcionka o stałej szerokości znaków** została użyta w przykładowych poleceniach do wskazania danych, które powinieneś wpisać.
- *Pochylona czcionka o stałej szerokości znaków* została użyta w poleceniach do wskazania tekstu, który powinieneś zastąpić własnymi wartościami.

W przykładach interaktywnych przyjęto założenie, że polecenia będą wprowadzane w oknie terminala lub konsoli. Aby zapewnić kontekst, znak zachęty w przykładzie wskazuje program, z poziomu którego powinno być wykonane polecenie. Na przykład, polecenia SQL wydawane w kliencie mysql są poprzedzone znakiem zachęty `mysql>`. W przypadku poleceń wydawanych z poziomu powłoki (interpretera wiersza poleceń) najczęściej używany jest znak zachęty `%`. Ogólnie rzecz biorąc, to wskazuje polecenia wydawane w systemie Linux/UNIX lub Windows, a sam znak zachęty zależy od konkretnego typu interpretera wiersza poleceń. (Dostęp do interpretera wiersza poleceń masz po zalogowaniu się w systemie Linux/UNIX lub po wydaniu polecenia `cmd.exe` w Windows). Inny znak zachęty to `#`, który wskazuje polecenie wykonywane w systemie Linux/UNIX przez użytkownika z uprawnieniami użytkownika root lub przy użyciu mechanizmu `su` bądź `sudo`, a także `C:\>`, wskazujący polecenie przeznaczone dla systemu Windows.

Przedstawiony poniżej przykład pokazuje polecenie, które powinno być wprowadzone z poziomu interpretera wiersza poleceń. Znak `%` wskazuje znak zachęty (tego znaku nie wpisujesz). Aby wydać polecenie, wpisz tekst, który został przedstawiony **pogrubioną czcionką o stałej szerokości znaków**, a fragment przedstawiony *pochyloną czcionką o stałej szerokości znaków* (*nazwa_użytkownika*) zastąp własną, odpowiednią wartością:

```
% mysql --user=nazwa_użytkownika sampdb
```

W poleceniach SQL słowa kluczowe SQL i nazwy funkcji są pisane wielkimi literami. Z kolei nazwy baz danych, tabel i kolumn są pisane małymi literami.

W opisach składni nawiasy kwadratowe (`[]`) oznaczają informacje opcjonalne. W przypadku alternatyw stosowana jest pionowa kreska (`|`) jako separator poszczególnych elementów. Lista ujęta w nawias kwadratowy `[]` jest opcjonalna i wskazuje na możliwość wyboru elementu z listy. Natomiast lista ujęta w nawias klamrowy `{}` jest obowiązkowa i wskazuje na konieczność wyboru elementu z listy.

Zasoby dodatkowe

Co możesz zrobić, jeżeli masz pytanie, na które odpowiedzi nie znajdziesz w książce? W poniższej tabeli wymieniono użyteczne źródła dokumentacji w postaci witryn internetowych dotyczących oprogramowania, dla którego możesz potrzebować pomocy.

Pakiet	Podstawowa witryna internetowa
MySQL	http://dev.mysql.com/doc/
Perl DBI	http://dbi.perl.org/
PHP	http://www.php.net/
Apache	http://httpd.apache.org/
CGI.pm	http://search.cpan.org/dist/CGI/

W wymienionych witrynach znajdziesz materiały takie jak podręczniki użytkownika, odpowiedzi na najczęściej zadawane pytania (FAQ) i listy dyskusyjne.

- **Podręcznik użytkownika.** Podstawowa dokumentacja dołączona do MySQL to podręcznik użytkownika. Dostępny jest w wielu formatach, także w postaci dokumentów HTML oraz postaci możliwej do pobrania.
- **Dokumentacja w danym narzędziu.** Dokumentację modułu DBI i jego sterownika dla MySQL (DBD::mysql) można wyświetlić bezpośrednio w wierszu poleceń przy użyciu `perl doc`. Spróbuj wydać polecenia `perl doc DBI` i `perl doc DBD::mysql`. W dokumencie poświęconym DBI znajdziesz opis ogólnych koncepcji, natomiast w dokumencie poświęconym sterownikowi dla MySQL omówiono możliwości związane z obsługą bazy danych MySQL.
- **Dokumenty FAQ.** Dla DBI, PHP i Apache zostały przygotowane dokumenty FAQ zawierające odpowiedzi na najczęściej zadawane pytania.
- **Listy dyskusyjne.** Oprogramowaniu omówionemu w tej książce poświęcono wiele list dyskusyjnych. Dobrym pomysłem jest przystąpienie do listy przeznaczonej dla używanych narzędzi. Warto także korzystać z archiwów tych list dyskusyjnych. Kiedy dopiero zaczynasz pracę z danym narzędziem, możesz mieć wiele tych samych pytań, które już wcześniej inni zadali (i otrzymali na nie odpowiedzi). Nie ma więc żadnego powodu do zadawania tych samych pytań, skoro odpowiedzi można szybko odnaleźć w archiwum. Sposób rozpoczęcia subskrypcji listy dyskusyjnej zależy od konkretnej listy. Poniżej przedstawiono tabelę, w której znajdziesz niezbędne informacje.

Pakiet	Informacje o liście dyskusyjnej
MySQL	http://lists.mysql.com/
Perl DBI	http://dbi.perl.org/support/
PHP	http://www.php.net/mailling-lists.php
Apache	http://httpd.apache.org/lists.html

- **Pomocnicze witryny internetowe.** Poza oficjalnymi witrynami internetowymi, dla niektórych z omówionych w książce narzędzi powstały pomocnicze witryny internetowe, zawierające informacje dodatkowe, takie jak przykładowe fragmenty kodu lub artykuły. Adresy tego rodzaju witryn możesz znaleźć w dziale *Links* oficjalnej witryny internetowej.

Rozpoczęcie pracy z MySQL

W tym rozdziale zostanie przedstawione wprowadzenie do systemu zarządzania relacyjną bazą danych MySQL (ang. *Relational Database Management System* — RDBMS) i strukturalnego języka zapytań (ang. *Structured Query Language* — SQL) rozumianego przez MySQL. Omówione będą podstawowe pojęcia i koncepcje, które powinienś zrozumieć. Ponadto zostanie przedstawiona baza danych sampdb używana w przykładach zaprezentowanych w książce. Rozdział zawiera także przewodnik pokazujący, jak używać MySQL w celu utworzenia bazy danych i pracy z nią.

Lekturę książki rozpocznij od tego rozdziału, jeśli dopiero zaczynasz pracę z bazami danych i prawdopodobnie nie jesteś pewien, czy potrzebujesz bazy danych lub czy możesz jej używać. Ten rozdział powinienś także przeczytać, jeśli nic nie wiesz na temat MySQL lub SQL i potrzebujesz wprowadzenia pozwalającego na rozpoczęcie pracy z wymienionymi technologiami. Czytelnicy posiadający doświadczenie w pracy z MySQL lub innymi systemami baz danych mogą po prostu przejrzeć materiał przedstawiony w tym rozdziale. Jednak aby poznać przeznaczenie i zawartość przykładowej bazy danych sampdb używanej w tej książce, każdy powinien zapoznać się z podrozdziałem 1.2, zatytułowanym „Przykładowa baza danych”.

1.1. W jaki sposób MySQL może Ci pomóc?

W tym podrozdziale zostaną przedstawione sytuacje, w których zastosowanie systemu bazy danych MySQL może okazać się użyteczne. Dzięki temu przekonasz się, jakie rodzaje zadań może wykonywać MySQL i jak może Ci pomóc. Jeżeli nie trzeba Cię przekonywać o użyteczności systemu bazy danych — prawdopodobnie doskonale przemyślałeś problem i teraz szukasz sposobu, na jaki MySQL może pomóc Ci go rozwiązać — to od razu możesz przejść do podrozdziału 1.2, zatytułowanego „Przykładowa baza danych”.

System bazy danych to w zasadzie oferujący potężne możliwości sposób zarządzania informacjami. Wspomniane informacje mogą pochodzić z wielu źródeł. To mogą być dane uzyskane w wyniku badań, biznesowe, żądania klientów, sportowe dane statystyczne, raporty sprzedaży, informacje osobiste, informacje dotyczące pracowników, zgłoszenia błędów, oceny uczniów itd.

Potężne możliwości oferowane przez system bazy danych stają się przydatne, gdy ilość informacji, które trzeba uporządkować i którymi trzeba zarządzać, stała się tak duża lub są one na tyle skomplikowane, że ręczna ich obsługa jest uciążliwa. W przypadku ogromnych korporacji przetwarzających miliony transakcji dziennie baza danych jest po prostu koniecznością. Jednak także w pewnych sytuacjach pojedyncza osoba może zarządzać tak dużą ilością informacji, że użycie bazy danych stanie się konieczne. Przeanalizuj wymienione poniżej sytuacje.

- Jesteś zatrudniony w gabinecie dentysty, w którym odpowiadasz za obsługę rekordów dotyczących pacjentów (data wizyty, co zostało zrobione w trakcie wizyty, data kolejnej wizyty, informacje o ubezpieczeniu itd.).
- Ogromna ilość danych zebranych w wyniku badań prowadzonych latami musi być przeanalizowana w celu przygotowania publikacji. Dlatego też na podstawie tych ogromnych ilości danych chcesz wygenerować podsumowanie, a wybrane podzbiory danych chcesz wyodrębnić do dalszej szczegółowej analizy.
- Jesteś nauczycielem, który musi zarządzać informacjami o uczniach i ich postępach w nauce. Za każdym razem, gdy przeprowadzasz test lub sprawdzian, wystawiasz ocenę uczniom. Poszczególne oceny można bardzo łatwo zapisać w papierowym dzienniku, ale późniejsza ich analiza i obliczanie średniej jest żmudne. Raczej chcesz uniknąć sortowania ocen z poszczególnych testów i sprawdzianów w celu sprawdzenia postępów w przyswajaniu wiedzy przez uczniów, a także wolisz uniknąć ręcznego wyliczania średniej ocen wszystkich uczniów. Zliczanie nieobecności uczniów na zajęciach również jest nużącym zadaniem.
- Organizacja, w której pracujesz jako sekretarka, posiada katalog członków. (Rodzaj organizacji nie ma znaczenia, to może być towarzystwo, klub, orkiestra symfoniczna, klub atletyczny itd.). Każdego roku dla członków organizacji drukujesz katalog na podstawie dokumentu procesora tekstów, w którym wprowadzasz wszelkie zmiany dotyczące członków organizacji. Jesteś zmęczona obsługą katalogu w dotychczasowy sposób, ponieważ ogranicza Ci to możliwości działania. Sortowanie wierszy na różne sposoby jest trudne. Nie możesz w łatwy sposób zaznaczyć tylko wybranych informacji o członkach organizacji, na przykład utworzyć listy składającej się z nazwisk i numerów telefonów. Brakuje też łatwej możliwości wyszukiwania członków organizacji, na przykład tych, którzy wkrótce powinni odnowić członkostwo. Gdybyś tylko mogła, to chciałabyś wyeliminować konieczność przeglądania co miesiąc listy i wyszukiwania osób, którym trzeba wysłać przypomnienie o potrzebie przedłużenia członkostwa. Słyszałaś o „skomputeryzowanym biurze”, w którym informacje są przechowywane w postaci elektronicznej, ale nie dostrzegasz żadnych korzyści z jego stosowania. Informacje o członkach organizacji są w postaci elektronicznej, ale — o ironio! — nie są w formacie, który może być łatwo użyty do innego celu *poza* jedynie wydrukowaniem katalogu!

Wymienione sytuacje mogą dotyczyć różnej ilości informacji, od niewielkiej do ogromnej. Ich cechą wspólną jest konieczność ręcznego wykonywania zadań, które znacznie efektywniej można wykonać w przypadku użycia systemu bazy danych.

Jakiej korzyści można się spodziewać po użyciu systemu bazy danych takiego jak MySQL? Tak naprawdę to zależy od konkretnych potrzeb i wymagań, które — jak przedstawiono w powyższych sytuacjach — mogą być zupełnie odmienne. Jednak cechą wspólną zastosowania systemu bazy danych jest możliwość automatycznego wykonywania żmudnych zadań wcześniej ręcznie wykonywanych przez człowieka przeglądającego szafki z dokumentami. Istotnie, baza danych to rodzaj szafki na dokumenty, ale posiadającej wbudowany system zarządzania. Istnieją ważne zalety przechowywania dokumentów w postaci elektronicznej zamiast ręcznego wypełniania papierów. Powróćmy do przedstawionego wcześniej przykładu gabinetu dentystycznego. Poniżej wymieniono kilka sposobów, w jakie MySQL może pomóc w zarządzaniu rekordami pacjentów.

Skrócenie czasu wypełniania dokumentacji medycznej. Nie musisz przeglądać szuflad szafki i zastanawiać się, gdzie umieścić nową kartę pacjenta. Umieszczenie rekordu w odpowiednim miejscu to zadanie, które spada na bazę danych MySQL.

Skrócenie czasu pobierania informacji. Kiedy szukasz informacji, nie musisz ręcznie przeglądać wszystkich kart pacjentów. Aby wysłać pacjentom przypomnienia o wizycie, wystarczy po prostu nakazać bazie danych MySQL pobranie odpowiednich rekordów. To oczywiście zupełnie inna sytuacja niż rozmowa z inną osobą i wydanie polecenia „proszę ustalić pacjentów, którzy nie byli na wizycie już od pół roku”. Zamiast tego wykonujesz w bazie danych zapytanie podobne do poniższego:

```
SELECT last_name, first_name, last_visit FROM patient  
WHERE last_visit < DATE_SUB(CURDATE(), INTERVAL 6 MONTH);
```

To może budzić grozę, jeśli nigdy wcześniej nie widziałeś tego rodzaju polecenia, ale perspektywa otrzymania wyników w ciągu sekundy lub dwóch zamiast poświęcenia godziny na przeglądanie kart pacjentów powinna być wystarczającą zachętą. W każdym razie ten dziwny bełkot nie będzie taki groźny przez zbyt długi okres. Tak naprawdę znaczenie przedstawionego polecenia będziesz doskonale znał jeszcze przed ukończeniem lektury tego rozdziału.

Elastyczna kolejność pobierania informacji. Rekordy nie muszą być pobierane w kolejności ich przechowywania (na przykład według nazwiska pacjenta). MySQL pozwala na otrzymanie rekordów posortowanych w dowolny sposób — według nazwiska, nazwy firmy ubezpieczeniowej, daty ostatniej wizyty itd.

Elastyczny format danych wyjściowych. Po wyszukaniu interesujących Cię rekordów nie ma potrzeby ręcznego kopiowania informacji. Baza danych MySQL generuje listę za użytkownika. Czasami możesz po prostu wydrukować informacje, w innych sytuacjach możesz je wykorzystać w innym programie. Na przykład, po wygenerowaniu listy pacjentów, którzy od dawna nie byli na wizycie kontrolnej, informacje możesz umieścić w procesorze tekstów i wydrukować odpowiednie przypomnienia wysyłane następnie pacjentom. Ewentualnie możesz być zainteresowany jedynie podsumowaniem, na przykład liczbą wybranych rekordów. Nie musisz ich zliczać ręcznie, odpowiednie podsumowanie baza danych MySQL wygeneruje za Ciebie.

Możliwość dostępu do danych jednocześnie przez wielu użytkowników. W przypadku dokumentów papierowych, jeśli dwie osoby jednocześnie chcą przeglądać ten sam dokument, druga musi poczekać, aż pierwsza odłoży dokument na miejsce. MySQL daje możliwość jednoczesnej obsługi wielu użytkowników. Dlatego też w przedstawionym przypadku obaj użytkownicy mogą jednocześnie przeglądać dany dokument.

Zdalny dostęp i elektroniczna transmisja rekordów. Aby móc przeglądać dokumenty papierowe, trzeba znajdować się w miejscu ich przechowywania, chyba że zostanie wykonana ich kopia. W przypadku dokumentów elektronicznych istnieje możliwość uzyskiwania do nich zdalnego dostępu lub ich elektronicznego przekazywania. Jeżeli klinika dentystyczna ma wiele oddziałów, wtedy dostęp do dokumentów można mieć z każdego gabinetu. Nie ma konieczności wysyłania kopii dokumentów kurierem. Gdy inny pracownik kliniki potrzebujący rekordu pacjenta nie używa tej samej bazy danych, z której Ty korzystasz, możesz wybrać wskazane rekordy i przez sieć przesłać ich treść.

Jeżeli już wcześniej używałeś systemu bazy danych, to doskonale znasz wymienione powyżej zalety i być może zastanawiasz się, jak wyjść poza zwykłe aplikacje „zastępujące szafki z dokumentami”. Dobrym przykładem jest tutaj rozwiązanie stosowane przez wiele organizacji, czyli używanie bazy danych w połączeniu z witryną internetową. Przyjmijmy założenie, że Twoja firma ma bazę danych produktów używaną przez personel, gdy klient dzwoni zapytać o dostępność produktu i jego cenę. To jest dość tradycyjny sposób użycia bazy danych. Jednak jeśli firma zdecyduje się na przygotowanie witryny internetowej dla klientów, wtedy może dostarczyć pewnych usług dodatkowych, na przykład funkcję wyszukiwania pozwalającą klientowi na samodzielne sprawdzenie dostępności produktu i jego ceny. W przypadku zaoferowania sklepu internetowego klient nie musi nawet opuszczać domu, aby dokonać zakupu. W ten sposób klienci otrzymują szukane informacje, a baza danych dostarcza je automatycznie, bez konieczności udziału personelu w tym procesie. Klient otrzymuje informacje niemal natychmiast, nie musi czekać na połączenie z handlowcem, wysłuchując w tym czasie szablonowego komunikatu bądź muzyki. Ponadto dostępność produktu może sprawdzić w dowolnej chwili, a nie tylko w godzinach pracy firmy. Dla firmy to pewne oszczędności, ponieważ każdy klient samodzielnie uzyskujący informacje za pomocą wspomnianej witryny internetowej oznacza eliminację jednego połączenia telefonicznego, które musiałoby zostać obsłużone przez pracownika. Witryna internetowa prawdopodobnie sama na siebie zarobi.

Bazę danych można wykorzystać w jeszcze lepszy sposób niż przedstawiony powyżej. Zapytania wyszukiwania wykonywane przez klientów mogą dostarczać informacje nie tylko klientom, ale również samej firmie. Zapytania pokazują, jakie informacje były szukane przez klientów, a wyniki tych zapytań wskazują, czy firma jest w stanie spełnić oczekiwania klientów. Przynajmniej części z szukanych produktów firma nie posiada w ofercie, a tym samym traci potencjalnego klienta i możliwość zarobku. Dlatego dobrym rozwiązaniem jest rejestrowanie informacji dotyczących operacji wyszukiwania wykonywanych przez klientów: czego szukali i czy szukany produkt znajdował się w magazynie firmy. Następnie tego rodzaju informacje można wykorzystać do dopasowania asortymentu i zaoferowania lepszych usług klientom.

W jaki sposób działa baza danych MySQL? Najlepszym sposobem przekonania się o tym jest samodzielne jej przetestowanie, ale do tego potrzeba przykładowej bazy danych.

1.2. Przykładowa baza danych

W tym podrozdziale zostanie omówiona przykładowa baza danych używana w tej książce. Zawiera ona kod źródłowy przykładów, które powinieneś wypróbować podczas poznawania tajników pracy z MySQL, używając dwóch przedstawionych poniżej sytuacji.

- Sekretarka w organizacji. Przykładowa organizacja charakteryzuje się następującymi cechami: członkowie organizacji interesują się historią USA (przyjęli nazwę „Liga Historyczna USA”). Członkowie muszą odnawiać członkostwo co pewien okres i wносить wówczas odpowiednią opłatę. Wpłaty członków Ligi są przeznaczone na pokrycie jej wydatków, takich jak publikacja gazetki „Kronika Historii USA”. Liga dysponuje również małą witryną internetową, ale to ulegnie zmianie.
- Nauczyciel. Jesteś nauczycielem odpowiedzialnym za przeprowadzanie sprawdzianów i testów, rejestrację wyników uzyskanych przez uczniów, a także wystawianie ocen. Na koniec roku szkolnego wystawiasz oceny końcowe, które wraz z podsumowaniem informacji o poszczególnych uczniach zdajesz do sekretariatu szkoły.

Teraz obie wymienione sytuacje przeanalizujemy znacznie dokładniej pod kątem dwóch wymagań.

- Musisz zdecydować, jakie informacje chcesz pobierać z bazy danych, czyli określić cele, które chcesz osiągnąć.
- Musisz zdecydować, jakie informacje chcesz umieszczać w bazie danych, czyli określić dane, które chcesz zachować.

Prawdopodobnie dziwnie wygląda próba ustalenia danych pobieranych z bazy danych, a dopiero później umieszczanych w niej. W końcu dane trzeba najpierw umieścić w bazie danych, aby można było je stamtąd pobrać. Jednak sposób użycia bazy danych jest podyktowany celami, które z kolei są ściśle powiązane z danymi pobieranymi z bazy danych, a nie umieszczanymi w niej. Nie ma sensu marnować czasu i wysiłku na umieszczanie informacji w bazie danych, o ile nie zamierzasz ich później wykorzystać.

1.2.1. Projekt „Liga Historyczna USA”

W tym scenariuszu jesteś sekretarką, która listę członków organizacji obecnie posiada w dokumencie procesora tekstów. Takie rozwiązanie sprawdza się jedynie w przypadku drukowania listy, poza tym ogranicza możliwości, na jakie można przetwarzać informacje. Musisz pamiętać o kilku celach, które chcesz osiągnąć.

- Wygenerowanie w różnych formatach katalogu członków Ligi, używając do tego informacji odpowiednich dla aplikacji. Jednym z celów jest wygenerowanie co roku katalogu gotowego do wydruku — to wymóg organizacji ustalony w przeszłości i nadal konieczny do spełnienia. Możesz zastanowić się nad innym sposobem wykorzystania informacji z katalogu, na przykład umieszczeniem bieżącej listy

członków Ligi w drukowanym programie, który następnie jest rozdawany wszystkim uczestnikom dorocznego spotkania Ligi. Wspomniane aplikacje pobierają różne zestawy informacji. W katalogu drukowanym umieszczana jest cała zawartość wpisu poświęconego poszczególnym członkom Ligi. W przypadku programu spotkania trzeba pobrać jedynie imiona i nazwiska członków Ligi (to nie będzie łatwe zadanie przy użyciu jedynie procesora tekstu).

- Konieczność przeszukiwania katalogu członków Ligi przy użyciu różnorodnych kryteriów. Na przykład, chcesz dowiedzieć się, którzy członkowie Ligi muszą wkrótce odnowić członkostwo w niej. Inna operacja wyszukiwania może generować listę słów kluczowych stosowanych przez poszczególnych członków Ligi. Wspomniane słowa kluczowe opisują epoki w historii USA szczególnie interesujące danego członka Ligi (na przykład „wojna domowa”, „Wielki Kryzys”, „prawa obywatelskie”, „życie Thomasa Jeffersona” itd.). Członkowie Ligi mogą czasami prosić Cię o podanie listy innych osób o podobnych zainteresowaniach, a dzięki bazie danych i wspomnianej operacji wyszukiwania będziesz mogła spełnić to życzenie.
- Katalog Ligi ma zostać umieszczony w jej witrynie internetowej. Korzyści z takiego kroku odniosą zarówno członkowie Ligi, jak i Ty. Jeżeli katalog będzie można skonwertować na strony internetowe dzięki zastosowaniu rozsądnie zautomatyzowanego procesu, to wersja internetowa katalogu zawsze będzie aktualna, co nie zawsze jest prawdą w przypadku katalogu drukowanego. Jeśli na dodatek katalog w internecie będzie oferował funkcję wyszukiwania, członkowie Ligi łatwo będą mogli samodzielnie wyszukiwać interesujące ich informacje. Na przykład, jeśli członek interesujący się wojną domową będzie chciał znaleźć innych członków Ligi o podobnych zainteresowaniach, wówczas będzie mógł samodzielnie przeprowadzić wyszukiwanie i nie zlecać Ci wykonania tego zadania. Ty z kolei nie będziesz musiała poświęcać czasu na samodzielne przeprowadzanie tego rodzaju operacji.

Bazy danych na pewno nie należą do najbardziej ekscytujących rzeczy na świecie i nie zamierzam przekonywać Cię, że ich używanie stymuluje kreatywne myślenie. Kiedy przestaniesz traktować informacje jako coś, z czym musisz się zmagać (na przykład jak w trakcie używania procesora tekstów), a potraktujesz je jako coś, czym można względnie łatwo operować (przecież takie masz nadzieje względem MySQL), to efektem będzie możliwość użycia informacji na wiele nowych sposobów.

- Jeżeli informacje przechowywane w bazie danych mogą być przeniesione do witryny internetowej w formie katalogu internetowego, oznacza to także możliwość wykorzystania ich na wiele innych sposobów. Przyjmijmy założenie, że członek Ligi będzie mógł samodzielnie edytować informacje, uaktualniając tym samym bazę danych. W ten sposób sekretarce odpadnie konieczność samodzielnej edycji danych, a informacje w katalogu będą znacznie dokładniejsze i aktualniejsze.

- Jeżeli w bazie danych są przechowywane adresy e-mail, można je wykorzystać w celu wysłania wiadomości e-mail tym członkom Ligi, którzy od dłuższego czasu nie uaktualniali informacji o sobie. Wspomniana wiadomość może pokazywać członkowi Ligi aktualne informacje o nim, zawierać prośbę o ich przejrzenie oraz wskazówki, jak wprowadzić wszelkie wymagane modyfikacje w celu wykorzystania możliwości oferowanych przez witrynę internetową.
- Baza danych może znacznie zwiększyć użyteczność witryny internetowej na sposoby nawet niezwiązane z listą członków Ligi. W każdym wydaniu gazetki Ligi, zatytułowanej „Kronika Historii USA”, znajduje się sekcja z quizem historycznym. We wcześniejszych wydaniach przedstawiono sporo faktów dotyczących prezydentów USA. W witrynie internetowej również można umieścić wspomnianą sekcję wraz z quizami. Tego rodzaju sekcja może być interaktywna dzięki umieszczeniu w bazie danych informacji wykorzystywanych w quizach. Następnie serwer WWW będzie pobierał z bazy danych odpowiednie pytania i wyświetlał je użytkownikom.

Doskonale! Na tym etapie liczba potencjalnych zastosowań bazy danych jest na tyle duża, że być może dałeś się zbyt ponieść. Po ochłonięciu pora wrócić na ziemię i zadać kilka praktycznych pytań.

- Czy ten projekt nie jest zbyt ambitny? Czy jego przygotowanie wiąże się z koniecznością włożenia ogromnej ilości pracy?
Wszystko wydaje się łatwiejsze, gdy tylko o tym myślisz, ale nic z tym nie robisz. Oczywiście jest, że idee nie są łatwe do implementacji. Jednak ta książka powstała po to, aby przedstawione idee wcielić w życie. Pamiętaj, że nie ma konieczności zajmowania się wszystkim naraz. Podziel całe zadanie na mniejsze fragmenty, a następnie po kolei je wykonuj.
- Czy bazę danych MySQL można wykorzystać do osiągnięcia wymienionych celów?
Nie, przynajmniej nie samodzielnie. Na przykład, baza danych MySQL nie posiada wbudowanych możliwości w zakresie tworzenia aplikacji sieciowych. Jednak dzięki połączeniu MySQL z innymi narzędziami można uzupełnić i rozszerzyć jej możliwości.

W tej książce wykorzystamy język skryptowy Perl i moduł interfejsu bazy danych Perl DBI w celu przygotowania skryptów, które mają dostęp do baz danych MySQL. Perl oferuje doskonałe możliwości w zakresie przetwarzania tekstu, co pozwala na bardzo elastyczne operowanie zapytaniami i generowanie danych wyjściowych w różnych formatach. Na przykład, języka Perl można użyć do wygenerowania katalogu w formacie RTF (ang. *Rich Text Format*), który jest odczytywany przez wiele rodzajów procesorów tekstów, lub w formacie HTML, obsługiwany przez przeglądarki internetowe.

Wykorzystamy także PHP, czyli inny język skryptowy. Język PHP został przystosowany do tworzenia aplikacji sieciowych i bardzo łatwo współpracuje

z bazami danych. Daje możliwość inicjowania zapytań MySQL z poziomu stron internetowych oraz generowania nowych stron zawierających wyniki zapytań do bazy danych. PHP może być stosowany w wielu serwerach WWW (między innymi w Apache, czyli najpopularniejszym serwerze WWW na świecie), co znacznie ułatwia wykonywanie zadań takich jak wyświetlanie formularza wyszukiwania lub wyników wyszukiwania.

MySQL bardzo łatwo integruje się z wymienionymi narzędziami, daje elastyczność w zakresie łączenia technologii w celu osiągnięcia żądanych celów. Nie jesteś zmuszony do stosowania rozwiązań typu „wszystko w jednym”, których komponenty są reklamowane jako zapewniające wysoką integrację, ale tak naprawdę doskonale sprawdzają się tylko ze sobą.

- I wreszcie najważniejsze pytanie: ile to będzie kosztowało? W końcu budżet Ligi jest ograniczony.

Odpowiedź może Cię bardzo zaskoczyć, ponieważ prawdopodobnie nie poniesiesz żadnych dodatkowych kosztów. Jeżeli znasz inne systemy baz danych, to zapewne wiesz, że ogólnie rzecz biorąc, to są kosztowne rozwiązania. Natomiast MySQL bardzo często może być używana bezpłatnie. Nawet w środowiskach przemysłowych, wymagających umów gwarantujących pomoc techniczną i obsługę, koszt zastosowania systemu bazy danych MySQL będzie stosunkowo niewielki.

(Informacje szczegółowe na ten temat znajdziesz na witrynie

<http://www.mysql.com/>). Inne wykorzystywane narzędzia (Perl, DBI, PHP,

Apache) są bezpłatne. Uwzględniając wszystkie komponenty, całkowity koszt przygotowania działającego systemu będzie względnie niski.

Wybór systemu operacyjnego, w którym będzie uruchomiona baza danych, zależy od Ciebie. Praktycznie całe omawiane w książce oprogramowanie działa w systemie UNIX (tym ogólnym pojęciem określam systemy BSD UNIX, Linux, OS X itd.) i Windows. Wyjątkiem są skrypty powłoki lub wiersza poleceń przeznaczone specjalnie dla systemu UNIX lub Windows.

1.2.2. Projekt „Oceny uczniów”

Przeanalizujmy teraz inną sytuację, w której użyjemy przykładowej bazy danych. W tym scenariuszu jesteś nauczycielem, do obowiązków którego należy rejestracja informacji o poczynionych przez uczniów postępach w nauce. Proces oceniania uczniów chcesz zmienić z operacji całkowicie ręcznej na elektroniczną, opierając się na bazie danych MySQL. W takim przypadku informacje pobierasz z bazy danych w dokładnie taki sam sposób, jak obecnie używasz dziennika papierowego.

- Rejestrujesz wynik każdego sprawdzianu i testu. W przypadku testów oceny są umieszczane w kolejności, co pozwala na ich szybkie sprawdzenie i określenie granic poszczególnych ocen (1, 2, 3, 4, 5 i 6).

- Na końcu semestru obliczasz średnią każdego ucznia, sortujesz średnie i wystawiasz oceny końcowe. Ocena końcowa może być wystawiana przy użyciu wag, ponieważ prawdopodobnie ocenę z testu uznajesz za ważniejszą niż ze sprawdzianu.
- Na końcu każdego semestru wystawione oceny uczniów przekazujesz do sekretariatu szkoły.

Celem jest uniknięcie konieczności ręcznego sortowania ocen i podliczania ogólnych wyników uzyskanych przez uczniów. Innymi słowy, chcesz, aby baza danych MySQL na koniec semestru sortowała oceny końcowe uczniów i zliczała nieobecności. Realizacja wskazanego celu wymaga przygotowania listy uczniów w klasie, wprowadzania otrzymanych przez nich ocen i dat nieobecności na zajęciach.

1.2.3. Jak przykładowa baza danych może Ci pomóc?

Jeżeli nie jesteś szczególnie zainteresowany Ligą Historyczną lub wystawianiem ocen, to być może zastanawiasz się, czy istnieje scenariusz odpowiedni dla Twoich potrzeb. Odpowiedź brzmi: to nie są cele same w sobie. Zamiast tego pokazują, co można osiągnąć przy użyciu bazy danych MySQL i powiązanych z nią narzędzi.

Przy odrobinie wyobraźni możesz przekonać się, jak przykładowe zapytania do bazy danych zastosować w celu rozwiązania innych konkretnych problemów. Przyjmijmy założenie, że pracujesz we wspomnianym wcześniej gabinecie dentystycznym. W tej książce nie znajdziesz wielu zapytań powiązanych z gabinetem dentystycznym, ale za to wiele zapytań, które można wykorzystać do obsługi rekordów pacjentów, planowania wizyt itd. Na przykład, ustalanie członków Ligi, którzy powinni wkrótce odnowić członkostwo, jest podobne do ustalania pacjentów, którzy od dawna nie byli na wizycie kontrolnej. Oba wymienione zapytania są oparte na danych, więc jeśli poznasz zapytanie dotyczące odnawiania członkostwa, nabyte umiejętności będziesz mógł wykorzystać do przygotowania odpowiedniego zapytania wymaganego w innej aplikacji.

1.3. Podstawowa terminologia bazy danych

Prawdopodobnie zauważyłeś, że chociaż to jest książka poświęcona bazie danych, jak dotąd nie użyto w niej zbyt dużo żargonu i terminologii technicznej. Tak naprawdę, nadal jeszcze nie powiedziałem, czym faktycznie jest baza danych, nie przedstawiłem nawet jej ogólnej specyfikacji lub chociaż sposobu użycia przykładowej bazy danych. Jednak zabieramy się za przygotowanie projektu bazy danych, a następnie ją zaimplementujemy, więc nie możemy dłużej unikać stosowania odpowiedniej terminologii. Został jej poświęcony ten podrozdział. Omówiono w nim używane w książce podstawowe pojęcia, które powinienś znać. Na szczęście, wiele koncepcji relacyjnych baz danych jest całkiem prostych. Szczерze mówiąc, atrakcyjność relacyjnych baz danych bardzo często wynika z prostoty ich podstawowych koncepcji.

1.3.1. Terminologia strukturalna

W świecie baz danych MySQL jest klasyfikowana jako system zarządzania relacyjną bazą danych (RDBMS). Odszyfrujmy teraz tę nazwę.

Baza danych (czyli człon DB w RDBMS) to repozytorium, w którym przechowywane są informacje. Wspomniane repozytorium jest zorganizowane w prosty sposób:

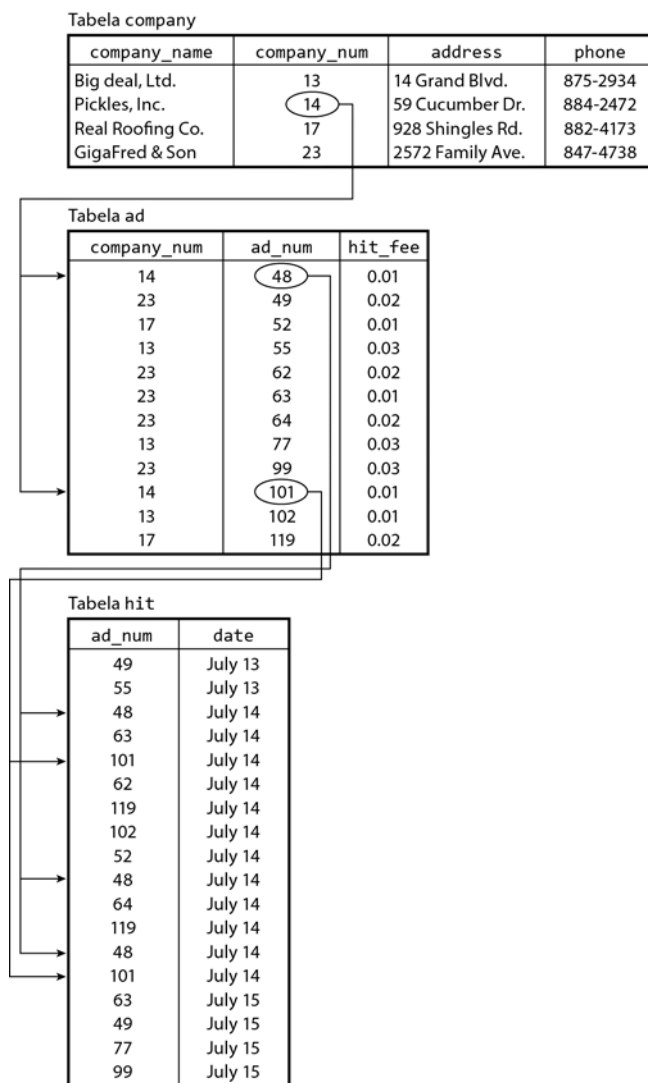
- Kolekcje danych w bazie danych są umieszczane w tabelach.
- Każda tabela składa się z wierszy i kolumn.
- Każdy wiersz tabeli jest rekordem.
- Rekordy mogą zawierać wiele różnych fragmentów informacji, natomiast każda kolumna w tabeli odpowiada jednemu fragmentowi informacji.

System zarządzania (człon MS) to oprogramowanie pozwalające na użycie danych dzięki możliwości przeprowadzania operacji wstawiania, modyfikacji lub usuwania rekordów.

Słowo „relacyjny” (człon R) wskazuje na określony rodzaj DBMS; w omawianym przypadku to rodzaj doskonale sprawdzający się w dopasowywaniu informacji przechowywanych w jednej tabeli z informacjami przechowywanymi w innej. Dopasowanie odbywa się przez wyszukanie wspólnych elementów w analizowanych tabelach. Potęgą systemu DBMS kryje się w możliwości wygodnego pobierania danych z tabel i łączenia informacji z powiązanych tabel w celu udzielenia odpowiedzi na pytania, na które nie można odpowiedzieć, używając informacji tylko z jednej tabeli. (W rzeczywistości słowo „relacja” ma oficjalną definicję inną od przedstawionej powyżej. Przepraszam purystów, ale uważam, że moja definicja jest znacznie bardziej przydatna podczas prezentacji użyteczności systemu RDBMS).

Poniżej zaprezentowano przykład pokazujący sposób ułożenia danych w relacyjnej bazie danych. Informacje są umieszczone w powiązanych ze sobą tabelach. Przyjmujemy założenie, że masz witrynę internetową wraz z umieszczonymi w niej banerami reklamowymi. Zawarłeś odpowiednie umowy z firmami, które chcą wyświetlać swoje reklamy odwiedzającym Twoją witrynę internetową. Za każdym razem, gdy odwiedzający wyświetli stronę internetową, następuje wyświetlenie osadzonej w niej reklamy, a reklamodawca wnosi za to niewielką opłatę. To nosi nazwę „odsłony” reklamy (ang. *hit*). Aby przedstawić tę informację, trzeba przygotować trzy tabele (patrz rysunek 1.1). Pierwsza tabela, o nazwie *company*, zawiera kolumny przeznaczone do przechowywania nazwy firmy, jej numeru, adresu i numeru telefonu. Druga tabela, o nazwie *ad*, zawiera numery reklam i firm oraz kwotę pobieraną za wyświetlenie reklamy. Z kolei trzecia tabela, o nazwie *hit*, rejestruje każdą odsłonę reklamy (numer reklamy i datę jej wyświetlenia).

Odpowiedzi na pewne pytania można udzielić na podstawie informacji pochodzących z jednej tabeli. Aby podać liczbę firm, z którymi masz umowy na wyświetlanie reklam, wystarczy zliczyć kolumny w tabeli *company*. Podobnie, ustalenie liczby odsłon w trakcie wskazanego okresu wymaga użycia jedynie tabeli *ad*. Inne pytania mogą być znacznie bardziej skomplikowane i konieczne jest wówczas wykorzystanie informacji pochodzących z wielu tabel. Na przykład, w celu ustalenia liczby odsłon reklam firmy Pickles, Inc. w dniu 14 lipca konieczne jest użycie trzech tabel w następujący sposób:



Rysunek 1.1. Tabele służące do obsługi wyświetlania reklam na stronach internetowych

1. Wyszukanie nazwy firmy (Pickles, Inc.) w tabeli company, aby ustalić numer przypisany firmie (14).
2. Użycie numeru firmy w celu znalezienia dopasowanych rekordów w tabeli ad, co pozwala na ustalenie numerów reklam wskazanej firmy. W tabeli ad znaleziono dwie reklamy wymienionej firmy; mają one numery 48 i 101.

3. Dla każdego dopasowanego rekordu w tabeli `ad` należy użyć numeru reklamy w celu odszukania w tabeli `hit` dopasowanych rekordów we wskazanym przedziale czasu, a następnie zliczyć liczbę dopasowań. Istnieją trzy dopasowania reklamy o numerze 48 i dwa dopasowania reklamy 101.

Brzmi to skomplikowanie! To jednak są zadania, podczas wykonywania których systemy relacyjnych baz danych sprawdzają się doskonale. Tak naprawdę, poziom skomplikowania nie jest duży, ponieważ każdy opisany powyżej krok sprowadza się do prostej operacji dopasowania: powiązanie jednej tabeli z drugą poprzez dopasowanie wartości z rekordów jednej tabeli z wartościami w rekordach innej. Ta sama prosta operacja może być wykonana na wiele sposobów w celu udzielenia odpowiedzi na wiele różnych pytań, na przykład: Ile różnych reklam ma dana firma? Która firma jest najpopularniejsza? Jakie zyski generują poszczególne reklamy? Jaka jest całkowita kwota rachunku dla danej firmy za wyświetlanie reklam we wskazanym przedziale czasu?

Teraz powinienś mieć wystarczającą ilość wiedzy teoretycznej o relacyjnej bazie danych, aby zrozumieć pozostałą część tej książki. Nie będziemy się zagłębiać w trzecią postać normalizacyjną, wykresy relacji pomiędzy encjami i pozostałe tego rodzaju rzeczy. (Jeżeli chcesz się zapoznać z wymienionymi zagadnieniami, to sugeruję zapoznanie się z pracami C.J. Date'a i E.F. Codda).

1.3.2. Terminologia języka zapytań

Komunikacja z bazą danych MySQL odbywa się przy użyciu strukturalnego języka zapytań (ang. *Structured Query Language* — SQL). Wszystkie najważniejsze systemy baz danych zawierają obsługę języka SQL, choć poszczególne implementacje stosują pewne odmienne cechy charakterystyczne. SQL obsługuje wiele różnych poleceń, co pozwala na pracę z bazą danych na wiele interesujących i użytecznych sposobów.

Podobnie jak ma to miejsce w każdym języku, także SQL na początku wydaje się dziwny. Na przykład, w celu utworzenia tabeli należy MySQL podać jej strukturę. Ty i ja możemy tabelę traktować w kategoriach wykresu lub obrazu. Z kolei MySQL nie ma takiej możliwości i tabelę trzeba utworzyć, wykonując w serwerze MySQL zapytanie podobne do przedstawionego poniżej:

```
CREATE TABLE company
(
  company_name VARCHAR(30),
  company_num  INT,
  address      VARCHAR(30),
  phone        VARCHAR(12)
);
```

Tego rodzaju zapytania mogą wydawać się trudne, gdy dopiero poznasz SQL, ale tak naprawdę nie trzeba być programistą, aby nauczyć się, jak efektywnie korzystać z języka SQL. Kiedy nieco poznasz język, na zapytanie `CREATE TABLE` spojrzysz zupełnie inaczej — to nie dziwaczne zapytanie, a raczej oferujący potężne możliwości sojusznik pomagający w opisanii informacji.

1.3.3. Terminologia architekuralna MySQL

Kiedy używasz MySQL, to w rzeczywistości korzystasz z przynajmniej dwóch programów, ponieważ działanie MySQL opiera się na architekturze klient-serwer. Jednym ze wspomnianych programów jest serwer MySQL (`mysqld`). Serwer działa w komputerze, w którym przechowywana jest baza danych. Nasłuchuje żądań klientów nadchodzących przez sieć, uzyskuje dostęp do zawartości bazy danych zgodnie z żądaniami i dostarcza klientom informacje, o które prosili. Z kolei drugim programem jest klient; łączy się z serwerem bazy danych i wykonuje zapytania, wskazując informacje, które chce otrzymać z bazy danych.

Większość dystrybucji zawiera serwer bazy danych i kilka programów klienckich. (Jeżeli używasz pakietów RPM w systemie Linux, wtedy serwer i programy klientów znajdują się w oddzielnych pakietach, a więc powinieneś zainstalować oba wspomniane pakiety). Programu klienta używaj zgodnie z celem, który chcesz osiągnąć. Najczęściej wykorzystywany klient to `mysql`, interaktywny program pozwalający na wykonywanie zapytań i wyświetlanie ich wyników. Dwa administracyjne programy klienckie to `mysqldump`, służący do umieszczenia zawartości tabel w pliku, oraz `mysqladmin`, pozwalający na sprawdzenie stanu serwera i wykonanie innych zadań administracyjnych, takich jak wyłączenie serwera. Dystrybucje MySQL zawierają także inne programy klienckie. Jeżeli wymagań aplikacji nie spełniają żadne standardowe programy klienckie, MySQL dostarcza bibliotekę programowania klientów, która umożliwia opracowanie własnych programów klienckich. Wspomniana biblioteka jest możliwa do użycia bezpośrednio z poziomu programów utworzonych w języku C. Jeżeli preferujesz języki inne niż C, to masz dostępne interfejsy do wielu innych języków, między innymi Perl, PHP, Python, Java i Ruby.

Wszystkie programy klienckie zastosowane w tej książce są używane z poziomu wiersza poleceń. Jeżeli chcesz, możesz wypróbować MySQL Workbench, czyli narzędzie zapewniające graficzny interfejs użytkownika (GUI) i możliwość wykonywania operacji myszą (wymienione narzędzie znajdziesz na stronie <http://www.mysql.com/products/workbench/>).

Stosowana przez MySQL architektura klient-serwer ma pewne zalety.

- Serwer wymusza kontrolę współbieżności w celu uniemożliwienia dwóm użytkownikom jednoczesnej modyfikacji tego samego rekordu. Wszystkie żądania klientów przechodzą przez serwer, który decyduje o tym, kto i kiedy uzyska dostęp do danych. Jeżeli wiele klientów chce jednocześnie uzyskać dostęp do tej samej tabeli, nie muszą negocjować między sobą. Po prostu wysyłają żądania do serwera, który ustala kolejność, w jakiej klienty będą mogły wykonać swoje operacje.
- Nie trzeba być zalogowanym w komputerze, w którym znajduje się baza danych. MySQL działa w środowisku sieciowym, więc program klienta można uruchomić w dowolnym komputerze, a następnie przez sieć połączyć się on z serwerem. Odległość nie ma tutaj znaczenia, można uzyskać dostęp do serwera znajdującego się w dowolnym miejscu na świecie. Jeżeli serwerem jest komputer znajdujący się w Australii, to swojego laptopa możesz zabrać na wycieczkę do Islandii i nadal

mieć dostęp do bazy danych. Możesz zapytać: czy to oznacza, że dostęp do danych może mieć każdy, kto po prostu połączy się z internetem? Odpowiedź brzmi: nie. MySQL zawiera elastyczny system bezpieczeństwa, który pozwala na uzyskanie dostępu jedynie upoważnionym użytkownikom. Ponadto można zagwarantować, że użytkownicy będą mogli przeprowadzać tylko operacje wymagane do wykonania ich pracy. Samanta w dziale księgowości powinna mieć możliwość odczytu i uaktualniania (modyfikacji) rekordów, natomiast Filip w dziale pomocy jedynie ich odczytu. Każdemu użytkownikowi można dokładnie zdefiniować uprawnienia. Jeżeli chcesz skonfigurować system działający tylko lokalnie, ustaw uprawnienia w taki sposób, aby klienci mogli nawiązywać połączenie jedynie z komputera, w którym działa serwer bazy danych.

Wskazówka

Różnica między „MySQL” i „mysql”

Aby uniknąć nieporozumień, powinienem wyraźnie powiedzieć, że pojęcie „MySQL” odnosi się do całego systemu RDBMS MySQL, natomiast „mysql” to nazwa jednego z programów klienckich. Wymowa wymienionych nazw jest taka sama, natomiast różnią się wielkością liter w nazwach.

Wracając do wymowy, nazwę MySQL wymawiamy „maj-es-kju-el”. Informacje dotyczące wymowy zostały umieszczone w podręczniku użytkownika MySQL. Z drugiej strony, w zależności od adresata pytania, SQL jest wymawiane jako „es-kju-el” lub „si-kju-el”. W tej książce przyjęto stosowanie wymowy „es-kju-el”.

1.4. Przewodnik po MySQL

Jak dotąd poznałeś niezbędne podstawy. Teraz przystąpimy do pracy z MySQL!

W tym podrozdziale poznasz MySQL dzięki przedstawionemu przewodnikowi wraz z przykładami. Utworzymy przykładową bazę danych wraz z pewnymi tabelami, a następnie będziemy pracować z tą bazą danych, dodawać, pobierać, usuwać i modyfikować informacje w tabelach. W trakcie tego procesu poznasz następujące zagadnienia:

- Podstawy języka SQL zrozumiałego dla MySQL. (Jeżeli znasz już język SQL, ponieważ pracowałeś z innymi systemami RDBMS, wtedy dobrym pomysłem jest przejrzenie przewodnika, aby przekonać się, jak dialekt SQL w bazie danych MySQL różni się od znanego Tobie).
- Komunikacja z serwerem MySQL przy użyciu kilku standardowych programów klienckich. Jak już wspomniano w poprzednim podrozdziale, MySQL działa na zasadzie klient-serwer. W tego rodzaju architekturze serwer działa w komputerze przechowującym bazę danych, natomiast klienci przez sieć łączą się z serwerem. Ten przewodnik jest w dużej mierze oparty na kliencie `mysql`, który odczytuje wykonywane zapytania, wysyła je do serwera w celu wykonania, a następnie wyświetla wyniki wykonania tych zapytań. Klient `mysql` działa na wszystkich platformach obsługiwanych przez MySQL i zapewnia bezpośrednią pracę z serwerem

i dlatego wybór tego klienta wydaje się logiczny. W niektórych przykładach przewodnika wykorzystano programy klienckie `mysqlimport` i `mysqlshow`.

W książce użyto przykładowej bazy danych o nazwie `sampdb`. Jeżeli ktokolwiek w Twoim systemie używa tej nazwy dla bazy danych, wtedy możesz wybrać inną nazwę. Ewentualnie administrator MySQL może przypisać Ci inną nazwę bazy danych. W każdym razie, gdy w przykładach zobaczysz nazwę `sampdb`, zastąp ją rzeczywistą nazwą bazy danych, której używasz.

Nazwy tabel mogą być dokładnie takie same jak w omawianych przykładach, nawet jeśli wielu użytkowników w systemie posiada własne przykładowe bazy danych. W MySQL nie ma znaczenia, czy użytkownicy używają tych samych nazw tabel, o ile są one umieszczone w oddzielnych bazach danych. MySQL nie pozwala na używanie tabel pochodzących z różnych baz danych.

1.4.1. Pobranie przykładowej bazy danych

W kilku miejscach przewodnika znajdziesz odniesienie do plików z „dystrybucji przykładowej bazy danych” (nazywanej także dystrybucją `sampdb` — nazwa zaczerpnięta od nazwy bazy danych `sampdb`). Wspomniane pliki zawierają zapytania i dane pozwalające na przygotowanie przykładowej bazy danych. Procedura opisująca pobranie przykładowej dystrybucji została umieszczona w dodatku A, zatytułowanym „Oprogramowanie wymagane do użycia tej książki”. Po rozpakowaniu dystrybucji następuje utworzenie katalogu `sampdb`, zawierającego potrzebne pliki. Zalecam użycie wspomnianego katalogu jako roboczego podczas pracy z przykładami wykorzystującymi przykładową bazę danych.

Aby ułatwić wywoływanie programów MySQL niezależnie od katalogu roboczego, katalog `bin` MySQL warto dodać do ścieżki wyszukiwania w powłoce. W tym celu ścieżkę dostępu do katalogu dodać do zmiennej środowiskowej `PATH`, stosując zapytania przedstawione w punkcie A.3.3, zatytułowanym „Konfiguracja zmiennej środowiskowej `PATH`”.

1.4.2. Wymagania wstępne

Poza pobraniem dystrybucji przykładowej bazy danych konieczne jest spełnienie jeszcze kilku wymagań wstępnych, aby możliwe było wykonanie przykładów przedstawionych w przewodniku:

- Konieczna jest instalacja oprogramowania MySQL.
- Konieczne jest posiadanie konta w bazie danych MySQL, aby móc nawiązać połączenie z serwerem.
- Potrzebna jest baza danych, z którą będziesz pracować.

Wymagane oprogramowanie obejmuje klienty MySQL i serwer MySQL. Programy klienckie muszą być zainstalowane w komputerze, z którego korzystasz. Natomiast serwer może być umieszczony w Twoim komputerze, choć nie jest to konieczne. O ile masz uprawnienia pozwalające na nawiązanie połączenia z serwerem, to może być on umieszczony

w dowolnym miejscu na świecie. Jeżeli dostawca usług internetowych oferuje MySQL jako usługę, to z bazy danych MySQL możesz korzystać w taki właśnie sposób. Natomiast jeśli chcesz samodzielnie pobrać i zainstalować MySQL, wtedy zapoznaj się z dodatkiem A, zatytułowanym „Oprogramowanie wymagane do użycia tej książki”.

Oprócz oprogramowania MySQL konieczne jest posiadanie konta MySQL w serwerze, a także uprawnień pozwalających na nawiązanie połączenia oraz utworzenie przykładowej bazy danych wraz z tabelami. (Jeśli masz konto w serwerze MySQL, to możesz go użyć, choć równie dobrze możesz utworzyć oddzielne konto dla materiału przedstawionego w tej książce).

Na tym etapie możemy napotkać problem kury i jajka: aby skonfigurować konto MySQL pozwalające na nawiązanie połączenia z serwerem, konieczne jest wcześniejsze połączenie się z serwerem. Zwykle odbywa się to przez połączenie z MySQL jako użytkownik root, a następnie wykonanie zapytań CREATE USER i GRANT w celu utworzenia nowego konta MySQL i nadania mu odpowiednich uprawnień. Jeżeli zainstalowałeś MySQL we własnym komputerze i serwer działa, wtedy możesz się z nim połączyć jako root, a następnie utworzyć nowe konto administratora dla przykładowej bazy danych, podając nazwę użytkownika sampadm i hasło secret, jak w poniższym fragmencie kodu. Nazwę użytkownika i hasło możesz zmienić na własne, a następnie stosować je w przykładach przedstawionych w książce.

```
% mysql -p -u root
Enter password: *****
mysql> CREATE USER 'sampadm'@'localhost' IDENTIFIED BY 'secret';
Query OK, 0 rows affected (0.04 sec)
mysql> GRANT ALL ON sampdb.* TO 'sampadm'@'localhost';
Query OK, 0 rows affected (0.01 sec)
```

Polecenie mysql zawiera opcję -p, powodującą wyświetlenie pytania o hasło użytkownika root bazy danych MySQL. W miejscu gwiazdek (*****) w powyższym przykładzie podaj odpowiednie hasło. Przyjęto tutaj założenie, że hasło użytkownika root bazy danych MySQL zostało zdefiniowane i znasz je. Jeżeli użytkownikowi root nie przypisałeś hasła, to po wyświetleniu pytania Enter password: po prostu naciśnij klawisz *Enter*. Brak zdefiniowanego hasła dla użytkownika root jest jednak błędem, który powinien zostać jak najszybciej naprawiony. Więcej informacji na temat zapytań CREATE USER i GRANT, konfiguracji kont użytkowników MySQL i zmiany haseł znajdziesz w rozdziale 13, zatytułowanym „Bezpieczeństwo i kontrola dostępu”.

Po utworzeniu konta sampadm wydaj polecenie quit i naciśnij klawisz *Enter*, aby w ten sposób zakończyć działanie programu mysql.

Przedstawione powyżej polecenia są odpowiednie do nawiązania połączenia z MySQL z poziomu komputera, w którym został uruchomiony serwer. Zyskujesz możliwość nawiązania połączenia z serwerem, podając nazwę użytkownika sampadm i hasło secret. Dają one pełny dostęp do bazy danych sampdb. Jednak zapytanie GRANT nie powoduje utworzenia bazy danych (możesz zdefiniować uprawnienia do bazy danych, nawet jeśli ona jeszcze nie istnieje). Tworzeniem bazy danych zajmiemy się już wkrótce.

Jeżeli planujesz nawiązanie połączenia z serwerem MySQL z poziomu innego komputera niż ten, w którym został uruchomiony serwer, wówczas zmień parametr

localhost na nazwę używanego przez Ciebie komputera. Na przykład, jeśli nawiązujesz połączenie z serwerem z poziomu komputera `boa.example.com`, to odpowiednie zapytania przedstawiają się następująco:

```
mysql> CREATE USER 'sampadm'@'boa.example.com' IDENTIFIED BY 'secret';
mysql> GRANT ALL ON sampdb.* TO 'sampadm'@'boa.example.com';
```

Jeżeli nie masz kontroli nad serwerem i nie możesz tworzyć kont, poproś administratora MySQL o utworzenie odpowiedniego konta. Następnie nazwę użytkownika, hasło i nazwę bazy danych (odpowiednio `sampadm`, `secret` i `sampdb`) zastąp tymi, które otrzymasz od administratora, i stosuj je w przykładach przedstawionych w książce.

1.4.3. Nawiązywanie i zrywanie połączenia z serwerem MySQL

Aby nawiązać połączenie z serwerem, wywołaj program `mysql` z poziomu wiersza poleceń (to znaczy z powłoki systemu UNIX lub okna wiersza poleceń w Windows). Wydawane polecenie przedstawia się następująco:

```
% mysql opcje
```

W książce używany jest znak procentu (%) do wskazania znaku zachęty. To jeden ze standardowych znaków zachęty w systemie UNIX, innym jest symbol dolara (\$). Z kolei znakiem zachęty w Windows jest `C:\>`. Kiedy wprowadzasz polecenia przedstawione w przykładach, pomijaj znak zachęty.

Złotn *opcje* w poleceniu `mysql` może być pominięty, ale najczęściej będziesz wydawał polecenia podobne do przedstawionego poniżej:

```
% mysql -h nazwa_komputera -p -u nazwa_uzytkownika
```

Nie zawsze trzeba podawać wszystkie opcje podczas wywoływania `mysql`, choć prawdopodobnie konieczne będzie podanie przynajmniej nazwy użytkownika i hasła. Poniżej przedstawiono znaczenie użytych wcześniej opcji:

- `-h nazwa_komputera` (forma alternatywna: `--host=nazwa_komputera`)

Komputer, w którym jest uruchomiony serwer MySQL. Jeżeli to będzie ten sam komputer, w którym uruchomiono program klienta `mysql`, wtedy tę opcję można pominąć.

- `-u nazwa_uzytkownika` (forma alternatywna: `--user=nazwa_uzytkownika`)

Twoja nazwa użytkownika w MySQL. Jeżeli używasz systemu UNIX i nazwa użytkownika MySQL jest taka sama jak nazwa stosowana do zalogowania się, wtedy możesz pominąć tę opcję. Klient `mysql` użyje nazwy logowania jako nazwy użytkownika MySQL.

W systemie Windows domyślna nazwa to ODBC, co nie jest zbyt użyteczne. Musisz więc podać opcję `-u` w wierszu poleceń lub dodać domyślną nazwę użytkownika do danych środowiskowych przez zdefiniowanie zmiennej `USER`.

Na przykład użyj poniższego polecenia `set` do wskazania nazwy użytkownika `sampadm`:

```
C:\> set USER=sampadm
```

Jeżeli zdefiniujesz zmienną środowiskową `USER` przy pomocy elementu *System* w *Panelu Sterowania*, efekt będzie widoczny we wszystkich oknach konsoli i nie trzeba będzie wydawać powyższego polecenia w wierszu poleceń.

■ `-p` (forma alternatywna: `--password`)

Ta opcja powoduje, że klient `mysql` wyświetla pytanie o hasło w postaci komunikatu `Enter password:`, na przykład:

```
% mysql -h nazwa_komputera -p -u nazwa_uzytkownika
Enter password:
```

Kiedy zobaczysz komunikat `Enter password:`, podaj swoje hasło. (Wpisywane hasło nie będzie wyświetlane na ekranie, więc jeśli ktoś spogląda Ci przez ramię, to i tak nie pozna hasła). Zwróć uwagę, że hasło MySQL niekoniecznie musi być takie samo, jak używane do zalogowania się do systemu.

Jeżeli pominiesz opcję `-p`, wtedy `mysql` uzna, że nie ma konieczności podawania hasła, i pytanie nie zostanie wyświetlone.

Innym sposobem podania hasła jest jego bezpośrednie umieszczenie w wierszu poleceń jako *-hasło* (forma alternatywna: `--password=hasło`). Jednak ze względów bezpieczeństwa lepiej unikać tego rodzaju rozwiązania. Hasło jest widoczne podczas wpisywania, a tym samym pozostaje widoczne dla każdego, kto patrzy na ekran. Ponadto, w systemie UNIX inni użytkownicy mogą użyć narzędzi systemowych w celu podglądania powłoki.

Jeżeli mimo wszystko zdecydujesz się na podanie hasła w wierszu poleceń, pamiętaj, że między opcją `-p` i hasłem *nie ma spacji*. Zachowanie opcji `-p` można uznać za mylące, ponieważ jest inne niż opcji `-h` i `-u`, które są powiązane ze znajdującą się po nich wartością niezależnie od użycia spacji pomiędzy opcją i wartością.

Przyjmujemy założenie, że nazwa użytkownika i hasło MySQL to odpowiednio `sampadm` i `secret`. Jeżeli serwer MySQL działa w tym samym komputerze, w którym uruchamiasz klienta `mysql`, wtedy możesz pominąć opcję `-h`, a polecenie `mysql`, służące do nawiązania połączenia z serwerem, będzie miało poniższą postać:

```
% mysql -p -u sampadm
Enter password: *****
```

Po wydaniu polecenia klient `mysql` wyświetla pytanie o hasło (`Enter password:`), które powinien wpisać (gwiazdki oznaczają podanie hasła, tutaj to `secret`).

Jeżeli nie będzie żadnych problemów, klient `mysql` wyświetli komunikat powitania i znak zachęty `mysql>`, oznaczający gotowość do otrzymywania zapytań. Poniżej przedstawiono pełną sekwencję startową:

```
% mysql -p -u sampadm
Enter password: *****
```

```
Welcome to the MySQL monitor.  Commands end with ; or \g.  
Your MySQL connection id is 13762  
Server version: 5.5.30-log
```

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

```
mysql>
```

Aby nawiązać połączenie z serwerem działającym w innym komputerze, konieczne jest podanie nazwy komputera przy użyciu opcji `-h`. Jeżeli wspomnianym komputerem jest `cobra.example.com`, wtedy odpowiednie polecenie przedstawia się następująco:

```
% mysql -h cobra.example.com -p -u sampadm
```

W większości przykładów przedstawionych w książce i używających klienta `mysql` pominięto opcje `-h`, `-u` i `-p` w celu zachowania przejrzystości — przyjęto założenie, że będziesz je stosował, jeśli wystąpi konieczność. Wymienionych opcji trzeba również używać podczas uruchamiania innych programów MySQL, na przykład `mysqlshow`.

Po nawiązaniu połączenia z serwerem sesję możesz zakończyć w dowolnej chwili, wydając polecenie `quit`:

```
mysql> quit  
Bye
```

Istnieje również możliwość zakończenia pracy przez wydanie polecenia `exit` lub `\q`. W systemie UNIX naciśnięcie kombinacji klawiszy `Ctrl+D` również kończy pracę.

Kiedy rozpoczynasz poznawanie bazy danych MySQL, system bezpieczeństwa prawdopodobnie uznasz za irytujący, ponieważ utrudnia uzyskanie oczekiwanego wyniku. (Konieczne jest posiadanie odpowiednich uprawnień w celu utworzenia bazy danych i uzyskania dostępu do niej, a ponadto istnieje wymóg podawania nazwy użytkownika i hasła w trakcie nawiązywania połączenia z serwerem). Jeżeli jednak wykroczysz poza przykładową bazę danych używaną w tej książce i zaczniesz pracować z własnymi rekordami, Twoja perspektywa ulegnie całkowitej zmianie. Wówczas docenisz sposób, w jaki MySQL uniemożliwia innym użytkownikom przeglądanie (lub jeszcze gorzej — niszczenie) Twoich danych.

Jeśli chcesz dowiedzieć się, w jaki sposób przeprowadzić konfigurację środowiska pracy, aby uniknąć konieczności podawania parametrów połączenia w wierszu poleceń w trakcie każdego uruchamiania klienta `mysql`, zapoznaj się z podrozdziałem 1.5, zatytułowanym „Wskazówki przydatne podczas pracy z klientem `mysql`”. Najczęściej spotykanym sposobem uproszczenia procesu nawiązywania połączenia jest przechowywanie parametrów połączenia w pliku opcji. Zajrzyj do wymienionego podrozdziału i przekonaj się, jak przygotować tego rodzaju plik.

1.4.4. Wykonywanie zapytań SQL

Po nawiązaniu połączenia z serwerem możesz przystąpić do wykonywania zapytań SQL. W tym punkcie zostaną przedstawione pewne ogólne reguły pracy z klientem `mysql`.

Aby wykonać zapytania w kliencie mysql, wystarczy je wpisać. Na końcu zapytania umieść średnik (;) i naciśnij klawisz *Enter*. Średnik informuje klienta mysql o zakończeniu zapytania. Po wprowadzeniu zapytania klient mysql wysyła je do serwera w celu jego wykonania. Serwer wykonuje otrzymane zapytanie, a następnie wynik przekazuje z powrotem klientowi mysql, który z kolei wyświetla go użytkownikowi.

Poniżej przedstawiono proste zapytanie wyświetlające bieżącą datę i godzinę:

```
mysql> SELECT NOW();
+-----+
| NOW() |
+-----+
| 2013-01-08 17:42:33 |
+-----+
1 row in set (0.00 sec)
```

Innym sposobem wskazania końca zapytania jest użycie \g (ang. *go*, *idź*) zamiast średnika:

```
mysql> SELECT NOW()\g
+-----+
| NOW() |
+-----+
| 2013-01-08 17:42:40 |
+-----+
1 row in set (0.00 sec)
```

Ewentualnie można użyć \G, co powoduje wyświetlenie wyników w formacie „pionowym”, czyli po jednej wartości w każdym wierszu:

```
mysql> SELECT NOW(), USER(), VERSION()\G
***** 1. row *****
NOW(): 2013-01-08 17:54:24
USER(): sampadm@localhost
VERSION(): 5.5.30-log
1 row in set (0.00 sec)
```

W przypadku zapytań generujących krótkie wiersze danych wyjściowych użycie \G nie jest zbyt użyteczne. Natomiast jeśli wiersze danych wyjściowych są długie i zawinięte na ekranie, wtedy użycie \G powoduje ich wyświetlenie w formacie łatwiejszym do odczytu.

Klient mysql wyświetla wynik wykonania zapytania, a także wiersz dodatkowy z informacjami o liczbie rekordów użytych w danych wyjściowych oraz czasie przetwarzania zapytania. W kolejnych zapytaniach wspomniany wiersz będzie najczęściej pominięty.

Ponieważ klient mysql oczekuje na znak kończący zapytanie, całe zapytanie nie musi znajdować się w pojedynczym wierszu. Jeśli chcesz, możesz je umieścić w kilku wierszach:

```
mysql> SELECT NOW(),
-> USER(),
-> VERSION()
-> ;
+-----+-----+-----+
| NOW() | USER() | VERSION() |
+-----+-----+-----+
| 2013-01-08 17:54:56 | sampadm@localhost | 5.5.30-log |
+-----+-----+-----+
```

Zwróć uwagę na zmianę znaku zachęty z `mysql>` na `->` po wprowadzeniu pierwszego wiersza zapytania. W ten sposób `mysql` wskazuje, że oczekuje na dalszą część zapytania, co jest cenną wskazówką dla użytkownika. Jeśli zapomnisz umieścić znak kończący zapytanie, to zmieniony znak zachęty przypomni, że `mysql` wciąż czeka na zakończenie wprowadzania zapytania. W przeciwnym razie mógłbyś niecierpliwie czekać i zastanawiać się, dlaczego MySQL potrzebuje tak długiego czasu na wykonanie zapytania! Klient `mysql` stosuje także kilka innych znaków zachęty; zostały one omówione w dodatku F, zatytułowanym „Przewodnik po programie MySQL”.

Jeżeli rozpoczniesz wprowadzanie zapytania składającego się z wielu wierszy i nagle postanowisz, że nie chcesz go wykonać, wtedy umieść wpis `\c` w celu jego anulowania (wyczyszczenia):

```
mysql> SELECT NOW(),
-> VERSION(),
-> \c
mysql>
```

Zauważ, jak znak zachęty powrócił do standardowej postaci `mysql->` i wskazuje, że klient `mysql` jest gotowy do otrzymania nowego zapytania.

Przeciwieństwem wprowadzenia zapytania w wielu wierszach jest umieszczenie wielu zapytań w jednym wierszu. W takim przypadku powinny być rozdzielone średnikami:

```
mysql> SELECT NOW();SELECT USER();SELECT VERSION();
+-----+
| NOW() |
+-----+
| 2013-01-08 17:55:20 |
+-----+
+-----+
| USER() |
+-----+
| sampadm@localhost |
+-----+
+-----+
| VERSION() |
+-----+
| 5.5.30-log |
+-----+
```

W większości przypadków nie ma znaczenia wielkość liter użytych w zapytaniu. Pobrane informacje będą dokładnie takie same, choć nagłówki kolumn w wynikach zostaną wyświetlone w taki sposób, w jaki zapisano je w zapytaniu:

```
SELECT USER();
select user();
SeLeCt UsEr();
```

W przykładach przedstawionych w książce słowa kluczowe SQL i nazwy funkcji są zapisywane wielkimi literami, natomiast nazwy baz danych, tabel i kolumn małymi.

Kiedy wywołujesz funkcję w zapytaniu, między nazwą funkcji i nawiasem otwierającym nie umieszczaj spacji, ponieważ może ona spowodować wystąpienie błędu składni.

Zapytania mogą być przechowywane w pliku, tworząc w ten sposób skrypt SQL. Następnie klientowi `mysql` można nakazać odczyt zapytań z pliku zamiast z klawiatury. To, oczywiście, wymaga użycia funkcji przekierowania danych wejściowych oferowanej przez wiersz poleceń. Na przykład, jeśli zapytania znajdują się w pliku o nazwie *myscript.sql*, wówczas można je wykonać przy użyciu poniższego polecenia (pamiętaj o podaniu wszelkich wymaganych opcji połączenia):

```
% mysql < myscript.sql
```

Plikowi zawierającemu zapytania można nadać dowolną nazwę. W powyższym przykładzie użyto rozszerzenia *.sql* w celu wskazania, że plik zawiera zapytania SQL.

Wywołanie klienta `mysql` w pokazany sposób w celu wykonania zapytań zawartych w pliku to rozwiązanie, które będzie przedstawione w punkcie 1.4.7, zatytułowanym „Dodawanie nowych rekordów”, gdy będziemy wstawiać dane do bazy danych `sampdb`. Znacznie wygodniejszym rozwiązaniem jest wczytanie tabeli przez klienta `mysql` odczytującego zapytania `INSERT` z pliku niż wprowadzanie każdego z nich ręcznie z klawiatury.

W pozostałej części tego przewodnika poznasz wiele zapytań SQL, które będziesz mógł samodzielnie wypróbować. Łatwo je rozpoznasz po znaku zachęty `mysql>` przed zapytaniem, ponadto bardzo często towarzyszą im dane wyjściowe wygenerowane przez to zapytanie. Zapytania możesz wykonywać w postaci przedstawionej w książce, a otrzymane dane wyjściowe powinny być takie same. Zapytania przedstawione bez znaku zachęty na początku mają na celu jedynie ilustrację omawianego zagadnienia. Nie musisz ich wykonywać, choć oczywiście możesz, jeśli chcesz. Pamiętaj o zakończeniu każdego zapytania odpowiednim znakiem kończącym, na przykład średnikiem.

1.4.5. Tworzenie bazy danych

Użycie bazy danych obejmuje kilka kroków:

1. Utworzenie bazy danych.
2. Utworzenie tabel w bazie danych.
3. Operacje na tabelach, na przykład wstawianie, pobieranie, modyfikowanie lub usuwanie danych.

W celu utworzenia nowej bazy danych połącz się z serwerem, używając klienta `mysql`. Następnie wykonaj zapytanie `CREATE DATABASE` wraz z nazwą tworzonej bazy danych:

```
mysql> CREATE DATABASE sampdb;
```

Musisz utworzyć bazę danych `sampdb`, zanim będziesz mógł przystąpić do utworzenia jakiegokolwiek tabeli umieszczanej w `sampdb` lub zanim będziesz mógł zrobić cokolwiek z zawartością jej tabel.

Być może oczekujesz, że utworzenie bazy danych automatycznie uczyni ją domyślną (lub bieżącą), ale to nieprawda. Możesz się o tym przekonać, wykonując poniższe zapytanie, które wyświetla nazwę domyślnej bazy danych:

```
mysql> SELECT DATABASE();
+-----+
| DATABASE() |
+-----+
| NULL      |
+-----+
```

Wartość NULL oznacza „żadna baza danych nie została wybrana”. W celu wskazania sampdb jako domyślnej bazy danych wykonaj zapytanie USE:

```
mysql> USE sampdb;
mysql> SELECT DATABASE();
+-----+
| DATABASE() |
+-----+
| sampdb     |
+-----+
```

Innym sposobem wyboru domyślnej bazy danych jest podanie jej nazwy w wierszu poleceń w trakcie wywoływania klienta mysql:

```
% mysql sampdb
```

W rzeczywistości to jest najczęściej stosowany sposób wyboru bazy danych. Jeżeli konieczne jest użycie jakichkolwiek parametrów połączenia, podaj je w wierszu poleceń. Na przykład, przedstawione poniżej polecenie pozwala użytkownikowi sampadm na nawiązanie połączenia z bazą danych sampdb w komputerze lokalnym (zachowanie domyślne w przypadku pominięcia nazwy komputera):

```
% mysql -p -u sampadm sampdb
```

W celu nawiązania połączenia z serwerem MySQL działającym w zdalnym komputerze należy jego nazwę podać w wierszu poleceń:

```
% mysql -h cobra.example.com -p -u sampadm sampdb
```

O ile nie zostanie podane inaczej, we wszystkich przykładach przedstawionych w książce przyjęto założenie, że podczas wywoływania klienta mysql w wierszu poleceń podajesz nazwę bazy danych sampdb, czyniąc ją tym samym domyślną bazą danych. Jeśli wywołasz mysql i zapomnisz o podaniu nazwy bazy danych w wierszu poleceń, wtedy po prostu wykonaj zapytanie USE sampdb po znaku zachęty mysql>.

1.4.6. Tworzenie tabel

W tym punkcie utworzymy tabele niezbędne w przykładowej bazie danych sampdb. W pierwszej kolejności zastanowimy się, jakie tabele są konieczne do obsługi projektu Ligi Historycznej, a następnie przejdziemy do projektu ocen uczniów. W trakcie omawiania sposobów wykonywania tego rodzaju operacji w niektórych książkach poświęconych bazom danych wprowadzane są tematy analizy i projektowania, wykresy relacji w encjach, procedury normalizacyjne itd. To, oczywiście, odpowiedni moment na poruszenie wymienionych tematów, ale osobiście wolę zwrócić uwagę na konieczność zastanowienia się nad wyglądem bazy danych: jakie tabele powinna zawierać, co będzie w nich umieszczone

i jakie problemy można napotkać, wybierając odpowiednie rozwiązanie w zakresie przedstawienia danych?

Dokonane w tym miejscu wybory dotyczące sposobu prezentacji danych nie są ostateczne. W innych sytuacjach możesz zdecydować się na przedstawienie podobnych danych w nieco inny sposób, w zależności od wymagań aplikacji i sposobu ich użycia.

1.4.6.1. Tabele dla projektu Ligi Historycznej

Układ tabel dla projektu Ligi Historycznej jest całkiem prosty.

- Tabela `president`. W tej tabeli znajdują się rekordy opisujące poszczególnych prezydentów USA. To niezbędne do przeprowadzania quizów na witrynie internetowej (interaktywna wersja drukowanego quizu pojawi się w gazecie wydawanej przez Ligę).
- Tabela `member`. Ta tabela jest przeznaczona do przechowywania informacji o poszczególnych członkach Ligi. Będzie ona wykorzystywana do tworzenia zarówno drukowanego, jak i umieszczonego na witrynie internetowej katalogu członków Ligi, a także do automatycznego wysyłania informacji o konieczności przedłużenia członkostwa itd.

1.4.6.1.1. Tabela `president`

Tabela `president` będzie zawierała pewne podstawowe informacje o poszczególnych prezydentach USA.

- **Imię i nazwisko.** Imię i nazwisko można przedstawić w tabeli na wiele różnych sposobów, na przykład umieścić je w pojedynczej kolumnie lub utworzyć oddzielne dla imienia i nazwiska. Użycie pojedynczej kolumny jest prostsze, choć jednocześnie nakłada pewne ograniczenia:
 - ◆ Jeżeli imię wprowadzisz jako pierwsze, nie będzie możliwości sortowania według nazwiska.
 - ◆ Jeżeli jako pierwsze wprowadzisz nazwisko, nie będziesz mógł wyświetlić imienia jako pierwszego.
 - ◆ Trudniej jest przeprowadzić operację wyszukiwania po imionach. Na przykład, aby wyszukać konkretne nazwisko, trzeba użyć wzorca i wyszukać imiona dopasowane do wzorca. To znacznie mniej efektywne i wolniejsze niż wyszukanie konkretnego nazwiska.

Aby uniknąć wymienionych ograniczeń, w tabeli `president` użyjemy oddzielnych kolumn dla imion i nazwisk.

Kolumna imienia będzie zawierała także drugie imię lub inicjał. To nie powinno być problemem w trakcie sortowania, ponieważ mało prawdopodobne jest przeprowadzanie operacji sortowania według drugiego lub nawet pierwszego imienia. Imię będzie wyświetlone prawidłowo, ponieważ drugie jest umieszczane zaraz po pierwszym niezależnie od zastosowanego stylu, na przykład „Bush, George W.” lub „George W. Bush”.

Mamy jeszcze jedno drobne utrudnienie. Jeden prezydent (Jimmy Carter) ma przyrostek „Jr.” na końcu imienia. Gdzie powinien być umieszczony wymieniony przyrostek? W zależności od stylu wyświetlania informacji, imię i nazwisko prezydenta zostanie podane jako „James E. Carter, Jr.” lub „Carter, James E., Jr.”. Przyrostek nie może więc zostać powiązany z kolumną imienia lub nazwiska i dlatego trzeba utworzyć oddzielną kolumnę dla przyrostka. To pokazuje, że nawet pojedyncza wartość może powodować problemy, gdy próbujesz znaleźć sposób odpowiedniego przedstawienia danych. Teraz już wiesz, dlaczego dobrym pomysłem jest zebranie maksymalnej ilości informacji o danych, z którymi będziesz pracować, zanim rozpocznie ich umieszczanie w bazie danych. Jeśli nie masz wystarczającej wiedzy o używanych wartościach, może wystąpić konieczność zmiany struktury tabeli już po rozpoczęciu jej używania. To niekoniecznie będzie katastrofą, ale ogólnie rzecz biorąc lepiej tego unikać.

- **Miejsce urodzenia (miasto i stan).** Podobnie jak w przypadku imienia i nazwiska, także miasto i stan można umieścić w pojedynczej kolumnie lub w wielu. Prostszym rozwiązaniem jest użycie pojedynczej kolumny, ale podobnie jak wcześniej rozdzielanie kolumn daje możliwość przeprowadzenia pewnych operacji, które w przeciwnym razie nie będą łatwe do wykonania. Na przykład, łatwiej wyszukać prezydentów urodzonych w danym stanie, gdy miasto i stan umieszczono w oddzielnych kolumnach. W omawianym przykładzie użyjemy dwóch kolumn.
- **Data urodzenia i śmierci.** W tym miejscu trzeba pamiętać, że nie powinniśmy wymagać podania daty śmierci prezydenta, ponieważ niektórzy mogą nadal żyć. Wartość specjalna NULL oznacza „brak wartości”, więc można jej użyć w kolumnie zawierającej datę śmierci i tym samym wskazać, że prezydent nadal żyje.

1.4.6.1.2. Tabela member

Tabela member dla projektu Ligi Historycznej jest podobna do tabeli pres ident pod tym względem, że każdy rekord zawiera podstawowe informacje o jednej osobie. Jednak w przeciwieństwie do tabeli pres ident, każdy rekord tabeli member zawiera więcej kolumn.

- **Imię i nazwisko.** Wykorzystamy to samo oparte na trzech kolumnach rozwiązanie, które zastosowaliśmy w tabeli pres ident: nazwisko, imię i prefiks.
- **Identyfikator.** To jest unikalna wartość przypisywana każdemu członkowi po rozpoczęciu okresu członkostwa. Liga nigdy wcześniej nie stosowała identyfikatorów, ale gdy teraz rekordy stały się bardziej systematyczne, dobrym pomysłem jest rozpoczęcie stosowania identyfikatorów. (Spodziewam się, że korzystanie z bazy danych MySQL uznasz za zaletę i znajdziesz inne zastosowania dla rekordów Ligi. Kiedy tak się stanie, powiązanie rekordów tabeli member z innymi tabelami dotyczącymi członków Ligi stanie się łatwiejsze, jeśli będziesz używał liczb zamiast nazwisk).
- **Data wygaśnięcia członkostwa.** Członkowie Ligi muszą co pewien czas odnawiać członkostwo. W niektórych aplikacjach wystarczy przechowywać datę ostatniego odnowienia członkostwa, ale takie rozwiązanie nie jest wystarczające na potrzeby

projektu Ligi. Członkostwo można odnowić na różne okresy (najczęściej rok, dwa, trzy lub pięć lat), a data ostatniego odnowienia nie informuje, kiedy członkostwo powinno zostać odnowione. Dlatego też będziemy przechowywać datę wygaśnięcia członkostwa. Poza tym Liga oferuje możliwość dożywotniego członkostwa. Wprawdzie można to przedstawić przy użyciu daty z odległej przyszłości, ale wartość NULL wydaje się bardziej odpowiednia, ponieważ „brak wartości” logicznie odpowiada „nigdy nie wygasa”.

- **Adres e-mail.** Publikacja adresu e-mail ułatwia członkom Ligi wzajemną komunikację. Na potrzeby sekretariatu Ligi te adresy pozwalają na wysyłanie elektroniczne zamiast tradycyjną pocztą powiadomień o zbliżającym się terminie odnowienia członkostwa. To znacznie łatwiejsze i tańsze niż konieczność skorzystania z placówki poczty. Adres e-mail można również wykorzystać do wysyłania członkom Ligi bieżącej zawartości katalogu i proszenia ich o uaktualnienie wymaganych informacji.
- **Adres pocztowy.** Ten adres jest konieczny w celu kontaktu z członkami Ligi, którzy nie mają poczty elektronicznej (lub nie odpowiadają na wiadomości e-mail). Użyjemy kolumn pozwalających na przechowywanie ulicy, miasta, stanu i kodu pocztowego.
Przyjmujemy założenie, że wszyscy członkowie Ligi mieszkają w USA. W przypadku organizacji, których członkowie pochodzą z całego świata, przyjęte przez nas założenie jest nadmiernym uproszczeniem. Jeżeli chcesz zapewnić obsługę adresów z wielu krajów, wówczas będziesz musiał się zmierzyć z pewnymi problemami, na przykład różnymi formatami adresów stosowanymi w poszczególnych krajach. Kod pocztowy nie jest standardem międzynarodowym, a niektóre kraje zamiast stanów używają na przykład prowincji.
- **Numer telefonu.** Podobnie jak adres pocztowy, numer telefonu jest użyteczny w celu nawiązywania kontaktu z członkiem Ligi.
- **Słowa kluczowe opisujące zainteresowania.** Przyjmuje się założenie, że każdy członek Ligi jest ogólnie zainteresowany historią USA, ale członkowie Ligi prawdopodobnie mają specjalnie obszary zainteresowań. Wspomniane zainteresowania można zapisać w tej kolumnie. Członkowie Ligi zyskują możliwość wyszukiwania innych osób o podobnych zainteresowaniach. (Szczepnie mówiąc, znacznie lepszym rozwiązaniem będzie przygotowanie oddzielnej tabeli słów kluczowych, w której każdy rekord będzie składał się z jednego słowa kluczowego i powiązanego z nim identyfikatora członka Ligi. Takie rozwiązanie jednak nieco komplikuje nasz projekt i nie będziemy go stosować).

1.4.6.1.3. Utworzenie tabel dla projektu Ligi Historycznej

W tym momencie jesteśmy gotowi do utworzenia tabel dla projektu Ligi Historycznej. Wykorzystamy więc do tego celu zapytanie CREATE TABLE, którego ogólna składnia przedstawia się następująco:

```
CREATE TABLE nazwa_tabeli (specyfikacja_kolumn);
```

Parametr *nazwa_tabeli* wskazuje nazwę nadawaną tabeli, natomiast *specyfikacja_kolumn* zawiera całą specyfikację tworzonych kolumn. Zapytanie obejmuje także definicje indeksów, o ile takie mają zostać utworzone. Wspomniane indeksy przyspieszają operacje wyszukiwania i dokładnie omówimy je w rozdziale 5, zatytułowanym „Optymalizacja zapytań”.

Do utworzenia tabeli *president* użyjemy przedstawionego poniżej zapytania CREATE TABLE:

```
CREATE TABLE president
(
    last_name  VARCHAR(15) NOT NULL,
    first_name VARCHAR(15) NOT NULL,
    suffix     VARCHAR(5)  NULL,
    city       VARCHAR(20) NOT NULL,
    state      VARCHAR(2)  NOT NULL,
    birth      DATE NOT NULL,
    death      DATE NULL
);
```

Przedstawione zapytanie można wykonać na wiele sposobów. Wpisz je ręcznie z klawiatury lub wykorzystaj zapytanie umieszczone w pliku *create_president.sql* znajdującym się w dystrybucji *sampdb*.

W celu samodzielnego wpisania zapytania uruchom klienta *mysql* i wskaż *sampdb* jako domyślną bazę danych.

```
% mysql sampdb
```

Następnie wpisz przedstawione powyżej zapytanie CREATE TABLE wraz ze średnikiem na końcu wskazującym miejsce zakończenia zapytania. Wcięcia nie mają znaczenia, poszczególne wiersze nie muszą być takie same jak zaprezentowano w przykładzie. Jeśli chcesz, całe zapytanie możesz umieścić w jednym długim wierszu.

Aby utworzyć tabelę *president* na podstawie zapytań umieszczonych w pliku, wykorzystaj plik *create_president.sql* z dystrybucji *sampdb*. Wymieniony plik znajduje się w katalogu *sampdb*, który jest tworzony podczas rozpakowywania dystrybucji. Przejdź do wymienionego katalogu, a następnie wydaj poniższe polecenie:

```
% mysql sampdb < create_president.sql
```

Niezależnie od sposobu uruchomienia klienta *mysql*, wszystkie wymagane parametry połączenia (nazwa komputera, użytkownika i hasło) podaj w wierszu polecenia po samym poleceniu *mysql*.

Spójrz nieco dokładniej na zapytanie CREATE TABLE. Każda specyfikacja kolumny w zapytaniu składa się z nazwy kolumny, typu przechowywanych danych (czyli rodzaju wartości umieszczonych w tej kolumnie) i ewentualnie z pewnych atrybutów kolumny.

Dwa typy danych użyte w tabeli *president* to VARCHAR i DATE. Typ VARCHAR(*n*) oznacza, że kolumna zawiera dane tekstowe o różnej wielkości, przy czym ich wielkość maksymalna wynosi *n* znaków. Oznacza to możliwość przechowywania ciągów tekstowych o zmiennej wielkości, ale nie większych niż podana górna granica. Wartość *n* oznacza maksymalną wielkość przechowywanego ciągu tekstowego. Kolumna *state* została zdefiniowana jako VARCHAR(2), ponieważ dwa znaki wystarczą do podania skrótu nazwy stanu. Pozostałe

kolumny mają zdefiniowaną większą wartość graniczną, ponieważ muszą przechowywać dłuższe ciągi tekstowe.

Drugi użyty typ danych to DATE. Jak nietrudno zgadnąć, ten typ wskazuje, że kolumna przechowuje datę. Jednak zaskoczeniem może być format przedstawienia daty. Baza danych MySQL oczekuje podania daty w formacie 'CCYY-MM-DD', gdzie CC, YY, MM i DD przedstawiają wiek, rok w danym wieku, miesiąc i dzień miesiąca. To jest standard SQL do prezentacji daty (ten standard jest nazywany również „formatem ISO 8601”). Na przykład, w celu podania daty 18 lipca 2013 roku w MySQL należy użyć '2013-07-18', a nie '07-18-2013' lub '18-07-2013'.

Jedynie atrybuty użyte w kolumnach tabeli `president` to NULL (brak wartości) i NOT NULL (wartość musi być podana). Większość kolumn ma zdefiniowany atrybut NOT NULL, ponieważ chcemy, aby ich wartości zawsze były podawane. Dwie kolumny, którym przypisano atrybut NULL, to `suffix` (większość imion nie ma przyrostków) i `death` (w przypadku żyjących prezydentów nie można podać daty ich śmierci).

Do utworzenia tabeli `member` użyjemy przedstawionego poniżej zapytania CREATE TABLE:

```
CREATE TABLE member
(
  member_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (member_id),
  last_name VARCHAR(20) NOT NULL,
  first_name VARCHAR(20) NOT NULL,
  suffix VARCHAR(5) NULL,
  expiration DATE NULL,
  email VARCHAR(100) NULL,
  street VARCHAR(50) NULL,
  city VARCHAR(50) NULL,
  state VARCHAR(2) NULL,
  zip VARCHAR(10) NULL,
  phone VARCHAR(20) NULL,
  interests VARCHAR(255) NULL
);
```

Podobnie jak wcześniej, zapytanie możesz wpisać ręcznie w kliencie `mysql` lub wczytać je z pliku. W dystrybucji `sampdb` znajdziesz plik o nazwie `create_member.sql` zawierający zapytanie CREATE TABLE przeznaczone do utworzenia tabeli `member`. Aby użyć wymienionego pliku, musisz wykonać przedstawione poniżej polecenie:

```
% mysql sampdb < create_member.sql
```

Biorąc pod uwagę typy danych, większość kolumn tabeli `member`, poza dwoma wyjątkami, nie jest szczególnie interesująca, ponieważ są utworzone jako ciągi tekstowe o zmiennej wielkości. Wyjątkami są kolumny `member_id` i `expiration`, które przechowują odpowiednio sekwencję liczb i daty.

Podstawową cechą identyfikatora liczbowego przechowywanego w kolumnie `member_id` jest to, że wszystkie wartości powinny być unikalne, aby nie zachodziły żadne pomyłki. Kolumna typu AUTO_INCREMENT jest użyteczna, ponieważ to baza danych MySQL staje się odpowiedzialna za automatyczne wygenerowanie unikalnych liczb podczas dodawania nowych członków Ligi. Wprawdzie kolumna `member_id` przechowuje jedynie liczby, ale jej definicja składa się z kilku części:

- **INT** oznacza, że kolumna przechowuje liczby całkowite (czyli liczby bez części dziesiętnych).
- **UNSIGNED** nie pozwala na przechowywanie liczb ujemnych.
- **NOT NULL** oznacza, że kolumna musi mieć wartość. Dzięki temu nie ma możliwości utworzenia rekordu członka Ligi bez przypisanego mu identyfikatora.
- **AUTO_INCREMENT** to specjalny atrybut w MySQL. Wskazuje, że kolumna przechowuje sekwencję liczb. Mechanizm **AUTO_INCREMENT** działa w następujący sposób: jeżeli podczas tworzenia nowego rekordu tabeli member nie podasz wartości kolumny `member_id`, wówczas MySQL automatycznie wygeneruje następną liczbę w sekwencji i przypisze ją kolumnie w danym rekordzie. Takie specjalne zachowanie występuje także po przypisaniu wartości **NULL** kolumnie `member_id`. Dzięki mechanizmowi **AUTO_INCREMENT** można bardzo łatwo każdemu członkowi Ligi przypisać unikalny identyfikator, ponieważ MySQL automatycznie wygeneruje odpowiednią wartość.

Klauzula **PRIMARY KEY** wskazuje, że kolumna `member_id` jest indeksowana w celu zapewnienia szybkiego przeprowadzania operacji wyszukiwania. Ponadto nakłada wymóg, że wszystkie wartości w danej kolumnie muszą być unikalne. Druga z wymienionych cech jest bardzo pożądana w przypadku identyfikatorów członków Ligi, ponieważ chroni przed pomyłkowym dwukrotnym użyciem tego samego identyfikatora. Poza tym, baza danych MySQL wymaga, aby każda kolumna o atrybucie **AUTO_INCREMENT** miała zdefiniowany pewien indeks, więc definicja tabeli byłaby nieprawidłowa w przypadku braku indeksu. (Każda kolumna **PRIMARY KEY** musi mieć również przypisany atrybut **NOT NULL**, więc w przypadku jego pominięcia w definicji kolumny `member_id` MySQL automatycznie przypisze kolumnie wymieniony atrybut).

Jeżeli nie rozumiesz w pełni działania atrybutów **AUTO_INCREMENT** i **PRIMARY KEY**, to pomyśl o nich jako o magicznym sposobie generowania zindeksowanych numerów będących identyfikatorami. Nie ma znaczenia, jakie to będą wartości, o ile pozostaną unikalne. (Kiedy będziesz gotowy do dokładniejszego poznania kolumn o atrybucie **AUTO_INCREMENT**, zajrzyj do rozdziału 3, zatytułowanego „Typy danych”).

Kolumna `expiration` jest typu **DATE**. Pozwala na użycie wartości **NULL**, która jest również wartością domyślną i oznacza, że data nie została podana. Jak już wcześniej wspomniano, stosujemy konwencję, w której wartość **NULL** kolumny `expiration` oznacza członkostwo dożywotnie.

Skoro nakazałeś bazie danych MySQL utworzenie kilku tabel, warto się upewnić o ich utworzeniu zgodnie z oczekiwaniami. W kliencie `mysql` wykonaj przedstawione poniżej zapytanie, które spowoduje wyświetlenie struktury tabeli `president`:

```
mysql> DESCRIBE president;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| last_name | varchar(15) | NO | | NULL | |
```

first_name	varchar(15)	NO		NULL		
suffix	varchar(5)	YES		NULL		
city	varchar(20)	NO		NULL		
state	varchar(2)	NO		NULL		
birth	date	NO		NULL		
death	date	YES		NULL		
+-----+-----+-----+-----+-----+-----+						

Aby wyświetlić podobne informacje na temat tabeli member, wykonaj zapytanie `DESCRIBE member`. (Jeżeli zastanawiasz się, dlaczego kolumna `Default` zawiera wartość `NULL` dla kolumn niedopuszczających użycia wartości `NULL`, to wyjaśniam, że oznacza to brak użycia klauzuli `DEFAULT` dla danej kolumny).

Zapytanie `DESCRIBE` jest użyteczne, gdy zapomnisz nazwę kolumny w tabeli, chcesz znać typ przechowywanych w niej danych, chcesz znać wielkość kolumny itd. Przydaje się również do ustalenia kolejności, w której MySQL przechowuje kolumny w rekordach tabeli. Wspomniana kolejność jest ważna w trakcie używania zapytań `INSERT` lub `LOAD DATA`, które oczekują, aby wartości kolumn były wymienione w domyślnej kolejności kolumn.

Informacje wygenerowane przez zapytanie `DESCRIBE` można uzyskać na wiele sposobów. Mogą zostać skrócone, jak w przypadku `DESC`, lub zapisane w całości, jak w przypadku poleceń `EXPLAIN` bądź `SHOW`. Wymienione poniżej zapytania są synonimami:

```
DESCRIBE president;
DESC president;
EXPLAIN president;
SHOW COLUMNS FROM president;
SHOW FIELDS FROM president;
```

Powyższe zapytania pozwalają na ograniczenie danych wyjściowych do wskazanych kolumn. Na przykład, na końcu zapytania `SHOW` możesz umieścić klauzulę `LIKE` i wyświetlić informacje pochodzące jedynie z kolumn, których nazwy zostały dopasowane do danego wzorca:

```
mysql> SHOW COLUMNS FROM president LIKE '%name';
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| last_name  | varchar(15) | NO   |     |         |       |
| first_name | varchar(15) | NO   |     |         |       |
+-----+-----+-----+-----+-----+-----+
```

Odpowiednikiem powyższego zapytania jest `DESCRIBE president '%name'`. Użyty tutaj znak `'%'` to specjalny znak wieloznaczny omówiony w podpunkcie 1.4.9.7, zatytułowanym „Dopasowanie wzorca”.

Zapytanie `SHOW FULL COLUMNS` działa jak `SHOW COLUMNS`, ale wyświetla dodatkowe informacje o kolumnach. Wypróbuj wymienione zapytania i przekonaj się sam.

Zapytanie `SHOW` ma inne formy, które są użyteczne podczas pobierania różnego typu informacji z bazy danych MySQL. Zapytanie `SHOW TABLES` wyświetla listę tabel znajdujących się w domyślnej bazie danych. Z dwiema utworzonymi dotąd tabelami w bazie danych `sampdb` wygenerowane dane wyjściowe przedstawiają się następująco:

```
mysql> SHOW TABLES;
+-----+
| Tables_in_sampdb |
+-----+
| member           |
| president        |
+-----+
```

Z kolei zapytanie `SHOW DATABASES` wyświetla listę baz danych zarządzanych przez serwer, z którym nawiązano połączenie:

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql      |
| sampdb     |
| test       |
+-----+
```

Lista baz danych zależy od konkretnego serwera, ale powinieneś na niej otrzymać przynajmniej `information_schema` i `sampdb`. Baza `information_schema` to specjalna baza danych, która zawsze istnieje, natomiast `sampdb` utworzyłeś samodzielnie. Na liście prawdopodobnie znajduje się również baza danych o nazwie `test`, tworzona w trakcie instalacji MySQL. W zależności od posiadanych uprawnień dostępu, na liście możesz zobaczyć bazę danych `mysql` przechowującą tabele z informacjami o uprawnieniach użytkowników.

Program kliencki `mysqlshow` oferuje interfejs wiersza poleceń pozwalający na wyświetlenie tych samych informacji, które dostarcza zapytanie `SHOW`. Pamiętaj, że podczas wywoływania `mysqlshow` może wystąpić konieczność podania odpowiednich opcji, takich jak nazwa użytkownika, hasło i nazwa komputera. Wymienione opcje są takie same jak używane w trakcie wywołania klienta `mysql`.

Po wywołaniu bez argumentów `mysqlshow` wyświetla listę baz danych:

```
% mysqlshow
+-----+
| Databases |
+-----+
| information_schema |
| mysql      |
| sampdb     |
| test       |
+-----+
```

W przypadku podania nazwy bazy danych `mysqlshow` wyświetla tabele we wskazanej bazie danych:

```
% mysqlshow sampdb
Database: sampdb
+-----+
| Tables |
+-----+
```

```
| member |
| president |
+-----+
```

Natomiast po podaniu nazwy bazy danych i tabeli `mysql show` wyświetla informacje o kolumnach znajdujących się we wskazanej tabeli. Dane wyjściowe są więc podobne do wygenerowanych przez zapytanie `SHOW FULL COLUMNS`.

1.4.6.2. Tabele dla projektu ocen uczniów

W celu określenia tabel wymaganych w projekcie ocen uczniów przeanalizujemy sposób, w jaki oceny mogą być zapisywane w dzienniku papierowym. Na rysunku 1.2 pokazano stronę pochodzącą z takiego dziennika. Część główna strony zawiera tabelę służącą do zapisywania wyników uzyskanych przez poszczególnych uczniów. Oczywiście, znalazło się również miejsce na inne informacje wymagane, aby nadać sens wspomnianym wynikom. Imiona i identyfikatory uczniów są więc wymienione obok tabeli wyników. (W celu zachowania prostoty na rysunku pokazano tylko dane czworga uczniów). Na górze strony znajdują się daty przeprowadzenia sprawdzianów i testów. Z rysunku wynika, że sprawdziany zostały przeprowadzone 3, 6, 16 i 23 września, natomiast testy 9 września i 1 października.

uczniowie		wyniki						
ID	imię	Q	Q	T	Q	Q	T	
		3/9	6/9	9/9	16/9	23/9	1/10	...
1	Billy	14	10	73	14	15	67	...
2	Missy	17	10	68	17	14	73	...
3	Johnny	15	10	78	12	17	82	...
4	Jenny	14	13	85	13	19	79	...
...

Rysunek 1.2. Przykładowa strona papierowego dziennika ocen

Aby wymienione informacje przechowywać w bazie danych, konieczne jest utworzenie tabeli `score`. Co powinien zawierać rekord tej tabeli? To łatwe — w każdym rekordzie należy umieścić imię ucznia, datę przeprowadzenia sprawdzianu lub testu oraz uzyskany wynik. Na rysunku 1.3 pokazano przykładowe wyniki z dziennika umieszczone we wspomnianej tabeli. (Data jest zapisana w formacie stosowanym przez MySQL, czyli `'CCYY-MM-DD'`).

Niestety, przygotowanie tabeli w przedstawiony powyżej sposób powoduje pominięcie pewnych informacji. Na przykład, z rekordów widocznych na rysunku 1.3 nie wynika, czy wynik został uzyskany w sprawdzianie czy w teście. Jeżeli wyniki sprawdzianów i testów mają inną wagę, to podczas wystawiania oceny końcowej trzeba znać kategorie wyników. Wprawdzie kategorię można wywnioskować na podstawie zakresu punktów w konkretnym dniu (sprawdziany zwykle mają mniejszą liczbę punktów niż testy), ale to problematyczne rozwiązanie, ponieważ nie opiera się bezpośrednio na danych.

Tabela score

name	date	score
Billy	2012-09-23	15
Missy	2012-09-23	14
Johnny	2012-09-23	17
Jenny	2012-09-23	19
Billy	2012-10-01	67
Missy	2012-10-01	73
Johnny	2012-10-01	82
Jenny	2012-10-01	79

Rysunek 1.3. Początkowy układ tabeli score

Istnieje możliwość rozróżnienia wyników po dodaniu do każdego rekordu informacji o kategorii. W tym celu do tabeli score trzeba dodać kolumnę zawierającą wartość T lub Q, oznaczającą odpowiednio test lub sprawdzian, jak pokazano na rysunku 1.4. W ten sposób kategoria wyniku jest wyraźnie określona w danych. Wadą takiego rozwiązania jest nadmiarowość danych. Zwróć uwagę, że dla wszystkich kolumn o podanej dacie kolumna kategorii zawsze zawiera tę samą wartość. Wszystkie wyniki z 23 września mają wartość Q w kolumnie kategorii, natomiast z 1 października mają wartość T. To jest niezbyt zachęcające rozwiązanie. Jeżeli w przedstawiony sposób w tabeli umieścimy zestaw wyników sprawdzianów lub testów, nie tylko będziemy wprowadzać tę samą datę dla każdego nowego rekordu w zestawie, ale również tę samą kategorię. Brr. Kto chciałby wprowadzać te wszystkie nadmiarowe informacje?

Tabela score

name	date	score	category
Billy	2012-09-23	15	Q
Missy	2012-09-23	14	Q
Johnny	2012-09-23	17	Q
Jenny	2012-09-23	19	Q
Billy	2012-10-01	67	T
Missy	2012-10-01	73	T
Johnny	2012-10-01	82	T
Jenny	2012-10-01	79	T

Rysunek 1.4. Układ tabeli score zmodyfikowany do przechowywania typu wyniku

Wypróbujmy alternatywne rozwiązanie. Zamiast umieszczać informacje o typie wyniku w tabeli score, będziemy je określać na podstawie daty. Można przechowywać listę dat i używać jej do określenia rodzaju „ocenianego zdarzenia” (sprawdzian lub test) przeprowadzonego tego dnia. Następnie kategorię wyniku będzie można ustalić przez połączenie informacji z obu tabel: dopasowując datę w rekordzie tabeli score z datą w rekordzie tabeli grade_event. Na rysunku 1.5. pokazano nowy układ tabel oraz sposób działania powiązania rekordu tabeli score zawierającego datę 23 września. Dzięki dopasowaniu rekordu z odpowiednim rekordem w tabeli grade_event można ustalić, czy wynik został otrzymany w sprawdzianie, czy w teście.

Tabela score

name	date	score
Billy	2012-09-23	15
Missy	2012-09-23	14
Johnny	2012-09-23	17
Jenny	2012-09-23	19
Billy	2012-10-01	67
Missy	2012-10-01	73
Johnny	2012-10-01	82
Jenny	2012-10-01	79

Tabela grade_event

date	category
2012-09-03	Q
2012-09-06	Q
2012-09-09	T
2012-09-16	Q
2012-09-23	Q
2012-10-01	T

Rysunek 1.5. Tabele score i grade_event powiązane datą

To znacznie lepsze rozwiązanie niż próba określenia kategorii poprzez zgadywanie. Teraz kategoria jest ustalana na podstawie konkretnych informacji przechowywanych w bazie danych. Preferowane jest również przechowywanie informacji o kategorii wyniku w tabeli score; teraz można to zrobić tylko jednokrotnie zamiast dla każdego rekordu.

Jednak w przedstawionym rozwiązaniu trzeba łączyć informacje pochodzące z dwóch tabel. Być może, kiedy słyszysz o takim rozwiązaniu, podobnie jak ja zadajesz sobie pytania: „Tak, to wydaje się dobrym pomysłem, ale czy nie wiąże się z większą ilością pracy podczas wyszukiwania informacji? Czy całe rozwiązanie nie zostało niepotrzebnie skomplikowane?”.

Pod pewnymi względami masz rację — rozwiązanie oznacza większą ilość pracy do wykonania. Obsługa dwóch list rekordów jest znacznie bardziej skomplikowana niż jednej. Spójrz jednak na kartkę w papierowym dzienniku pokazaną na rysunku 1.2. Czy w tamtej wersji nie mamy przypadkiem dwóch zestawów rekordów? Przeanalizuj następujące fakty:

- Wyniki są przechowywane w komórkach tabeli wyników, każda komórka jest indeksowana imieniem ucznia i datą (po lewej stronie i na górze tabeli wyników). To można uznać za jeden zestaw rekordów, analogiczny do zawartości tabeli score.
- Skąd wiesz, co oznacza konkretna data? Nad datą umieściłeś małą literę T lub Q! Tym samym nad tabelą wyników przechowujesz relacje pomiędzy datą i kategorią wyniku. To można uznać za drugi zestaw rekordów, analogiczny do zawartości tabeli grade_event.

Innymi słowy, mogłeś nawet się nad tym nie zastanawiać, ale przechowywanie informacji w dwóch tabelach tak naprawdę niczym nie różni się od sposobu użycia dziennika papierowego. Jedyna różnica polega na tym, że w dzienniku papierowym wymienione informacje nie są wyraźnie rozdzielone.

Strona dziennika pokazuje sposób traktowania przez nas informacji, a także pewną trudność w ich odpowiednim umieszczeniu w bazie danych. Nasz umysł ma tendencję do integrowania różnego rodzaju informacji i interpretacji ich jako całości. Bazy danych nie działają w ten sposób i dlatego czasami wydają się sztuczne lub nienaturalne. Nasza naturalna tendencja do unifikacji informacji bardzo często nawet utrudnia uświadomienie sobie, że mamy do czynienia z wieloma typami danych, a nie tylko z jednym. Z tego powodu może być trudne „myślenie w taki sposób, w jaki działa system bazy danych” w zakresie przedstawiania danych.

Jedynym wymaganiem narzucanym w tabeli `grade_event` przez układ pokazany na rysunku 1.5 to konieczność zachowania unikalności dat, ponieważ każda z nich łączy rekordy tabel `score` i `grade_event`. Innymi słowy, w jednym dniu nie można przeprowadzić dwóch zdarzeń ocenających, na przykład dwóch sprawdzianów lub sprawdzianu i testu. Jeżeli jednak to zrobisz, otrzymasz po dwa zestawy rekordów w tabelach `score` i `grade_event` o takiej samej dacie i nie będziesz miał możliwości określenia, jak dopasować rekordy tabeli `score` z rekordami tabeli `grade_event`.

Ten problem nigdy nie wystąpi, jeśli w danym dniu nie przeprowadzisz więcej niż tylko jednego zdarzenia ocenającego. Rodzi się pytanie, czy można przyjąć tego rodzaju założenie. Wydaje się, że tak, ponieważ nie chcesz być nauczycielem sadystą, który w jednym dniu przeprowadza sprawdzian i test. Jednak zbyt często spotykałem się z wyrażanymi przez użytkowników opiniami dotyczącymi ich danych: „Taka sytuacja nigdy się nie zdarzy”. Następnie okazuje się, że sytuacja, która nigdy nie miała się zdarzyć, jednak wystąpiła. W takim przypadku zwykle trzeba zmienić układ tabel, aby rozwiązać problemy powodowane przez wspomnianą sytuację.

O potencjalnych problemach lepiej pomyśleć wcześniej i przygotować się na ich rozwiązanie. Przyjmujemy więc założenie, że pewnego dnia może wystąpić konieczność umieszczenia w bazie danych dwóch zestawów rekordów.

W jaki sposób można rozwiązać ten problem? Jak się okazuje, nie jest to trudne. Po wprowadzeniu niewielkich zmian w układzie danych wiele zdarzeń w jednym dniu nie będzie żadnym problemem.

1. Dodajemy kolumnę do tabeli `grade_event` i używamy jej do przypisania unikalnego numeru każdemu rekordowi tabeli. W efekcie otrzymujemy identyfikator zdarzenia, a więc kolumnie nadajemy nazwę `event_id`. Jeżeli wydaje Ci się to dziwnym rozwiązaniem, ponownie spójrz na pokazaną na rysunku 1.2 kartę z dziennika papierowego: identyfikator zdarzenia można porównać do kolumny numeru w tabeli wyników. Ten numer nie został wyraźnie oznaczony jako „identyfikator zdarzenia”, ale działa właśnie w takim charakterze.
2. Podczas umieszczania wyniku w tabeli `score` zamiast daty podawaj identyfikator zdarzenia.

Wynik wprowadzonych zmian pokazano na rysunku 1.6. Tabele `score` i `grade_event` można połączyć przy użyciu identyfikatora zdarzenia zamiast daty. Tabela `grade_event` służy do określenia nie tylko kategorii wyniku, ale również daty jego otrzymania. Ponadto, w tabeli `grade_event` data nie musi być unikalna, ponieważ unikalny jest identyfikator zdarzenia. Oznacza to, że można przeprowadzić wiele sprawdzianów i testów w jednym dniu (uczniowie będą tym przerażeni) i wyniki bardzo łatwo umieścić w bazie danych.

Niestety, z punktu widzenia człowieka układ tabel pokazany na rysunku 1.6 wydaje się mniej zadowalający od pokazanego na wcześniejszym rysunku. Tabela `score` stała się bardziej abstrakcyjna, ponieważ zawiera mniejszą liczbę kolumn, które pozornie są czytelniejsze. Układ tabel pokazany na rysunku 1.4 był łatwy do zrozumienia, ponieważ tabela `score` zawierała kolumny zarówno dla daty, jak i kategorii wyniku. Obecnie tabela

Tabela score

name	event_id	score
Billy	5	15
Missy	5	14
Johnny	5	17
Jenny	5	19
Billy	6	67
Missy	6	73
Johnny	6	82
Jenny	6	79

Tabela grade_event

event_id	date	category
1	2012-09-03	Q
2	2012-09-06	Q
3	2012-09-09	T
4	2012-09-16	Q
5	2012-09-23	Q
6	2012-10-01	T

Rysunek 1.6. Tabele score i grade_event połączone przy użyciu identyfikatora zdarzenia

score, widoczna na rysunku 1.6, nie ma wymienionych kolumn. Wydaje się, że w ten sposób utrudniliśmy sobie wyszukiwanie informacji. Kto chce patrzeć na tabelę score zawierającą „identyfikator zdarzenia”? To przecież niezrozumiałe.

W tym momencie stanęliśmy na rozdrożu. Z jednej strony, pociąga Cię możliwość prowadzenia elektronicznego dziennika i uniknięcia konieczności ręcznego przeprowadzania obliczeń podczas wystawiania ocen końcowych. Z drugiej strony, po zapoznaniu się ze sposobem umieszczania informacji w bazie danych odstrasza Cię świadomość, jak abstrakcyjnie informacje są przechowywane w bazie danych.

W ten sposób docieramy do naturalnej wątpliwości: „Być może MySQL nie jest dla mnie? Czy nie lepszym rozwiązaniem będzie rezygnacja w ogóle z bazy danych?”. Jak możesz zgadnąć, odpowiadam negatywnie na oba pytania, ponieważ w przeciwnym razie ta książka mogłaby się skończyć w tym właśnie miejscu. Kiedy zastanawiasz się, jak wykonać zadanie, rozważenie wszelkich rozwiązań alternatywnych nie jest błędem, podobnie jak zadawanie sobie pytań o możliwość użycia innego systemu bazy danych niż MySQL lub czegoś zupełnie innego, na przykład arkusza kalkulacyjnego:

- Dziennik ocen składa się z wierszy i kolumn, podobnie jak arkusz kalkulacyjny. Dlatego też dziennik ocen i arkusz kalkulacyjny są pod względem koncepcji podobne do siebie.
- Arkusz kalkulacyjny przeprowadza obliczenia, a więc można łatwo obliczyć całkowitą liczbę punktów zebranych przez poszczególnych uczniów. Nieco trudniejsze będzie zastosowanie wagi dla poszczególnych wyników, ale na pewno da się to zrobić.

Z drugiej strony, jeśli chcesz przejrzeć tylko część danych (na przykład tylko sprawdziany lub tylko testy), przeprowadzić porównanie chłopcy kontra dziewczynki lub wyświetlić podsumowanie w elastyczny sposób, to całkowicie zmienia postać rzeczy. Arkusz kalkulacyjny w takich przypadkach nie sprawdzi się zbyt dobrze, natomiast system relacyjnej bazy danych bardzo łatwo może wykonać wymienione operacje.

Warto jednak pamiętać, że stosowany w relacyjnej bazie danych sposób umieszczania informacji oparty na ich abstrakcji i separacji tak naprawdę nie jest dużym problemem. O sposobie przedstawiania danych trzeba myśleć w trakcie konfiguracji bazy danych, aby nie rozmieścić ich w sposób pozbawiony sensu i utrudniający pracę z nimi. Jednak po rozmieszczeniu danych polegasz na silniku bazy danych, który staje się odpowiedzialny

za ich pobieranie i prezentację Tobie w użyteczny sposób. Nie będziesz patrzył na dane w postaci oddzielnych i niepowiązanych ze sobą fragmentów.

Na przykład, gdy pobierasz wyniki z tabeli `score`, chcesz widzieć daty, a nie identyfikatory zdarzeń. To żaden problem. Baza danych może odszukać daty w tabeli `grade_event` na podstawie identyfikatora zdarzenia, a następnie wyświetlić odpowiednie daty. Być może będziesz chciał wyświetlić wyniki jedynie sprawdzianów lub testów. To również nie stanowi żadnego problemu. Baza danych może wyszukać kategorie wyników w ten sam sposób, czyli na podstawie identyfikatora zdarzenia. Pamiętaj, że system bazy danych taki jak MySQL sprawdza się doskonale podczas łączenia danych ze sobą w celu pobrania informacji z wielu źródeł i wyświetlenia ich w żądany sposób. W przypadku wyników uzyskiwanych przez uczniów MySQL pobiera informacje na podstawie identyfikatora zdarzenia.

Teraz zaprezentuję przykład, jak nakazać bazie danych MySQL powiązanie danych w celu pobrania żądanych informacji. Przyjmijmy założenie, że chcesz wyświetlić wyniki uzyskane przez uczniów dnia 23 września 2012 roku. Zapytanie pobierające wyniki dla wskazanego dnia przedstawia się następująco:

```
SELECT score.name, grade_event.date, score.score, grade_event.category
FROM score INNER JOIN grade_event
ON score.event_id = grade_event.event_id
WHERE grade_event.date = '2012-09-23';
```

Wygląda przerażająco? Powyższe zapytanie pobiera imię ucznia, datę zdarzenia, uzyskany wynik i kategorię zdarzenia dzięki połączeniu (relacji) rekordów tabeli `score` z rekordami tabeli `grade_event`. Wyświetlony wynik przedstawia się następująco:

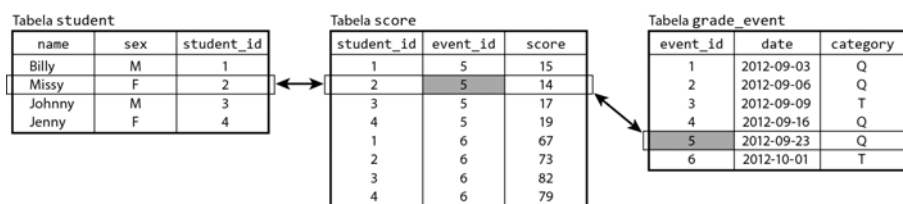
name	date	score	category
Billy	2012-09-23	15	Q
Missy	2012-09-23	14	Q
Johnny	2012-09-23	17	Q
Jenny	2012-09-23	19	Q

Czy zauważyłeś coś znajomego w formacie wyświetlonych informacji? Powinieneś, gdyż zastosowany format przypomina układ tabeli pokazany na rysunku 1.4! Aby otrzymać pokazane wyniki, nie musisz nawet znać identyfikatora zdarzenia. Po prostu podajesz interesującą Cię datę i pozwalasz bazie danych MySQL na wybór odpowiednich rekordów. Dlatego też, jeśli obawiałeś się, że abstrakcja i separacja danych spowoduje jakiegokolwiek straty podczas pobierania informacji z bazy danych i prezentacji ich w czytelnym formacie, to teraz już wiesz — byłeś w błędzie.

Oczywiście, przyglądając się zapytaniu, możesz martwić się jeszcze jedną kwestią: zapytanie wygląda na długie i skomplikowane. Czy tworzenie tego rodzaju zapytania w celu pobrania wyników dla wskazanego dnia nie jest zbyt pracochłonne? Tak, na pewno jest. Jednak istnieje wiele sposobów uniknięcia konieczności wprowadzania wielu wierszy kodu SQL za każdym razem, gdy chcesz wykonać zapytanie. Ogólnie rzecz biorąc, po ustaleniu sposobu wykonania zapytania takiego jak przedstawione powyżej przechowujesz je

w celu późniejszego łatwego ponownego wykonania. Do tego tematu powrócimy w podrozdziale 1.5, zatytułowanym „Wskazówki przydatne podczas pracy z klientem mysql”.

W rzeczywistości nieco się pośpieszyłem, pokazując poprzednie zapytanie. Możesz wierzyć lub nie, ale do pobierania wyników wykorzystamy nieco prostsze zapytanie niż wcześniej przedstawione. W układzie tabel trzeba wprowadzić jeszcze jedną zmianę. Zamiast przechowywać w tabeli score imię ucznia, umieścimy tam unikalny identyfikator ucznia. (Oznacza to użycie wartości z kolumny identyfikatora dziennika zamiast z kolumny imienia). Następnie utworzymy kolejną tabelę, o nazwie student, zawierającą kolumny identyfikatora ucznia (student_id) i jego imienia (name), jak pokazano na rysunku 1.7.



Rysunek 1.7. Tabele score, student i grade_event połączone identyfikatorami ucznia i zdarzenia

Dlaczego wprowadzamy tę modyfikację? W klasie kilku uczniów może mieć to samo imię. Dzięki użyciu unikalnego identyfikatora ucznia nie ma problemów z przechowywaniem ich wyników. (To analogiczne do przechowywania wyników sprawdzianów i testów przeprowadzonych jednego dnia, gdzie używany jest unikalny identyfikator zdarzenia zamiast data). Po wprowadzeniu wspomnianej zmiany w układzie tabel zapytanie używane do pobierania wyników uzyskanych przez uczniów we wskazanym dniu stanie się nieco bardziej skomplikowane:

```
SELECT student.name, grade_event.date, score.score, grade_event.category
FROM grade_event INNER JOIN score INNER JOIN student
ON grade_event.event_id = score.event_id
AND score.student_id = student.student_id
WHERE grade_event.date = '2012-09-23';
```

Nie powinieneś się niepokoić, jeśli nie rozumiesz w pełni sposobu działania powyższego zapytania. Większość osób tego nie rozumie. Do tego zapytania powrócimy w dalszej części przewodnika, ale różnica pomiędzy teraz i później jest taka, że później to zapytanie stanie się dla Ciebie całkowicie zrozumiałe. Nie, nie żartuję sobie.

Patrząc na rysunek 1.7, możesz dostrzec, że do tabeli student dodałem coś, czego nie znajdziesz w dzienniku papierowym: kolumnę wskazującą płeć. W ten sposób będzie można bardzo łatwo przeprowadzić operacje takie jak obliczenie liczby chłopców i dziewczynek w klasie lub porównanie wyników osiągniętych przez chłopców i dziewczynki.

Niemal skończyliśmy tworzenie tabel dla projektu ocen uczniów. Potrzebujemy jeszcze jednej tabeli w celu zapisu informacji o nieobecności uczniów. Zawartość wspomnianej tabeli

jest całkiem prosta: identyfikator ucznia i data nieobecności (patrz rysunek 1.8). Każdy wiersz tabeli wskazuje, że uczeń o podanym identyfikatorze był nieobecny danego dnia. Na koniec semestru wykorzystasz możliwości bazy danych MySQL w zakresie obliczeń i pozwolisz jej na automatyczne obliczenie liczby dni nieobecności poszczególnych uczniów.

Tabela absence

student_id	date
2	2012-09-02
4	2012-09-15
2	2012-09-20

Rysunek 1.8. Tabela absence

1.4.6.2.1. Tabela student

Skoro wiemy, jak powinny wyglądać tabele dla projektu ocen uczniów, to pora przystąpić do ich utworzenia. Zapytanie `CREATE TABLE` dla tabeli `student` przedstawia się następująco:

```
CREATE TABLE student
(
    name          VARCHAR(20) NOT NULL,
    sex           ENUM('F','M') NOT NULL,
    student_id    INT UNSIGNED NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (student_id)
) ENGINE=InnoDB;
```

Zwróć uwagę na dodanie czegoś nowego do zapytania `CREATE TABLE` — klauzuli `ENGINE` na końcu. Jej przeznaczenie wkrótce wyjaśnię.

Wpisz przedstawione powyżej zapytanie w kliencie `mysql` lub wydaj poniższe polecenie:

```
% mysql sampdb < create_student.sql
```

Zapytanie `CREATE TABLE` spowoduje utworzenie tabeli `student` wraz z trzema kolumnami o nazwach `name`, `sex` i `student_id`.

Kolumna `name` może przechowywać ciągi tekstowe o długości do 20 znaków. To prostsza wersja kolumny z projektu Ligi Historycznej; użyto jednej kolumny zamiast oddzielnych dla imienia i nazwiska. Przyjęto takie rozwiązanie, ponieważ z góry wiadomo, że żadne zapytania dotyczące ocen nie będą lepiej działały po zastosowaniu oddzielnych kolumn dla imienia i nazwiska. (Przyznaję, że trochę tutaj oszukuję. W rzeczywistości powinieneś stosować oddzielne kolumny dla imion i nazwisk).

Kolumna `sex` wskazuje płeć ucznia. Jej typ to `ENUM` (typ wyliczeniowy), co oznacza możliwość przyjęcia tylko jednej z wartości wymienionych w specyfikacji kolumny: `'F'` w przypadku dziewczynki i `'M'` w przypadku chłopca. Typ `ENUM` jest użyteczny, gdy zachodzi potrzeba ograniczenia wartości, które mogą być przechowywane w danej kolumnie. Wprowadzić można byłoby użyć typu `CHAR(1)`, ale `ENUM` znacznie dokładniej wskazuje dozwolone wartości. Jeżeli zapomnisz, jakie wartości można przechowywać w kolumnie,

wykonaj zapytanie DESCRIBE. W przypadku kolumny typu ENUM baza danych MySQL wyświetli dozwolone wartości typu wyliczeniowego:

```
mysql> DESCRIBE student 'sex';
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| sex   | enum('F','M') | NO   |     |         |       |
+-----+-----+-----+-----+-----+-----+
```

W kolumnie typu ENUM wartościami nie muszą być tylko pojedyncze znaki. Kolumna sex równie dobrze mogłaby zostać zdefiniowana w postaci ENUM('kobieta', 'mężczyzna').

Kolumna student_id przechowuje liczbę całkowitą będącą unikalnym identyfikatorem ucznia. Normalnie identyfikatory uczniów prawdopodobnie pobierasz z centralnego źródła, na przykład sekretariatu szkoły. W omawianym przykładzie generujemy je samodzielnie w kolumnie z przypisanym atrybutem AUTO_INCREMENT, zdefiniowanej w bardzo podobny sposób, jak kolumna member_id będąca częścią utworzonej wcześniej tabeli member.

Jeżeli identyfikatory uczniów faktycznie będą pobierane z sekretariatu szkoły zamiast generowane automatycznie w tabeli, wtedy kolumna student_id powinna być zdefiniowana bez atrybutu AUTO_INCREMENT. Pozostaw jednak klauzulę PRIMARY KEY, aby uniemożliwić tworzenie duplikatów lub użycie wartości NULL.

Powróćmy teraz do klauzuli ENGINE użytej na końcu zapytania CREATE TABLE. Jakie może być jej przeznaczenie? Jeżeli zostanie użyta, to wskazuje nazwę silnika, który baza danych MySQL powinna zastosować do utworzenia tej tabeli. Pojęcie „silnik bazy danych” oznacza mechanizm zarządzający określonego typu tabelą. W MySQL znajdziesz wiele różnych silników baz danych; każdy z nich ma pewne odmienne właściwości. Dwa najczęściej używane silniki w MySQL to InnoDB (domyślny w MySQL, począwszy od wersji 5.5) i MyISAM (domyślny w wersjach wcześniejszych niż 5.5).

Omówienie różnic pomiędzy wymienionymi silnikami znajdziesz w punkcie 2.6.1, zatytułowanym „Cechy charakterystyczne silników bazy danych”. W tym miejscu wystarczy wspomnieć, że w definicji tabel dla projektu ocen uczniów wyraźnie wskazano silnik InnoDB, ponieważ potrzebujemy obsługi oferowanej przez InnoDB funkcji o nazwie *referential integrity*, aby używać kluczy zewnętrznych. Oznacza to możliwość użycia MySQL w celu wymuszenia pewnych ograniczeń w relacjach *między* obiektami, co jest bardzo ważne w tabelach projektu ocen uczniów:

- Rekordy wyników są powiązane z ocenianymi zdarzeniami i uczniami. Nie chcemy zezwolić na wstawienie rekordu do tabeli score, jeśli w tabelach student i grade_event nie są umieszczone odpowiednie wartości identyfikatorów ucznia i zdarzenia.
- Podobnie, rekordy zawierające informacje o nieobecnościach są powiązane z uczniami. Nie chcemy zezwolić na wstawienie rekordu do tabeli absence, jeśli w tabeli student nie znajduje się odpowiednia wartość identyfikatora ucznia.

Aby wymusić stosowanie tego rodzaju ograniczeń, konieczne jest skonfigurowanie relacji klucza zewnętrznego. W przypadku bazy danych pojęcie „zewnętrzny” oznacza „w innej tabeli”, natomiast „klucz zewnętrzny” wskazuje wartość, która musi być dopasowana do wartości klucza w innej tabeli. Te koncepcje staną się jasne po utworzeniu pozostałych tabel projektu ocen uczniów.

We wcześniejszej części przewodnika, podczas tworzenia tabel dla projektu Ligi Historycznej (president i member), nie zastosowaliśmy klauzuli ENGINE. Dlatego też wymienione tabele zostały utworzone przy użyciu domyślnego silnika bazy danych w serwerze. Jak wcześniej wspomniano, to będzie InnoDB (o ile konfiguracja serwera nie została zmodyfikowana). W definicji tabeli student umieszczono klauzulę ENGINE = InnoDB na wypadek, gdyby konfiguracja serwera została zmieniona i domyślnie był używany inny silnik bazy danych.

1.4.6.2.2. Tabela grade_event

Poniżej przedstawiono definicję tabeli grade_event:

```
CREATE TABLE grade_event
(
    date      DATE NOT NULL,
    category  ENUM('T','Q') NOT NULL,
    event_id  INT UNSIGNED NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (event_id)
) ENGINE=InnoDB;
```

W celu utworzenia tabeli grade_event wpisz powyższe zapytanie w kliencie mysql lub wydaj poniższe polecenie:

```
% mysql sampdb < create_grade_event.sql
```

Kolumna date przechowuje standardową wartość MySQL DATE w formacie 'CCYY-MM-DD' (rok podawany na początku).

Kolumna category przedstawia kategorię wyniku. Podobnie jak kolumna sex w tabeli student, także kolumna category jest typu wyliczeniowego. Dozwolonymi wartościami są 'T' i 'Q', oznaczające odpowiednio test i sprawdzian.

Kolumna event_id ma przypisany atrybut AUTO_INCREMENT i została zdefiniowana wraz z klauzulą PRIMARY KEY, czyli podobnie do kolumny student_id w tabeli student. Użycie atrybutu AUTO_INCREMENT pozwala na łatwe wygenerowanie unikalnych wartości identyfikatorów. Podobnie jak w przypadku kolumny student_id w tabeli student, także tutaj konkretna wartość identyfikatora jest mniej istotna niż jego unikalność.

Wszystkie kolumny zdefiniowano jako NOT NULL, ponieważ wszystkie wartości są niezbędne.

1.4.6.2.3. Tabela score

Poniżej przedstawiono definicję tabeli score:

```
CREATE TABLE score
(
    student_id INT UNSIGNED NOT NULL,
    event_id   INT UNSIGNED NOT NULL,
```

```

score      INT NOT NULL,
PRIMARY KEY (event_id, student_id),
INDEX (student_id),
FOREIGN KEY (event_id) REFERENCES grade_event (event_id),
FOREIGN KEY (student_id) REFERENCES student (student_id)
) ENGINE=InnoDB;

```

Także i ta definicja tabeli zawiera coś nowego: konstrukcję `FOREIGN KEY`. Powrócimy do niej za chwilę.

W celu utworzenia tabeli `score` wpisz powyższe zapytanie bezpośrednio w kliencie `mysql` lub wydaj poniższe:

```
% mysql sampdb < create_score.sql
```

Kolumna `score` jest typu `INT` i przechowuje wartości w postaci liczb całkowitych. Aby umożliwić przechowywanie wartości takich jak 58.5 (czyli z częścią dziesiętną), trzeba użyć innego typu danych, pozwalającego na obsługę liczb zmiennoprzecinkowych, na przykład `DECIMAL`.

Kolumny `student_id` i `event_id` przechowują liczby całkowite będące identyfikatorami ucznia i zdarzenia, których dotyczą dane wyniki. Dzięki użyciu wymienionych kolumn i połączeniu z odpowiednimi wartościami w tabelach `student` i `grade_event` mamy możliwość ustalenia imienia ucznia i daty zdarzenia. W tym miejscu trzeba jeszcze wspomnieć o kilku ważnych punktach dotyczących kolumn `student_id` i `event_id`:

- Wykorzystywane jest połączenie dwóch kolumn `PRIMARY KEY`. W ten sposób mamy pewność uniknięcia duplikatów wyników ucznia dla danego sprawdzianu lub testu. Zwróć uwagę, że połączenie `event_id` i `student_id` jest unikalne. W tabeli `score` żadna wartość z definicji nie jest unikalna. Dla każdej wartości `event_id` będzie wiele rekordów (po jednym na każdego ucznia), a także wiele rekordów wartości `student_id` (po jednym dla danego sprawdzianu lub testu) na ucznia.
- Dla każdej kolumny identyfikatora klauzula `FOREIGN KEY` definiuje ograniczenie. Część `REFERENCES` klauzuli wskazuje tabelę i kolumnę, do której odwołuje się kolumna identyfikatora. Ograniczenie w `event_id` określa, że każda wartość kolumny musi dopasować pewną wartość `event_id` w tabeli `grade_event`. Podobnie, każda wartość `student_id` w tabeli `score` musi być dopasowana do wartości `student_id` w tabeli `student`.

Definicja `PRIMARY KEY` gwarantuje, że nie nastąpi utworzenie powielonych rekordów wyników. Z kolei definicje `FOREIGN KEY` gwarantują, że nie będziemy mieli rekordów z nieprawdziwymi wartościami identyfikatorów, które nie istnieją w tabelach `grade_event` i `student`.

Dlaczego indeks jest w kolumnie `student_id`? Powód jest prosty: każda kolumna zdefiniowana jako `FOREIGN KEY` powinna mieć indeks; w przypadku indeksów obejmujących wiele kolumn powinna być umieszczona jako pierwsza, ponieważ to przyspiesza operację wyszukiwania. W przypadku definicji `FOREIGN KEY` w kolumnie `student_id` nie można użyć klauzuli `PRIMARY KEY`, ponieważ kolumna `student_id` nie została umieszczona jako pierwsza. Dlatego też tworzymy oddzielny indeks w `student_id`.

Jeśli będzie to konieczne, silnik InnoDB automatycznie tworzy indeks w kolumnach klucza zewnętrznego, ale nie musi używać tej samej definicji indeksu, której Ty byś użył (dokładniejsze omówienie tego zagadnienia znajdziesz w podrozdziale 2.13, zatytułowanym „Klucze zewnętrzne i integralność odwołań”). Samodzielne zdefiniowanie indeksu pozwala na uniknięcie problemów.

1.4.6.2.4. Tabela absence

Utworzenie tabeli absence, przeznaczonej do przechowywania nieobecności, następuje po wykonaniu poniższego zapytania:

```
CREATE TABLE absence
(
    student_id INT UNSIGNED NOT NULL,
    date       DATE NOT NULL,
    PRIMARY KEY (student_id, date),
    FOREIGN KEY (student_id) REFERENCES student (student_id)
) ENGINE=InnoDB;
```

Powyższe zapytanie wpisz bezpośrednio w kliencie mysql lub wydaj poniższe polecenie:

```
% mysql sampdb < create_absence.sql
```

Kolumny student_id i date zostały zdefiniowane jako NOT NULL, ponieważ wymagane jest podanie ich wartości. Obie wymienione kolumny razem tworzą klucz podstawowy, co chroni przed przypadkowym utworzeniem powielających się rekordów. Byłoby nie w porządku, gdyby policzyć uczniowi dwukrotnie nieobecność danego dnia.

Tabela absence zawiera również klucz zewnętrzny zdefiniowany w celu zagwarantowania, że każda wartość student_id będzie dopasowana do wartości student_id w tabeli student.

Umieszczenie relacji klucza zewnętrznego w tabelach projektu ocen uczniów ma nieustanny wpływ na dane: chcemy wstawić tylko te rekordy, które zawierają prawidłową wartość ocenianego zdarzenia i identyfikatora ucznia. Jednak relacja klucza zewnętrznego niesie ze sobą jeszcze inny efekt. Powoduje konfigurację zależności nakładających ograniczenia na kolejność, w której można tworzyć i usuwać tabele:

- Tabela score odwołuje się do tabel grade_event i student, więc muszą być one utworzone przed nią. Podobnie, tabela absence odwołuje się do tabeli student, która musi istnieć, zanim przystąpisz do tworzenia tabeli absence.
- Jeżeli chcesz usunąć tabele, to należy zachować kolejność odwrotną do wymienionej powyżej. Nie można usunąć tabeli grade_event, jeśli wcześniej nie usunięto tabeli score. Podobnie, nie można usunąć tabeli student przed wcześniejszym usunięciem tabel score i absence.

1.4.7. Dodawanie nowych rekordów

Na tym etapie nasza baza danych i jej tabele zostały utworzone. Możemy więc przystąpić do umieszczenia pewnych rekordów w tabelach. Wcześniej warto jednak dowiedzieć się, jak sprawdzić zawartość tabeli po umieszczeniu w niej pewnych danych. Wprawdzie

pobieranie danych z tabeli zostanie omówione w punkcie 1.4.9, zatytułowanym „Pobieranie informacji”, ale teraz wystarczy wiedzieć, że poniższe zapytanie wyświetli pełną zawartość tabeli o wskazanej nazwie:

```
SELECT * FROM nazwa_tabeli;
```

Na przykład:

```
mysql> SELECT * FROM student;
Empty set (0.00 sec)
```

Obecnie klient mysql wskazuje, że tabela student jest pusta. Po wypróbowaniu różnych przykładów przedstawionych w tym punkcie wykonanie powyższego zapytania wyświetli zupełnie inne wyniki.

Istnieje wiele sposobów wstawiania danych do bazy danych. Rekordy możesz wstawiać ręcznie do tabeli, wykonując kolejne zapytanie INSERT. Inna możliwość to dodanie rekordów przez ich odczyt z pliku. Wspomniany plik może zawierać przygotowane zapytania INSERT przekazywane klientowi mysql bądź po prostu same dane wczytywane przy użyciu zapytania LOAD DATA lub w programie klienta mysql import.

W tym punkcie zostaną przedstawione obie wymienione metody wstawiania rekordów do tabel. Powinieneś wypróbować je samodzielnie, aby poznać ich sposób działania. Po wypróbowaniu obu metod przejdź do punktu 1.4.8, zatytułowanego „Przywrócenie bazy danych sampdb do znanego stanu”, i wykonaj przedstawione tam zapytania. Wspomniane zapytania powodują usunięcie tabel, ponowne ich utworzenie, a następnie wczytanie przygotowanych danych. Wykonując zapytania przedstawione w punkcie 1.4.8, masz gwarancję pracy z tabelami zawierającymi takie same rekordy, z jakimi ja pracowałem, pisząc tę książkę, a więc otrzymasz wyniki identyczne z prezentowanymi w tekście. (Jeżeli wiesz, jak wstawiać rekordy, i jedynie chcesz wypełnić tabele danymi, od razu możesz przejść do punktu 1.4.8.).

1.4.7.1. Wstawianie rekordu przy użyciu zapytania INSERT

Rozpocniemy od wstawiania rekordów przy użyciu INSERT, czyli zapytania SQL pozwalającego na wskazanie tabeli, do której będzie wstawiany rekord danych oraz odpowiednich wartości. Zapytanie INSERT jest spotykane w kilku formatach.

Podanie wartości dla wszystkich kolumn. Składnia tego rodzaju zapytania przedstawia się następująco:

```
INSERT INTO nazwa_tabeli VALUES(wartość1, wartość2,...);
```

Na przykład:

```
mysql> INSERT INTO student VALUES('Kyle','M',NULL);
mysql> INSERT INTO grade_event VALUES('2012-09-03','Q',NULL);
```

W przypadku powyższej składni lista wartości (VALUES) musi zawierać wartości dla każdej kolumny w tabeli, z zachowaniem kolejności, w jakiej tabele są przechowywane w bazie danych. (Zwykle oznacza to kolejność kolumn użytą w zapytaniu CREATE TABLE). Jeżeli nie jesteś pewien kolejności tabel, wykonaj zapytanie DESCRIBE *nazwa_tabeli* i sprawdź.

Ciągi tekstowe i datę możesz cytować w MySQL, ujmując wartość w cudzysłów lub apostrofy, ale najczęściej stosuje się apostrofy. Wartości NULL są dla kolumn AUTO_INCREMENT w tabelach student i grade_event. Wstawienie „pominiętej wartości” w kolumnie zdefiniowanej jako AUTO_INCREMENT powoduje, że MySQL wygeneruje kolejny numer w sekwencji dla tej kolumny.

Baza danych MySQL pozwala na wstawienie wielu rekordów do tabeli przy pomocy tylko jednego zapytania INSERT i wielu list wartości:

```
INSERT INTO nazwa_tabeli VALUES(...),(...),... ;
```

Na przykład:

```
mysql> INSERT INTO student VALUES('Avery','F',NULL),('Nathan','M',NULL);
```

W ten sposób można zaoszczędzić sobie wpisywania wielu zapytań INSERT, a dla serwera będzie to znacznie efektywniejsze rozwiązanie. Zwróć uwagę na wymaganą parę nawiasów, w które ujęte są wartości kolumn dla *każdego* rekordu. Poniższe zapytanie jest nieprawidłowe z powodu błędnego użycia nawiasów:

```
mysql> INSERT INTO student VALUES('Avery','F',NULL,'Nathan','M',NULL);
ERROR 1136 (21S01): Column count doesn't match value count at row 1
```

Wskazanie kolumn, którym zostaną przypisane wartości, i podanie tych wartości. To jest użyteczny sposób, jeśli chcesz utworzyć rekord, w którym na początku wartości zostaną przypisane wybranym kolumnom.

```
INSERT INTO nazwa_tabeli (nazwa_kolumny1, nazwa_kolumny2,...) VALUES(wartość1,
wartość2,...);
```

Na przykład:

```
mysql> INSERT INTO member (last_name,first_name) VALUES('Stein','Waldo');
```

Ta forma również obsługuje wiele list wartości:

```
mysql> INSERT INTO student (name,sex) VALUES('Abby','F'),('Joseph','M');
```

Każdej kolumnie niewymienionej na liście kolumn MySQL przypisuje wartość domyślną. Na przykład, powyższe zapytania nie zawierały wartości dla kolumn member_id i student_id, więc MySQL przypisze im wartość domyślną NULL. Kolumny member_id i student_id zostały zdefiniowane jako AUTO_INCREMENT, więc skutkiem będzie wygenerowanie i przypisanie kolejnej wartości w sekwencji, podobnie jak w przypadku wyraźnego użycia wartości NULL.

Podanie listy przypisań kolumna-wartość. Ta składnia zamiast listy wartości VALUES() używa klauzuli SET zawierającej przypisania nazwa_kolumny=wartość:

```
INSERT INTO nazwa_tabeli SET nazwa_kolumny1=wartość1, nazwa_kolumny2=wartość2, ... ;
```

Na przykład:

```
mysql> INSERT INTO member SET last_name='Stein',first_name='Waldo';
```

Każdej kolumnie niewymienionej w klauzuli SET baza danych MySQL przypisuje wartość domyślną. Tej formy zapytania INSERT nie można wykorzystać do wstawienia wielu rekordów przy pomocy tylko jednego zapytania.

Skoro poznałeś sposób działania zapytania INSERT, to możemy go użyć i przekonać się, czy zdefiniowana przez nas relacja klucza zewnętrznego naprawdę uniemożliwi wstawienie nieprawidłowych rekordów do tabel `score` i `absence`. Spróbuj więc wstawić rekordy zawierające wartości identyfikatorów nieistniejących w tabelach `grade_event` lub `student`:

```
mysql> INSERT INTO score (event_id,student_id,score) VALUES(9999,9999,0);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key
constraint fails ('sampdb'.score', CONSTRAINT 'score_ibfk_1' FOREIGN
KEY ('event_id') REFERENCES 'grade_event' ('event_id'))
mysql> INSERT INTO absence SET student_id=9999, date='2012-09-16';
ERROR 1452 (23000): Cannot add or update a child row: a foreign key
constraint fails ('sampdb'.absence', CONSTRAINT 'absence_ibfk_1'
FOREIGN KEY ('student_id') REFERENCES 'student' ('student_id'))
```

Wyświetlony komunikat błędu pokazuje, że zdefiniowane ograniczenia spełniają swoje role.

1.4.7.2. Wstawienie nowych rekordów z pliku

Inną metodą wstawienia rekordów w tabeli jest ich wczytanie bezpośrednio z pliku. Wspomniany plik może zawierać odpowiednio spreparowane zapytania INSERT lub po prostu zwykłe dane. Na przykład, dystrybucja `sampdb` zawiera plik o nazwie `insert_president.sql`, w którym znajdują się zapytania INSERT wstawiające nowe rekordy do tabeli `president`. Przyjmując założenie, że plik z danymi znajduje się w katalogu roboczym, dane możesz wczytać z pliku, wydając poniższe polecenie:

```
% mysql sampdb < insert_president.sql
```

Jeżeli pracujesz już w uruchomionym kliencie `mysql`, do wczytania zawartości pliku użyj zapytania SOURCE:

```
mysql> SOURCE insert_president.sql;
```

W przypadku, gdy rekordy danych są przechowywane w pliku w postaci zwykłych wartości zamiast odpowiednio spreparowanych zapytań INSERT, możesz je wczytać przy użyciu zapytania `LOAD DATA` lub programu klienckiego `mysql import`.

Zapytanie `LOAD DATA` wczytuje dane odczytane z pliku. Wymienione zapytania używasz z poziomu klienta `mysql`:

```
mysql> LOAD DATA LOCAL INFILE 'member.txt' INTO TABLE member;
```

Przyjmując założenie, że plik danych `member.txt` znajduje się w katalogu roboczym komputera, powyższe zapytanie odczyta zawartość pliku, przekaże ją do serwera, który z kolei umieści dane w tabeli `member`. (Plik `member.txt` również znajdziesz w dystrybucji `sampdb`).

Domyślnie zapytanie `LOAD DATA` uznaje, że wartości kolumn są oddzielone tabulatorami, a wiersz kończy się znakiem nowego wiersza (nazywanym również „wysuw wiersza”). Przyjmowane jest również założenie, że wartości znajdują się w kolejności kolumn przechowywanych w tabeli (N w pliku oznacza wartość NULL). Istnieje możliwość odczytu plików w innych formatach, a także wskazania innej kolejności kolumn. Więcej informacji na ten temat znajdziesz w dodatku E, zatytułowanym „Przewodnik po składni SQL”.

Słowo kluczowe `LOCAL` w zapytaniu `LOAD DATA` powoduje, że plik danych będzie odczytany przez program klienta (w omawianym przypadku to `mysql`) i przekazany do serwera, który wstawi dane. Istnieje możliwość pominięcia słowa kluczowego `LOCAL`, ale wówczas plik musi znajdować się w komputerze, w którym działa serwer bazy danych. Ponadto, wymagane są uprawnienia dostępu do serwera plików, których to uprawnień większość użytkowników MySQL po prostu nie posiada. Trzeba również podać pełną ścieżkę dostępu do pliku, aby serwer mógł go zlokalizować.

Jeżeli po wykonaniu zapytania `LOAD DATA LOCAL` otrzymasz poniższy komunikat błędu, oznacza to, że domyślnie wyłączona jest możliwość użycia słowa kluczowego `LOCAL`:

```
ERROR 1148 (42000): The used command is not allowed with this MySQL version
```

W takim przypadku spróbuj uruchomić klienta `mysql` wraz z opcją `--local-infile`, na przykład:

```
% mysql --local-infile sampdb
mysql> LOAD DATA LOCAL INFILE 'member.txt' INTO TABLE member;
```

Jeśli powyższa metoda nie działa, serwer bazy danych trzeba uruchomić wraz z opcją `--local-infile`.

Innym sposobem wczytania danych z pliku jest użycie programu klienckiego `mysqlimport` z poziomu wiersza poleceń. Ten program automatycznie generuje zapytanie `LOAD DATA`:

```
% mysqlimport --local sampdb member.txt
```

Podobnie jak w przypadku klienta `mysql`, jeśli konieczne jest podanie parametrów połączenia, to należy je umieścić w wierszu poleceń przed nazwą bazy danych.

Dla przedstawionego powyżej zapytania narzędzie `mysqlimport` generuje zapytanie `LOAD DATA` w celu wczytania zawartości pliku `member.txt` do tabeli `member`. Program `mysqlimport` określa nazwę tabeli docelowej na podstawie nazwy pliku danych — jako nazwę tabeli przyjmuje wszystkie znaki aż do pierwszej kropki w nazwie. Na przykład, program `mysqlimport` wczyta zawartość plików `member.txt` i `president.txt` do tabel odpowiednio `member` i `president`. Oznacza to konieczność starannego wybierania nazw plików, ponieważ w przeciwnym razie `mysqlimport` nie zaimportuje danych do właściwej tabeli. Jeżeli spróbujesz wczytać zawartość plików `member1.txt` i `member2.txt`, program `mysqlimport` uzna, że powinien wczytać zawartość wymienionych plików do tabel `member1` i `member2`. Jeżeli faktycznie zawartość obu wymienionych plików chcesz umieścić w tabeli `member`, nadaj im nazwy takie jak `member.1.txt` i `member.2.txt` lub `member.txt1` i `member.txt2`.

1.4.8. Przywrócenie bazy danych `sampdb` do znanego stanu

Po wypróbowaniu przedstawionych w poprzednim punkcie metod wstawiania rekordów powinieneś ponownie utworzyć bazę danych `sampdb` i wczytać tabele, aby jej zawartość była taka sama, jaką przyjęto w dalszej części książki. Używając programu `mysql` uruchomionego z poziomu katalogu zawierającego pliki dystrybucji `sampdb`, wykonaj wymienione poniżej zapytania:

```
% mysql sampdb
mysql> SOURCE create_member.sql;
mysql> SOURCE create_president.sql;
mysql> SOURCE insert_member.sql;
mysql> SOURCE insert_president.sql;
mysql> DROP TABLE IF EXISTS absence, score, grade_event, student;
mysql> SOURCE create_student.sql;
mysql> SOURCE create_grade_event.sql;
mysql> SOURCE create_score.sql;
mysql> SOURCE create_absence.sql;
mysql> SOURCE insert_student.sql;
mysql> SOURCE insert_grade_event.sql;
mysql> SOURCE insert_score.sql;
mysql> SOURCE insert_absence.sql;
```

Jeśli wymienionych powyżej poleceń nie chcesz wpisywać ręcznie, w systemie UNIX wydaj poniższe polecenie:

```
% sh init_all_tables.sh sampdb
```

Natomiast jeżeli pracujesz w Windows, wtedy należy wydać polecenie:

```
C:\> init_all_tables.bat sampdb
```

Niezależnie od wydawanego polecenia, jeśli konieczne jest podanie parametrów połączenia, musisz je wymienić w wierszu polecenia po nazwie wydawanego polecenia.

1.4.9. Pobieranie informacji

Tabele robocze zostały już utworzone i wypełnione danymi, więc przekonajmy się, co można zrobić ze wspomnianymi danymi. Aby pobrać i wyświetlić informacje, używane jest zapytanie SELECT. Pozwala ono na pobranie informacji w sposób ogólny lub w dowolnie wybrany. Cała zawartość tabeli jest wyświetlana po wykonaniu poniższego zapytania:

```
SELECT * FROM president;
```

Ewentualnie możesz wyświetlić tylko pojedynczą kolumnę jednego rekordu, na przykład:

```
SELECT birth FROM president WHERE last_name = 'Eisenhower';
```

Zapytanie SELECT ma wiele klauzul, których połączenie pozwala na pobranie żądanych informacji. Każda z klauzul może być prosta lub złożona, a tym samym całe zapytanie SELECT stanie się proste lub skomplikowane. Jednak w tej książce nie znajdziesz zapytań SELECT obejmujących całą stronę i wymagających godziny do zrozumienia jego działania. Kiedy widzę ogromne zapytania w czytanych tekstach, to najczęściej je pomijam. Zgaduję, że robisz dokładnie tak samo.

Uproszczona składnia zapytania SELECT przedstawia się następująco:

```
SELECT  informacje do pobrania
FROM    tabela lub tabele
WHERE   warunki, które muszą być spełnione przez dane;
```


Aby utworzyć zapytanie SELECT, musisz wskazać informacje do pobrania, a następnie opcjonalne klauzule. Wymienione powyżej klauzule FROM i SELECT są najczęściej stosowane, choć można użyć także innych, na przykład GROUP BY, ORDER BY i LIMIT. Pamiętaj, że SQL to język o swobodnym formacie i podczas tworzenia własnych zapytań SELECT nie wolno Ci umieszczać znaków nowego wiersza w miejscach, które widzisz w książce.

Klauzula FROM zwykle jest stosowana, ale możesz ją pominąć, jeśli nie trzeba wymieniać nazwy żadnej tabeli. Na przykład, przedstawione poniżej zapytanie po prostu wyświetla wartości pewnych wyrażeń. Mogą zostać obliczone bez odwoływania się do jakiegokolwiek tabeli, stąd brak konieczności użycia klauzuli FROM:

```
mysql> SELECT 2+2, 'Witaj, świecie', VERSION();
+-----+
| 2+2 | Witaj, świecie | VERSION() |
+-----+
| 4 | Witaj, świecie | 5.5.30-log |
+-----+
```

Kiedy używasz klauzuli FROM w celu wskazania tabeli, z której mają pochodzić dane, wtedy określasz także kolumny przeznaczone do wyświetlenia w danych wyjściowych. Najbardziej „ogólna” postać zapytania SELECT używa gwiazdki (*) do wskazania kolumny, co jest skrótem oznaczającym „wszystkie kolumny”. Przedstawione poniżej zapytanie pobiera i wyświetla wszystkie kolumny tabeli student:

```
mysql> SELECT * FROM student;
+-----+
| name | sex | student_id |
+-----+
| Megan | F | 1 |
| Joseph | M | 2 |
| Kyle | M | 3 |
| Katie | F | 4 |
...
```

Kolumny są wyświetlane w kolejności ich przechowywania w tabeli bazy danych MySQL. To jest dokładnie ta sama kolejność, w której kolumny są wyświetlane po wykonaniu zapytania DESCRIBE student. (Wielokropek pokazany na końcu danych wyjściowych oznacza, że wyświetlonych na ekranie danych wyjściowych jest znacznie więcej, niż przedstawiono w książce).

Istnieje możliwość dokładnego wskazania nazwy kolumny lub kolumn, które mają być wyświetlone. W celu wyświetlenia jedynie imion uczniów wykonaj poniższe zapytanie:

```
mysql> SELECT name FROM student;
+-----+
| name |
+-----+
| Megan |
| Joseph |
| Kyle |
| Katie |
...
```

Jeżeli chcesz wymienić nazwy więcej niż tylko jednej kolumny, rozdziel je przecinkami. Poniższe zapytanie jest odpowiednikiem zapytania `SELECT * FROM student;`, ale wyraźnie wymienia wszystkie kolumny przeznaczone do wyświetlenia:

```
mysql> SELECT name, sex, student_id FROM student;
```

```
+-----+-----+-----+
| name   | sex  | student_id |
+-----+-----+-----+
| Megan  | F    | 1           |
| Joseph | M    | 2           |
| Kyle   | M    | 3           |
| Katie  | F    | 4           |
...

```

Kolumny można wymienić w dowolnej kolejności:

```
SELECT name, student_id FROM student;
SELECT student_id, name FROM student;
```

Jeżeli chcesz, to nazwę kolumny możesz wymienić więcej niż tylko jednokrotnie, choć ogólnie rzecz biorąc, jest to bezcelowe.

Istnieje nawet możliwość wybrania kolumn z więcej niż tylko jednej tabeli. Taka operacja nosi nazwę „złączenia” tabel. Złączenia zostaną omówione w punkcie 1.4.9.10, zatytułowanym „Pobieranie informacji z wielu tabel”.

W MySQL nazwy kolumn nie rozróżniają wielkości znaków i dlatego wszystkie wymienione poniżej zapytania pobiorą te same informacje:

```
SELECT name, student_id FROM student;
SELECT NAME, STUDENT_ID FROM student;
SELECT nAmE, sTuDeNt_Id FROM student;
```

Jednak nazwy baz danych i tabel mogą rozróżniać wielkość liter. Wszystko zależy od systemu plików zastosowanego w komputerze, w którym został uruchomiony serwer, oraz od samej konfiguracji MySQL. Nazwy plików w Windows nie rozróżniają wielkości liter, a tym samym nazwy baz danych i tabel również. Z kolei w systemach UNIX system plików zwykle rozróżnia wielkość liter i dlatego serwer MySQL również będzie rozróżniał wielkość liter w nazwach baz danych i tabel. Wyjątkiem jest tutaj system plików HFS+ (platforma OS X), który nie rozróżnia wielkości liter.

Jeżeli nie chcesz, aby baza danych MySQL rozróżniała wielkość liter w nazwach baz danych i tabel, możesz ją odpowiednio skonfigurować. Więcej informacji na ten temat znajdziesz w punkcie 11.2.6, zatytułowanym „Ograniczenia systemu operacyjnego w zakresie nazw obiektów bazy danych”.

1.4.9.1. Określenie kryteriów pobierania informacji

Aby ograniczyć zestaw rekordów pobieranych przez zapytanie `SELECT`, użyj klauzuli `WHERE` wraz z kryteriami, które muszą być spełnione przez wartości kolumn. Na przykład, w celu wyszukania zakresu wartości liczbowych możesz wykonać poniższe zapytanie:

```
mysql> SELECT * FROM score WHERE score > 95;
```

student_id	event_id	score
5	3	97
18	3	96
1	6	100
5	6	97
11	6	98
16	6	98

Możesz wyszukać ciągu tekstowe zawierające dane znakowe. W przypadku domyślnego zestawu znaków i kolejności sortowania operacja porównania ciągów tekstowych nie rozróżnia wielkości znaków:

```
mysql> SELECT last_name, first_name FROM president
-> WHERE last_name='ROOSEVELT';
```

last_name	first_name
Roosevelt	Theodore
Roosevelt	Franklin D.

```
mysql> SELECT last_name, first_name FROM president
-> WHERE last_name='roosevelt';
```

last_name	first_name
Roosevelt	Theodore
Roosevelt	Franklin D.

Istnieje również możliwość wyszukania daty:

```
mysql> SELECT last_name, first_name, birth FROM president
-> WHERE birth < '1750-1-1';
```

last_name	first_name	birth
Washington	George	1732-02-22
Adams	John	1735-10-30
Jefferson	Thomas	1743-04-13

Inna możliwość to wyszukanie połączenia wartości:

```
mysql> SELECT last_name, first_name, birth, state FROM president
-> WHERE birth < '1750-1-1' AND (state='VA' OR state='MA');
```

last_name	first_name	birth	state
Washington	George	1732-02-22	VA
Adams	John	1735-10-30	MA
Jefferson	Thomas	1743-04-13	VA

Wyrażenia w klauzuli WHERE mogą stosować operatory arytmetyczne (patrz tabela 1.1), operatory porównania (patrz tabela 1.2) i operatory logiczne (patrz tabela 1.3). Do grupowania części wyrażenia stosowane są nawiasy. Operacje mogą być przeprowadzane z użyciem stałych, kolumn tabel i wywołań funkcji. W tym przewodniku będziemy mieli okazję użyć kilku funkcji MySQL w zapytaniach, ale jest ich zbyt wiele, aby wszystkie tutaj wymienić. Zapoznaj się z dodatkiem C, zatytułowanym „Przewodnik po operatorach i funkcjach”.

Tabela 1.1. Operatory arytmetyczne

Operator	Opis
+	Dodawanie
-	Odejmowanie
*	Mnożenie
/	Dzielenie
DIV	Dzielenie liczby całkowitej
%	Modulo (reszta z dzielenia)

Tabela 1.2. Operatory porównania

Operator	Opis
<	Mniejszy niż
<=	Mniejszy niż lub równy
=	Równy
<=>	Równy (działa nawet z wartościami NULL)
<> lub !=	Nierówny
>=	Większy niż lub równy
>	Większy niż

Tabela 1.3. Operatory logiczne

Operator	Opis
AND	Logiczne AND
OR	Logiczne OR
XOR	Logiczne XOR
NOT	Logiczna negacja

Kiedy przygotowujesz zapytania wymagające operatorów logicznych, zachowaj szczególną ostrożność, aby nie pomylić znaczenia operatora logicznego AND ze sposobem, w jaki

codziennie używamy spójnika *i* (ang. *and*). Przyjmujemy założenie, że chcemy znaleźć „prezydentów urodzonych w stanie Virginia i prezydentów urodzonych w stanie Massachusetts”. Użycie spójnika „i” w poprzednim zdaniu sugeruje, że chcemy utworzyć poniższe zapytanie:

```
mysql> SELECT last_name, first_name, state FROM president
-> WHERE state='VA' AND state='MA';
Empty set (0.01 sec)
```

Pusty zbiór danych wyjściowych bez wątplenia oznacza, że zapytanie nie działa. Dlaczego? Ponieważ powyższe zapytanie tak naprawdę oznacza „wybierz prezydentów urodzonych w stanie Virginia i Massachusetts”, co jest bezsensowne. W języku polskim nasze żądanie można wyrazić z użyciem spójnika „i”, natomiast w SQL trzeba dwa warunki połączyć operatorem OR (z ang. *lub*):

```
mysql> SELECT last_name, first_name, state FROM president
-> WHERE state='VA' OR state='MA';
```

last_name	first_name	state
Washington	George	VA
Adams	John	MA
Jefferson	Thomas	VA
Madison	James	VA
Monroe	James	VA
Adams	John Quincy	MA
Harrison	William H.	VA
Tyler	John	VA
Taylor	Zachary	VA
Wilson	Woodrow	VA
Kennedy	John F.	MA
Bush	George H.W.	MA

O wspomnianej różnicy pomiędzy językami naturalnym i SQL należy pamiętać nie tylko podczas tworzenia własnych zapytań, ale również w trakcie przygotowywania zapytań dla innych osób. Najlepiej uważnie słuchać, gdy opisują dane, które chcą otrzymać, ale niekoniecznie bezpośrednio konwertować ten opis na zapytanie w języku SQL z użyciem tych samych operatorów logicznych. W przypadku przedstawionego wcześniej zapytania prawidłowe jego wyrażenie w języku polskim brzmi „wybierz prezydentów, którzy urodzili się w stanie Virginia lub w stanie Massachusetts”.

W trakcie tworzenia zapytań wyszukujących kilka pojedynczych wartości łatwiejsze może być użycie operatora IN(). Poprzednie zapytanie, ale w wersji z operatorem IN(), przedstawia się następująco:

```
SELECT last_name, first_name, state FROM president
WHERE state IN('VA', 'MA');
```

Operator IN() jest szczególnie wygodny podczas porównywania kolumny do ogromnej liczby wartości.

1.4.9.2. Wartość NULL

NULL to wartość specjalna. Oznacza „brak wartości” lub „wartość nieznana” i nie może być w zwykły sposób porównywana ze znanymi wartościami. W przypadku próby użycia wartości NULL ze zwykłymi arytmetycznymi operatorami porównania wynik będzie nieustalony:

```
mysql> SELECT NULL < 0, NULL = 0, NULL <> 0, NULL > 0;
+-----+-----+-----+-----+
| NULL < 0 | NULL = 0 | NULL <> 0 | NULL > 0 |
+-----+-----+-----+-----+
| NULL | NULL | NULL | NULL |
+-----+-----+-----+-----+
```

W rzeczywistości nie można nawet porównać dwóch wartości NULL, ponieważ nie ma możliwości ustalenia wyniku porównania dwóch wartości nieznanych:

```
mysql> SELECT NULL = NULL, NULL <> NULL;
+-----+-----+
| NULL = NULL | NULL <> NULL |
+-----+-----+
| NULL | NULL |
+-----+-----+
```

Zamiast użycia operatorów =, <> lub != do ustalenia równości bądź nierówności z wartością NULL wykorzystaj operator IS NULL lub IS NOT NULL. Na przykład, dla żyjących prezydentów data ich śmierci jest w tabeli president zdefiniowana jako NULL. Aby pobrać z tabeli tych prezydentów, należy użyć poniższego zapytania:

```
mysql> SELECT last_name, first_name FROM president WHERE death IS NULL;
+-----+-----+
| last_name | first_name |
+-----+-----+
| Carter   | James E.   |
| Bush     | George H.W. |
| Clinton  | William J. |
| Bush     | George W.   |
| Obama    | Barack H.   |
+-----+-----+
```

W celu wyszukania wartości innych niż NULL trzeba zastosować operator IS NOT NULL. Zaprezentowane poniżej zapytanie wyszukuje imiona zawierające przyrostek:

```
mysql> SELECT last_name, first_name, suffix
-> FROM president WHERE suffix IS NOT NULL;
+-----+-----+-----+
| last_name | first_name | suffix |
+-----+-----+-----+
| Carter   | James E.   | Jr.    |
+-----+-----+-----+
```

Obecny w bazie danych MySQL operator porównania <=> można stosować także do porównywania wartości NULL. Dwa wcześniejsze zapytania mogą być utworzone w wersji opartej na wymienionym operatorze:

```
SELECT last_name, first_name FROM president WHERE death <=> NULL;
```

```
SELECT last_name, first_name, suffix
FROM president WHERE NOT (suffix <=> NULL);
```

1.4.9.3. Sortowanie wyników zapytania

Każdy użytkownik bazy danych MySQL wcześniej czy później przekona się, że po utworzeniu tabeli, umieszczeniu w niej rekordów, a następnie wykonaniu zapytania `SELECT * FROM nazwa_tabeli` rekordy zostaną pobrane w kolejności ich wstawiania. To ma pewien sens, więc naturalne jest przyjęcie założenia o domyślnym pobieraniu rekordów w kolejności ich wstawiania. Jednak takie rozwiązanie nie zawsze będzie pożądane. Po początkowym wstawieniu danych do tabeli, usunięciu i wstawieniu kolejnych rekordów może ulec zmianie kolejność, w jakiej serwer będzie pobierał rekordy.

W kwestii kolejności pobierania rekordów musisz pamiętać o jednym: *nie ma żadnej gwarancji* użycia konkretnej kolejności pobierania rekordów, o ile samodzielnie takiej nie wskażesz. Aby wskazać kolejność, należy dodać klauzulę `ORDER BY` do zapytania. Wymieniona klauzula definiuje żadaną kolejność sortowania. Przedstawione poniżej zapytanie powoduje posortowanie prezydentów w kolejności leksykalnej (alfabetycznej) według nazwisk:

```
mysql> SELECT last_name, first_name FROM president
-> ORDER BY last_name;
```

last_name	first_name
Adams	John Quincy
Adams	John
Arthur	Chester A.
Buchanan	James

...

Domyślnie, klauzula `ORDER BY` powoduje sortowanie kolumn w kolejności rosnącej. Dodanie słowa kluczowego `ASC` lub `DESC` po nazwie kolumny wyraźnie wskazuje odpowiednio rosnącą lub malejącą kolejność sortowania. Na przykład, w celu posortowania listy prezydentów w malejącej kolejności nazwisk trzeba użyć słowa kluczowego `DESC` w następujący sposób:

```
mysql> SELECT last_name, first_name FROM president
-> ORDER BY last_name DESC;
```

last_name	first_name
Wilson	Woodrow
Washington	George
Van Buren	Martin
Tyler	John

...

Istnieje możliwość przeprowadzenia sortowania na podstawie wielu kolumn, z których każda może być niezależnie sortowana w kolejności rosnącej lub malejącej. Poniższe zapytanie pobiera rekordy z tabeli `presidents`, sortuje je w kolejności malejącej

(według nazwy stanu urodzenia prezydenta) oraz w kolejności rosnącej (według nazwisk prezydentów pochodzących z danego stanu):

```
mysql> SELECT last_name, first_name, state FROM president
-> ORDER BY state DESC, last_name ASC;
```

last_name	first_name	state
Arthur	Chester A.	VT
Coolidge	Calvin	VT
Harrison	William H.	VA
Jefferson	Thomas	VA
Madison	James	VA
Monroe	James	VA
Taylor	Zachary	VA
Tyler	John	VA
Washington	George	VA
Wilson	Woodrow	VA
Eisenhower	Dwight D.	TX
Johnson	Lyndon B.	TX

...

Wartości NULL są umieszczane na początku kolumny sortowanej w kolejności rosnącej oraz na końcu w przypadku kolumn sortowanych w kolejności malejącej. Aby zagwarantować umieszczenie wartości NULL w odpowiednim miejscu w trakcie sortowania, konieczne jest dodanie kolejnej kolumny sortowania, odróżniającej wartości NULL od innych niż NULL. Na przykład, jeśli sortujesz prezydentów w malejącej kolejności daty ich śmierci, wówczas żyjący prezydenci (o dacie śmierci podanej jako NULL) zostaną umieszczeni na końcu tak posortowanej listy. Aby umieścić ich na początku, należy użyć przedstawionego poniżej zapytania. Funkcja IF() oblicza wyrażenie przekazane jej w pierwszym argumencie i zwraca wartość drugiego lub trzeciego argumentu w zależności od obliczonej wartości wyrażenia (true lub false). W przypadku poniższego zapytania funkcja IF() przypisze 0 dla wartości NULL i 1 dla wartości innych niż NULL. W ten sposób wszystkie wartości NULL zostaną umieszczone na liście przed wartościami innymi niż NULL. Sortowanie względem last_name jako drugiej kolumny sortowania powoduje umieszczenie rekordów o takiej samej wartości death w kolejności ułożonej według nazwisk.

```
mysql> SELECT last_name, first_name, death FROM president
-> ORDER BY IF(death IS NULL,0,1), death DESC, last_name;
```

last_name	first_name	death
Bush	George W.	NULL
Bush	George H.W.	NULL
Carter	James E.	NULL
Clinton	William J.	NULL
Obama	Barack H.	NULL
Ford	Gerald R.	2006-12-26
Reagan	Ronald W.	2004-06-05
Nixon	Richard M.	1994-04-22

...

Adams	John	1826-07-04
-------	------	------------

Jefferson	Thomas	1826-07-04
Washington	George	1799-12-14

1.4.9.4. Ograniczanie wyników zapytania

Aby wyświetlić jedynie część rekordów zwróconych przez zapytanie, trzeba dodać klauzulę LIMIT. Wymieniona klauzula jest szczególnie użyteczna w połączeniu z klauzulą ORDER BY. Baza danych MySQL pozwala na ograniczenie ilości danych wyjściowych do pierwszych *n* rekordów wyniku zapytania. Poniższe zapytanie wyświetla pierwszych pięciu urodzonych prezydentów:

```
mysql> SELECT last_name, first_name, birth FROM president
-> ORDER BY birth LIMIT 5;
```

last_name	first_name	birth
Washington	George	1732-02-22
Adams	John	1735-10-30
Jefferson	Thomas	1743-04-13
Madison	James	1751-03-16
Monroe	James	1758-04-28

Po użyciu sortowania w kolejności malejącej (słowo kluczowe DESC) otrzymujemy wynik wymieniający pięciu ostatnio urodzonych prezydentów:

```
mysql> SELECT last_name, first_name, birth FROM president
-> ORDER BY birth DESC LIMIT 5;
```

last_name	first_name	birth
Obama	Barack H.	1961-08-04
Clinton	William J.	1946-08-19
Bush	George W.	1946-07-06
Carter	James E.	1924-10-01
Bush	George H.W.	1924-06-12

Klauzula LIMIT pozwala również na wyświetlenie sekcji rekordów pochodzącej z środka zestawu wynikowego. Konieczne jest podanie dwóch wartości: liczby rekordów do pominięcia na początku zestawu wynikowego oraz liczby rekordów, które mają być wyświetlone. Poniższe zapytanie jest podobne do poprzedniego, ale zwraca pięć rekordów po pominięciu pierwszych dziesięciu:

```
mysql> SELECT last_name, first_name, birth FROM president
-> ORDER BY birth DESC LIMIT 10, 5;
```

last_name	first_name	birth
Eisenhower	Dwight D.	1890-10-14
Truman	Harry S	1884-05-08
Roosevelt	Franklin D.	1882-01-30
Hoover	Herbert C.	1874-08-10
Coolidge	Calvin	1872-07-04

Aby losowo wybrać rekord lub zestaw rekordów z tabeli, użyj klauzuli `ORDER BY RAND()` w połączeniu z `LIMIT`:

```
mysql> SELECT last_name, first_name FROM president
-> ORDER BY RAND() LIMIT 1;
+-----+-----+
| last_name | first_name |
+-----+-----+
| Johnson   | Lyndon B.  |
+-----+-----+

mysql> SELECT last_name, first_name FROM president
-> ORDER BY RAND() LIMIT 3;
+-----+-----+
| last_name | first_name |
+-----+-----+
| Harding   | Warren G.  |
| Bush      | George H.W. |
| Jefferson | Thomas     |
+-----+-----+
```

1.4.9.5. Obliczanie wartości i nazywanie kolumn danych wyjściowych

Większość przedstawionych dotąd zapytań generowała dane wyjściowe przez pobieranie wartości z tabel. Istnieje również możliwość obliczenia wartości danych wyjściowych na podstawie wyrażeń wraz z odwoływaniem się do tabel lub bez niego. Przedstawione poniżej zapytanie oblicza wartość prostego wyrażenia (stała) oraz bardziej skomplikowanego wyrażenia zawierającego operacje arytmetyczne. W zapytaniu mamy kilka wywołań funkcji, których celem jest obliczenie kwadratu wyrażenia i wyświetlenie wyniku w formacie zawierającym trzy cyfry po przecinku dziesiętnym:

```
mysql> SELECT 17, FORMAT(SQRT(25+13),3);
+-----+-----+
| 17 | FORMAT(SQRT(25+13),3) |
+-----+-----+
| 17 | 6.164                  |
+-----+-----+
```

Wyrażenie może także odwoływać się do kolumn tabeli:

```
mysql> SELECT CONCAT(first_name, ' ',last_name),CONCAT(city,', ',state)
-> FROM president;
+-----+-----+
| CONCAT(first_name, ' ',last_name) | CONCAT(city,', ',state) |
+-----+-----+
| George Washington                 | Wakefield, VA          |
| John Adams                        | Braintree, MA          |
| Thomas Jefferson                  | Albemarle County, VA   |
| James Madison                     | Port Conway, VA        |
| ...                               | ...                     |
```

Powyższe zapytanie formatuje imię i nazwisko prezydenta jako pojedynczy ciąg tekstowy przez połączenie imienia i nazwiska, rozdzielając je spacją. Ponadto formatuje nazwę miasta urodzenia i stanu przez ich połączenie i rozdzielenie przecinkiem oraz spacją.

Wyrażenie obliczające wartość kolumny staje się nazwą kolumny i jest wyświetlane jako jej nagłówek. Jeśli wyrażenie jest długie, to skutkiem będzie powstanie szerokiej kolumny, jak pokazano w danych wyjściowych poprzedniego zapytania. Aby dane wyjściowe były czytelniejsze dla użytkownika, kolumnie można przypisać inną nazwę (tak zwany alias), używając do tego konstrukcji *AS nazwa*:

```
mysql> SELECT CONCAT(first_name, ' ', last_name) AS Name,
->    CONCAT(city, ', ', state) AS Birthplace
->    FROM president;
```

Name	Birthplace
George Washington	Wakefield, VA
John Adams	Braintree, MA
Thomas Jefferson	Albemarle County, VA
James Madison	Port Conway, VA

...

Jeżeli alias kolumny zawiera spację, trzeba go ująć w apostrofy:

```
mysql> SELECT CONCAT(first_name, ' ', last_name) AS 'President Name',
->    CONCAT(city, ', ', state) AS 'Place of Birth'
->    FROM president;
```

President Name	Place of Birth
George Washington	Wakefield, VA
John Adams	Braintree, MA
Thomas Jefferson	Albemarle County, VA
James Madison	Port Conway, VA

...

Podczas stosowania aliasu kolumny użycie słowa kluczowego *AS* jest opcjonalne:

```
mysql> SELECT 1 one, 2 two, 3 three;
```

one	two	three
1	2	3

Jeżeli wynik zapytania zawiera kolumnę o niewłaściwej nazwie lub brakuje w nim kolumny, wówczas sprawdź, czy nie zapomniałeś umieścić przecinka między nazwami kolumn. W takim przypadku druga nazwa będzie traktowana jako alias dla pierwszej. Na przykład, możesz przygotować zapytanie jak przedstawiono poniżej i zapomnieć umieścić przecinek między kolumnami `first_name` i `last_name`. W wyniku wspomnianego przeoczenia kolumna `first_name` będzie błędnie nazwana `last_name`, natomiast kolumna `last_name` nie zostanie uwzględniona w wyświetlonych danych wyjściowych:

```
mysql> SELECT first_name last_name FROM president;
```

last_name
George

John	
Thomas	
James	
...	

1.4.9.6. Praca z datami

Przed wszystkim należy pamiętać, że podczas używania dat baza danych MySQL zawsze oczekuje podania roku jako pierwszego członu. Aby wskazać datę 27 lipca 2012 roku, należy użyć zapisu '2012-07-27'. Nie używaj zapisu '07-27-2012' lub '27-07-2012'. W przypadku wartości danych wejściowych w innych formatach masz możliwość ich konwersji przy użyciu funkcji `STR_TO_DATE()`. Odpowiedni przykład przedstawiono w punkcie 3.2.6, zatytułowanym „Typy danych dla wartości daty i czasu”.

Baza danych MySQL obsługuje wiele rodzajów operacji przeprowadzanych na danych:

- sortowanie względem daty (kilkukrotnie użyliśmy już tej możliwości);
- wyszukiwanie określonych dat lub zakresu dat;
- wyodrębnianie fragmentów wartości daty, na przykład roku, miesiąca lub dnia;
- obliczanie różnic między datami;
- obliczanie daty przez dodanie lub odjęcie pewnej wartości od innej daty.

Poniżej przedstawiono pewne przykłady wspomnianych operacji.

Aby wyszukać określoną datę — zarówno konkretną wartość, jak i względną do innej — konieczne jest porównanie kolumny typu `DATE` z interesującą Cię wartością:

```
mysql> SELECT * FROM grade_event WHERE date = '2012-10-01';
+-----+-----+-----+
| date   | category | event_id |
+-----+-----+-----+
| 2012-10-01 | T       | 6        |
+-----+-----+-----+

mysql> SELECT last_name, first_name, death
-> FROM president
-> WHERE death >= '1970-01-01' AND death < '1980-01-01';
+-----+-----+-----+
| last_name | first_name | death      |
+-----+-----+-----+
| Truman   | Harry S   | 1972-12-26 |
| Johnson  | Lyndon B. | 1973-01-22 |
+-----+-----+-----+
```

W celu przetestowania lub pobrania fragmentu daty trzeba użyć funkcji takiej jak `YEAR()`, `MONTH()` lub `DAYOFMONTH()`. Na przykład, aby wyszukać prezydentów urodzonych w marcu, należy znaleźć daty, w których wartość miesiąca wynosi 3:

```
mysql> SELECT last_name, first_name, birth
-> FROM president WHERE MONTH(birth) = 3;
+-----+-----+-----+
| last_name | first_name | birth      |
+-----+-----+-----+
| Madison  | James     | 1751-03-16 |
+-----+-----+-----+
```

Jackson	Andrew	1767-03-15
Tyler	John	1790-03-29
Cleveland	Grover	1837-03-18

Powyższe zapytanie można zapisać także w wersji zawierającej podaną nazwę miesiąca:

```
mysql> SELECT last_name, first_name, birth
-> FROM president WHERE MONTHNAME(birth) = 'March';
```

last_name	first_name	birth
Madison	James	1751-03-16
Jackson	Andrew	1767-03-15
Tyler	John	1790-03-29
Cleveland	Grover	1837-03-18

Istnieje możliwość wyszukania prezydentów urodzonych konkretnego dnia w marcu. Wymaga to połączenia funkcji MONTH() i DAYOFMONTH():

```
mysql> SELECT last_name, first_name, birth
-> FROM president WHERE MONTH(birth) = 3 AND DAYOFMONTH(birth) = 29;
```

last_name	first_name	birth
Tyler	John	1790-03-29

Powyższe zapytanie można wykorzystać do wygenerowania spotykanych czasami list „sławnych osób, które mają dzisiaj urodziny”. Jednak jeżeli trzeba wybrać rekordy, których wartości dnia i miesiąca odpowiadają „dniu dzisiejszemu”, wówczas nie wolno stosować wartości dosłownych w sposób przedstawiony w poprzednim zapytaniu. Aby wyszukać prezydentów świętujących dzisiaj urodziny, niezależnie od roku urodzenia, ich daty urodzenia trzeba porównać do wartości dnia i miesiąca wyodrębnionych z funkcji CURDATE(), która zawsze zwraca bieżącą datę:

```
SELECT last_name, first_name, birth
FROM president WHERE MONTH(birth) = MONTH(CURDATE())
AND DAYOFMONTH(birth) = DAYOFMONTH(CURDATE());
```

W celu obliczenia odstępu między datami trzeba odjąć jedną od drugiej. Na przykład, jeśli chcesz dowiedzieć się, który prezydent żył najdłużej, wtedy oblicz jego wiek w chwili śmierci przez odjęcie daty urodzenia od daty jego śmierci. Funkcja TIMESTAMPDIFF() jest tutaj użyteczna, ponieważ pobiera argument wskazujący jednostkę daty, w której ma zostać wyrażony wynik (w omawianym przykładzie to rok — YEAR).

```
mysql> SELECT last_name, first_name, birth, death,
-> TIMESTAMPDIFF(YEAR, birth, death) AS age
-> FROM president WHERE death IS NOT NULL
-> ORDER BY age DESC LIMIT 5;
```

last_name	first_name	birth	death	age
-----------	------------	-------	-------	-----

Reagan	Ronald W.	1911-02-06	2004-06-05	93
Ford	Gerald R.	1913-07-14	2006-12-26	93
Adams	John	1735-10-30	1826-07-04	90
Hoover	Herbert C.	1874-08-10	1964-10-20	90
Truman	Harry S	1884-05-08	1972-12-26	88

Jeżeli różnicę chcesz wyrazić w dniach, alternatywnym sposobem obliczenia różnicy między datami jest użycie funkcji `TO_DAYS()`, konwertującej datę na liczbę dni. Określenie liczby dni pomiędzy dwiema wskazanymi datami to jedno z zastosowań wymienionej funkcji. Na przykład, jeśli chcesz dowiedzieć się, kto w Lidze Historycznej powinien wkrótce odnowić członkostwo, oblicz liczbę dni między datą wygaśnięcia członkostwa i dniem dzisiejszym. Gdy obliczona liczba dni będzie mniejsza niż pewna ustalona wartość graniczna, oznacza to konieczność odnowienia członkostwa w niedługim czasie. Przedstawione poniżej zapytanie wyszukuje członków Ligi, których członkostwo wygasło lub wygaśnie w ciągu następnych 60 dni:

```
SELECT last_name, first_name, expiration FROM member
WHERE (TO_DAYS(expiration) - TO_DAYS(CURDATE())) < 60;
```

Odpowiednik powyższego zapytania, ale wykorzystujący funkcję `TIMESTAMPDIFF()`, przedstawia się następująco:

```
SELECT last_name, first_name, expiration FROM member
WHERE TIMESTAMPDIFF(DAY, CURDATE(), expiration) < 60;
```

Aby obliczyć datę na podstawie innej daty, należy użyć funkcji `DATE_ADD()` lub `DATE_SUB()`. Wymienione funkcje pobierają datę, odstęp czasu i generują nową datę. Na przykład:

```
mysql> SELECT DATE_ADD('1970-1-1', INTERVAL 10 YEAR);
+-----+
| DATE_ADD('1970-1-1', INTERVAL 10 YEAR) |
+-----+
| 1980-01-01                               |
+-----+
mysql> SELECT DATE_SUB('1970-1-1', INTERVAL 10 YEAR);
+-----+
| DATE_SUB('1970-1-1', INTERVAL 10 YEAR) |
+-----+
| 1960-01-01                               |
+-----+
```

Przedstawione nieco wcześniej w tekście zapytanie pobierało rekordy prezydentów zmarłych w latach siedemdziesiątych ubiegłego wieku. Wspomniane zapytanie było oparte na dosłownych datach wskazujących sprawdzany przedział czasu. Innym sposobem zapisu tego rodzaju zapytania jest użycie dosłownej daty początkowej i obliczenie końcowej na podstawie daty początkowej i odstępu czasu:

```
mysql> SELECT last_name, first_name, death
-> FROM president
-> WHERE death >= '1970-1-1'
```

```
-> AND death < DATE_ADD('1970-1-1', INTERVAL 10 YEAR);
```

last_name	first_name	death
Truman	Harry S	1972-12-26
Johnson	Lyndon B.	1973-01-22

Poniżej przedstawiono jeszcze inny sposób utworzenia zapytania wskazującego członków Ligi, którzy powinni odnowić członkostwo. Tym razem wykorzystano funkcję `DATE_ADD()`:

```
SELECT last_name, first_name, expiration FROM member
WHERE expiration < DATE_ADD(CURDATE(), INTERVAL 60 DAY);
```

Jeżeli kolumna `expiration` jest indeksowana, powyższe zapytanie będzie znacznie efektywniejsze niż przedstawione nieco wcześniej w tekście. Powody tego zostaną wyjaśnione w rozdziale 5, zatytułowanym „Optymalizacja zapytań”.

Gdzieś na początku rozdziału widziałeś zapytanie wybierające pacjentów, którzy od dłuższego czasu nie byli na wizycie kontrolnej:

```
SELECT last_name, first_name, last_visit FROM patient
WHERE last_visit < DATE_SUB(CURDATE(), INTERVAL 6 MONTH);
```

Wtedy powyższe zapytanie było dla Ciebie tajemnicze. Czy teraz ma większy sens?

1.4.9.7. Dopasowanie wzorca

MySQL obsługuje operacje dopasowania wzorca, pozwalające na wybór rekordów bez konieczności podawania dokładnej wartości do porównania. Wystarczy użyć operatora `LIKE` lub `NOT LIKE` i wskazać ciąg tekstowy zawierający znaki wieloznaczne. Znak `_` powoduje dopasowanie jednego, dowolnego znaku, natomiast `%` dopasowuje dowolną sekwencję znaków (także pustą).

Wzorzec użyty w poniższym zapytaniu powoduje dopasowanie nazwisk rozpoczynających się na literę `W` lub `w`:

```
mysql> SELECT last_name, first_name FROM president
-> WHERE last_name LIKE 'W%';
```

last_name	first_name
Washington	George
Wilson	Woodrow

W kolejnym zapytaniu zaprezentowano najczęściej popełniany błąd. Dopasowanie wzorca nie zwraca żadnych wyników, ponieważ wzorzec został podany po arytmetycznym operatorze porównania, a nie po operatorze `LIKE`:

```
mysql> SELECT last_name, first_name FROM president
-> WHERE last_name = 'W%';
Empty set (0.00 sec)
```

Powyższe zapytanie zwróciłoby rekordy jedynie wtedy, gdy kolumna zawierałaby dosłowny ciąg tekstowy o treści `W%` lub `w%`.

Dopasowanie wzorca użyte w poniższym zapytaniu powoduje dopasowanie nazwisk zawierających literę W lub w znajdującą się w dowolnym miejscu, a nie tylko na początku:

```
mysql> SELECT last_name, first_name FROM president
-> WHERE last_name LIKE '%W%';
```

last_name	first_name
Washington	George
Wilson	Woodrow
Eisenhower	Dwight D.

Z kolei dopasowanie wzorca użyte w poniższym zapytaniu dopasowuje nazwiska składające się z dokładnie czterech liter:

```
mysql> SELECT last_name, first_name FROM president
-> WHERE last_name LIKE '____';
```

last_name	first_name
Polk	James K.
Taft	William H.
Ford	Gerald R.
Bush	George H.W.
Bush	George W.

MySQL oferuje także inną formę dopasowania wzorca, opartą na wyrażeniach regularnych i operatorze REXEXP. Dokładne omówienie operatorów znajdziesz w podpunkcie 3.5.1.1, zatytułowanym „Typy operatorów”, oraz w dodatku C, zatytułowanym „Przewodnik po operatorach i funkcjach”.

1.4.9.8. Konfiguracja i używanie zmiennych zdefiniowanych przez użytkownika

Baza danych MySQL pozwala na definiowanie własnych zmiennych. Można je ustawić przy użyciu wyników zapytania, co stanowi wygodny sposób zapisania wartości do wykorzystania w późniejszych zapytaniach. Przyjmijmy założenie, że chcesz wyświetlić prezydentów urodzonych przed Andrew Jacksonem. Aby pobrać odpowiednie rekordy, należy datę urodzenia Andrew Jacksona umieścić w zmiennej, a następnie wybrać prezydentów, których data urodzenia jest wcześniejsza niż wartość zdefiniowanej zmiennej:

```
mysql> SELECT @jackson_birth := birth FROM president
-> WHERE last_name = 'Jackson' AND first_name = 'Andrew';
```

@jackson_birth := birth
1767-03-15

```
mysql> SELECT last_name, first_name, birth FROM president
-> WHERE birth < @jackson_birth ORDER BY birth;
```


last_name	first_name	birth
Washington	George	1732-02-22
Adams	John	1735-10-30
Jefferson	Thomas	1743-04-13
Madison	James	1751-03-16
Monroe	James	1758-04-28

Do utworzenia zmiennej użytkownika używana jest składnia *@nazwa_zmiennej*. W zapytaniu SELECT przypisanie wartości zmiennej odbywa się przy użyciu wyrażenia w postaci *@nazwa_zmiennej := wartość*. Pierwsze zapytanie wyszukuje datę urodzenia Andrew Jacksona i przypisuje ją zmiennej *@jackson_birth*. (Wynik wykonania zapytania SELECT będzie nadal wyświetlony na ekranie, przypisanie zmiennej wyniku zapytania nie powoduje zawieszenia wyświetlania danych wyjściowych zapytania). Drugie zapytanie odwołuje się do zmiennej i wykorzystuje jej wartość do wyszukania innych rekordów w tabeli *presidents* o mniejszej wartości *birth*.

Poprzedni problem można jeszcze rozwiązać przy użyciu pojedynczego zapytania wykorzystującego złączenie lub podzapytanie. Jednak czasami znacznie czytelniejsze jest użycie zmiennej. (Rozwiązanie oparte na podzapytaniu znajdziesz w podpunkcie 1.4.9.10, zatytułowanym „Pobieranie informacji z wielu tabel”).

Zmienne można przypisywać także za pomocą zapytania SET. W takim przypadku operatorem przypisania może być zarówno *=*, jak i *:=*.

```
mysql> SET @today = CURDATE();
mysql> SET @one_week_ago := DATE_SUB(@today, INTERVAL 7 DAY);
mysql> SELECT @today, @one_week_ago;
+-----+-----+
| @today | @one_week_ago |
+-----+-----+
| 2012-04-21 | 2012-04-14 |
+-----+-----+
```

1.4.9.9. Generowanie podsumowania

Jedną z najużyteczniejszych możliwości oferowanych przez MySQL jest skondensowanie ogromnej ilości danych i wygenerowanie na ich podstawie podsumowania. Baza danych MySQL staje się potężnym sprzymierzeńcem, kiedy uczysz się generować podsumowania, ponieważ ich ręczne przygotowywanie jest zadaniem bardzo żmudnym, czasochłonnym i podatnym na powstawanie błędów.

Jednym z prostych typów podsumowania jest wyświetlenie unikalnych wartości w pewnym ich zestawie. Słowo kluczowe *DISTINCT* pozwala na usunięcie duplikatów z wyniku. Na przykład, różne stany, w których urodzili się prezydenci, można pobrać w następujący sposób:

```
mysql> SELECT DISTINCT state FROM president ORDER BY state;
+-----+
| state |
+-----+
| AR    |
```

CA
CT
GA
HI
IA
IL
KY
MA
MO
...

Inna forma podsumowania obejmuje zliczanie przy użyciu funkcji `COUNT()`. Wywołanie `COUNT(*)` podaje liczbę rekordów wybranych przez zapytanie. Jeżeli w zapytaniu brakuje klauzuli `WHERE`, wybrane zostaną wszystkie rekordy, a tym samym wywołanie `COUNT(*)` poda całkowitą liczbę rekordów w tabeli. Poniższe zapytanie podaje liczbę rekordów znajdujących się w tabeli `member` projektu Ligi Historycznej:

```
mysql> SELECT COUNT(*) FROM member;
```

COUNT(*)
102

Jeżeli w zapytaniu znajduje się klauzula `WHERE`, wywołanie `COUNT(*)` podaje liczbę rekordów dopasowanych przez tę klauzulę. Poniższe zapytanie wyświetla liczbę przeprowadzonych dotąd sprawdzianów w klasie:

```
mysql> SELECT COUNT(*) FROM grade_event WHERE category = 'Q';
```

COUNT(*)
4

Wywołanie `COUNT(*)` zlicza wszystkie wybrane rekordy. Z kolei wywołanie `COUNT(nazwa_kolumny)` zlicza jedynie wartości inne niż `NULL`. Poniższe zapytanie demonstruje wspomnianą różnicę:

```
mysql> SELECT COUNT(*), COUNT(email), COUNT(expiration) FROM member;
```

COUNT(*)	COUNT(email)	COUNT(expiration)
102	80	96

Wyświetlone dane wyjściowe pokazują, że choć w tabeli `member` znajdują się 102 rekordy, to tylko 80 z nich ma wartość w kolumnie `email`. Widać także, że sześciu członków Ligi ma wykupione członkostwo dożywotnio. (Wartość `NULL` w kolumnie `expiration` oznacza członkostwo dożywotnie. Ponieważ 96 spośród 102 rekordów ma wartość inną niż `NULL`, pozostaje sześć rekordów z wartością `NULL` w kolumnie `expiration`).

Funkcja `COUNT()` w połączeniu ze słowem kluczowym `DISTINCT` powoduje obliczenie unikalnych wartości innych niż `NULL` w wyniku zapytania. Aby podać liczbę różnych stanów, w których urodzili się prezydenci, należy wykonać poniższe zapytanie:

```
mysql> SELECT COUNT(DISTINCT state) FROM president;
+-----+
| COUNT(DISTINCT state) |
+-----+
| 21 |
+-----+
```

Istnieje możliwość obliczenia ogólnej liczby wartości w kolumnie lub wartości poszczególnych kategorii. Na przykład, całkowita liczba uczniów w klasie zostanie wyświetlona po wykonaniu poniższego zapytania:

```
mysql> SELECT COUNT(*) FROM student;
+-----+
| COUNT(*) |
+-----+
| 31 |
+-----+
```

Mógłbyś zapytać, ilu uczniów to chłopcy, a ile to dziewczynki. Jednym ze sposobów uzyskania takich informacji jest oddzielne zliczenie uczniów danej płci:

```
mysql> SELECT COUNT(*) FROM student WHERE sex='f';
+-----+
| COUNT(*) |
+-----+
| 15 |
+-----+
mysql> SELECT COUNT(*) FROM student WHERE sex='m';
+-----+
| COUNT(*) |
+-----+
| 16 |
+-----+
```

Jednak tego rodzaju podejście jest żmudne i bardzo niedoskonałe w przypadku kolumn, które mogą zawierać wiele różnych wartości. Zastanów się, w jaki sposób można podać liczbę prezydentów urodzonych w poszczególnych stanach. Stosując powyższe rozwiązanie, najpierw trzeba ustalić wszystkie stany, w których urodzili się prezydenci (`SELECT DISTINCT state FROM president`), a następnie dla każdego stanu wykonać zapytanie `SELECT COUNT(*)`. Konieczności wykonania tej żmudnej pracy na pewno chcesz uniknąć.

Na szczęście, istnieje możliwość wykonania pojedynczego zapytania podającego liczbę wystąpień poszczególnych wartości w kolumnie. W przypadku listy uczniów obliczenie liczby chłopców i dziewczynek jest możliwe dzięki użyciu klauzuli `GROUP BY`:

```
mysql> SELECT sex, COUNT(*) FROM student GROUP BY sex;
+-----+-----+
| sex | COUNT(*) |
+-----+-----+
| F | 15 |
+-----+-----+
```

```
| M |      16 |
+---+-----+
```

Tego samego rodzaju zapytanie można wykorzystać do podania liczby prezydentów urodzonych w poszczególnych stanach.

```
mysql> SELECT state, COUNT(*) FROM president GROUP BY state;
```

```
+-----+-----+
| state | COUNT(*) |
+-----+-----+
| AR    |         1 |
| CA    |         1 |
| CT    |         1 |
| GA    |         1 |
| HI    |         1 |
| IA    |         1 |
| IL    |         1 |
| KY    |         1 |
| MA    |         4 |
| MO    |         1 |
...

```

Kiedy w ten sposób zliczasz wartości w grupach, klauzula `GROUP BY` informuje MySQL, jak klastrować wartości przed ich zliczeniem.

Użycie wywołania funkcji `COUNT(*)` wraz z klauzulą `GROUP BY` w celu zliczenia wartości ma wiele zalet względem oddzielnego zliczania liczby wystąpień unikalnych wartości kolumny:

- Nie trzeba wcześniej znać wartości znajdujących się w kolumnie.
- Operację można przeprowadzić przy użyciu tylko jednego zapytania zamiast konieczności użycia wielu.
- Całe wyniki są otrzymywane po wykonaniu pojedynczego zapytania, co daje możliwość sortowania danych wyjściowych.

Pierwsze dwie wymienione zalety są ważne, ponieważ ułatwiają tworzenie zapytań. Z kolei trzecia zaleta jest ważna, ponieważ daje dużą elastyczność w zakresie wyświetlania wyników. Domyślnie do sortowania wyników baza danych MySQL używa kolumn wymienionych w klauzuli `GROUP BY`, ale możesz użyć klauzuli `ORDER BY` w celu zastosowania innej kolejności sortowania. Na przykład, jeśli chcesz otrzymać liczbę prezydentów pogrupowanych względem stanów ich urodzenia, a listę posortować w kolejności od stanu, w którym urodziło się najwięcej prezydentów, wtedy możesz w następujący sposób użyć klauzuli `ORDER BY`:

```
mysql> SELECT state, COUNT(*) AS count FROM president
-> GROUP BY state ORDER BY count DESC;
```

```
+-----+-----+
| state | count |
+-----+-----+
| VA    |      8 |
| OH    |      7 |
| MA    |      4 |
| NY    |      4 |
| NC    |      2 |

```

VT		2	
TX		2	
GA		1	
IL		1	
SC		1	

...

Kiedy kolumna używana do sortowania powstała na skutek działania funkcji generującej podsumowanie, do funkcji nie można się odnieść bezpośrednio w klauzuli ORDER BY. Zamiast tego kolumnie trzeba przypisać alias, a następnie użyć go. Takie rozwiązanie pokazano w poprzednim zapytaniu, gdy kolumna COUNT(*) to alias o nazwie count.

Aby pogrupować wyniki za pomocą klauzuli GROUP BY i wygenerowanej kolumny, musisz się do niej odwołać, używając aliasu lub położenia kolumny, czyli podobnie jak w przypadku klauzuli ORDER BY. Przedstawione poniżej zapytanie pokazuje, jak obliczyć liczbę prezydentów urodzonych w poszczególnych miesiącach:

```
mysql> SELECT MONTH(birth) AS Month, MONTHNAME(birth) AS Name,
-> COUNT(*) AS count
-> FROM president GROUP BY Name ORDER BY Month;
```

Month	Name	count
1	January	4
2	February	4
3	March	4
4	April	4
5	May	2
6	June	1
7	July	4
8	August	5
9	September	1
10	October	6
11	November	5
12	December	3

Wywołanie funkcji COUNT() można połączyć z klauzulami ORDER BY i LIMIT. Na przykład, aby ustalić cztery najczęściej wymienione stany w tabeli president użyj poniższego zapytania:

```
mysql> SELECT state, COUNT(*) AS count FROM president
-> GROUP BY state ORDER BY count DESC LIMIT 4;
```

state	count
VA	8
OH	7
MA	4
NY	4

Jeżeli nie chcesz ograniczać danych wyjściowych zapytania za pomocą klauzuli LIMIT, a raczej przez wyszukanie określonych wartości przez wywołanie COUNT(), to użyj klauzuli HAVING. Działanie klauzuli HAVING jest podobne do WHERE pod względem podawania warunków, które muszą być spełnione przez rekordy danych wyjściowych. Różnica

w stosunku do klauzuli WHERE polega na możliwości odwołania się do wyniku działania funkcji generującej podsumowanie, na przykład COUNT(). Poniższe zapytanie wyświetla stany, z których pochodzi przynajmniej dwóch prezydentów:

```
mysql> SELECT state, COUNT(*) AS count FROM president
-> GROUP BY state HAVING count > 1 ORDER BY count DESC;
```

state	count
VA	8
OH	7
MA	4
NY	4
NC	2
VT	2
TX	2

Ogólnie rzecz biorąc, tego typu zapytanie jest wykonywane, gdy chcesz odszukać powtarzające się wartości w kolumnie. W celu wyszukania niepowtarzających się wartości należy użyć HAVING count = 1.

Poza COUNT() istnieje jeszcze wiele innych funkcji generujących podsumowanie. Funkcje MIN(), MAX(), SUM() i AVG() są użyteczne podczas obliczania w kolumnie wartości minimalnej, maksymalnej, całkowitej i średniej. Wymienione funkcje mogą być używane jednocześnie. Przedstawione poniżej zapytanie wyświetla cechy charakterystyczne poszczególnych sprawdzianów i testów przeprowadzonych w klasie. Wyświetla również liczbę ocen użytą do obliczenia poszczególnych wartości. (Niektórzy uczniowie mogli być nieobecni na sprawdzianie lub nie być brani pod uwagę).

```
mysql> SELECT
-> event_id,
-> MIN(score) AS minimum,
-> MAX(score) AS maximum,
-> MAX(score)-MIN(score)+1 AS span,
-> SUM(score) AS total,
-> AVG(score) AS average,
-> COUNT(score) AS count
-> FROM score
-> GROUP BY event_id;
```

event_id	minimum	maximum	span	total	average	count
1	9	20	12	439	15.1379	29
2	8	19	12	425	14.1667	30
3	60	97	38	2425	78.2258	31
4	7	20	14	379	14.0370	27
5	8	20	13	383	14.1852	27
6	62	100	39	2325	80.1724	29

Te informacje będą znacznie czytelniejsze, gdy będzie wiadomo, czy wartości event_id przedstawiają sprawdziany czy testy. Jednak w celu wygenerowania tego rodzaju informacji konieczne jest użycie również tabeli grade_event. Do tego zapytania powrócimy w podpunkcie 1.4.9.10, zatytułowanym „Pobieranie informacji z wielu tabel”.

Aby wygenerować dodatkowe wiersze danych wyjściowych zawierających „podsumowanie podsumowań”, należy dodać klauzulę `WITH ROLLUP`. W ten sposób baza danych MySQL otrzymuje polecenie obliczenia „superzagregowanych” wartości zgrupowanych rekordów. Oto prosty przykład oparty na wcześniejszym zapytaniu zliczającym uczniów poszczególnych płci. Użycie klauzuli `WITH ROLLUP` powoduje wygenerowanie dodatkowego wiersza podsumowującego wartości uzyskane dla poszczególnych płci.

```
mysql> SELECT sex, COUNT(*) FROM student GROUP BY sex WITH ROLLUP;
```

sex	COUNT(*)
F	15
M	16
NULL	31

Wartość `NULL` w zgrupowanej kolumnie wskazuje, że odpowiadająca jej wartość liczbową jest sumą poprzednich grup.

Klauzula `WITH ROLLUP` może być używana także w innych funkcjach agregujących. Przedstawione poniżej zapytanie tworzy podsumowanie ocen podobne jak przedstawione nieco wcześniej w tekście, ale generuje także dodatkowy wiersz danych wyjściowych zawierający podsumowanie całkowite kolumn.

```
mysql> SELECT
-> event_id,
-> MIN(score) AS minimum,
-> MAX(score) AS maximum,
-> MAX(score)-MIN(score)+1 AS span,
-> SUM(score) AS total,
-> AVG(score) AS average,
-> COUNT(score) AS count
-> FROM score
-> GROUP BY event_id WITH ROLLUP;
```

event_id	minimum	maximum	span	total	average	count
1	9	20	12	439	15.1379	29
2	8	19	12	425	14.1667	30
3	60	97	38	2425	78.2258	31
4	7	20	14	379	14.0370	27
5	8	20	13	383	14.1852	27
6	62	100	39	2325	80.1724	29
NULL	7	100	94	6376	36.8555	173

W powyższych danych wyjściowych ostatni wiersz zawiera zagregowane wartości obliczone na podstawie wszystkich wcześniejszych wartości podsumowań grup.

Klauzula `WITH ROLLUP` jest użyteczna, ponieważ dostarcza informacje dodatkowe, których otrzymanie bez użycia wymienionej klauzuli wymagałoby wykonania dodatkowego zapytania. Użycie tylko pojedynczego zapytania jest znacznie efektywniejszym rozwiązaniem, ponieważ serwer nie musi dwukrotnie analizować danych. Jeżeli w klauzuli `GROUP BY` zostaną podane nazwy więcej niż tylko jednej kolumny, klauzula `WITH ROLLUP` spowoduje

wygenerowanie kolejnych wierszy podsumowania zawierających wartości podsumowania na wyższym poziomie.

1.4.9.10. Pobieranie informacji z wielu tabel

Zaprezentowane dotąd zapytania pobierały dane z pojedynczej tabeli. Możliwości oferowane przez MySQL są jednak znacznie większe. Jak wcześniej wspomniano, potęgą relacyjnego systemu baz danych kryje się w możliwości łączenia informacji pochodzących z wielu tabel w celu udzielenia odpowiedzi za zapytania, na które nie można odpowiedzieć tylko na podstawie danych z jednej tabeli. W tym punkcie dowiesz się, jak tworzyć zapytania pobierające dane z wielu tabel.

Jeden z typów operacji, które pobierają informacje z wielu tabel, nosi nazwę „złączenia”, ponieważ wynik jest otrzymywany przez złączenie informacji pochodzących z różnych tabel. Złączenie następuje przez dopasowanie tych samych wartości w tabelach. Inny typ operacji pobierającej informacje z wielu tabel to zagnieżdżone zapytanie SELECT w innym zapytaniu SELECT. Wspomniane zagnieżdżone zapytanie SELECT nosi nazwę „podzapytania”. W tym punkcie będą omówione oba rodzaje operacji.

Przeanalizujmy przykład złączenia. We wcześniejszej części rozdziału, a dokładnie w podpunkcie 1.4.6.2, zatytułowanym „Tabele dla projektu ocen uczniów”, bez wyjaśnienia przedstawiono zapytanie pobierające wynik sprawdzianu lub testu przeprowadzonego we wskazanym dniu. Teraz nadeszła pora na objaśnienie tego zapytania. Omawiane zapytanie wykorzystuje trzykierunkowe złączenie, więc utworzymy je w dwóch krokach. Pierwszy to utworzenie zapytania pobierającego wyniki z wskazanego dnia:

```
mysql> SELECT student_id, date, score, category
-> FROM grade_event INNER JOIN score
-> ON grade_event.event_id = score.event_id
-> WHERE date = '2012-09-23';
```

student_id	date	score	category
1	2012-09-23	15	Q
2	2012-09-23	12	Q
3	2012-09-23	11	Q
5	2012-09-23	13	Q
6	2012-09-23	18	Q

...

Działanie powyższego zapytania polega na wyszukaniu rekordu `grade_event` o podanej dacie ('2012-09-23') i użyciu identyfikatora zdarzenia tego rekordu do wyszukania ocen o tym samym identyfikatorze zdarzenia. Dla każdego dopasowania rekordów `grade_event` i `score` następuje wyświetlenie identyfikatora ucznia, wyniku, daty i kategorii zdarzenia.

Omawiane zapytanie różni się od innych dotąd przedstawionych pod dwoma ważnymi względami:

- W klauzuli `FROM` podano więcej niż tylko jedną tabelę, ponieważ informacje są pobierane z wielu tabel:

```
FROM grade_event INNER JOIN score
```


- Klauzula `ON` wskazuje, że tabele `grade_event` i `score` są związane, na podstawie dopasowanych wartości `event_id` w obu tabelach:

```
ON grade_event.event_id = score.event_id
```

Zwróć uwagę na odwołanie do kolumn `event_id` w postaci `grade_event.event_id` i `score.event_id` za pomocą składni *`nazwa_tabeli.nazwa_kolumny`*, co pozwala MySQL na użycie właściwych kolumn. Zastosowano wymienione rozwiązanie, ponieważ kolumna `event_id` istnieje w obu tabelach i podanie samej nazwy kolumny byłoby niejednoznaczne. Pozostałe kolumny w zapytaniu (`date`, `score` i `category`) mogą być używane bez specyfikatora tabeli, ponieważ pojawiają się tylko jednokrotnie w tabelach, a więc nie występuje problem niejednoznaczności.

Ogólnie rzecz biorąc, w złączeniach preferuję podawanie nazw tabel we wszystkich przypadkach, przez co zapytanie stosujące złączenie jest czytelniejsze. Od tej chwili zapytania zawierające złączenia będą tworzone w taki właśnie sposób. W pełni kwalifikowana postać omawianego zapytania prezentuje się następująco:

```
SELECT score.student_id, grade_event.date, score.score, grade_event.category
FROM grade_event INNER JOIN score
ON grade_event.event_id = score.event_id
WHERE grade_event.date = '2012-09-23';
```

Zapytanie pierwszego kroku używa tabeli `grade_event` do mapowania daty na identyfikator zdarzenia oraz wykorzystuje wspomniany identyfikator w celu dopasowania ocen w tabeli `score`. Dane wyjściowe zapytania zawierają wartości `student_id`, choć imiona uczniów byłyby znacznie czytelniejsze. Dzięki użyciu tabeli `student` można mapować identyfikator ucznia na jego imię, co jest drugim krokiem operacji. W celu wyświetlenia imienia ucznia wykorzystywany jest fakt, że tabele `score` i `student` posiadają kolumny `student_id` pozwalające na łączenie rekordów między wymienionymi tabelami.

Ostateczna postać zapytania przedstawia się następująco:

```
mysql> SELECT
-> student.name, grade_event.date, score.score, grade_event.category
-> FROM grade_event INNER JOIN score INNER JOIN student
-> ON grade_event.event_id = score.event_id
-> AND score.student_id = student.student_id
-> WHERE grade_event.date = '2012-09-23';
```

```
+-----+-----+-----+-----+
| name   | date   | score | category |
+-----+-----+-----+-----+
| Megan  | 2012-09-23 | 15 | Q |
| Joseph | 2012-09-23 | 12 | Q |
| Kyle   | 2012-09-23 | 11 | Q |
| Abby   | 2012-09-23 | 13 | Q |
| Nathan | 2012-09-23 | 18 | Q |
...

```

Przedstawione zapytanie pod kilkoma względami różni się od poprzedniego:

- Klauzula `FROM` zawiera tabelę `student`, ponieważ zapytanie używa tej tabeli obok wymienionych wcześniej tabel `grade_event` i `score`.

- Kolumna `student_id` była wcześniej jednoznaczna i istniała możliwość odwołania się do niej w sposób niekwalifikowany (`student_id`) lub kwalifikowany (`score.student_id`). Obecnie jest niejednoznaczna, ponieważ kolumna `student_id` istnieje w tabelach `score` i `student`. Dlatego też *konieczne* jest stosowanie formy kwalifikowanej `score.student_id` i `student.student_id`, aby jasno wskazać tabelę, która powinna być użyta.
- Klauzula `ON` ma dodatkowe pojęcie wskazujące, że rekordy tabeli `score` są dopasowywane do rekordów tabeli `student` na podstawie identyfikatora ucznia.
`ON ... score.student_id = student.student_id`
- Zapytanie wyświetla imię ucznia zamiast jego identyfikatora. (Aby wyświetlić zarówno imię, jak i identyfikator ucznia, należy dodać `student.student_id` do listy kolumn danych wyjściowych).

W tak przygotowanym zapytaniu można wskazać dowolną datę i pobrać pełne wyniki uzyskane przez uczniów wskazanego dnia, wraz z imionami uczniów i kategorią ocenianego zdarzenia. Nie trzeba nic wiedzieć na temat identyfikatorów uczniów lub zdarzeń. Baza danych MySQL samodzielnie ustala odpowiednie wartości identyfikatorów, a następnie wykorzystuje je w celu dopasowania rekordów tabel.

We wcześniejszej części rozdziału, a dokładnie w podpunkcie 1.4.9.9, zatytułowanym „Generowanie podsumowania”, przedstawiono zapytanie generujące liczbową charakterystykę danych umieszczonych w tabeli `score`. Dane wyjściowe wspomnianego zapytania zawierały identyfikator zdarzenia, ale nie zawierały dat zdarzeń i ich kategorii, ponieważ nie wiedzieliśmy jeszcze, jak złączyć tabele `score` i `grade_event` w celu mapowania identyfikatorów na daty i kategorie. Teraz już wiemy, jak można to zrobić. Przedstawione poniżej zapytanie jest podobne do użytego wcześniej, ale jego dane wyjściowe wyświetlają daty i kategorie zamiast po prostu liczbowych wartości identyfikatorów zdarzeń:

```
mysql> SELECT
-> grade_event.date, grade_event.category,
-> MIN(score.score) AS minimum,
-> MAX(score.score) AS maximum,
-> MAX(score.score)-MIN(score.score)+1 AS span,
-> SUM(score.score) AS total,
-> AVG(score.score) AS average,
-> COUNT(score.score) AS count
-> FROM score INNER JOIN grade_event
-> ON score.event_id = grade_event.event_id
-> GROUP BY grade_event.date;
```

date	category	minimum	maximum	span	total	average	count
2012-09-03	Q	9	20	12	439	15.1379	29
2012-09-06	Q	8	19	12	425	14.1667	30
2012-09-09	T	60	97	38	2425	78.2258	31
2012-09-16	Q	7	20	14	379	14.0370	27
2012-09-23	Q	8	20	13	383	14.1852	27
2012-10-01	T	62	100	39	2325	80.1724	29

Wprowadź kolumna GROUP BY ma kwalifikator, jednak nie jest absolutnie niezbędna w przedstawionym zapytaniu. Klauzula GROUP BY odwołuje się do kolumn danych wyjściowych, a tam znajduje się tylko jedna kolumna o nazwie date, stąd MySQL wie, która kolumna powinna być użyta.

Istnieje możliwość użycia funkcji takich jak COUNT() i AVG() w celu wygenerowania podsumowania dotyczącego wielu kolumn, nawet pochodzących z różnych tabel. Przedstawione poniżej zapytanie ustala liczbę ocen i przeciętną ocenę dla każdego połączenia daty zdarzenia i płci ucznia:

```
mysql> SELECT grade_event.date, student.sex,
-> COUNT(score.score) AS count, AVG(score.score) AS average
-> FROM grade_event INNER JOIN score INNER JOIN student
-> ON grade_event.event_id = score.event_id
-> AND score.student_id = student.student_id
-> GROUP BY grade_event.date, student.sex;
```

date	sex	count	average
2012-09-03	F	14	14.6429
2012-09-03	M	15	15.6000
2012-09-06	F	14	14.7143
2012-09-06	M	16	13.6875
2012-09-09	F	15	77.4000
2012-09-09	M	16	79.0000
2012-09-16	F	13	15.3077
2012-09-16	M	14	12.8571
2012-09-23	F	12	14.0833
2012-09-23	M	15	14.2667
2012-10-01	F	14	77.7857
2012-10-01	M	15	82.4000

Podobnego zapytania można użyć do wykonania jednego z zadań projektu ocen uczniów, czyli do obliczenia oceny końcowej ucznia na koniec semestru:

```
SELECT student.student_id, student.name,
SUM(score.score) AS total, COUNT(score.score) AS n
FROM grade_event INNER JOIN score INNER JOIN student
ON grade_event.event_id = score.event_id
AND score.student_id = student.student_id
GROUP BY score.student_id
ORDER BY total;
```

Inne zadanie projektu ocen uczniów wymagające utworzenia podsumowania to obliczenie liczby nieobecności. Wspomniane nieobecności są rejestrowane w tabeli absence przy użyciu identyfikatora ucznia i daty. Aby pobrać imiona uczniów (a nie jedynie identyfikatory), konieczne jest zastosowanie złączenia tabel absence i student na podstawie wartości student_id. Przedstawione poniżej zapytanie wyświetla identyfikator ucznia, jego imię i liczbę nieobecności:

```
mysql> SELECT student.student_id, student.name,
-> COUNT(absence.date) AS absences
-> FROM student INNER JOIN absence
```

```

-> ON student.student_id = absence.student_id
-> GROUP BY student.student_id;
+-----+-----+-----+
| student_id | name | absences |
+-----+-----+-----+
|          3 | Kyle |         1 |
|          5 | Abby |         1 |
|         10 | Peter |         2 |
|         17 | Will |         1 |
|         20 | Avery |         1 |
+-----+-----+-----+

```

Dane wyjściowe wygenerowane przez powyższe zapytanie są wystarczające, gdy chcemy wyświetlić jedynie uczniów, którzy mają jakiekolwiek nieobecności. Jeśli tego rodzaju listę przekażesz do sekretariatu szkoły, wtedy możesz usłyszeć pytanie: „A co z pozostałymi uczniami? Proszę dostarczyć liczby nieobecności dla wszystkich uczniów”. To jest nieco inne pytanie, które dotyczy liczby nieobecności nawet w przypadku uczniów zawsze obecnych na zajęciach. Ponieważ pytanie jest inne, to również zapytanie udzielające na nie odpowiedzi będzie inne.

Aby udzielić odpowiedzi na wymienione pytanie, należy użyć klauzuli `LEFT JOIN` zamiast złączenia wewnętrznego. Klauzula `LEFT JOIN` nakazuje MySQL wygenerowanie rekordu danych wyjściowych dla każdego rekordu pobranego z tabeli wymienionej jako pierwsza w złączeniu (to znaczy po lewej stronie słów kluczowych `LEFT JOIN`). Wymieniając tabelę `student` jako pierwszą, otrzymamy dane wyjściowe dla każdego ucznia, nawet nieumieszczonego w tabeli `absence`. W celu utworzenia zapytania klauzulę `LEFT JOIN` należy umieścić między nazwami tabel wymienionymi w klauzuli `FROM` (zamiast rozdzielacze przecinkami) i klauzulą `ON`, wskazującą sposób dopasowania rekordów w obu tabelach. Zapytanie przedstawia się następująco:

```

mysql> SELECT student.student_id, student.name,
-> COUNT(absence.date) AS absences
-> FROM student LEFT JOIN absence
-> ON student.student_id = absence.student_id
-> GROUP BY student.student_id;
+-----+-----+-----+
| student_id | name   | absences |
+-----+-----+-----+
|          1 | Megan  |         0 |
|          2 | Joseph |         0 |
|          3 | Kyle   |         1 |
|          4 | Katie  |         0 |
|          5 | Abby   |         1 |
|          6 | Nathan |         0 |
|          7 | Liesl  |         0 |
...

```

Nie jest wymagane, aby złączenie było przeprowadzane między różnymi tabelami. W pierwszej chwili to może wydawać się dziwne, ale istnieje możliwość przeprowadzenia złączenia w jednej tabeli. Na przykład, w celu ustalenia, czy jakikolwiek prezydent urodził się w mieście urodzenia innego prezydenta, można porównać ich miejsca urodzenia:

```
mysql> SELECT p1.last_name, p1.first_name, p1.city, p1.state
-> FROM president AS p1 INNER JOIN president AS p2
-> ON p1.city = p2.city AND p1.state = p2.state
-> WHERE (p1.last_name <> p2.last_name OR p1.first_name <> p2.first_name)
-> ORDER BY state, city, last_name;
```

last_name	first_name	city	state
Adams	John Quincy	Braintree	MA
Adams	John	Braintree	MA

Trzeba pamiętać o dwóch kwestiach związanych z tego rodzaju zapytaniem:

- Konieczne jest odwołanie się do dwóch egzemplarzy tej samej tabeli. Dlatego w tym celu utworzono aliasy tabeli (p1 i p2), a następnie użyto ich w odniesieniach do kolumn tabeli. Podobnie jak w przypadku aliasów kolumn, słowo kluczowe AS jest opcjonalne podczas definiowania aliasów tabel.
- Każdy rekord prezydenta powoduje dopasowanie samego siebie, ale tego nie chcemy widzieć w danych wyjściowych. Klauzula WHERE uniemożliwia rekordowi dopasowanie samego siebie przez zagwarantowanie, że porównywane rekordy należą do różnych prezydentów.

Podobne zapytanie wyszukuje prezydentów urodzonych tego samego dnia roku. Jednak dat nie można porównywać bezpośrednio, ponieważ w takim przypadku nie zostaną dopasowani prezydenci urodzeni w różnych latach. Zamiast tego używamy funkcji MONTH() i DAYOFMONTH() w celu porównania dnia i miesiąca daty urodzenia:

```
mysql> SELECT p1.last_name, p1.first_name, p1.birth
-> FROM president AS p1 INNER JOIN president AS p2
-> WHERE MONTH(p1.birth) = MONTH(p2.birth)
-> AND DAYOFMONTH(p1.birth) = DAYOFMONTH(p2.birth)
-> AND (p1.last_name <> p2.last_name OR p1.first_name <> p2.first_name)
-> ORDER BY p1.last_name;
```

last_name	first_name	birth
Harding	Warren G.	1865-11-02
Polk	James K.	1795-11-02

Użycie funkcji DAYOFYEAR() zamiast połączenia MONTH() i DAYOFMONTH() spowoduje uproszczenie zapytania, ale jednocześnie wygeneruje nieprawidłowe wyniki podczas porównywania dat z lat przestępnych z datami z lat nieprzestępnych.

Inny rodzaj pobierania danych z wielu tabel opiera się na wykorzystaniu podzapytania, czyli zapytania SELECT zagnieżdżonego w innym zapytaniu SELECT. Mamy jeszcze inne typy podzapytań, ale one zostaną dokładnie omówione w podrozdziale 2.9, zatytułowanym „Pobieranie informacji z wielu tabel za pomocą podzapytań”. Teraz zobaczysz kilka przykładów tego typu zapytań. Przyjmujemy założenie, że chcesz wyświetlić uczniów o najmniejszej liczbie nieobecności. Odpowiada to ustaleniu, którzy uczniowie nie są wymienieni w tabeli absence, do czego można użyć następującego zapytania:

```
mysql> SELECT * FROM student
-> WHERE student_id NOT IN (SELECT student_id FROM absence);
```

name	sex	student_id
Megan	F	1
Joseph	M	2
Katie	F	4
Nathan	M	6
Liesl	F	7

...

Zagnieżdżone zapytanie SELECT wybiera zestaw wartości student_id istniejących w tabeli absence, a następnie zewnętrzne zapytanie SELECT pobiera z tabeli student rekordy niedopasowane do wspomnianych identyfikatorów.

Podzapytanie może być również opartym na jednym zapytaniu rozwiązaniem problemu przedstawionego w podpunkcie 1.4.9.8, zatytułowanym „Konfiguracja i używanie zmiennych zdefiniowanych przez użytkownika” i dotyczącym wyboru prezydentów urodzonych przed Andrew Jacksonem. Przedstawione w wymienionym punkcie rozwiązanie używało dwóch poleceń i zmiennej użytkownika. Rozwiązanie alternatywne opiera się na podzapytaniu:

```
mysql> SELECT last_name, first_name, birth FROM president
-> WHERE birth < (SELECT birth FROM president
-> WHERE last_name = 'Jackson' AND first_name = 'Andrew');
```

last_name	first_name	birth
Washington	George	1732-02-22
Adams	John	1735-10-30
Jefferson	Thomas	1743-04-13
Madison	James	1751-03-16
Monroe	James	1758-04-28

Wewnętrzne zapytanie SELECT ustala datę urodzenia Andrew Jacksona, natomiast zewnętrzne pobiera rekordy prezydentów urodzonych wcześniej.

1.4.10. Usuwanie lub uaktualnianie istniejących rekordów

Czasami zachodzi potrzeba pozbycia się rekordów lub zmiany ich zawartości. Do tego celu służą zapytania DELETE i UPDATE.

Zapytanie DELETE ma następującą składnię:

```
DELETE FROM nazwa_tabeli
WHERE rekordy do usunięcia;
```

Klauzula WHERE wskazująca rekordy do usunięcia jest opcjonalna, ale jej pominięcie spowoduje usunięcie wszystkich rekordów w tabeli. Innymi słowy, najprostsze zapytanie DELETE jest jednocześnie najniebezpieczniejsze:

```
DELETE FROM nazwa_tabeli;
```

Powyższe zapytanie spowoduje całkowite usunięcie zawartości tabeli, więc zachowaj ostrożność! Aby usunąć jedynie wybrane rekordy, należy użyć klauzuli WHERE i wskazać rekordy przeznaczone do usunięcia. Przypomina to użycie klauzuli WHERE w zapytaniu SELECT w celu uniknięcia wyboru całej tabeli. Na przykład, aby z tabeli `president` usunąć jedynie prezydentów urodzonych w Ohio, trzeba wykonać poniższe zapytanie:

```
mysql> DELETE FROM president WHERE state='OH';
Query OK, 7 rows affected (0.01 sec)
```

Jeżeli nie jesteś pewien, które rekordy będą usunięte przez zapytanie DELETE, najpierw przetestuj klauzulę WHERE z zapytaniem SELECT, a poznasz dopasowane przez nią rekordy. W ten sposób upewnisz się, że usunięte zostaną tylko te rekordy, których faktycznie chcesz się pozbyć. Przyjmujemy założenie, że chcesz usunąć rekord dotyczący prezydenta Teddiego Roosevelta. Czy możesz to zrobić za pomocą poniższego zapytania?

```
DELETE FROM president WHERE last_name='Roosevelt';
```

Tak, w tym sensie, że usunięty będzie rekord wymienionego prezydenta. Nie, w tym sensie, że usunięty zostanie również rekord dotyczący prezydenta Franklina Roosevelta. Najbezpieczniejszym rozwiązaniem będzie wcześniejsze sprawdzenie klauzuli WHERE w zapytaniu SELECT:

```
mysql> SELECT last_name, first_name FROM president
-> WHERE last_name='Roosevelt';
+-----+-----+
| last_name | first_name |
+-----+-----+
| Roosevelt | Theodore   |
| Roosevelt | Franklin D. |
+-----+-----+
```

Wyniki zapytania pokazują, że trzeba dokładniej wskazać rekord do usunięcia, a więc podać jeszcze imię prezydenta:

```
mysql> SELECT last_name, first_name FROM president
-> WHERE last_name='Roosevelt' AND first_name='Theodore';
+-----+-----+
| last_name | first_name |
+-----+-----+
| Roosevelt | Theodore   |
+-----+-----+
```

Kiedy przekonaś się, że klauzula WHERE prawidłowo wskazuje rekordy przeznaczone do usunięcia, możesz przystąpić do utworzenia poprawnego zapytania DELETE:

```
mysql> DELETE FROM president
WHERE last_name='Roosevelt' AND first_name='Theodore';
```

Być może sądzisz, że usunięcie rekordu wymaga dużej ilości kodu do wpisania. Pamiętaj jednak, że lepiej zrobić to bezpiecznie, niż później żałować usunięcia niewłaściwych danych! Ponadto, w sytuacjach takich jak przedstawiona powyżej można wykorzystać technikę „kopiuj i wklej” lub edycji tekstu zapytania. Więcej informacji na ten temat

znajdziesz w podrozdziale 1.5, zatytułowanym „Wskazówki przydatne podczas pracy z klientem mysql”.

Aby zmodyfikować istniejące rekordy, należy użyć zapytania UPDATE, które ma następującą składnię:

```
UPDATE nazwa_tabeli
SET kolumny do zmiany
WHERE rekordy do uaktualnienia;
```

Klauzula WHERE działa podobnie jak w przypadku zapytania DELETE. Jest opcjonalna, więc jej pominięcie spowoduje uaktualnienie wszystkich rekordów w tabeli. Na przykład, poniższe zapytanie spowoduje zmianę imienia każdego ucznia na „George”:

```
mysql> UPDATE student SET name='George';
```

Oczywiście, należy zachować ostrożność podczas stosowania tego typu zapytań. Zwykle dodawana jest klauzula WHERE, precyzyjnie wskazująca rekordy przeznaczone do uaktualnienia. Przyjmijmy założenie, że do Ligi Historycznej ostatnio dodano nowego członka Ligi, ale wypełniono jedynie kilka kolumn w dotyczącym go rekordzie:

```
mysql> INSERT INTO member (last_name,first_name)
-> VALUES('York','Jerome');
```

Następnie zdałeś sobie sprawę, że zapomniałeś ustawić datę wygaśnięcia członkostwa. Można to naprawić za pomocą zapytania UPDATE zawierającego odpowiednią klauzulę WHERE dokładnie wskazującą modyfikowany rekord:

```
mysql> UPDATE member
-> SET expiration='2013-7-20'
-> WHERE last_name='York' AND first_name='Jerome';
```

Istnieje możliwość uaktualnienia wielu kolumn w pojedynczym zapytaniu. Przedstawione poniżej zapytanie UPDATE modyfikuje adres e-mail oraz pocztowy członka Ligi o imieniu Jerome:

```
mysql> UPDATE member
-> SET email='jerome@aol.com', street='123 Elm St',
-> city='Anytown', state='NY', zip='01003'
-> WHERE last_name='York' AND first_name='Jerome';
```

Istnieje także możliwość usunięcia wartości w kolumnie i przypisania jej wartości NULL (o ile kolumna pozwala na ich stosowanie). Jeżeli w pewnym momencie Jerome zdecyduje się na opłacenie członkostwa dożywotniego, jego rekord można oznaczyć jako członkostwo dożywotnie przez ustawienie wartości NULL w kolumnie expiration:

```
mysql> UPDATE member
-> SET expiration=NULL
-> WHERE last_name='York' AND first_name='Jerome';
```

W przypadku zapytania UPDATE, podobnie jak DELETE, dobrym pomysłem jest przetestowanie klauzuli WHERE w zapytaniu SELECT, aby przekonać się o wyborze odpowiednich rekordów do uaktualnienia. W przeciwnym razie, jeśli kryterium jest zbyt szerokie lub wąskie, to nastąpi uaktualnienie zbyt dużej lub małej liczby rekordów.

Jeżeli wypróbowałeś zapytania przedstawione w tym punkcie, tym samym usunąłeś i zmodyfikowałeś pewne rekordy w tabelach bazy danych sampdb. Przed przejściem do kolejnego podrozdziału powinieneś cofnąć wspomniane zmiany. Najlepszym rozwiązaniem jest ponowne wczytanie danych do tabel za pomocą poleceń przedstawionych wcześniej w punkcie 1.4.8, zatytułowanym „Przywrócenie bazy danych sampdb do znanego stanu”.

1.5. Wskazówki przydatne podczas pracy z klientem mysql

W tym podrozdziale dowiesz się, jak efektywniej pracować z programem klienckim mysql oraz wpisywać mniejszą liczbę zapytań. Przedstawiony zostanie też łatwiejszy sposób nawiązywania połączenia z serwerem, a także sposoby wprowadzania poleceń bez konieczności ich każdorazowego wpisywania.

1.5.1. Uproszczenie procesu nawiązywania połączenia

Kiedy wywołujesz program mysql, prawdopodobnie musisz podać parametry połączenia, czyli między innymi nazwy komputera i użytkownika oraz hasło. To oznacza sporą ilość danych do wprowadzenia w celu jedynie uruchomienia programu i szybko stanie się uciążliwe. Istnieje kilka sposobów ograniczenia ilości danych koniecznych do wprowadzenia, aby nawiązać połączenie z serwerem MySQL:

- przechowywanie parametrów połączenia w pliku opcji;
- powtórzenie poleceń przy użyciu funkcji historii używanej powłoki;
- zdefiniowanie skrótu wiersza poleceń mysql za pomocą aliasu powłoki lub skryptu.

1.5.1.1. Użycie pliku opcji

Baza danych MySQL pozwala na przechowywanie parametrów połączenia w pliku opcji. W takim przypadku nie musisz ich wprowadzać za każdym razem podczas uruchamiania klienta mysql; są używane w dokładnie taki sam sposób, jakbyś ręcznie wprowadził je w wierszu poleceń. Ogromną zaletą tej techniki jest możliwość wykorzystania tych parametrów także w innych klientach MySQL, takich jak mysql import lub mysql show. Innymi słowy, plik opcji ułatwia użycie nie tylko mysql, ale również innych programów. W tym punkcie pokrótce przedstawiono konfigurację pliku opcji i przygotowanie go do użycia przez programy klienckie. Więcej informacji szczegółowych na temat pliku opcji znajdziesz w podrozdziale F.2.2, zatytułowanym „Pliki opcji”.

W systemach UNIX konfigurację pliku opcji rozpocznij od utworzenia pliku o nazwie `~/.my.cnf` (czyli pliku o nazwie `.my.cnf` umieszczonego w katalogu domowym). Z kolei w Windows utwórz plik o nazwie `my.ini` w katalogu instalacyjnym MySQL lub w katalogu głównym dysku C (to znaczy `C:\my.ini`). Plik opcji to zwykły plik tekstowy, do jego edycji można wykorzystać dowolny edytor tekstowy. Zawartość pliku powinna być w poniższym formacie:

```
[client]
host=komputer_serwera
user=nazwa_uzytkownika
password=haslo
```

Wiersz `[client]` oznacza początek grupy opcji `client`. Programy klienckie MySQL odczytują kolejne wiersze w celu pobrania wartości opcji. Wiersze są odczytywane aż do końca pliku lub do początku kolejnej grupy opcji. W miejsce *komputer_serwera*, *nazwa_uzytkownika* i *haslo* podaj odpowiednie dane, czyli komputer, w którym działa serwer MySQL, nazwę użytkownika i hasło używane podczas nawiązywania połączenia z serwerem. Jeżeli serwer działa w komputerze o nazwie *cobra.example.com*, nazwa użytkownika MySQL to *sampadm*, a jego hasło to *secret*, wówczas zawartość pliku *.my.cnf* powinna przedstawiać się następująco:

```
[client]
host=cobra.example.com
user=sampadm
password=secret
```

Wiersz `[client]` jest wymagany w celu wskazania początku grupy opcji, natomiast wiersze definiujące wartości parametrów są opcjonalne i możesz podać tylko te, które chcesz. Na przykład, jeśli używasz systemu UNIX i nazwa użytkownika MySQL jest taka sama jak nazwa logowania do systemu, wówczas nie ma konieczności umieszczania wiersza `user`. Domyślny komputer serwera to `localhost` i jeśli nawiązujesz połączenie z serwerem MySQL uruchomionym w komputerze lokalnym, wtedy możesz pominąć wiersz `host`.

W systemach UNIX trzeba pamiętać, aby po utworzeniu pliku opcji zdefiniować mu najbardziej restrykcyjne prawa dostępu. W ten sposób nikt poza właścicielem pliku nie będzie mógł go odczytywać lub modyfikować. Wydanie dowolnego z poniższych poleceń powoduje, że plik będzie dostępny tylko dla jego właściciela:

```
% chmod 600 .my.cnf
% chmod u=rw,go-rwx .my.cnf
```

1.5.1.2. Użycie funkcji historii w powłoce

Powłoki takie jak `tcsh` i `bash` umieszczają na liście historii wydane polecenia i pozwalają na ich ponowny wybór z listy. Jeżeli używasz jednej z tego typu powłok, to dzięki liście historii możesz uniknąć konieczności wpisywania całych poleceń. Na przykład, jeśli ostatnio wywołałeś `mysql`, to polecenie możesz ponownie wywołać w następujący sposób:

```
% !my
```

Znak `!` nakazuje powłoce przeszukanie listy historii poleceń, odszukanie ostatniego rozpoczynającego się od znaków `my` i jego wykonanie tak, jakbyś ponownie je wprowadził z klawiatury. W niektórych powłokach istnieje nawet możliwość poruszania się po liście historii za pomocą klawiszy kursora w górę i w dół (lub `Ctrl+P` i `Ctrl+N`). Po wybraniu polecenia w taki sposób naciśnięcie klawisza `Enter` powoduje jego wykonanie. Wspomnianą funkcjonalność na pewno znajdziesz w powłokach `bash` i `tcsh`, a także wielu innych. Sprawdź dokumentację powłoki, aby dowiedzieć się więcej na temat zaimplementowanej w niej funkcji listy historii.

1.5.1.3. Użycie aliasów powłoki i skryptów

Używana powłoka może oferować możliwość tworzenia aliasów, które pozwalają na mapowanie długich poleceń na ich krótkie odpowiedniki. Na przykład, w powłokach `csh` i `tcsh` można użyć polecenia `alias` w celu utworzenia aliasu o nazwie `sampdb`:

```
alias sampdb 'mysql -h cobra.example.com -p -u sampadm sampdb'
```

Składnia stosowana w powłoce `bash` jest nieco inna:

```
alias sampdb='mysql -h cobra.example.com -p -u sampadm sampdb'
```

Zdefiniowanie powyższego aliasu powoduje, że dwa polecenia przedstawione poniżej działają dokładnie w taki sam sposób:

```
% sampdb
% mysql -h cobra.example.com -p -u sampadm sampdb
```

Bez wątpienia pierwsze z nich jest łatwiejsze do wprowadzenia od drugiego. Aby alias był dostępny zawsze po zalogowaniu, polecenie `alias` należy umieścić w pliku startowym powłoki (na przykład `.tcshrc` dla powłoki `tcsh` i `.bashrc` lub `.bash_profile` dla powłoki `bash`).

W systemie Windows podobna technika pozwala na utworzenie skrótu prowadzącego do programu `mysql`. Następnie we właściwościach skrótu trzeba umieścić odpowiednie parametry połączenia.

Innym sposobem wywoływania poleceń przy jednocześnie mniejszej ilości wprowadzanego tekstu jest utworzenie skryptu uruchamiającego program `mysql` wraz z odpowiednimi opcjami. W systemach UNIX zaprezentowany poniżej skrypt powłoki będzie odpowiednikiem przedstawionego wcześniej aliasu o nazwie `sampdb`:

```
#!/bin/sh
exec mysql -h cobra.example.com -p -u sampadm sampdb
```

Jeżeli skryptowi nadasz nazwę `sampdb` i zdefiniujesz dla niego prawo uruchamiania (wydając polecenie `chmod +x sampdb`), wówczas wydanie polecenia `sampdb` po znaku zachęty spowoduje uruchomienie programu `mysql` i nawiązanie połączenia z bazą danych `sampdb`.

W systemie Windows do tego samego celu służy plik wsadowy. Utwórz plik o nazwie `sampdb.bat` i umieść w nim następujący wiersz:

```
mysql -h cobra.example.com -p -u sampadm sampdb
```

Ten plik wsadowy można uruchomić, podając jego nazwę w wierszu poleceń lub dwukrotnie klikając jego ikonę Windows.

Jeżeli zachodzi potrzeba uzyskania dostępu do wielu baz danych lub nawiązania połączenia z wieloma serwerami, wtedy można zdefiniować wiele aliasów, skrótów lub skryptów, z których każdy będzie wywoływał program `mysql` wraz z odpowiednimi opcjami.

1.5.2. Wpisywanie mniejszej ilości tekstu przy wykonywaniu zapytań

Program `mysql` jest wyjątkowo użyteczny podczas pracy z bazą danych, ale jego interfejs najlepiej sprawdza się do wykonywania krótkich zapytań, składających się z pojedynczego

wiersza. Wprawdzie mysql pozwala na stosowanie zapytań obejmujących wiele wierszy, ale ich wprowadzanie nie należy do przyjemności. Ponadto, irytujące jest wprowadzenie zapytania tylko po to, aby się przekonać o konieczności jego ponownego wpisania z powodu błędu w składni. Wymienione poniżej techniki pomagają w uniknięciu niepotrzebnego ponownego wpisywania zapytań:

- użycie oferowanych przez mysql możliwości edycji wiersza danych wejściowych;
- użycie techniki „kopiuj i wklej”;
- uruchomienie mysql w trybie wsadowym.

1.5.2.1. Użycie oferowanych przez mysql możliwości edycji wiersza danych wejściowych

Program mysql pozwala na edycję wiersza danych wejściowych. Oznacza to możliwość manipulacji aktualnie wprowadzanym wierszem, przywoływania poprzednich wierszy danych wejściowych i użycia w niezmienionej postaci lub po dalszej modyfikacji. Takie rozwiązanie jest wygodne po wykryciu błędu we wprowadzanym wierszu; po prostu usuwasz znaleziony błąd przed naciśnięciem klawisza *Enter*. Jeżeli wprowadziłeś zapytanie z błędem, wtedy możesz je przywołać, usunąć błąd, a następnie ponownie wykonać. (Najłatwiej jest, jeśli całe zapytanie zostało wprowadzone w pojedynczym wierszu).

W tabeli 1.4 wymieniono pewne kombinacje klawiszy, które są użyteczne w systemach UNIX.

Tabela 1.4. Polecenia edycji wiersza danych wejściowych w mysql

Kombinacja klawiszy	Opis
Kursor w górę lub <i>Ctrl+P</i>	Przywołanie poprzedniego wiersza.
Kursor w dół lub <i>Ctrl+N</i>	Przywołanie kolejnego wiersza.
Kursor w lewo lub <i>Ctrl+B</i>	Przeniesienie kursora w lewo (do tyłu).
Kursor w prawo lub <i>Ctrl+F</i>	Przeniesienie kursora w prawo (do przodu).
<i>Escape b</i>	Przeniesienie kursora do tyłu o jedno słowo.
<i>Escape f</i>	Przeniesienie kursora do przodu o jedno słowo.
<i>Ctrl+A</i>	Przeniesienie kursora na początek wiersza.
<i>Ctrl+E</i>	Przeniesienie kursora na koniec wiersza.
<i>Ctrl+D</i>	Usunięcie znaku znajdującego się w miejscu kursora.
<i>Delete</i>	Usunięcie znaku znajdującego się po lewej stronie kursora.
<i>Escape D</i>	Usunięcie słowa.
<i>Escape Backspace</i>	Usunięcie słowa znajdującego się po lewej stronie kursora.
<i>Ctrl+K</i>	Usunięcie wszystkiego od kursora do końca wiersza.

W systemie Windows możliwości edycji wiersza danych wejściowych nie są oferowane przez mysql. Jednak sam system Windows obsługuje polecenia wymienione w tabeli 1.5, a tym samym są one dostępne także w programie mysql.

Tabela 1.5. Polecenia edycji wiersza danych wejściowych w Windows

Kombinacja klawiszy	Opis
Kursor w górę	Przywołanie poprzedniego wiersza.
Kursor w dół	Przywołanie kolejnego wiersza.
Kursor w lewo	Przeniesienie kursora w lewo (do tyłu).
Kursor w prawo	Przeniesienie kursora w prawo (do przodu).
Ctrl+kursor w lewo	Przeniesienie kursora do tyłu o jedno słowo.
Ctrl+kursor w prawo	Przeniesienie kursora do przodu o jedno słowo.
Home	Przeniesienie kursora na początek wiersza.
End	Przeniesienie kursora na koniec wiersza.
Delete	Usunięcie znaku znajdującego się w miejscu kursora.
Backspace	Usunięcie znaku znajdującego się po lewej stronie kursora.
Esc	Usunięcie wiersza.
Page up	Przywołanie pierwszego wprowadzonego polecenia.
Page down	Przywołanie ostatniego wprowadzonego polecenia.
F3	Przywołanie ostatniego wprowadzonego polecenia.
F7	Wyświetlenie okna poleceń; odpowiednie można wybrać za pomocą kursora w górę i dół.
F9	Wyświetlenie okna poleceń; odpowiednie można wybrać, podając jego numer.
F8, F5	Iteracja przez listę poleceń.

Poniższy przykład pokazuje proste użycie możliwości edycji wiersza danych wejściowych. Przyjmujemy założenie, że w programie mysql wprowadzono poniższe zapytanie:

```
mysql> SHOW COLUMNS FROM persident;
```

Jak możesz dostrzec, błąd popełniono w nazwie tabeli — zamiast `president` jest `persident`. Przed naciśnięciem klawisza *Enter* ten błąd można usunąć w następujący sposób:

1. Naciśnij kilkakrotnie klawisz kursora w lewo, aż znajdzie się na literze `s`.
2. W celu usunięcia liter `er` należy dwukrotnie nacisnąć klawisz *Delete* lub *Backspace* (oba wymienione powodują usunięcie znaku znajdującego się po lewej stronie kursora).

3. Wpisz `re`, poprawiając tym samym błąd.
4. Naciśnij klawisz `Enter` i wykonaj zapytanie.

Jeżeli naciśniesz klawisz `Enter` przed wykryciem błędu, to także nie będzie problemem. Po wyświetleniu przez `mysql` komunikatu błędu naciśnij kursor w górę, przywołując ostatnie zapytanie, a następnie przeprowadź jego edycję zgodnie z przedstawionym powyżej opisem.

1.5.2.2. Użycie techniki „kopiuj i wklej”

Jeżeli używasz środowiska opartego na oknach, wówczas polecenia tekstowe, które uznasz za użyteczne, możesz zapisywać w pliku, a następnie ponownie je wykorzystywać za pomocą techniki „kopiuj i wklej”:

1. Uruchom program `mysql` w oknie terminala lub konsoli.
2. Otwórz plik zawierający zapytania (na przykład, w systemach UNIX używam do tego edytora `vi`, natomiast w Windows edytora `gvim`).
3. W celu wykonania zapytania zapisanego w pliku zaznacz je i skopiuj do schowka. Następnie powróć do okna terminala i wklej zapytanie w `mysql`.

Przedstawiona powyżej procedura wydaje się kłopotliwa, ale kiedy faktycznie ją zastosujesz, przekonasz się, że pozwala na szybkie wprowadzanie poleceń bez konieczności ich wpisywania. Po odrobinie praktyki zaprezentowana procedura może stać się Twoją drugą naturą.

Technikę „kopiuj i wklej” można również stosować w drugą stronę (to znaczy kopiować zapytania w oknie terminala i wklejać je w przygotowanym pliku). W systemach UNIX wykonywane w `mysql` zapytania są zapisywane w pliku o nazwie `.mysql_history`, który znajduje się w katalogu domowym użytkownika. Jeżeli ręcznie wprowadziłeś zapytanie, które chcesz zachować na przyszłość, zakończ pracę z programem `mysql`, otwórz plik `.mysql_history` w edytorze tekstów, skopiuj żądane zapytanie z wymienionego pliku do schowka, a następnie wklej je w przygotowanym pliku.

1.5.2.3. Użycie `mysql` do uruchamiania plików skryptów

Nie ma konieczności interaktywnego uruchamiania programu `mysql`, ponieważ potrafi on również odczytywać dane wejściowe z pliku w trybie nieinteraktywnym (wsadowym). To jest użyteczna możliwość w przypadku okresowo wykonywanych poleceń, ponieważ nie trzeba ich wpisywać za każdym razem. Umieść zapytania w pliku raz, a następnie pozwól programowi `mysql` na wykonywanie zawartości pliku, gdy zajdzie potrzeba.

Przyjmujemy założenie, że masz zapytanie wyszukujące członków Ligi Historycznej zainteresowanych konkretnym tematem w historii USA. Operacja polega na wyszukaniu odpowiednich informacji w kolumnie `interests` tabeli `member`. Na przykład, aby wyszukać członków Ligi interesujących się Wielkim Kryzysem, należy wykonać poniższe zapytanie:

```
SELECT last_name, first_name, email, interests FROM member
WHERE interests LIKE '%depression%'
ORDER BY last_name, first_name;
```

Po umieszczeniu przedstawionego zapytania w pliku *interests.sql* można je wykonać w następujący sposób:

```
% mysql sampdb < interests.sql
```

Po uruchomieniu w trybie wsadowym program mysql domyślnie wyświetla dane wyjściowe w formacie, w którym poszczególne dane są oddzielone tabulatorem. Jeżeli chcesz uzyskać dane wyjściowe w postaci tabeli, czyli podobnie jak podczas interaktywnego używania mysql, wtedy trzeba użyć opcji -t:

```
% mysql -t sampdb < interests.sql
```

Aby zapisać dane wyjściowe, należy je przekierować do pliku:

```
% mysql -t sampdb < interests.sql > interests.out
```

Jeżeli program mysql jest już uruchomiony, wykonanie zawartości pliku następuje po wykonaniu zapytania SOURCE:

```
mysql> SOURCE interests.sql
```

W celu użycia poprzedniego zapytania do wyszukania członków Ligi interesujących się Thomasem Jeffersonem można przeprowadzić edycję zapytania w pliku i zmienić słowo *depression* na *Jefferson*, a następnie ponownie wykonać je w mysql. Takie rozwiązanie sprawdza się doskonale, o ile zapytania nie używasz zbyt często. W przeciwnym razie do dyspozycji masz znacznie lepsze metody. W systemach UNIX jednym ze sposobów zwiększenia elastyczności zapytania jest jego umieszczenie w skrypcie powłoki pobierającym argumenty, które następnie są wstawiane w tekst zapytania. W ten sposób powstaje sparаметryzowane zapytanie pozwalające na przykład na podanie wartości kolumny *interests* podczas uruchamiania skryptu. Aby przekonać się, jak takie rozwiązanie działa, utworzymy niewielki skrypt powłoki o nazwie *interests.sh*:

```
#!/bin/sh
# interests.sh - wyszukanie członków USHL o określonych zainteresowaniach
if [ $# -ne 1 ]; then echo 'Proszę podać jedno słowo kluczowe'; exit; fi
mysql -t sampdb <<QUERY_INPUT
SELECT last_name, first_name, email, interests FROM member
WHERE interests LIKE '%$1%'
ORDER BY last_name, first_name;
QUERY_INPUT
```

Polecenie umieszczone w wierszu trzecim gwarantuje pobranie tylko jednego argumentu wiersza poleceń; w przypadku podania ich większej liczby nastąpi wyświetlenie odpowiedniego komunikatu i zakończenie działania skryptu. Cały tekst znajdujący się między <<QUERY_INPUT i QUERY_INPUT staje się danymi wejściowymi dla mysql. W tekście zapytania skrypt zastępuje odwołanie do \$1 argumentem podanym w powłocie. (W skryptach powłoki zmienne \$1, \$2, ... odwołują się do argumentów polecenia). W ten sposób w zapytaniu zostanie użyte słowo kluczowe podane jako argument dla uruchamianego skryptu.

Przed uruchomieniem skryptu należy się upewnić o nadaniu mu uprawnień do wykonywania:

```
% chmod +x interests.sh
```

Teraz nie musisz już przeprowadzać edycji skryptu przed jego każdorazowym uruchomieniem. To, czego szukasz, po prostu podaj w wierszu powłoki:

```
% ./interests.sh depression
% ./interests.sh Jefferson
```

Skrypt *interests.sh* znajduje się w katalogu *misc* dystrybucji *sampdb*. Ponadto dostarczono plik *interests.bat* będący odpowiadającym mu plikiem wsadowym dla systemu Windows.

Uwaga

Zalecam, aby skryptów takich jak przedstawiony powyżej nie instalować jako dostępnych publicznie, ponieważ nie przeprowadzają one żadnego sprawdzenia argumentów, a tym samym są podatne na ataki typu SQL injection. Przyjmijmy założenie, że użytkownik w następujący sposób uruchomi skrypt:

```
% ./interests.sh "Jefferson";DROP DATABASE sampdb;"
```

Efektom będzie „wstrzyknięcie” zapytania `DROP DATABASE` do danych wejściowych programu `mysql` i jego rzeczywiste wykonanie.

1.6. Co dalej?

Na tym etapie masz już podstawową wiedzę z zakresu bazy danych MySQL. Potrafisz skonfigurować bazę danych i utworzyć tabele. Wiesz, jak dodawać rekordy do wspomnianych tabel, pobierać je na różne sposoby, zmieniać, a nawet usuwać. Jednak przewodnik przedstawiony w rozdziale zaprezentował jedynie ułamek możliwości MySQL i nadal sporo pozostało do poznania. Aby się o tym przekonać, rozważ bieżący stan przykładowej bazy danych *sampdb*. Utworzyliśmy tę bazę danych i jej tabele, które następnie wypełniliśmy danymi początkowymi. W trakcie tego procesu przekonałeś się, jak tworzyć zapytania niezbędne do udzielenia odpowiedzi dotyczących informacji znajdujących się w bazie danych. Wiele pozostało jeszcze do zrobienia. Na przykład, brakuje wygodnego, interaktywnego sposobu wprowadzania nowych rekordów ocen w projekcie ocen uczniów lub nowych członków w projekcie Ligi Historycznej. Brakuje też wygodnego sposobu edycji istniejących rekordów. Nadal nie możemy wygenerować katalogu Ligi Historycznej w postaci gotowej do wydruku lub formularza sieciowego. Do wymienionych zadań i wielu innych powrócimy w kolejnych rozdziałach, a zwłaszcza w 8., „Tworzenie programów MySQL przy użyciu Perl DBI”, i 9., „Tworzenie programów MySQL przy użyciu języka PHP”.

Rozdział, którego lekturę teraz rozpocznesz, zależy od Twoich zainteresowań. Jeśli chcesz się dowiedzieć, jak zakończyć rozpoczętą pracę nad projektami Ligi Historycznej i ocen uczniów, to w części drugiej książki znajdziesz omówienie sposobu tworzenia programów opartych na bazie danych MySQL. Jeżeli chcesz zająć się administracją MySQL, omówienie odpowiednich zagadnień znajdziesz w części trzeciej. Jednak proponuję wcześniej zdobyć dodatkowej wiedzy na temat używania MySQL dzięki lekturze pozostałych rozdziałów w części pierwszej. W tych rozdziałach przedstawiono dalsze informacje na temat składni i użycia poleceń SQL, przeanalizowano sposób obsługi

danych przez MySQL, a także zaprezentowano, jak zoptymalizować zapytania, aby były szybciej wykonywane. Dobre zrozumienie tych zagadnień zapewni Ci solidne podstawy przydatne niezależnie od kontekstu, w jakim chcesz używać MySQL — praca w programie klienckim `mysql`, tworzenie własnych programów lub praca w charakterze administratora baz danych.

Użycie MySQL do zarządzania danymi

Serwer MySQL potrafi wykonywać zapytania w strukturalnym języku zapytań (ang. *Structured Query Language* — SQL). Dlatego też użycie języka SQL to sposób, w jaki serwerowi MySQL nakazujesz przeprowadzanie operacji zarządzania danymi, a biegłość w posługiwaniu się wymienionym językiem jest niezbędna do prowadzenia efektywnej komunikacji z serwerem. Kiedy używasz programu takiego jak klient `mysql`, jego podstawowym zadaniem jest umożliwienie Ci wysyłania serwerowi zapytań SQL do wykonania. Jeżeli tworzysz programy w językach, dla których MySQL posiada wbudowany interfejs, na przykład moduł Perl DBI lub rozszerzenie PHP PDO, wspomniane interfejsy pozwalają na komunikację z serwerem za pomocą wysyłanych mu zapytań SQL do wykonania.

W rozdziale 1., zatytułowanym „Rozpoczęcie pracy z MySQL”, zaprezentowano przewodnik stanowiący wprowadzenie do wielu możliwości oferowanych przez MySQL, między innymi poznałeś podstawowy sposób pracy z SQL. W tym rozdziale będziemy się opierać na materiale przedstawionym w poprzednim, ale znacznie dokładniej omówimy wiele tematów:

- zmiana trybu SQL w celu wpłynięcia na zachowanie serwera;
- odwoływanie się do elementów baz danych;
- używanie wielu kodowań znaków;
- tworzenie i usuwanie baz danych, tabel oraz indeksów;
- pobieranie informacji o bazach danych i ich zawartości;
- pobieranie danych za pomocą złączeń, podzapytań i unii;
- uaktualnianie i usuwanie danych w wielu tabelach;
- przeprowadzanie transakcji gwarantujących wykonanie wszystkich wskazanych zapytań lub żadnego;
- konfiguracja relacji klucza zewnętrznego;
- użycie silnika wyszukiwania pełnego tekstu.

Każde wymienione powyżej zagadnienie obejmuje szeroką gamę zadań, które można wykonać przy użyciu SQL. W innych rozdziałach znajdziesz dodatkowe informacje dotyczące SQL:

- Rozdział 4., „Widoki i programy składowane”. W rozdziale poruszono temat tworzenia i używania widoków (tabele wirtualne, które zapewniają alternatywny sposób przglądania się danym) oraz omówiono programy składowane (funkcje i procedury, wyzwalacze oraz zdarzenia).
- Rozdział 12., „Ogólna administracja bazą danych MySQL”. W tym rozdziale dowiesz się, jak używać zapytań administracyjnych takich jak GRANT i REVOKE w celu zarządzania kontami użytkowników. Ponadto poruszono temat systemu uprawnień nadzorującego operacje, jakie mogą być wykonywane przez poszczególnych użytkowników.
- Dodatek E, „Przewodnik po składni SQL”. W tym dodatku przedstawiono składnię zapytań SQL zaimplementowanych w bazie danych MySQL oraz uprawnienia wymagane do ich używania. Ponadto, omówiono składnię komentarzy w zapytaniach SQL.

Zapoznaj się również z podręcznikiem użytkownika MySQL, zwłaszcza ze zmianami wprowadzonymi w ostatnich wersjach serwera.

2.1. Tryby SQL serwera

Tryb SQL serwera MySQL wpływa na wiele aspektów wykonywania zapytań SQL. Serwer zawiera zmienną systemową o nazwie `sql_mode`, za pomocą której można ustawić odpowiedni tryb. Wymieniona zmienna może być zdefiniowana globalnie i wpływać na wszystkie klienty, a poszczególne klienty mają możliwość zmiany trybu, a tym samym wpływają na tryb pracy danej sesji (połączenia) z serwerem. Oznacza to, że każdy klient może zmienić sposób pracy z serwerem, nie wpływając przy tym na pozostałe klienty.

Tryb SQL ma wpływ na zachowanie, na przykład sposób obsługi nieprawidłowych wartości podczas wstawiania danych i identyfikacji cytowania. Poniższa lista przedstawia kilka z dostępnych wartości, które można przypisać zmiennej `sql_mode`:

- `STRICT_ALL_TABLES` i `STRICT_TRANS_TABLES` — włączenie trybu „ścisłego”. W tym trybie serwer zachowuje się znacznie bardziej restrykcyjnie w zakresie akceptacji nieprawidłowych wartości danych. (W szczególności oznacza to odrzucanie nieprawidłowych wartości zamiast ich zmiany na najbliższą poprawną wartość).
- `TRADITIONAL` — to jest tryb mieszany. Nieco podobny do trybu ścisłego, ale dopuszcza inne tryby nakładające dodatkowe ograniczenia w celu jeszcze ściślejszego sprawdzania danych. Po włączeniu trybu `TRADITIONAL` pod względem obsługi nieprawidłowych wartości danych działanie serwera przypomina tradycyjne serwery SQL.

- **ANSI_QUOTES** — ten tryb nakazuje serwerowi uznanie cudzysłowu za identyfikator znaku cytowania.
- **PIPES_AS_CONCAT** — ten tryb powoduje, że `||` będzie uznane za standardowy operator łączenia ciągu tekstowego w SQL, a nie za synonim operatora `OR`.
- **ANSI** — to inny tryb mieszany. Powoduje włączenie **ANSI_QUOTES**, **PIPES_AS_CONCAT** i kilku innych wartości trybów, które powodują, że serwer jest nieco bardziej zgodny ze standardem SQL.

W celu ustawienia trybu SQL podaj wartość składającą się z jednej lub więcej rozdzielonych przecinkami wartości trybów. Przypisanie pustego ciągu tekstowego powoduje wyzerowanie trybu. W przypadku wartości trybów wielkość liter nie ma znaczenia.

Aby ustawić tryb SQL podczas uruchamiania serwera, zmienną systemową `sql_mode` należy podać w wierszu poleceń `mysql` lub w pliku opcji. W wierszu poleceń używana jest następująca składnia:

```
--sql_mode="TRADITIONAL"
--sql_mode="ANSI_QUOTES,PIPES_AS_CONCAT"
```

Istnieje również możliwość zmiany trybu SQL w trakcie działania serwera; wówczas zmienną systemową `sql_mode` trzeba zmienić przy użyciu zapytania `SET`. Każdy klient może ustawić tryb SQL dla swojej sesji:

```
SET sql_mode = 'TRADITIONAL';
```

Globalne ustawienie trybu SQL wymaga użycia słowa kluczowego `GLOBAL`:

```
SET GLOBAL sql_mode = 'TRADITIONAL';
```

Globalne ustawienie zmiennej wymaga uprawnienia administracyjnego `SUPER`. Wartość globalna staje się domyślnym trybem SQL dla klientów, które nawiążą połączenie już po globalnym ustawieniu zmiennej.

W celu określenia bieżącej wartości trybu SQL dla sesji lub globalnego należy użyć poniższych zapytań:

```
SELECT @@SESSION.sql_mode;
SELECT @@GLOBAL.sql_mode;
```

Wartość zwrótana składa się z rozdzielonej przecinkami listy włączonych trybów lub pustej wartości, jeśli nie został włączony żaden tryb.

W podrozdziale 3.3, zatytułowanym „Jak MySQL obsługuje nieprawidłowe wartości danych?”, omówiono wartości trybów SQL, które wpływają na obsługę niepoprawnych lub brakujących wartości podczas wstawiania danych. W dodatku D, zatytułowanym „Przewodnik po zmiennych systemowych, stanu i użytkownika”, przedstawiono wszystkie dozwolone wartości zmiennej `sql_mode`. Więcej informacji na temat używania zmiennych systemowych znajdziesz w punkcie 12.3.1, zatytułowanym „Sprawdzanie i ustawienie wartości zmiennych systemowych”.

2.2. Identyfikatory składni MySQL i reguły nadawania nazw

Praktycznie każde zapytanie SQL używa w pewien sposób identyfikatorów, aby odwołać się do bazy danych lub jej elementów, takich jak tabele, widoki, kolumny, indeksy, procedury składowane, wyzwalacze lub zdarzenia. Kiedy odwołujesz się do elementów bazy danych, identyfikatory muszą spełniać przedstawione poniżej reguły.

Dozwolone znaki w identyfikatorach. Identyfikatory nieujęte w cudzysłów mogą składać się z liter łacińskich a-z o dowolnej wielkości, cyfr 0-9, znaków dolara i podkreślenia, a także znaków Unicode z zakresu od U+0080 do U+FFFF. Identyfikator może zaczynać się dowolnym dozwolonym znakiem, w tym także cyfrą. Jednak identyfikator nieujęty w cudzysłów nie może składać się wyłącznie z cyfr, ponieważ w ten sposób nie można go odróżnić od liczby. MySQL nieco inaczej niż pozostałe systemy baz danych obsługuje identyfikatory rozpoczynające się od cyfry. Jeżeli używasz tego typu identyfikatora, musisz zwrócić szczególną uwagę na to, czy rozpoczyna się on od litery E lub e, ponieważ może to doprowadzić do powstania niejednoznacznego wyrażenia. Na przykład, wyrażenie $23e + 14$ (wraz ze spacjami wokół znaku plus) oznacza kolumnę 23e plus liczbę 14. Co oznacza więc $23e+14$? Czy ma takie samo znaczenie, czy raczej oznacza liczbę w zapisie naukowym?

Identyfikatory mogą być cytowane (ograniczane) odwróconym apostrofem (`), co pozwala na użycie dowolnego znaku poza bajtem NULL i znakami Unicode z zakresu od U+10000:

```
CREATE TABLE `moja tabela` (`moja-kolumna-int` INT);
```

Cytowanie okazuje się użyteczne, gdy identyfikator jest słowem zarezerwowanym SQL lub zawiera spacje bądź inne znaki specjalne. Cytowanie pozwala również na stosowanie identyfikatorów wyłącznie liczbowych, co jest niemożliwe w przypadku identyfikatorów niecytowanych. Aby w cytowanym identyfikatorze użyć znaku cytowania, podaj go dwukrotnie.

Twój system operacyjny może nakładać dodatkowe ograniczenia na identyfikatory baz danych i tabel. Zapoznaj się z punktem 11.2.6, zatytułowanym „Ograniczenia systemu operacyjnego w zakresie nazw obiektów bazy danych”.

Alias kolumn i nazwy tabel mogą mieć dowolne nazwy. Alias powinien być ujęty w znaki cytowania identyfikatora tylko wtedy, gdy jest słowem zarezerwowanym SQL, składa się wyłącznie z cyfr, zawiera spacje lub inne znaki specjalne. Alias kolumn można cytować za pomocą cudzysłowu lub apostrofów.

Tryb SQL serwera. Po włączeniu trybu SQL o nazwie ANSI_QUOTES identyfikatory można cytować za pomocą cudzysłowu, choć nadal dozwolone jest stosowanie odwrotnych apostrofów.

```
CREATE TABLE "moja tabela" ("moja-kolumna-int" INT);
```

Włączenie trybu SQL o nazwie ANSI_QUOTES ma dodatkowy efekt: dosłowne ciągi tekstowe *muszą* być cytowane za pomocą apostrofów. Jeżeli użyjesz cudzysłowu, wówczas serwer zinterpretuje cytowaną wartość jako identyfikator, a nie ciąg tekstowy.

Nazwy funkcji wbudowanych normalnie nie są zarezerwowane i bez cytowania mogą być używane jako identyfikatory. Jednak po włączeniu trybu SQL o nazwie `IGNORE_SPACE` nazwy funkcji stają się zarezerwowane i muszą być cytowane, jeśli są używane jako identyfikatory.

Informacje na temat ustawiania trybu SQL znajdziesz w podrozdziale 2.1, zatytułowanym „Tryby SQL serwera”.

Długość identyfikatora. Większość identyfikatorów może mieć maksymalną długość 64 znaków. Z kolei maksymalna długość aliasu wynosi 256 znaków.

Kwalifikatory identyfikatora. W zależności od kontekstu, może wystąpić potrzeba stosowania w pełni kwalifikowanego identyfikatora, aby jasno wskazać element, do którego się on odwołuje. W celu odniesienia się do bazy danych należy podać jedynie jej nazwę:

```
USE nazwa_bazy_danych;  
SHOW TABLES FROM nazwa_bazy_danych;
```

Aby odwołać się do tabeli, masz dwie możliwości:

- Użycie w pełni kwalifikowanej nazwy tabeli składającej się z identyfikatorów bazy danych i tabeli:

```
SHOW COLUMNS FROM nazwa_bazy_danych.nazwa_tabeli;  
SELECT * FROM nazwa_bazy_danych.nazwa_tabeli;
```

- Użycie identyfikatora tabeli odwołującego się do tabeli w domyślnej (bieżącej) bazie danych. Jeżeli `sampdb` jest domyślną bazą danych, wtedy poniższe zapytania działają identycznie:

```
SELECT * FROM member;  
SELECT * FROM sampdb.member;
```

Jeżeli nie wybrano żadnej bazy danych, odwołanie się do tabeli bez podania identyfikatora bazy danych jest błędem, ponieważ serwer nie wie, do której bazy danych należy wskazana tabela.

Te same uwagi dotyczące nazw tabel mają zastosowanie względem nazw widoków (będących tabelami „wirtualnymi”) i programów składowanych.

Aby odwołać się do kolumny tabeli, masz trzy możliwości:

- Użycie w pełni kwalifikowanej nazwy w postaci `nazwa_bazy_danych.nazwa_tabeli.nazwa_kolumny`.
- Użycie częściowo kwalifikowanej nazwy w postaci `nazwa_tabeli.nazwa_kolumny`, która odwołuje się do kolumny we wskazanej tabeli w bieżącej bazie danych.
- Użycie niekwalifikowanej nazwy, na przykład `nazwa_kolumny`, odwołującej się do kolumny wskazanej w kontekście zapytania. Dwa poniższe zapytania używają tej samej nazwy kolumny, ale kontekst wskazany w klauzuli `FROM` zapytania wskazuje tabelę, w której znajduje się wymieniona kolumna:

```
SELECT last_name, first_name FROM president;  
SELECT last_name, first_name FROM member;
```

Zwykle nie ma konieczności stosowania w pełni kwalifikowanych nazw, choć zawsze można to zrobić. Jeżeli baza danych została wybrana za pomocą zapytania USE, wtedy staje się domyślną bazą danych dla kolejnych zapytań i jest stosowana w przypadku niekwalifikowanych odwołań do tabel. Gdy utworzysz zapytanie SELECT odwołujące się tylko do jednej tabeli, dana tabela będzie stosowana we wszystkich odwołaniach do kolumn w tym zapytaniu. Używanie kwalifikowanych identyfikatorów jest wymagane, gdy nazwa tabeli lub bazy danych nie może być określona na podstawie kontekstu. Na przykład, jeśli zapytanie odwołuje się do tabel z wielu baz danych, odwołania do tabel w bazach danych innych niż domyślna muszą mieć składnię *nazwa_bazy_danych.nazwa_tabeli*, aby serwer MySQL wiedział, gdzie znajduje się wskazana tabela. Podobnie, jeśli zapytanie używa wielu tabel i odwołuje się do kolumny znajdującej się w więcej niż tylko jednej tabeli, wtedy kwalifikowany identyfikator jasno wskazuje kolumnę, która powinna być użyta.

W przypadku użycia cytowania podczas odwoływania się do kwalifikowanej nazwy, poszczególne identyfikatory powinny być cytowane oddzielnie. Na przykład:

```
SELECT * FROM `sampdb`.`member` WHERE `sampdb`.`member`.`member_id` > 100;
```

Nie cytuj nazwy jako całości. Poniższe zapytanie jest nieprawidłowe:

```
SELECT * FROM `sampdb.member` WHERE `sampdb.member.member_id` > 100;
```

Wymóg cytowania słowa zarezerwowanego używanego w charakterze identyfikatora ustępuje, jeśli słowo znajduje się po kropce kwalifikatora, ponieważ wtedy kontekst wskazuje, czy słowo zarezerwowane jest identyfikatorem.

2.3. Wielkość liter w zapytaniach SQL

Reguły dotyczące wielkości liter w zapytaniach SQL są odmienne dla różnych elementów zapytania, a ponadto zależą od elementu docelowego i systemu operacyjnego komputera, w którym został uruchomiony serwer.

Słowa kluczowe SQL i nazwy funkcji. W słowach kluczowych i nazwach funkcji wielkość liter nie ma znaczenia, mogą być stosowane dowolne. Przedstawione poniżej zapytania mają identyczne działanie:

```
SELECT NOW();
select now();
sELeCt nOw();
```

Nazwy baz danych, tabel i widoków. MySQL przedstawia bazy danych i tabele za pomocą katalogów i plików w systemie plików komputera, w którym uruchomiono serwer. Dlatego też wielkość liter dla nazw baz danych i tabel zależy od sposobu obsługi nazw plików przez system operacyjny. W przypadku Windows wielkość liter nie ma znaczenia, a tym samym serwer MySQL uruchomiony w Windows nie rozróżnia wielkości liter w nazwach baz danych i tabel. Z kolei serwer uruchomiony w systemie UNIX zwykle rozróżnia wielkość liter w nazwach baz danych i tabel, ponieważ wielkość liter w nazwach plików ma znaczenie. Wyjątkiem są tutaj nazwy w systemie plików Mac OS X Extended, który może nie rozróżniać wielkości liter.

Poszczególne widoki w MySQL przedstawiane są w postaci plików, więc powyższe reguły dotyczące tabel mają zastosowanie również w nazwach widoków.

Nazwy programów składowanych. Nazwy funkcji i procedur składowanych, a także zdarzeń nie rozróżniają wielkości liter. Natomiast wielkość liter ma znaczenie w nazwach wyzwalaczy, co jest odstępstwem od standardu SQL.

Nazwy kolumn i indeksów. W bazie danych MySQL wielkość liter w nazwach kolumn i indeksów nie ma znaczenia. Dlatego też działanie poniższych zapytań jest identyczne:

```
SELECT name FROM student;  
SELECT NAME FROM student;  
SELECT nAmE FROM student;
```

Nazwy aliasów. Domyślnie, aliasy tabel rozróżniają wielkość liter. Nazwę aliasu możesz podać za pomocą liter o dowolnej wielkości (małe, wielkie, mieszane), ale jeśli dany alias stosujesz wielokrotnie w zapytaniu, za każdym razem musisz użyć liter o dokładnie takiej samej wielkości jak w definicji aliasu. Gdy wartość zmiennej systemowej `lower_case_table_names` jest inna niż zero, aliasy tabel nie rozróżniają wielkości liter.

Wartości w ciągu tekstowym. Rozróżnianie wielkości liter w ciągu tekstowym zależy od jego typu (binarny lub niebinarny), a w niebinarnym także od kodowania znaków. To dotyczy dosłownych ciągów tekstowych oraz zawartości kolumn tekstowych. Więcej informacji znajdziesz w punkcie 3.1.2, zatytułowanym „Wartości ciągu tekstowego”.

Kwestie dotyczące wielkości liter powinieneś rozważyć podczas tworzenia baz danych i tabel w komputerze, w którym wielkość liter ma znaczenie w nazwach plików, ponieważ pewnego dnia może wystąpić potrzeba ich przeniesienia do systemu nierozróżniającego wielkości liter. Przyjmijmy założenie, że w serwerze UNIX stworzysz dwie tabele, o nazwach `abc` i `ABC`. Wymienione nazwy będą uznawane za zupełnie oddzielne. Po ich przeniesieniu do systemu Windows zaczną się problemy: nazwy nie będą rozróżniane, ponieważ wielkość liter w nazwach plików w Windows nie ma znaczenia. Problemy wystąpią także podczas replikacji tabel z serwera głównego działającego pod kontrolą systemu UNIX w serwerze podległym, który działa pod kontrolą systemu Windows.

Aby uniknąć wspomnianych problemów, zdecyduj się na konkretną wielkość liter, a następnie zawsze nadawaj bazom danych i ich tabelom nazwy zapisane w wybranej wielkości liter. W takim przypadku przeniesienie bazy danych do innego serwera nie będzie wiązało się z problemem wielkości liter w nazwach. Osobiście zalecam stosowanie małych liter, zwłaszcza podczas używania tabel InnoDB, ponieważ wewnętrznie silnik InnoDB przechowuje nazwy baz danych i tabel zapisane małymi literami.

W celu wymuszenia tworzenia baz danych i tabel o nazwach zapisanych małymi literami, nawet jeśli nie będą tak podane w zapytaniach `SELECT`, należy odpowiednio skonfigurować serwer przez ustawienie zmiennej systemowej `lower_case_table_names`. Więcej informacji na ten temat znajdziesz w punkcie 11.2.6, zatytułowanym „Ograniczenia systemu operacyjnego w zakresie nazw obiektów bazy danych”.

Niezależnie od tego, czy wielkość liter w nazwach baz danych lub tabel ma znaczenie w używanym systemie operacyjnym, w danym zapytaniu musisz się do nich odwoływać za pomocą tej samej wielkości liter. To nie dotyczy słów kluczowych SQL, nazw funkcji,

kolumn i indeksów — w zapytaniu można się do nich odwoływać przy użyciu dowolnej wielkości liter.

2.4. Obsługa kodowania znaków

MySQL obsługuje wiele różnych kodowań znaków; można je stosować niezależnie na poziomie serwera, bazy danych, tabeli, kolumny lub ciągu tekstowego. Na przykład, jeżeli kolumny tabeli mają domyślnie stosować kodowanie `latin1`, ale chcesz dołączyć także kolumny z kodowaniem hebrajskim i greckim, możesz to zrobić. Oprócz tego istnieje możliwość wyraźnego zdefiniowania kolejności sortowania. Możesz także sprawdzić bieżące kodowanie i kolejność sortowania, a następnie przeprowadzić konwersję danych z jednego kodowania na inne.

W tym podrozdziale zostaną przedstawione ogólne informacje dotyczące obsługi kodowania znaków w bazie danych MySQL. Rozdział 3., zatytułowany „Typy danych”, zawiera dokładniejszą analizę poświęconą kodowaniu znaków, kolejności sortowania, binarnym i niebinarnym ciągom tekstowym oraz definiowaniu tabel opartych na znakach i pracy z nimi.

MySQL oferuje następujące funkcje dotyczące kodowania znaków:

- Serwer zapewnia jednoczesną obsługę wielu kodowań znaków.
- W danym kodowaniu znaków może być ustawiona jedna lub więcej kolejności sortowania. Masz możliwość wyboru kolejności sortowania najbardziej odpowiadającej danej aplikacji.
- Obsługa Unicode odbywa się przez kodowania znaków `utf8` i `ucs2` zawierające znaki BMP (ang. *Basic Multilingual Plane*), a także `utf16`, `utf32` i `utf8mb4` zawierające znaki MBP i dodatkowe. W MySQL 5.6.1 mamy jeszcze `utf16le`, które jest jak `utf16`, ale stosuje kodowanie little-endian zamiast big-endian.
- Istnieje możliwość zdefiniowania kodowania znaków na poziomie serwera, bazy danych, tabeli, kolumny lub ciągu tekstowego:
 - ◆ Serwer ma domyślnie określone kodowanie znaków.
 - ◆ W zapytaniu `CREATE DATABASE` można przypisać bazie danych kodowanie znaków, natomiast w zapytaniu `ALTER DATABASE` można je zmienić.
 - ◆ Zapytania `CREATE TABLE` i `ALTER TABLE` mają klauzule pozwalające na określenie kodowania znaków na poziomie tabeli i kolumny.
 - ◆ Kodowanie znaków w ciągach tekstowych jest określone przez kontekst lub może być wyraźnie wskazane.
- Kilka funkcji i operatorów zostało przeznaczonych do konwersji poszczególnych wartości z jednego kodowania znaków na inne, a wartością zwrótną funkcji `CHARSET()` jest kodowanie znaków stosowane we wskazanej wartości. Podobnie, operator `COLLATE` można wykorzystać do zmiany kolejności sortowania ciągu tekstowego, natomiast funkcja `COLLATION()` informuje o kolejności sortowania we wskazanym ciągu tekstowym.

- Zapytania SHOW i tabele w bazie danych INFORMATION_SCHEMA zawierają informacje o dostępnych kodowaniach znaków i kolejności sortowania.
- Serwer automatycznie zmienia kolejność indeksów po zmianie kolejności sortowania zindeksowanej kolumny znakowej.

W ramach pojedynczego ciągu tekstowego nie można łączyć kodowań znaków lub używać różnych kodowań znaków dla poszczególnych rekordów danej kolumny. Jednak obsługę wielu języków można zapewnić za pomocą kodowania znaków Unicode (w ramach pojedynczego kodowania przedstawia znaki używane w wielu językach).

2.4.1. Określenie kodowania znaków

Określenie kodowania znaków i kolejności sortowania można przeprowadzić na wielu poziomach, począwszy od domyślnego stosowanego przez serwer aż po używane w poszczególnych ciągach tekstowych.

Domyślne kodowanie znaków i kolejność sortowania w serwerze są definiowane w trakcie kompilacji MySQL. Te ustawienia można zmienić w trakcie uruchamiania lub działania serwera za pomocą zmiennych systemowych `character_set_server` i `collation_server`, jak to zostało przedstawione w punkcie 12.6.2, zatytułowanym „Ustawienie domyślnego kodowania znaków i kolejności sortowania”. Jeżeli wskażesz jedynie kodowanie znaków, wbudowana w serwer kolejność sortowania stanie się wartością domyślną. W przypadku podania kolejności sortowania musi być ona zgodna ze wskazanym kodowaniem znaków. Kolejność sortowania będzie zgodna z kodowaniem znaków, gdy jego nazwa rozpoczyna się od nazwy kodowania znaków. Na przykład, kolejność sortowania `utf8_danish_ci` jest zgodna z `utf8`, ale nie z `latin1`.

W zapytaniach SQL przeznaczonych do tworzenia baz danych i tabel dostępne są dwie klauzule pozwalające na podanie kodowania znaków i kolejności sortowania w bazie danych oraz tabeli:

```
CHARACTER SET kodowanie_znaków
COLLATE kolejność_sortowania
```

Klauzula `CHARSET` może być używana jako synonim `CHARSET SET`. Podane *kodowanie_znaków* to nazwa kodowania obsługiwanego przez serwer, natomiast *kolejność_sortowania* to nazwa jednej z kolejności sortowania obsługiwanego przez wybrane kodowanie znaków. Wymienionych klauzul można używać razem lub oddzielnie. W przypadku podania obu kolejność sortowania musi być zgodna z kodowaniem znaków. Gdy podasz tylko kodowanie znaków, użyta zostanie domyślna kolejność sortowania. Z kolei po podaniu kolejności sortowania ustawione zostanie kodowanie znaków, którego nazwa jest pierwszą częścią w nazwie kolejności sortowania. Wymienione reguły mają zastosowanie na kilku poziomach:

- Aby w chwili tworzenia bazy danych wskazać domyślne kodowanie znaków i kolejność sortowania, należy użyć poniższego zapytania:

```
CREATE DATABASE nazwa_bazy_danych CHARACTER SET kodowanie_znaków COLLATE kolejność_sortowania;
```

W przypadku braku kodowania znaków lub kolejności sortowania baza danych użyje wartości domyślnych serwera.

- Aby w chwili tworzenia tabeli wskazać domyślne kodowanie znaków i kolejność sortowania, należy użyć klauzul `CHARACTER SET` i `COLLATE`, jak pokazano w poniższym zapytaniu:

```
CREATE TABLE nazwa_tabeli (...) CHARACTER SET kodowanie_znaków COLLATE
kolejność_sortowania;
```

W przypadku braku kodowania znaków lub kolejności sortowania tabela użyje wartości domyślnych bazy danych.

- Kolumnom tabeli można wyraźnie przypisać kodowanie znaków i kolejność sortowania za pomocą atrybutów `CHARACTER SET` i `COLLATE`, na przykład:

```
c CHAR(10) CHARACTER SET kodowanie_znaków COLLATE kolejność_sortowania
```

W przypadku braku kodowania znaków lub kolejności sortowania kolumna użyje wartości domyślnych tabeli. Wymienione atrybuty można stosować dla typów danych `CHAR`, `VARCHAR`, `TEXT`, `ENUM` i `SET`.

Istnieje również możliwość sortowania ciągów tekstowych według określonej kolejności wskazanej przez operator `COLLATE`. Na przykład, jeśli `c` to kolumna `latin1` o kolejności sortowania `latin1_swedish_ci`, a Ty chcesz zastosować kolejność sortowania z użyciem reguł języka hiszpańskiego, możesz wykonać poniższe zapytanie:

```
SELECT c FROM t ORDER BY c COLLATE latin1_spanish_ci;
```

2.4.2. Określenie dostępności kodowania znaków i ustawień bieżących

Aby poznać dostępne kodowania znaków i kolejności sortowania, wykonaj wymienione poniżej zapytania:

```
SHOW CHARACTER SET;
SHOW COLLATION;
```

Każde z wymienionych zapytań obsługuje klauzulę `LIKE`, ograniczającą wyniki do tych kodowań znaków lub kolejności sortowania, których nazwy zostały dopasowane do wzorca. Na przykład, poniższe zapytanie wyświetla łacińskie kodowania znaków i kolejności sortowania dostępne w kodowaniu Unicode (`utf8`):

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+-----+-----+-----+-----+
| Charset | Description | Default collation | Maxlen |
+-----+-----+-----+-----+
| latin1  | cp1252 West European | latin1_swedish_ci | 1 |
| latin2  | ISO 8859-2 Central European | latin2_general_ci | 1 |
| latin5  | ISO 8859-9 Turkish | latin5_turkish_ci | 1 |
| latin7  | ISO 8859-13 Baltic | latin7_general_ci | 1 |
+-----+-----+-----+-----+
mysql> SHOW COLLATION LIKE 'utf8%';
```

Collation	Charset	Id	Default	Compiled	Sortlen
utf8_general_ci	utf8	33	Yes	Yes	1
utf8_bin	utf8	83		Yes	1
utf8_unicode_ci	utf8	192		Yes	8
utf8_icelandic_ci	utf8	193		Yes	8
utf8_latvian_ci	utf8	194		Yes	8
utf8_romanian_ci	utf8	195		Yes	8
utf8_slovenian_ci	utf8	196		Yes	8
...					

Nazwy kolejności sortowania zawsze rozpoczynają się od nazwy kodowania znaków. Każde kodowanie znaków ma przynajmniej jedną kolejność sortowania, a jedna z nich zawsze jest domyślna.

Informacje dotyczące dostępnych kodowań znaków lub kolejności sortowania mogą być pobrane również z tabel CHARACTER_SETS i COLLATIONS znajdujących się w bazie danych INFORMATION_SCHEMA (patrz podrozdział 2.7, zatytułowany „Pobieranie metadanych bazy danych”).

Aby wyświetlić bieżące ustawienia serwera w zakresie kodowania znaków i kolejności sortowania, wykonuj zapytania SHOW VARIABLES:

```
mysql> SHOW VARIABLES LIKE 'character\_set\_%';
```

Variable_name	Value
character_set_client	utf8
character_set_connection	utf8
character_set_database	latin1
character_set_filesystem	binary
character_set_results	utf8
character_set_server	latin1
character_set_system	utf8

```
mysql> SHOW VARIABLES LIKE 'collation\_%';
```

Variable_name	Value
collation_connection	utf8_general_ci
collation_database	latin1_swedish_ci
collation_server	latin1_swedish_ci

Wiele z wymienionych powyżej zmiennych systemowych wpływa na sposób komunikacji klienta z serwerem po nawiązaniu połączenia. Więcej szczegółowych informacji na ten temat znajdziesz w podpunkcie 3.1.2.2, zatytułowanym „Zmienne systemowe dotyczące kodowania znaków”.

2.4.3. Obsługa Unicode

Jednym z powodów istnienia tak dużej liczby kodowań znaków jest fakt, że dla poszczególnych języków opracowano różne kodowania znaków. To wiąże się z wieloma problemami. Na przykład, dany znak, powszechnie stosowany w wielu językach, może mieć inne wartości

numeryczne w różnych kodowaniach. Ponadto, poszczególne języki mogą wymagać innej liczby bajtów do wyrażenia znaków. Kodowanie `latin1` jest na tyle małe, że każdy znak mieści się w pojedynczym bajcie. Jednak języki używane na przykład w Japonii lub Chinach zawierają tak dużo znaków, że wymagają wielu bajtów do wyrażenia znaku.

Kodowanie Unicode stanowi rozwiązanie wspomnianych problemów i oferuje zunifikowany system kodowania znaków zawierający kodowania dla wszystkich języków, które można przedstawić w spójny sposób.

Kodowania znaków Unicode `utf8` i `ucs2` zawierają jedynie znaki w BMP, co ogranicza ich liczbę do 65 536. Nie są obsługiwane znaki dodatkowe poza BMP.

- Kodowanie znaków `ucs2` odpowiada kodowaniu Unicode UCS-2. Każdy znak jest przedstawiany przy użyciu dwóch bajtów; jako pierwszy jest najbardziej znaczący bajt (ang. *Most Significant Byte* — MSB). UCS to skrót oznaczający uniwersalny zestaw znaków (ang. *Universal Character Set*).
- Kodowanie znaków `utf8` stosuje format o zmiennej długości i do przedstawienia znaku wykorzystuje od jednego do trzech bajtów. Odpowiada kodowaniu Unicode UTF-8. UTF to skrót oznaczający *Unicode Transformation Format*.

Począwszy od MySQL 5.5.3, dostępne są także inne kodowania znaków Unicode, zawierające oprócz BMP także znaki dodatkowe.

- Kodowania `utf16` i `utf32` są podobne do `ucs2`, ale zawierają także znaki dodatkowe. W przypadku `utf16` znaki BMP zabierają dwa bajty (podobnie jak w `ucs2`), natomiast znaki dodatkowe po cztery bajty. W kodowaniu `utf32` wszystkie znaki zabierają po cztery bajty.
- Kodowanie `utf8mb4` zawiera wszystkie znaki `utf8` (zabierające od jednego do trzech bajtów), natomiast znaki dodatkowe zabierają po cztery bajty.

W MySQL 5.6.1 mamy jeszcze `utf16le`, które jest jak `utf16`, ale stosuje kodowanie little-endian zamiast big-endian.

2.5. Wybór, utworzenie, usunięcie i zmiana bazy danych

MySQL zawiera wiele zapytań na poziomie bazy danych: `USE` do wyboru domyślnej bazy danych, `CREATE DATABASE` do tworzenia baz danych, `DROP DATABASE` do usuwania baz danych oraz `ALTER DATABASE` do modyfikacji globalnej charakterystyki bazy danych.

W kolejnych zapytaniach słowo kluczowe `SCHEMA` jest synonimem `DATABASE`.

2.5.1. Wybór bazy danych

Zapytanie `USE` powoduje wybór bazy danych i czyni ją domyślną (bieżącą) w danej sesji pracy z serwerem:

```
USE nazwa_bazy_danych;
```

Musisz mieć uprawnienia dostępu do bazy danych, w przeciwnym razie zostanie wyświetlony komunikat błędu.

Nie ma konieczności wyraźnego wskazywania bazy danych. Do tabel bazy danych można się odwoływać bez jej wcześniejszego wyboru, a przez stosowanie kwalifikowanych nazw wskazujących zarówno bazę danych, jak i tabelę. Na przykład, w celu pobrania zawartości tabeli `president` w bazie danych `sampdb` bez konieczności jej wcześniejszego wyboru możesz użyć zapytań takich jak poniższe:

```
SELECT * FROM sampdb.president;
```

Wybór bazy danych nie oznacza, że musi ona pozostać domyślna w ciągu całej sesji. Zawsze możesz wykonać zapytanie `USE` i przełączać się między bazami danych, gdy zajdzie konieczność. Ponadto, wybór bazy danych nie ogranicza Cię do używania jedynie znajdujących się w niej tabel. Kiedy jedna baza danych jest domyślna, do tabel znajdujących się w innych można się odwoływać za pomocą kwalifikowanych nazw wraz z odpowiednimi identyfikatorami baz danych.

Po zamknięciu połączenia z serwerem znikają wszelkie informacje o domyślnej bazie danych w trakcie sesji. Oznacza to, że jeżeli ponownie nawiążesz połączenie z serwerem, nie będzie on pamiętał, która baza danych była wybrana w trakcie poprzedniej sesji.

2.5.2. Utworzenie bazy danych

Do utworzenia bazy danych służy zapytanie `CREATE DATABASE`:

```
CREATE DATABASE nazwa_bazy_danych;
```

Baza danych o podanej nazwie nie może istnieć, a ponadto musisz mieć uprawnienia do jej utworzenia.

W zapytaniu `CREATE DATABASE` można użyć kilku opcjonalnych klauzul. Pełna składnia zapytania przedstawia się następująco:

```
CREATE DATABASE [IF NOT EXISTS] nazwa_bazy_danych  
[CHARACTER SET kodowanie_znaków] [COLLATE kolejność_sortowania];
```

Próba utworzenia bazy danych o istniejącej nazwie domyślnie spowoduje wygenerowanie błędu. Aby zmienić to zachowanie i spowodować utworzenie bazy danych tylko wtedy, gdy nie istnieje żadna baza o podanej nazwie, dodaj klauzulę `IF NOT EXISTS`:

```
CREATE DATABASE IF NOT EXISTS nazwa_bazy_danych;
```

Domyślnie, kodowanie znaków i kolejność sortowania są takie same jak domyślnie stosowane w serwerze. Aby wyraźnie zdefiniować kodowanie znaków i kolejność sortowania, użyj klauzul `CHARACTER SET` i `COLLATE`, na przykład:

```
CREATE DATABASE mydb CHARACTER SET utf8 COLLATE utf8_icelandic_ci;
```

Użycie klauzuli `CHARACTER SET` bez `COLLATE` spowoduje zastosowanie domyślnej kolejności sortowania w danym kodowaniu znaków. Z kolei w przypadku użycia klauzuli `COLLATE` bez `CHARACTER SET` pierwsza część nazwy kolejności sortowania będzie wskazywała kodowanie znaków.

Kodowanie znaków musi być jednym z obsługiwanych przez serwer, na przykład `latin1` lub `sjis`. Natomiast kolejność sortowania powinna być prawidłową i obsługiwaną w danym kodowaniu znaków. Więcej informacji na temat kodowania znaków i kolejności sortowania znajdziesz w podrozdziale 2.4, zatytułowanym „Obsługa kodowania znaków”.

Kiedy stworzysz bazę danych, MySQL tworzy w katalogu *data* podkatalog o takiej samej nazwie jak baza danych. Wspomniany katalog jest nazywany katalogiem bazy danych. Serwer tworzy również plik *db.opt* w katalogu bazy danych, służący do przechowywania jej atrybutów, takich jak kodowanie znaków i kolejność sortowania. Kiedy później utworzysz tabelę w bazie danych, wartości domyślne bazy danych staną się domyślne dla definicji tabeli, o ile wyraźnie nie wskażesz kodowania znaków i kolejności sortowania w tworzonej tabeli.

Aby wyświetlić definicję istniejącej bazy danych, wykonaj zapytanie `SHOW CREATE DATABASE:`

```
mysql> SHOW CREATE DATABASE mydb\G
***** 1. row *****
Database: mydb
Create Database: CREATE DATABASE `mydb`
/*140100 DEFAULT CHARACTER SET utf8
COLLATE utf8_icelandic_ci */
```

2.5.3. Usunięcie bazy danych

Usunięcie bazy danych jest równie łatwe jak jej utworzenie, przy założeniu, że masz uprawnienia do wykonania zapytania `DROP`:

```
DROP DATABASE nazwa_bazy_danych;
```

Zapytanie `DROP DATABASE` nie jest przeznaczone do lekkomyślnego używania. Powoduje usunięcie bazy danych wraz z jej całą zawartością (tabele, procedury składowane itd.), która jest tracona na zawsze, o ile nie posiadasz zrobionej kopii zapasowej.

Baza danych jest przedstawiana przez podkatalog w katalogu *data*. Wspomniany podkatalog jest przeznaczony do przechowywania obiektów takich jak tabele, widoki i wyzwalacze. Jeżeli wykonanie zapytania `DROP DATABASE` zakończy się niepowodzeniem, powodem najczęściej jest obecność w katalogu bazy danych plików niepowiązanych z obiektami bazy danych. Zapytanie `DROP DATABASE` nie usuwa tego rodzaju plików, a tym samym nie usuwa katalogu bazy danych. Oznacza to, że katalog bazy danych nadal będzie istniał i zostanie wyświetlony po wykonaniu zapytania `SHOW DATABASES`. Aby naprawdę usunąć bazę danych w takim przypadku, musisz ręcznie usunąć wszelkie pliki i podkatalogi z katalogu bazy danych, a następnie ponownie wykonać zapytania `DROP DATABASE`.

2.5.4. Zmiana bazy danych

Zapytanie `ALTER DATABASE` powoduje zmianę globalnych atrybutów bazy danych, o ile masz uprawnienia do jego wykonania. Obecnie jedynymi tego typu atrybutami są domyślne kodowanie znaków i kolejność sortowania.


```
ALTER DATABASE [nazwa_bazy_danych] [CHARACTER SET kodowanie_znaków] [COLLATE kolejność_sortowania];
```

W wcześniejszej części rozdziału, dotyczącej zapytania `CREATE DATABASE`, przedstawiono efekt użycia klauzul `CHARACTER SET` i `COLLATE`; przynajmniej jedna z nich musi być podana.

Jeżeli pominiesz nazwę bazy danych, zapytanie `ALTER DATABASE` zostanie użyte względem domyślnej bazy danych.

2.6. Tworzenie, usuwanie, indeksowanie i modyfikowanie tabel

MySQL pozwala na tworzenie tabel, usuwanie ich, a także zmianę struktury tabel za pomocą zapytań odpowiednio `CREATE TABLE`, `DROP TABLE` i `ALTER TABLE`. Zapytania `CREATE INDEX` i `DROP INDEX` pozwalają na dodawanie lub usuwanie indeksów z istniejących tabel. W tym podrozdziale znajdziesz informacje szczegółowe na temat wymienionych zapytań, ale najpierw zapoznamy się z silnikami stosowanymi przez MySQL do obsługi różnego typu tabel.

2.6.1. Cechy charakterystyczne silników bazy danych

MySQL obsługuje wiele różnych silników baz danych. Każdy z nich implementuje tabele o pewnym określonym zestawie właściwości i cech charakterystycznych. W tabeli 2.1 pokrótce wymieniono silniki baz danych, w dalszej części książki znajdziesz więcej informacji szczegółowych odnośnie niektórych z nich (przede wszystkim InnoDB i MyISAM). Pozostałe są rzadziej używane lub, jak ma to miejsce w przypadku NDB, wymagają omówienia wykraczającego poza to, które można przedstawić w tej książce. Dlatego też w pozostałej części książki znajdziesz niewiele informacji na ich temat.

Tabela 2.1. Silniki bazy danych dostępne w MySQL

Silnik bazy danych	Opis
ARCHIVE	Archiwalny silnik bazy danych (brak możliwości modyfikacji rekordów po ich wstawieniu).
BLACKHOLE	Silnik odrzucający operacje zapisu i zwracający puste odczyty.
CSV	Silnik bazy danych w formacie wartości rozdzielonych przecinkami.
FEDERATED	Silnik przeznaczony do uzyskania dostępu do zdalnych tabel.
InnoDB	Transakcyjny silnik bazy danych wraz z obsługą kluczy zewnętrznych.
MEMORY	Tabele w pamięci.
MERGE	Zarządza kolekcją tabel MyISAM.
MyISAM	Podstawowy silnik bazy danych, który nie obsługuje transakcji.
NDB	Silnik dla klastra MySQL.

Nazwy pewnych silników mają synonimy. MRG_MyISAM i NDBCLUSTER to synonimy silników odpowiednio MERGE i NDB. Z kolei silniki MEMORY i InnoDB początkowo były znane jako odpowiednio HEAP i Innobase. Wcześniej wymienione nazwy nadal są rozpoznawane, choć uznawane za przestarzałe.

Początkowo serwer MySQL był tworzony w taki sposób, że zawierał wbudowane wszystkie wymienione silniki. Obecnie serwer stosuje architekturę opartą na „wtyczkach”, pozwalającą na selektywne wczytywanie wtyczek, a wiele silników bazy danych jest dostępnych w postaci wtyczek. W ten sposób administrator może traktować je jako opcjonalne i wczytywać jedynie niezbędne. Architektura oparta na wtyczkach pozwala również na integrację z serwerem silników opracowanych przez firmy trzecie. Więcej informacji na ten temat znajdziesz w podrozdziale 12.4, zatytułowanym „Interfejs wtyczek”.

2.6.1.1. Ustalenie dostępnych silników bazy danych

Dostępność silników baz danych dla danego serwera tak naprawdę zależy od konkretnej wersji MySQL, sposobu konfiguracji serwera podczas jego kompilacji oraz od używanych opcji startowych. Więcej informacji dotyczących wyboru silników baz danych znajdziesz w podrozdziale 12.5, zatytułowanym „Konfiguracja silnika bazy danych”.

Aby przekonać się, które silniki baz danych są dostępne w serwerze, wykonaj zapytanie `SHOW ENGINES`:

```
mysql> SHOW ENGINES\G
***** 1. row *****
      Engine: InnoDB
      Support: DEFAULT
      Comment: Supports transactions, row-level locking, and foreign keys
      Transactions: YES
        XA: YES
      Savepoints: YES
...
***** 8. row *****
      Engine: MyISAM
      Support: YES
      Comment: MyISAM storage engine
      Transactions: NO
        XA: NO
      Savepoints: NO
...
```

Wartością kolumny `Support` może być `YES` lub `NO` i oznacza dostępność lub niedostępność danego silnika, wartość `DISABLED` oznacza silnik dostępny, choć wyłączony, natomiast `DEFAULT` wskazuje silnik domyślnie używany przez serwer. Silnik oznaczony jako `DEFAULT` powinien być uznany za dostępny. W kolumnie `Transactions` znajduje się informacja o tym, czy dany silnik bazy danych obsługuje transakcje. Z kolei kolumny `XA` i `Savepoints` informują o obsłudze transakcji rozproszonych (nie będą omawiane w tej książce) i częściowym wycofaniu transakcji.

Tabela `ENGINES` w bazie danych `INFORMATION_SCHEMA` zawiera te same informacje, które są wyświetlane przez zapytanie `SHOW ENGINES`, ale dzięki użyciu zapytania `SELECT` możesz utworzyć zapytania zwracające jedynie interesujące Cię informacje. Na przykład,

poniższe zapytanie wykorzystuje tabelę `ENGINES` w celu sprawdzenia dostępnych silników zapewniających obsługę transakcji:

```
mysql> SELECT ENGINE FROM INFORMATION_SCHEMA.ENGINES
-> WHERE TRANSACTIONS = 'YES';
+-----+
| ENGINE |
+-----+
| InnoDB |
+-----+
```

2.6.1.2. Reprezentacja tabeli na dysku

Za każdym razem, gdy stworzysz tabelę, MySQL tworzy na dysku plik zawierający format tabeli (to znaczy jej definicję). Format pliku ma nazwę bazową taką samą jak nazwa tabeli oraz rozszerzenie *.frm*. Dla tabeli o nazwie *t* plik formatu nosi nazwę *t.frm*. Serwer tworzy plik w katalogu bazy danych, do której należy dana tabela. Plik *.frm* jest niezmienny, ponieważ jest jeden dla wszystkich tabel, niezależnie od silnika bazy danych obsługującego daną tabelę. Nazwa tabeli używana w zapytaniach SQL może się różnić od nazwy powiązanego z nią pliku *.frm*, jeśli zawiera znaki sprawiające problemy w nazwach plików. Omówienie reguł mapowania nazw SQL na nazwy plików znajdziesz w punkcie 11.2.6, zatytułowanym „Ograniczenia systemu operacyjnego w zakresie nazw obiektów bazy danych”.

Poszczególne silniki baz danych mogą również tworzyć inne pliki unikalne dla tabeli i przeznaczone do przechowywania jej zawartości. W przypadku danej tabeli wszystkie powiązane z nią pliki są umieszczane w katalogu bazy danych, do której należy tabela. W tabeli 2.2 wymieniono rozszerzenia nazw plików tworzonych dla tabel przez niektóre silniki baz danych.

Tabela 2.2. Pliki tabel tworzone przez niektóre silniki baz danych

Silnik bazy danych	Pliki na dysku
InnoDB	<i>.ibd</i> (dane i indeksy)
MyISAM	<i>.myd</i> (dane), <i>.myi</i> (indeksy)
CSV	<i>.csv</i> (dane), <i>.csm</i> (metadane)

W przypadku niektórych silników baz danych plik formatu to jedyny plik powiązany z daną tabelą. Z kolei inne silniki baz danych mogą przechowywać zawartość tabeli w innym miejscu na dysku lub też wykorzystywać jedną lub więcej przestrzeni tabel (obszary pamięci masowej współdzielone przez wiele tabel):

- Silnik MEMORY zawartość tabeli przechowuje w pamięci, a nie na dysku.
- Domyślnie, silnik InnoDB przechowuje dane tabeli i indeksy w systemowej przestrzeni tabel. Oznacza to, że cała zawartość tabeli InnoDB jest zarządzana w ramach współdzielonego obszaru pamięci masowej, a nie w plikach powiązanych z poszczególnymi tabelami. Alternatywnie, silnik InnoDB tworzy pliki *.ibd*, jeśli skonfigurujesz go do używania przestrzeni tabel dla poszczególnych tabel.

W poniższych punktach pokrótce przedstawiono cechy charakterystyczne i zachowanie wybranych silników w MySQL. Więcej informacji na temat fizycznego przedstawiania tabel przez poszczególne silniki znajdziesz w punkcie 11.2.3, zatytułowanym „Przedstawienie tabel w systemie plików”.

2.6.1.3. Silnik InnoDB

Silnik InnoDB jest domyślnym silnikiem bazy danych w MySQL, o ile w inny sposób nie skonfigurujesz serwera. Poniższa lista wymienia niektóre z zalet silnika InnoDB.

- Tabele zapewniające obsługę transakcji (zatwierdzanie i wycofywanie). Punkty pośrednie można tworzyć, aby umożliwić częściowe wycofanie transakcji.
- Automatyczna naprawa po awarii.
- Obsługa kluczy zewnętrznych i spójności odniesień (ang. *Referential Integrity*), włączając kaskadowe operacje usuwania i uaktualniania.
- Blokowanie na poziomie rekordów i tworzenie wersji to rozwiązania zapewniające dobrą wydajność podczas obsługi współbieżności w zapytaniach zawierających operacje zarówno pobierania, jak i uaktualniania danych.
- W wersji MySQL 5.6 silnik InnoDB obsługuje wyszukiwanie pełnego tekstu oraz indeksy FULLTEXT.

Domyślnie, silnik InnoDB zarządza tabelami za pomocą pojedynczego systemu przestrzeni tabel, zamiast używać plików dla poszczególnych tabel, jak ma to miejsce w większości innych silników bazy danych. Przestrzeń tabeli składa się z jednego lub więcej plików i może zawierać zwykłe partycje. Silnik InnoDB traktuje przestrzeń tabeli jako wirtualny system plików, w którym zarządza zawartością wszystkich tabel InnoDB. Dlatego też wielkość tabel może przekraczać dozwoloną przez używany system plików dla pojedynczych plików. Istnieje również możliwość konfiguracji InnoDB w celu używania oddzielnej przestrzeni tabeli dla każdej tabeli. W tym przypadku dla każdej tabeli w katalogu bazy danych zostanie utworzony plik *.idb*.

Aby skonfigurować poszczególne przestrzenie tabel, należy ustawić zmienną systemową `innodb_file_per_table` w trakcie uruchamiania lub już podczas działania serwera. Włączenie wymienionej zmiennej powoduje włączenie także innych funkcji InnoDB, między innymi szybkiego skracania tabeli i stosowania takiego formatu przechowywania rekordów, który pozwala na znacznie efektywniejsze przetwarzanie tabeli dla pewnych rodzajów danych. Więcej informacji znajdziesz w podpunkcie 12.5.3.1.4, zatytułowanym „Używanie oddzielnych przestrzeni tabel dla każdej tabeli InnoDB”.

2.6.1.4. Silnik MyISAM

Silnik MyISAM oferuje następujące funkcje:

- Kompresję kluczy, gdy przechowywane są kolejne, podobne wartości ciągu tekstowego w indeksie. MyISAM może także kompresować podobne wartości liczbowe indeksu, ponieważ tego typu wartości są przechowywane w najbardziej

znaczącym bajcie. (Wartości indeksu mają tendencję do większej różnorodności w dolnych bajtach, więc najbardziej znaczące bajty częściej są poddawane kompresji). W celu włączenia kompresji wartości liczbowych należy użyć opcji `PACK_KEYS=1` podczas tworzenia tabeli MyISAM.

- Więcej funkcji dla kolumn `AUTO_INCREMENT` niż oferowane przez inne silniki bazy danych. Więcej informacji na ten temat znajdziesz w podrozdziale 3.4, zatytułowanym „Praca z sekwencjami”.
- Każda tabela MyISAM ma ustawioną flagę podczas przeprowadzania operacji sprawdzania tabeli. Tabele MyISAM mają również flagi wskazujące, czy tabela została prawidłowo zamknięta po ostatnim użyciu. Jeżeli zamknięcie serwera nastąpiło w niestandardowy sposób lub doszło do awarii komputera, dzięki wspomnianym flagom można określić, czy tabele wymagają sprawdzenia. Aby przeprowadzić to automatycznie, należy uruchomić serwer wraz ze zmienną systemową `myisam_recover_options` z przypisaną wartością zawierającą opcję `FORCE`. To spowoduje, że serwer będzie sprawdzał flagi tabeli w trakcie każdego otwierania tabeli MyISAM i automatycznie przeprowadzi jej naprawę, gdy wystąpi potrzeba. Zapoznaj się z punktem 14.3.1, zatytułowanym „Używanie możliwości serwera w zakresie automatycznej naprawy”.
- Wyszukiwanie pełnego tekstu i indeksy `FULLTEXT`.
- Przestrzenne typy danych i indeksy `SPATIAL`.

2.6.1.5. Silnik MEMORY

Silnik bazy danych MEMORY używa tabel przechowywanych w pamięci i posiadających rekordy o stałej wielkości. Dwie wymienione cechy powodują, że tego typu silnik działa bardzo szybko.

Tabele silnika MEMORY są tymczasowe pod tym względem, że ich zawartość znika po zamknięciu serwera. Tabela MEMORY będzie oczywiście istniała po ponownym uruchomieniu serwera, ale będzie pusta. Jednak, w przeciwieństwie do tabel tymczasowych utworzonych za pomocą zapytania `CREATE TEMPORARY TABLE`, tabele MEMORY są widoczne dla innych klientów.

Tabele typu MEMORY posiadają cechy charakterystyczne pozwalające na ich prostszą, a tym samym znacznie szybszą obsługę:

- Domyślnie, tabele MEMORY używają indeksów hash, które są bardzo szybkie podczas porównywania równości, ale wolne w trakcie porównań zakresu. Dlatego też indeksy hash są używane jedynie do porównań przeprowadzanych za pomocą operatorów równości `=` i `<=>`, natomiast nie do operacji porównań wykonywanych przez operatory `<` lub `>`. Z wymienionego powodu indeksy hash nie są używane w klauzulach `ORDER BY`.
- Rekordy są przechowywane w tabelach MEMORY przy użyciu formatu o stałej wielkości rekordu, co ułatwia jego przetwarzanie. Dlatego też nie można używać typów `BLOB` i `TEXT`, które są typami o zmiennej wielkości danych.

Wprowadź `VARCHAR` to również typ o zmiennej wielkości danych, ale jest on dozwolony, ponieważ wewnętrznie jest traktowany jako `CHAR`, czyli typ o stałej wielkości danych.

Aby użyć tabeli `MEMORY` do porównania wartości zakresu za pomocą operatorów takich jak `<`, `>` lub `BETWEEN`, zamiast indeksu `hash` należy skorzystać z indeksu `BTREE`. Zapoznaj się z podpunktem 2.6.4.2, „Tworzenie indeksów”, i punktem 5.1.3, „Wybór indeksów”.

2.6.1.6. Silnik NDB

NDB to silnik bazy danych dla klastra MySQL. Dla tego silnika bazy danych serwer MySQL w rzeczywistości działa jako klient klastra innych procesów zapewniających dostęp do tabel NDB. Procesy węzłów klastra komunikują się wzajemnie w celu zarządzania tabelami w pamięci. Same tabele są replikowane w procesach klastra, co ma na celu zapewnienie redundancji. Przechowywanie danych w pamięci zapewnia wysoką wydajność, ponieważ daje odporność na awarię dowolnego węzła.

Konfiguracja silnika NDB wykracza poza zakres tematyczny niniejszej książki, a więc nie będzie omówiona. Niezbędne informacje znajdziesz jednak w podręczniku użytkownika MySQL.

2.6.1.7. Inne silniki bazy danych

MySQL oferuje także inne silniki bazy danych, które umieściłem w tym punkcie w ramach kategorii „różne”:

- Silnik `ARCHIVE` zapewnia archiwalną pamięć masową. Jest przeznaczony do przechowywania ogromnej liczby rekordów, które po wstawieniu nigdy nie będą modyfikowane. Z tego powodu obsługuje jedynie ograniczoną liczbę zapytań. Zapytania `INSERT` i `SELECT` działają, natomiast `REPLACE` zawsze działa jak `INSERT`, a ponadto nie można używać `DELETE` lub `UPDATE`. W celu oszczędności pamięci masowej rekordy są przechowywane w postaci skompresowanej i dekompresowane w trakcie pobierania. Tabela `ARCHIVE` może zawierać zindeksowaną kolumnę `AUTO_INCREMENT`, pozostałe kolumny nie mogą być indeksowane.
- Silnik `BLACKHOLE` tworzy tabele, dla których operacje zapisu są ignorowane, natomiast odczytu nic nie zwracają. To jest bazodanowy odpowiednik urządzenia `/dev/null` w systemie UNIX.
- Silnik `CSV` przechowuje dane w formacie wartości rozdzielonych przecinkami. Dla każdej tabeli w katalogu bazy danych tworzony jest plik `.csv`. To jest zwykły plik tekstowy, w którym każdy rekord tabeli zajmuje jeden wiersz. Silnik `CSV` nie obsługuje indeksowania.
- Silnik `FEDERATED` zapewnia dostęp do tabel zarządzanych przez inne serwery MySQL. Innymi słowy, zawartość tabeli `FEDERATED` tak naprawdę jest zdalna. Dla tabeli `FEDERATED` trzeba podać nazwę komputera, w którym działa inny serwer, a także nazwę użytkownika i hasło do konta we wspomnianym serwerze. Kiedy uzyskujesz dostęp do tabeli `FEDERATED`, serwer lokalny nawiązuje połączenie ze zdalnym serwerem, używając podanego konta.

- Silnik MERGE zapewnia możliwość zgrupowania zestawu tabel MyISAM w pojedynczą logiczną jednostkę. Wykonanie zapytania do tabeli MERGE w efekcie powoduje zapytanie do wszystkich zgrupowanych tabel. Jedną z zalet takiego rozwiązania jest to, że można przekroczyć maksymalną wielkość tabeli określoną przez system plików dla poszczególnych tabel MyISAM. Tabele partycjonowane są alternatywą dla tabel MERGE i jednocześnie nie są ograniczone jedynie do tabel MyISAM. Zapoznaj się z podpunktem 2.6.2.5, zatytułowanym „Używanie tabel partycjonowanych”.

2.6.2. Tworzenie tabel

W celu utworzenia tabeli użyj zapytania `CREATE TABLE`. Musisz mieć uprawnienie `CREATE` dla tabeli. Pełna składnia zapytania jest złożona z powodu istnienia dużej liczby klauzul opcjonalnych, ale w praktyce używanie zapytania jest całkiem proste. Na przykład, większość zapytań `CREATE TABLE` przedstawionych w rozdziale 1. było całkiem nieskomplikowanych. Jeżeli rozpoczniesz od podstawowych form, które następnie będziesz rozbudowywał, to nie powinieneś mieć zbyt dużych problemów.

W minimalnej postaci zapytanie `CREATE TABLE` wskazuje nazwę tabeli oraz listę jej kolumn, na przykład:

```
CREATE TABLE mytbl
(
  name CHAR(20),
  birth DATE NOT NULL,
  weight INT,
  sex ENUM('F','M')
);
```

Oprócz definicji kolumn, w chwili tworzenia tabeli możesz również wskazać sposób jej indeksowania. Inną możliwością jest pozostawienie nieindeksowanej tabeli w trakcie jej tworzenia i dodanie indeksu dopiero później. W przypadku tabel MyISAM to jest dobra strategia, jeśli planujesz wstawienie do tabeli ogromnej ilości danych, zanim zaczniesz wykonywać zapytania do niej. Uaktualnienie indeksów podczas wstawiania kolejnych rekordów jest znacznie wolniejsze niż wstawienie danych do nieindeksowanej tabeli MyISAM, a dopiero później utworzenie indeksów.

Podstawowa składnia zapytania `CREATE TABLE` została przedstawiona w rozdziale 1., zatytułowanym „Rozpoczęcie pracy z MySQL”. Informacje szczegółowe na temat tworzenia definicji kolumn znajdziesz w rozdziale 3., zatytułowanym „Typy danych”. Tutaj ogólnie zajmiemy się pewnymi ważnymi rozszerzeniami zapytania `CREATE TABLE`, dzięki którym zyskasz większą elastyczność podczas tworzenia tabel.

- Opcje tabeli modyfikujące cechy charakterystyczne tabeli w zakresie przechowywania danych.
- Tworzenie tabeli tylko wtedy, gdy tabela o podanej nazwie jeszcze nie istnieje.
- Tabele tymczasowe automatycznie usuwane przez serwer na koniec sesji klienta.

- Tworzenie tabeli na podstawie innej tabeli lub wyniku zapytania SELECT.
- Używanie tabel partycjonowanych.

2.6.2.1. Opcje tabeli

Aby zmodyfikować cechy charakterystyczne tabeli w zakresie przechowywania danych, w zapytaniu CREATE TABLE umieścić jedną lub więcej opcji po nawiasach okrągłych. Pełną listę opcji znajdziesz w opisie zapytania CREATE TABLE przedstawionym w dodatku E, zatytułowanym „Przewodnik po składni SQL”.

Jedną z opcji tabeli jest ENGINE=*nazwa_silnika*. Pozwala ona na wskazanie silnika bazy danych używanego przez tę tabelę. Na przykład, w celu utworzenia tabeli MEMORY lub MyISAM użyj następujących zapytań:

```
CREATE TABLE mytbl ( ... ) ENGINE=MEMORY;
CREATE TABLE mytbl ( ... ) ENGINE=MyISAM;
```

Nazwa silnika bazy danych nie rozróżnia wielkości liter. W przypadku braku opcji ENGINE serwer utworzy tabelę opartą na domyślnym silniku bazy danych. Domyślnym wbudowanym silnikiem w MySQL jest InnoDB, ale korzystając z instrukcji przedstawionych w punkcie 12.5.2, zatytułowanym „Wybór domyślnego silnika bazy danych”, można ustawić inny.

Jeżeli podasz nazwę niedostępnego silnika bazy danych, wyświetlone zostaną dwa komunikaty ostrzeżenia:

```
mysql> CREATE TABLE t (i INT) ENGINE=ARCHIVE;
Query OK, 0 rows affected, 2 warnings (0.01 sec)
mysql> SHOW WARNINGS;
```

Level	Code	Message
Warning	1286	Unknown storage engine 'ARCHIVE'
Warning	1266	Using storage engine InnoDB for table 't'

Aby mieć pewność, że dana tabela będzie używała konkretnego silnika bazy danych, musisz pamiętać o dodaniu opcji ENGINE do zapytania tworzącego tabelę. Ponieważ istnieje możliwość zmiany silnika domyślnego, jeśli pominiesz opcję ENGINE, to domyślny silnik bazy danych nie musi być tym, którego oczekujesz. Ponadto upewnij się, że zapytanie CREATE TABLE nie powoduje wygenerowania komunikatów ostrzeżeń. Wspomniane komunikaty często informują, że wskazany silnik jest niedostępny i zamiast niego został użyty domyślny.

Istnieje możliwość nakazania MySQL wygenerowania błędu, jeśli wskazany silnik bazy danych jest niedostępny (zamiast po prostu jego zamiany na domyślny). W tym celu trzeba włączyć tryb SQL o nazwie NO_ENGINE_SUBSTITUTION.

W celu określenia silnika bazy danych używanego przez daną tabelę należy wykonać zapytanie SHOW CREATE TABLE, a następnie w wyświetlonych danych wyjściowych odszukać opcję ENGINE:


```
mysql> SHOW CREATE TABLE t\G
***** 1. row *****
      Table: t
Create Table: CREATE TABLE `t` (
  `i` int(11) DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

Silnik bazy danych jest wymieniany również w danych wyjściowych zapytania `SHOW TABLE STATUS` lub w tabeli `INFORMATION_SCHEMA.TABLES`.

Opcje `MAX_ROWS` i `AVG_ROW_LENGTH` mogą pomóc w określeniu wielkości tabeli MyISAM. Domyślnie, silnik MyISAM tworzy table z wewnętrznym wskaźnikiem rekordu pozwalającym, aby pliki tabel osiągnęły maksymalnie 256 TB. Jeżeli użyjesz opcji `MAX_ROWS` i `AVG_ROW_LENGTH`, wówczas przekazujesz MyISAM informacje, że powinien użyć dla tabeli wskaźnika pozwalającego na przechowywanie co najmniej `MAX_ROWS` rekordów.

Aby zmodyfikować cechy charakterystyczne istniejącej tabeli w zakresie przechowywania danych, opcji tabeli można użyć w zapytaniu `ALTER TABLE`. Na przykład, w celu zmiany bieżącego silnika bazy danych na InnoDB dla tabeli `mytbl` należy wykonać poniższe zapytanie:

```
ALTER TABLE mytbl ENGINE=InnoDB;
```

Więcej informacji na temat zmiany silników bazy danych znajdziesz w punkcie 2.6.5, zatytułowanym „Zmiana struktury tabeli”.

2.6.2.2. Warunkowe tworzenie tabeli

W celu utworzenia tabeli tylko wtedy, gdy jeszcze nie istnieje, należy użyć zapytania `CREATE TABLE IF NOT EXISTS`. Wymienionego zapytania można użyć w aplikacji, w której nie przyjęto założenia, czy tabela powinna być wcześniej skonfigurowana. Oczywiście, aplikacja może podjąć próbę utworzenia tabeli. Modyfikator `IF NOT EXISTS` jest szczególnie użyteczny w przypadku skryptów uruchamianych jako zadania wsadowe w programie `mysql`. W tym kontekście zwykle zapytanie `CREATE TABLE` nie działa zbyt dobrze. W trakcie pierwszego uruchomienia zadania nastąpi utworzenie tabeli, jednak w trakcie drugiego uruchomienia wystąpi błąd, ponieważ tabela o podanej nazwie już istnieje. Użycie modyfikatora `IF NOT EXISTS` rozwiązuje ten problem. W trakcie pierwszego uruchomienia skryptu nastąpi jak wcześniej utworzenie tabeli. W trakcie drugiego i kolejnych uruchomień skryptu zapytanie odpowiedzialne za utworzenie tabeli zostanie zignorowane. W ten sposób wykonywanie zadania będzie kontynuowane.

Jeżeli używasz `IF NOT EXISTS`, to musisz mieć świadomość, że MySQL nie porównuje struktury tabeli w zapytaniu `CREATE TABLE` z istniejącą tabelą. Kiedy więc istnieje tabela o podanej nazwie, ale innej strukturze, wykonanie zapytania nie zakończy się niepowodzeniem. Jeżeli nie chcesz podjąć takiego ryzyka, lepszym rozwiązaniem jest poprzedzenie zapytania `CREATE TABLE` zapytaniem `DROP TABLE IF EXISTS`.

2.6.2.3. Tabele tymczasowe

Dodanie słowa kluczowego `TEMPORARY` do zapytania tworzącego tabelę powoduje, że serwer utworzy tabelę tymczasową, która jest automatycznie usuwana po zakończeniu danej sesji z serwerem:

```
CREATE TEMPORARY TABLE nazwa_tabeli ... ;
```

To jest użyteczne rozwiązanie, ponieważ nie trzeba wykonywać zapytania `DROP TABLE` w celu pozbycia się tabeli, a po nagłym zakończeniu sesji tabela również nie będzie istniała. Na przykład, jeśli masz skomplikowane zapytanie przechowywane w pliku wsadowym uruchamianym w programie `mysql` i nie chcesz czekać na jego zakończenie, wtedy możesz zamknąć skrypt, a serwer usunie wszystkie tabele tymczasowe utworzone przez ten skrypt.

W celu utworzenia tabeli tymczasowej używającej określonego silnika bazy danych dodaj opcję `ENGINE` do zapytania `CREATE TEMPORARY TABLE`.

Wprawdzie serwer automatycznie usuwa tabele tymczasowe po zakończeniu danej sesji, ale istnieje również możliwość ręcznego usunięcia danej tabeli po zakończeniu pracy z nią i tym samym zwolnienia zasobów serwera. To jest dobre rozwiązanie, jeśli sesja ma trwać jeszcze przez dłuższy czas oraz zwłaszcza w przypadku tabel tymczasowych typu `MEMORY`.

Tabela tymczasowa jest widoczna jedynie dla klienta, który ją utworzył. Poszczególne klienty mogą więc tworzyć tabele tymczasowe o takiej samej nazwie, a konflikty pomiędzy nimi i tak nie wystąpią, ponieważ każdy klient widzi jedynie tabele tymczasowe utworzone samodzielnie.

Nazwa tabeli tymczasowej może być taka sama jak istniejącej. To nie jest błąd, istniejąca (trwała) tabela nie zostanie również w żaden sposób uszkodzona. Zamiast tego istniejąca tabela stanie się ukryta (nieдоступna) dla klienta tworzącego tabelę tymczasową. Przyjmujemy założenie, że w bazie danych `sampdb` tworzysz tabelę tymczasową o nazwie `member`. Oryginalna tabela `member` stanie się ukryta, a odwołania do tabeli `member` będą kierowane do utworzonej tabeli tymczasowej. Jeżeli wykonasz zapytanie `DROP TABLE member`, nastąpi usunięcie tabeli tymczasowej i przywrócenie oryginalnej. Po zerwaniu połączenia z serwerem bez wcześniejszego usunięcia tabeli tymczasowej serwer automatycznie ją usunie. Po kolejnym nawiązaniu połączenia z powrotem dostępna będzie oryginalna tabela `member`. (Oryginalna tabela stanie się dostępna także po zmianie nazwy tabeli tymczasowej).

Mechanizm ukrywania nazwy działa tylko do jednego poziomu. Oznacza to, że nie możesz utworzyć dwóch tabel tymczasowych o takiej samej nazwie.

Gdy rozważasz użycie tabeli tymczasowej, musisz pamiętać o następujących kwestiach:

- Jeżeli program kliencki automatycznie nawiązuje ponowne połączenie z serwerem po jego zerwaniu, po ponownym połączeniu nie będzie żadnej tabeli tymczasowej. Jeżeli tabeli tymczasowej używasz do „ukrycia” istniejącej na stałe tabeli o takiej samej nazwie, to wspomniana trwała tabela będzie teraz używana. Na przykład, zapytanie `DROP TABLE` wykonane po niewykrytym ponownym nawiązaniu połączenia z serwerem spowoduje usunięcie trwałej tabeli. Aby uniknąć tego problemu, powinieneś używać zapytania `DROP TEMPORARY TABLE`.
- Ponieważ tabele tymczasowe są dostępne jedynie w sesji, w której zostały utworzone, nie są użyteczne przy stosowaniu mechanizmu puli połączeń. Wspomniany mechanizm po prostu nie gwarantuje, że do wykonania każdego zapytania będzie stosowane to samo zapytanie.
- W przypadku puli połączeń lub trwałego połączenia połączenie z serwerem MySQL niekoniecznie zostanie zamknięte po zakończeniu działania aplikacji.

Wspomniane mechanizmy mogą zachować otwarte połączenie dla innych klientów. Dlatego też nie można przyjąć założenia, że tabele tymczasowe zostaną automatycznie usunięte po zakończeniu działania aplikacji.

2.6.2.4. Utworzenie tabeli na podstawie innej lub wyniku zapytania

Czasami użyteczne jest utworzenie kopii tabeli. Na przykład, masz plik danych, który chcesz wczytać do tabeli za pomocą zapytania `LOAD DATA`, ale nie jesteś do końca pewien, jakie powinny być opcje określające format danych. Jeżeli za pierwszym razem nie użyjesz odpowiednich opcji, skutkiem może być uszkodzenie rekordów w oryginalnej tabeli. Wykorzystanie pustej kopii oryginalnej tabeli pozwala na przeprowadzenie eksperymentów z opcjami zapytania `LOAD DATA` określającymi ograniczniki kolumn i wiersza, aż uznasz, że nowe rekordy są wstawiane prawidłowo. Następnie dane z pliku możesz wstawić do oryginalnej tabeli, ponownie wykonując zapytanie `LOAD DATA`, ale już z nazwą właściwej tabeli i odpowiednio dobranymi opcjami.

Czasami żądane jest zapisanie wyniku zapytania w tabeli zamiast jego wyświetlenia na ekranie. Dzięki zachowaniu wyniku zapytania możesz się później do niego odnieść (na przykład w celu przeprowadzenia dalszej analizy) bez konieczności ponownego wykonywania zapytania.

MySQL oferuje dwa zapytania pozwalające na tworzenie nowych tabel na podstawie innych tabel lub wyniku zapytania. Te zapytania oczywiście mają wady i zalety:

- Zapytanie `CREATE TABLE ... LIKE` powoduje utworzenie nowej tabeli jako pustej kopii oryginalnej. Następuje dokładne skopiowanie struktury wskazanej tabeli, więc każda kolumna zachowuje wszystkie zdefiniowane dla niej atrybuty. Struktura indeksu również zostaje skopiowana. Jednak nowa tabela jest pusta; jeśli chcesz ją wypełnić, to trzeba wykonać drugie zapytanie (na przykład `INSERT INTO ... SELECT`). Ponadto, zapytanie `CREATE TABLE ... LIKE` nie tworzy nowej tabeli na podstawie jedynie części kolumn oryginalnej tabeli, nie mogą być też używane kolumny inne niż znajdujące się w oryginalnej tabeli.
- Zapytanie `CREATE TABLE ... SELECT` tworzy nową tabelę na podstawie wyniku zapytania `SELECT`. Domyślnie, zapytanie nie kopiuje wszystkich atrybutów kolumn, na przykład takich jak `AUTO_INCREMENT`. Ponadto, nie tworzy tabeli przez automatyczny wybór i skopiowanie danych oraz indeksów z oryginalnej tabeli, ponieważ zbiory wynikowe nie są indeksowane. Z drugiej strony, zapytanie `CREATE TABLE ... SELECT` pozwala na utworzenie i wypełnienie nowej tabeli danymi w pojedynczym zapytaniu. Umożliwia także utworzenie nowej tabeli jako fragmentu oryginalnej i wstawienie kolumn z innych tabel bądź utworzonych jako wynik wykonania wyrażeń.

Aby użyć zapytania `CREATE TABLE ... LIKE` do utworzenia pustej kopii istniejącej tabeli, wykonaj zapytanie podobne do przedstawionego poniżej:

```
CREATE TABLE nowa_nazwa_tabeli LIKE nazwa_tabeli;
```

W celu utworzenia pustej kopii tabeli, a następnie wypełnienia jej danymi pochodzącymi z oryginalnej tabeli użyj zapytań `CREATE TABLE ... LIKE` i `INSERT INTO ... SELECT`:

```
CREATE TABLE nowa_nazwa_tabeli LIKE nazwa_tabeli;
INSERT INTO nowa_nazwa_tabeli SELECT * FROM nazwa_tabeli;
```

Aby utworzyć tabelę jako tymczasową kopię samej siebie, należy użyć słowa kluczowego `TEMPORARY`:

```
CREATE TEMPORARY TABLE nazwa_tabeli LIKE nazwa_tabeli;
INSERT INTO nazwa_tabeli SELECT * FROM nazwa_tabeli;
```

Użycie tabeli tymczasowej o takiej samej nazwie jak oryginalna może być użyteczne, gdy chcesz wypróbować zapytania modyfikujące zawartość tabeli, ale nie chcesz zmieniać oryginalnej. W celu wykorzystania utworzonych wcześniej skryptów używających nazwy oryginalnej tabeli nie musisz ich edytować, aby skrypty odwoływały się do innej tabeli. Po prostu dodaj zapytania `CREATE TEMPORARY TABLE` i `INSERT` na początku skryptu. Skrypt utworzy kopię tymczasową i będzie działał na tej kopii. Po zakończeniu działania skryptu serwer usunie utworzone tabele tymczasowe. (Pamiętaj o kwestiach związanych z automatycznym ponownym nawiązywaniem połączenia, o których wspomniano w poprzednim punkcie).

W celu wstawienia do nowej tabeli jedynie wybranych rekordów z oryginalnej tabeli trzeba dodać klauzulę `WHERE` i wskazać rekordy przeznaczone do wstawienia. Poniższe zapytanie tworzy nową tabelę o nazwie `student_f`, zawierającą jedynie rekordy tabeli `student` opisujące dziewczynki:

```
CREATE TABLE student_f LIKE student;
INSERT INTO student_f SELECT * FROM student WHERE sex = 'f';
```

Jeżeli nie przejmujesz się zachowaniem dokładnych definicji z oryginalnej tabeli, zapytanie `CREATE TABLE ... SELECT` czasami jest łatwiejsze w użyciu niż `CREATE TABLE ... LIKE`, ponieważ powoduje utworzenie i wypełnienie tabeli za pomocą tylko jednego zapytania:

```
CREATE TABLE student_f SELECT * FROM student WHERE sex = 'f';
```

Zapytanie `CREATE TABLE ... SELECT` może również utworzyć nowe tabele, które nie będą zawierały dokładnie tego samego zestawu kolumn jak w istniejącej tabeli. Możesz je więc wykorzystać do utworzenia nowej tabeli przedstawiającej wynik dowolnego zapytania `SELECT`. W ten sposób niespotykane łatwo tworzona jest tabela wypełniona już interesującymi Cię danymi i gotowa do wykorzystania w kolejnych zapytaniach. Jednak jeśli nie zachowasz ostrożności, wtedy nowa tabela może mieć zdefiniowane dziwne nazwy kolumn. Podczas tworzenia tabeli przez wybór umieszczanych w niej danych, nazwy kolumn są pobierane z wybieranych kolumn. Dlatego też gdy kolumna jest wynikiem wykonania wyrażenia, nazwą kolumny stanie się tekst danego wyrażenia, co powoduje utworzenie tabeli wraz z dziwnymi nazwami kolumn:

```
mysql> CREATE TABLE mytb1 SELECT PI() * 2;
mysql> SELECT * FROM mytb1;
+-----+
| PI() * 2 |
+-----+
| 6.283185 |
+-----+
```

To niezbyt szczęśliwe rozwiązanie, ponieważ w takim przypadku do nazwy kolumny będzie się można bezpośrednio odwoływać jedynie jak do cytowanego identyfikatora:

```
mysql> SELECT `PI() * 2` FROM mytb1;
+-----+
| PI() * 2 |
+-----+
| 6.283185 |
+-----+
```

Aby uniknąć takiego problemu, należy stosować aliasy kolumn zdefiniowane jako nazwy, z którymi łatwiej pracować:

```
mysql> DROP TABLE mytb1;
mysql> CREATE TABLE mytb1 SELECT PI() * 2 AS mycol;
mysql> SELECT mycol FROM mytb1;
+-----+
| mycol |
+-----+
| 6.283185 |
+-----+
```

Podobna trudność wystąpi w przypadku wybrania z różnych tabel kolumn o takich samych nazwach. Przyjmujemy założenie, że tabele t1 i t2 mają kolumnę c, a naszym zadaniem jest utworzenie nowej tabeli na podstawie wszystkich kolumn pochodzących z wymienionych tabel. Wykonanie poniższego zapytania zakończy się niepowodzeniem, ponieważ nastąpi próba utworzenia tabeli wraz z dwoma kolumnami o nazwie c:

```
mysql> CREATE TABLE t3 SELECT * FROM t1 INNER JOIN t2;
ERROR 1060 (42S21): Duplicate column name 'c'
```

Rozwiązaniem problemu jest użycie aliasów i nadanie każdej kolumnie nowej tabeli unikalnej nazwy:

```
mysql> CREATE TABLE t3 SELECT t1.c, t2.c AS c2
-> FROM t1 INNER JOIN t2;
```

Jak już wcześniej wspomniano, wadą zapytania `CREATE TABLE ... SELECT` jest fakt, że nie uwzględnia ono w strukturze nowo tworzonej tabeli wszystkich cech charakterystycznych oryginalnych danych. Na przykład, utworzenie tabeli przez wybór danych dla niej nie powoduje skopiowania indeksów z oryginalnej tabeli, a ponadto może doprowadzić do utraty atrybutów kolumny. Zachowane atrybuty obejmują: informacje, czy kolumna ma wartość NULL lub NOT NULL, kodowanie znaków, kolejność sortowania, wartość domyślną i komentarz kolumny.

W pewnych przypadkach można wymusić użycie określonych atrybutów w nowej tabeli za pomocą wywołania funkcji `CAST()` w części `SELECT` zapytania. Przedstawione poniżej zapytanie `CREATE TABLE ... SELECT` wymusza, aby kolumny wybrane przez `SELECT` zostały potraktowane jako `INT UNSIGNED`, `TIME` i `DECIMAL(10,5)`, co można zweryfikować, wykonując zapytanie `DESCRIBE`:

```
mysql> CREATE TABLE mytbl SELECT
-> CAST(1 AS UNSIGNED) AS i,
-> CAST(CURTIME() AS TIME) AS t,
-> CAST(PI() AS DECIMAL(10,5)) AS d;
mysql> DESCRIBE mytbl;
```

Field	Type	Null	Key	Default	Extra
i	int(1) unsigned	NO		0	
t	time	YES		NULL	
d	decimal(10,5)	NO		0.00000	

Dozwolone typy rzutowania to `BINARY` (binarny ciąg tekstowy), `CHAR`, `DATE`, `DATETIME`, `TIME`, `SIGNED`, `SIGNED INTEGER`, `UNSIGNED`, `UNSIGNED INTEGER` i `DECIMAL`.

Istnieje również możliwość podania dokładnych definicji kolumn w części `CREATE TABLE`, aby zostały użyte dla kolumn wybranych przez zapytanie `SELECT`. Kolumny w dwóch częściach są dopasowane za pomocą nazwy (nie położenia), więc dostarczenie aliasów w części `SELECT` jest niezbędne w celu zapewnienia prawidłowego dopasowania:

```
mysql> CREATE TABLE mytbl (i INT UNSIGNED, t TIME, d DECIMAL(10,5))
-> SELECT
-> 1 AS i,
-> CAST(CURTIME() AS TIME) AS t,
-> CAST(PI() AS DECIMAL(10,5)) AS d;
mysql> DESCRIBE mytbl;
```

Field	Type	Null	Key	Default	Extra
i	int(10) unsigned	YES		NULL	
t	time	YES		NULL	
d	decimal(10,5)	YES		NULL	

Technika podawania dokładnych definicji pozwala na utworzenie kolumn liczbowych wraz z określoną precyzją i skalą, a także kolumn znakowych posiadających inną szerokość niż najdłuższa wartość w zbiorze wynikowym i na odwrót. Zwróć uwagę, że w tym przykładzie atrybuty `Null` i `Default` w niektórych kolumnach różnią się od przedstawionych w poprzednim. Jeśli zachodzi potrzeba, w części `CREATE TABLE` można podać dokładne definicje wymienionych atrybutów.

2.6.2.5. Używanie tabel partycjonowanych

MySQL obsługuje partycjonowanie tabeli, co pozwala na podział zawartości tabeli pomiędzy fizycznie różne położenia pamięci masowej. Dzięki sekcjonowaniu zawartości tabeli powstała w ten sposób tabela partycjonowana oferuje następujące korzyści:

- Zawartość tabeli może zostać rozproszona między wiele urządzeń, co poprawia czas dostępu na skutek zalet wynikających z jednocześnie przeprowadzanych operacji wejścia-wyjścia w wielu urządzeniach.
- Optymalizator może być w stanie zlokalizować operacje wyszukiwania w określonych partycjach lub równolegle przeszukiwać partycje.

W celu utworzenia tabeli partycjonowanej w zapytaniu `CREATE TABLE` jak zwykle podaj listę kolumn i indeksów. Oprócz tego trzeba dodać klauzulę `PARTITION BY`, definiującą funkcję partycjonowania, która będzie użyta do przypisania rekordów partycjom i prawdopodobnie innych operacji powiązanych z partycjami. Funkcja partycjonowania powoduje przypisanie rekordów na podstawie zakresów, list wartości lub wartości hash:

- Użyj zakresu partycjonowania, kiedy rekordy zawierają domeny wartości takich jak daty, poziom dochodu lub waga i można je podzielić na oddzielne zakresy.
- Użyj list partycjonowania, kiedy sensownym rozwiązaniem jest podanie dokładnych list wartości dla poszczególnych partycji, na przykład zestawów kodów pocztowych, prefiksów numerów telefonów lub identyfikatorów danych, które można pogrupować na regiony geograficzne.
- Użyj partycjonowania hash do umieszczenia rekordów w partycjach na podstawie wartości hash obliczonych dla kluczy rekordów. Możesz samodzielnie dostarczyć funkcję hash lub wskazać bazie danych MySQL kolumny, na podstawie których mają być obliczone wartości hash za pomocą wbudowanej funkcji.

Funkcja partycjonowania musi być deterministyczna, aby te same wartości danych wejściowych skutkowały umieszczaniem rekordów w tej samej partycji. To eliminuje funkcje takie jak `RAND()` lub `NOW()`.

Przyjmujemy założenie, że chcesz utworzyć tabelę przeznaczoną do przechowywania wpisów dziennika zdarzeń składających się z daty i opisowego ciągu tekstowego. Ponadto, masz gromadzone przez kilka lat dane wspomnianego dziennika, które chcesz umieścić w tabeli. W przypadku danych zawierających datę najbardziej naturalnym rozwiązaniem wydaje się zastosowanie partycjonowania na podstawie zakresu. Aby przypisać rekordy z poszczególnych lat danej partycji, należy użyć roku z wartości daty:

```
CREATE TABLE log_partition
(
  dt    DATETIME NOT NULL,
  info  VARCHAR(100) NOT NULL,
  INDEX (dt)
)
PARTITION BY RANGE(YEAR(dt))
(
  PARTITION p0 VALUES LESS THAN (2010),
  PARTITION p1 VALUES LESS THAN (2011),
  PARTITION p2 VALUES LESS THAN (2012),
  PARTITION p3 VALUES LESS THAN (2013),
  PARTITION pmax VALUES LESS THAN MAXVALUE
);
```

Partycji MAXVALUE zostały przypisane wszystkie rekordy zawierające datę z roku 2014 lub późniejszego. Po nadejściu roku 2014 można podzielić tę partycję, aby wpisy dotyczące roku 2014 znajdowały się w oddzielnej partycji, a dotyczące roku 2015 były w partycji MAXVALUE:

```
ALTER TABLE log_partition REORGANIZE PARTITION pmax
INTO (
    PARTITION p4 VALUES LESS THAN (2014),
    PARTITION pmax VALUES LESS THAN MAXVALUE
);
```

Domyślnie, MySQL przechowuje partycje w katalogu bazy danych, do której należy tabela partycjonowana. Aby umieścić partycje w różnych lokalizacjach (na przykład w odmiennych urządzeniach fizycznych), konieczne jest użycie opcji partycji DATA_DIRECTORY i INDEX_DIRECTORY. Więcej informacji na temat składni wymienionych oraz innych opcji partycjonowania znajdziesz w opisie zapytania CREATE TABLE, który przedstawiono w dodatku E, zatytułowanym „Przewodnik po składni SQL”.

2.6.3. Usuwanie tabel

Usunięcie tabeli jest znacznie łatwiejsze niż jej utworzenie, ponieważ nie trzeba podawać żadnych danych dotyczących formatu zawartości tabeli. Trzeba jedynie wskazać tabelę oraz oczywiście posiadać uprawnienia do wykonania zapytania DROP:

```
DROP TABLE nazwa_tabeli;
```

W bazie danych MySQL zapytanie DROP TABLE ma kilka użytecznych rozszerzeń. Aby usunąć więcej niż tylko jedną tabelę, można je wymienić w pojedynczym zapytaniu:

```
DROP TABLE nazwa_tabeli1, nazwa_tabeli2, ... ;
```

Próba usunięcia nieistniejącej tabeli domyślnie powoduje wygenerowanie błędu. Zawieszenie wspomnianego błędu i wygenerowanie jedynie ostrzeżenia o nieistniejącej tabeli następuje po użyciu w zapytaniu modyfikatora IF EXISTS:

```
DROP TABLE IF EXISTS nazwa_tabeli;
```

Jeżeli zapytanie generuje ostrzeżenia, można je wyświetlić za pomocą zapytania SHOW WARNINGS.

Modyfikator IF EXISTS jest szczególnie użyteczny w skryptach uruchamianych w programie klienckim mysql. Domyślnie, po wystąpieniu błędu program mysql kończy działanie, a błędem jest próba usunięcia nieistniejącej tabeli. Na przykład, możesz mieć skrypt konfiguracyjny tworzący tabele używane jako podstawa do późniejszych operacji przetwarzania zdefiniowanych w innych skryptach. W takim przypadku chcesz mieć pewność, że przed uruchomieniem skryptu konfiguracyjnego wszystko zostało odpowiednio przygotowane. Jeśli użyjesz zapytania DROP TABLE na początku skryptu, to podczas pierwszego uruchomienia skryptu wykonanie wymienionego zapytania zakończy się niepowodzeniem, ponieważ tabele nigdy nie zostały utworzone. Użycie modyfikatora IF EXISTS rozwiązuje

ten problem. Jeśli tabele istnieją, to zostaną usunięte, natomiast jeśli nie istnieją, to nie nastąpi wygenerowanie błędu, a skrypt będzie kontynuował działanie.

W celu usunięcia tabeli tylko wtedy, gdy jest tabelą tymczasową, w zapytaniu trzeba użyć słowa kluczowego `TEMPORARY`:

```
DROP TEMPORARY TABLE nazwa_tabeli;
```

2.6.4. Indeksowanie tabel

Indeks to podstawowy sposób skrócenia czasu dostępu do zawartości tabeli, szczególnie w przypadku zapytań obejmujących złączenia w wielu tabelach. To jest wyjątkowo ważny temat, dlatego większość analizy przedstawionej w rozdziale dotyczy indeksów: dlaczego należy ich używać, w jaki sposób działają oraz jaki jest najlepszy sposób ich wykorzystania w celu optymalizacji zapytań (patrz rozdział 5., zatytułowany „Optymalizacja zapytań”). W tym punkcie zostaną przedstawione cechy charakterystyczne indeksów w różnych typach tabel, a także składnia tworzenia i usuwania indeksów.

2.6.4.1. Cechy charakterystyczne w silnikach bazy danych

MySQL zapewnia dość dużą elastyczność w zakresie tworzenia indeksów:

- Można indeksować pojedynczą kolumnę lub wiele kolumn. Indeksy obejmujące wiele kolumn są znane jako indeksy złożone.
- Indeks może być ograniczony do przechowywania tylko unikalnych wartości lub może być dozwolone przechowywanie także powtarzających się wartości.
- W tabeli może znajdować się więcej niż tylko jeden indeks, aby w ten sposób pomóc w optymalizacji różnego typu zapytań wykonywanych względem tabeli.
- W przypadku tekstowych typów danych innych niż `ENUM` i `SET` istnieje możliwość indeksowania prefiksu kolumny, to znaczy jedynie n znaków od lewej strony lub n bajtów w binarnych ciągach tekstowych. (Dla kolumn typu `BLOB` i `TEXT` indeks można skonfigurować tylko po podaniu długości prefiksu). Jeżeli kolumna jest w przeważającej większości unikalna i ma zdefiniowany prefiks o pewnej długości, takie rozwiązanie zwykle nie wpływa negatywnie na wydajność, a może nawet ją zwiększyć. Wynika to z faktu, że indeksowanie prefiksu kolumny zamiast całej kolumny może znacznie zmniejszyć wielkość indeksu, a tym samym skrócić czas dostępu do niego.

Nie każdy silnik bazy danych oferuje wszystkie funkcje indeksu. W tabeli 2.3 podsumowano właściwości indeksu w wybranych silnikach bazy danych MySQL. Ta tabela nie zawiera silnika `MERGE`, ponieważ tabele w silniku `MERGE` to po prostu tabele `MyISAM` i mają podobne cechy charakterystyczne indeksów. W tabeli zabrakło również silników `ARCHIVE`, `BLACKHOLE` i `CSV`, które obsługują indeksy w ograniczonym zakresie lub w ogóle.

Tabela 2.3. Cechy charakterystyczne indeksów w silnikach bazy danych

Cecha charakterystyczna indeksu	InnoDB	MyISAM	MEMORY
Czy wartości NULL są dozwolone?	Tak	Tak	Tak
Maksymalna liczba kolumn w indeksie	16	16	16
Maksymalna liczba indeksów w tabeli	64	64	64
Wyrażona w bajtach maksymalna wielkość rekordu indeksu	3072	1000	3072
Prefiksy kolumny indeksu	Tak	Tak	Tak
Wyrażona w bajtach maksymalna wielkość prefiksu	767	1000	3072
Indeks typu BLOB/TEXT	Tak	Tak	Nie
Indeks typu FULLTEXT	Począwszy od wersji 5.6.4	Tak	Nie
Indeks typu SPATIAL	Nie	Tak	Nie
Indeks typu HASH	Nie	Nie	Tak

Jedną z implikacji istnienia różnic w cechach charakterystycznych indeksów dla poszczególnych silników bazy danych jest fakt, że jeśli wymagasz indeksu o określonych właściwościach, to możesz nie mieć możliwości użycia pewnych rodzajów tabel. Na przykład, aby użyć indeksu HASH, konieczne jest utworzenie tabeli typu MEMORY. Z kolei do zindeksowania kolumny typu TEXT trzeba użyć tabeli InnoDB lub MyISAM.

W celu przeprowadzenia konwersji istniejącej tabeli na inny silnik bazy danych, pozwalający na użycie indeksu o bardziej odpowiednich cechach charakterystycznych, możesz wykorzystać zapytanie `ALTER TABLE`. Przyjmijmy założenie, że masz tabelę InnoDB w serwerze MySQL 5.5, ale zachodzi potrzeba przeprowadzania operacji wyszukiwania przy użyciu indeksu typu FULLTEXT. W serwerze MySQL 5.5 jest on obsługiwany tylko przez silnik MyISAM. Konwersję tabeli możesz przeprowadzić za pomocą poniższego zapytania:

```
ALTER TABLE nazwa_tabeli ENGINE=MyISAM;
```

2.6.4.2. Tworzenie indeksów

MySQL pozwala na tworzenie wielu rodzajów indeksów:

- Indeks unikalny. Niedozwolone są powtarzające się wartości w indeksach obejmujących jedną kolumnę oraz powtarzające się kombinacje wartości w indeksach obejmujących wiele kolumn (złożonych).
- Indeks zwykły (nieunikalny). Ten indeks daje zalety wynikające z indeksu, ale pozwala na przechowywanie powtarzających się wartości.

- Indeks typu FULLTEXT, używany do przeprowadzania operacji wyszukiwania pełnego tekstu. Ten rodzaj indeksu jest obsługiwany jedynie dla tabel MyISAM (a począwszy od MySQL 5.6.4, także InnoDB). Więcej informacji na jego temat znajdziesz w podrozdziale 2.14, zatytułowanym „Wyszukiwanie pełnego tekstu”.
- Indeks typu SPATIAL. Ten rodzaj indeksu może być używany jedynie w tabelach MyISAM zawierających wartości przestrzenne, które pokrótce zostały opisane w punkcie 3.1.4, zatytułowanym „Wartości przestrzenne”.
- Indeks typu HASH. To jest domyślny typ indeksu dla tabel MEMORY, choć możesz to zmienić i zamiast indeksu typu HASH utworzyć indeks typu BTREE.

Definicje indeksu dla nowej tabeli można umieszczać w zapytaniu CREATE TABLE. Na przykład, zapoznaj się z punktem 1.4.6, zatytułowanym „Tworzenie tabel”. Aby dodać indeks do istniejącej tabeli, należy użyć zapytania ALTER TABLE lub CREATE INDEX. (Wewnętrznie MySQL mapuje zapytania CREATE INDEX na ALTER TABLE).

Zapytanie ALTER TABLE ma znacznie większe możliwości niż CREATE INDEX, ponieważ pozwala na utworzenie dowolnego rodzaju indeksu obsługiwanego przez MySQL, na przykład:

```
ALTER TABLE nazwa_tabeli ADD INDEX nazwa_indeksu (kolumny_indeksu);
ALTER TABLE nazwa_tabeli ADD UNIQUE nazwa_indeksu (kolumny_indeksu);
ALTER TABLE nazwa_tabeli ADD PRIMARY KEY (kolumny_indeksu);
ALTER TABLE nazwa_tabeli ADD FULLTEXT nazwa_indeksu (kolumny_indeksu);
ALTER TABLE nazwa_tabeli ADD SPATIAL nazwa_indeksu (kolumny_indeksu);
```

W powyższych zapytaniach *nazwa_tabeli* to nazwa tabeli, do której powinien zostać dodany indeks, *kolumny_indeksu* to nazwa kolumny lub rozdzielone przecinkami nazwy kolumn obejmujących indeks. Nazwa indeksu (*nazwa_indeksu*) jest opcjonalna. Jeśli ją pominiesz, MySQL wybierze nazwę na podstawie pierwszej indeksowanej kolumny.

Jeśli indeks jest typu PRIMARY KEY lub SPATIAL, indeksowana kolumna musi być zdefiniowana jako NOT NULL. Pozostałe indeksy zezwalają na indeksowanie kolumn zawierających wartości NULL.

Pojedyncze zapytanie ALTER TABLE może zawierać wiele operacji zmiany tabeli, o ile zostaną rozdzielone przecinkami. Dzięki temu masz możliwość jednoczesnego utworzenia wielu indeksów, co jest szybsze niż ich dodawanie pojedynczo za pomocą kolejnych zapytań ALTER TABLE.

Aby ograniczyć indeks do przechowywania jedynie unikalnych wartości, należy go utworzyć jako PRIMARY KEY lub UNIQUE. Dwa wymienione typy indeksów są bardzo podobne, ale między nimi występują dwie różnice:

- Tabela może zawierać tylko jeden indeks typu PRIMARY KEY. Wynika to z faktu, że wymieniony indeks zawsze jest podstawowy (PRIMARY), a tabela nie może mieć dwóch indeksów o takiej samej nazwie. Za to w tabeli może znajdować się wiele indeksów typu UNIQUE.
- W przeciwieństwie do indeksu typu UNIQUE, indeks typu PRIMARY KEY nie może zawierać wartości NULL. Jeżeli indeks UNIQUE może zawierać wartości NULL, może też zawierać wiele wartości NULL. (Wartość NULL jest uznawana za nierówną każdej innej wartości; dotyczy to także innej wartości NULL).

Zapytanie `CREATE INDEX` pozwala na dodanie większości typów indeksów, za wyjątkiem `PRIMARY KEY`:

```
CREATE INDEX nazwa_indeksu ON nazwa_tabeli (kolumny_indeksu);
CREATE UNIQUE INDEX nazwa_indeksu ON nazwa_tabeli (kolumny_indeksu);
CREATE FULLTEXT INDEX nazwa_indeksu ON nazwa_tabeli (kolumny_indeksu);
CREATE SPATIAL INDEX nazwa_indeksu ON nazwa_tabeli (kolumny_indeksu);
```

W powyższych zapytaniach *`nazwa_tabeli`*, *`nazwa_indeksu`* i *`kolumny_indeksu`* mają takie samo znaczenie jak w przypadku zapytania `ALTER TABLE`. Jednak w przeciwieństwie do zapytania `ALTER TABLE`, tutaj nazwa indeksu jest wymagana i nie ma możliwości utworzenia wielu indeksów za pomocą pojedynczego zapytania.

Podczas definiowania indeksów w nowej tabeli tworzonej za pomocą zapytania `CREATE TABLE`, składnia zapytania jest podobna do stosowanej w `ALTER TABLE`, ale oprócz definicji kolumn podaje się jeszcze klauzule powodujące utworzenie indeksów:

```
CREATE TABLE nazwa_tabeli
(
    ... definicje kolumn ...
    INDEX nazwa_indeksu (kolumny_indeksu),
    UNIQUE nazwa_indeksu (kolumny_indeksu),
    PRIMARY KEY (kolumny_indeksu),
    FULLTEXT nazwa_indeksu (kolumny_indeksu),
    SPATIAL nazwa_indeksu (kolumny_indeksu),
    ...
);
```

Podobnie jak w przypadku zapytania `ALTER TABLE`, także w powyższym *`nazwa_indeksu`* jest opcjonalna. Jeśli ją pominiesz, MySQL wybierze ją za Ciebie.

Przypadkiem specjalnym jest utworzenie obejmującego pojedynczą kolumnę indeksu typu `PRIMARY KEY` lub `UNIQUE` przez dodanie klauzuli `PRIMARY KEY` lub `UNIQUE` na końcu definicji kolumn. Na przykład, poniższe zapytania `CREATE TABLE` mają dokładnie takie samo działanie:

```
CREATE TABLE mytbl
(
    i INT NOT NULL PRIMARY KEY,
    j CHAR(10) NOT NULL UNIQUE
);
```

```
CREATE TABLE mytbl
(
    i INT NOT NULL,
    j CHAR(10) NOT NULL,
    PRIMARY KEY (i),
    UNIQUE (j)
);
```

Domyślnym typem indeksu dla tabeli `MEMORY` jest `HASH`. Indeks w postaci wartości hash charakteryzuje się bardzo szybkimi operacjami wyszukiwania, a to jest typowe przeznaczenie tabel `MEMORY`. Jednak jeśli planujesz użycie tabeli `MEMORY` do przeprowadzania operacji porównań zakresu wartości (na przykład `id < 100`), wtedy

indeks typu HASH nie sprawdza się zbyt dobrze. Lepszym rozwiązaniem będzie wówczas utworzenie indeksu typu BTREE przez dodanie klauzuli USING BTREE do definicji indeksu:

```
CREATE TABLE name1ist
(
  id    INT NOT NULL,
  name  CHAR(100),
  INDEX (id) USING BTREE
) ENGINE=MEMORY;
```

Aby zindeksować prefiks kolumny ciągu tekstowego, składnia dla nazwy kolumny w definicji indeksu to *nazwa_kolumny(n)* zamiast po prostu *nazwa_kolumny*. Wartość prefiksu (*n*) wskazuje, że indeks powinien zawierać pierwszych *n* bajtów wartości kolumny (typy binarnych ciągów tekstowych) lub pierwsze *n* znaków (typy niebinarnych ciągów tekstowych). Na przykład, poniższe zapytanie tworzy tabelę wraz z kolumnami typu CHAR i BINARY. Indeksowane będzie pierwszych 10 znaków kolumny CHAR i pierwsze 15 bajtów kolumny BINARY:

```
CREATE TABLE addresslist
(
  name    CHAR(30) NOT NULL,
  address BINARY(60) NOT NULL,
  INDEX (name(10)),
  INDEX (address(15))
);
```

Podczas indeksowania prefiksu kolumny ciągu tekstowego długość prefiksu, podobnie jak długość kolumny, jest wyrażona w tych samych jednostkach, co typ danych kolumny — bajtach dla binarnych ciągów tekstowych i znakach dla niebinarnych ciągów tekstowych. Jednak maksymalna wielkość wpisu indeksu jest wewnętrznie określana w bajtach. Wspomniane dwie jednostki miary będą takie same w przypadku kodowań znaków używających jednego bajta na znak, ale już nie w kodowaniach stosujących więcej niż jeden bajt na znak. Dla niebinarnych ciągów tekstowych stosujących kodowanie znaków, w którym używa się więcej niż jednego bajta na znak, baza danych MySQL przechowuje w indeksie tyle kompletnych znaków, na ile pozwala maksymalna wielkość wpisu indeksu wyrażona w bajtach.

W pewnych sytuacjach może się okazać nie tylko pożądane, ale również konieczne indeksowanie prefiksu kolumny zamiast całej kolumny:

- Prefiks jest wymagany do indeksowania kolumny typu BLOB i TEXT.
- Długość rekordów indeksu jest równa sumie długości części indeksu tworzących dany indeks. Jeżeli ta długość jest większa niż maksymalna dozwolona wielkość rekordu indeksu wyrażona w bajtach, wtedy można „zwęzić” indeks przez indeksowanie jedynie prefiksu kolumny. Przyjmijmy założenie, że tabela MyISAM używa kodowania latin1 (jeden bajt na znak) i zawiera cztery kolumny typu CHAR(255) o nazwach od c1 do c4. Wartość indeksu każdej pełnej kolumny to 255 bajtów, a więc indeks wszystkich czterech kolumn wymaga 1020 bajtów. Jednak maksymalna długość rekordu indeksu w MyISAM to 1000 bajtów, a więc

nie można utworzyć indeksu złożonego obejmującego całą zawartość wszystkich czterech kolumn. Istnieje jednak możliwość utworzenia indeksu obejmującego krótsze części niektórych lub wszystkich kolumn. Na przykład, możesz utworzyć indeks obejmujący pierwsze 250 znaków każdej kolumny.

Kolumny w indeksach typu FULLTEXT są indeksowane w pełni i nie posiadają prefiksów. Jeżeli podasz długość prefiksu dla kolumny w indeksie typu FULLTEXT, MySQL po prostu to zignoruje.

2.6.4.3. Usunięcie indeksu

W celu usunięcia indeksu należy użyć zapytania `DROP INDEX` lub `ALTER TABLE`. W przypadku zapytania `DROP INDEX` konieczne jest podanie nazwy usuwanego indeksu:

```
DROP INDEX nazwa_indeksu ON nazwa_tabeli;
```

Aby usunąć indeks PRIMARY KEY za pomocą zapytania `DROP INDEX`, nazwę PRIMARY trzeba podać jako cytowany identyfikator:

```
DROP INDEX `PRIMARY` ON nazwa_tabeli;
```

Powyższe zapytanie jest jednoznaczne, ponieważ w tabeli może znajdować się tylko jeden indeks PRIMARY KEY, a jego nazwą zawsze jest PRIMARY.

Podobnie jak zapytanie `CREATE INDEX`, także `DROP INDEX` jest wewnętrznie obsługiwane jako `ALTER TABLE`. Poprzednie zapytania `DROP INDEX` odpowiadają poniższym zapytaniom `ALTER TABLE`:

```
ALTER TABLE nazwa_tabeli DROP INDEX nazwa_indeksu;
ALTER TABLE nazwa_tabeli DROP PRIMARY KEY;
```

Jeżeli nie znasz nazw indeksów tabeli, możesz je sprawdzić za pomocą zapytań `SHOW CREATE TABLE` lub `SHOW INDEX`.

Operacja usunięcia kolumn w tabeli może mieć wpływ na indeksy. Usunięcie kolumny będącej częścią indeksu powoduje usunięcie tej kolumny również z indeksu. Jeżeli usuniesz wszystkie kolumny w indeksie, MySQL usunie cały indeks.

2.6.5. Zmiana struktury tabeli

Zapytanie `ALTER TABLE` jest wszechstronne i ma wiele zastosowań. We wcześniejszej części rozdziału przedstawiono już pewne możliwości wymienionego zapytania (zmiana silnika bazy danych oraz tworzenie i usuwanie indeksów). Zapytanie `ALTER TABLE` można również wykorzystać do zmiany nazw tabel, dodawania i usuwania kolumn, zmiany typu danych kolumn i wielu innych operacji. W tym punkcie zostaną przedstawione niektóre zastosowania wymienionego zapytania. Jego pełną składnię znajdziesz w dodatku E, zatytułowanym „Przewodnik po składni SQL”.

Zapytanie `ALTER TABLE` jest użyteczne, kiedy przekonasz się, że struktura tabeli nie odzwierciedla już planowanego sposobu jej użycia. Prawdopodobnie chcesz umieszczać w niej informacje dodatkowe lub tabela zawiera informacje, które stały się zbędne. Być może

istniejące kolumny są zbyt małe lub okazało się, że zdefiniowałeś kolumny znacznie większe, niż potrzeba, a teraz chcesz je zmniejszyć w celu zaoszczędzenia miejsca i poprawienia wydajności wykonywania zapytań. Poniżej zaprezentowano kilka sytuacji, w których użycie zapytania `ALTER TABLE` okaże się cenne:

- W projekcie badawczym rekordom kolejnych sprawdzanych przypadków przypisujesz liczby za pomocą kolumny `AUTO_INCREMENT`. Nie spodziewałeś się, że posiadane fundusze pozwolą na wygenerowanie ponad 50 000 rekordów i dlatego użyłeś typu danych `SMALLINT UNSIGNED`, który przechowuje maksymalnie 65 535 unikalnych wartości. Jednak fundusze projektu zostały ponownie zasilone i otrzymałeś możliwość wygenerowania kolejnych 50 000 rekordów. W takim przypadku potrzebujesz typu danych pozwalającego na przechowywanie większych liczb dla sprawdzanych przypadków.
- Wielkość może się zmienić w drugą stronę. Być może utworzyłeś kolumnę `CHAR(255)`, ale teraz przekonałeś się, że żadna wartość w tabeli nie jest większa niż 100 znaków. Możesz zmniejszyć kolumnę lub skonwertować ją na `VARCHAR(255)`, aby zaoszczędzić miejsce.
- Chcesz skonwertować tabelę na używającą innego silnika bazy danych i wykorzystać funkcje oferowane przez nowy silnik. Na przykład, tabele `MyISAM` nie zapewniają bezpieczeństwa transakcji, a aplikacja wymaga teraz obsługi transakcji. Odpowiednie tabele możesz więc skonwertować na używające silnika `InnoDB`, który obsługuje transakcje. Ewentualnie, używałeś `MyISAM` w `MySQL 5.5` ze względu na obsługę wyszukiwania pełnego tekstu, a teraz uaktualniłeś serwer do `MySQL 5.6`, który zapewnia obsługę wyszukiwania pełnego tekstu również w tabelach używających silnika `InnoDB`.

Składnia zapytania `ALTER TABLE` przedstawia się następująco:

```
ALTER TABLE nazwa_tabeli action [, akcja] ... ;
```

Każda akcja określa modyfikację wprowadzaną w tabeli. Niektóre systemy bazy danych pozwalają na użycie tylko jednej akcji w zapytaniu `ALTER TABLE`, ale `MySQL` obsługuje wiele akcji rozdzielonych przecinkami.

Wskazówka

Jeżeli przed użyciem zapytania `ALTER TABLE` chcesz przypomnieć sobie bieżącą definicję tabeli, wykonaj zapytanie `SHOW CREATE TABLE`. Wymienione zapytanie jest użyteczne także po wykonaniu `ALTER TABLE`, ponieważ pozwala zweryfikować, czy zmiana w oczekiwany sposób wpłynęła na definicję tabeli.

W poniższych przykładach przedstawiono niektóre zastosowania zapytania `ALTER TABLE`.

Zmiana typu danych kolumny. Aby zmienić typ danych, użyj klauzuli `CHANGE` lub `MODIFY`. Przyjmujemy założenie, że kolumna `i` w tabeli `mytbl1` jest typu `SMALLINT UNSIGNED`. W celu zmiany jej typu danych na `MEDIUMINT UNSIGNED` należy użyć dowolnego z poniższych zapytań:

```
ALTER TABLE mytbl MODIFY i MEDIUMINT UNSIGNED;
ALTER TABLE mytbl CHANGE i i MEDIUMINT UNSIGNED;
```

Dlaczego w zapytaniu używającym klauzuli `CHANGE` dwukrotnie podano nazwę kolumny? Ponieważ w przeciwieństwie do klauzuli `MODIFY`, `CHANGE` może również zmienić nazwę kolumny, a nie tylko jej typ danych. Jeżeli chcesz zmienić nazwę kolumny `i` na `k` w trakcie zmiany jej typu danych, wtedy możesz wykonać poniższe zapytanie:

```
ALTER TABLE mytbl CHANGE i k MEDIUMINT UNSIGNED;
```

Pamiętaj, że w klauzuli `CHANGE` najpierw podajesz nazwę zmienianej kolumny, a dopiero później jej nową nazwę i definicję. Aby pozostawić dotychczasową nazwę kolumny, musisz ją podać dwukrotnie.

W celu zmiany nazwy kolumny bez zmiany jej typu danych użyj klauzuli `CHANGE` *stara_nazwa* *nowa_nazwa* i bieżącej definicji kolumny.

Aby zmienić kodowanie znaków w kolumnie, użyj atrybutu `CHARACTER SET` w definicji kolumny:

```
ALTER TABLE t MODIFY c CHAR(20) CHARACTER SET ucs2;
```

Ważnym powodem zmiany typu danych jest poprawienie wydajności zapytania w złączeniach porównujących kolumny z dwóch tabel. Indeksy bardzo często mogą być używane do porównań w złączeniach między kolumnami podobnych typów, ale operacje porównania będą wykonywane szybciej, gdy obie kolumny są dokładnie tego samego typu. Przyjmujemy założenie, że są wykonywane zapytania podobne do poniższego:

```
SELECT ... FROM t1 INNER JOIN t2 WHERE t1.name = t2.name;
```

Jeżeli `t1.name` jest typu `CHAR(10)`, a `t2.name` typu `CHAR(15)`, wtedy zapytanie nie będzie wykonywane tak szybko, jakby obie kolumny były typu `CHAR(15)`. Kolumny mogą stać się takiego samego typu, jeśli `t1.name` zmienisz za pomocą jednego z poniższych zapytań:

```
ALTER TABLE t1 MODIFY name CHAR(15);
ALTER TABLE t1 CHANGE name name CHAR(15);
```

Konwersja tabeli na używającą innego silnika bazy danych. W celu konwersji tabeli z używającej jednego silnika bazy danych na inny należy użyć klauzuli `ENGINE` wskazującej nazwę nowego silnika:

```
ALTER TABLE nazwa_tabeli ENGINE=nazwa_silnika;
```

Specyfikator *nazwa_silnika* to nazwa taka jak `InnoDB`, `MyISAM` lub `MEMORY`. Wielkość liter nie ma tutaj znaczenia.

Jednym z powodów zmiany silnika używanego przez tabelę jest zapewnienie bezpieczeństwa transakcji. Przyjmujemy założenie, że w aplikacji jest używana tabela `MyISAM`. Następnie okazuje się, że ta aplikacja musi wykonywać operacje transakcyjne, między innymi wycofanie w przypadku niepowodzenia transakcji. Tabele `MyISAM` nie obsługują transakcji. Dlatego też bezpieczeństwo transakcji możesz zapewnić przez konwersję tabeli na używającą silnika `InnoDB`:

```
ALTER TABLE nazwa_tabeli ENGINE=InnoDB;
```


Kiedy konwertujesz tabelę na używającą innego silnika bazy danych, dozwolona lub sensowna konwersja może zależeć od zgodności funkcji oferowanych przez stary i nowy silnik. Na przykład, jeśli masz tabelę zawierającą kolumnę typu BLOB, to nie możesz jej skonwertować na tabelę używającą silnika MEMORY, ponieważ wymieniony silnik nie obsługuje kolumn typu BLOB.

Istnieją pewne sytuacje, w których nie powinieneś używać zapytania `ALTER TABLE` do konwersji tabeli na używającą innego silnika bazy danych, na przykład:

- Tabela InnoDB może być skonwertowana na używającą innego silnika bazy danych. Jednak jeśli tabela ma zdefiniowane ograniczenia klucza zewnętrznego, to zostaną one utracone, ponieważ tylko InnoDB obsługuje klucze zewnętrzne.
- Tabele MEMORY są przechowywane w pamięci i usuwane po zamknięciu serwera. Jeżeli wymagane jest, aby zawartość tabel pozostała między kolejnymi uruchomieniami serwera, to nie konwertuj ich na używające silnika MEMORY.

Zmiana nazwy tabeli. Użyj klauzuli `RENAME` zawierającej nową nazwę tabeli:

```
ALTER TABLE nazwa_tabeli RENAME TO nowa_nazwa_tabeli;
```

Innym sposobem zmiany nazwy tabeli jest użycie zapytania `RENAME TABLE`, którego składnia przedstawia się następująco:

```
RENAME TABLE nazwa_tabeli TO nowa_nazwa_tabeli;
```

W przeciwieństwie do zapytania `ALTER TABLE`, `RENAME TABLE` ma możliwość zmiany nazw wielu tabel w pojedynczym zapytaniu. Na przykład, masz możliwość zmiany nazw dwóch tabel w następujący sposób:

```
RENAME TABLE t1 TO tmp, t2 TO t1, tmp TO t2;
```

W przypadku użycia kwalifikowanej nazwy tabeli wraz z nazwą bazy danych, tabelę można przenieść z jednej bazy danych do innej przez zmianę jej nazwy. Wykonanie dowolnego z poniższych zapytań powoduje przeniesienie tabeli `t` z bazy danych `sampdb` do bazy danych `test`:

```
ALTER TABLE sampdb.t RENAME TO test.t;  
RENAME TABLE sampdb.t TO test.t;
```

Tabeli nie można zmienić nazwy na nazwę już istniejącej tabeli.

2.7. Pobieranie metadanych bazy danych

MySQL oferuje kilka sposobów pobierania metadanych bazy danych, czyli informacji o bazach danych i przechowywanych w nich obiektach:

- zapytania `SHOW`, takie jak `SHOW DATABASES` lub `SHOW TABLES`;
- tabele w bazie danych `INFORMATION_SCHEMA`;
- programy wiersza poleceń, takie jak `mysql show` lub `mysql dump`.

W poniższych punktach omówiono wykorzystanie wymienionych źródeł informacji w celu uzyskania dostępu do metadanych.

2.7.1. Pobieranie metadanych za pomocą zapytania SHOW

MySQL oferuje zapytanie SHOW, pozwalające na wyświetlanie wielu różnych typów metadanych bazy danych. Zapytanie SHOW jest użyteczne podczas sprawdzania zawartości baz danych i przypominania struktury tabel. Przedstawione poniżej przykłady pokazują kilka zastosowań zapytania SHOW.

Wyświetlenie listy baz danych, do których masz dostęp, następuje po wykonaniu następującego zapytania:

```
SHOW DATABASES;
```

Wyświetlenie zapytania CREATE DATABASE użytego do utworzenia bazy danych:

```
SHOW CREATE DATABASE nazwa_bazy_danych;
```

Wyświetlenie listy tabel w domyślnej lub wskazanej bazie danych:

```
SHOW TABLES;
SHOW TABLES FROM nazwa_bazy_danych;
```

Zapytanie SHOW TABLES nie wyświetla tabel tymczasowych.

Wyświetlenie zapytania CREATE TABLE użytego do utworzenia tabeli:

```
SHOW CREATE TABLE nazwa_tabeli;
```

Wyświetlenie informacji o kolumnach lub indeksach w tabeli:

```
SHOW COLUMNS FROM nazwa_tabeli;
SHOW INDEX FROM nazwa_tabeli;
```

Zapytania DESCRIBE *nazwa_tabeli* i EXPLAIN *nazwa_tabeli* są synonimami zapytania SHOW COLUMNS FROM *nazwa_tabeli*.

Wyświetlenie opisowych informacji o tabelach w domyślnej lub wskazanej bazie danych:

```
SHOW TABLE STATUS;
SHOW TABLE STATUS FROM nazwa_bazy_danych;
```

Kilka form zapytania SHOW pobiera klauzulę LIKE *'wzorzec'* powodującą ograniczenie wyświetlanych danych wyjściowych. Serwer MySQL interpretuje *'wzorzec'* jako wzorzec SQL, który może zawierać znaki wieloznaczne % i _. Na przykład, poniższe zapytanie wyświetla nazwy kolumn w tabeli student, których nazwa rozpoczyna się od litery s:

```
mysql> SHOW COLUMNS FROM student LIKE 's%';
```

Field	Type	Null	Key	Default	Extra
sex	enum('F','M')	NO		NULL	
student_id	int(10) unsigned	NO	PRI	NULL	auto_increment

Aby dopasować dosłowne wystąpienie znaku wieloznacznego we wzorcu LIKE, należy poprzedzić go ukośnikiem. Najczęściej dotyczy to dopasowania znaku podkreślenia, który często występuje w nazwach baz danych, tabel i kolumn.

Każde zapytanie SHOW obsługujące klauzulę LIKE może być również utworzone wraz z klauzulą WHERE. Zapytanie wyświetla te same kolumny, ale klauzula WHERE oferuje znacznie większą elastyczność w zakresie wskazania zwracanych rekordów. Klauzula WHERE powinna odwoływać się do nazw kolumn zapytania SHOW. Jeżeli nazwa kolumny jest słowem zarezerwowanym, takim jak KEY, wtedy trzeba ją podać jako cytowany identyfikator. Poniższe zapytanie informuje, która kolumna w tabeli student jest kluczem podstawowym:

```
mysql> SHOW COLUMNS FROM student WHERE `Key` = `PRI`;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| student_id | int(10) unsigned | NO | PRI | NULL | auto_increment |
+-----+-----+-----+-----+-----+-----+
```

Czasami użyteczne jest sprawdzenie z poziomu aplikacji, czy dana tabela istnieje. Do tego celu można użyć zapytania SHOW TABLES (o ile tabela nie jest tabelą tymczasową):

```
SHOW TABLES LIKE 'nazwa_tabeli';
SHOW TABLES FROM nazwa_bazy_danych LIKE 'nazwa_tabeli';
```

Jeżeli zapytanie SHOW TABLES wyświetla informacje o wskazanej tabeli, ona na pewno istnieje. Inną możliwością ustalenia istnienia tabeli (nawet tymczasowej) jest użycie jednego z poniższych zapytań:

```
SELECT COUNT(*) FROM nazwa_tabeli;
SELECT * FROM nazwa_tabeli WHERE FALSE;
```

Każde z powyższych zapytań zakończy działanie powodzeniem, jeśli tabela istnieje, w przeciwnym razie działanie zakończy się niepowodzeniem. Pierwsze zapytanie jest najbardziej odpowiednie dla tabel MyISAM, dla których COUNT(*) bez klauzuli WHERE jest wysoce zoptymalizowane. Nie nadaje się zbyt dobrze dla tabel InnoDB, wymagających przeprowadzenia pełnego skanowania w celu zliczenia rekordów. Z kolei drugie zapytanie jest znacznie ogólniejsze, ponieważ jest wykonywane szybko w każdym silniku bazy danych. Wymienione zapytania są przygotowane do użycia w aplikacjach utworzonych w językach takich jak Perl i PHP, ponieważ możesz wówczas sprawdzić wynik zapytania i podjąć odpowiednie działania. Nie są zbyt użyteczne w skryptach wsadowych uruchamianych z poziomu programu mysql, ponieważ po wystąpieniu błędu nie możesz nic zrobić poza zakończeniem skryptu (lub zignorowaniem błędu, ale wówczas nie ma żadnego powodu wykonywania tego rodzaju zapytania). Inną strategią, która sprawdza się w dowolnym kontekście, jest wykonanie zapytania do bazy danych INFORMATION_SCHEMA. Zapoznaj się z kolejnym punktem 2.7.2, zatytułowanym „Pobieranie metadanych z bazy danych INFORMATION_SCHEMA”.

Aby ustalić silnik bazy danych używany przez poszczególne tabele, możesz użyć zapytania SHOW TABLE STATUS lub SHOW CREATE TABLE. Dane wyjściowe wygenerowane przez wymienione zapytania zawierają informacje o używanym silniku bazy danych.

2.7.2. Pobieranie metadanych z bazy danych INFORMATION_SCHEMA

Innym sposobem pobrania informacji o bazach danych jest uzyskanie dostępu do bazy danych INFORMATION_SCHEMA. Wymieniona baza danych jest oparta na standardzie SQL. Oznacza to, że mechanizm dostępu jest standardowy, nawet jeśli jej pewna zawartość pozostaje charakterystyczna dla MySQL. W ten sposób baza danych INFORMATION_SCHEMA jest bardziej przenośna niż różne zapytania SHOW, które w całości są charakterystyczne dla MySQL.

Dostęp do bazy danych INFORMATION_SCHEMA odbywa się za pomocą zapytań SELECT i może być bardzo elastyczny. Zapytania SHOW zawsze wyświetlają ustalony zestaw kolumn, a ich danych wyjściowych nie można umieścić w tabeli. W przypadku bazy danych INFORMATION_SCHEMA zapytanie SELECT może wskazywać kolumny, które mają pojawić się w danych wyjściowych, natomiast klauzula WHERE umożliwia zdefiniowanie wyrażenia wymaganego do wybrania żądanych informacji. Ponadto, z powodu możliwości wykorzystania złączeń i podzapytań możesz użyć zapytania CREATE TABLE ... SELECT lub INSERT INTO ... SELECT, aby zapisać wynik zapytania w innej tabeli w celu jego dalszego przetwarzania.

Bazę danych INFORMATION_SCHEMA możesz potraktować jako wirtualną bazę danych, w której tabelę są widokami dla różnego rodzaju metadanych bazy danych. Aby przekonać się, jakie tabelę zawiera baza danych INFORMATION_SCHEMA, wykonaj zapytanie SHOW TABLES:

```
mysql> SHOW TABLES IN INFORMATION_SCHEMA;
```

Tables_in_information_schema
CHARACTER_SETS
COLLATIONS
COLLATION_CHARACTER_SET_APPLICABILITY
COLUMNS
COLUMN_PRIVILEGES
ENGINES
EVENTS
FILES
GLOBAL_STATUS
GLOBAL_VARIABLES
KEY_COLUMN_USAGE
PARAMETERS
PARTITIONS
PLUGINS
PROCESSLIST
PROFILING
REFERENTIAL_CONSTRAINTS
ROUTINES
SCHEMATA
SCHEMA_PRIVILEGES
SESSION_STATUS
SESSION_VARIABLES
STATISTICS
TABLES
TABLESPACES
TABLE_CONSTRAINTS

	TABLE_PRIVILEGES	
	TRIGGERS	
	USER_PRIVILEGES	
	VIEWS	
+-----+-----+		

Poniższa lista pokrótce opisuje niektóre z wymienionych powyżej tabel bazy danych INFORMATION_SCHEMA:

- SCHEMATA, TABLES, VIEWS, ROUTINES, TRIGGERS, EVENTS, PARAMETERS, PARTITIONS, COLUMNS
Informacje o bazach danych, tabelach, widokach, procedurach składowych, wyzwalaczach i zdarzeniach w bazach danych, a także o parametrach składowanych, partycjach tabel i kolumnach w tabelach.
- FILES
Informacje o plikach używanych do przechowywania danych przestrzeni tabel.
- TABLE_CONSTRAINTS, KEY_COLUMN_USAGE
Informacje o tabelach i kolumnach, dla których zdefiniowano ograniczenia, takie jak indeksy o unikalnych wartościach lub klucze zewnętrzne.
- STATISTICS
Informacje o cechach charakterystycznych indeksów tabeli.
- REFERENTIAL_CONSTRAINTS
Informacje o kluczach zewnętrznych.
- CHARACTER_SETS, COLLATIONS, COLLATION_CHARACTER_SET_APPLICABILITY
Informacje o obsługiwanych kodowaniach znaków, kolejności sortowania w danym kodowaniu znaków, a także o mapowaniu z każdej kolejności sortowania na jego kodowanie znaków.
- ENGINES, PLUGINS
Informacje o silnikach bazy danych i wtyczkach serwera.
- USER_PRIVILEGES, SCHEMA_PRIVILEGES, TABLE_PRIVILEGES, COLUMN_PRIVILEGES
Informacje o uprawnieniach globalnych, bazy danych, tabeli i kolumny pochodzące z tabel user, db, tables_priv i columns_priv w bazie danych mysql.
- GLOBAL_VARIABLES, SESSION_VARIABLES, GLOBAL_STATUS, SESSION_STATUS
Wartości zmiennych systemowych i stanu, zarówno globalne, jak i sesji.
- PROCESSLIST
Informacje o wątkach działających w serwerze.

Poszczególne silniki bazy danych mogą dodawać własne tabele do bazy danych INFORMATION_SCHEMA. Na przykład, silnik InnoDB dodaje pewne tabele.

W celu ustalenia kolumn znajdujących się w tabeli bazy danych INFORMATION_SCHEMA użyj zapytania SHOW COLUMNS lub DESCRIBE:

```
mysql> DESCRIBE INFORMATION_SCHEMA.CHARACTER_SETS;
```

Field	Type	Null	Key	Default	Extra
CHARACTER_SET_NAME	varchar(32)	NO			
DEFAULT_COLLATE_NAME	varchar(32)	NO			
DESCRIPTION	varchar(60)	NO			
MAXLEN	bigint(3)	NO		0	

W celu wyświetlenia informacji z tabeli użyj zapytania SELECT. (Ani baza danych INFORMATION_SCHEMA, ani żadna z jej tabel lub kolumn nie rozróżnia wielkości liter). Ogólne zapytanie wyświetlające wszystkie kolumny ze wskazanej tabeli bazy danych INFORMATION_SCHEMA przedstawia się następująco:

```
SELECT * FROM INFORMATION_SCHEMA.nazwa_tabeli;
```

Dodanie klauzuli WHERE pozwala na dokładne wskazanie informacji, które chcesz wyświetlić.

W poprzednim punkcie omówiono wykorzystanie zapytań SHOW w celu ustalenia istnienia tabeli lub używanego przez nią silnika bazy danych. Tabele bazy danych INFORMATION_SCHEMA mogą dostarczyć tych samych informacji. Poniższe zapytanie używa bazy danych INFORMATION_SCHEMA do sprawdzenia istnienia wskazanej tabeli i zwraca wartość 1 lub 0, wskazującą odpowiednio istnienie lub brak danej tabeli:

```
mysql> SELECT COUNT(*) FROM INFORMATION_SCHEMA.TABLES
-> WHERE TABLE_SCHEMA='sampdb' AND TABLE_NAME='member';
```

COUNT(*)
1

Poniższe zapytanie pozwala sprawdzić, który silnik bazy danych jest używany przez wskazaną tabelę:

```
mysql> SELECT ENGINE FROM INFORMATION_SCHEMA.TABLES
-> WHERE TABLE_SCHEMA='sampdb' AND TABLE_NAME='student';
```

ENGINE
InnoDB

2.7.3. Pobieranie metadanych z poziomu wiersza poleceń

Narzędzie mysql show dostarcza niektóre z informacji wyświetlanych przez pewne zapytania SHOW. W ten sposób informacje o bazie danych i tabeli możesz pobrać bezpośrednio z wiersza zapytań.

Wyświetlenie listy baz danych zarządzanych przez serwer:

```
% mysqlshow
```

Wyświetlenie tabel znajdujących się w bazie danych:

```
% mysqlshow nazwa_bazy_danych
```

Wyświetlenie informacji o kolumnach tabeli:

```
% mysqlshow nazwa_bazy_danych nazwa_tabeli
```

Wyświetlenie informacji o indeksach tabeli:

```
% mysqlshow --keys nazwa_bazy_danych nazwa_tabeli
```

Wyświetlenie opisowych informacji o tabelach w bazie danych:

```
% mysqlshow --status nazwa_bazy_danych
```

Program kliencki mysqldump pozwala na poznanie struktury tabel w postaci zapytań CREATE TABLE (czyli podobnie do SHOW CREATE TABLE). Jeżeli chcesz użyć mysqldump do przejrzenia struktury tabeli, wywołaj je wraz z opcją --no-data, aby nie uszkodzić danych tabeli!

```
% mysqldump --no-data nazwa_bazy_danych [nazwa_tabeli] ...
```

Jeżeli podasz jedynie nazwę bazy danych i pominiesz nazwy tabel, polecenie mysqldump wyświetli strukturę wszystkich tabel w bazie danych. W przeciwnym razie wyświetlone zostaną informacje o wskazanych tabelach.

W przypadku zarówno mysqlshow, jak i mysqldump, o ile to konieczne, istnieje możliwość podania opcji parametru połączenia, na przykład --host, --user lub --password.

2.8. Pobieranie danych z wielu tabel za pomocą złączeń

Umieszczenie rekordów w bazie danych nie przynosi pożytku, jeśli później nie będą one pobierane. Do tego właśnie służy zapytanie SELECT: pomaga w pobraniu danych. Zapytanie SELECT jest prawdopodobnie używane znacznie częściej niż jakiegokolwiek inne zapytanie w języku SQL. Jednocześnie może być trudne w użyciu, ponieważ zdefiniowane warunki do pobierania rekordów mogą być dowolnie skomplikowane i obejmować operacje porównań kolumn znajdujących się w wielu tabelach.

Podstawowa składnia zapytania SELECT przedstawia się następująco:

```
SELECT lista_kolumn          # Wybrane kolumny.
FROM lista_tabel             # Tabele zawierające wybrane rekordy.
WHERE ograniczenie_rekordu   # Warunki, które muszą spełnić rekordy.
GROUP BY kolumny_grupowania  # Sposób grupowania wyniku.
ORDER BY kolumny_sortowania  # Sposób sortowania wyniku.
HAVING ograniczenie_grupy    # Warunki, które muszą spełnić grupy.
LIMIT liczba;               # Ograniczenie liczby rekordów w wyniku.
```

Wszystko w powyższej składni jest opcjonalne poza słowem SELECT i *lista_kolumn* wskazującą informacje, które powinny znaleźć się w danych wyjściowych. Niektóre bazy

danym wymagają również klauzuli FROM. MySQL nie wymaga podania klauzuli FROM, co pozwala na obliczenie wartości wyrażenia bez odwoływania się do jakiegokolwiek kolumny:

```
SELECT SQRT(POW(3,2)+POW(4,2));
```

W rozdziale 1., zatytułowanym „Rozpoczęcie pracy z MySQL”, dość dużą ilość miejsca poświęcono zapytaniom SELECT pobierającym dane z pojedynczej tabeli. Koncentrowaliśmy się przede wszystkim na liście kolumn danych wyjściowych i klauzulach WHERE, GROUP BY, ORDER BY, HAVING i LIMIT. W tym miejscu zostaną przedstawione aspekty zapytania SELECT sprawiające najwięcej problemów, czyli tworzenie złączeń. Wspomniane złączenia pozwalają zapytaniom SELECT na pobieranie rekordów z wielu tabel. Omówione zostaną typy złączeń obsługiwane przez MySQL, sposób ich działania oraz używania. Dzięki temu będziesz potrafił znacznie efektywniej wykorzystać MySQL, ponieważ w wielu przypadkach prawdziwym problemem w utworzeniu zapytania jest ustalenie poprawnego sposobu złączenia tabel.

Jednym z problemów podczas używania zapytania SELECT jest to, że po napotkaniu po raz pierwszy nowego typu problemu nie zawsze wiadomo, jak utworzyć odpowiednie zapytanie SELECT w celu jego rozwiązania. Jednak po rozwiązaniu problemu zdobyte doświadczenie można wykorzystać w przyszłości po napotkaniu podobnych problemów. W przypadku zapytania SELECT wcześniejsze doświadczenie odgrywa ogromną rolę w możliwości jego efektywnego używania. Wynika to z dużej liczby problemów, do rozwiązania których można je zastosować. Kiedy zdobędziesz doświadczenie, złączenia będziesz bardzo łatwo potrafił zaadaptować do nowych problemów i przekonasz się, że myślisz w stylu „och tak, to jedno z tych złączeń LEFT JOIN” lub „aha, to trzykierunkowe złączenie ograniczone przez wspólną parę kolumna kluczy”. (Na pewno za zachęcające uznasz, że doświadczenie okazuje się pomocne. Ewentualnie możesz być przerażony, że mógłbyś myśleć w kategoriach takich, jak przedstawiono powyżej).

Wiele przykładów demonstrujących, jak używać form operacji złączeń obsługiwanych przez MySQL, posługuje się poniższymi dwiema tabelami o nazwach t1 i t2:

Tabela t1:	Tabela t2:
+---+---+	+---+---+
i1 c1	i2 c2
+---+---+	+---+---+
1 a	2 c
2 b	3 b
3 c	4 a
+---+---+	+---+---+

Powyższe tabele są na tyle małe, że efekt każdego rodzaju złączenia można łatwo zobaczyć.

Innym typem zapytań SELECT obejmujących wiele tabel są podzapytania (to znaczy jedno zapytanie SELECT zagnieżdżone w innym) oraz zapytania UNION. Zostaną one omówione w podrozdziałach 2.9, „Pobieranie informacji z wielu tabel za pomocą podzapytań”, i 2.10, „Pobieranie informacji z wielu tabel za pomocą zapytań UNION”.

Obsługiwaną przez MySQL funkcją powiązaną z wieloma tabelami jest możliwość usuwania lub uaktualniania rekordów w jednej tabeli na podstawie zawartości innej.

Na przykład, w jednej tabeli możesz chcieć usunąć rekordy niedopasowane do rekordów znajdujących się w innej tabeli lub skopiować wartości z kolumn w jednej tabeli do kolumn w innej. Tego rodzaju operacje zostaną omówione w podrozdziale 2.11, zatytułowanym „Usuwanie i uaktualnianie rekordów w wielu tabelach”.

2.8.1. Złączenia wewnętrzne

Jeżeli zapytanie SELECT wymienia wiele nazw tabel w klauzuli FROM, a ich nazwy są rozdzielone INNER JOIN, wtedy MySQL przeprowadza złączenie wewnętrzne, które generuje wynik przez dopasowanie rekordów w jednej tabeli z rekordami innej. Na przykład, jeśli zastosujesz złączenie tabel t1 i t2 w poniższy sposób, każdy rekord z tabeli t1 będzie połączony z rekordem tabeli t2:

```
mysql> SELECT * FROM t1 INNER JOIN t2;
```

i1	c1	i2	c2
1	a	2	c
2	b	2	c
3	c	2	c
1	a	3	b
2	b	3	b
3	c	3	b
1	a	4	a
2	b	4	a
3	c	4	a

W przedstawionym powyżej zapytaniu SELECT * oznacza „wybierz wszystkie kolumny z wszystkich tabel wymienionych w klauzuli FROM”. To zapytanie mógłbyś również zapisać w postaci SELECT t1.*, t2.*:

```
SELECT t1.*, t2.* FROM t1 INNER JOIN t2;
```

Jeżeli nie chcesz wybrać wszystkich kolumn lub chcesz je wyświetlić w innej kolejności od lewej do prawej strony, wówczas wymień kolumny, rozdzielając je przecinkami.

Złączenie łączy każdy rekord wszystkich kolumn z każdym rekordem w pozostałych kolumnach w celu wygenerowania wszystkich możliwych kombinacji nazywanych „produktem kartezjańskim”. Złączenie tabel w taki sposób może doprowadzić do utworzenia naprawdę ogromnej liczby rekordów. Dlatego też złączenie trzech tabel zawierających odpowiednio 100, 200 i 300 rekordów może zwrócić $100 \times 200 \times 300 = 6\,000\,000$ rekordów. To jest ogromna liczba rekordów pomimo tego, że poszczególne tabele są małe. W takich przypadkach zwykła klauzula WHERE jest bardzo użyteczna i pozwala na zmniejszenie zbioru wynikowego do rozmiarów znacznie łatwiejszych w zarządzaniu.

Jeżeli dodasz klauzulę WHERE, powodującą, że tabele będą dopasowywały wartości do wskazanych kolumn, złączenie wybierze jedynie rekordy o identycznych wartościach w podanych kolumnach:

```
mysql> SELECT t1.*, t2.* FROM t1 INNER JOIN t2 WHERE t1.i1 = t2.i2;
+-----+-----+
| i1 | c1 | i2 | c2 |
+-----+-----+
| 2 | b | 2 | c |
| 3 | c | 3 | b |
+-----+-----+
```

Złączenia typu `CROSS JOIN` i `JOIN` są takie same jak `INNER JOIN`, a więc poniższe zapytania mają identyczne działanie:

```
SELECT t1.*, t2.* FROM t1 INNER JOIN t2 WHERE t1.i1 = t2.i2;
SELECT t1.*, t2.* FROM t1 CROSS JOIN t2 WHERE t1.i1 = t2.i2;
SELECT t1.*, t2.* FROM t1 JOIN t2 WHERE t1.i1 = t2.i2;
```

Operator złączenia to przecinek (,):

```
SELECT t1.*, t2.* FROM t1, t2 WHERE t1.i1 = t2.i2;
```

Jednak operator `,` ma inne pierwszeństwo niż pozostałe typy złączeń i czasami może doprowadzić do powstania błędów składni w sytuacjach, w których inne typy złączeń błędów nie generują. Dlatego też zalecam unikanie operatora w postaci przecinka.

Złączenia `INNER JOIN`, `CROSS JOIN` i `JOIN` (ale nie operator w postaci przecinka) obsługują także alternatywne składnie pozwalające na określenie sposobu dopasowania kolumn tabel:

- Jedna ze składni używa klauzuli `ON` zamiast `WHERE`. Poniższy przykład pokazuje użycie złączenia typu `INNER JOIN`:

```
SELECT t1.*, t2.* FROM t1 INNER JOIN t2 ON t1.i1 = t2.i2;
```

Klauzula `ON` może być użyta niezależnie od tego, czy złączane kolumny mają takie same nazwy.

- Kolejna składnia opiera się na klauzuli `USING()`. Koncepcja jest podobna do stosowanej przez klauzulę `ON`, ale nazwa złączonej kolumny lub kolumn musi być taka sama w każdej tabeli. Na przykład, poniższe zapytanie łączy tabele `mytb11.b` i `mytb12.b`:

```
SELECT mytb11.*, mytb12.* FROM mytb11 INNER JOIN mytb12 USING (b);
```

2.8.2. Kwalifikowane odwołania do kolumn z poziomu złączonych tabel

Odwołania do poszczególnych kolumn tabel w zapytaniu `SELECT` muszą jednoznacznie prowadzić do pojedynczej tabeli wymienionej w klauzuli `FROM`. Jeżeli podana została tylko jedna tabela, wówczas nie ma problemu i wszystkie kolumny muszą znajdować się w tej tabeli. Natomiast w przypadku podania wielu tabel w klauzuli `FROM`, jeśli dana nazwa kolumny występuje tylko w jednej tabeli, ponownie nie ma żadnego problemu. Jednak gdy kolumny w różnych tabelach mają takie same nazwy, odniesienia do nich muszą być kwalifikowane i zawierać identyfikator tabeli: *nazwa_tabeli.nazwa_kolumny*. W ten sposób dokładnie wskazujesz interesującą Cię tabelę i kolumnę. Przyjmujemy założenie, że tabela `mytb11` ma kolumny o nazwach `a` i `b`, natomiast tabela `mytb12` zawiera kolumny `b` i `c`.

Odwołania do kolumn a i c są jednoznaczne, natomiast odwołania do kolumn b muszą być kwalifikowane w postaci mytb1.b i mytb2.b:

```
SELECT a, mytb1.b, mytb2.b, c FROM mytb1 INNER JOIN mytb2 ... ;
```

Czasami kwalifikator nazwy tabeli nie jest wystarczający do jednoznacznego wskazania kolumny. Na przykład, w przypadku złączenia tabeli z samą sobą będzie ona wielokrotnie używana w zapytaniu i kwalifikowanie nazwy kolumny nie będzie wystarczające. W takiej sytuacji użyteczne jest wykorzystanie aliasów. Wspomniany alias można przypisać dowolnemu egzemplarzowi tabeli, a następnie odwoływać się do kolumn tego egzemplarza w postaci *nazwa_alias.nazwa_kolumny*. Poniższe zapytanie powoduje złączenie tabeli z samą sobą, ale jednemu egzemplarzowi tabeli przypisuje alias, co pozwala na jednoznaczne odwoływanie się do kolumn:

```
SELECT mytb1.col1, m.col2 FROM mytb1 INNER JOIN mytb1 AS m
WHERE mytb1.col1 > m.col1;
```

2.8.3. Złączenia typu LEFT i RIGHT (OUTER)

Złączenie typu INNER JOIN wyświetla jedynie rekordy, które zostały dopasowane w obu tabelach. Z kolei złączenie typu OUTER JOIN również wyświetla dopasowania, ale może także pokazać rekordy jednej tabeli, które nie mają dopasowania w drugiej. Dwa rodzaje złączeń typu OUTER to LEFT i RIGHT. Większość przedstawionych tutaj przykładów używa złączenia LEFT JOIN, identyfikującego rekordy w lewej tabeli, które nie zostały dopasowane w prawej. Złączenie typu RIGHT JOIN działa tak samo, ale odwracają się role tabel.

Złączenie LEFT JOIN działa w następujący sposób: wskazujesz kolumny, które będą porównywane w dwóch tabelach. Kiedy rekord z lewej tabeli zostanie dopasowany do rekordu w prawej tabeli, zawartość obu zostanie pobrana i umieszczona w rekordzie danych wyjściowych. Natomiast jeśli rekord w lewej tabeli nie ma dopasowania w prawej, nadal będzie wybrany do danych wyjściowych, ale złączony z „fałszywym” rekordem z prawej tabeli, zawierającym NULL we wszystkich kolumnach.

Innymi słowy, złączenie LEFT JOIN wymusza, aby zbiór wynikowy zawierał wszystkie rekordy z lewej tabeli, niezależnie od znalezienia dla nich dopasowania w prawej tabeli. Pozbawione dopasowania rekordy w lewej tabeli mogą być zidentyfikowane przez fakt, że wszystkie kolumny z prawej tabeli mają wartość NULL. Taki zestaw rekordów wynikowych wskazuje rekordy nieistniejące w prawej tabeli. To jest interesująca i ważna właściwość, ponieważ tego rodzaju problem pojawia się w wielu różnych kontekstach. Któremu klientowi nie został przypisany osobisty doradca? Które produkty w ogóle się nie sprzedają? W przypadku omawianej w książce bazy danych sampdb to mogą być pytania: którzy uczniowie nie uczestniczyli w danym egzaminie? Dla których uczniów nie ma rekordów w tabeli absence (to znaczy byli obecni na wszystkich zajęciach)?

Ponownie powróćmy do dwóch tabel t1 i t2:

Table t1:	Table t2:
+-----+	+-----+
i1 c1	i2 c2

1	a	2	c
2	b	3	b
3	c	4	a

Jeżeli użyjemy złączenia typu `INNER JOIN` w celu dopasowania tabel `t1.i1` i `t2.i2`, wtedy dane wyjściowe będą zawierały jedynie wartości 2 i 3, ponieważ pojawiają się one w obu tabelach:

```
mysql> SELECT t1.*, t2.* FROM t1 INNER JOIN t2 ON t1.i1 = t2.i2;
```

i1	c1	i2	c2
2	b	2	c
3	c	3	b

Złączenie typu `LEFT JOIN` powoduje wygenerowanie danych wyjściowych dla każdego rekordu w `t1`, niezależnie od znalezienia dla niego dopasowania w tabeli `t2`. W celu utworzenia tego typu zapytania wymien nazwy tabel i między nimi umieść `LEFT JOIN` zamiast `INNER JOIN`:

```
mysql> SELECT t1.*, t2.* FROM t1 LEFT JOIN t2 ON t1.i1 = t2.i2;
```

i1	c1	i2	c2
1	a	NULL	NULL
2	b	2	c
3	c	3	b

Dane wyjściowe zawierają teraz rekord dla wartości 1 w `t1.i1` pomimo braku dopasowania w `t2`. Wszystkie kolumny w tym rekordzie odpowiadające kolumnom `t2` mają wartość `NULL`.

W przypadku złączenia typu `LEFT JOIN` trzeba pamiętać o jednym: jeśli kolumny w prawej tabeli nie zostaną zdefiniowane jako `NOT NULL`, możesz mieć problemy z rekordami w zbiorze wynikowym. Na przykład, jeśli prawa tabela zawiera kolumny z wartościami `NULL`, wtedy nie będziesz w stanie odróżnić wartości `NULL` od wartości `NULL` oznaczających niedopasowane rekordy.

Jak już wcześniej wspomniano, złączenie typu `RIGHT JOIN` działa tak samo jak `LEFT JOIN`, ale role tabel są odwrócone. Dwa przedstawione poniżej zapytania mają identyczne działanie:

```
SELECT t1.*, t2.* FROM t1 LEFT JOIN t2 ON t1.i1 = t2.i2;
SELECT t1.*, t2.* FROM t2 RIGHT JOIN t1 ON t1.i1 = t2.i2;
```

W dalszym omówieniu odwołujemy się do złączenia typu `LEFT JOIN`. Aby informacje dopasować do złączenia typu `RIGHT JOIN`, wystarczy po prostu odwrócić role tabel.

Złączenie typu `LEFT JOIN` jest szczególnie użyteczne, gdy chcesz znaleźć *jedynie* te rekordy w lewej tabeli, które nie zostały dopasowane do prawej. W tym celu należy dodać

klauzulę WHERE powodującą wybór tylko rekordów zawierających wartości NULL w prawej tabeli. Innymi słowy, rekordy w jednej tabeli nieistniejące w drugiej:

```
mysql> SELECT t1.*, t2.* FROM t1 LEFT JOIN t2 ON t1.i1 = t2.i2
-> WHERE t2.i2 IS NULL;
+-----+
| i1 | c1 | i2 | c2 |
+-----+
| 1 | a | NULL | NULL |
+-----+
```

Normalnie podczas tworzenia tego typu zapytania najczęściej jesteś zainteresowany niedopasowanymi wartościami w lewej tabeli. Kolumny NULL z prawej tabeli nie są zbyt interesujące z punktu widzenia wyświetlanych wyników, więc można je pominąć w danych wyjściowych:

```
mysql> SELECT t1.* FROM t1 LEFT JOIN t2 ON t1.i1 = t2.i2
-> WHERE t2.i2 IS NULL;
+-----+
| i1 | c1 |
+-----+
| 1 | a |
+-----+
```

Podobnie jak INNER JOIN, złączenie LEFT JOIN może zostać utworzone na podstawie klauzul ON lub USING() w celu wskazania warunków dopasowania. Podobnie jak w przypadku INNER JOIN, klauzula ON może być używana niezależnie od tego, czy kolumny w obu tabelach mają takie same nazwy, natomiast klauzula USING() wymaga, aby obie tabele miały taką samą nazwę.

Złączenie NATURAL LEFT JOIN jest podobne do LEFT JOIN; przeprowadza złączenie LEFT JOIN, dopasowując wszystkie kolumny, które mają takie same nazwy w lewej i prawej tabeli. (Dlatego też nie jest stosowana klauzula ON lub USING).

Jak już wcześniej wspomniano, złączenie LEFT JOIN jest użyteczne podczas udzielania odpowiedzi na pytanie: których wartości brakuje? Zastosujmy to względem tabel w bazie danych sampdb i przeanalizujmy znacznie bardziej skomplikowany przykład niż przedstawiony wcześniej z użyciem tabel t1 i t2.

W projekcie ocen uczniów przedstawionym w rozdziale 1. mamy tabelę student przechowującą listę uczniów, tabelę grade_event wskazującą oceniane zdarzenie oraz tabelę score zawierającą wyniki osiągane przez uczniów w poszczególnych zdarzeniach. Jednak jeśli uczeń był chory w dniu sprawdzianu lub testu, wtedy tabela nie będzie zawierała żadnych wyników dla ucznia dla tego konkretnego zdarzenia. W takich przypadkach powinien odbyć się dodatkowy sprawdzian lub test dla danego ucznia, ale jak wyszukiwać brakujące rekordy?

Problem polega na ustaleniu uczniów, którzy nie mają wyników z danego zdarzenia. Dlatego też trzeba wyszukiwać brakujące w tabeli score kombinacje uczniów i ocenianych zdarzeń. Słowo „brakujące” jest wskazówką, że należy użyć złączenia typu LEFT JOIN. Samo złączenie jednak nie jest takie proste jak w poprzednich przykładach.

Nie szukamy jedynie wartości nieistniejących w pojedynczej kolumnie, ale w dwóch. Interesują nas połączenia uczeń-zdarzenie. Tego rodzaju kombinacje można wygenerować przez złączenie tabel `student` i `grade_event`:

```
FROM student INNER JOIN grade_event
```

Następnie na wyniku powyższego złączenia przeprowadzamy złączenie typu `LEFT JOIN` z tabelą `score`, aby znaleźć dopasowane pary identyfikator ucznia-identyfikator zdarzenia:

```
FROM student INNER JOIN grade_event
  LEFT JOIN score ON student.student_id = score.student_id
                  AND grade_event.event_id = score.event_id
```

Zwróć uwagę, że klauzula `ON` powoduje złączenie rekordów tabeli `score` ze złączonymi w innych tabelach w zupełnie innej operacji złączenia. To jest klucz do rozwiązania przedstawionego tutaj problemu. Złączenie typu `LEFT JOIN` wymusza wygenerowanie rekordu dla każdego rekordu wybranego przez złączenie tabel `student` i `grade_event`, nawet jeśli nie odpowiada mu rekord w tabeli `score`. Brakujące rekordy wyników w otrzymanych danych wyjściowych można szybko zidentyfikować, ponieważ kolumna `score` będzie miała wartość `NULL`. Wspomniane rekordy można zidentyfikować za pomocą warunku w klauzuli `WHERE`. Użyć można dowolnej kolumny z tabeli `score`, ale ponieważ szukamy brakujących wyników, to prawdopodobnie najlepszym rozwiązaniem jest sprawdzenie kolumny `score`:

```
WHERE score.score IS NULL
```

Wynik można posortować, dodając klauzulę `ORDER BY`. Dwie najlogiczniejsze kolejności sortowania to względem uczniów lub zdarzeń. W omawianym przykładzie decydujemy się na wariant pierwszy:

```
ORDER BY student.student_id, grade_event.event_id
```

W tym momencie trzeba jeszcze wskazać kolumny, które mają zostać umieszczone w danych wyjściowych. Oto ostateczna wersja zapytania:

```
SELECT
  student.name, student.student_id,
  grade_event.date, grade_event.event_id, grade_event.category
FROM
  student INNER JOIN grade_event
    LEFT JOIN score ON student.student_id = score.student_id
                  AND grade_event.event_id = score.event_id
WHERE
  score.score IS NULL
ORDER BY
  student.student_id, grade_event.event_id;
```

Jego wykonanie powoduje wygenerowanie następujących danych wyjściowych:

name	student_id	date	event_id	category
Megan	1	2012-09-16	4	Q

Joseph	2	2012-09-03	1	Q
Katie	4	2012-09-23	5	Q
Devri	13	2012-09-03	1	Q
Devri	13	2012-10-01	6	T
Will	17	2012-09-16	4	Q
Avery	20	2012-09-06	2	Q
Gregory	23	2012-10-01	6	T
Sarah	24	2012-09-23	5	Q
Carter	27	2012-09-16	4	Q
Carter	27	2012-09-23	5	Q
Gabrielle	29	2012-09-16	4	Q
Grace	30	2012-09-23	5	Q

Warto w tym miejscu pamiętać o jednym. Dane wyjściowe wyświetlają identyfikatory uczniów i zdarzeń. Kolumna `student_id` występuje w tabelach `student` i `score`, więc w pierwszej chwili możesz uznać za stosowne użycie `student.student_id` i `score.student_id`. Niestety, nie można zastosować takiego rozwiązania, ponieważ podstawą do znalezienia interesujących nas rekordów jest to, aby kolumny z tabeli `score` zostały przez złączenie `LEFT JOIN` zwrócone jako `NULL`. Użycie `score.student_id` spowoduje wygenerowanie w danych wyjściowych kolumny wartości `NULL`. To samo dotyczy wyświetlenia kolumny `event_id`. Występuje ona w tabelach `grade_event` i `score`, ale zapytanie wybiera `grade_event.event_id`, ponieważ wartości `score.event_id` zawsze będą `NULL`.

2.9. Pobieranie informacji z wielu tabel za pomocą podzapytań

Podzapytanie to zapytanie `SELECT` umieszczone w nawiasie i zagnieżdżone w innym zapytaniu `SELECT`. Poniżej przedstawiono przykład zapytania wyszukującego identyfikatory dla rekordów zdarzeń odpowiadających testom ('T') i używającego ich do pobrania wyników tych testów:

```
SELECT * FROM score
WHERE event_id IN (SELECT event_id FROM grade_event WHERE category = 'T');
```

Podzapytania mogą zwracać różne typy informacji:

- Podzapytanie skalarne zwraca pojedynczą wartość.
- Podzapytanie kolumny zwraca pojedynczą kolumnę jednej lub więcej wartości.
- Podzapytanie rekordu zwraca pojedynczy rekord jednej lub więcej wartości.
- Podzapytanie tabeli zwraca tabelę jednego lub więcej rekordów jednej lub więcej kolumn.

Wynik podzapytania można sprawdzić na wiele sposobów:

- Wynik podzapytania skalarnego może być oceniony przy użyciu względnych operatora porównania takiego jak `=` lub `<`.

- Operatory `IN` i `NOT IN` pozwalają na sprawdzenie, czy wartość znajduje się w zbiorze wartości zwróconym przez podzapytanie.
- Operatory `ALL`, `ANY` i `SOME` pozwalają na porównanie wartości ze zbiorem wartości zwróconym przez podzapytanie.
- Operatory `EXISTS` i `NOT EXISTS` pozwalają na sprawdzenie, czy wynik podzapytania jest pusty.

Podzapytanie skalarne jest najbardziej restrykcyjne, ponieważ powoduje wygenerowanie tylko pojedynczej wartości. Jednak z tego powodu może być stosowane w największej liczbie różnych kontekstów. Tego rodzaju podzapytania sprawdzają się tam, gdzie można używać operandów skalarnych, na przykład jako komponentu wyrażenia, argumentu funkcji lub na liście kolumn danych wyjściowych. Podzapytania kolumny, rekordu i tabeli zwracają większą ilość informacji i tym samym nie mogą być używane w kontekstach wymagających pojedynczej wartości.

Podzapytania mogą być skorelowane lub nieskorelowane. To jest funkcja, do której podzapytanie się odwołuje, a sposób jej działania zależy od wartości w zewnętrznym zapytaniu.

Istnieje możliwość stosowania podzapytań z zapytaniami innymi niż `SELECT`. Jednak w przypadku zapytań modyfikujących tabele (`DELETE`, `INSERT`, `REPLACE`, `UPDATE`, `LOAD DATA`) MySQL nakłada następujące ograniczenie: podzapytanie nie może wybierać danych z modyfikowanej tabeli.

W pewnych sytuacjach podzapytania mogą być tworzone w postaci złączeń. Techniki przepisywania podzapytań możesz uznać za użyteczne w celu przekonania się, czy optymalizator MySQL lepiej się sprawdza w przypadku złączenia niż odpowiadającego mu podzapytania.

Poniższe punkty prezentują różne rodzaje operacji, które można wykonać podczas sprawdzania wyników podzapytania. Przekonasz się, jak tworzyć skorelowane podzapytania, a także jak je przepisywać na postać złączeń.

2.9.1. Podzapytania używające względnych operatorów porównania

Operatory `=`, `<>`, `>`, `>=`, `<`, i `<=` pozwalają na przeprowadzanie operacji porównywania wartości. Kiedy są używane w podzapytaniu skalarnym, wtedy w zewnętrznym zapytaniu wyszukują wszystkie rekordy znajdujące się w określonym związku z wartością zwróconą przez podzapytanie. Na przykład, aby zidentyfikować wyniki sprawdzianu przeprowadzonego dnia '2012-09-23', należy użyć podzapytania skalarnego w celu wyszukania identyfikatora tego sprawdzianu, a następnie w zewnętrznym zapytaniu `SELECT` dopasować rekordy tabeli `score` do znalezionego identyfikatora:

```
SELECT * FROM score
WHERE event_id =
  (SELECT event_id FROM grade_event
   WHERE date = '2012-09-23' AND category = 'Q');
```


W przedstawionej postaci zapytania, gdy podzapytanie jest poprzedzone wartością i operatorem porównania, wspomniane podzapytanie musi wygenerować tylko pojedynczą wartość. Oznacza to, że musi być podzapytaniem skalarnym. Jeżeli zapytanie wygeneruje wiele wartości, wykonanie zapytania zakończy się niepowodzeniem. W pewnych sytuacjach spełnienie wymogu zwrotu pojedynczej wartości może odbyć się przez ograniczenie wyniku zapytania klauzulą LIMIT 1.

Użycie podzapytań skalarnych wraz ze względnymi operatorami porównania przydaje się podczas rozwiązywania problemów, do rozwiązania których często rozważa się użycie funkcji agregującej w klauzuli WHERE. Na przykład, aby z tabeli president pobrać prezydentów, którzy urodzili się jako pierwsi, można wypróbować poniższe zapytanie:

```
SELECT * FROM president WHERE birth = MIN(birth);
```

Powyższe zapytanie nie działa, ponieważ w klauzuli WHERE nie można używać funkcji agregujących. (Klauzula WHERE wskazuje rekordy do pobrania, natomiast wartość funkcji MIN() pozostaje nieznana *aż do chwili wybrania* rekordów). Jednak podzapytanie można wykorzystać do pobrania w następujący sposób najwcześniejszych dat urodzenia:

```
SELECT * FROM president
WHERE birth = (SELECT MIN(birth) FROM president);
```

Inne funkcje agregujące mogą być używane do rozwiązywania podobnych problemów. Poniższe zapytanie stosuje podzapytanie w celu wybrania średnich wyników dla wskazanego ocenianego zdarzenia:

```
SELECT * FROM score WHERE event_id = 5
AND score > (SELECT AVG(score) FROM score WHERE event_id = 5);
```

Jeżeli podzapytanie zwraca pojedynczy rekord, można użyć konstruktora rekordu do porównania zbioru wartości (to znaczy krotki) z wynikiem podzapytania. Poniższe zapytanie zwraca rekordy prezydentów, którzy urodzili się w tym samym mieście i stanie, w którym urodził się John Adams:

```
mysql> SELECT last_name, first_name, city, state FROM president
-> WHERE (city, state) =
-> (SELECT city, state FROM president
-> WHERE last_name = 'Adams' AND first_name = 'John');
```

```
+-----+-----+-----+-----+
| last_name | first_name | city      | state |
+-----+-----+-----+-----+
| Adams    | John      | Braintree | MA     |
| Adams    | John Quincy | Braintree | MA     |
+-----+-----+-----+-----+
```

Istnieje możliwość użycia zapisu ROW(city, state), odpowiadającego zapisowi (city, state). Oba działają w charakterze konstruktorów rekordu.

2.9.2. Podzapytania IN i NOT IN

Operatory IN i NOT IN mogą być używane, gdy podzapytanie zwraca wiele rekordów do porównania z zewnętrznym zapytaniem. Sprawdzają one, czy porównywana wartość znajduje się w zbiorze wartości. Operator NOT IN zwraca true dla rekordów w zewnętrznym zapytaniu, które nie dopasowały rekordów zwróconych przez podzapytanie. Poniższe zapytania używają operatorów IN i NOT IN do wyszukania uczniów posiadających wpisy w tabeli absence oraz tych, którzy nie opuścili żadnych zajęć:

```
mysql> SELECT * FROM student
-> WHERE student_id IN (SELECT student_id FROM absence);
```

name	sex	student_id
Kyle	M	3
Abby	F	5
Peter	M	10
Will	M	17
Avery	F	20

```
mysql> SELECT * FROM student
-> WHERE student_id NOT IN (SELECT student_id FROM absence);
```

name	sex	student_id
Megan	F	1
Joseph	M	2
Katie	F	4
Nathan	M	6
Liesl	F	7

...

Operatory IN i NOT IN sprawdzają się również w podzapytaniach zwracających wiele kolumn. Innymi słowy, można je zastosować w podzapytaniach tabel. W takim przypadku skorzystaj z konstruktora rekordu, aby wskazać porównywane wartości, które powinny być sprawdzone względem wszystkich kolumn:

```
mysql> SELECT last_name, first_name, city, state FROM president
-> WHERE (city, state) IN
-> (SELECT city, state FROM president
-> WHERE last_name = 'Roosevelt');
```

last_name	first_name	city	state
Roosevelt	Theodore	New York	NY
Roosevelt	Franklin D.	Hyde Park	NY

Operatory IN i NOT IN to w rzeczywistości synonimy operatorów = ANY i <> ALL, które zostaną omówione w kolejnym punkcie.

2.9.3. Podzapytania ALL, ANY i SOME

Operatory ALL i ANY są używane w połączeniu ze względny operator porównania, aby sprawdzić wynik podzapytania kolumny. Sprawdzają one, czy porównywana wartość znajduje się w związku ze wszystkimi lub niektórymi wartościami zwróconymi przez podzapytanie. Na przykład, `<= ALL` zwraca wartość true, jeśli porównywana wartość jest mniejsza od wartości zwróconej przez podzapytanie lub jej równa. Natomiast `<= ANY` zwraca wartość true, jeśli porównywana wartość jest mniejsza od dowolnej wartości zwracanej przez podzapytania lub jej równa. Operator SOME jest synonimem ANY.

Poniższe zapytanie ustala, który prezydent urodził się jako pierwszy. Działanie zapytania polega na wybraniu rekordu wraz datą urodzenia mniejszą od wszystkich dat urodzenia w tabeli `president` lub im równą (tylko najwcześniejsza data spełnia ten warunek):

```
mysql> SELECT last_name, first_name, birth FROM president
-> WHERE birth <= ALL (SELECT birth FROM president);
```

last_name	first_name	birth
Washington	George	1732-02-22

Nieco mniej użyteczne jest poniższe zapytanie, które zwraca wszystkie rekordy, ponieważ każda data jest mniejsza przynajmniej od jednej innej daty lub jej równa (samej sobie):

```
mysql> SELECT last_name, first_name, birth FROM president
-> WHERE birth <= ANY (SELECT birth FROM president);
```

last_name	first_name	birth
Washington	George	1732-02-22
Adams	John	1735-10-30
Jefferson	Thomas	1743-04-13
Madison	James	1751-03-16
Monroe	James	1758-04-28

...

Kiedy operatory ALL, ANY lub SOME są używane w połączeniu z operatorem `=`, podzapytanie może być podzapytaniem tabeli. W takim przypadku zwrócone rekordy są sprawdzane za pomocą konstruktora rekordu w celu dostarczenia porównywanych wartości.

```
mysql> SELECT last_name, first_name, city, state FROM president
-> WHERE (city, state) = ANY
-> (SELECT city, state FROM president
-> WHERE last_name = 'Roosevelt');
```

last_name	first_name	city	state
Roosevelt	Theodore	New York	NY
Roosevelt	Franklin D.	Hyde Park	NY

Jak już wspomniano w poprzednim punkcie, operatory `IN` i `NOT IN` są skrótami dla `= ANY i <> ALL`. Dlatego też `IN` oznacza „równy dowolnym rekordom zwróconym przez podzapytanie”, natomiast `NOT IN` oznacza „nierówny wszystkim rekordom zwróconym przez podzapytanie”.

2.9.4. Podzapytania `EXISTS` i `NOT EXISTS`

Operatory `EXISTS` i `NOT EXISTS` pozwalają jedynie na sprawdzenie, czy podzapytanie zwróciło jakiekolwiek rekordy. Jeżeli tak, wartością zwracaną przez `EXISTS` będzie `true`, natomiast przez `NOT EXISTS` będzie `false`. Poniższe zapytania pokazują proste przykłady tego rodzaju podzapytań. Pierwsze z nich zwraca wartość 0, jeśli tabela `absence` jest pusta, z kolei drugie zwraca wartość 1:

```
SELECT EXISTS (SELECT * FROM absence);
SELECT NOT EXISTS (SELECT * FROM absence);
```

Operatory `EXISTS` i `NOT EXISTS` są najczęściej stosowane w podzapytaniach skorelowanych. Odpowiednie przykłady zostaną przedstawione w punkcie 2.9.5, zatytułowanym „Podzapytania skorelowane”.

W przypadku operatorów `EXISTS` i `NOT EXISTS` podzapytanie używa gwiazdki jako listy kolumn danych wyjściowych. Nie ma konieczności wyraźnego podawania kolumn, ponieważ podzapytanie będzie miało przypisaną wartość `true` lub `false` w zależności od tego, czy zwróciło jakiekolwiek rekordy, a nie na podstawie określonych wartości, które znajdują się w zwróconych rekordach. W liście kolumn podzapytania możesz praktycznie podać cokolwiek. Jeżeli chcesz wyraźnie wskazać zwrot wartości `true` w przypadku zakończonego powodzeniem wykonania podzapytania, możesz je wówczas zapisać w postaci `SELECT 1` zamiast `SELECT *`.

2.9.5. Podzapytania skorelowane

Podzapytania mogą być skorelowane lub nieskorelowane:

- Podzapytanie nieskorelowane nie zawiera żadnych odwołań do wartości zapytania zewnętrznego, a więc może być wykonane jako samodzielne zapytanie. Na przykład, podzapytanie w poniższym zapytaniu jest nieskorelowane, ponieważ odwołuje się jedynie do tabel `t1` i `t2`:

```
SELECT j FROM t2 WHERE j IN (SELECT i FROM t1);
```

- Podzapytanie skorelowane zawiera odwołania do wartości zapytania zewnętrznego, a tym samym jest od niego zależne. Ze względu na tego rodzaju powiązanie skorelowane podzapytanie nie może być wykonane jako samodzielne zapytanie. Na przykład, podzapytanie w poniższym zapytaniu jest prawdziwe dla każdej wartości kolumny `j` tabeli `t2` dopasowanej do wartości kolumny `i` tabeli `t1`:

```
SELECT j FROM t2 WHERE (SELECT i FROM t1 WHERE i = j);
```

Skorelowane podzapytania są najczęściej używane wraz z operatorami `EXISTS` i `NOT EXISTS`, co jest użyteczne do wyszukiwania rekordów jednej tabeli dopasowanych lub

niedopasowanych do rekordów w innej tabeli. Skorelowane podzapytania działają przez przekazanie wartości z zewnętrznego zapytania do podzapytania w celu sprawdzenia, czy dopasowują warunki zdefiniowane w podzapytaniu. Z tego powodu konieczne jest stosowanie kwalifikowanych nazw kolumn wraz z tabelami, jeśli są one niejednoznaczne (to znaczy występują w więcej niż tylko jednej tabeli).

Przedstawione poniżej podzapytanie EXISTS wyszukuje dopasowania między tabelami, czyli wartości istniejące w obu. Zapytanie wyszukuje uczniów, którzy przynajmniej raz opuścili zajęcia (mają co najmniej jeden wpis w tabeli absence):

```
SELECT student_id, name FROM student WHERE EXISTS
(SELECT * FROM absence WHERE absence.student_id = student.student_id);
```

Z kolei podzapytanie NOT EXISTS wyszukuje brak dopasowania, czyli wartości jednej tabeli nieistniejące w drugiej. Poniższe zapytanie wyszukuje uczniów, którzy nie opuścili żadnych zajęć:

```
SELECT student_id, name FROM student WHERE NOT EXISTS
(SELECT * FROM absence WHERE absence.student_id = student.student_id);
```

2.9.6. Podzapytania w klauzuli FROM

Podzapytania mogą być używane w klauzuli FROM w celu wygenerowania wartości. W takim przypadku wynik wykonania podzapytania działa w charakterze tabeli. Podzapytanie w klauzuli FROM może partycypować w złączeniach, jego wartości mogą być sprawdzane w klauzuli WHERE itd. W tego rodzaju podzapytaniach konieczne jest użycie aliasu tabeli, aby wynik podzapytania miał zdefiniowaną nazwę:

```
mysql> SELECT * FROM (SELECT 1, 2) AS t1 INNER JOIN (SELECT 3, 4) AS t2;
+-----+-----+
| 1 | 2 | 3 | 4 |
+-----+-----+
| 1 | 2 | 3 | 4 |
+-----+-----+
```

2.9.7. Przepisywanie podzapytań na postać złączeń

Bardzo często istnieje możliwość przepisania zapytania używającego podzapytania na postać złączenia. Dobrym rozwiązaniem jest przeanalizowanie zapytań, które zostały przygotowane jako podzapytania. Ponieważ zdarza się, że złączenie jest znacznie wydajniejsze niż podzapytanie, więc jeśli wykonanie zapytania SELECT utworzonego jako podzapytanie zabiera ogromną ilość czasu, spróbuj przepisać całe zapytanie na postać złączenia i sprawdź jego wydajność działania. Z lektury poniższych punktów dowiesz się, jak to zrobić.

2.9.7.1. Przepisywanie podzapytań wyszukujących dopasowane wartości

Poniżej przedstawiono przykład zapytania zawierającego podzapytanie, które pobiera wyniki z tabeli score uzyskane jedynie w testach (czyli zupełnie ignoruje sprawdziany):

```
SELECT * FROM score
WHERE event_id IN (SELECT event_id FROM grade_event WHERE category = 'T');
```

To samo zapytanie można przepisać na postać pozbawioną zapytania przez konwersję zapytania na proste złączenie:

```
SELECT score.* FROM score INNER JOIN grade_event
ON score.event_id = grade_event.event_id WHERE grade_event.category = 'T';
```

A teraz inny przykład. Poniższe zapytanie pobiera wyniki uzyskane przez dziewczynki:

```
SELECT * from score
WHERE student_id IN (SELECT student_id FROM student WHERE sex = 'F');
```

To zapytanie również można skonwertować na postać złączenia:

```
SELECT score.* FROM score INNER JOIN student
ON score.student_id = student.student_id WHERE student.sex = 'F';
```

W przedstawionych zapytaniach można dostrzec pewien wzorzec. Poniższe podzapytanie stosuje poniższą postać:

```
SELECT * FROM tabela1
WHERE kolumna1 IN (SELECT kolumna2a FROM tabela2 WHERE kolumna2b = wartość);
```

Tego rodzaju zapytania można skonwertować na postać złączenia, używając poniższej formy:

```
SELECT tabela1.* FROM tabela1 INNER JOIN tabela2
ON tabela1.kolumna1 = tabela2.kolumna2a WHERE tabela2.kolumna2b = wartość;
```

W pewnych przypadkach wyniki zwracane przez podzapytanie i złączenie mogą być różne. Zdarza się to, jeżeli *tabela2* zawiera wiele egzemplarzy *kolumny2a*. Postać podzapytania generuje tylko jeden egzemplarz każdej wartości *kolumny2a*, natomiast złączenie generuje je wszystkie, więc dane wyjściowe zawierają powtarzające się rekordy. Aby wyeliminować wspomniane rekordy, złączenie należy rozpocząć od słów kluczowych `SELECT DISTINCT` zamiast po prostu od `SELECT`.

2.9.7.2. Przepisywanie podzapytań wyszukiwujących niedopasowane (brakujące) wartości

Inny często spotykany typ podzapytania powoduje wyszukanie wartości w jednej tabeli nieistniejących w innej. Jak już się wcześniej przekonałeś, problem typu „nieistniejące wartości” wskazuje, że pomocne może być użycie złączenia `LEFT JOIN`. Poniżej przedstawiono zaprezentowane już wcześniej zapytanie wraz z podzapytaniem wyszukujące uczniów, dla których *nie* ma wpisów w tabeli *absence* (wyszukiwani są więc uczniowie obecni na wszystkich zajęciach):

```
SELECT * FROM student
WHERE student_id NOT IN (SELECT student_id FROM absence);
```

Powyższe zapytanie można przepisać na postać złączenia typu `LEFT JOIN`:

```
SELECT student.*
FROM student LEFT JOIN absence ON student.student_id = absence.student_id
WHERE absence.student_id IS NULL;
```

Ogólnie rzecz ujmując, podzapytanie ma następującą postać:

```
SELECT * FROM tabela1
WHERE kolumna1 NOT IN (SELECT kolumna2 FROM tabela2);
```

i można je przepisać na zapytanie o poniższej postaci:

```
SELECT tabela1.*
FROM tabela1 LEFT JOIN tabela2 ON tabela1.kolumna1 = tabela2.kolumna2
WHERE tabela2.kolumna2 IS NULL;
```

Przyjmuje się założenie, że *tabela2.kolumna2* została zdefiniowana jako NOT NULL.

Podzapytanie ma tę zaletę, że jest znacznie bardziej intuicyjne niż LEFT JOIN. „Brak” to koncepcja, którą większość osób rozumie bez problemów, ponieważ występuje ona poza kontekstem programowania bazy danych. Tego samego nie można powiedzieć o koncepcji „lewego złączenia”, dla którego nie ma tego rodzaju odniesienia w rzeczywistym świecie.

2.10. Pobieranie informacji z wielu tabel za pomocą zapytań UNION

W celu utworzenia zbioru wynikowego zawierającego wyniki wykonania wielu zapytań należy użyć zapytania UNION. Na potrzeby przykładów przedstawionych w tym podrozdziale przyjmujemy założenie, że mamy trzy tabele o nazwach t1, t2 i t3 oraz o następującej zawartości:

```
mysql> SELECT * FROM t1;
```

i	c
1	red
2	blue
3	green

```
mysql> SELECT * FROM t2;
```

j	c
-1	tan
1	red

```
mysql> SELECT * FROM t3;
```

d	k
1904-01-01	100
2004-01-01	200
2004-01-01	200

W tabelach t1 i t2 znajdują się kolumny przechowujące liczby całkowite i ciągi tekstowe, natomiast tabela t3 ma kolumny przechowujące daty i ciągi tekstowe. Aby utworzyć

zapytanie UNION łączące wyniki wielu zapytań, konieczne jest przygotowanie wielu zapytań SELECT i umieszczenie między nimi słowa kluczowego UNION. Każde zapytanie SELECT musi pobierać tę samą liczbę kolumn. Na przykład, w celu wybrania kolumny liczb całkowitych ze wszystkich tabel trzeba użyć poniższego zapytania:

```
mysql> SELECT i FROM t1 UNION SELECT j FROM t2 UNION SELECT k FROM t3;
+-----+
| i |
+-----+
| 1 |
| 2 |
| 3 |
| -1 |
| 100 |
| 200 |
+-----+
```

Zapytanie UNION ma omówione poniżej właściwości.

Nazwy kolumn i typy danych. Nazwy kolumn dla wyniku wygenerowanego przez zapytanie UNION są nazwami kolumn w pierwszym zapytaniu SELECT. Drugie i kolejne zapytania SELECT w zapytaniu UNION muszą pobierać dane z tej samej liczby kolumn, ale odpowiadające im kolumny nie muszą mieć tych samych nazw lub typów danych. (Normalnie tworzysz zapytania UNION, w których odpowiadające sobie kolumny mają te same typy danych. Jeśli są inne, to baza danych MySQL przeprowadza konwersję typów, o ile zachodzi potrzeba). Dopasowania kolumn są przeprowadzane na podstawie ich położenia, a nie nazwy. Dlatego też przedstawione poniżej dwa zapytania zwracają różne wyniki, pomimo faktu pobierania tych samych wartości z dwóch tabel:

```
mysql> SELECT i, c FROM t1 UNION SELECT k, d FROM t3;
+-----+-----+
| i | c |
+-----+-----+
| 1 | red |
| 2 | blue |
| 3 | green |
| 100 | 1904-01-01 |
| 200 | 2004-01-01 |
+-----+-----+
mysql> SELECT i, c FROM t1 UNION SELECT d, k FROM t3;
+-----+-----+
| i | c |
+-----+-----+
| 1 | red |
| 2 | blue |
| 3 | green |
| 1904-01-01 | 100 |
| 2004-01-01 | 200 |
+-----+-----+
```

W powyższych zapytaniach typ danych dla każdej kolumny wyniku jest określany na podstawie pobranych wartości. W pierwszym zapytaniu dla drugiej kolumny pobierane są ciągi tekstowe i daty, a wynikiem jest kolumna ciągów tekstowych. Z kolei w drugim

zapytaniu dla pierwszej kolumny pobierane są liczby całkowite i daty, natomiast dla drugiej ciągi tekstowe i liczby całkowite. W obu przypadkach wynikiem jest kolumna ciągów tekstowych.

Obsługa powtarzających się rekordów. Domyślnie, zapytanie UNION usuwa ze zbioru wynikowego powtarzające się rekordy:

```
mysql> SELECT * FROM t1 UNION SELECT * FROM t2 UNION SELECT * FROM t3;
```

i	c
1	red
2	blue
3	green
-1	tan
1904-01-01	100
2004-01-01	200

Tabele t1 i t2 mają rekordy przechowujące wartości 1 i 'red', ale tylko jeden tego rodzaju rekord znalazł się w danych wyjściowych. Ponadto, tabela t3 ma dwa rekordy zawierające dane, '2004-01-01' i 200; jeden z nich został usunięty.

Zapytanie UNION DISTINCT jest synonimem UNION, oba zachowują tylko unikalne rekordy.

Aby zachować powtarzające się rekordy, zamiast UNION trzeba użyć UNION ALL:

```
mysql> SELECT * FROM t1 UNION ALL SELECT * FROM t2 UNION ALL SELECT * FROM t3;
```

i	c
1	red
2	blue
3	green
-1	tan
1	red
1904-01-01	100
2004-01-01	200
2004-01-01	200

Jeżeli połączysz zapytania UNION lub UNION DISTINCT wraz z UNION ALL, wówczas wszystkie operacje wyboru unikalnych wartości będą miały pierwszeństwo przed operacjami UNION ALL.

Obsługa klauzul ORDER BY i LIMIT. W celu posortowania wyniku działania zapytania UNION jako całości każde zapytanie SELECT powinno zostać ujęte w nawias, a po ostatnim należy dodać klauzulę ORDER BY. Ponieważ zapytanie UNION używa nazw kolumn z pierwszego zapytania SELECT, klauzula ORDER BY powinna się odwoływać do nich, a nie do nazw kolumn w ostatnim zapytaniu SELECT:

```
mysql> (SELECT i, c FROM t1) UNION (SELECT k, d FROM t3)
-> ORDER BY c;
```

i	c
---	---

	100	1904-01-01
	200	2004-01-01
	2	blue
	3	green
	1	red

Jeżeli sortowana kolumna jest aliasem, klauzula `ORDER BY` na końcu zapytania `UNION` musi odwoływać się do aliasu. Ponadto, klauzula `ORDER BY` nie może odwoływać się do nazw tabel. Jeżeli musisz przeprowadzić sortowanie według kolumny określonej w pierwszym zapytaniu `SELECT` jako *nazwa_tabeli.nazwa_kolumny*, utwórz alias do kolumny, a następnie użyj go w klauzuli `ORDER BY`.

Podobnie, w celu ograniczenia liczby rekordów zwróconych przez zapytanie `UNION`, dodaj klauzulę `LIMIT` na końcu zapytania:

```
mysql> (SELECT * FROM t1) UNION (SELECT * FROM t2) UNION (SELECT * FROM t3)
-> LIMIT 2;
```

i	c
1	red
2	blue

Klauzule `ORDER BY` i `LIMIT` mogą być używane w poszczególnych zapytaniach `SELECT` ujętych w nawiasach, wtedy działają jedynie w danym zapytaniu:

```
mysql> (SELECT * FROM t1 ORDER BY i LIMIT 2)
-> UNION (SELECT * FROM t2 ORDER BY j LIMIT 1)
-> UNION (SELECT * FROM t3 ORDER BY d LIMIT 2);
```

i	c
1	red
2	blue
-1	tan
1904-01-01	100
2004-01-01	200

Klauzula `ORDER BY` w danym zapytaniu `SELECT` jest używana tylko wraz z klauzulą `LIMIT`, co ma na celu określenie rekordów, względem których ma zastosowanie klauzula `LIMIT`. To nie ma wpływu na kolejność rekordów w ostatecznym wyniku zapytania `UNION`.

2.11. Usuwanie i uaktualnianie rekordów w wielu tabelach

Czasami użyteczne jest usunięcie rekordów na podstawie tego, czy zostały lub nie zostały dopasowane do rekordów w innej tabeli. Podobnie, użyteczne może się okazać uaktualnienie rekordów w jednej tabeli zawartością rekordów pochodzących z innej tabeli. W tym

podrozdziale dowiesz się, jak wykonywać zapytania DELETE i UPDATE względem wielu tabel. Tego rodzaju zapytania opierają się na koncepcji stosowanej w złączeniach, więc upewnij się, że opanowałeś materiał przedstawiony w podrozdziale 2.8, zatytułowanym „Pobieranie danych z wielu tabel za pomocą złączeń”.

Aby wykonać zapytanie DELETE lub UPDATE wykorzystujące tylko jedną tabelę, odwołujesz się do kolumn w pojedynczej tabeli, a tym samym nie musisz stosować kwalifikowanych nazw kolumn. Na przykład, zapytanie usuwające wszystkie rekordy w tabeli t o identyfikatorze większym niż 100 przedstawia się następująco:

```
DELETE FROM t WHERE id > 100;
```

Co zrobić w sytuacji, gdy chcesz usunąć rekordy nie na podstawie właściwości dziedziczonych przez same rekordy, ale ich związków z rekordami w innych tabelach? Przyjmujemy założenie, że z tabeli t chcesz usunąć wszystkie rekordy, których wartości id istnieją lub nie istnieją w tabeli t2.

W celu utworzenia zapytania DELETE obejmującego wiele tabel wszystkie trzeba wymienić w klauzuli FROM, natomiast w klauzuli WHERE podać warunki dopasowujące rekordy w wymienionych tabelach. Poniższe zapytanie powoduje usunięcie w tabeli t1 tych rekordów, których wartość id została dopasowana w tabeli t2:

```
DELETE t1 FROM t1 INNER JOIN t2 ON t1.id = t2.id;
```

Zwróć uwagę, że jeśli taka sama nazwa kolumny występuje w więcej niż tylko jednej tabeli, wtedy jest niejednoznaczna i musi być kwalifikowana nazwą tabeli.

Omówiona składnia pozwala również na jednoczesne usunięcie rekordów z wielu tabel. Aby usunąć rekordy z *obu* tabel, w których zostały dopasowane ich wartości id, podaj je po słowie kluczowym DELETE:

```
DELETE t1, t2 FROM t1 INNER JOIN t2 ON t1.id = t2.id;
```

Co zrobić w sytuacji, gdy chcesz usunąć niedopasowane rekordy? Obejmujące wiele tabel zapytanie DELETE może używać dowolnego złączenia zdefiniowanego w zapytaniu SELECT. Wykorzystaj więc tę samą strategię, której używałeś do utworzenia zapytania SELECT wyszukującego niedopasowane rekordy. Oznacza to zastosowanie złączenia LEFT JOIN lub RIGHT JOIN. Na przykład, aby w tabeli t1 wyszukać rekordy niedopasowane w t2, utwórz zapytanie SELECT takie jak poniższe:

```
SELECT t1.* FROM t1 LEFT JOIN t2 ON t1.id = t2.id WHERE t2.id IS NULL;
```

Analogiczne zapytanie DELETE wyszukujące i usuwające rekordy w tabeli t1 także używa złączenia LEFT JOIN:

```
DELETE t1 FROM t1 LEFT JOIN t2 ON t1.id = t2.id WHERE t2.id IS NULL;
```

Baza danych MySQL obsługuje jeszcze drugą składnię zapytania DELETE obejmującego wiele tabel. Wspomniana druga składnia używa klauzuli FROM do wskazania tabel, w których będą usuwane rekordy, oraz klauzuli USING do złączenia tabel określających rekordy przeznaczone do usunięcia. Przedstawione wcześniej zapytanie DELETE usuwające rekordy z wielu tabel można przepisać na następującą postać:

```
DELETE FROM t1 USING t1 INNER JOIN t2 ON t1.id = t2.id;
DELETE FROM t1, t2 USING t1 INNER JOIN t2 ON t1.id = t2.id;
DELETE FROM t1 USING t1 LEFT JOIN t2 ON t1.id = t2.id WHERE t2.id IS NULL;
```

Reguły dotyczące tworzenia zapytań UPDATE obejmujących wiele tabel są całkiem podobne do stosowanych w zapytaniach DELETE. Należy wymienić tabele biorące udział w operacji, w razie potrzeby stosując kwalifikowane nazwy kolumn. Przyjmujemy założenie, że sprawdzian przeprowadzony 23 września 2012 roku zawierał pytanie, na które wszyscy uczniowie odpowiedzieli błędnie. Następnie odkrywasz, że powodem tego był nieprawidłowy klucz odpowiedzi. Dlatego też do wyniku każdego ucznia chcesz dodać jeden punkt. Dzięki zapytaniu UPDATE obejmującemu wiele tabel możesz to zrobić w następujący sposób:

```
UPDATE score, grade_event SET score.score = score.score + 1
WHERE score.event_id = grade_event.event_id
AND grade_event.date = '2012-09-23' AND grade_event.category = 'Q';
```

W takim przypadku zadanie można wykonać także za pomocą uaktualnienia pojedynczej tabeli i użycia podzapytania:

```
UPDATE score SET score = score + 1
WHERE event_id = (SELECT event_id FROM grade_event
WHERE date = '2012-09-23' AND category = 'Q');
```

Jednak inne uaktualnienia nie mogą być przeprowadzane z użyciem podzapytań. Na przykład, chcesz nie tylko na podstawie zawartości innej tabeli wyszukać rekordy przeznaczone do uaktualnienia, ale także skopiować wartości kolumn z jednej tabeli do drugiej. Poniższe zapytanie kopiuje t1.a do t2.a dla rekordów, w których dopasowano wartość kolumny id:

```
UPDATE t1, t2 SET t2.a = t1.a WHERE t2.id = t1.id;
```

W celu przeprowadzenia obejmującej wiele tabel operacji usunięcia lub uaktualnienia tabel InnoDB nie musisz używać przedstawionej tutaj składni. Zamiast tego zdefiniuj relację klucza zewnętrznego między tabelami zawierającymi ograniczenie ON DELETE CASCADE lub ON UPDATE CASCADE. Więcej informacji szczegółowych na ten temat znajdziesz w podrozdziale 2.13, zatytułowanym „Klucze zewnętrzne i integralność odwołań”.

2.12. Przeprowadzanie transakcji

Transakcja to zestaw zapytań SQL wykonywanych jako całość, które w razie potrzeby można wycofać. Oznacza to, że wykonanie wszystkich zapytań musi zakończyć się powodzeniem lub żadne z nich nie powinno być wykonane. Wspomnianą możliwość uzyskuje się dzięki opcji zatwierdzania i wycofywania transakcji. Jeżeli wykonanie wszystkich zapytań w transakcji zakończy się powodzeniem, zostaje ona zatwierdzona i ma trwały efekt w bazie danych. W przypadku wystąpienia błędu w transakcji następuje jej wycofanie i anulowanie. Wszelkie wykonane dotąd zapytania transakcji zostają cofnięte, a baza danych pozostaje w stanie sprzed rozpoczęcia transakcji.

Zatwierdzanie i wycofywanie transakcji to gwarancja, że w bazie danych nie będą częściowo przeprowadzane operacje, które pozostawiłyby ją w częściowo uaktualnionym (niespójnym) stanie. Kanonicznym przykładem jest tutaj przelew pieniężny z jednego konta na inne. Przyjmujemy założenie, że Bartek zleca wykonanie Tomaszowi przelewu na kwotę 100 zł. W takim przypadku saldo na koncie Bartka powinno zostać zmniejszone o 100 zł i jednocześnie saldo na koncie Tomka zwiększone o 100 zł.

```
UPDATE account SET balance = balance - 100 WHERE name = 'Bartek';  
UPDATE account SET balance = balance + 100 WHERE name = 'Tomek';
```

Jeżeli między dwoma wymienionymi operacjami dojdzie do awarii, operacja pozostaje niekompletna. W zależności od tego, która operacja została wykonana jako pierwsza, Bartek zostanie obciążony kwotą 100 zł, a Tomek nie otrzyma tych pieniędzy lub Tomek otrzyma 100 zł, ale Bartek nie zostanie obciążony wymienioną kwotą. Żadne z przedstawionych rozwiązań nie jest dobre. Jeżeli mechanizm transakcji jest niedostępny, stan operacji wykonywanych w trakcie awarii trzeba ustalić ręcznie przez analizę plików dzienników zdarzeń, a następnie określić, jak wycofać lub dokończyć operacje. Oferowana przez transakcję możliwość jej wycofania pozwala na prawidłową obsługę tego rodzaju sytuacji przez wycofanie operacji wykonanych przed wystąpieniem awarii. (Nadal może wystąpić konieczność ustalenia niewykonanych transakcji i ich powtórzenia, ale przynajmniej nie trzeba się martwić częściowo wykonanymi transakcjami, które pozostawiłyby bazę danych w niespójnym stanie).

Innym przykładem użycia transakcji jest zagwarantowanie, że rekordy zaangażowane w daną operację nie będą w jej trakcie modyfikowane przez inne klienty. MySQL automatycznie nakłada blokadę na rekordy w trakcie wykonywania pojedynczych zapytań SQL, aby uniemożliwić zakłócenia. To jednak nie zawsze jest wystarczającą gwarancją otrzymaniażądanego efektu w bazie danych, ponieważ niektóre operacje składają się z wielu zapytań. W takim przypadku różne klienty mogą po prostu wzajemnie sobie przeszkadzać. Transakcja powoduje zgrupowanie zapytań w pojedynczą jednostkę wykonywania, co chroni przed problemami związanymi ze współbieżnością, które w przeciwnym razie mogłyby wystąpić w środowisku obejmującym wiele klientów.

Systemy transakcyjne są zwykle charakteryzowane jako dostarczające właściwości ACID. To jest akronim od *Atomic* (niepodzielność), *Consistent* (spójność), *Isolated* (izolacja) i *Durable* (trwałość), czyli czterech właściwości, którymi powinna charakteryzować się każda transakcja:

- **Niepodzielność** — zapytania składające się na transakcję tworzą logiczną jednostkę. Nie można więc wykonać tylko niektórych z nich.
- **Spójność** — baza danych musi być spójna przed przeprowadzeniem i po przeprowadzeniu transakcji. Na przykład, jeżeli rekordy w jednej tabeli nie mogą mieć identyfikatorów niewymienionych w drugiej tabeli, transakcja próbująca wstawić rekordy o nieprawidłowych identyfikatorach powinna zakończyć się niepowodzeniem i zostać wycofana.

- **Izolacja** — jedna transakcja nie może mieć wpływu na inną, więc jednocześnie przeprowadzane transakcje powinny mieć taki sam efekt, jakby zostały przeprowadzone po kolei.
- **Trwałość** — kiedy transakcja zakończy się powodzeniem, jej efekt powinien być na trwałe zarejestrowany w bazie danych.

Przetwarzanie transakcyjne daje mocne gwarancje w zakresie wyniku operacji bazy danych, ale jednocześnie wiąże się z większym obciążeniem wyrażonym w używanych cyklach procesora, pamięci i miejsca na dysku. MySQL oferuje silniki bazy danych zapewniające bezpieczeństwo transakcji (na przykład InnoDB), a także nieobsługujące transakcji (na przykład MyISAM i MEMORY). Właściwości transakcyjne mają wręcz znaczenie krytyczne dla niektórych aplikacji, a dla innych pozostają bez znaczenia. Samodzielnie musisz zdecydować, czy są wymagane w tworzonej przez Ciebie aplikacji. Operacje finansowe z reguły wymagają stosowania gwarancji, a gwarancja zachowania spójności danych jest cenniejsza niż dodatkowe obciążenie związane z przeprowadzeniem transakcji. Z drugiej strony, w przypadku aplikacji rejestrującej w tabeli bazy danych informacje o dostępie do strony internetowej utrata kilku rekordów, jeśli serwer ulegnie awarii, jest do zaakceptowania. W takim przypadku użycie nietransakcyjnego silnika bazy danych pozwala na uniknięcie obciążenia związanego z przetwarzaniem transakcji.

2.12.1. Użycie transakcji do zapewnienia bezpiecznego środowiska wykonywania

Użycie transakcji wymaga transakcyjnego silnika bazy danych, takiego jak InnoDB. Silniki takie jak MyISAM i MEMORY nie spełniają tego warunku. Jeżeli nie jesteś pewien, czy Twój serwer MySQL obsługuje transakcyjne silniki bazy danych, to zajrzyj do podpunktu 2.6.1.1, zatytułowanego „Ustalenie dostępnych silników bazy danych”.

Domyślnie, MySQL działa w trybie automatycznego zatwierdzania. Oznacza to, że zmiany wprowadzone przez poszczególne zapytania są natychmiast zatwierdzane w bazie danych i mają trwały efekt. Dlatego też każde zapytanie w praktyce stanowi transakcję. Aby transakcje zatwierdzać samodzielnie, trzeba wyłączyć tryb automatycznego zatwierdzania, a następnie wskazywać MySQL, kiedy zatwierdzić lub wycofać zmiany.

Jednym ze sposobów przeprowadzenia transakcji jest wykonanie zapytania `START TRANSACTION` (lub `BEGIN`) w celu zawieszenia trybu automatycznego zatwierdzania, wykonanie zapytań tworzących transakcję, a następnie zakończenie jej zapytaniem `COMMIT`, które na trwałe wprowadza zmiany w bazie danych. Jeśli w trakcie transakcji wystąpi błąd, należy ją anulować, wykonując zapytanie `ROLLBACK`, które wycofuje wszelkie zmiany wprowadzone przez transakcję.

Zapytanie `START TRANSACTION` tylko zawiesza aktualny tryb automatycznego zatwierdzania, więc po zatwierdzeniu lub wycofaniu transakcji wspomniany tryb wraca do pierwotnego stanu. Jeżeli przed rozpoczęciem transakcji włączony był tryb automatycznego zatwierdzania, zakończenie transakcji przywraca ten tryb. Z kolei,

jeśli tryb automatycznego zatwierdzania był wyłączony, koniec bieżącej transakcji powoduje rozpoczęcie kolejnej.

Przedstawiony poniżej przykład ilustruje omówione podejście. Najpierw tworzymy tabelę, której będziemy używali w przykładzie.

```
mysql> CREATE TABLE t (name CHAR(20), UNIQUE (name)) ENGINE=InnoDB;
```

Następne kroki to inicjalizacja transakcji za pomocą zapytania `START TRANSACTION`, dodanie kilku rekordów do tabeli, zatwierdzenie transakcji i sprawdzenie, jaka jest zawartość tabeli:

```
mysql> START TRANSACTION;
mysql> INSERT INTO t SET name = 'William';
mysql> INSERT INTO t SET name = 'Wallace';
mysql> COMMIT;
mysql> SELECT * FROM t;
+-----+
| name  |
+-----+
| Wallace |
| William |
+-----+
```

Jak możesz się przekonać, rekordy zostały wstawione do tabeli. Jeżeli uruchomisz drugi egzemplarz `mysql` i wybierzesz zawartość tabeli `t` po wstawieniu rekordów, ale jeszcze przed zatwierdzeniem transakcji, rekordy nie zostaną wyświetlone. Po prostu nie będą widoczne dla drugiego procesu `mysql` aż do chwili wykonania zapytania `COMMIT` w pierwszym egzemplarzu `mysql`.

Jeżeli w trakcie transakcji wystąpi błąd, można ją wycofać za pomocą zapytania `ROLLBACK`. Ponownie używając tabeli `t`, możesz się o tym przekonać, wykonując poniższe zapytania:

```
mysql> START TRANSACTION;
mysql> INSERT INTO t SET name = 'Gromit';
mysql> INSERT INTO t SET name = 'Wallace';
ERROR 1062 (23000): Duplicate entry 'Wallace' for key 'name'
mysql> ROLLBACK;
mysql> SELECT * FROM t;
+-----+
| name  |
+-----+
| Wallace |
| William |
+-----+
```

Drugie zapytanie `INSERT` podjęło próbę wstawienia do tabeli rekordu powielającego istniejącą wartość `name`. Ta próba zakończyła się niepowodzeniem, ponieważ dla `name` zdefiniowany jest indeks `UNIQUE`. Po wykonaniu zapytania `ROLLBACK` tabela nadal ma tylko dwa rekordy, które zawierała przed rozpoczęciem transakcji. Zwróć uwagę na wycofanie skutku zapytania `INSERT`, którego wykonanie zakończyło się powodzeniem. Jego efekt nie został trwale zapisany w tabeli.

Wykonanie zapytania `START TRANSACTION` w czasie trwania transakcji powoduje zatwierdzenie bieżącej i rozpoczęcie nowej.

Innym sposobem przeprowadzania transakcji jest bezpośrednia zmiana trybu automatycznego zatwierdzania za pomocą zapytań SET:

```
SET autocommit = 0;
SET autocommit = 1;
```

Przypisanie zmiennej `autocommit` wartości zero powoduje wyłączenie automatycznego zatwierdzania transakcji. Efekt wykonania kolejnych zapytań staje się częścią bieżącej transakcji, która kończy się w chwili wykonania zapytania `COMMIT` lub `ROLLBACK` w celu odpowiednio zatwierdzenia lub wycofania transakcji. W przypadku tej metody automatyczne zatwierdzanie pozostanie wyłączone aż do chwili jego ręcznego ponownego włączenia. Dlatego też zakończenie jednej transakcji powoduje rozpoczęcie kolejnej. Transakcję możesz również zatwierdzić przez ponowne włączenie trybu automatycznego zatwierdzania.

Aby przekonać się, jak działa omówione rozwiązanie, przeanalizujemy przykład wykorzystujący te same tabele, z których korzystaliśmy w poprzednim przykładzie.

```
mysql> DROP TABLE t;
mysql> CREATE TABLE t (name CHAR(20), UNIQUE (name)) ENGINE=InnoDB;
```

Następnie wyłączamy tryb automatycznego zatwierdzania, wstawiamy nowe rekordy i ręcznie zatwierdzamy transakcję:

```
mysql> SET autocommit = 0;
mysql> INSERT INTO t SET name = 'William';
mysql> INSERT INTO t SET name = 'Wallace';
mysql> COMMIT;
mysql> SELECT * FROM t;
+-----+
| name  |
+-----+
| Wallace |
| William |
+-----+
```

Na tym etapie dwa rekordy zostały wstawione do tabeli, ale tryb automatycznego zatwierdzania nadal pozostaje wyłączony. Jeżeli będziesz wykonywał kolejne zapytania, staną się one częścią nowej transakcji, którą będzie można zatwierdzić lub wycofać niezależnie od pierwszej. Aby potwierdzić, że tryb automatycznego zatwierdzania nadal pozostaje wyłączony i zapytanie `ROLLBACK` wycofa niezatwierdzone jeszcze zapytania, wykonaj następujące zapytania:

```
mysql> INSERT INTO t SET name = 'Gromit';
mysql> INSERT INTO t SET name = 'Wallace';
ERROR 1062 (23000): Duplicate entry 'Wallace' for key 'name'
mysql> ROLLBACK;
mysql> SELECT * FROM t;
+-----+
| name  |
+-----+
| Wallace |
| William |
+-----+
```


W celu ponownego włączenia trybu automatycznego zatwierdzania wykonaj poniższe zapytanie:

```
mysql> SET autocommit = 1;
```

Jak już wcześniej napisano, transakcja kończy się po wykonaniu zapytania COMMIT lub ROLLBACK bądź też po ponownym włączeniu trybu automatycznego zatwierdzania, o ile był wcześniej wyłączony. Transakcje kończą się również w innych sytuacjach. Poza zapytaniami SET autocommit, START TRANSACTION, BEGIN, COMMIT i ROLLBACK, wyraźnie wpływającymi na transakcje, także inne zapytania mają niejawni wpływ na transakcje, ponieważ nie mogą być ich częścią. Ogólnie rzecz biorąc, to zwykle zapytania typu DDL (ang. *Data Definition Language*) tworzące, zmieniające lub usuwające bazy danych bądź ich obiekty, a także zapytania powiązane z nakładaniem blokad. Na przykład, wykonanie dowolnego z poniższych zapytań w trakcie przeprowadzanej transakcji spowoduje, że najpierw serwer zatwierdzi transakcję, a dopiero później wykona zapytanie:

```
ALTER TABLE
CREATE INDEX
DROP DATABASE
DROP INDEX
DROP TABLE
LOCK TABLES
RENAME TABLE
SET autocommit = 1 (if not already set to 1)
TRUNCATE TABLE
UNLOCK TABLES (if tables currently are locked)
```

Pełną listę zapytań powodujących zatwierdzenie transakcji w używanej wersji serwera MySQL znajdziesz w podręczniku użytkownika.

Transakcja kończy się również wraz z zakończeniem lub awarią sesji pracy klienta. W tym drugim przypadku serwer automatycznie wycofuje transakcję aktualnie przeprowadzaną przez klienta.

Jeżeli program klienta automatycznie nawiązuje ponowne połączenie po przerwaniu sesji, połączenie jest zerowane do jego stanu domyślnego, czyli z włączonym trybem automatycznego zatwierdzania.

Transakcje są użyteczne w wielu różnych sytuacjach. Przyjmijmy założenie, że pracujesz z tabelą score będącą częścią projektu ocen uczniów. W trakcie pracy odkrywasz, że oceny dwóch uczniów zostały zamienione i trzeba je zamienić miejscami. Nieprawidłowe oceny można wyświetlić za pomocą poniższego zapytania:

```
mysql> SELECT * FROM score WHERE event_id = 5 AND student_id IN (8,9);
```

student_id	event_id	score
8	5	18
9	5	13

Aby naprawić błąd, uczeń 8 powinien otrzymać wynik 13, natomiast uczeń 9 wynik 18. Rozwiązanie jest proste i wymaga wykonania dwóch poniższych zapytań:

```
UPDATE score SET score = 13 WHERE event_id = 5 AND student_id = 8;
UPDATE score SET score = 18 WHERE event_id = 5 AND student_id = 9;
```

Jednak konieczne jest zagwarantowanie wykonania obu zapytań jako jednostki. Tutaj z pomocą przychodzi transakcja, którą wykorzystujemy w następujący sposób:

```
mysql> START TRANSACTION;
mysql> UPDATE score SET score = 13 WHERE event_id = 5 AND student_id = 8;
mysql> UPDATE score SET score = 18 WHERE event_id = 5 AND student_id = 9;
mysql> COMMIT;
```

Wykonanie tego samego zadania przez operowanie trybem automatycznego zatwierdzania wymaga wykonania poniższych zapytań:

```
mysql> SET autocommit = 0;
mysql> UPDATE score SET score = 13 WHERE event_id = 5 AND student_id = 8;
mysql> UPDATE score SET score = 18 WHERE event_id = 5 AND student_id = 9;
mysql> COMMIT;
mysql> SET autocommit = 1;
```

Niezależnie od przyjętego wariantu, wynikiem jest prawidłowa zamiana ocen miejscami:

```
mysql> SELECT * FROM score WHERE event_id = 5 AND student_id IN (8,9);
```

student_id	event_id	score
8	5	13
9	5	18

2.12.2. Użycie punktów pośrednich transakcji

MySQL pozwala na częściowe wycofanie transakcji. Wymaga to wcześniejszego wykonania zapytania `SAVEPOINT` w transakcji, które tworzy punkt pośredni o wskazanej nazwie.

W celu wycofania później transakcji od wskazanego punktu należy wykonać zapytanie `ROLLBACK` wraz z nazwą punktu pośredniego. Poniższe zapytania pokazują sposób działania omówionego mechanizmu:

```
mysql> CREATE TABLE t (i INT) ENGINE=InnoDB;
mysql> START TRANSACTION;
mysql> INSERT INTO t VALUES(1);
mysql> SAVEPOINT my_savepoint;
mysql> INSERT INTO t VALUES(2);
mysql> ROLLBACK TO SAVEPOINT my_savepoint;
mysql> INSERT INTO t VALUES(3);
mysql> COMMIT;
mysql> SELECT * FROM t;
```

i
1
3

Po wykonaniu powyższych zapytań pierwszy i trzeci rekord zostały wstawione, natomiast drugi został wycofany przez operację częściowego wycofania transakcji od punktu pośredniego `my_savepoint`.

2.12.3. Izolacja transakcji

Ponieważ MySQL to wielodostępny system bazy danych, w tym samym czasie różne klienty mogą próbować używać dowolnych z jego tabel. Silniki bazy danych, takie jak MyISAM, stosują mechanizm nakładania blokad, aby uniemożliwić wielu klientom jednoczesną modyfikację tabel. Takie rozwiązanie jednak nie zapewnia dobrej wydajności w przypadku konieczności przeprowadzenia wielu operacji uaktualniania danych. W silniku InnoDB zastosowano inne podejście — blokowanie na poziomie rekordów, które pozwala na zachowanie dokładniejszej kontroli nad uzyskaniem dostępu do tabeli przez klienty. Jeden klient może modyfikować rekord, w tym samym czasie inny klient może odczytywać lub modyfikować zupełnie inny rekord w tej samej tabeli. Jeżeli oba klienty będą chciały jednocześnie zmodyfikować ten sam rekord, ten, który pierwszy nałoży blokadę, będzie mógł jako pierwszy przeprowadzić jego modyfikację. W ten sposób otrzymujemy lepszą wydajność współbieżności niż podczas nakładania blokad na tabele. Rodzi się jednak pytanie, czy transakcja jednego klienta powinna „widzieć” zmiany wprowadzane przez transakcję innego klienta.

Silnik InnoDB implementuje poziomy izolacji transakcji, co pozwala klientom na zachowanie kontroli nad rodzajami zmian, które chcą widzieć po ich wprowadzeniu przez inne transakcje. Poszczególne poziomy izolacji prowadzą do problemów występujących w trakcie jednoczesnego przeprowadzania różnych transakcji lub pozwalają ich uniknąć:

- **Niezatwierdzone odczyty** (ang. *dirty reads*). Niezatwierdzone odczyty występują, gdy zmiany wprowadzone przez jedną transakcję mogą być widziane przez inne transakcje, zanim sama transakcja zostanie zatwierdzona. W ten sposób druga transakcja może uznać rekord za zmodyfikowany, nawet jeśli to nie do końca jest prawdą, ponieważ transakcja modyfikująca wspomniany rekord może zostać wycofana.
- **Niepowtarzalne odczyty** (ang. *non-repeatable reads*). Niepowtarzalne odczyty oznaczają niepowodzenie transakcji objawiające się brakiem możliwości uzyskania tego samego wyniku podczas wykonywania danego zapytania SELECT za każdym razem. Tego rodzaju sytuacja może wystąpić, jeśli jedna transakcja dwukrotnie wykonuje zapytanie SELECT, natomiast inna transakcja zmienia pewne rekordy między wykonaniem dwóch wspomnianych zapytań.
- **Odczyty widma** (ang. *phantom reads*). Widmo to rekord widoczny dla transakcji, choć wcześniej pozostawał niewidoczny. Przyjmujemy założenie, że transakcja wykonuje zapytanie SELECT, a następnie inna transakcja wstawia rekord. Jeżeli pierwsza transakcja ponownie wykona to samo zapytanie SELECT i zobaczy nowy rekord, wtedy właśnie mówimy o rekordzie widmo.

W celu rozwiązywania wymienionych problemów InnoDB obsługuje cztery poziomy izolacji transakcji. Wspomniane poziomy pozwalają na wskazanie, które modyfikacje wprowadzane przez jedną transakcję mogą być widoczne w innych, jednocześnie przeprowadzanych transakcjach:

- **READ UNCOMMITTED**: transakcja może zobaczyć modyfikacje rekordów wprowadzone przez inne transakcje, nawet jeśli jeszcze nie zostały zatwierdzone.
- **READ COMMITTED**: transakcja może zobaczyć modyfikacje rekordów wprowadzone przez inne transakcje, tylko jeśli zostały zatwierdzone.
- **REPEATABLE READ**: jeżeli transakcja dwukrotnie wykonuje dane zapytanie **SELECT**, wynik jest powtarzalny. Oznacza to, że za każdym razem otrzymuje ten sam wynik, nawet jeśli w międzyczasie inne transakcje zmieniły lub wstawiły rekordy.
- **SERIALIZABLE**: poziom izolacji podobny do **REPEATABLE READ**, ale niemalże całkowicie izoluje transakcje. Rekordy używane przez jedną transakcję nie mogą być modyfikowane przez inne transakcje aż do chwili ukończenia pierwszej. W ten sposób, gdy jedna transakcja odczytuje rekordy, w tym samym czasie inne transakcje nie mogą ich modyfikować.

W tabeli 2.4 wymieniono poziomy izolacji i wskazano, czy obsługują odczyty niezatwierdzone i niepowtarzalne oraz rekordy widma. Tabela została przygotowana pod kątem silnika InnoDB, w którym poziom izolacji **REPEATABLE READ** nie zezwala na występowanie rekordów widm. Pewne systemy baz danych pozwalają na rekordy widma na poziomie izolacji **REPEATABLE READ**.

Tabela 2.4. Problemy związane z poziomami izolacji

Poziom izolacji	Niezatwierdzone odczyty	Niepowtarzalne odczyty	Rekordy widma
READ UNCOMMITTED	Tak	Tak	Tak
READ COMMITTED	Nie	Tak	Tak
REPEATABLE READ	Nie	Nie	Nie
SERIALIZABLE	Nie	Nie	Nie

Domyślny poziom izolacji InnoDB to **REPEATABLE READ**. Ustawienie można zmienić w trakcie uruchamiania serwera (opcja `--transaction-isolation`) lub w trakcie jego działania (za pomocą zapytania **SET TRANSACTION**). Wymienione zapytanie ma trzy następujące składnie:

```
SET GLOBAL TRANSACTION ISOLATION LEVEL poziom;
SET SESSION TRANSACTION ISOLATION LEVEL poziom;
SET TRANSACTION ISOLATION LEVEL poziom;
```

Klient z uprawnieniem **SUPER** może użyć zapytania **SET TRANSACTION** do zmiany globalnego poziomu izolacji, który następnie jest stosowany przez wszystkie klienty

nawiązujące połączenie z serwerem. Poza tym, poszczególne klienty mogą zmienić własny poziom izolacji dla wszystkich kolejnych transakcji w danej sesji z serwerem (po podaniu opcji `SESSION`) lub tylko dla następnej transakcji (w przypadku pominięcia opcji `SESSION`). Ustawienie poziomu izolacji na poziomie klienta nie wymaga żadnych szczególnych uprawnień.

Czy można łączyć tabele transakcyjne z nietransakcyjnymi?

W trakcie transakcji istnieje możliwość użycia obu wymienionych rodzajów tabel, ale otrzymany wynik może odbiegać od oczekiwanego. Zapytania wykonywane względem tabel nietransakcyjnych będą wprowadzone natychmiast, nawet w przypadku wyłączonego trybu automatycznego zatwierdzania. Dlatego też tabele nietransakcyjne zawsze działają w trybie automatycznego zatwierdzania, a wszystkie zapytania są natychmiast zatwierdzane. Jeśli w transakcji użyjesz obu wymienionych rodzajów tabel, a następnie zdecydujesz się na wycofanie transakcji, to zmian wprowadzonych w tabelach nietransakcyjnych nie będzie można wycofać.

2.13. Klucze zewnętrzne i integralność odwołań

Relacja klucza zewnętrznego pozwala na zadeklarowanie, że indeks w jednej tabeli jest powiązany z indeksem w innej. W ten sposób można nałożyć ograniczenia określające to, co może być zrobione w powiązanych ze sobą tabelach. Baza danych wymusza, aby reguły w takiej relacji zapewniły zachowanie integralności odwołań. Na przykład, tabela `score` w bazie danych `sampdb` zawiera kolumnę `student_id`, która jest używana do powiązania rekordów ocen z rekordami uczniów w tabeli `student`. Podczas tworzenia wymienionych tabel w rozdziale 1. skonfigurowaliśmy pewne relacje między nimi. Między innymi zdefiniowaliśmy, że `score.student_id` jest kluczem zewnętrznym dla kolumny `student.student_id`. W ten sposób do tabeli `score` nie można wstawić rekordu, jeśli jego wartość `student_id` nie istnieje w tabeli `student`. Innymi słowy, klucz zewnętrzny uniemożliwia wstawianie ocen dla nieistniejących uczniów.

Klucze zewnętrzne są użyteczne nie tylko podczas wstawiania rekordów, ale również w trakcie ich usuwania i uaktualniania. Na przykład, istnieje możliwość zdefiniowania ograniczenia polegającego na tym, że po usunięciu ucznia z tabeli `student` nastąpi automatyczne usunięcie wszystkich powiązanych z nim rekordów w tabeli `score`. Takie rozwiązanie nosi nazwę „kaskadowego usuwania”, ponieważ operacja usunięcia w jednej tabeli prowadzi również do operacji usunięcia w innej. Istnieją także kaskadowe uaktualnienia. Na przykład, w przypadku kaskadowego uaktualnienia zmiana identyfikatora ucznia (`student_id`) w tabeli `student` prowadzi również do zmiany wartości w znajdujących się w tabeli `score` rekordach dotyczących tego ucznia.

Klucze zewnętrzne zachowują spójność danych oraz zapewniają pewnego rodzaju wygodę. Bez ich użycia trzeba samodzielnie śledzić zależności między tabelami i zapewnić

ich spójność z poziomu aplikacji. W pewnych przypadkach to może oznaczać wykonanie jedynie kilku dodatkowych zapytań DELETE, aby upewnić się, że po usunięciu rekordów z jednej tabeli nastąpi usunięcie odpowiadających im rekordów w innych tabelach. To jednak *jest* dodatkowa praca do wykonania, a skoro silnik bazy danych potrafi samodzielnie zapewnić spójność danych, to dlaczego nie skorzystać z tej możliwości? Automatyczne zapewnienie spójności jest szczególnie użyteczne, gdy dla tabel zdefiniowano skomplikowane relacje. Prawdopodobnie nie chcesz być odpowiedzialny za implementację tego rodzaju zależności we własnej aplikacji.

W MySQL silnik InnoDB zapewnia obsługę klucza zewnętrznego. W tym podrozdziale dowiesz się, jak skonfigurować tabele InnoDB w celu zdefiniowania kluczy zewnętrznych. Przekonasz się również, jak klucze zewnętrzne wpływają na sposób używania tabel. W pierwszej kolejności konieczne jest zdefiniowanie pewnych terminów:

- Tabela nadrzędna to taka, która zawiera oryginalne wartości klucza.
- Tabela potomna to tabela powiązana, która odwołuje się do wartości klucza umieszczonych w tabeli nadrzędnej.

Znajdujące się w tabeli nadrzędnej wartości klucza są używane do powiązania dwóch tabel. W szczególności, indeks w tabeli potomnej odwołuje się do indeksu w tabeli nadrzędnej. Wartości indeksu potomnego muszą być dopasowane do wartości indeksu nadrzędnego lub zawierać wartość NULL, wskazującą brak powiązanego rekordu w tabeli nadrzędnej. Indeks w tabeli potomnej jest nazywany „kluczem zewnętrznym” — to znaczy, klucz pozostaje zewnętrzny dla tabeli nadrzędnej, ale zawiera wartości do niej prowadzące. Relacja klucza zewnętrznego może mieć przypisaną wartość NULL; w takim przypadku wszystkie wartości klucza zewnętrznego muszą mieć dopasowane wartości w tabeli nadrzędnej.

Silnik InnoDB wymusza stosowanie wymienionych reguł, aby zagwarantować, że w relacji klucza zewnętrznego nie wystąpią niewłaściwe dopasowania. To nosi nazwę „integralności odwołań” (ang. *referential integrity*).

Poniższa składnia pokazuje, w jaki sposób można zdefiniować klucz zewnętrzny w tabeli potomnej:

```
[CONSTRAINT nazwa_ograniczenia]
FOREIGN KEY [nazwa_klucza_zewnetrznego] (kolumny_indeksu)
REFERENCES nazwa_tabeli (kolumny_indeksu)
[ON DELETE akcja]
[ON UPDATE akcja]
[MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
```

Wprawdzie wszystkie części powyższej składni są przetwarzane, ale InnoDB nie implementuje semantyki dla wszystkich klauzul. Dlatego też klauzula MATCH nie jest obsługiwana i zostanie zignorowana, jeśli jej użyjesz. Ponadto, pewne wartości akcji są rozpoznawane, choć nie mają żadnego efektu. (W przypadku silników bazy danych innych niż InnoDB cała definicja FOREIGN KEY będzie przetworzona, ale zostanie zignorowana).

Silnik InnoDB zwraca szczególną uwagę na wymienione poniżej części definicji:

- W klauzuli CONSTRAINT (o ile będzie podana) następuje zdefiniowanie ograniczenia klucza zewnętrznego. Jeżeli pominiesz nazwę, silnik InnoDB utworzy ją samodzielnie.

- **FOREIGN KEY** wskazuje zindeksowane kolumny w tabeli potomnej, których wartości muszą być dopasowane do wartości indeksu w tabeli nadrzędnej. Identyfikatorem klucza zewnętrznego jest *nazwa_klucza_zewnętrznego*. Jeśli zostanie podana, będzie ignorowana, o ile InnoDB automatycznie nie utworzy indeksu dla klucza zewnętrznego. W takim przypadku *nazwa_klucza_zewnętrznego* stanie się nazwą indeksu.
- **REFERENCES** wskazuje nazwy tabeli nadrzędnej i kolumn indeksu w tej tabeli, do których będzie odwoływał się klucz zewnętrzny w tabeli potomnej. Część *kolumny_indeksu* w klauzuli **REFERENCES** musi mieć tę samą liczbę kolumn, jak w części *kolumny_indeksu* po słowach kluczowych **FOREIGN KEY**.
- **ON DELETE** pozwala na określenie zachowania tabeli potomnej po usunięciu rekordów w tabeli nadrzędnej. W przypadku pominięcia tej klauzuli domyślnie następuje odrzucenie wszelkich prób usunięcia rekordów w tabeli nadrzędnej, do których odwołują się rekordy w tabeli potomnej. W celu wyraźnego zdefiniowania *akcji* należy użyć jednej z wymienionych poniżej klauzul:
 - ◆ **ON DELETE NO ACTION** i **ON DELETE RESTRICT** mają taki sam skutek jak pominięcie klauzuli **ON DELETE**. W niektórych systemach bazy danych istnieje odroczone sprawdzanie, a **NO ACTION** to właśnie rodzaj takiego sprawdzenia. W silniku InnoDB ograniczenia klucza zewnętrznego są sprawdzane natychmiast, więc efekt działania **NO ACTION** i **RESTRICT** jest taki sam.
 - ◆ **ON DELETE CASCADE** powoduje wyszukanie w tabeli potomnej rekordów dopasowanych do usuwanych w tabeli nadrzędnej i ich usunięcie. W efekcie oznacza to operację usunięcia kaskadowego. W ten sposób możesz przeprowadzać usunięcia rekordów w wielu tabelach przez usunięcie jedynie rekordów w tabeli nadrzędnej i pozwolenie silnikowi InnoDB na usunięcie odpowiednich rekordów z tabeli potomnej.
 - ◆ **ON DELETE SET NULL** powoduje, że kolumny indeksu w dopasowanych rekordach tabeli potomnej będą miały przypisaną wartość **NULL** po usunięciu rekordów w tabeli nadrzędnej. Jeżeli użyjesz tej opcji, wszystkie zindeksowane kolumny tabeli potomnej wymienione w definicji klucza zewnętrznego muszą być zdefiniowane w sposób dopuszczający przechowywanie wartości **NULL**. (Jedną z implikacji użycia tej akcji jest brak możliwości zdefiniowania klucza zewnętrznego jako **PRIMARY KEY**; klucze podstawowe nie mogą mieć wartości **NULL**).
 - ◆ **ON DELETE SET DEFAULT** — ta klauzula jest rozpoznawana, ale pozostaje niezaimplementowana w silniku InnoDB i powoduje wygenerowanie błędu.
- **ON UPDATE** pozwala na określenie zachowania tabeli potomnej po uaktualnieniu rekordów w tabeli nadrzędnej. W przypadku pominięcia tej klauzuli domyślnie następuje odrzucenie wszelkich operacji wstawienia lub uaktualnienia rekordów w tabeli potomnej, które mogłyby doprowadzić do sytuacji, że wartości klucza zewnętrznego byłyby niedopasowane do wartości w indeksie tabeli nadrzędnej.

Ponadto, odrzucane będą uaktualnienia wartości indeksu tabeli nadrzędnej, do których odwołują się rekordy w tabeli potomnej. Wartości możliwe do zdefiniowania dla *akcji* są takie same jak dla `ON DELETE` i mają podobne działanie.

Aby zdefiniować relację klucza zewnętrznego, należy stosować się do przedstawionych poniżej wskazówek:

- Tabela potomna musi mieć indeks, w którym kolumny klucza zewnętrznego zostaną wymienione jako jej pierwsze kolumny. Tabela nadrzędna także musi mieć indeks, w którym kolumny wymienione w klauzuli `REFERENCES` zostaną podane jako jej pierwsze kolumny. (Innymi słowy, kolumny klucza muszą być zindeksowane w tabelach po obu stronach relacji klucza zewnętrznego). Najpierw trzeba wyraźnie utworzyć indeks w kolumnie nadrzędnej, a dopiero później zdefiniować związek klucza zewnętrznego. W tabeli potomnej silnik InnoDB automatycznie tworzy indeks oparty na kolumnach klucza zewnętrznego, o ile zapytanie `CREATE TABLE` nie zawiera tego rodzaju indeksu. W pewnych sytuacjach to niewątpliwie ułatwia tworzenie zapytań `CREATE TABLE`. Jednak automatycznie utworzony indeks jest nieunikalny i obejmuje tylko kolumny klucza zewnętrznego. Jeżeli chcesz użyć indeksu typu `PRIMARY KEY` bądź `UNIQUE` lub jeśli powinien zawierać jeszcze inne kolumny niż wymienione w kluczu zewnętrznym, to musisz samodzielnie zdefiniować indeks w tabeli potomnej.
- Odpowiadające sobie kolumny w indeksach znajdujących się w tabeli nadrzędnej i potomnej muszą mieć zgodne typy danych. Na przykład, nie można próbować dopasować kolumny typu `INT` do `CHAR`. Odpowiadające sobie kolumny muszą mieć również tę samą długość. Ponadto, odpowiadające sobie kolumny liczb całkowitych muszą mieć tę samą wielkość, obie muszą też przechowywać liczby ze znakiem lub bez znaku (`UNSIGNED`).
- W relacji klucza zewnętrznego nie ma możliwości indeksowania prefiksów kolumn ciągów tekstowych. (Oznacza to, że w przypadku kolumn przechowujących ciągi tekstowe konieczne jest indeksowanie całych kolumn, a nie tylko ich prefiksów).

W rozdziale 1., zatytułowanym „Rozpoczęcie pracy z MySQL”, dla projektu ocen uczniów utworzyliśmy table, które miały proste relacje klucza zewnętrznego. Teraz przeanalizujemy znacznie bardziej skomplikowany przykład. Rozpoczniemy od utworzenia tabel o nazwach `parent` i `child` w taki sposób, aby tabela `child` zawierała klucz zewnętrzny odwołujący się do kolumny `par_id` w tabeli `parent`:

```
CREATE TABLE parent
(
    par_id    INT NOT NULL,
    PRIMARY KEY (par_id)
) ENGINE = INNODB;
CREATE TABLE child
(
    par_id    INT NOT NULL,
    child_id  INT NOT NULL,
    PRIMARY KEY (par_id, child_id),
```



```
FOREIGN KEY (par_id) REFERENCES parent (par_id)
ON DELETE CASCADE
ON UPDATE CASCADE
) ENGINE = INNODB;
```

W omawianym przykładzie klucz zewnętrzny używa `ON DELETE CASCADE` w celu określenia, że po usunięciu rekordu w tabeli `parent` serwer MySQL powinien również automatycznie usunąć z tabeli `child` rekordy o dopasowanej wartości `par_id`. Klauzula `ON UPDATE CASCADE` wskazuje, że jeśli nastąpi zmiana wartości `par_id` w rekordzie nadrzędnym, serwer MySQL powinien zmienić wszystkie dopasowane wartości `par_id` w tabeli `child`.

Następnie wstawiamy kilka rekordów do tabeli `parent` oraz kilka do tabeli `child`.

Wstawiane rekordy mają powiązane ze sobą wartości klucza:

```
mysql> INSERT INTO parent (par_id) VALUES(1),(2),(3);
mysql> INSERT INTO child (par_id,child_id) VALUES(1,1),(1,2);
mysql> INSERT INTO child (par_id,child_id) VALUES(2,1),(2,2),(2,3);
mysql> INSERT INTO child (par_id,child_id) VALUES(3,1);
```

Wykonanie powyższych zapytań powoduje otrzymanie przedstawionej poniżej zawartości tabel, gdzie każda wartość `par_id` w tabeli `child` jest dopasowana do wartości `par_id` w tabeli `parent`:

```
mysql> SELECT * FROM parent;
```

```
+-----+
| par_id |
+-----+
|      1 |
|      2 |
|      3 |
+-----+
```

```
mysql> SELECT * FROM child;
```

```
+-----+-----+
| par_id | child_id |
+-----+-----+
|      1 |         1 |
|      1 |         2 |
|      2 |         1 |
|      2 |         2 |
|      2 |         3 |
|      3 |         1 |
+-----+-----+
```

Aby przekonać się, że silnik InnoDB faktycznie wymusza zachowanie relacji klucza podczas wstawiania danych, spróbuj do tabeli `child` wstawić rekord, którego wartość `par_id` nie istnieje w tabeli `parent`:

```
mysql> INSERT INTO child (par_id,child_id) VALUES(4,1);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key
constraint fails (`sampdb`.`child`, CONSTRAINT `child_ibfk_1` FOREIGN
KEY (`par_id`) REFERENCES `parent` (`par_id`) ON DELETE CASCADE
ON UPDATE CASCADE)
```

W celu przetestowania kaskadowej operacji usunięcia przekonajmy się, co będzie po usunięciu rekordu w tabeli nadrzędnej:

```
mysql> DELETE FROM parent WHERE par_id = 1;
```

MySQL usuwa rekord z tabeli parent:

```
mysql> SELECT * FROM parent;
```

```
+-----+
| par_id |
+-----+
|      2 |
|      3 |
+-----+
```

Ponadto, operacja DELETE jest kaskadowo przekazywana do tabeli child:

```
mysql> SELECT * FROM child;
```

```
+-----+-----+
| par_id | child_id |
+-----+-----+
|      2 |        1 |
|      2 |        2 |
|      2 |        3 |
|      3 |        1 |
+-----+-----+
```

W celu przetestowania kaskadowej operacji uaktualnienia przekonajmy się, co będzie po uaktualnieniu rekordu w tabeli nadrzędnej:

```
mysql> UPDATE parent SET par_id = 100 WHERE par_id =2;
```

```
mysql> SELECT * FROM parent;
```

```
+-----+
| par_id |
+-----+
|      3 |
|     100 |
+-----+
```

```
mysql> SELECT * FROM child;
```

```
+-----+-----+
| par_id | child_id |
+-----+-----+
|      3 |        1 |
|     100 |        1 |
|     100 |        2 |
|     100 |        3 |
+-----+-----+
```

Powyższe zapytania pokazują, w jaki sposób zorganizować operacje usunięcia lub uaktualnienia rekordów tabeli parent, aby kaskadowo spowodowały wykonanie operacji usunięcia lub uaktualnienia wszystkich odpowiadających im rekordów tabeli child. Klauzule ON DELETE i ON UPDATE pozwalają na przeprowadzenie tego rodzaju akcji. Na przykład, jedną z możliwości jest pozostawienie rekordów w tabeli child i jednocześnie przypisanie wartości NULL ich kolumnom klucza zewnętrznego. Takie rozwiązanie wymaga wprowadzenia kilku zmian w definicji tabeli child:

- Użycie ON DELETE SET NULL zamiast ON DELETE CASCADE. W ten sposób silnik InnoDB zostanie poinformowany o konieczności przypisania wartości NULL kolumnie klucza zewnętrznego (par_id), zamiast po prostu usunąć rekordy.

- Użycie `ON UPDATE SET NULL` zamiast `ON UPDATE CASCADE`. W ten sposób silnik InnoDB zostanie poinformowany o konieczności przypisania wartości `NULL` kolumnie klucza zewnętrznego (`par_id`) po uaktualnieniu dopasowanych rekordów.
- Początkowa definicja tabeli `child` definiuje `par_id` jako `NOT NULL`. Takie rozwiązanie nie będzie działało wraz z `ON DELETE SET NULL` lub `ON UPDATE SET NULL`. Definicje kolumn muszą być zmienione w taki sposób, aby dopuszczały przechowywanie wartości `NULL`.
- Początkowa definicja tabeli `child` definiuje również kolumnę `par_id` jako część indeksu typu `PRIMARY KEY`. Jednak indeks wymienionego typu nie może zawierać wartości `NULL`. Zmiana kolumny `par_id` pozwalająca na przechowywanie wartości `NULL` pociąga za sobą konieczność zmiany indeksu z typu `PRIMARY KEY` na `UNIQUE`. Indeks typu `UNIQUE` wymusza stosowanie unikalnych wartości, choć z wyjątkiem `NULL`, które mogą wielokrotnie wystąpić w indeksie.

Aby przekonać się, jaki będzie efekt wprowadzonych zmian, ponownie utworzymy tabelę `parent` za pomocą początkowej definicji, a następnie wstawimy do niej pewne rekordy. Kolejnym krokiem jest utworzenie tabeli `child` na podstawie nowej, przedstawionej poniżej definicji:

```
CREATE TABLE child
(
  par_id    INT NULL,
  child_id INT NOT NULL,
  UNIQUE (par_id, child_id),
  FOREIGN KEY (par_id) REFERENCES parent (par_id)
  ON DELETE SET NULL
  ON UPDATE SET NULL
) ENGINE = INNODB;
```

Nowa tabela `child` zachowuje się podobnie do utworzonej za pomocą początkowej definicji. Oznacza to, że pozwala na wstawienie rekordów, dla których wartości `par_id` znajdują się w tabeli nadrzędnej, ale uniemożliwia wstawienie rekordów o wartościach `par_id` nieistniejących w tabeli nadrzędnej:

```
mysql> INSERT INTO child (par_id,child_id) VALUES(1,1),(1,2);
mysql> INSERT INTO child (par_id,child_id) VALUES(2,1),(2,2),(2,3);
mysql> INSERT INTO child (par_id,child_id) VALUES(3,1);
mysql> INSERT INTO child (par_id,child_id) VALUES(4,1);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key
constraint fails ('sampdb`.`child`, CONSTRAINT `child_ibfk_1` FOREIGN
KEY (`par_id`) REFERENCES `parent` (`par_id`) ON DELETE SET NULL
ON UPDATE SET NULL)
```

To jedyna różnica dotycząca wstawiania rekordów. Ponieważ kolumna `par_id` została teraz zdefiniowana jako dopuszczająca wartości `NULL`, do tabeli potomnej można bez problemów wstawiać rekordy zawierające wartości `NULL` i to nie spowoduje wygenerowania żadnego błędu. Inne zachowanie zaobserwujemy także podczas usuwania rekordu w tabeli

nadrzędnej. Spróbuj usunąć rekord w tabeli nadrzędnej, a następnie sprawdź zawartość tabeli potomnej i przekonaj się, co się dzieje:

```
mysql> DELETE FROM parent WHERE par_id = 1;
mysql> SELECT * FROM child;
```

par_id	child_id
NULL	1
NULL	2
2	1
2	2
2	3
3	1

W tym przypadku w tabeli potomnej rekord o wartości 1 kolumny `par_id` nie został usunięty. Zamiast tego kolumnie `par_id` przypisano wartość `NULL`, zgodnie z definicją ograniczenia `ON DELETE SET NULL`.

Uaktualnienie rekordu nadrzędnego ma podobny efekt: `mysql> UPDATE parent SET par_id = 100 WHERE par_id = 2;`

```
mysql> SELECT * FROM child;
```

par_id	child_id
NULL	1
NULL	1
NULL	2
NULL	2
NULL	3
3	1

Aby przekonać się, jakie klucze zewnętrzne znajdują się w tabeli InnoDB, wykonaj zapytanie `SHOW CREATE TABLE`.

Jeżeli wystąpi błąd w trakcie próby utworzenia tabeli z kluczem zewnętrznym, wykonaj zapytanie `SHOW ENGINE INNODB STATUS` i zapoznaj się z pełnym komunikatem błędu.

2.14. Wyszukiwanie pełnego tekstu

MySQL ma możliwość przeprowadzania wyszukiwania pełnego tekstu, co pozwala na znalezienie słów lub wyrażeń bez konieczności stosowania operacji dopasowania wzorca. Istnieją trzy rodzaje wyszukiwania pełnego tekstu:

- Wyszukiwanie w języku naturalnym (domyślne). MySQL rozbija szukany ciąg tekstowy na słowa, a następnie wyszukuje rekordy zawierające te słowa.
- Wyszukiwanie w trybie boolowskim. Słowa w szukanym ciągu tekstowym mogą zawierać znaki modyfikujące wskazujące na pewne wymagania, na przykład dane słowo powinno lub nie powinno znajdować się w dopasowanych rekordach bądź też rekordy muszą zawierać dokładnie wskazane wyrażenie.

- Wyszukiwanie rozszerzone o wynik zapytania. Ten rodzaj wyszukiwania występuje w dwóch fazach. Pierwsza faza to wyszukiwanie w języku naturalnym. Następnie przeprowadzane jest drugie wyszukiwanie z użyciem początkowego szukanego ciągu tekstowego w połączeniu z najbardziej dopasowanymi rekordami z pierwszego wyszukiwania. W ten sposób następuje rozszerzenie operacji wyszukiwania na podstawie założenia, że słowa powiązane z początkowym szukanym ciągiem tekstowym dopasują odpowiednie rekordy, które nie zostały wcześniej dopasowane przez początkowy szukany ciąg tekstowy.

Możliwość wyszukiwania pełnego tekstu zostaje włączona dla danej tabeli przez utworzenie specjalnego rodzaju indeksu o następujących cechach charakterystycznych:

- Wyszukiwanie pełnego tekstu opiera się na indeksach typu FULLTEXT. W MySQL 5.5 wymienione indeksy mogą być tylko dla tabel MyISAM. W wersji 5.6 serwera MySQL wprowadzono obsługę wyszukiwania pełnego tekstu także dla tabel InnoDB. Tutaj pozostaniemy jednak przy MyISAM, ponieważ możesz jeszcze nie używać MySQL w wersji 5.6. W indeksie typu FULLTEXT mogą znajdować się jedynie kolumny typów CHAR, VARCHAR i TEXT.
- Często występujące słowa są ignorowane w wyszukiwaniu pełnego tekstu, przy czym „często występujące” oznacza „obecne w przynajmniej połowie rekordów”. Warto o tym szczególnie pamiętać podczas tworzenia tabeli testowej do prowadzenia eksperymentów z indeksami typu FULLTEXT. Upewnij się, że do tabeli zostały wstawione co najmniej trzy rekordy. Jeżeli tabela ma tylko jeden lub dwa rekordy, każde słowo wystąpi w przynajmniej połowie rekordów i nigdy nie otrzymasz wyników!
- Baza danych ma listę pewnych słów w języku angielskim, takich jak *the*, *after* i *other*, które są nazywane słowami przestankowymi i zawsze są ignorowane.
- Zbyt krótkie słowa również są ignorowane. Domyślnie, „zbyt krótkie” oznacza mniej niż cztery znaki, ale możesz zmienić to ustawienie w serwerze i określić inną minimalną długość słowa. (Zapoznaj się z punktem 2.14.4, zatytułowanym „Konfiguracja silnika wyszukiwania pełnego tekstu”).
- Słowa są definiowane jako sekwencje znaków zawierających litery, cyfry, apostrofy i znaki podkreślenia. Oznacza to, że ciąg tekstowy w stylu „czarno-biały” jest uznawany za składający się z dwóch słów: czarno i biały. Zwykle operacja wyszukiwania pełnego tekstu powoduje dopasowanie pełnych słów, a nie ich fragmentów. Silnik FULLTEXT uznaje rekord za dopasowany do szukanego ciągu tekstowego, jeśli zawiera dowolne ze słów wspomnianego ciągu tekstowego. Jeżeli używasz wyszukiwania pełnego tekstu w trybie boolowskim, możesz nałożyć dodatkowe ograniczenie, na przykład konieczność wystąpienia wszystkich słów (w dowolnej kolejności lub w dokładnie takiej, jaka została podana w szukanym ciągu tekstowym — to drugie oznacza wyszukanie wyrażenia). W wyszukiwaniu pełnego tekstu w trybie boolowskim istnieje również możliwość dopasowania rekordów *niezawierających* pewnych słów lub dodania modyfikatora w postaci znaku

wieloznacznego i tym samym dopasowania wszystkich słów rozpoczynających się danym prefiksem.

- Indeks typu FULLTEXT może być utworzony dla jednej lub wielu kolumn. Jeżeli obejmuje wiele kolumn, wyszukiwanie oparte na indeksie będzie jednocześnie przeszukiwało wszystkie kolumny. Wadą tego rozwiązania jest to, że w trakcie wyszukiwania trzeba podać listę kolumn odpowiadającą dokładnie zestawowi kolumn, które dopasowały pewien indeks typu FULLTEXT. Na przykład, jeśli czasami chcesz przeszukać kolumnę ko11, innym razem ko12, a jeszcze innym ko11 i ko12, to musisz utworzyć trzy indeksy: po jednym dla każdej kolumny i jeden obejmujący obie kolumny.

Przedstawione poniżej przykłady pokazują, jak używać wyszukiwania pełnego tekstu przez utworzenie indeksów typu FULLTEXT, a następnie wykonywanie względem nich zapytań z użyciem operatora MATCH. Skrypt tworzący tabelę i umieszczane w niej przykładowe dane znajduje się w katalogu *full-text* dystrybucji *sampdb*.

Utworzenie indeksu typu FULLTEXT następuje praktycznie w taki sam sposób jak innych indeksów: można go zdefiniować w zapytaniu CREATE TABLE w chwili tworzenia tabeli lub dodać później za pomocą zapytania ALTER TABLE bądź CREATE INDEX. Ponieważ indeks typu FULLTEXT wymaga użycia tabel MyISAM, możesz wykorzystać zaletę jednej z właściwości silnika MyISAM, o ile stworzysz nową tabelę do użycia dla operacji wyszukiwania FULLTEXT. Wczytanie tabeli nastąpi znacznie szybciej, jeśli najpierw wypełnisz ją danymi, a dopiero później dodasz indeksy, zamiast wstawiać dane do już zindeksowanej tabeli. Przyjmijmy założenie, że masz plik danych o nazwie *apothegm.txt* zawierający sławne powiedzenia i autorów tych wypowiedzi:

Aeschylus	Time as he grows old teaches many lessons
Alexander Graham Bell	Mr. Watson, come here. I want you!
Benjamin Franklin	It is hard for an empty bag to stand upright
Benjamin Franklin	Little strokes fell great oaks
Benjamin Franklin	Remember that time is money
Miguel de Cervantes	Bell, book, and candle
Proverbs 15:1	A soft answer turneth away wrath
Theodore Roosevelt	Speak softly and carry a big stick
William Shakespeare	But, soft! what light through yonder window breaks?
Robert Burton	I light my candle from their torches.

Jeżeli chcesz oddzielnie lub razem wyszukać wyrażenie i autora, to konieczne jest oddzielne zindeksowanie każdej z kolumn oraz utworzenie indeksu obejmującego obie kolumny. Utworzyć, wypełnić i zindeksować tabelę o nazwie *apothegm* możesz w następujący sposób:

```
CREATE TABLE apothegm (attribution VARCHAR(40), phrase TEXT) ENGINE=MyISAM;
LOAD DATA LOCAL INFILE 'apothegm.txt' INTO TABLE apothegm;
ALTER TABLE apothegm
  ADD FULLTEXT (phrase),
  ADD FULLTEXT (attribution),
  ADD FULLTEXT (phrase, attribution);
```

2.14.1. Wyszukiwanie pełnego tekstu w języku naturalnym

Po przygotowaniu tabeli wyszukiwanie pełnego tekstu w języku naturalnym można przeprowadzić za pomocą operatora MATCH wraz z nazwą przeszukiwanej kolumny lub kolumn oraz funkcji AGAINST() wskazującej szukany ciąg tekstowy, na przykład:

```
mysql> SELECT * FROM apothegm WHERE MATCH(attribution) AGAINST('roosevelt');
```

attribution	phrase
Theodore Roosevelt	Speak softly and carry a big stick

```
mysql> SELECT * FROM apothegm WHERE MATCH(phrase) AGAINST('time');
```

attribution	phrase
Benjamin Franklin	Remember that time is money
Aeschylus	Time as he grows old teaches many lessons

```
mysql> SELECT * FROM apothegm WHERE MATCH(attribution, phrase)
-> AGAINST('bell');
```

attribution	phrase
Alexander Graham Bell	Mr. Watson, come here. I want you!
Miguel de Cervantes	Bell, book, and candle

W ostatnim przykładzie zwróć uwagę, jak zapytanie wyszukuje rekordy zawierające szukane słowo w innych kolumnach, co demonstrowa możliwości silnika FULLTEXT w zakresie jednoczesnego wyszukiwania wielu kolumn. Zwróć także uwagę na kolejność kolumn w zapytaniu (attribution, phrase). Różni się ona od kolejności, w której te kolumny zostały podane podczas tworzenia indeksu (phrase, attribution), co pokazuje, że kolejność nie ma znaczenia. Ważne jest, aby indeks typu FULLTEXT zawierał dokładnie wymienione nazwy kolumn.

Aby jedynie sprawdzić, ile rekordów zostanie dopasowanych, użyj funkcji COUNT(*):

```
mysql> SELECT COUNT(*) FROM apothegm WHERE MATCH(phrase) AGAINST('time');
```

COUNT(*)
2

Gdy w klauzuli WHERE zostanie użyte wyrażenie MATCH, rekordy danych wyjściowych wyszukiwania pełnego tekstu w języku naturalnym są ułożone w malejącej kolejności trafności. Wartość określająca trafność to dodatnia liczba zmiennoprzecinkowa, zero oznacza „brak związku, nietrafiony”. Aby wyświetlić wspomniane wartości liczbowe, użyj wyrażenia MATCH na liście kolumn danych wyjściowych:

```
mysql> SELECT phrase, MATCH(phrase) AGAINST('time') AS relevance
-> FROM apothegm;
```

phrase	relevance
Time as he grows old teaches many lessons	1.3253291845321655
Mr. Watson, come here. I want you!	0
It is hard for an empty bag to stand upright	0
Little strokes fell great oaks	0
Remember that time is money	1.340062141418457
Bell, book, and candle	0
A soft answer turneth away wrath	0
Speak softly and carry a big stick	0
But, soft! what light through yonder window breaks?	0
I light my candle from their torches.	0

Wyszukiwanie w języku naturalnym znajduje rekordy zawierające dowolne z szukanych słów. Dlatego też przedstawione poniżej zapytanie zwróci rekordy zawierające słowo *hard* lub *soft*:

```
mysql> SELECT * FROM apothegm WHERE MATCH(phrase)
-> AGAINST('hard soft');
```

attribution	phrase
Benjamin Franklin	It is hard for an empty bag to stand upright
Proverbs 15:1	A soft answer turneth away wrath
William Shakespeare	But, soft! what light through yonder window breaks?

Tryb języka naturalnego jest domyślnym trybem wyszukiwania pełnego tekstu. Aby wyraźnie wskazać ten tryb, należy po szukanych ciągu tekstowym dodać słowa kluczowe `IN NATURAL LANGUAGE MODE`. Poniższe zapytanie przeprowadza taką samą operację wyszukiwania jak w poprzednim przykładzie:

```
SELECT * FROM apothegm WHERE MATCH(phrase)
AGAINST('hard soft' IN NATURAL LANGUAGE MODE);
```

2.14.2. Wyszukiwanie pełnego tekstu w trybie boolowskim

Większą kontrolę nad dopasowaniem wielu słów można uzyskać dzięki użyciu wyszukiwania pełnego tekstu w trybie boolowskim. Ten rodzaj wyszukiwania jest przeprowadzany po dodaniu `IN BOOLEAN MODE` w funkcji `AGAINST()` po szukanych ciągu tekstowym. Wyszukiwanie pełnego tekstu w trybie boolowskim charakteryzuje się następującymi cechami:

- Reguła 50% jest ignorowana. Operacja wyszukiwania dopasowuje słowa nawet wtedy, gdy występują w więcej niż w połowie rekordów.
- Nie ma sortowania względem trafności.
- Wyszukiwanie może wymagać, aby wszystkie słowa w wyrażeniu znajdowały się w określonej kolejności. Aby dopasować wyrażenie, musisz je ująć w cudzysłów. Dopasowanie zostanie znalezione w wierszach zawierających te same słowa w kolejności, w której zostały wymienione w wyrażeniu:


```
mysql> SELECT * FROM apothegm
-> WHERE MATCH(attribution, phrase)
-> AGAINST('bell book and candle' IN BOOLEAN MODE);
```

attribution	phrase
Miguel de Cervantes	Bell, book, and candle

- Istnieje możliwość przeprowadzenia wyszukiwania pełnego tekstu w trybie boolowskim względem kolumn, które nie są częścią indeksu typu FULLTEXT, choć taka operacja będzie znacznie wolniejsza niż oparta na zindeksowanych kolumnach.

W przypadku wyszukiwania boolowskiego można zastosować modyfikatory słów znajdujących się w szukanym ciągu tekstowym. Znak plus lub minus umieszczony przed słowem oznacza, że powinno lub nie powinno się ono znajdować w dopasowanych rekordach. Na przykład, szukany ciąg tekstowy `bell` spowoduje dopasowanie rekordów zawierających słowo `bell`, natomiast szukany ciąg tekstowy `+bell -candle` w trybie boolowskim powoduje dopasowanie jedynie rekordów zawierających słowo `bell` i jednocześnie niezawierających słowa `candle`.

```
mysql> SELECT * FROM apothegm
-> WHERE MATCH(attribution, phrase)
-> AGAINST('bell');

mysql> SELECT * FROM apothegm
-> WHERE MATCH(attribution, phrase)
-> AGAINST('+bell -candle' IN BOOLEAN MODE);
```

attribution	phrase
Alexander Graham Bell	Mr. Watson, come here. I want you!
Miguel de Cervantes	Bell, book, and candle

attribution	phrase
Alexander Graham Bell	Mr. Watson, come here. I want you!

Gwiazdka na końcu jest traktowana jako znak wieloznaczny, a więc zostaną dopasowane wszystkie rekordy zawierające słowa rozpoczynające się od dopasowanego szukanego słowa. Na przykład, `soft*` powoduje dopasowanie `soft`, `softly`, `softness` itd.

```
mysql> SELECT * FROM apothegm WHERE MATCH(phrase)
-> AGAINST('soft*' IN BOOLEAN MODE);
```

attribution	phrase
Proverbs 15:1	A soft answer turneth away wrath
William Shakespeare	But, soft! what light through yonder window breaks?
Theodore Roosevelt	Speak softly and carry a big stick

Jednak funkcja znaku wieloznacznego nie może być użyta w celu dopasowania słów krótszych niż minimalna długość słowa indeksu.

W dodatku C, zatytułowanym „Przewodnik po operatorach i funkcjach”, znajdziesz opis operatora MATCH wraz z pełną listą obsługiwanych trybów boolowskich.

Słowa przestankowe są ignorowane, podobnie jak w wyszukiwaniu w języku naturalnym, nawet po ich zaznaczeniu jako wymaganych. Wyszukiwanie +Alexander +the +great spowoduje dopasowanie rekordów zawierających słowa Alexander i great, ale zignoruje słowo the.

2.14.3. Wyszukiwanie rozszerzone o wynik zapytania

Wyszukiwanie pełnego tekstu rozszerzone o wynik zapytania przeprowadza operację dwufazową. Początkowe wyszukiwanie jest przeprowadzane jak w przypadku wyszukiwania w języku naturalnym. Następnie rekordy ocenione jako najtrafniejsze w wynikach pierwszego wyszukiwania są używane w drugiej fazie. Słowa ze wspomnianych rekordów są używane wraz z początkowym szukanym ciągiem tekstowym w celu przeprowadzenia drugiej operacji wyszukiwania. Ponieważ zbiór szukanых terminów jest większy, wynik powinien zawierać rekordy, które nie zostały znalezione w pierwszej fazie, ale są powiązane z początkowym szukanym ciągiem tekstowym.

Aby przeprowadzić tego rodzaju wyszukiwanie, należy dodać słowa kluczowe WITH QUERY EXPANSION po szukanym ciągu tekstowym. Poniższy przykład prezentuje tego typu wyszukiwanie. Pierwsze zapytanie to wyszukiwanie w języku naturalnym. Z kolei drugie zapytanie pokazuje wyszukiwanie rozszerzone o wynik wcześniejszego zapytania. Dane wyjściowe drugiego zapytania zawierają dodatkowy rekord, w którym nie znajduje się żadne słowo z początkowo podanego szukanego ciągu tekstowego. Ten rekord został znaleziony, ponieważ zawiera słowo candle, istniejące w jednym z rekordów znalezionych przez wyszukiwanie w języku naturalnym.

```
mysql> SELECT * FROM apothegm
-> WHERE MATCH(attribution, phrase)
-> AGAINST('bell book');
+-----+-----+
| attribution | phrase |
+-----+-----+
| Miguel de Cervantes | Bell, book, and candle |
| Alexander Graham Bell | Mr. Watson, come here. I want you! |
+-----+-----+
mysql> SELECT * FROM apothegm
-> WHERE MATCH(attribution, phrase)
-> AGAINST('bell book' WITH QUERY EXPANSION);
+-----+-----+
| attribution | phrase |
+-----+-----+
| Miguel de Cervantes | Bell, book, and candle |
| Alexander Graham Bell | Mr. Watson, come here. I want you! |
| Robert Burton | I light my candle from their torches. |
+-----+-----+
```

2.14.4. Konfiguracja silnika wyszukiwania pełnego tekstu

Wiele parametrów wyszukiwania pełnego tekstu można skonfigurować i zmodyfikować przez odpowiednie ustawienie zmiennych systemowych. Zmienne `ft_min_word_len` i `ft_max_word_len` określają najkrótsze i najdłuższe słowa do zindeksowania w indeksach typu FULLTEXT. Słowa o wielkości poza zakresem definiowanym przez dwie wymienione zmienne są ignorowane podczas tworzenia indeksu typu FULLTEXT. Domyślne wartości minimalna i maksymalna to odpowiednio 4 i 84.

Przyjmujemy założenie, że chcesz zmienić minimalną długość słowa z 4 znaków na 3. Możesz to zrobić w następujący sposób:

1. Uruchom serwer wraz ze zmienną `ft_min_word_len`, której przypisano wartość 3. Aby zagwarantować stosowanie tego ustawienia w trakcie każdego uruchomienia serwera, zmienną najlepiej umieścić w pliku opcji, takim jak `/etc/my.cnf`:

```
[mysqld]  
ft_min_word_len=3
```
2. Dla wszystkich istniejących tabel posiadających indeksy typu FULLTEXT konieczne jest ich odbudowanie. Wprawdzie możesz usunąć indeks i ponownie go dodać, ale łatwiejszym i wystarczającym rozwiązaniem jest przeprowadzanie szybkiej operacji naprawy:

```
REPAIR TABLE nazwa_tabeli QUICK;
```
3. Dla wszystkich nowych indeksów typu FULLTEXT, które utworzysz po zmianie parametrów, jego nowa wartość będzie użyta automatycznie.

Więcej informacji na temat ustawiania zmiennych systemowych znajdziesz w dodatku D, zatytułowanym „Przewodnik po zmiennych systemowych, stanu i użytkownika”. Z kolei informacje szczegółowe dotyczące używania pliku opcji znajdziesz w dodatku F, zatytułowanym „Przewodnik po programie MySQL”.

Uwaga

Jeżeli używasz narzędzia `mysamchk` do odbudowy indeksów w tabeli zawierającej jakiegokolwiek indeks typu FULLTEXT, zapoznaj się z uwagami dotyczącymi tego typu indeksów. Wspomniane uwagi przedstawiono w opisie narzędzia `mysamchk` w dodatku F, zatytułowanym „Przewodnik po programie MySQL”.

Typy danych

Praktycznie wszystko, co robisz w MySQL, z definicji w pewien sposób wymaga użycia danych, ponieważ przeznaczeniem systemu zarządzania bazą danych jest zarządzanie danymi. Nawet proste zapytanie w stylu `SELECT 1` oznacza obliczenie wartości wyrażenia w celu wygenerowania danych w postaci liczby całkowitej.

Każda wartość danych w MySQL ma swój typ. Na przykład, `37.4` jest liczbą, a `'abc'` ciągiem tekstowym. Czasami typy danych są wyraźnie zdefiniowane. I tak, w trakcie wykonywania zapytania `CREATE TABLE` wskazujesz typ danych dla każdej kolumny tworzonej w danej tabeli:

```
CREATE TABLE mytbl
(
    int_col INT,           # Kolumna zawierająca wartość w postaci liczby całkowitej.
    str_col CHAR(20),      # Kolumna zawierająca wartość w postaci ciągu tekstowego.
    date_col DATE          # Kolumna zawierająca wartość w postaci daty.
);
```

W innych przypadkach typ danych nie musi być wyraźnie podany. Tego rodzaju sytuacja zachodzi, gdy odwołujesz się do dosłownych wartości w wyrażeniu, przekazujesz wartości funkcji lub używasz wartości zwróconych przez funkcję. Poniższe zapytanie `INSERT` pokazuje wszystkie wymienione sytuacje:

```
INSERT INTO mytbl (int_col,str_col,date_col)
VALUES(14,CONCAT('a','b'),20130815);
```

Powyższe zapytanie przeprowadza następujące operacje opierające się na typach danych:

- Wartość w postaci liczby całkowitej `14` jest przypisywana kolumnie `int_col`, przechowującej liczby całkowite.
- Ciągi tekstowe `'a'` i `'b'` są przekazywane funkcji `CONCAT()`, która łączy wymienione ciągi tekstowe, a następnie powstały ciąg tekstowy `'ab'` przypisuje kolumnie `str_col`, przechowującej ciągi tekstowe.
- Wartość w postaci liczby całkowitej `20130815` jest przypisywana kolumnie `date_col`, przechowującej datę. W przypadku tego przypisania można dostrzec pewną odmienność, ale wartość w postaci liczby całkowitej może być zinterpretowana

jako wartość daty. Dlatego też MySQL automatycznie przeprowadza operację konwersji typu liczby całkowitej 20130815 na datę '2013-08-15'.

Aby efektywnie używać MySQL, konieczne jest dokładne zrozumienie sposobu, w jaki MySQL obsługuje dane. W tym rozdziale omówiono typy danych, których możesz używać:

- Ogólne kategorie wartości danych, które MySQL potrafi obsługiwać, łącznie z wartością NULL.
- Charakterystyczne typy danych dostarczane przez MySQL kolumnom tabel i właściwości charakteryzujące poszczególne typy danych. Niektóre typy danych w MySQL są bardziej ogólne, na przykład BLOB. Z kolei inne zachowują się w specjalny sposób, który należy zrozumieć, aby uniknąć późniejszych niespodzianek. Do takich typów danych zaliczamy TIMESTAMP i typy liczb całkowitych, które mają atrybut AUTO_INCREMENT.

W rozdziale zostaną także poruszone kwestie pojawiające się podczas pracy ze wspomnianymi typami danych:

- Jak tryb SQL serwera wpływa na traktowanie nieprawidłowych wartości danych oraz jak używać trybu „ściśłego” do odrzucania niepoprawnych wartości.
- Jak wygenerować sekwencje i z nimi pracować.
- Jakie są reguły stosowane przez MySQL podczas obliczania wartości wyrażeń. W wyrażeniach masz możliwość użycia szerokiej gamy operatorów i funkcji w celu pobierania, wyświetlania i przeprowadzania operacji na danych. W trakcie obliczania wartości wyrażenia stosowane są reguły regulujące konwersję typu, o ile taką operację trzeba przeprowadzić, gdy wartość jednego typu jest używana w kontekście wymagającym innego typu. Bardzo ważne jest zrozumienie, kiedy następuje konwersja typu oraz jak działa. Niektóre operacje konwersji nie mają sensu, a ich wynikiem są bezsensowne wartości. Przypisanie ciągu tekstowego '13' kolumnie przechowującej liczby całkowite spowoduje wstawienie wartości 13. Jednak przypisanie ciągu tekstowego 'abc' tej samej kolumnie spowoduje wstawienie wartości 0 (lub wystąpienie błędu w przypadku używania ściśłego trybu SQL), ponieważ 'abc' nie jest liczbą. Co gorsza, jeśli przeprowadzisz porównanie, nie znając reguł dotyczących konwersji, wtedy możesz doprowadzić do ogromnych szkód, na przykład uaktualniając i usuwając wszystkie rekordy, choć tak naprawdę operację chciałeś przeprowadzić jedynie na wybranej ich grupie.
- Jak odpowiednio wybrać typy danych dla kolumn tabeli. Bardzo cenna jest umiejętność wyboru najlepszego typu danych dla kolumn podczas tworzenia tabeli. Warto też wiedzieć, kiedy wybrać jeden typ zamiast innego, gdy dla rodzaju przechowywanych wartości odpowiednich jest kilka powiązanych ze sobą typów danych.

Uzupełnieniem przedstawionej w tym rozdziale analizy dotyczącej typów danych, operatorów i funkcji w MySQL są dodatki: B, „Przewodnik po typach danych”, i C, „Przewodnik po operatorach i funkcjach”.

Wiele przykładów przedstawionych w tym rozdziale wykorzystuje zapytania CREATE TABLE i ALTER TABLE. Wymienione zapytania zostały dokładnie omówione w rozdziałach 1., „Rozpoczęcie pracy z MySQL”, i 2., „Użycie MySQL do zarządzania danymi”, a także w dodatku E, „Przewodnik po składni SQL”.

W pewnych przypadkach obsługa danych zależy od sposobu zdefiniowania wartości domyślnych oraz bieżącego trybu SQL. Ogólne informacje dotyczące ustawiania trybu SQL znajdziesz w podrozdziale 2.1, zatytułowanym „Tryby SQL serwera”. Podrozdział 3.2.3, zatytułowany „Definiowanie wartości domyślnych kolumn”, przedstawia temat obsługi wartości domyślnych. Z kolei w podrozdziale 3.3, zatytułowanym „Jak MySQL obsługuje nieprawidłowe wartości danych?”, omówiono tryb ścisły oraz reguły dotyczące traktowania nieprawidłowych danych.

3.1. Kategorie wartości danych

MySQL obsługuje wiele ogólnych kategorii wartości danych. Obejmują one liczby, ciągi tekstowe, wartości daty i godziny, wartości przestrzenne oraz wartość NULL.

3.1.1. Wartości liczbowe

Liczby są wartościami takimi jak 48, 193.62 lub -2.378E12. MySQL potrafi obsługiwać liczby zdefiniowane jako liczby całkowite (pozbawione części po przecinku dziesiętnym), liczby zmiennoprzecinkowe (mogą posiadać część po przecinku dziesiętnym) oraz wartości pól bitowych.

Kiedy podajesz liczbę, nie wolno używać przecinka jako separatora. Na przykład, 12345678.90 to wartość prawidłowa, natomiast 12,345,678.90 to wartość nieprawidłowa.

3.1.1.1. Dokładne i przybliżone wartości liczbowe

MySQL obsługuje zarówno dokładne, jak i przybliżone wartości liczbowe.

Dokładne wartości liczbowe, gdy tylko to możliwe, są używane w takiej postaci, w jakiej zostały zdefiniowane. Przykładami dokładnych wartości są liczby całkowite (0, 14, -382) oraz liczby zmiennoprzecinkowe (0.0, 38.5, -18.247).

Liczby całkowite mogą być podane w formacie dziesiętnym lub szesnastkowym. W formacie dziesiętnym liczba całkowita składa się z sekwencji cyfr pozbawionych przecinka dziesiętnego. Z kolei wartości szesnastkowe są domyślnie traktowane jako ciągi tekstowe, ale w kontekście liczbowym. Stała szesnastkowa jest traktowana jako 64-bitowa liczba całkowita. Na przykład, wartość 0x10 to 16 dziesiętnie. W punkcie 3.1.2, zatytułowanym „Wartości ciągu tekstowego”, omówiono składnię wartości szesnastkowej.

Dokładna wartość wraz z częścią ułamkową składa się z sekwencji cyfr, przecinka dziesiętnego oraz kolejnej sekwencji cyfr. Wspomniane sekwencje cyfr przed przecinkiem dziesiętnym i po nim mogą być puste, ale nie obie jednocześnie.

Wartości przybliżone są przedstawiane w postaci liczb zmiennoprzecinkowych podanych w notacji naukowej wraz z mantysą i wykładnikiem. Tego rodzaju zapis można poznać

po literze e lub E znajdującej się bezpośrednio po liczbie całkowitej lub części ułamkowej, opcjonalnym znaku (+ lub -) oraz wykładniku w postaci liczby całkowitej. Zarówno mantysa, jak i wykładnik mogą mieć znak w dowolnej kombinacji: 1.58E5, -1.58E5, 1.58E-5, -1.58E-5.

Liczby szesnastkowe nie mogą być używane w notacji naukowej, ponieważ litera e rozpoczynająca część wykładnika jest poprawną cyfrą szesnastkową, a tym samym powstaje niejednoznaczność.

Dowolna liczba może być poprzedzona znakiem plus lub minus, wskazując tym samym wartość dodatnią lub ujemną.

Na wspomnianych dokładnych wartościach można przeprowadzać obliczenia bez utraty precyzji, ale z uwzględnieniem ograniczeń nakładanych na tego typu wartości. Na przykład, do kolumny pozwalającej na przechowywanie jedynie dwóch cyfr po przecinku dziesiętnym nie można bez straty precyzji wstawić wartości 1.23456. Obliczenia przeprowadzane na wartościach przybliżonych podlegają błędowi wynikającemu z zaokrąglania.

MySQL oblicza dokładną lub przybliżoną wartość wyrażenia, stosując wymienione poniżej reguły:

- Wyrażenie zawierające jakąkolwiek wartość przybliżoną jest obliczane jako wyrażenie zmiennoprzecinkowe (przybliżone).
- Wyrażenie zawierające jedynie dokładne wartości w postaci liczb całkowitych jest obliczane z użyciem precyzji BIGINT (64-bitowej).
- Wyrażenie zawierające jedynie dokładne wartości, z których pewne mają część ułamkową, używa typu DECIMAL, oferującego 65 cyfr precyzji.
- Jeżeli ciąg tekstowy musi być skonwertowany na liczbę w celu obliczenia wyrażenia, jest konwertowany na wartość zmiennoprzecinkową o podwójnej precyzji. Dlatego też wyrażenie jest przybliżone zgodnie z wcześniejszymi regułami.

3.1.1.2. Wartości pól bitowych

Wartości pól bitowych mogą być zapisane jako *b'wartość'* lub *0bwartość*, gdzie *wartość* składa się z jednej lub więcej cyfr binarnych (0 lub 1). Na przykład, *b'1001'* i *0b1001* dziesiętnie oznaczają liczbę 9.

Wartość BIT w zbiorze wynikowym jest wyświetlana jako binarny ciąg tekstowy, który może nie być poprawnie wyświetlany. Aby skonwertować go na liczbę całkowitą, musisz dodać zero lub użyć funkcji CAST():

```
mysql> SELECT b'1001' + 0, CAST(b'1001' AS UNSIGNED);
+-----+-----+
| b'1001' + 0 | CAST(b'1001' AS UNSIGNED) |
+-----+-----+
|          9 |                9          |
+-----+-----+
```


3.1.2. Wartości ciągu tekstowego

Ciągi tekstowe to wartości takie jak 'Łódzkie, Polska', 'stan pacjenta poprawia się' lub nawet '12345' (która wygląda jak liczba, ale nią nie jest). Wartość ciągu tekstowego najczęściej jest ujmowana w apostrofy lub cudzysłów. Istnieją jednak dwa powody, dla których warto stosować apostrofy:

- Standard SQL wskazuje apostrofy, a tym samym zapytania z ciągami tekstowymi ujętymi w apostrofy są znacznie bardziej przenośne do innych silników bazy danych.
- Jeżeli włączony jest tryb SQL ANSI_QUOTES, MySQL traktuje cudzysłów jako identyfikator cytowania znaku, a nie ciągu tekstowego. Oznacza to, że wartość ujęta w cudzysłów musi odwoływać się do komponentu takiego jak baza danych lub nazwa tabeli. Rozważmy poniższe zapytanie:

```
SELECT "last_name" from president;
```

Po włączeniu trybu ANSI_QUOTES zapytanie pobiera wartości kolumny last_name w tabeli president. Natomiast po wyłączeniu wymienionego trybu zapytanie pobierze dosłowny ciąg tekstowy "last_name" dla każdego rekordu w tabeli president.

W kolejnych przykładach używających cudzysłowu jako identyfikatora cytowania znaku przyjmujemy założenie, że tryb ANSI_QUOTES nie został włączony.

W ciągach tekstowych MySQL rozpoznaje kilka sekwencji sterujących wskazujących znaki specjalne (patrz tabela 3.1). Każda sekwencja rozpoczyna się ukośnikiem, który wskazuje tymczasową zmianę reguł interpretacji znaku. Zwróć uwagę, że bajt NUL nie jest taki sam jak wartość SQL NULL — NUL to bajt o wartości zero, podczas gdy NULL w SQL oznacza brak wartości.

Tabela 3.1. Sekwencje sterujące w ciągach tekstowych

Sekwencja	Opis
\0	NUL (bajt o wartości zero)
\'	Apostrof
\"	Cudzysłów
\b	Backspace
\n	Znak nowego wiersza
\r	Znak powrotu karetki (wysuw wiersza)
\t	Tabulator
\\	Pojedynczy ukośnik
\z	Ctrl+Z (znak EOF w Windows)

Sekwencje sterujące wymienione w tabeli rozróżniają wielkość liter i każdy znak nieujęty w tabeli będzie zinterpretowany w taki sposób, jakby został poprzedzony ukośnikiem. Na przykład, \t oznacza tabulator, natomiast \T to zwykła litera T.

W tabeli 3.1 pokazano, że zarówno apostrof, jak i cudzysłów można poprzedzić ukośnikiem w celu zmiany ich interpretacji. W rzeczywistości masz kilka opcji pozwalających na umieszczenie dosłownych znaków cytowania w ciągach tekstowych:

- Dwukrotne użycie tego samego znaku cytowania, jeśli sam ciąg tekstowy jest cytowany za pomocą danego znaku:

```
'Restauracja Mc'Donalds'
"On powiedział ""Nie idź tam.""
```

- Zacytowanie ciągu tekstowego innymi znakami cytowania. W takim przypadku nie trzeba stosować podwójnych znaków cytowania w ciągu tekstowym:

```
"Restauracja Mc'Donalds"
'On powiedział "Nie idź tam."'
```

- Poprzedzenie znaku cytowania ukośnikiem. Takie rozwiązanie działa niezależnie od znaków cytowania, w które został ujęty ciąg tekstowy:

```
'Restauracja Mc\'Donalds'
"Restauracja Mc\'Donalds"
"On powiedział \"Nie idź tam.\""
'On powiedział \"Nie idź tam.\"'
```

Aby wyłączyć specjalne znaczenie ukośnika i traktować go jak zwykły znak, należy włączyć tryb SQL o nazwie `NO_BACKSLASH_ESCAPES`.

Dwie formy notacji szesnastkowej zapewniają alternatywne rozwiązanie w zakresie użycia znaków cytowania w ciągach tekstowych. Wartości mogą być podawane za pomocą standardowej notacji SQL `X'wartość'`, gdzie *wartość* składa się z pary cyfr szesnastkowych (od 0 do 9 i od a do f). Na przykład, `X'0a'` to 10 dziesiętnie, natomiast `X'ffff'` to dziesiętnie liczba 65535. Początkowy znak `X` i niedziesiętne cyfry liczby szesnastkowej (od a do f) mogą być podane zarówno jako małe, jak i wielkie:

```
mysql> SELECT X'4A', x'4a';
+-----+-----+
| X'4A' | x'4a' |
+-----+-----+
| J     | J     |
+-----+-----+
```

W kontekście ciągu tekstowego para cyfr szesnastkowych jest interpretowana jako 8-bitowa wartość bajtowa w zakresie od 0 do 255, a wynik jest używany jako ciąg tekstowy. W kontekście liczbowym stała szesnastkowa jest traktowana jako liczba. Poniższe zapytanie ilustruje interpretację stałej szesnastkowej w wymienionych kontekstach:

```
mysql> SELECT X'61626364', X'61626364'+0;
+-----+-----+
| X'61626364' | X'61626364'+0 |
+-----+-----+
| abcd        | 1633837924     |
+-----+-----+
```

Wartości szesnastkowe mogą być również zapisane z użyciem przyrostka `0x` poprzedzającego jedną lub więcej cyfr szesnastkowych. Przyrostek `0x` rozróżnia

wielkość liter. Dlatego też 0x0a i 0x0A to prawidłowe wartości szesnastkowe, podczas gdy 0X0a i 0X0A są nieprawidłowe.

Podobnie jak w przypadku notacji X'*wartość*', także wartości 0x są interpretowane jako ciągi tekstowe, ale mogą być używane jako liczby w kontekstach liczbowych:

```
mysql> SELECT 0x61626364, 0x61626364+0;
+-----+-----+
| 0x61626364 | 0x61626364+0 |
+-----+-----+
| abcd       | 1633837924    |
+-----+-----+
```

Notacja X'*wartość*' wymaga parzystej liczby cyfr. Wartość taka jak X'a' jest nieprawidłowa. Jeżeli wartość szesnastkowa zapisana z użyciem notacji 0x ma nieparzystą liczbę cyfr szesnastkowych, MySQL traktuje ją jako poprzedzoną zerem. Na przykład, 0xa jest traktowane jako 0x0a.

3.1.2.1. Typy ciągów tekstowych i obsługa kodowania znaków

Ciągi tekstowe zaliczamy do dwóch ogólnych kategorii: binarnych i niebinarnych.

- Binarny ciąg tekstowy to sekwencja bajtów. Wspomniane bajty są interpretowane bez uwzględniania koncepcji kodowania znaków. Binarny ciąg tekstowy nie ma specjalnych właściwości porównywania lub sortowania. Operacja porównania jest przeprowadzana bajt po bajcie na podstawie liczbowych wartości bajtów. Wszystkie bajty są ważne, także spacje na końcu.
- Niebinarny ciąg tekstowy to sekwencja znaków. Powiązany jest z kodowaniem znaków, które określa znaki dozwolone do stosowania, a także wskazuje sposób, w jaki MySQL interpretuje zawartość ciągu tekstowego. Kodowanie znaków ma zdefiniowaną jedną lub więcej kolejności sortowania. Konkretna kolejność sortowania użyta w ciągu tekstowym określa kolejność znaków w kodowaniu, co ma znaczenie w trakcie operacji porównania. Domyślne kodowanie to latin1, natomiast domyślna kolejność sortowania to latin1_swedish_ci. Spacje na końcu w niebinarnych ciągach tekstowych nie mają znaczenia podczas operacji porównania, choć istnieje tutaj pewien wyjątek. W przypadku typów TEXT, gdy porównanie jest oparte na indeksie, wartości są do końca dopełniane spacjami. Jeżeli do indeksu z unikalnymi wartościami typu TEXT spróbujesz wstawić wartość różniącą się jedynie liczbą spacji na końcu, wówczas nastąpi zgłoszenie błędu wskazującego na powielenie klucza.

Jednostki znaków mają różne wymagania w zakresie pamięci masowej. Kodowanie jednobajtowe, takie jak latin1, używa po jednym bajcie dla każdego znaku. Istnieją jednak wielobajtowe kodowania znaków, w których pewne lub wszystkie znaki wymagają więcej niż tylko jednego bajta. Na przykład, kodowanie Unicode dostępne w MySQL jest kodowaniem wielobajtowym. W kodowaniu ucs2 każdy znak jest opisywany przez dwa bajty. Z kolei utf8 to kodowanie o zmiennej liczbie bajtów na znak — do opisanego znaku używanych

jest od jednego do trzech bajtów. Niektóre kodowania Unicode używają do czterech bajtów na znak. Zapoznaj się z punktem 2.4.3, zatytułowanym „Obsługa Unicode”.

Aby dowiedzieć się, jakie kodowania znaków i kolejności sortowania są dostępne w serwerze, wykonaj dwa przedstawione poniżej zapytania:

```
mysql> SHOW CHARACTER SET;
```

Charset	Description	Default collation	Maxlen
big5	Big5 Traditional Chinese	big5_chinese_ci	2
dec8	DEC West European	dec8_swedish_ci	1
cp850	DOS West European	cp850_general_ci	1
hp8	HP West European	hp8_english_ci	1
koi8r	KOI8-R Relcom Russian	koi8r_general_ci	1
latin1	cp1252 West European	latin1_swedish_ci	1
...			
utf8	UTF-8 Unicode	utf8_general_ci	3
ucs2	UCS-2 Unicode	ucs2_general_ci	2
...			

```
mysql> SHOW COLLATION;
```

Collation	Charset	Id	Default	Compiled	Sortlen
big5_chinese_ci	big5	1	Yes	Yes	1
big5_bin	big5	84		Yes	1
...					
latin1_german1_ci	latin1	5		Yes	1
latin1_swedish_ci	latin1	8	Yes	Yes	1
latin1_danish_ci	latin1	15		Yes	1
latin1_german2_ci	latin1	31		Yes	2
latin1_bin	latin1	47		Yes	1
latin1_general_ci	latin1	48		Yes	1
latin1_general_cs	latin1	49		Yes	1
latin1_spanish_ci	latin1	94		Yes	1
...					

Jak pokazano w danych wyjściowych zapytania `SHOW COLLATION`, każda kolejność sortowania jest powiązana z określonym kodowaniem znaków, które z kolei może zawierać wiele zdefiniowanych kolejności sortowania. Nazwa kolejności sortowania zwykle składa się z nazwy kodowania znaków, języka i dodatkowego przyrostka. Na przykład, `utf8_polish_ci` to kolejność sortowania dla kodowania znaków Unicode `utf8`, w którym porównywanie będzie odbywało się z użyciem reguł języka polskiego, a wielkość liter nie ma znaczenia. Przyrostki w nazwie kolejności sortowania mają następujące znaczenie:

- `_ci` oznacza sortowanie uwzględniające wielkość znaków.
- `_cs` oznacza sortowanie nieuwzględniające wielkości znaków.
- `_bin` oznacza sortowanie binarne. W tym przypadku porównywanie odbywa się na podstawie wartości liczbowych kodów znaku bez odwoływania się do języka. Z tego powodu nazwa sortowania z przyrostkiem `_bin` nie zawiera żadnej nazwy języka, na przykład `latin1_bin` i `utf8_bin`.

Binarne i niebinarne ciągi tekstowe mają różne właściwości sortowania:

- Binarne ciągi tekstowe są przetwarzane bajt po bajcie w operacjach porównania opartych wyłącznie na liczbowej wartości każdego bajta. Dlatego też wydaje się, że w binarnym ciągu tekstowym wielkość liter ma znaczenie ('abc' < 'ABC'). Wynika to jednak z faktu, że mała i wielka litera mają inne liczbowe wartości bajta. W binarnych ciągach tekstowych naprawdę nie ma żadnej notacji wielkości liter. Wspomniana wielkość liter to funkcja kolejności sortowania, które z kolei jest stosowane jedynie względem znaków w niebinarnych ciągach tekstowych.
- Operacja porównania niebinarnego ciągu tekstowego polega na jego przetwarzaniu znak po znaku z uwzględnieniem wartości poszczególnych znaków określonych na podstawie sekwencji kolejności sortowania dla danego kodowania znaków. W wielu kolejnościach sortowania mała i wielka litera mają tę samą wartość sortowania. Dlatego też operacja porównania niebinarnego ciągu tekstowego zwykle nie rozróżnia wielkości liter. To jednak nie dotyczy kolejności sortowania binarnej lub uwzględniającej wielkość liter.

Ponieważ kolejność sortowania jest używana do operacji porównywania i sortowania, wpływa także na wiele innych operacji:

- Operacje porównania przeprowadzane za pomocą operatorów <, <=, =, <>, >=, > i LIKE.
- Operacje sortowania: ORDER BY, MIN() i MAX().
- Operacje grupowania: GROUP BY i DISTINCT.

Aby określić kodowanie znaków lub kolejność sortowania w ciągu tekstowym, należy użyć funkcji CHARSET() lub COLLATION().

Cytowane dosłowne ciągi tekstowe są interpretowane zgodnie z aktualnymi ustawieniami serwera. Domyślne kodowanie to latin1, natomiast domyślna kolejność sortowania to latin1_swedish_ci. Jednak zarówno mysql, jak i inne programy klienckie mogą wykryć i odpowiednio dostosować wymienione ustawienia, jeśli użyto zmiennej środowiskowej LANG lub LC_ALL do konfiguracji ustawień językowych. Zapoznaj się z podpunktem 3.1.2.2, zatytułowanym „Zmienne systemowe dotyczące kodowania znaków”.

Domyślnie MySQL traktuje stałe szesnastkowe jako binarne ciągi tekstowe:

```
mysql> SELECT CHARSET(X'0123'), COLLATION(X'0123');
+-----+-----+
| CHARSET(X'0123') | COLLATION(X'0123') |
+-----+-----+
| binary           | binary              |
+-----+-----+
```

W celu wymuszenia interpretacji dosłownego ciągu tekstowego za pomocą wskazanego kodowania znaków można wykorzystać dwie konwencje zapisu. Pierwsza, stała ciągu tekstowego, może być interpretowana przez wskazane kodowanie znaków po zastosowaniu poniższej notacji, w której *kodowanie* wskazuje nazwę obsługiwanego kodowania znaków;

_kodowanie ciąg_textowy

Notacja *_kodowanie* jest nazywana „wskazaniem kodowania znaków”. Ciąg tekstowy znajdujący się po wskazanym kodowaniu znaków powinien być cytowany lub podany w postaci wartości szesnastkowej. Poniższe przykłady pokazują, jak zmusić ciąg tekstowy do interpretacji w kodowaniu znaków `latin2` lub `utf8`:

```
_latin2 'abc'
_latin2 X'616263'
_latin2 0x616263
_utf8 'def'
_utf8 X'646566'
_utf8 0x646566
```

W cytowanych ciągach tekstowych umieszczenie znaku odstępu pomiędzy kodowaniem znaków i samym ciągiem tekstowym jest opcjonalne. Natomiast w wartościach szesnastkowych znak odstępu jest wymagany.

Druga konwencja, notacja *N'ciąg tekstowy'*, jest odpowiednikiem `_utf8'ciąg tekstowy'`. Natychmiast po literze *N* (wielkość nie ma znaczenia) musi znajdować się cytowany dosłowny ciąg tekstowy, niedozwolone jest użycie znaku odstępu pomiędzy literą *N* i ciągiem tekstowym.

Pierwsza notacja sprawdza się doskonale w przypadku cytowanych dosłownych ciągów tekstowych i wartości szesnastkowych, ale nie dla wyrażeń ciągu tekstowego lub wartości kolumn. Jednak za pomocą funkcji `CONVERT()` dowolny ciąg tekstowy może być wygenerowany z zastosowaniem wskazanego kodowania znaków:

```
CONVERT(ciąg_textowy USING kodowanie_znaków);
```

Metoda polegająca na wskazaniu kodowania znaków i funkcja `CONVERT()` nie są tym samym. Metoda polegająca na wskazaniu kodowania znaków nie powoduje modyfikacji ciągu tekstowego w trakcie jego interpretacji i nie zmienia jego wartości (za wyjątkiem wielobajtowych kodowań znaków, w których trzeba zastosować dopełnienia, gdy ciąg tekstowy nie zawiera wystarczającej liczby bajtów). Z kolei funkcja `CONVERT()` pobiera ciąg tekstowy jako argument i generuje nowy ciąg tekstowy stosujący wskazane kodowanie znaków. Aby zobaczyć różnicę między metodą polegającą na wskazaniu kodowania znaków i funkcją `CONVERT()`, przeanalizujemy dwa poniższe zapytania, odwołujące się do dwubajтового kodowania znaków `ucs2`:

```
mysql> SET @s1 = _ucs2 'ABCD';
mysql> SET @s2 = CONVERT('ABCD' USING ucs2);
```

Przyjmujemy założenie, że bieżące kodowanie znaków to `latin1` (używające jednego bajta na znak). Pierwsze zapytanie interpretuje każdą parę znaków w ciągu tekstowym `'ABCD'` jako pojedynczy dwubajtowy znak `ucs2`, co oznacza wygenerowanie dwuznakowego ciągu tekstowego stosującego kodowanie `ucs2`. Z kolei drugie zapytanie konwertuje każdy znak ciągu tekstowego `'ABCD'` na odpowiadający mu znak `ucs2`, co oznacza wygenerowanie czteroznakowego ciągu tekstowego stosującego kodowanie `ucs2`.

Jaka jest „długość” ciągu tekstowego? To zależy od wielu czynników. Jeżeli długość będzie sprawdzana za pomocą funkcji `CHAR_LENGTH()`, wtedy zostanie wyrażona w znakach. W przypadku użycia funkcji `LENGTH()` długość będzie wyrażona w bajtach. W przypadku

ciągów tekstowych stosujących wielobajtowe kodowanie znaków oznacza to dwie różne wartości:

```
mysql> SELECT CHAR_LENGTH(@s1), LENGTH(@s1), CHAR_LENGTH(@s2), LENGTH(@s2);
+-----+-----+-----+-----+
| CHAR_LENGTH(@s1) | LENGTH(@s1) | CHAR_LENGTH(@s2) | LENGTH(@s2) |
+-----+-----+-----+-----+
|                2 |           4 |                4 |           8 |
+-----+-----+-----+-----+
```

Warto w tym miejscu pamiętać o jednym: binarny ciąg tekstowy to nie jest to samo, co niebinarny ciąg tekstowy stosujący binarną kolejność sortowania:

- Binarny ciąg tekstowy nie ma zdefiniowanego kodowania znaków. Jest interpretowany za pomocą semantyki bajtów, a w trakcie porównania używane są kody liczbowe o wielkości jednego bajta.
- Niebinarny ciąg tekstowy stosujący binarną kolejność sortowania ma semantykę znaków, a operacja porównania używa wartości liczbowych znaków, które mogą opierać się na wielu bajtach opisujących poszczególne znaki.

Poniżej pokazano różnicę między binarnym i niebinarnym ciągiem tekstowym. Utworzono binarny ciąg tekstowy i niebinarny ciąg tekstowy, któremu następnie przypisano binarną kolejność sortowania. Dalej oba wspomniane ciągi tekstowe przekazano funkcji UPPER():

```
mysql> SET @s1 = BINARY 'abcd';
mysql> SET @s2 = _latin1 'abcd' COLLATE latin1_bin;
mysql> SELECT UPPER(@s1), UPPER(@s2);
+-----+-----+
| UPPER(@s1) | UPPER(@s2) |
+-----+-----+
| abcd      | ABCD       |
+-----+-----+
```

Dlaczego funkcja UPPER() nie skonwertowała liter w binarnym ciągu tekstowym na wielkie? Odpowiedź jest prosta: binarny ciąg tekstowy nie ma zdefiniowanego kodowania znaków, a tym samym brak sposobu określenia, które wartości bajtów odpowiadają małym lub wielkim literom. Aby binarny ciąg tekstowy mógł być użyty wraz z funkcjami takimi jak UPPER() i LOWER(), w pierwszej kolejności trzeba go skonwertować na niebinarny ciąg tekstowy:

```
mysql> SELECT @s1, UPPER(CONVERT(@s1 USING latin1));
+-----+-----+
| @s1 | UPPER(CONVERT(@s1 USING latin1)) |
+-----+-----+
| abcd | ABCD                             |
+-----+-----+
```

3.1.2.2. Zmienne systemowe dotyczące kodowania znaków

Serwer obsługuje wiele zmiennych systemowych zaangażowanych w różne aspekty obsługi kodowania znaków. Z grupy wspomnianych zmiennych większość odwołuje się do

kodowania znaków, a pozostałe dotyczą kolejności sortowania. Poszczególne zmienne kolejności sortowania są powiązane z odpowiadającymi im zmiennymi kodowania znaków.

Pewne zmienne kodowania znaków wskazują właściwości serwera lub bieżącej bazy danych:

- Zmienna `character_set_system` wskazuje kodowanie znaków używane do przechowywania identyfikatorów. To zawsze jest kodowanie `utf8`.
- Zmienne `character_set_server` i `collation_server` wskazują domyślne w serwerze kodowanie znaków i kolejność sortowania.
- Zmienne `character_set_database` i `collation_database` wskazują kodowanie znaków i kolejność sortowania w domyślnej bazie danych. To są zmienne tylko do odczytu ustawiane automatycznie przez serwer po wyborze domyślnej bazy danych. Jeżeli nie ma domyślnej bazy danych, wartościami wymienionych zmiennych są domyślne w serwerze kodowanie znaków i kolejność sortowania. Zmienne `character_set_database` i `collation_database` są używane, gdy utworzysz tabelę, ale wyraźnie nie podasz kodowania znaków i kolejności sortowania. W takim przypadku tabela stosuje wartości domyślne określone na podstawie wartości domyślnych bazy danych.

Inne zmienne dotyczące kodowania znaków mają wpływ na sposób komunikacji prowadzonej między klientem i serwerem:

- Zmienna `character_set_client` wskazuje kodowanie znaków, w jakim klient wysłał zapytania SQL do serwera.
- Zmienna `character_set_results` wskazuje kodowanie znaków, w jakim serwer zwraca klientowi wynik zapytania. Pojęcie „wynik” oznacza wartości danych, a także metadane, takie jak nazwy kolumn.
- Zmienna `character_set_connection` jest używana przez serwer. Po otrzymaniu od klienta ciągu tekstowego zapytania serwer konwertuje ten ciąg tekstowy ze stosującego kodowanie `character_set_client` na stosujący kodowanie `character_set_connection`, a następnie pracuje z zapytaniem, używając drugiego z wymienionych kodowań znaków. (Mamy tutaj wyjątek: każdy dosłowny ciąg tekstowy w zapytaniu, jeśli zostanie poprzedzony nazwą kodowania znaków, będzie interpretowany we wskazanym kodowaniu znaków). Zmienna `collation_connection` jest używana do porównania dosłownych ciągów tekstowych z ciągami tekstowymi zapytań.
- Zmienna `character_set_filesystem` wskazuje kodowanie znaków stosowane przez system plików. MySQL używa wymienionej zmiennej do interpretacji dosłownych ciągów tekstowych odwołujących się w zapytaniach SQL do nazw plików, na przykład `LOAD DATA`. Przed utworzeniem pliku wspomniane ciągi tekstowe nazw plików są konwertowane ze stosującego kodowanie `character_set_client` na stosujące kodowanie `character_set_filesystem`. Wartością domyślną jest binarny (brak konwersji).

Prawdopodobnie przekonasz się, że większość zmiennych dotyczących kodowania znaków i kolejności sortowania domyślnie ma ustawione te same wartości. Na przykład, dane wyjściowe poniższych zapytań pokazują, że komunikacja między klientem i serwerem jest prowadzona z zastosowaniem kodowania znaków `latin1`:

```
mysql> SHOW VARIABLES LIKE 'character\_set\_%';
```

Variable_name	Value
character_set_client	latin1
character_set_connection	latin1
character_set_database	latin1
character_set_filesystem	binary
character_set_results	latin1
character_set_server	latin1
character_set_system	utf8

```
mysql> SHOW VARIABLES LIKE 'collation\_%';
```

Variable_name	Value
collation_connection	latin1_swedish_ci
collation_database	latin1_swedish_ci
collation_server	latin1_swedish_ci

Jeżeli komunikacja między klientem i serwerem ma być prowadzona z zastosowaniem innego kodowania znaków, można zmienić wartość odpowiednich zmiennych. Na przykład, w celu użycia kodowania `utf8` trzeba zmienić wartości trzech zmiennych:

```
mysql> SET character_set_client = utf8;
mysql> SET character_set_results = utf8;
mysql> SET character_set_connection = utf8;
```

Jednak znacznie wygodniejsze w takim przypadku jest użycie zapytania `SET NAMES`. Poniższe zapytanie jest odpowiednikiem trzech poprzednich zapytań `SET`:

```
mysql> SET NAMES 'utf8';
```

Jedynym ograniczeniem podczas definiowania kodowania znaków stosowanego podczas komunikacji jest brak możliwości użycia `ucs2`, `utf16`, `utf16le` i `utf32`.

Wiele programów klienckich obsługuje opcję `--default-character-set`, generującą taki sam efekt jak zapytanie `SET NAMES` i wskazującą serwerowi żądane kodowanie znaków, które powinno być używane podczas komunikacji.

Jeszcze łatwiejsza metoda jest dostępna dla standardowych programów klienckich MySQL, takich jak `mysql` i `mysqladmin`. Wystarczy ustawienia językowe zdefiniować za pomocą zmiennej środowiskowej `LANG` lub `LC_ALL`, a wymienione programy klienckie automatycznie wykryją i zastosują odpowiednie ustawienia. Na przykład, jeśli zmiennej `LC_ALL` zostanie przypisana wartość `en_US.UTF-8`, wówczas domyślne kodowanie znaków i kolejność sortowania dosłownych ciągów tekstowych będą określone następująco:

```
mysql> SELECT CHARSET('abcd'), COLLATION('abcd');
+-----+-----+
| CHARSET('abcd') | COLLATION('abcd') |
+-----+-----+
| utf8           | utf8_general_ci   |
+-----+-----+
```

W przypadku zmiennych działających w parach (zmienna kodowania znaków i zmienna kolejności sortowania) poszczególne komponenty pary są ze sobą połączone w następujący sposób:

- Ustawienie wartości zmiennej kodowania znaków jednocześnie powoduje ustawienie powiązanej z nią zmiennej kolejności sortowania, która otrzymuje wartość domyślną dla danego kodowania znaków.
- Ustawienie wartości zmiennej kolejności sortowania powoduje jednocześnie ustawienie powiązanej z nią zmiennej kodowania znaków, która otrzymuje wartość wskazaną przez pierwszą część nazwy kolejności sortowania.

Na przykład, ustawienie zmiennej `character_set_connection` wartości `utf8` powoduje jednoczesne przypisanie wartości `utf8_general_ci` zmiennej `collation_connection`. Natomiast ustawienie zmiennej `collation_connection` wartości `latin1_spanish_ci` powoduje jednoczesne przypisanie wartości `latin1` zmiennej `character_set_connection`.

3.1.3. Wartości daty i godziny

Wartości daty i godziny mają postać taką jak `'2012-06-17'` lub `'12:30:43'`. MySQL obsługuje również połączone wartości daty i godziny, na przykład `'2012-06-17 12:30:43'`. Trzeba pamiętać, że MySQL wyświetla datę w kolejności rok-miesiąc-dzień i dane wejściowe muszą być w podanym formacie. Wspomniana składnia często bywa zaskoczeniem dla początkujących, ale jest standardowym formatem SQL (nazywanym także formatem ISO 8601). Za pomocą funkcji `DATE_FORMAT()` wartości daty można wyświetlić w dowolny sposób, lecz domyślnie rok jest wyświetlany jako pierwszy. W przypadku wartości danych wejściowych w innych formatach istnieje możliwość ich konwersji za pomocą funkcji `STR_TO_DATE()`. Odpowiedni przykład znajdziesz w punkcie 3.2.6, zatytułowanym „Typy danych dla wartości daty i czasu”.

W przypadku połączonych wartości daty i godziny dozwolone jest użycie między datą i godziną znaku `T` zamiast spacji, na przykład `'2008-12-31T12:00:00'`.

Składnia połączonych wartości daty i godziny pozwala również na użycie części ułamkowej po wartości godziny, składającej się z przecinka dziesiętnego i do sześciu cyfr wskazujących mikrosekundy, na przykład `'12:30:15.000045'` lub `'2008-06-15 10:30:12.5'`.

3.1.4. Wartości przestrzenne

MySQL obsługuje wartości przestrzenne, ale tylko dla silników InnoDB, MyISAM, NDB i ARCHIVE. W ten sposób istnieje możliwość przedstawiania wartości takich jak punkty, linie i wielokąty. Wspomniane typy są implementowane przez specyfikację OpenGIS,

dostępna na witrynie internetowej konsorcjum Open Geospatial Consortium (<http://www.opengeospatial.org/>). Na przykład, poniższe zapytanie używa tekstu do przedstawienia wartości punktu o współrzędnych X i Y wynoszących (10,20). Wymienione dane są używane do utworzenia wartości typu POINT i przypisania zmiennej zdefiniowanej przez użytkownika otrzymanego wyniku:

```
SET @pt = POINTFROMTEXT('POINT(10 20)');
```

W tej książce wartości przestrzenne będą omówione w minimalnym zakresie. Więcej informacji na ich temat znajdziesz w podręczniku użytkownika MySQL.

3.1.5. Wartości boolowskie

W wyrażeniach zero jest uznawane za fałsz, natomiast dowolna wartość inna niż zero i NULL jest uznawana za prawdę.

Stałe specjalne o nazwach TRUE i FALSE mają wartości odpowiednio 1 i 0. Wielkość liter w nazwach stałych nie ma znaczenia.

3.1.6. Wartość NULL

Wartość NULL to wartość pozbawiona typu. Ogólnie rzecz biorąc, oznacza „brak wartości”, „nieznaną wartość”, „brakującą wartość”, „wartość spoza zakresu”, „wartość nieodpowiednią”, „wartość żadną z powyższych” itd. Istnieje możliwość wstawiania wartości NULL do tabel, pobierania ich z tabel i sprawdzania, czy wartością jest NULL. Jednak nie ma możliwości przeprowadzania operacji arytmetycznych na wartościach NULL; jeśli mimo wszystko spróbujesz, wynikiem będzie NULL. Ponadto, wiele funkcji zwraca NULL, jeśli zostaną wywołane z argumentem NULL lub nieprawidłowym.

Słowo kluczowe NULL jest pisane bez znaków cytowania, a wielkość liter nie ma znaczenia. MySQL traktuje \N (wielkość litery N ma znaczenie) jako NULL:

```
mysql> SELECT \N, ISNULL(\N);
+-----+-----+
| NULL | ISNULL(\N) |
+-----+-----+
| NULL |          1 |
+-----+-----+
```

3.2. Typy danych w MySQL

Każda tabela w bazie danych zawiera jedną lub więcej kolumn, których typ możesz zdefiniować w trakcie tworzenia tabeli za pomocą zapytania CREATE TABLE. Typ danych kolumny jest bardziej szczegółowy niż ogólna kategoria, taka jak „liczba” bądź „ciąg tekstowy”. W ten sposób dokładnie wskazujesz rodzaj wartości przechowywanych przez daną kolumnę, na przykład SMALLINT lub VARCHAR(32), co z kolei określa sposób, w jaki serwer MySQL będzie traktował poszczególne wartości. Na przykład, wartości liczbowe można przechowywać w kolumnie liczbowej lub ciągu tekstowego, ale w obu przypadkach MySQL odmiennie traktuje te wartości. Każdy typ danych ma kilka cech charakterystycznych:

- Rodzaj przechowywanych wartości.
- Ilość pamięci masowej wymaganej do przechowywania wartości.
- Określenie, czy wartości mają długość stałą (wszystkie wartości danego typu zabierają taką samą ilość pamięci masowej) czy zmienną (ilość zajmowanej pamięci masowej zależy od konkretnej wartości).
- Sposób, w jaki MySQL porównuje i sortuje wartości danego typu.
- Określenie, czy typ może być indeksowany i w jaki sposób.

Poniżej przedstawiono ogólne omówienie typów danych obsługiwanych przez MySQL. Następnie przejdziemy do dokładniejszej składni definicji poszczególnych typów i właściwości, jakie je charakteryzują, na przykład zakres i wymagania dotyczące pamięci masowej. Specyfikacje typu są przedstawiane w postaci, w jakiej ich używasz w zapytaniach `CREATE TABLE`. Nawiasy kwadratowe wskazują na informacje opcjonalne. Na przykład, składnia `MEDIUMINT [(M)]` wskazuje, że maksymalna wyświetlana szerokość wyrażona jako *(M)* jest opcjonalna. Z drugiej strony, w przypadku `VARCHAR(M)` brak nawiasu kwadratowego oznacza, że *(M)* jest wymagane.

3.2.1. Ogólny opis typów danych

MySQL posiada liczbowe typy danych przeznaczone do obsługi liczb całkowitych, liczb o stałej ilości cyfr, liczb zmiennoprzecinkowych i wartości bitowych, jak przedstawiono w tabeli 3.2. Za wyjątkiem `BIT` typy liczbowe mogą być ze znakiem lub bez znaku. Atrybut specjalny pozwala na automatyczne generowanie kolejnych wartości kolumny typu liczb całkowitych lub zmiennoprzecinkowych, co jest użyteczne w przypadku aplikacji wymagających do działania serii unikalnych liczb identyfikacyjnych.

Tabela 3.2. Liczbowe typy danych

Nazwa typu	Opis
<code>TINYINT</code>	Bardzo mała liczba całkowita.
<code>SMALLINT</code>	Mała liczba całkowita.
<code>MEDIUMINT</code>	Średniej wielkości liczba całkowita.
<code>INT</code>	Standardowa liczba całkowita.
<code>BIGINT</code>	Duża liczba całkowita.
<code>DECIMAL</code>	Liczba o stałej ilości cyfr i przecinku dziesiętnym.
<code>FLOAT</code>	Liczba zmiennoprzecinkowa o pojedynczej precyzji.
<code>DOUBLE</code>	Liczba zmiennoprzecinkowa o podwójnej precyzji.
<code>BIT</code>	Pole bitowe.

W tabeli 3.3 wymieniono typy danych używane przez MySQL do obsługi ciągów tekstowych. Wspomniane ciągi tekstowe mogą przechowywać cokolwiek, nawet dane binarne, takie jak używane do przedstawiania obrazów lub dźwięków. Ciągi tekstowe mogą rozróżniać wielkość liter. Ponadto, istnieje możliwość przeprowadzania operacji dopasowania wzorca w ciągu tekstowym. (Operacje dopasowania wzorca można przeprowadzać także na liczbowych typach danych, ale znacznie częściej używane są typy danych obsługujące ciągi tekstowe).

Tabela 3.3. Typy danych ciągu tekstowego

Nazwa typu	Opis
CHAR	Niebinarny ciąg tekstowy o stałej długości.
VARCHAR	Niebinarny ciąg tekstowy o zmiennej długości.
BINARY	Binarny ciąg tekstowy o stałej długości.
VARBINARY	Binarny ciąg tekstowy o zmiennej długości.
TINYBLOB	Bardzo mały obiekt typu BLOB (obiekt binarny).
BLOB	Mały obiekt typu BLOB.
MEDIUMBLOB	Średniej wielkości obiekt typu BLOB.
LONGBLOB	Duży obiekt typu BLOB.
TINYTEXT	Bardzo mały niebinarny ciąg tekstowy.
TEXT	Mały niebinarny ciąg tekstowy.
MEDIUMTEXT	Średniej wielkości niebinarny ciąg tekstowy.
LONGTEXT	Duży niebinarny ciąg tekstowy.
ENUM	Typ wyliczeniowy; wartością każdej kolumny jest jeden z elementów typu wyliczeniowego.
SET	Zbiór; każdej wartości kolumny przypisywane jest zero lub więcej elementów zbioru.

W tabeli 3.4 wymieniono obsługiwane przez MySQL typy danych przeznaczone do przechowywania wartości daty i godziny. W tym przypadku *RRRR*, *MM*, *DD*, *hh*, *mm* i *ss* przedstawiają odpowiednio rok, miesiąc, dzień miesiąca, godzinę, minutę i sekundy. Wspomniane wartości daty i godziny mogą być przedstawiane oddzielnie lub połączone ze sobą. Oprócz tego MySQL zapewnia obsługę znacznika czasu (to typ specjalny, pozwalający na śledzenie ostatnich zmian wprowadzonych w rekordzie). Istnieje również typ pozwalający na efektywne przechowywanie wartości roku, gdy nie ma potrzeby przechowywania całej daty.

Tabela 3.4. Wartości daty i godziny

Nazwa typu	Opis
DATE	Wartość daty w formacie 'RRRR-MM-DD'.
TIME	Wartość godziny w formacie 'gg:mm:ss'.
DATETIME	Wartość daty i godziny w formacie 'RRRR-MM-DD gg:mm:ss'.
TIMESTAMP	Wartość znacznika czasu w formacie 'RRRR-MM-DD gg:mm:ss'.
YEAR	Wartość roku w formacie RRRR lub RR.

3.2.2. Określenie typów kolumn w definicji tabeli

W celu utworzenia tabeli należy wykonać zapytanie CREATE TABLE zawierające listę tabel, które mają znajdować się w tabeli. Poniżej przedstawiono przykład zapytania tworzącego tabelę o nazwie mytbl wraz z kolumnami o nazwach f, c oraz i:

```
CREATE TABLE mytbl
(
  f FLOAT(10,4),
  c CHAR(15) NOT NULL DEFAULT 'none',
  i TINYINT UNSIGNED NULL
);
```

Każda kolumna ma nazwę, typ oraz ewentualnie atrybuty opcjonalne. Definicja kolumny ma następującą składnię:

nazwa_kolumny typ_kolumny [atrybuty_typu] [atrybuty_ogólne]

Na początku definicji zawsze znajduje się nazwa kolumny (parametr *nazwa_kolumny*), która musi być poprawnym identyfikatorem. Dokładne reguły dotyczące składni identyfikatora znajdziesz w podrozdziale 2.2, zatytułowanym „Identyfikatory składni MySQL i reguły nadawania nazw”. Krótko podsumowując, identyfikator kolumny może mieć do 64 znaków długości, może składać się z liter łacińskich, cyfr, znaków podkreślenia i dolara, a także rozszerzonych znaków Unicode. Słowa kluczowe, takie jak SELECT, DELETE i CREATE, są zarezerwowane i nie mogą być używane. Istnieje możliwość umieszczenia innych znaków w identyfikatorze lub użycia słowa zarezerwowanego, ale wówczas trzeba pamiętać o cytowaniu takiego identyfikatora w każdym odwołaniu do niego. Aby zacytować identyfikator, wystarczy umieścić go w odwrotnych apostrofach ('). Jeżeli włączony jest tryb SQL o nazwie ANSI_QUOTES, dozwolone jest cytowanie identyfikatorów poprzez ich ujęcie w cudzysłów.

Typ kolumny (parametr *typ_kolumny*) określa typ danych kolumny, czyli rodzaj wartości, jakie mogą być w niej przechowywane. W przypadku niektórych typów podawana jest maksymalna wielkość wartości, jakie kolumna może przechowywać. Z kolei w innych wielkość wartości wynika z nazwy typu. Na przykład, CHAR(10) wskazuje dokładnie długość wynoszącą 10 znaków, podczas gdy wartość typu TINYTEXT może mieć maksymalnie 255 bajtów. Niektóre typy pozwalają również na określenie maksymalnej wyświetlanej szerokości (to znaczy liczby znaków używanych do wyświetlenia wartości). W przypadku typu o stałej liczbie cyfr można podać liczbę cyfr znaczących oraz po przecinku dziesiętnym.

Po typie danych kolumny można podać opcjonalne atrybuty typu, a także znacznie bardziej ogólne atrybuty. Wspomniane atrybuty pełnią funkcję modyfikatorów typu i powodują zmianę sposobu, w jaki MySQL traktuje wartości w danej kolumnie:

- Dozwolone atrybuty charakterystyczne dla typu zależą od konkretnego typu danych. Na przykład, typy UNSIGNED i ZEROFILL są dozwolone jedynie dla typów liczbowych, natomiast CHARACTER SET i COLLATE jedynie dla typów niebinarnych ciągów tekstowych.
- Ogólne atrybuty mogą być stosowane w dowolnym typie danych, ale z kilkoma wyjątkami. Istnieje możliwość użycia atrybutu NULL i NOT NULL w celu określenia, czy dana kolumna może przechowywać wartości NULL. Dla większości typów danych można użyć klauzuli DEFAULT, wskazującej wartość domyślną kolumny. Omówienie sposobu obsługi wartości domyślnej znajdziesz w punkcie 3.2.3, zatytułowanym „Definiowanie wartości domyślnych kolumn”.

Jeżeli podanych zostanie wiele atrybutów kolumn, definiowane przez nie ograniczenia są nakładane w kolejności podania atrybutów. Ogólnie rzecz biorąc, nie powinno być problemów, jeśli typy charakterystyczne dla kolumn, takie jak UNSIGNED lub ZEROFILL, będziesz podawał przed ogólnymi, takimi jak NULL i NOT NULL.

3.2.3. Definiowanie wartości domyślnych kolumn

Dla wszystkich typów poza BLOB, TEXT, typami przestrzennymi i kolumnami z atrybutem AUTO_INCREMENT istnieje możliwość użycia klauzuli DEFAULT *wartość_domyślna*. Oznacza ona, że kolumna powinna przypisać wskazaną wartość domyślną, gdy rekord zostanie utworzony bez wyraźnie podanej wartości kolumny. Poza pewnymi wyjątkami dla kolumn TIMESTAMP i DATETIME, podana *wartość_domyślna* musi być stałą. Nie może być wyrażeniem lub odwoływać się do innych kolumn.

Jeżeli definicja kolumny nie zawiera wyraźnie zdefiniowanej klauzuli DEFAULT, a kolumna może pobierać wartości NULL, wtedy wartością domyślną jest NULL. W przeciwnym razie przyjmuje się, że kolumna została utworzona bez klauzuli DEFAULT, czyli nie ma zdefiniowanej wartości domyślnej. To wpływa na sposób, w jaki MySQL obsługuje wstawianie nowych rekordów do tabeli, w której nie zdefiniowano wartości dla kolumny:

- Kiedy nie jest używany tryb ścisłego SQL, wartością domyślną kolumny będzie ogólnie przyjęta wartość domyślna dla danego typu. (Ogólnie przyjęte wartości domyślne zostaną wkrótce przedstawione).
- Kiedy używany jest tryb ścisłego SQL, w przypadku tabel transakcyjnych zostanie wygenerowany błąd. Wykonywanie zapytania będzie przerwane i nastąpi wycofanie transakcji. Dla tabel nietransakcyjnych nastąpi zgłoszenie błędu i przerwanie wykonywania zapytania, jeśli rekord jest pierwszym wstawianym przez to zapytanie. Jeżeli to nie jest pierwszy rekord, możesz wybrać pomiędzy przerwaniem wykonywania zapytania a jego kontynuacją, wygenerowaniem ostrzeżenia i przypisaniem kolumnie ogólnie przyjętej wartości domyślnej dla kolumny

danego typu. Dostępny wybór zależy od wybranego typu ścisłego SQL. Więcej informacji szczegółowych na ten temat znajdziesz w podrozdziale 3.3, zatytułowanym „Jak MySQL obsługuje nieprawidłowe wartości danych?”. Z kolei informacje dotyczące transakcji przedstawiono w podrozdziale 2.12, zatytułowanym „Przeprowadzanie transakcji”.

Ogólnie przyjęte wartości domyślne dla kolumny zależą od jej typu danych:

- W przypadku kolumn liczbowych wartością domyślną jest 0, za wyjątkiem kolumn, dla których zdefiniowano atrybut `AUTO_INCREMENT`. Dla `AUTO_INCREMENT` wartością domyślną jest kolejna liczba w sekwencji kolumny.
- W przypadku większości kolumn przechowujących datę i godzinę wartością domyślną jest „zero” dla danego typu, na przykład '0000-00-00' dla typu `DATE`. W przypadku `TIMESTAMP` (i `DATETIME` w MySQL 5.6) mają zastosowanie specjalne reguły automatycznej inicjalizacji. Zapoznaj się z podpunktem 3.2.6.6, zatytułowanym „Automatyczne właściwości dla typów wartości daty i godziny”.
- W przypadku typów ciągu tekstowego innych niż `ENUM` wartością domyślną jest pusty ciąg tekstowy. Natomiast dla `ENUM` wartością domyślną jest pierwszy element danego typu wyliczeniowego. Z kolei dla `SET`, gdy kolumna nie może przyjmować wartości `NULL`, wartością domyślną jest zbiór pusty — odpowiednik pustego ciągu tekstowego.

Aby przekonać się, które kolumny w tabeli mają klauzulę `DEFAULT` i jaka została zdefiniowana dla nich wartość domyślna, wykonaj zapytanie `SHOW CREATE TABLE nazwa_tabeli`.

3.2.4. Liczbowe typy danych

W bazie danych MySQL liczbowe typy danych zaliczają się do trzech grup:

- Typy dokładnych wartości obejmują typy liczb całkowitych i `DECIMAL`. Typy liczb całkowitych są przeznaczone do przechowywania liczb pozbawionych części ułamkowej, na przykład 43, -3, 0 lub -798432. Kolumny liczb całkowitych możesz używać do przechowywania danych przedstawianych w postaci całych liczb, na przykład wagi zaokrąglonej do najbliższego kilograma, liczby osób w gospodarstwie domowym lub ilości danego produktu w magazynie. Typ `DECIMAL` przechowuje dokładne wartości, które mogą mieć część ułamkową, na przykład 3.14159, -.00273 lub -4.78. To jest dobry typ danych do przechowywania informacji takich jak wartości walutowe. Wartości w postaci liczb całkowitych i `DECIMAL` są przechowywane w dokładnie takiej postaci, w jakiej je podano, o ile to możliwe bez zaokrąglania, a przeprowadzane na nich obliczenia są dokładne.
- Typy zmiennoprzecinkowe są dostępne jako pojedynczej (`FLOAT`) i podwójnej (`DOUBLE`) precyzji. Wymienione typy, podobnie jak `DECIMAL`, są przeznaczone dla liczb, które mogą mieć część ułamkową, ale przechowują przybliżone wartości,

na przykład $3.9E+4$ lub $-0.1E-100$ poddawane zaokrągłaniu. Mogą być stosowane w sytuacjach, gdy ogromna precyzja nie jest wymagana, lub w przypadku tak dużych wartości, że nie mogą być obsługiwane przez typ DECIMAL. Przykładowe rodzaje informacji, które można przedstawić za pomocą wartości zmiennoprzecinkowych, to średnie zużycie paliwa lub odległości między gwiazdami.

- Typ BIT jest przeznaczony do przechowywania wartości pola bitowego.

Wartości zawierające część ułamkową również można przypisać kolumnie przechowującej liczby całkowite, ale zostaną one zaokrąglone, zgodnie z regułami matematycznymi. Dlatego też, jeżeli część ułamkowa wynosi .5 lub więcej, wartość zostanie zaokrąglona do kolejnej liczby całkowitej (w górę dla wartości dodatnich i w dół dla wartości ujemnych). Warto w tym miejscu dodać, że wartości w postaci liczb całkowitych mogą być przypisane typom pozwalającym na przechowywanie części ułamkowych. W takim przypadku część ułamkowa wynosi zero.

W tabeli 3.5 wymieniono nazwy i zakresy poszczególnych typów liczbowych, natomiast w tabeli 3.6 wymieniono ilości pamięci masowej wymaganej do przechowywania wartości poszczególnych typów. W przedstawionych poniżej tabelach M oznacza maksymalną szerokość wyświetlania dla typów w postaci liczb całkowitych, precyzję (liczbę znaczących cyfr) dla typów zmiennoprzecinkowych i DECIMAL, a także liczbę bitów dla typu BIT. Z kolei D oznacza skalę (liczbę cyfr po przecinku dziesiętnym) dla typów posiadających część ułamkową.

Tabela 3.5. Zakresy liczbowych typów danych

Nazwa typu	Zakres
TINYINT[(M)]	Wartości ze znakiem: od -128 do 127 (od -2^7 do 2^7-1); wartości bez znaku: od 0 do 255 (od 0 do 2^8-1)
SMALLINT[(M)]	Wartości ze znakiem: od -32768 do 32767 (od -2^{15} do $2^{15}-1$); wartości bez znaku: od 0 do 65535 (od 0 do $2^{16}-1$)
MEDIUMINT[(M)]	Wartości ze znakiem: od -8388608 do 8388607 (od -2^{23} do $2^{23}-1$); wartości bez znaku: od 0 do 16777215 (od 0 do $2^{24}-1$)
INT[(M)]	Wartości ze znakiem: od -2147483648 do 2147483647 (od -2^{31} do $2^{31}-1$); wartości bez znaku: od 0 do 4294967295 (od 0 do $2^{32}-1$)
BIGINT[(M)]	Wartości ze znakiem: od -9223372036854775808 do 9223372036854775807 (od -2^{63} do $2^{63}-1$); wartości bez znaku: od 0 do 18446744073709551615 (od 0 do $2^{64}-1$)
DECIMAL([M], [D]))	Zależy od M i D .
FLOAT([M], [D])	Minimalne wartości niezerowe: $\pm 1.175494351E-38$; maksymalne wartości niezerowe: $\pm 3.402823466E+38$
DOUBLE([M], [D])	Minimalne wartości niezerowe: $\pm 2.2250738585072014E-308$; maksymalne wartości niezerowe: $\pm 1.7976931348623157E+308$
BIT[(M)]	Od 0 do 2^M-1 , $1 \leq M \leq 64$

Tabela 3.6. Wymagania dotyczące pamięci masowej dla poszczególnych liczbowych typów danych

Nazwa typu	Wymagana ilość pamięci masowej
TINYINT[(M)]	1 bajt
SMALLINT[(M)]	2 bajty
MEDIUMINT[(M)]	3 bajty
INT[(M)]	4 bajty
BIGINT[(M)]	8 bajtów
DECIMAL[(M[,D])]	Zależy od M i D .
FLOAT[(M,D)]	4 bajty
DOUBLE[(M,D)]	8 bajtów
BIT[(M)]	Zależy od M .

Ilość wymaganej pamięci masowej dla wartości typu DECIMAL zależy od liczby cyfr po lewej i prawej stronie przecinka dziesiętnego. Po każdej stronie wymagane są cztery bajty dla każdych dziewięciu cyfr plus od jednego do czterech bajtów dla pozostałych cyfr. Całkowita ilość wymaganej pamięci jest sumą pamięci wymaganej do przechowywania cyfr po lewej i prawej stronie przecinka dziesiętnego.

Wartość typu BIT(M) wymaga około $(M+7)/8$ bajtów.

3.2.4.1. Liczbowe typy danych dokładnych wartości

Typy danych dokładnych wartości obejmują typy dla liczb całkowitych oraz typ DECIMAL.

W bazie danych MySQL typy dla liczb całkowitych to TINYINT, SMALLINT, MEDIUMINT, INT i BIGINT. Typ INTEGER jest synonimem dla INT. Wymienione typy różnią się zakresem wartości, jakie mogą przedstawiać, oraz ilością pamięci masowej wymaganej do ich przechowywania. (Im większy zakres, tym większa ilość wymaganej pamięci masowej). Kolumny liczb całkowitych mogą być zdefiniowane jako UNSIGNED w celu przechowywania wartości nieujemnych, co powoduje przesunięcie zakresu w górę, a początkiem zakresu staje się zero.

Definicja kolumny liczb całkowitych może zawierać opcjonalną wielkość wyświetlanej liczby (M), wyrażoną wartością od 1 do 255. Wspomniana wielkość określa liczbę znaków używanych do wyświetlenia wartości kolumny. Na przykład, MEDIUMINT(4) definiuje kolumnę typu MEDIUMINT wyświetlającą liczby czterocyfrowe. W przypadku gdy dla kolumny nie zostanie wyraźnie zdefiniowana wielkość wyświetlania, zostanie użyta wartość domyślna. Wspomniana wartość domyślna to wielkość „najdłuższej” wartości danego typu.

Wielkość wyświetlania (M) dla kolumny liczby całkowitej opiera się na liczbie znaków używanych do wyświetlania wartości kolumny. Nie ma *nic* wspólnego z liczbą bajtów pamięci masowej wymaganej do przechowywania wartości. Na przykład, wartość BIGINT wymaga ośmiu bajtów pamięci masowej niezależnie od wielkości wyświetlania. Nie ma możliwości magicznego zmniejszenia ilości pamięci masowej wymaganej do przechowywania

wartości BIGINT, jeśli zostanie zdefiniowana jako BIGINT(4). Opcja *M* nie ma także nic wspólnego z dozwolonym zakresem wartości. Jeżeli kolumna będzie zdefiniowana jako INT(3), nie oznacza to, że może przechowywać wartość maksymalnie 999. Ponadto, wyświetlane wartości nie są obcinane do *M* znaków. Jeżeli do wyświetlenia danej wartości będzie wymagane więcej niż *M* znaków, wtedy MySQL wyświetli pełną wartość.

Typ DECIMAL pozwala na przechowywanie liczby o zdefiniowanej ilości cyfr. Bardzo ważną cechą wartości DECIMAL jest fakt, że nie podlegają one błędowi wynikającemu z zaokrąglenia, co ma na przykład miejsce w przypadku wartości zmiennoprzecinkowych. Dzięki tej właściwości typ DECIMAL doskonale nadaje się do przechowywania wartości walutowych.

Typy NUMERIC i FIXED są synonimami typu DECIMAL.

Kolumny typu DECIMAL mogą być zdefiniowane jako UNSIGNED. W przeciwieństwie do typów liczb całkowitych, zdefiniowanie typu DECIMAL jako UNSIGNED nie powoduje przesunięcia zakresu w górę, a jedynie wyeliminowanie możliwości przechowywania wartości ujemnych.

Definicja kolumny typu DECIMAL może zawierać określenie maksymalnej liczby znaczących cyfr (*M*) oraz liczbę miejsc po przecinku dziesiętnym (*D*), czyli precyzję. Wartość *M* powinna być z zakresu od 1 do 65, natomiast wartość *D* powinna być z zakresu od 0 do 30, ale nie większa niż *M*.

Wartości *M* i *D* są opcjonalne. W przypadku pominięcia opcji *D* jej wartością domyślną jest zero. Z kolei po pominięciu opcji *M* jej wartością domyślną jest 10. Innymi słowy, poniższe zapytania są równoznaczne:

```
DECIMAL = DECIMAL(10) = DECIMAL(10,0)
DECIMAL(n) = DECIMAL(n,0)
```

Wartości *M* i *D* określają maksymalny możliwy zakres wartości DECIMAL. W przypadku zmiany *M* i pozostawienia stałej *D* zakres staje się większy wraz ze zwiększaniem wartości opcji *M* (patrz tabela 3.7). Z kolei w przypadku pozostawienia stałej wartości *M* i zmiany *D* zakres staje się mniejszy wraz ze wzrostem wartości *D* (patrz tabela 3.8).

Tabela 3.7. Wpływ opcji *M* na zakres wartości DECIMAL(*M*, *D*)

Specyfikacja typu	Zakres
DECIMAL(4,1)	od -999.9 do 999.9
DECIMAL(5,1)	od -9999.9 do 9999.9
DECIMAL(6,1)	od -99999.9 do 99999.9

Tabela 3.8. Wpływ opcji *D* na zakres wartości DECIMAL(*M*, *D*)

Specyfikacja typu	Zakres
DECIMAL(4,0)	od -9999 do 9999
DECIMAL(4,1)	od -999.9 do 999.9
DECIMAL(4,2)	od -99.99 do 99.99

3.2.4.2. Liczbowe typy danych wartości przybliżonych

MySQL oferuje dwa typy (FLOAT i DOUBLE) przeznaczone do obsługi liczb zmiennoprzecinkowych przechowujące wartości przybliżone. DOUBLE PRECISION jest synonimem DOUBLE. Z kolei typ REAL jest domyślnie synonimem DOUBLE lub FLOAT po włączeniu trybu SQL o nazwie REAL_AS_DEFAULT.

Typy zmiennoprzecinkowe mogą być zdefiniowane jako UNSIGNED, co powoduje uniemożliwienie przechowywania wartości ujemnych.

Podobnie jak w typie DECIMAL, definicja kolumny typu zmiennoprzecinkowego może zawierać określenie maksymalnej liczby znaczących cyfr (M) oraz liczbę miejsc po przecinku dziesiętnym (D), czyli precyzję. Wartość M powinna być z zakresu od 1 do 255, natomiast wartość D powinna być z zakresu od 0 do 30, ale nie większa niż M .

Wartości M i D są opcjonalne. W przypadku ich pominięcia w definicji kolumn, wartości są przechowywane do pełnej precyzji obsługiwanej przez używany sprzęt.

Dozwolona jest również składnia FLOAT(p). Jednak wprowadzie p oznacza wymaganą liczbę bitów precyzji w standardzie SQL, ale jest nieco inaczej traktowane w MySQL. Wartość opcji p może być z zakresu od 0 do 53 i jest używana jedynie do określenia, czy kolumna przechowuje wartości o pojedynczej, czy podwójnej precyzji. W przypadku opcji p z zakresu od 0 do 24 kolumna jest traktowana jako kolumna o pojedynczej precyzji. Z kolei wartości od 25 do 53 powodują traktowanie kolumny jako mającej podwójną precyzję. Oznacza to traktowanie kolumny jako FLOAT lub DOUBLE bez wartości M lub D .

3.2.4.3. Typ danych BIT

Typ danych BIT przechowuje wartości pola bitowego. Definicja kolumny typu BIT może zawierać opcjonalną maksymalną szerokość (M) wskazującą „wielkość” kolumny w bitach. Wartość wymienionej opcji powinna być z zakresu od 1 do 64; w przypadku jej pominięcia wartością domyślną jest 1.

Wartości pobrane z kolumn BIT nie są domyślnie w formie gotowej do wyświetlenia. Aby wyświetlić wartości pola bitowego, należy dodać zero lub użyć funkcji CAST():

```
mysql> CREATE TABLE t (b BIT(3)); # 3-bit column; holds values 0 to 7
mysql> INSERT INTO t (b) VALUES(0), (b'11'), (b'101'), (b'111');
mysql> SELECT b+0, CAST(b AS UNSIGNED) FROM t;
```

b+0	CAST(b AS UNSIGNED)
0	0
3	3
5	5
7	7

Funkcja BIN() jest użyteczna do wyświetlania w notacji binarnej wartości pola bitowego lub wyniku obliczeń przeprowadzanych na wspomnianych wartościach:

```
mysql> SELECT BIN(b), BIN(b & b'101'), BIN(b | b'101') FROM t;
```

BIN(b)	BIN(b & b'101')	BIN(b b'101')
0	0	101
3	101	111
5	101	111
7	101	111

0	0	101
11	1	111
101	101	101
111	101	111

W celu wyświetlenia wartości w notacji ósemkowej lub szesnastkowej należy użyć funkcji odpowiednio OCT() i HEX().

3.2.4.4. Atrybuty liczbowych typów danych

Atrybut UNSIGNED uniemożliwia przechowywanie wartości ujemnych. Można go stosować we wszystkich typach liczbowych poza BIT, choć najczęściej jest używany z typami liczb całkowitych. Zdefiniowanie kolumny liczb całkowitych jako UNSIGNED nie zmienia wielkości zakresu typu danych, a jedynie przesuwa zakres w górę. Rozważmy poniższą definicję tabeli:

```
CREATE TABLE mytbl
(
    itiny TINYINT,
    itiny_u TINYINT UNSIGNED
);
```

itiny i itiny_u to kolumny typu TINYINT z zakresem 256 wartości, choć różnią się zakresem dozwolonych wartości. Zakres itiny to wartości od -128 do 127, natomiast zakres itiny_u został przesunięty w górę i obejmuje wartości od 0 do 255.

Atrybut UNSIGNED jest użyteczny w przypadku kolumn, w których mają być przechowywane informacje niewymagające wartości ujemnych, na przykład wielkość populacji lub liczba uczestników. Kiedy do przechowywania wymienionych informacji używasz typu ze znakiem, to tak naprawdę do dyspozycji masz jedynie połowę zakresu tego typu danych. Dlatego też użycie atrybutu UNSIGNED w rzeczywistości podwaja wielkość zakresu. Na przykład, jeśli kolumna jest przeznaczona do przechowywania sekwencji liczb, użycie atrybutu UNSIGNED dwukrotnie zwiększa liczbę dostępnych wartości.

Atrybut UNSIGNED można również stosować w przypadku kolumn typu DECIMAL i wartości zmiennoprzecinkowych, choć otrzymany efekt nieco różni się od efektu zastosowania wymienionego atrybutu względem kolumn liczb całkowitych. Zakres nie zostaje przesunięty w górę, górna granica zakresu pozostaje bez zmian, natomiast dolną staje się zero (czyli efektywnie zakres zmniejsza się o połowę).

Atrybut SIGNED jest dozwolony dla wszystkich typów liczbowych dopuszczających użycie UNSIGNED. Ponieważ tego rodzaju typy są domyślnie definiowane jako SIGNED, użycie atrybutu SIGNED nie ma efektu innego niż wyraźne wskazanie, że kolumna dopuszcza możliwość przechowywania wartości ujemnych.

Atrybut ZEROFILL jest dozwolony dla wszystkich typów liczbowych poza BIT. Jego użycie powoduje, że wyświetlane wartości kolumny są dopełniane zerami do zdefiniowanej szerokości kolumny. Atrybut ZEROFILL możesz więc stosować w celu zagwarantowania, że wartości kolumny zawsze będą wyświetlane za pomocą wskazanej liczby cyfr. W rzeczywistości bardziej odpowiednim określeniem jest „za pomocą wskazanej *minimalnej* liczby cyfr”, ponieważ wartości większe niż zdefiniowana szerokość kolumny są wyświetlane

w pełni, bez ich obcinania. Możesz się o tym przekonać, wykonując przedstawione poniżej zapytania. Zauważ, że ostatnia wartość została wyświetlona w pełni, nawet pomimo tego, że jest szersza niż wskazana szerokość kolumny.

```
mysql> DROP TABLE IF EXISTS mytbl;
mysql> CREATE TABLE mytbl (my_zerofill INT(5) ZEROFILL);
mysql> INSERT INTO mytbl VALUES(1),(100),(10000),(1000000);
mysql> SELECT my_zerofill FROM mytbl;
```

my_zerofill
00001
00100
10000
1000000

W przypadku użycia atrybutu ZEROFILL dla kolumny, automatycznie staje się ona UNSIGNED.

Inny atrybut, AUTO_INCREMENT, jest dozwolony dla typów danych liczb całkowitych i zmiennoprzecinkowych, choć w tym drugim przypadku rzadko stosowany. Użycie atrybutu AUTO_INCREMENT powoduje wygenerowanie serii unikalnych wartości identyfikatora. W przypadku próby wstawienia wartości NULL do kolumny ze zdefiniowanym atrybutem AUTO_INCREMENT serwer MySQL wygeneruje kolejną wartość w sekwencji i umieści ją w kolumnie. O ile nie zdecydujesz inaczej, wartości AUTO_INCREMENT rozpoczynają się od 1 i są zwiększane o 1 dla każdego kolejnego rekordu. Usunięcie rekordu z tabeli ma wpływ na sekwencję. Od silnika bazy danych zależy, czy wartości w sekwencji mogą być ponownie użyte.

W tabeli może znajdować się co najwyżej jedna kolumna ze zdefiniowanym atrybutem AUTO_INCREMENT. Wspomniana kolumna powinna mieć ograniczenie NOT NULL i musi być zindeksowana. Ogólnie rzecz biorąc, kolumna AUTO_INCREMENT jest indeksowana jako PRIMARY KEY lub UNIQUE. Ponadto, ponieważ wartości sekwencji zawsze są dodatnie, kolumnę można zdefiniować jako UNSIGNED. Na przykład, kolumna AUTO_INCREMENT może być zdefiniowana na dowolny z wymienionych poniżej sposobów:

```
CREATE TABLE ai (i INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY);
CREATE TABLE ai (i INT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE);
CREATE TABLE ai (i INT UNSIGNED NOT NULL AUTO_INCREMENT, PRIMARY KEY (i));
CREATE TABLE ai (i INT UNSIGNED NOT NULL AUTO_INCREMENT, UNIQUE (i));
```

Pierwsze dwie formy pokazują określenie informacji o indeksie jako części definicji kolumny. W dwóch kolejnych określenie indeksu następuje za pomocą oddzielnej klauzuli zapytania CREATE TABLE. Użycie oddzielnej klauzuli jest opcjonalne, jeśli indeks zawiera tylko kolumnę AUTO_INCREMENT. W celu utworzenia indeksu obejmującego wiele kolumn, w tym AUTO_INCREMENT, konieczne jest użycie oddzielnej klauzuli. (Przykład znajdziesz w podpunkcie 3.4.2.1, zatytułowanym „Atrybut AUTO_INCREMENT w tabelach MyISAM”).

Kolumnę AUTO_INCREMENT zawsze można wyraźnie zdefiniować jako NOT NULL, w przypadku pominięcia NOT NULL serwer MySQL automatycznie dołączy ten atrybut.

W podrozdziale 3.4, zatytułowanym „Praca z sekwencjami”, również poruszono temat zachowania kolumn wraz ze zdefiniowanym atrybutem `AUTO_INCREMENT`.

Po podaniu omówionych dotąd atrybutów, które są charakterystyczne dla kolumn liczbowych, można podać atrybut `NULL` lub `NOT NULL`. W przypadku braku wyraźnie zdefiniowanego atrybutu `NULL` lub `NOT NULL` dla kolumn przechowujących liczby, domyślnie dozwolone jest wstawianie wartości `NULL` (jak wcześniej wspomniano, kolumna `AUTO_INCREMENT` ma domyślnie zdefiniowany atrybut `NOT NULL`).

Wartość domyślną można również wskazać za pomocą atrybutu `DEFAULT`. Poniższa definicja tabeli powoduje utworzenie trzech kolumn typu `INT` i zdefiniowanie dla nich wartości domyślnych `-1`, `1` i `NULL`:

```
CREATE TABLE t
(
  i1 INT DEFAULT -1,
  i2 INT DEFAULT 1,
  i3 INT DEFAULT NULL
);
```

W punkcie 3.2.3, zatytułowanym „Definiowanie wartości domyślnych kolumn”, omówiono reguły stosowane przez MySQL podczas przypisywania wartości domyślnej, gdy definicja kolumny nie zawiera klauzuli `DEFAULT`.

3.2.4.5. Wybór liczbowego typu danych

Kiedy wybierasz typ dla kolumny liczbowej, weź pod uwagę przechowywane w niej wartości, a następnie zdecyduj się na najmniejszy typ pokrywający zakres. Wybór większego typu oznacza marnotrawstwo miejsca, niepotrzebne zwiększenie tabel, które nie będą przetwarzane tak efektywnie jak w przypadku wyboru mniejszego typu danych. `TINYINT` to najlepszy typ danych dla liczb całkowitych, o ile zakres danych jest mały, na przykład wiek lub liczba rodzeństwa. Typ `MEDIUMINT` może przedstawiać miliony wartości i być wykorzystany do przechowywania wielu innych typów informacji, oczywiście kosztem większego zapotrzebowania na pamięć masową. Typ `BIGINT` ma największy zakres, ale wymaga dwukrotnie większej ilości pamięci w porównaniu do typu `INT` i powinien być używany jedynie, gdy naprawdę zachodzi potrzeba. W przypadku wartości zmiennoprzecinkowych typ `DOUBLE` zabiera dwukrotnie więcej pamięci masowej niż `FLOAT`. O ile nie potrzebujesz wyjątkowo dużej precyzji lub ogromnego zakresu wartości, dane możesz przechowywać, używając zamiast `DOUBLE` typu `FLOAT`, który wymaga o połowę mniejszej ilości pamięci masowej.

Każdy typ kolumny liczbowej określa jej zakres wartości. Jeżeli próbujesz wstawić wartość wykraczającą poza zakres kolumny, wynik zależy od tego, czy włączony jest tryb ścisły SQL. Jeżeli wymieniony tryb jest włączony, nastąpi wygenerowanie błędu wykroczenia wartości poza zakres. W przypadku wyłączonego trybu ścisłego SQL nastąpi obcięcie wartości: MySQL obetnie wartość do odpowiedniego punktu końcowego zakresu, do tabeli wstawi obciętą wartość i wygeneruje komunikat ostrzeżenia.

Obcinanie wartości następuje odpowiednio do typu danych zakresu, a nie zdefiniowanej szerokości wyświetlania wartości kolumny. Na przykład, kolumna `SMALLINT(3)` ma zdefiniowane wyświetlanie trzycyfrowej liczby z zakresu od `-32768` do `32767`. Wartość `12345` jest szersza niż zdefiniowana szerokość kolumny, ale mieści się w zakresie. Będzie więc wstawiona bez obcięcia i pobierana w postaci `12345`. Z kolei wartość `99999` wykracza poza zakres. Dlatego też podczas wstawiania będzie obcięta do `32767`, a operacje pobierania danych będą zwracały wartość `32767`.

W przypadku kolumn przechowujących liczby zmiennoprzecinkowe lub o zdefiniowanej liczbie cyfr, jeśli wstawiana wartość ma więcej cyfr niż dozwolona dla części ułamkowej (zdefiniowana w specyfikacji kolumny), wtedy nastąpi zaokrąglenie. Dlatego też, jeśli spróbujesz wstawić wartość `1.23456` do kolumny zdefiniowanej jako `FLOAT(8,1)`, wstawiona będzie liczba `1.2`. Z kolei wstawienie tej samej wartości do kolumny `FLOAT(8,4)` spowoduje wstawienie liczby `1.2346`. Oznacza to, że omawiane kolumny powinny być definiować wraz z odpowiednią liczbą miejsc po przecinku dziesiętnym, aby zachować wymaganą precyzję. Jeśli wymagasz precyzji sięgającej tysięcznych części, nie definiuj typu kolumny z jedynie dwoma miejscami po przecinku dziesiętnym.

3.2.5. Typy danych w postaci ciągów tekstowych

MySQL oferuje wiele typów danych przeznaczonych do przechowywania ciągów tekstowych. Wspomniane ciągi tekstowe są często używane w przypadku wartości tekstowych takich jak poniższe:

```
'J. Kowalski, doktor'
'Ołówki (2 szt.)'
'u1. Dobra 123'
'Monograph Series IX'
```

W rzeczywistości ciągi tekstowe to „ogólne” typy pod tym względem, że można je wykorzystać do przechowywania dowolnych wartości. Na przykład, typów binarnego ciągu tekstowego można używać do przechowywania danych binarnych takich jak obrazy, dźwięki lub skompresowane dane wyjściowe (gzip).

W tabeli 3.9 wymieniono wszystkie dostarczane przez MySQL typy przeznaczone do definiowania kolumn przechowujących ciągi tekstowe. W tabeli znajdziesz informacje o ich maksymalnej wielkości i wymaganej ilości pamięci masowej. W tabeli *M* oznacza maksymalną wielkość wartości kolumny (wyrażoną w bajtach dla binarnych ciągów tekstowych oraz w znakach dla niebinarnych ciągów tekstowych), natomiast *L* oznacza rzeczywistą wielkość danej wartości wyrażoną w bajtach. Z kolei *w* oznacza liczbę bajtów wymaganych dla najszerzego znaku w kodowaniu znaków zdefiniowanym dla kolumny. Typy `BLOB` i `TEXT` mają po kilka wariantów różniących się maksymalną wielkością przechowywanych wartości.

Tabela 3.9. Typy danych w postaci ciągów tekstowych

Nazwa typu	Wielkość maksymalna	Wymagana ilość pamięci masowej
BINARY(<i>M</i>)	<i>M</i> bajtów	<i>M</i> bajtów
VARBINARY(<i>M</i>)	<i>M</i> bajtów	<i>L</i> + 1 lub 2 bajtów
CHAR(<i>M</i>)	<i>M</i> znaków	<i>M</i> -w bajtów
VARCHAR(<i>M</i>)	<i>M</i> znaków	<i>L</i> + 1 lub 2 bajtów
TINYBLOB, TINYTEXT	2 ⁸ –1 bajtów	<i>L</i> + 1 bajtów
BLOB, TEXT	2 ¹⁶ –1 bajtów	<i>L</i> + 2 bajtów
MEDIUMBLOB, MEDIUMTEXT	2 ²⁴ –1 bajtów	<i>L</i> + 3 bajtów
LONGBLOB, LONGTEXT	2 ³² –1 bajtów	<i>L</i> + 4 bajtów
ENUM('wartość1', 'wartość2', ...)	65535 elementów	1 lub 2 bajty
SET('wartość1', 'wartość2', ...)	64 elementy	1, 2, 3, 4 lub 8 bajtów

Pewne typy przechowują binarne ciągi tekstowe (bajty), natomiast inne niebinarne (znaki). Na przykład, kolumna BINARY(20) przechowuje 20 bajtów, podczas gdy CHAR(20) przechowuje 20 znaków (co w przypadku wielobajtowego kodowania znaków wymaga więcej niż 20 bajtów). W punkcie 3.1.2, zatytułowanym „Wartości ciągu tekstowego”, znajdziesz omówienie różnic między semantyką bajta i znaku w binarnych i niebinarnych ciągach tekstowych. Każdy typ binarnego ciągu tekstowego posiada odpowiadający mu typ niebinarny, jak przedstawiono w tabeli 3.10.

Tabela 3.10. Odpowiadające binarne i niebinarne typy danych w postaci ciągów tekstowych

Typ binarnego ciągu tekstowego	Typ niebinarnego ciągu tekstowego
BINARY	CHAR
VARBINARY	VARCHAR
BLOB	TEXT

Każdemu typowi niebinarnego ciągu tekstowego jak również typom ENUM i SET można przypisać kodowanie znaków oraz kolejność sortowania. Poszczególne kolumny mogą mieć zdefiniowane różne kodowania znaków. Temat przypisywania kodowania znaków zostanie omówiony w podpunkcie 3.2.5.5, zatytułowanym „Atrybuty typu danych ciągu tekstowego”.

Typy BINARY i CHAR są przeznaczone dla ciągów tekstowych o stałej wielkości. Dla kolumn wymienionych typów MySQL alokuje tę samą ilość pamięci masowej dla każdej wartości i krótsze dopełnia do pełnej długości kolumny. Dopełnienie opiera się na bajtach 0x00 dla typu BINARY oraz spacjach dla typu CHAR. Ponieważ typ CHAR(*M*) musi być w stanie przedstawić największy możliwy ciąg tekstowy w kodowaniu znaków kolumny, każda kolumna wymaga *M*-w bajtów, gdzie *w* oznacza liczbę bajtów wymaganych przez najszerszy

znak w używanym kodowaniu znaków. Na przykład, w kodowaniu `ujis` znaki wymagają od 1 do 3 bajtów, a tym samym `CHAR(20)` wymaga alokacji 60 bajtów, gdy wartość wymaga po trzy bajty dla każdego z dwudziestu znaków.

Inne typy są przeznaczone do przechowywania ciągów tekstowych o zmiennej wielkości. Ilość pamięci masowej wymaganej dla wartości w danym rekordzie zależy od maksymalnej dozwolonej wielkości wartości przechowywanej w kolumnie. Wspomniana wielkość jest przedstawiona przez literę *L* w tabeli wymieniającej typy ciągów tekstowych o zmiennej wielkości. Dodatkowe bajty wymagane poza *L* to liczba bajtów niezbędna do przechowywania informacji o wielkości wartości. MySQL obsługuje wartości o zmiennej wielkości przez przechowywanie treści wartości i prefiksu wskazującego jej wielkość. Wspomniane dodatkowe bajty wiążące się z „prefiksem wielkości” są traktowane jako liczba całkowita bez znaku. Istnieje związek między maksymalną wielkością typu o zmiennej wielkości, liczbą bajtów potrzebną do przechowywania informacji o wielkości i zakresem typu liczby całkowitej bez znaku używającej tej samej liczby bajtów. Na przykład, wartość `MEDIUMBLOB` może mieć wielkość do $2^{24}-1$ bajtów i wymaga trzech bajtów do zapisania informacji o wielkości. Trzybajtowa liczba całkowita typu `MEDIUMINT` ma maksymalną wartość bez znaku wynoszącą $2^{24}-1$. To nie jest zbieg okoliczności.

Prefiks przechowujący informacje o wielkości wartości typu `VARBINARY` i `VARCHAR` wymaga jednego bajta, o ile wyrażona w bajtach maksymalna wielkość wartości kolumny wynosi 256. W przeciwnym razie wymagane są dwa bajty.

W przypadku wszystkich typów ciągu tekstowego poza `ENUM` i `SET`, baza danych MySQL przechowuje wartości w postaci sekwencji bajtów i interpretuje je jako bajty lub znaki w zależności od tego, czy typ przechowuje binarne, czy niebinarne ciągi tekstowe. Zbyt długie wartości do wstawienia są obcinane. (W trybie ścisłym SQL wystąpi błąd, jeśli obcięte będą znaki inne niż spacje). Ciągi tekstowe są bardzo zróżnicowane, od niezwykle małych do ogromnych, a największy typ pozwala na przechowywanie niemal 4 GB danych. Dlatego też bez problemów powinien znaleźć odpowiedni typ pozwalający na uniknięcie obciążenia wstawianego ciągu tekstowego. (Efektywna wielkość kolumny tak naprawdę jest powiązana z maksymalną wielkością pakietu w protokole komunikacyjnym klient-serwer. Wielkość domyślna wspomnianego pakietu to 1 MB).

Wprawdzie dla typów `ENUM` i `SET` definicja kolumny zawiera listę poprawnych wartości ciągu tekstowego, ale wewnętrznie wartości `ENUM` i `SET` są przechowywane jako liczby, co będzie dokładnie omówione w podpunkcie 3.2.5.4, zatytułowanym „Typy danych `ENUM` i `SET`”. Próba przechowywania wartości nieistniejącej na liście spowoduje konwersję takiej wartości na `''` (pusty ciąg tekstowy), o ile nie jest włączony tryb ścisły SQL. W trybie ścisłym nastąpi wygenerowanie błędu.

3.2.5.1. Typy danych `CHAR` i `VARCHAR`

Typy `CHAR` i `VARCHAR` są przeznaczone do przechowywania niebinarnych ciągów tekstowych, a tym samym mają przypisane kodowanie znaków i kolejność sortowania.

Podstawowa różnica między typami `CHAR` i `VARCHAR` kryje się w ich wielkości przechowywanych wartości oraz sposobie obsługi spacji na końcu ciągu tekstowego:

- Typ CHAR ma stałą wielkość przechowywanych wartości, natomiast VARCHAR to typ przechowujący wartości o zmiennej wielkości.
- W wartościach pobieranych z kolumn typu CHAR spacje znajdujące się na końcu są usuwane. Dla kolumny CHAR(M) w trakcie operacji wstawiania wartości krótsze niż M znaków są dopełniane spacjami do długości M , natomiast podczas ich pobierania spacje na końcu wartości są usuwane. Włączenie trybu SQL o nazwie PAD_CHAR_TO_FULL_LENGTH powoduje pobieranie z kolumn typu CHAR wartości wraz z zachowaniem spacji na końcu.
- Dla typu VARCHAR spacje znajdujące się na końcu wartości są zachowywane zarówno podczas wstawiania, jak i pobierania wartości.

Kolumny typu CHAR mogą być zdefiniowane wraz z maksymalną wielkością M z zakresu od 0 do 255. Dla kolumny typu CHAR podanie M jest opcjonalne, w przypadku braku wartość domyślna M wynosi 1. Zwróć uwagę na możliwość użycia definicji CHAR(0). Tego rodzaju kolumna może być wykorzystana do przedstawiania wartości typu włączony/wyłączony, o ile zezwolisz na wstawianie NULL. Wartości w tego rodzaju kolumnie to tylko NULL lub pusty ciąg tekstowy. Kolumna zdefiniowana jako CHAR(0) pobiera bardzo małą ilość pamięci masowej w tabeli — tylko jeden bit.

Dozwolony zakres M dla VARCHAR(M) wynosi od 1 do 65535, ale efektywna maksymalna liczba znaków jest mniejsza niż 65535, ponieważ w MySQL maksymalna wielkość rekordu wynosi 65535 bajtów. To powoduje wymienione poniżej implikacje:

- Długa wartość VARCHAR wymaga dwóch bajtów, co ma wpływ na wielkość rekordu.
- Użycie znaków opisywanych więcej niż tylko jednym bajtem zmniejsza liczbę znaków, które można umieścić w danym rekordzie.
- Zdefiniowanie innych kolumn w tabeli zmniejsza ilość przestrzeni dla kolumny typu VARCHAR w rekordzie.

Podczas podejmowania decyzji o wyborze spośród typów danych CHAR i VARCHAR pamiętaj o dwóch ogólnych zasadach:

- Jeżeli wszystkie wartości składają się z M znaków, kolumna typu VARCHAR(M) zabierze więcej pamięci masowej niż kolumna CHAR(M) ze względu na dodatkowe bajty wymagane do zapisania informacji o wielkości wartości. Z drugiej strony, jeśli wartości mają różną wielkość, to kolumna VARCHAR zabierze mniejszą ilość pamięci masowej. Kolumna CHAR(M) zawsze pobiera M znaków, nawet jeśli wartość to pusty ciąg tekstowy lub NULL.
- Jeżeli używasz tabel MyISAM i wartości nie różnią się zbytnio wielkością, wtedy typ CHAR jest lepszym wyborem niż VARCHAR, ponieważ silnik bazy danych MyISAM może znacznie efektywniej przetwarzać rekordy o wielkości stałej niż zmiennej. Zapoznaj się z podrozdziałem 5.4, zatytułowanym „Wybór formatu tabeli dla efektywnych zapytań”.

3.2.5.2. Typy danych BINARY i VARBINARY

Typy BINARY i VARBINARY są podobne do CHAR i VARCHAR, ale z następującymi różnicami:

- Typy CHAR i VARCHAR są niebinarne, przechowują znaki, a także mają zdefiniowane kodowanie znaków i kolejność sortowania. Operacja porównania jest oparta na kolejności sortowania.
- Typy BINARY i VARBINARY są binarne, przechowują bajty, nie mają zdefiniowanego kodowania znaków i kolejności sortowania. Operacja porównania jest oparta na liczbowych wartościach bajtów.

Dla kolumny BINARY (M) w trakcie wstawiania informacji wartości krótsze niż M bajtów są dopełniane bajtami 0x00 aż do pełnej wielkości M . Podczas pobierania danych nie następuje ich obcinanie. Dla typu VARBINARY nie jest przeprowadzane dopełnianie w trakcie wstawiania danych, a podczas pobierania danych nie są one obcinane.

3.2.5.3. Typy danych BLOB i TEXT

BLOB to ogromny obiekt binarny, w zasadzie to kontener, który może przechowywać praktycznie każde umieszczone w nim dane o dowolnej wielkości do 4 GB. W MySQL typ BLOB tak naprawdę składa się z rodziny typów (TINYBLOB, BLOB, MEDIUMBLOB i LONGBLOB). Wymienione typy są identyczne, poza maksymalną ilością informacji, jakie mogą przechowywać (patrz tabela 3.9). Kolumna BLOB przechowuje binarne ciągi tekstowe; są one użyteczne do przechowywania danych, które mogą znacznie się rozrosnąć lub w dużym stopniu różnią się wielkością. Przykładami tego rodzaju danych są dane skompresowane, zaszyfrowane, obrazy, dźwięki itd.

Baza danych MySQL oferuje także rodzinę typów TEXT (TINYTEXT, TEXT, MEDIUMTEXT i LONGTEXT). Są one podobne do odpowiadających im typów BLOB za wyjątkiem faktu, że typy TEXT przechowują niebinarne ciągi tekstowe zamiast binarnych. Oznacza to, że przechowują znaki zamiast bajtów i mają przypisane kodowanie znaków oraz kolejność sortowania. Ogólne różnice między binarnymi i niebinarnymi ciągami tekstowymi przedstawiono we wcześniejszym punkcie 3.1.2, zatytułowanym „Wartości ciągu tekstowego”. Na przykład, podczas porównywania wartości BLOB używane są bajty, natomiast wartości TEXT są porównywane na podstawie znaków i kolejności sortowania przypisanej kolumnie.

Jednak maksymalna wielkość typu TEXT jest taka sama jak w przypadku BLOB. Oznacza to, że maksymalna wielkość dla obu typów jest wyrażana w bajtach zamiast w bajtach dla typu BLOB i znakach dla typu TEXT (patrz tabela 3.9).

Kolumny typu BLOB lub TEXT czasami mogą być indeksowane, ale to zależy od używanego silnika bazy danych:

- Silniki InnoDB i MyISAM obsługują indeksowanie kolumn typu BLOB i TEXT. Jednak konieczne jest podanie wielkości prefiksu, który będzie użyty do indeksowania. W ten sposób unika się tworzenia indeksów tak dużych, że niwelowałyby zalety wynikające z używania indeksu. Wyjątkiem jest to, aby prefiksy nie były używane dla indeksów typu FULLTEXT w kolumnach TEXT. Wyszukiwanie pełnego tekstu

opiera się na całej zawartości indeksowanych kolumn, a podawane prefiksy są ignorowane.

- Tabele MEMORY nie obsługują indeksów w kolumnach typu BLOB i TEXT, ponieważ silnik bazy danych MEMORY w ogóle nie obsługuje kolumn typu BLOB i TEXT.

Kolumny typu BLOB lub TEXT mogą wymagać specjalnego traktowania:

- Ze względu na ogromne wahania w wielkości wartości BLOB i TEXT, zawierające je tabele są podatne na dużą fragmentację w przypadku przeprowadzania wielu operacji usunięcia lub uaktualniania danych. Jeżeli do przechowywania wartości BLOB lub TEXT używasz tabeli MyISAM, okresowe wykonywanie zapytania OPTIMIZE TABLE zmniejsza fragmentację i zapewnia dobrą wydajność. Więcej informacji na ten temat znajdziesz w rozdziale 5., zatytułowanym „Optymalizacja zapytań”.
- Zmienna systemowa `max_sort_length` wpływa na operacje porównania i sortowania wartości BLOB oraz TEXT. Używanych jest jedynie pierwszych `max_sort_length` bajtów. (W przypadku kolumn TEXT stosujących wielobajtowe kodowanie znaków oznacza to, że operacja porównania może używać mniejszej liczby znaków niż wskazana przez zmienną `max_sort_length`). Jeżeli to powoduje problem przy wartości domyślnej `max_sort_length` wynoszącej 1024, zwiększ wymienioną wartość, zanim rozpoczniiesz operację porównania.
- W przypadku ogromnych wartości może wystąpić potrzeba konfiguracji serwera polegającej na zwiększeniu wartości parametru `max_allowed_packet`. Więcej informacji znajdziesz w punkcie 12.7.1, zatytułowanym „Zmienne systemowe ogólnego przeznaczenia do dostrajania serwera”. Konieczne jest również zwiększenie wielkości pakietu po stronie klienta, który ma używać ogromnych wartości. Programy klienckie `mysql` i `mysqldump` obsługują bezpośrednie ustawienie wspomnianej wartości za pomocą opcji startowej.

3.2.5.4. Typy danych ENUM i SET

ENUM i SET to specjalne typy danych ciągów tekstowych przeznaczone do przechowywania jedynie wartości wybieranych z przygotowanej listy dozwolonych ciągów tekstowych. Podstawowa różnica między wymienionymi typami polega na tym, że wartości kolumny ENUM muszą składać się dokładnie z jednego elementu pobranego z listy wartości, podczas gdy kolumna SET może zawierać dowolną lub wszystkie wartości z listy. Innymi słowy, typ ENUM jest używany w przypadku wartości wzajemnie się wykluczających, natomiast SET pozwala na wybranie kilku opcji z listy.

Typ danych ENUM definiuje typ wyczerpujący, który może składać się z maksymalnie 65535 elementów. Kolumnie typu ENUM można przypisać wartość składającą się z dokładnie jednego elementu wybranego z listy wartości zdefiniowanej w trakcie tworzenia tabeli. Typy wyczerpujące są powszechnie używane do przedstawiania kategorii wartości. Na przykład, wartości w kolumnie zdefiniowanej jako `ENUM('N', 'T')` może być tylko N lub T. Typu ENUM można używać do przechowywania informacji takich jak dostępne wielkości i kolory produktu, odpowiedzi na pytania w ankiecie, gdzie można wybrać kilka opcji, a także odpowiedzi na pytania w kwestionariuszu, w którym można wybrać tylko jedną opcję:

```
employees ENUM('mniej niż 100','100-500','501-1500','więcej niż 1500')
color ENUM('czerwony','zielony','niebieski','czarny')
size ENUM('S','M','L','XL','XXL')
vote ENUM('Tak','Nie','Nie wiem')
```

Jeżeli przetwarzane są odpowiedzi udzielone na stronie internetowej zawierającej wzajemnie wykluczające się opcje, typ ENUM można wykorzystać do przedstawienia opcji, spośród których odwiedzający witrynę wybiera odpowiedź. Na przykład, jeśli przyjmujesz przez internet zamówienia na pizzę, kolumny ENUM mogą być używane do przedstawienia rodzaju ciasta i wielkości pizzy:

```
crust ENUM('cienkie','zwykłe','średnie','grube')
size ENUM('mała','średnia','duża')
```

Typ SET jest podobny do ENUM pod tym względem, że podczas tworzenia kolumny SET definiujesz listę dozwolonych wartości. Wspomniana lista może składać się z maksymalnie 64 elementów. W przeciwieństwie do typu ENUM wartością każdej kolumny może być dowolna liczba elementów pobranych z listy. Jednym z przykładów użycia typu SET jest zestaw wartości, które nie są wzajemnie wykluczające się, jak ma to miejsce w przypadku typu ENUM. Na przykład, podczas przyjmowania przez internet zamówień na pizzę można wyświetlić klientowi zbiór pól wyboru opisujących dodatkowe składniki do pizzy. W ten sposób klient będzie mógł wskazać ewentualne dodatki, które chce zobaczyć na zamówionej pizzy. Do przedstawienia wspomnianych składników wykorzystujemy typ SET:

```
SET('pepperoni','kiełbasa','grzyby','cebula','oliwki')
```

Następnie konkretna wartość SET będzie składała się z elementów wybranych przez klienta:

```
'pepperoni,grzyby'
'kiełbasa,cebula'
'kiełbasa,grzyby,oliwki'
'cebula'
''
```

Ostatnia z powyższych wartości (pusty ciąg tekstowy) oznacza, że klient nie zdecydował się na dodatkowe składniki do pizzy. To jest dozwolona wartość dla każdej kolumny SET.

Definicja kolumny SET jest zapisana w postaci listy poszczególnych ciągów tekstowych rozdzielonych przecinkami w celu wskazania kolejnych elementów. Z kolei *wartość* kolumny SET jest zapisana w postaci pojedynczego ciągu tekstowego. Jeżeli wartość jest złożona z wielu elementów, wtedy poszczególne elementy w ciągu tekstowym są rozdzielone przecinkami. Dlatego też jako elementu listy SET nie powinien używać ciągu tekstowego zawierającego przecinki.

Sposób definiowania listy dozwolonych wartości dla kolumny ENUM i SET jest ważny pod wieloma względami:

- Lista wskazuje dozwolone wartości dla kolumny, co zostało już wcześniej omówione.
- Jeżeli dla kolumny ENUM lub SET zdefiniowano kolejność sortowania nierozróżniając wielkości znaków, wtedy dozwolone wartości można wstawiać

jako zapisane przy użyciu liter o dowolnej wielkości. Jednak wielkość znaków w ciągach tekstowych podanych w definicji kolumny wskazuje wielkość znaków w pobieranych później wartościach. Na przykład, jeśli zdefiniujesz kolumnę `ENUM('N', 'T')` i przechowujesz w niej wartości `t` i `n`, wówczas po ich pobraniu będą wyświetlone jako `T` i `N`. Jeżeli kolumna stosuje kolejność sortowania binarną lub rozróżniającą wielkość liter, wartości trzeba wstawiać, stosując wielkość liter dokładnie taką, jaka została użyta w definicji kolumny. W przeciwnym razie wartości zostaną uznane za nieprawidłowe. Z drugiej strony, to pozwala na stosowanie elementów różniących się jedynie wielkością liter w nazwie, co jest niemożliwe w przypadku kolejności sortowania nierozróżniającej wielkości liter.

- Kolejność wartości w definicji `ENUM` jest kolejnością używaną podczas sortowania. Kolejność wartości w definicji `SET` również wpływa na kolejność sortowania, choć sama relacja jest znacznie bardziej skomplikowana, ponieważ kolumny mogą zawierać wiele elementów z listy.
- Kiedy MySQL wyświetla wartość `SET` składającą się z wielu elementów listy, kolejność elementów w wartości jest określana na podstawie kolejności, w jakiej poszczególne elementy znajdują się w definicji kolumny `SET`.

Typy `ENUM` i `SET` są klasyfikowane jako typy ciągu tekstowego, ponieważ elementy typu wyliczeniowego i zbioru są definiowane w postaci ciągów tekstowych w trakcie tworzenia kolumn wymienionych typów. Jednak typy `ENUM` i `SET` tak naprawdę są podzielone: ich elementy składowe są wewnętrznie przechowywane w postaci liczb, a więc możesz z nimi pracować tak jak z liczbami. Oznacza to, że typy `ENUM` i `SET` są znacznie efektywniejsze od pozostałych typów ciągu tekstowego, ponieważ bardzo często mogą być obsługiwane za pomocą operacji liczbowych, a nie na ciągach tekstowych. Ponadto, zyskujesz możliwość użycia wartości `ENUM` i `SET` w kontekście ciągu tekstowego lub liczbowym. Wreszcie, kolumny `ENUM` i `SET` mogą powodować zamieszanie, jeśli używasz ich w kontekście ciągu tekstowego, ale oczekujesz, że będą zachowywały się jak liczby i na odwrót.

W definicji kolumny kolejnym elementom typu `ENUM` serwer MySQL nadaje liczby, począwszy od 1. (Wartość 0 jest zarezerwowana dla elementu błędu, który jest przedstawiany przez pusty ciąg tekstowy). Wspomniana liczba wpływa na ilość pamięci masowej wymaganej przez kolumnę `ENUM`. Jeden bajt przedstawia 256 wartości, natomiast dwa bajty mogą przedstawić 65535 wartości. (Porównaj to z zakresami jedno- i dwubajtowych typów liczb całkowitych `TINYINT`, `UNSIGNED` i `SMALLINT UNSIGNED`). Dlatego też, wliczając element błędu, maksymalna liczba elementów typu wyliczeniowego wynosi 65536, a ilość wymaganej pamięci masowej zależy od tego, czy typ zawiera ponad 256 elementów. W definicji typu `ENUM` możesz podać 65535 (a nie 65536) elementów, ponieważ MySQL rezerwuje miejsce dla elementu błędu. Kiedy dla kolumny `ENUM` zdefiniujesz nieprawidłową wartość, MySQL przypisze jej wartość elementu błędu. (W trybie ścisłym nastąpi wygenerowanie błędu).

Poniższy przykład pokazuje, że wartości typu `ENUM` można pobierać w postaci ciągu tekstowego lub liczb. (Ten przykład pokazuje także liczbową kolejność elementów typu wyliczeniowego oraz to, że wartość `NULL` nie ma przypisanego numeru).

```
mysql> CREATE TABLE e_table (e ENUM('jane','fred','will','marcia'));
mysql> INSERT INTO e_table
-> VALUES('jane'),('fred'),('will'),('marcia'),(NULL);
mysql> SELECT e, e+0, e+1, e*3 FROM e_table;
```

e	e+0	e+1	e*3
jane	1	2	3
fred	2	3	6
will	3	4	9
marcia	4	5	12
NULL	NULL	NULL	NULL

Elementy typu ENUM można porównywać przez nazwę lub przypisaną mu liczbę:

```
mysql> SELECT e FROM e_table WHERE e='will';
```

e
will

```
mysql> SELECT e FROM e_table WHERE e=3;
```

e
will

Istnieje możliwość zdefiniowania pustego ciągu tekstowego jako prawidłowego elementu typu wyliczeniowego, ale takie rozwiązanie spowoduje tylko zamieszanie. Wspomnianemu pustemu ciągowi tekstowemu zostanie przypisana wartość niezerowa, podobnie jak pozostałym elementom. Jednak pusty ciąg tekstowy jest używany dla elementu błędu, który również ma przypisaną wartość zero, a tym samym będzie odpowiadał dwóm wewnętrznym wartościom liczbowym elementów. W poniższym przykładzie przypisanie kolumnie ENUM nieprawidłowej wartości x spowoduje przypisanie elementu błędu. Odróżnić ten element od elementu pustego ciągu tekstowego wymienionego w definicji kolumny można jedynie podczas pobierania wartości liczbowej:

```
mysql> CREATE TABLE t (e ENUM('a','','b'));
mysql> INSERT INTO t VALUES('a'),(''),('b'),('x');
mysql> SELECT e, e+0 FROM t;
```

e	e+0
a	1
	2
b	3
x	0

W trybie ścisłym przypisanie nieprawidłowej wartości x spowoduje wygenerowanie błędu, a do tabeli nie zostanie wstawiona wartość.

Liczbowa reprezentacja kolumn SET jest nieco inna niż w przypadku kolumn ENUM. Elementy listy SET nie są kolejno numerowane. Zamiast tego elementy odpowiadają kolejnym poszczególnym bitom w wartości SET. Pierwszy element odpowiada bitowi 0, drugi element bitowi 1 itd. Innymi słowy, liczbowe wartości elementów SET to zawsze potęgi liczby 2. Pusty ciąg tekstowy odpowiada liczbowej wartości SET wynoszącej 0.

Wartości SET są przechowywane jako wartości bitowe. W ten sposób w jednym bajcie można przechowywać osiem elementów, a więc na ilość pamięci masowej wymaganej przez kolumnę SET wpływa liczba elementów, których maksymalna liczba wynosi 64. Wartości SET pobierają 1, 2, 3, 4 lub 8 bajtów dla wielkości zbioru odpowiednio od 1 do 8, od 9 do 16, od 17 do 24, od 25 do 32 i od 33 do 64.

Reprezentacja wartości SET jako zbioru bitów pozwala, aby wspomniana wartość składała się z wielu elementów. Każda kombinacja bitów może być zmieniona na wartość, a tym samym wartość może składać się z dowolnego połączenia ciągów tekstowych w definicji SET odpowiadającej tym bitom.

Poniższy przykład pokazuje związek między formami tekstową i liczbową kolumny SET. Wartość liczbową jest wyświetlana zarówno dziesiętnie, jak i binarnie:

```
mysql> CREATE TABLE s_table (s SET('table','lamp','chair','stool'));
mysql> INSERT INTO s_table
-> VALUES('table'),('lamp'),('chair'),('stool'),(''),(NULL);
mysql> SELECT s, s+0, BIN(s+0) FROM s_table;
```

s	s+0	BIN(s+0)
table	1	1
lamp	2	10
chair	4	100
stool	8	1000
	0	0
NULL	NULL	NULL

Jeżeli kolumnie s przypisana zostanie wartość 'lamp,stool', wówczas MySQL wewnętrznie przechowuje ją jako 10 (binarnie 1010), ponieważ 'lamp' ma wartość 2 (bit 1), natomiast 'stool' ma wartość 8 (bit 3).

Kiedy przypisujesz wartości kolumnom SET, podciągi tekstowe nie muszą być wymienione w tej samej kolejności, jaką zastosowano do zdefiniowania kolumny. Jednak podczas późniejszego pobierania wartości elementy są wyświetlane w kolejności ich zdefiniowania. Ponadto, jeśli kolumnie SET przypiszesz wartość zawierającą podciągi tekstowe niewymienione jako elementy zbioru, wspomniane ciągi tekstowe zostaną wyrzucone, a kolumnie będzie przypisana wartość składająca się z pozostałych podciągów tekstowych. W trakcie późniejszego pobierania wartości nieprawidłowe podciągi tekstowe nie będą wyświetlane.

Jeżeli kolumnie s w tabeli s_table przypiszesz wartość 'chair,couch,table', wówczas:

- Podciąg tekstowy 'couch' zostanie odrzucony, ponieważ nie istnieje jako element zbioru SET. Serwer MySQL po prostu ustala bity odpowiadające poszczególnym podciągom tekstowym wartości do przypisania, a następnie

zmienia je na przechowywaną wartość. Ponieważ podciągowi 'couch' nie odpowiada żaden bit, jest on odrzucany.

- Podczas późniejszego pobierania wartości ma ona postać 'table,chair'. W trakcie operacji pobierania MySQL tworzy ciąg tekstowy wartości na podstawie wartości liczbowych przez skanowanie bitów w kolejności. To powoduje automatyczne ułożenie podciągów tekstowych w kolejności użytej podczas definiowania kolumny. Przedstawione zachowanie oznacza również, że jeśli element zbioru zostanie w wartości użyty więcej niż tylko jeden raz, to w pobieranej wartości pojawi się tylko jednokrotnie. Dlatego też, jeśli dla kolumny SET zdefiniujesz wartość 'lamp,lamp,lamp', po pobraniu otrzymasz jedynie 'lamp'.

W trybie ścisłym użycie nieprawidłowego elementu SET spowoduje wygenerowanie błędu, a sama wartość nie zostanie wstawiona do bazy danych. W poprzednim przykładzie próba przypisania wartości zawierającej 'couch' spowoduje błąd i niepowodzenie operacji przypisania.

Fakt zmiany przez MySQL kolejności elementów w wartości SET oznacza, że jeśli wyszukujesz wartość za pomocą ciągu tekstowego, elementy musisz wymienić w prawidłowej kolejności. Dlatego też, jeżeli wstawisz 'chair,table', a następnie będziesz wyszukiwał 'table,chair' to nie znajdziesz rekordu; musisz szukać ciągu tekstowego 'chair,table'.

Sortowanie i indeksowanie kolumn ENUM i SET jest przeprowadzane według rosnącej kolejności (wewnętrznych) wartości przypisanych wartościom kolumny. Przedstawiony poniżej przykład może wydawać się nieprawidłowy, ponieważ wartości nie są wyświetlane w kolejności alfabetycznej:

```
mysql> SELECT e FROM e_table ORDER BY e;
```

```
+-----+
| e      |
+-----+
| NULL   |
| jane   |
| fred   |
| will   |
| marcia |
+-----+
```

Aby lepiej przekonać się, jak działa sortowanie w omawianych kolumnach, pobierzemy wartości ENUM w postaci zarówno ciągu tekstowego, jak i liczbowej:

```
mysql> SELECT e, e+0 FROM e_table ORDER BY e;
```

```
+-----+-----+
| e      | e+0 |
+-----+-----+
| NULL   | NULL |
| jane   | 1    |
| fred   | 2    |
| will   | 3    |
| marcia | 4    |
+-----+-----+
```

Jeżeli masz stały zestaw wartości, które chcesz sortować w określonej kolejności, wtedy możesz wykorzystać pewne właściwości sortowania oferowane przez typ ENUM. W tabeli wartości przedstaw jako kolumnę ENUM, a następnie wartości typu wyliczeniowego wymień w definicji kolumny w takiej kolejności, w jakiej chcesz je mieć posortowane. Przyjmujemy założenie, że masz tabelę przedstawiającą personel organizacji sportowej, na przykład drużyny piłkarskiej. Twoim celem jest posortowanie danych wyjściowych według pozycji personelu, aby były wyświetlone we wskazanej kolejności: trenerzy, asystenci trenerów, bramkarze, obrońcy, rozgrywający, napastnicy itd. Zdefiniuj więc kolumnę jako typu ENUM, a następnie elementy typu wyliczeniowego podaj w kolejności, którą chcesz otrzymać w danych wyjściowych. Następnie, podczas operacji sortowania wartości kolumny będą automatycznie wyświetlane w zdefiniowanej kolejności.

Jeżeli chcesz wartości ENUM posortować w normalnej kolejności leksykalnej, wtedy za pomocą funkcji CAST() kolumnę możesz skonwertować na postać ciągu tekstowego innego niż ENUM, a następnie posortować wynik:

```
mysql> SELECT CAST(e AS CHAR) AS e_str FROM e_table ORDER BY e_str;
+-----+
| e_str |
+-----+
| NULL  |
| fred  |
| jane  |
| marcia|
| will  |
+-----+
```

Funkcja CAST() nie zmienia wyświetlanych wartości, ale w przedstawionym zapytaniu przeprowadza konwersję wartości ENUM na ciąg tekstowy, co powoduje zmianę właściwości sortowania i pozwala na sortowanie wartości tak samo jak ciągów tekstowych.

3.2.5.5. Atrybuty typu danych ciągu tekstowego

Atrybuty unikalne dla typów danych ciągu tekstowego to CHARACTER SET (lub CHARSET) i COLLATE, odpowiedzialne za określenie kodowania znaków i kolejności sortowania. Można je zdefiniować jako opcje samej tabeli (stają się wówczas wartościami domyślnymi tabeli) lub dla poszczególnych kolumn (nadpisują wtedy wartości domyślne tabeli). W rzeczywistości, każda baza danych ma domyślnie zdefiniowane kodowanie znaków i kolejność sortowania, podobnie jak sam serwer. Wspomniane wartości domyślne mają czasami znaczenie podczas tworzenia tabeli, o czym przekonasz się w dalszej części książki.

Atrybuty CHARACTER SET i COLLATE mają zastosowanie w następujących typach danych: CHAR, VARCHAR, TEXT, ENUM i SET. Nie mają natomiast zastosowania w typach danych dla binarnych ciągów tekstowych (BINARY, VARBINARY i BLOB), ponieważ one zawierają ciągi tekstowe bajtów, a nie znaków.

Podczas określania atrybutów CHARACTER SET i COLLATE na poziomie kolumny, tabeli lub bazy danych stosowane są następujące reguły:

- Kodowanie znaków musi być jednym z obsługiwanych przez serwer. W celu wyświetlenia dostępnych kodowań znaków należy wykonać zapytanie `SHOW CHARACTER SET`.
- W przypadku definicji zawierającej atrybut zarówno `CHARACTER SET`, jak i `COLLATE` wskazane kodowanie znaków i kolejność sortowania muszą być zgodne ze sobą. Na przykład, wraz z kodowaniem `latin2` można użyć kolejności sortowania `latin2_croatian_ci`, ale już nie `latin1_bin`. W celu wyświetlenia kolejności sortowania dostępnych dla danego kodowania znaków należy wykonać zapytanie `SHOW COLLATION`.
- W przypadku definicji zawierającej atrybut `CHARACTER SET`, ale bez `COLLATE` użyta będzie domyślna kolejność sortowania dla wskazanego kodowania znaków.
- W przypadku definicji zawierającej atrybut `COLLATE`, ale bez `CHARACTER SET` kodowanie znaków zostanie określone na podstawie pierwszego członu nazwy kolejności sortowania.

Aby przekonać się, jak są stosowane powyższe reguły, rozważ poniższe zapytanie. Powoduje ono utworzenie tabeli używającej kilku kodowań znaków:

```
CREATE TABLE mytbl
(
  c1 CHAR(10),
  c2 CHAR(40) CHARACTER SET latin2,
  c3 CHAR(10) COLLATE latin1_german1_ci,
  c4 BINARY(40)
) CHARACTER SET utf8;
```

W powstałej tabeli domyślne kodowanie znaków to `utf8`. Ponieważ nie użyto opcji `COLLATE` dla tabeli, domyślną kolejnością sortowania w tabeli jest domyślna kolejność sortowania w kodowaniu znaków `utf8`, czyli `utf8_general_ci`. Definicja kolumny `c1` nie zawiera atrybutów `CHARACTER SET` i `COLLATE`, więc użyte zostaną wartości domyślne tabeli. Z kolei ustalone na poziomie tabeli kodowanie znaków i kolejność sortowania nie będą używane dla kolumn `c2`, `c3` i `c4`. Kolumny `c2` i `c3` mają zdefiniowane informacje o kodowaniu znaków, natomiast kolumna `c4` jest typu binarnego ciągu tekstowego, więc atrybuty kodowania znaków i kolejności sortowania nie mają tutaj zastosowania. Dla kolumny `c2` kolejnością sortowania jest `latin2_general_ci`, czyli domyślna kolejność sortowania w kodowaniu `latin2`. Z kolei dla kolumny `c3` kodowanie znaków to `latin1`, na co wskazuje nazwa kolejności sortowania `latin1_german1_ci`.

Aby wyświetlić informacje o kodowaniu znaków dla istniejącej tabeli, należy wykonać zapytanie `SHOW CREATE TABLE`:

```
mysql> SHOW CREATE TABLE mytbl\G
***** 1. row *****
Table: mytbl
Create Table: CREATE TABLE `mytbl` (
  `c1` char(10) DEFAULT NULL,
  `c2` char(40) CHARACTER SET latin2 DEFAULT NULL,
  `c3` char(10) CHARACTER SET latin1 COLLATE latin1_german1_ci DEFAULT NULL,
```

```
`c4` binary(40) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

Jeżeli zapytanie `SHOW CREATE TABLE` nie wyświetla informacji o kodowaniu znaków w kolumnie, oznacza to zastosowanie domyślnego kodowania znaków tabeli. W przypadku braku informacji o kolejności sortowania zastosowana będzie domyślna kolejność sortowania w danym kodowaniu znaków.

Do zapytania `SHOW COLUMNS` można dodać słowo kluczowe `FULL`, które spowoduje wyświetlenie informacji o kolejności sortowania. Na tej podstawie można określić kodowanie znaków:

```
mysql> SHOW FULL COLUMNS FROM mytbl;
```

Field	Type	Collation	Null	Key	Default	...
c1	char(10)	utf8_general_ci	YES		NULL	...
c2	char(40)	latin2_general_ci	YES		NULL	...
c3	char(10)	latin1_german1_ci	YES		NULL	...
c4	binary(40)	NULL	YES		NULL	...

W powyższej analizie poruszono temat zdefiniowania kodowania znaków dla kolumny i tabeli, ale kodowanie znaków można określić na poziomie kolumny, tabeli, bazy danych lub serwera. Kiedy MySQL przetwarza definicję kolumny, kodowanie znaków określa przez zastosowanie poniższych reguł w podanej kolejności:

1. Jeżeli definicja kolumny zawiera kodowanie znaków, to zostanie ono użyte. Obejmuje to także sytuację, gdy podany jest jedynie atrybut `COLLATE`, ponieważ na jego podstawie można określić używane kodowanie znaków.
2. Jeżeli definicja kolumny zawiera opcję użycia kodowania znaków tabeli, wtedy zostanie ono użyte.
3. W przeciwnym razie użyte zostanie kodowanie znaków bazy danych jako domyślne kodowanie znaków tabeli, które jednocześnie staje się obowiązujące dla kolumny. Jeżeli w bazie danych nigdy nie zostało wyraźnie ustawione kodowanie znaków, wówczas zostanie ono pobrane z serwera.

Innymi słowy, MySQL przechodzi na coraz wyższy poziom, szukając ustawionego kodowania znaków. Po jego znalezieniu będzie użyte dla danej kolumny. Ponieważ baza danych zawsze ma domyślne kodowanie znaków, masz gwarancję zakończenia procesu wyszukiwania na poziomie bazy danych, nawet jeśli na wcześniejszych poziomach nie zostało wyraźnie zdefiniowane żadne kodowanie znaków.

Kodowanie znaków o nazwie `binary` jest specjalne. Jego przypisanie do kolumny niebinarnego ciągu tekstowego odpowiada zdefiniowaniu kolumny za pomocą odpowiadającego mu typu binarnego ciągu tekstowego. Poniższe pary definicji kolumn pokazują odpowiadające sobie definicje:

```
c1 CHAR(10) CHARACTER SET binary
c1 BINARY(10)
```

```
c2 VARCHAR(10) CHARACTER SET binary
c2 VARBINARY(10)
```

```
c3 TEXT CHARACTER SET binary
c3 BLOB
```

Jeżeli dla kolumny binarnego ciągu tekstowego zdefiniujesz binarne kodowanie znaków za pomocą zapytania `CHARACTER SET binary`, wtedy zostanie ono zignorowane, ponieważ typ jest już binarny. W przypadku zdefiniowania binarnego kodowania znaków dla typu `ENUM` lub `SET` zostanie ono użyte w podanej postaci.

Jeżeli kodowanie znaków `binary` będzie przypisane tabeli, wówczas ma zastosowanie w każdej kolumnie ciągu tekstowego, dla której w definicji wyraźnie nie podano informacji o kodowaniu znaków.

MySQL oferuje pewne skróty atrybutów pozwalające na definiowanie kodowania znaków w kolumnach:

- Atrybut `ASCII` jest skrótem dla `CHARACTER SET latin1`.
- Atrybut `UNICODE` jest skrótem dla `CHARACTER SET ucs2`.
- Zdefiniowanie atrybutu `BINARY` dla kolumny niebinarnego ciągu tekstowego, `ENUM` lub `SET` jest skrótem wskazującym na binarną kolejność sortowania w kodowaniu znaków kolumny. Na przykład, przy założeniu, że domyślne kodowanie znaków w tabeli to `latin1`, poniższe definicje mają taki sam efekt:

```
c1 CHAR(10) BINARY
c2 CHAR(10) CHARACTER SET latin1 BINARY
c3 CHAR(10) CHARACTER SET latin1 COLLATE latin1_bin
```

Jeżeli użyjesz atrybutu `BINARY` dla kolumny binarnego ciągu tekstowego, wtedy atrybut zostanie zignorowany, ponieważ typ jest już binarny.

Ogólne atrybuty, takie jak `NULL` i `NOT NULL`, mogą być używane dla dowolnego typu ciągu tekstowego. Jeśli nie podasz żadnego z nich, domyślnie stosowany jest `NULL`. Jednak zdefiniowanie kolumny ciągu tekstowego jako `NOT NULL` nie chroni przed umieszczeniem w kolumnie pustego ciągu tekstowego (to znaczy `' '`). W MySQL wartość pusta jest traktowana inaczej od brakującej, a więc nie popełnij błędu, sądząc, że możesz wymusić przechowywanie w kolumnie niepustych wartości przez jej zdefiniowanie jako `NOT NULL`. Jeżeli istnieje wymóg, aby wartości ciągu tekstowego były niepuste, tego rodzaju ograniczenie trzeba zaimplementować na poziomie aplikacji.

Poza `BLOB` i `TEXT` dla pozostałych typów przechowujących ciągi tekstowe wartość domyślną można zdefiniować za pomocą klauzuli `DEFAULT`. W punkcie 3.2.3, zatytułowanym „Definiowanie wartości domyślnych kolumn”, opisano reguły stosowane przez MySQL podczas przypisywania wartości domyślnej, gdy w definicji kolumny nie znajduje się klauzula `DEFAULT`.

3.2.5.6. Wybór typu danych ciągu tekstowego

Kiedy wybierasz typ danych dla kolumny ciągu tekstowego, odpowiedz sobie na następujące pytania:

Czy przechowywane wartości będą przedstawiane w postaci znaków, czy danych binarnych? W przypadku znaków najodpowiedniejszym rozwiązaniem jest użycie typu niebinarnego ciągu tekstowego. Z kolei dla danych binarnych wybieraj typ binarnego ciągu tekstowego.

Czy podczas operacji porównania wielkość liter powinna mieć znaczenie? Jeżeli tak, to wybierz jeden z typów niebinarnego ciągu tekstowego, ponieważ przechowuje on znaki, a także definiuje kodowanie znaków i kolejność sortowania.

Wielkość znaków dla wartości niebinarnych ciągów tekstowych w celu przeprowadzania operacji porównania i sortowania jest kontrolowana przez przypisaną im kolejność sortowania. Aby ciągi tekstowe traktować równo niezależnie od wielkości znaków, należy użyć nierozróżniającej wielkości znaków kolejności sortowania. W przeciwnym razie użyj kolejności sortowania binarnej lub uwzględniającej wielkość znaków. Kolejność sortowania uwzględniająca wielkość znaków porównuje je, używając określonej kolejności sortowania, która nie musi odpowiadać kolejności kodów znaków. W takich przypadkach na potrzeby sortowania mała i wielka litera są uznawane za inne. Przyjmujemy założenie, że `mysql`, `MySQL` i `MYSQL` to ciągi tekstowe stosujące kodowanie znaków `latin1`. W przypadku kolejności sortowania nieuwzględniającej wielkości znaków, takiej jak `latin1_swedish_ci`, wszystkie wymienione ciągi tekstowe są uznawane za identyczne. Natomiast po zastosowaniu kolejności sortowania binarnej (`latin1_bin`) lub uwzględniającej wielkość znaków (`latin1_general_cs`) są uznawane za różne.

Aby użyć kolumny ciągu tekstowego do porównania zarówno uwzględniającego wielkość znaków, jak i jej nieuwzględniającego, należy zastosować kolejność sortowania odpowiadającą najczęściej przeprowadzanemu rodzajowi konwersji. Do wykonania konwersji drugiego rodzaju trzeba użyć operacji `COLLATE` w celu zmiany kolejności sortowania. Na przykład, jeśli `mycol` to kolumna typu `CHAR` stosująca kodowanie znaków `latin1`, można jej przypisać kolejność sortowania `latin1_swedish_ci` w celu domyślnego przeprowadzania operacji porównania bez uwzględniania wielkości liter. Poniższe porównanie nie uwzględnia wielkości liter:

```
mycol = 'ABC'
```

Gdy zajdzie potrzeba przeprowadzenia porównania uwzględniającego wielkość liter, trzeba użyć kolejności sortowania `latin1_general_cs` lub `latin1_bin`. Poniższe operacje porównania rozróżniają wielkość liter (nie ma znaczenia, do którego ciągu tekstowego — po lewej lub po prawej stronie — zostanie zastosowany operator `COLLATE`):

```
mycol COLLATE latin1_general_cs = 'ABC'
mycol COLLATE latin1_bin = 'ABC'
mycol = 'ABC' COLLATE latin1_general_cs
mycol = 'ABC' COLLATE latin1_bin
```

Czy chcesz zminimalizować ilość wymaganej pamięci masowej? Jeżeli tak, użyj typu przechowującego ciągi tekstowe o zmiennej wielkości, a nie o stałej.

Czy dozwolone wartości kolumny zawsze będą wybierane z wcześniej ustalonego zbioru wartości? Jeżeli tak, wtedy dobrym wyborem może być typ `ENUM` lub `SET`.

Typ ENUM będzie również użyteczny, gdy masz niewielki zbiór ciągów tekstowych, które chcesz sortować w pewnej nieleksykalnej kolejności. Sortowanie wartości ENUM następuje zgodnie z kolejnością umieszczenia poszczególnych wartości typu wyliczeniowego na liście w definicji kolumny. Dzięki temu wartości możesz sortować w dowolnej kolejności.

Czy wartości dopełnienia są ważne? Jeżeli wartości muszą być pobierane w dokładnie takiej postaci, w jakiej są przechowywane, bez usuwania lub dodawania spacji na końcu (lub bajtów 0x00 w binarnych typach danych) podczas wstawiania lub pobierania danych, wtedy użyj typu TEXT lub VARCHAR dla niebinarnych ciągów tekstowych i BLOB lub VARBINARY dla binarnych ciągów tekstowych. Wspomniany czynnik jest ważny również podczas przechowywania skompresowanych, skróconych lub zaszyfrowanych wartości obliczanych w sposób, który może spowodować dodanie spacji na końcu przez metodę kodującą. W tabeli 3.11 przedstawiono sposób, w jaki baza danych MySQL obsługuje dopełnianie podczas operacji wstawiania i pobierania danych typu ciągu tekstowego.

Tabela 3.11. Obsługa dopełnień w typach danych ciągu tekstowego

Typ danych	Pamięć masowa	Pobieranie	Wynik
CHAR	Dopełnienie spacjami	Obcinanie spacji	Pobrane wartości nie mają spacji dodanych jako dopełnienie.
BINARY	Dopełnienie bajtami 0x00	Brak akcji	Pobrane wartości zachowują spacje dodane jako dopełnienie.
VARCHAR, VARBINARY	Brak akcji	Brak akcji	Dopełnienie na końcu nie ulega zmianie.
TEXT, BLOB	Brak akcji	Brak akcji	Dopełnienie na końcu nie ulega zmianie.

Włączenie trybu SQL o nazwie PAD_CHAR_TO_FULL_LENGTH powoduje, że wartości pobierane z kolumny typu CHAR zachowują spacje dodane na końcu.

3.2.6. Typy danych dla wartości daty i czasu

MySQL oferuje typy danych DATE, TIME, DATETIME, TIMESTAMP i YEAR, przeznaczonych do przechowywania wartości daty i godziny. W wersji MySQL 5.6 wprowadzono pewne istotne zmiany:

- W wersji MySQL 5.6.4 wprowadzono obsługę części ułamkowych sekund dla typów danych TIME, DATETIME i TIMESTAMP. Dozwolone stało się zastosowanie opcjonalnej części ułamkowej składającej się z maksymalnie sześciu cyfr, co zapewnia precyzję na poziomie mikrosekund.
- W wersji MySQL 5.6.5 wprowadzono rozszerzoną obsługę automatycznego użycia bieżącego znacznika czasu jako wartości początkowej dla uaktualnień. W starszych wersjach wspomniane właściwości są dostępne dla co najwyżej pojedynczej kolumny typu TIMESTAMP w tabeli. Począwszy od tej wersji, mogą być używane z dowolnymi kolumnami typu TIMESTAMP i DATETIME.

- W wersji MySQL 5.6.6 porzucono typ YEAR(2), zamiast niego tworzone są kolumny YEAR(4).

W tabeli 3.12 wymieniono typy danych przeznaczone do obsługi wartości daty i godziny wraz z zakresem ich prawidłowych wartości. W zakresie odzwierciedlono wprowadzoną w MySQL 5.6.4 obsługę części ułamkowych sekund (fsp). W przypadku wersji wcześniejszych niż 5.6.4 zignoruj części ułamkowe.

Tabela 3.12. Zakresy typów danych przechowujących wartości daty i godziny

Nazwa typu	Zakres
DATE	od '1000-01-01' do '9999-12-31'
TIME	od '-838:59:59[.000000]' do '838:59:59[.000000]'
DATETIME	od '1000-01-01 00:00:00[.000000]' do '9999-12-31 23:59:59[.999999]'
TIMESTAMP	od '1970-01-01 00:00:01[.000000]' do '2038-01-19 03:14:07[.999999]'
YEAR	od 1901 do 2155

Aby dla wartości daty i godziny zadeklarować kolumnę wraz z częścią ułamkową, definicję trzeba podać w postaci *nazwa_typu(fsp)*, gdzie *nazwa_typu* to TIME, DATETIME lub TIMESTAMP, natomiast *fsp* to precyzja wyrażona w ułamkach sekund. Na przykład, poniższe kolumny typu TIME oferują precyzję na poziomie odpowiednio trzech i sześciu cyfr w części ułamkowej:

```
t1 TIME(3)
t2 TIME(6)
```

Wartość *fsp* musi być z zakresu od 0 do 6. Jeżeli nie zostanie podana, domyślnie wynosi 0. Więcej informacji na ten temat znajdziesz w podpunkcie 3.2.6.5, zatytułowanym „Część ułamkowa w typach przechowujących datę i godzinę”.

W tabeli 3.13 wymieniono wymagania w zakresie pamięci masowej dla każdego typu przechowującego datę i godzinę. Z kolei w tabeli 3.14 wymieniono wymagania dla typów zdefiniowanych wraz z częścią ułamkową.

Tabela 3.13. Ilość pamięci masowej wymaganej do przechowywania typów danych daty i godziny

Nazwa typu	Wymagana ilość pamięci masowej (przed wersją 5.6.4)	Wymagana ilość pamięci masowej (od wersji 5.6.4)
DATE	3 bajty	3 bajty
TIME	3 bajty	3 bajty + <i>fsp</i>
DATETIME	8 bajtów	5 bajtów + <i>fsp</i>
TIMESTAMP	4 bajty	4 bajty + <i>fsp</i>
YEAR	1 bajt	1 bajt

Tabela 3.14. Ilość pamięci masowej wymaganej przez część ułamkową

Precyzja wyrażona liczbą cyfr części ułamkowej	Wymagana ilość pamięci masowej
0	0 bajtów
1, 2	1 bajt
3, 4	2 bajty
5, 6	3 bajty

W tabeli 3.15 przedstawiono wartość „zero” przechowywaną przez poszczególne typy danych, gdy wstawiona zostanie wartość uznawana za nieprawidłową w danym typie. Jeżeli wolisz, aby wartości nieprawidłowe były traktowane jako błędy i odrzucane, ustaw odpowiedni tryb SQL. W tym celu zapoznaj się z podrozdziałem 3.3, zatytułowanym „Jak MySQL obsługuje nieprawidłowe wartości danych?”. Wspomniana wartość „zero” jest również wartością domyślną dla kolumny daty i godziny, które zostały zdefiniowane wraz z atrybutem NOT NULL.

Tabela 3.15. Wartości „zero” dla typów danych przechowujących datę i godzinę

Nazwa typu danych	Wartość zero
DATE	'0000-00-00'
TIME	'00:00:00 [.000000]'
DATETIME	'0000-00-00 00:00:00[.000000]'
TIMESTAMP	'0000-00-00 00:00:00 [.0000]'
YEAR	0000

MySQL przedstawia datę w formacie rok-miesiąc-dzień, czyli zgodnie ze standardem SQL i specyfikacją ISO 8601. Na przykład, dzień 3 grudnia 2015 jest zapisywany w postaci '2015-12-03'. Podczas pobierania daty wartości daty i godziny można wyświetlić w wielu różnych formatach za pomocą funkcji DATE_FORMAT() i TIME_FORMAT().

W przypadku danych wejściowych w postaci dat MySQL pozostawia pewną swobodę w zakresie, w jakim można je przedstawić. Na przykład, przeprowadzana jest konwersja z roku wyrażonego dwiema cyframi na zapisany czterema. Ponadto, nie ma konieczności stosowania początkowego zera dla wartości miesiąca i dnia mniejszych niż 10. Jednak wartości trzeba podać w kolejności rok-miesiąc-dzień. Inne formaty, takie jak '12-3-99' lub '3-12-99', nie są obsługiwane. W takich przypadkach użyteczna może okazać się funkcja STR_TO_DATE(), przeprowadzająca konwersję ciągu tekstowego w formacie innym niż ISO na datę w formacie ISO. Na przykład, jeżeli tabela mytb1 ma kolumnę daty o nazwie date_col, wartości można wstawiać w następujący sposób:

```
mysql> INSERT INTO mytb1 (date_col)
-> VALUES(STR_TO_DATE('12-3-99', '%m-%d-%Y'));
mysql> SELECT * FROM mytb1;
```

```
+-----+
| date_col |
+-----+
| 1999-12-03 |
+-----+
```

W podpunkcie 3.2.6.7, zatytułowanym „Praca z wartościami daty i godziny”, znajdziesz dalsze omówienie stosowanych przez bazę danych MySQL reguł interpretacji daty.

3.2.6.1. Typy danych DATE, TIME i DATETIME

Typy danych DATE, TIME i DATETIME są przeznaczone do przechowywania wartości odpowiednio daty, godziny oraz połączonych wartości daty i godziny. Formaty dla wymienionych trzech typów są następujące: 'CCYY-MM-DD', 'hh:mm:ss[.uuuuuu]' i 'CCYY-MM-DD hh:mm:ss[.uuuuuu]', gdzie CC, YY, MM, DD, hh, mm, ss i uuuuuu oznacza kolejno wiek, rok, miesiąc, dzień miesiąca, godzinę, minutę, sekundę i mikrosekundę. Przed wersją MySQL 5.6.4 części ułamkowe dla wartości TIME i DATETIME były dozwolone, ale odrzucane podczas ich wstawiania do bazy danych.

Począwszy od MySQL 5.6.5, kolumny typu DATETIME mogą automatycznie używać bieżącego znacznika czasu jako wartości początkowej dla uaktualnień. Więcej informacji na ten temat znajdziesz w podpunkcie 3.2.6.6, zatytułowanym „Automatyczne właściwości dla typów wartości daty i godziny”.

Jeżeli kolumnie typu DATETIME przypiszesz wartość DATE, wtedy MySQL automatycznie dołączy fragment '00:00:00'. Konwersja działa również w drugą stronę. W przypadku przypisania wartości DATETIME kolumnie typu DATE lub TIME baza danych MySQL odrzuci nadmiarową część:

```
mysql> CREATE TABLE t (dt DATETIME, d DATE, t TIME);
mysql> INSERT INTO t (dt, d, t) VALUES(NOW(), NOW(), NOW());
mysql> SELECT * FROM t;
+-----+-----+-----+
| dt                | d                | t                |
+-----+-----+-----+
| 2013-01-09 15:33:24 | 2013-01-09      | 15:33:24        |
+-----+-----+-----+
```

Konwersja wartości TIME na DATETIME zależy od wersji: począwszy od MySQL 5.6.4, godzina będzie dodawana do bieżącej daty. W starszych wersjach serwera konwersja niekoniecznie wygeneruje użyteczny wynik.

MySQL traktuje godzinę w wartościach DATETIME i TIME nieco odmiennie. W przypadku DATETIME część przedstawiająca godzinę musi być w formacie z zakresu od '00:00:00' do '23:59:59'. Z kolei wartość TIME przedstawia czas, który upłynął. Dlatego też zakres wymieniony w tabeli 3.12 dla kolumn TIME zawiera wartości ujemne i większe niż '23:59:59'.

Zachowaj ostrożność podczas wstawiania do tabeli „krótkich” wartości TIME, ponieważ mogą zostać one zinterpretowane inaczej, niż sądzisz. Na przykład, prawdopodobnie przekonałeś się już, że po wstawieniu '30' i '12:30' do kolumny TIME jedna wartość zostanie zinterpretowana od prawej do lewej strony, natomiast druga od lewej do prawej, co spowoduje wstawienie wartości '00:00:30' i '12:30:00'. Jeżeli chcesz, aby wartość

'12:30' została potraktowana jako 12 minut i 30 sekund, to musisz użyć w pełni kwalifikowanej formy, czyli '00:12:30'.

3.2.6.2. Typ danych TIMESTAMP

Typ `TIMESTAMP` jest przeznaczony do przechowywania połączonych wartości daty i godziny. (Słowo „timestamp”, czyli znacznik czasu, może się kojarzyć jedynie z godziną, ale tak nie jest w omawianym przypadku). Typ danych `TIMESTAMP` ma specjalne właściwości, które zostaną tutaj omówione.

Kolumny typu `TIMESTAMP` mogą przechowywać wartości z zakresu od '1970-01-01 00:00:01[.00000]' do '2038-01-19 03:14:07[.99999]'. Podobnie jak w przypadku typu `DATETIME`, przed MySQL 5.6.4 część ułamkowa wartości `TIMESTAMP` była dozwolona, ale odrzucana podczas wstawiania wartości do bazy danych. Zakres jest powiązany z czasem UNIX, gdzie pierwszy dzień roku 1970 jest uznawany za „dzień zero”, nazywany również „początkiem epoki”. Baza danych MySQL przechowuje każdą wartość `TIMESTAMP` jako 4-bajtową liczbę sekund, które upłynęły od początku epoki. Początek roku 1970 określa dolny koniec zakresu `TIMESTAMP`. (Zwróć uwagę, że zakres `TIMESTAMP` nie rozpoczyna się od '1970-01-01 00:00:00'. Mógłbyś oczekiwać, że będzie wewnętrznie jako 0 sekund od początku epoki, ale 0 jest używane do przedstawienia „zerowego” znacznika czasu, czyli '0000-00-00 00:00:00'). Górny koniec zakresu odpowiada maksymalnej 4-bajtowej wartości dla czasu UNIX.

MySQL przechowuje wartości `TIMESTAMP` jako uniwersalny czas koordynowany (ang. *Universal Coordinated Time*, UTC). Podczas przechowywania tego rodzaju wartości serwer konwertuje ją ze strefy czasowej sesji na UTC. Gdy później pobierasz wartość, serwer z powrotem konwertuje ją z UTC na strefę czasową sesji, a więc wyświetlana jest ta sama wartość, która znajduje się w bazie danych. Jednak jeśli inny klient nawiąże połączenie z serwerem i będzie używał innej strefy czasowej, po pobraniu wartości zostanie ona dostosowana do jego strefy czasowej. Ten efekt możesz zaobserwować w ramach pojedynczej sesji, jeśli zmienisz strefę czasową:

```
mysql> CREATE TABLE t (ts TIMESTAMP);
mysql> SET time_zone = '+00:00'; # Wybór strefy czasowej UTC.
mysql> INSERT INTO t VALUES('2000-01-01 00:00:00');
mysql> SELECT ts FROM t;
+-----+
| ts                |
+-----+
| 2000-01-01 00:00:00 |
+-----+
mysql> SET time_zone = '+03:00'; # Ustawienie przesunięcia o 3 godziny w strefie czasowej.
mysql> SELECT ts FROM t;
+-----+
| ts                |
+-----+
| 2000-01-01 03:00:00 |
+-----+
```

Powyższe przykłady wskazują strefy czasowe za pomocą wyrażonej w godzinach i minutach wartości przesunięcia względem UTC. Istnieje również możliwość użycia nazw stref czasowych, na przykład 'Europe/Zurich', o ile tabele stref czasowych serwera zostały skonfigurowane zgodnie z opisem przedstawionym w punkcie 12.6.1, zatytułowanym „Konfiguracja obsługi stref czasowych”.

Kolumny typu `TIMESTAMP` mogą automatycznie używać bieżącego znacznika czasu jako wartości początkowej dla uaktualnień. Ponadto, wstawienie wartości `NULL` do kolumny typu `TIMESTAMP` powoduje przypisanie jej wartości bieżącego znacznika czasu, o ile kolumna nie została zdefiniowana wraz z atrybutem `NULL`, zezwalającym na przechowywanie wartości `NULL`. Więcej informacji na ten temat znajdziesz w podpunkcie 3.2.6.6, zatytułowanym „Automatyczne właściwości dla typów wartości daty i godziny”.

3.2.6.3. Typ danych YEAR

Typ `YEAR` to 1-bajtowy typ danych przeznaczony do efektywnego przechowywania wartości roku. Definicja kolumny typu `YEAR` może zawierać specyfikację określającą szerokość kolumny (*M*), która może wynosić 2 lub 4. Jeżeli pominiesz *M* w definicji kolumny, domyślna szerokość wynosi 4. Zakres typu `YEAR` to od 1901 do 2155. Typ `YEAR` jest wystarczający do przechowywania informacji o dacie wymagającej jedynie roku, na przykład rok urodzenia, rok wyboru na stanowisko itd. Jeżeli nie jest wymagana pełna wartość daty, typ `YEAR` jest pod względem wymaganej pamięci masowej znacznie bardziej efektywny niż pozostałe typy danych przeznaczone do obsługi daty.

W przypadku zdefiniowania kolumny jako `YEAR(2)` tylko dwie ostatnie cyfry będą wyświetlane. Tego rodzaju rozwiązanie jest przeznaczone do przechowywania jedynie wartości z zakresu od 1970 do 2069. Jeżeli użyjesz `YEAR(2)` do przechowywania wartości spoza wymienionego zakresu, wtedy wyświetlane wartości mogą być mylące. Na przykład, wartości 1970 i 2070 przechowywane w kolumnie `YEAR(2)` wyświetlane są jako 70. Najłatwiejszym sposobem uniknięcia tego rodzaju problemu jest definiowanie kolumny jako `YEAR(4)`. Z powodu wspomnianych problemów typ `YEAR(2)` został w MySQL 5.6.6 uznany za przestarzały. W istniejących kolumnach typ `YEAR(2)` jest traktowany bez zmian, natomiast w nowych tabelach tak zdefiniowane kolumny są tworzone jako typu `YEAR(4)`.

MySQL konwertuje dwucyfrowe wartości `YEAR` na czterocyfrowe za pomocą zdefiniowanej w serwerze reguły zgadywania daty (patrz podpunkt 3.2.6.8, zatytułowany „Interpretacja niejednoznacznych wartości roku”). Na przykład, 97 i 14 stają się 1997 i 2014. Jednak pamiętaj, że wstawienie wartości liczbowej 00 do czterocyfrowej kolumny typu `YEAR` spowoduje przypisanie jej wartości 0000, a nie 2000. Jeżeli wartość 00 chcesz skonwertować na 2000, to musisz ją podać w ciągu tekstowym jako '0' lub '00'.

Typ `TINYINT` ma takie same wymagania w zakresie pamięci masowej (1 bajt) jak `YEAR`, ale zupełnie inny zakres. Aby za pomocą typu liczby całkowitej pokryć taki sam zakres lat jak w przypadku typu `YEAR`, konieczne jest użycie `SMALLINT`, która jednak zabiera dwukrotnie większą ilość pamięci masowej. Jeżeli zakres lat koniecznych do przedstawienia pokrywa się z zakresem typu `YEAR`, pod względem wymaganej pamięci masowej typ `YEAR` będzie znacznie efektywniejszy niż `SMALLINT`.

3.2.6.4. Atrybuty typu danych przechowującego daty

Definicje kolumn przechowujących daty mogą zawierać ogólne atrybuty NULL i NOT NULL. Jeżeli nie wskażesz żadnego, domyślnie stosowany będzie NULL, za wyjątkiem typu TIMESTAMP, dla którego domyślnym argumentem jest NOT NULL.

Istnieje możliwość zdefiniowania wartości domyślnej za pomocą klauzuli DEFAULT. W punkcie 3.2.3, zatytułowanym „Definiowanie wartości domyślnych kolumn”, omówiono reguły stosowane przez MySQL podczas przypisywania wartości domyślnych, gdy definicja kolumny nie zawiera klauzuli DEFAULT.

W większości przypadków wartość domyślna musi być stałą. Za wyjątkiem typu TIMESTAMP i (począwszy od MySQL 5.6.5) DATETIME nie można użyć funkcji takiej jak CURRENT_TIMESTAMP w celu dostarczenia wartości „bieżącej daty i godziny” jako wartości domyślnej kolumny. Kolumny typu TIMESTAMP i DATETIME są specjalne, ponieważ wartością domyślną może być bieżąca data i godzina. (Aby poznać reguły przypisywania wartości domyślnych wymienionym typom, zapoznaj się z podpunktem 3.2.6.6, zatytułowanym „Automatyczne właściwości dla typów wartości daty i godziny”). W celu osiągnięcia tego rodzaju efektu dla innych typów podczas tworzenia nowego rekordu wartość kolumny trzeba wyraźnie podać jako CURRENT_TIMESTAMP. Alternatywne rozwiązanie polega na użyciu kolumny typu TIMESTAMP lub DATETIME bądź też na konfiguracji wyzwalacza, który inicjalizuje kolumnie odpowiednią wartość — zapoznaj się z punktem 4.2.3, zatytułowanym „Wyzwalacze”.

3.2.6.5. Część ułamkowa w typach przechowujących datę i godzinę

W tym podpunkcie dowiesz się, jak MySQL obsługuje ułamkowe części sekund w wartościach określających datę i godzinę. Zaprezentowane tutaj informacje mają zastosowanie głównie w wersji MySQL 5.6.4 i nowszych.

Składnia deklarowania typu TIME, DATETIME i TIMESTAMP pozwala na zdefiniowanie opcjonalnej precyzji części ułamkowej sekund (*fsp*) o wielkości do sześciu cyfr. Wspomniana wartość *fsp* musi być z zakresu od 0 do 6, przy czym 0 oznacza brak części ułamkowej, natomiast 6 wskazuje precyzję na poziomie mikrosekund. Jeżeli *fsp* nie zostanie podane, to wartością domyślną jest 0. Na przykład, zapytania TIME i TIME(0) mają taki sam efekt, czyli brak części ułamkowej. Z kolei DATETIME(1) pozwala na stosowanie wartości daty i godziny z dokładnością (precyzją) jednej dziesiątej sekundy. Natomiast TIMESTAMP(6) pozwala na używanie znaczników czasu z pełną dokładnością na poziomie mikrosekund.

Funkcje pobierające argumenty w postaci wartości daty i godziny lub zwracające tego rodzaju wartości pobierają lub zwracają wartości wraz z częścią ułamkową. W pewnych przypadkach funkcje niepobierające argumentów w wersjach wcześniejszych niż MySQL 5.6.4, w późniejszych wersjach akceptują argument *fsp* w celu zapewnienia kontroli nad liczbą cyfr części ułamkowej, które mają się znajdować w wartości zwrótej. Na przykład, funkcja CURTIME() zwraca bieżącą godzinę bez części ułamkowej, natomiast CURTIME(3) zawiera część ułamkową o dokładności na poziomie tysięcznej części sekundy:

```
mysql> SELECT CURTIME(), CURTIME(3);
+-----+-----+
| CURTIME() | CURTIME(3) |
+-----+-----+
| 12:22:31 | 12:22:31.475 |
+-----+-----+
```

Opisy poszczególnych funkcji w punkcie C.2.5, zatytułowanym „Funkcje daty i godziny”, wskazują, czy argument *fsp* jest dozwolony.

W wersjach wcześniejszych niż MySQL 5.6.4 obsługa wartości mikrosekund była ograniczona. Pewne funkcje daty i czasu, takie jak `DATE_ADD()`, używały części ułamkowej, ale jeśli spróbowałeś wstawić do kolumny wartość zawierającą część ułamkową sekund, to serwer MySQL odrzucał tę część.

3.2.6.6. Automatyczne właściwości dla typów wartości daty i godziny

Kolumny typu `TIMESTAMP` i `DATETIME` mogą mieć właściwości automatycznej inicjalizacji i uaktualnienia:

- „Automatyczna inicjalizacja” oznacza, że w przypadku nowych rekordów kolumnie zostanie przypisany bieżący znacznik czasu, jeśli w zapytaniu `INSERT` nie będzie wyraźnie podana wartość kolumny.
- „Automatyczne uaktualnienie” oznacza uaktualnienie istniejących rekordów bieżącym znacznikiem czasu, gdy zmiana ulegnie wartość innej kolumny. (Przypisanie kolumnie jej bieżącej wartości się nie liczy; w rzeczywistości można to zrobić, aby uniemożliwić automatyczne uaktualnienie).

Przed wersją MySQL 5.6.5 istniała możliwość zdefiniowania w tabeli dowolnej pojedynczej kolumny typu `TIMESTAMP`, która mogła mieć jedną lub obie wymienione właściwości. Nie można było zdefiniować automatycznej inicjalizacji dla jednej kolumny typu `TIMESTAMP` i automatycznego uaktualniania dla innej. Ponadto, nie było możliwości stosowania automatycznej inicjalizacji i uaktualniania w więcej niż tylko jednej kolumnie.

W wydaniu MySQL 5.6.5 rozszerzono i uogólniono obsługę właściwości automatycznych: dowolna kolumna typu `TIMESTAMP` może mieć dowolną z nich lub obie. To samo dotyczy również kolumn typu `DATETIME`.

Inna specjalna właściwość, która ma zastosowanie tylko dla kolumn typu `TIMESTAMP`, jest taka, że po przypisaniu kolumnie wartości `NULL` może ona zostać zastąpiona przez bieżący znacznik czasu. Aby zezwolić na przechowywanie wartości `NULL` w kolumnie typu `TIMESTAMP`, konieczne jest jej zdefiniowanie wraz z atrybutem `NULL`.

Użyj poniższej składni w celu zdefiniowania kolumny typu `TIMESTAMP`:

```
nazwa_kolumny TIMESTAMP [DEFAULT wartość_domyślna] [ON UPDATE
CURRENT_TIMESTAMP]
```

Atrybuty `DEFAULT` i `ON UPDATE` można wymienić w dowolnej kolejności, jeśli oba są stosowane. Wartością domyślną może być `CURRENT_TIMESTAMP`, wartość stała taka jak `0` lub wartość w formacie `'RRRR-MM-DD gg:mm:ss '`. Dozwolone jest stosowanie synonimów `CURRENT_TIMESTAMP`, na przykład `NOW()`.

Począwszy od MySQL 5.6.5, kolumna typu DATETIME może mieć zdefiniowane te same atrybuty DEFAULT i ON UPDATE. Przed wydaniem MySQL 5.6.5 kolumna typu DATETIME dla atrybutu DEFAULT mogła mieć jedynie wartość stałą i nie zezwalała na użycie ON UPDATE.

Aby użyć jednej lub obu właściwości automatycznych dla pierwszej kolumny typu TIMESTAMP, w tabeli można ją zdefiniować za pomocą połączenia atrybutów DEFAULT i ON UPDATE:

- Po użyciu DEFAULT CURRENT_TIMESTAMP kolumna ma zdefiniowaną automatyczną inicjalizację. Będzie miała również automatyczne uaktualnianie, gdy użyty będzie atrybut ON UPDATE CURRENT_TIMESTAMP.
- Bez żadnego z wymienionych atrybutów MySQL definiuje kolumnę zarówno z DEFAULT CURRENT_TIMESTAMP, jak i ON UPDATE CURRENT_TIMESTAMP.
- Po użyciu atrybutu DEFAULT *wartość stałej* kolumna nie będzie miała automatycznej inicjalizacji. Automatyczne uaktualnienie będzie po użyciu atrybutu ON UPDATE CURRENT_TIMESTAMP.
- Bez atrybutu DEFAULT, ale z atrybutem ON UPDATE CURRENT_TIMESTAMP wartością domyślną jest 0, a kolumna ma automatyczne uaktualnianie.

W celu użycia automatycznej inicjalizacji lub uaktualnienia dla kolumny TIMESTAMP innej niż pierwsza w wersji MySQL wcześniejszej niż 5.6.5 konieczne jest zdefiniowanie pierwszej z wyrażnie podanym atrybutem DEFAULT *wartość stałej* i bez ON UPDATE CURRENT_TIMESTAMP. Następnie można użyć DEFAULT CURRENT_TIMESTAMP lub ON UPDATE CURRENT_TIMESTAMP (bądź obu) z inną dowolną pojedynczą kolumną typu TIMESTAMP.

Począwszy od MySQL 5.6.5, można dowolnie używać dowolnego lub obu atrybutów wraz z dowolną kolumną typu TIMESTAMP. Ponadto, kolumny typu DATETIME również mogą mieć zdefiniowane omówione atrybuty.

Aby zawieść automatyczną inicjalizację lub uaktualnienie dla kolumny typu TIMESTAMP lub DATETIME, dla której zdefiniowano te właściwości, kolumnie należy wyraźnie przypisać wartość przeznaczoną do użycia w operacji wstawiania lub uaktualniania. Na przykład, w celu uniemożliwienia przeprowadzenia uaktualnienia po zmianie wartości kolumny należy przypisać jej wartość bieżącą.

Definicje kolumn typu TIMESTAMP i DATETIME mogą zawierać także atrybuty NULL i NOT NULL. Atrybutem domyślnym dla kolumny typu TIMESTAMP jest NOT NULL. To ma specjalne działanie w przypadku, gdy kolumnie wyraźnie przypiszesz wartość NULL: w takiej sytuacji MySQL wstawi bieżący znacznik czasu. (Dotyczy to operacji zarówno wstawiania, jak i uaktualniania danych). Jeśli w definicji kolumny użyjesz atrybutu NULL, przypisanie kolumnie wartości NULL spowoduje jej wstawienie do tabeli zamiast bieżącego znacznika czasu. Domyślnym atrybutem dla DATETIME jest NULL, przypisanie kolumnie typu DATETIME wartości NULL nie ma żadnego specjalnego efektu.

Poniżej przedstawiono tabelę zawierającą kolumnę, której będzie przypisywany bieżący znacznik czasu po wstawieniu każdego nowego rekordu. Później ta kolumna nie będzie automatycznie uaktualniana:


```
CREATE TABLE t1
(
    ts_created TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    ... inne kolumny ...
);
```

W celu utworzenia nowego rekordu trzeba zainicjalizować kolumnę z bieżącym znacznikiem czasu przez przypisanie jej wartości NULL lub pominięcie jej w zapytaniu INSERT. Kolumna zachowa wartość w trakcie kolejnych operacji uaktualnienia, chyba że wartość kolumny zostanie wyraźnie zmieniona.

Poniżej przedstawiono tabelę zawierającą kolumny dla wartości zarówno godziny utworzenia rekordu, jak i godziny jego ostatniej modyfikacji. Obie wspomniane kolumny są typu `TIMESTAMP`:

```
CREATE TABLE t2
(
    ts_created TIMESTAMP DEFAULT 0,
    ts_modified TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        ON UPDATE CURRENT_TIMESTAMP,
    ... inne kolumny...
);
```

Aby wstawić nowy rekord, obu kolumnom typu `TIMESTAMP` przypisz wartość NULL, co spowoduje wstawienie bieżącego znacznika czasu. Aby uaktualnić istniejący rekord, w zapytaniu pomin obie kolumny. W takim przypadku kolumna `ts_modified` zostanie uaktualniona automatycznie bieżącym znacznikiem czasu, gdy zmianie ulegnie wartość którejkolwiek z pozostałych kolumn.

3.2.6.7. Praca z wartościami daty i godziny

Wartości danych wejściowych dla kolumn daty i godziny MySQL interpretuje w wielu różnych formatach, między innymi w postaci ciągu tekstowego i liczb. Ponadto, począwszy od wersji MySQL 5.6.4, opcjonalną część ułamkową składającą się maksymalnie z sześciu cyfr (mikrosekundy) można stosować w typach danych `TIME`, `DATETIME` i `TIMESTAMP`.

Na przykład, poniższe formaty wartości są dozwolone do stosowania w typach `DATETIME` i `TIMESTAMP`:

```
'CCYY-MM-DD hh:mm:ss[.uuuuuu]'
```

```
'YY-MM-DD hh:mm:ss[.uuuuuu]'
```

```
'CCYYMMDDhhmmss[.uuuuuu]'
```

```
'YYMMDDhhmmss[.uuuuuu]'
```

```
CCYYMMDDhhmmss[.uuuuuu]
```

```
YYMMDDhhmmss[.uuuuuu]
```

Dla wartości `DATE`, `TIME` i `YEAR` istnieją analogiczne odpowiedniki.

Format pozbawiony części wskazującej wiek (`CC`) MySQL interpretuje z użyciem reguł omówionych w podpunkcie 3.2.6.8, zatytułowanym „Interpretacja niejednoznacznych wartości roku”. W przypadku formy ciągu tekstowego zawierającego znaki ograniczające nie trzeba używać myślnika w dacie i dwukropka w godzinie. Dowolny znak przestankowy może być znakiem ograniczającym. Interpretacja wartości zależy od kontekstu, a nie od

znaku ograniczającego. Na przykład, wprawdzie w godzinie zwykle jest stosowany ogranicznik w postaci dwukropka, ale MySQL nie zinterpretuje jako godziny wartości zawierającej dwukropkę, jeśli będzie ona znajdowała się w kontekście, w którym oczekiwano podania daty. Ponadto, w przypadku form w postaci ciągu tekstowego wraz z ogranicznikami nie trzeba używać dwóch cyfr do wskazania mniejszej niż 10 wartości miesiąca, dnia, godziny, minuty lub sekundy. Wszystkie poniższe ciągi tekstowe powodują zdefiniowanie tej samej wartości daty i godziny:

```
'2012-02-03 05:04:09'
'2012-02-03 05:04:9'
'2012-02-03 05:4:9'
'2012-02-03 5:4:9'
'2012-02-3 5:4:9'
'2012-2-3 5:4:9'
```

Wartości wraz z zerami na początku mogą być przez MySQL interpretowane na różne sposoby, w zależności od tego, czy zostały określone jako ciągi tekstowe, czy jako liczby. Ciąg tekstowy '001231' będzie uznany za sześciocyfrową wartość interpretowaną jako '2000-12-31' w kolumnie DATE i '2000-12-31 00:00:00' w kolumnie DATETIME. Z drugiej strony, liczba 001231 będzie uznana za liczbę 1231, co może spowodować pewne problemy. Dlatego też najlepszym rozwiązaniem jest podawanie wartości w postaci ciągu tekstowego '001231' lub w pełni kwalifikowanej wartości, jeśli używane są liczby (to znaczy 20001231 dla DATE i 200012310000 dla DATETIME).

Ogólnie rzecz biorąc, masz możliwość dowolnego przypisywania wartości między typami DATE, DATETIME i TIMESTAMP, ale pamiętaj o kilku ograniczeniach:

- Jeżeli kolumnie typu DATE przypiszesz wartość DATETIME lub TIMESTAMP, wtedy człon godziny zostanie odrzucony.
- Jeżeli wartość DATE przypiszesz kolumnie DATETIME lub TIMESTAMP, wówczas godzina zostanie ustawiona jako zero ('00:00:00').
- Typy mają różne zakresy. W szczególności, typ TIMESTAMP ma najbardziej ograniczony zakres (od 1970 do 2038). Kolumnie typu TIMESTAMP nie możesz przypisać wartości DATETIME określającej rok wcześniejszy niż 1970 i oczekiwać rozsądnych wyników. Ponadto, kolumnie typu TIMESTAMP nie można przypisać wartości zbyt wybiegających w przyszłość.

MySQL dostarcza wiele funkcji przeznaczonych do pracy z wartościami daty i godziny. Więcej informacji na ich temat znajdziesz w dodatku C, zatytułowanym „Przewodnik po operatorach i funkcjach”.

3.2.6.8. Interpretacja niejednoznacznych wartości roku

W przypadku wszystkich typów danych zawierających informacje o roku (DATE, DATETIME, TIMESTAMP i YEAR) MySQL obsługuje wartości dwucyfrowe roku przez ich konwersję na czterocyfrowe:

- Wartości roku z zakresu od 00 do 69 są konwertowane na od 2000 do 2069.
- Wartości roku z zakresu od 70 do 99 są konwertowane na od 1970 do 1999.

Efekt zastosowania powyższych reguł możesz łatwo sprawdzić, umieszczając w kolumnie typu YEAR(4) dwucyfrowe wartości roku, a następnie je pobierając i wyświetlając:

```
mysql> CREATE TABLE y_table (y YEAR(4));
mysql> INSERT INTO y_table VALUES(68),(69),(99),(00),('00');
mysql> SELECT * FROM y_table;
```

y
2068
2069
1999
0000
2000

Powyższy przykład zademonstrował również coś, na co powinieneś zwrócić uwagę: wartość 00 jest konwertowana na 0000, a nie na 2000. Jeżeli do kolumny typu YEAR(4) wstawiś liczbowe zero, wspomniane zero będzie później pobrane. Aby otrzymać 2000 za pomocą wartości niezawierającej informacji o wieku, konieczne jest wstawienie ciągu tekstowego '0' lub '00'. W celu zagwarantowania, że MySQL zinterpretuje wartość jako ciąg tekstowy, a nie liczbę, wartość YEAR wstawiaj za pomocą funkcji CAST(*wartość* AS CHAR). To spowoduje wygenerowanie ciągu tekstowego niezależnie od tego, czy *wartość* jest ciągiem tekstowym, czy liczbą.

Pamiętaj, że reguły przeprowadzania konwersji dwucyfrowej wartości roku na czterocyfrową opierają się jedynie na zgadywaniu. Nie ma żadnej gwarancji, że MySQL prawidłowo zinterpretuje znaczenie wartości roku, jeśli zostanie on podany w postaci dwucyfrowej bez wskazania wieku. Stosowane przez MySQL reguły są odpowiednie w większości sytuacji, ale jeśli nie powodują wygenerowania oczekiwanych wartości, wtedy musisz podać jednoznaczną datę wraz z czterocyfrową wartością roku. Na przykład, aby w tabeli *president* zawierającej informacje o wszystkich prezydentach USA, począwszy od XVIII wieku, umieścić daty ich urodzenia i śmierci, wartości roku podano w postaci czterocyfrowej. Ponieważ wartości w kolumnach daty urodzenia i śmierci dotyczą kilku wieków, pozwolenie bazie danych MySQL na odgadywanie wieku na podstawie dwucyfrowej wartości roku jest błędnym rozwiązaniem.

3.3. Jak MySQL obsługuje nieprawidłowe wartości danych?

Historycznie, dominującą w MySQL zasadą dotyczącą obsługi danych było „śmieci na wejściu, śmieci na wyjściu”. Innymi słowy, baza danych MySQL próbowała przechowywać wszelkie podane jej wartości danych, ale jeśli nie sprawdziłeś ich przed wstawieniem, otrzymane później dane mogły nie być tymi, których oczekiwałeś. Jednak kilka dostępnych trybów SQL pozwala na odrzucanie nieprawidłowych danych i generowanie komunikatu błędu. To jest zachowanie, którego prawdopodobnie oczekujesz, jeśli masz doświadczenie

w pracy z innymi systemami bazy danych. Poniższa analiza na początku zawiera informacje o domyślnym sposobie obsługi niepoprawnych danych przez MySQL. Następnie przejdziemy do zmian powodowanych przez włączenie trybów SQL, które mają wpływ na obsługę danych.

Domyślnie, wartości spoza zakresu lub niepoprawne pod innym względem są przez MySQL obsługiwane w następujący sposób:

- W przypadku kolumn liczbowych lub typu TIME wartości spoza zakresu są przycinane do najbliższego punktu końcowego zakresu, a następnie wstawiane do bazy danych.
- W przypadku kolumn przechowujących typy daty i godziny inne niż TIME nieprawidłowe wartości są konwertowane na odpowiednie dla danego typu wartości „zero” (patrz tabela 3.15).
- W przypadku kolumn typów ciągu tekstowego innych niż ENUM i SET zbyt długie ciągi tekstowe są skracane i dopasowywane do maksymalnej długości kolumny.
- Przypisania wartości kolumn typu ENUM i SET zależą od wartości zdefiniowanych w definicji kolumny jako poprawne. Jeżeli kolumnie typu ENUM przypiszesz wartość niewymienioną jako element typu wyliczeniowego, wtedy nastąpi przypisanie elementu błędu (czyli pustego ciągu tekstowego odpowiadającego elementowi oznaczonemu jako zerowy). Jeśli kolumnie typu SET przypiszesz wartość zawierającą podciągi tekstowe niewymienione jako elementy zbioru, wspomniane podciągi tekstowe będą odrzucone, a kolumnie zostanie przypisana wartość składająca się z pozostałych elementów.

MySQL zgłasza przeprowadzenie konwersji za pomocą ostrzeżenia wygenerowanego dla zapytań takich jak INSERT, REPLACE, UPDATE, LOAD DATA i ALTER TABLE. Aby wyświetlić wspomniane komunikaty ostrzeżeń po wykonaniu jednego z wymienionych zapytań, należy wykonać zapytanie SHOW WARNINGS.

W celu włączenia ścisłego sprawdzania wstawianych lub uaktualnianych wartości danych trzeba włączyć jeden z poniższych trybów SQL:

```
mysql> SET sql_mode = 'STRICT_ALL_TABLES';
mysql> SET sql_mode = 'STRICT_TRANS_TABLES';
```

W przypadku tabel transakcyjnych oba wymienione tryby są identyczne: po odkryciu nieprawidłowej lub brakującej wartości nastąpi zgłoszenie błędu, wykonywanie zapytania będzie przerwane, transakcja zostanie wycofana i nie będzie miała żadnego efektu w bazie danych. Z kolei dla tabel nietransakcyjnych powyższe tryby mają następujące efekty:

- W przypadku obu trybów odkrycie nieprawidłowej lub brakującej wartości w pierwszym rekordzie zapytania wstawiającego lub uaktualniającego rekordy spowoduje zgłoszenie błędu. Wykonywanie zapytania zostanie przerwane i nie będzie miało żadnego efektu w bazie danych, podobnie jak dla tabel transakcyjnych.
- Jeżeli błąd wystąpi po pierwszym rekordzie w zapytaniu wstawiającym lub uaktualniającym wiele rekordów, część z nich została już zmodyfikowana.

Dwa wymienione wcześniej tryby ściśle określają, kiedy nastąpi przerwanie wykonywania zapytania na tym etapie, a kiedy będzie ono kontynuowane:

- ◆ W trybie `STRICT_ALL_TABLES` nastąpi zgłoszenie błędu i przerwanie wykonywania zapytania. Ponieważ rekordy wcześniej przetworzone przez zapytanie zostały zmodyfikowane, wynikiem jest przeprowadzenie częściowego uaktualnienia.
- ◆ W trybie `STRICT_TRANS_TABLES` MySQL przerywa wykonywanie zapytania dla tabel nietransakcyjnych tylko wtedy, jeśli przerwanie będzie miało taki sam efekt, jak w przypadku tabel transakcyjnych. Dotyczy to tylko sytuacji, gdy błąd zostanie zgłoszony w pierwszym rekordzie, ponieważ błąd w dalszym rekordzie pozostawia wcześniejsze rekordy już zmodyfikowane. Wspomniane zmiany nie mogą być cofnięte w tabelach nietransakcyjnych, a więc MySQL kontynuuje wykonywanie zapytania, aby uniknąć częściowego uaktualnienia. Każda nieprawidłowa wartość zostaje skonwertowana na najbliższą prawidłową, zgodnie z opisem przedstawionym we wcześniejszej części podrödziału. W przypadku brakującej wartości MySQL przypisuje kolumnie wartość domyślną odpowiednią dla jej typu danych, zgodnie z opisem przedstawionym w punkcie 3.2.3, zatytułowanym „Definiowanie wartości domyślnych kolumn”.

Tryb ścisły w rzeczywistości nie powoduje włączenia najbardziej restrykcyjnego sprawdzania wartości, jakie może przeprowadzać MySQL. Istnieje możliwość włączenia dowolnego lub wszystkich wymienionych poniżej trybów, aby nałożyć dodatkowe ograniczenia na dane wejściowe:

- `ERROR_FOR_DIVISION_BY_ZERO` uniemożliwia wstawienie wartości, jeśli w trybie ścisłym wystąpi dzielenie przez zero. (W trybie innym niż ścisły zostanie wygenerowany komunikat ostrzeżenia i będzie wstawiona wartość `NULL`).
- `NO_ZERO_DATE` uniemożliwia w trybie ścisłym wstawianie wartości „zero” dla daty.
- `NO_ZERO_IN_DATE` uniemożliwia w trybie ścisłym wstawianie niekompletnych wartości daty, na przykład gdy komponent dnia lub miesiąca ma wartość zero.

Na przykład, w celu włączenia trybu ścisłego dla wszystkich silników bazy danych, a także sprawdzania pod kątem dzielenia przez zero tryb SQL ustaw w następujący sposób:

```
mysql> SET sql_mode = 'STRICT_ALL_TABLES,ERROR_FOR_DIVISION_BY_ZERO';
```

Aby włączyć tryb ścisły oraz wszystkie dodatkowe ograniczenia, wystarczy po prostu włączyć tryb `TRADITIONAL`:

```
mysql> SET sql_mode = 'TRADITIONAL';
```

Tryb `TRADITIONAL` jest skrótem oznaczającym „oba tryby ściśle plus dodatkowe ograniczenia”. Przypomina działanie innych „tradycyjnych” systemów SQL DBMS podczas sprawdzania danych.

Istnieje również możliwość powolnego luzowania trybu ścisłego pod pewnymi względami. Jeżeli włączysz tryb SQL o nazwie `ALLOW_INVALID_DATES`, wtedy MySQL nie będzie przeprowadzać pełnego sprawdzania komponentów daty. Zamiast tego wymagane

będzie jedynie, aby wartość miesiąca zawierała się w zakresie od 1 do 12, natomiast dnia w zakresie od 1 do 31 (co pozwala na użycie wartości nieprawidłowych, takich jak '2000-02-30' lub '2000-06-31'). Innym sposobem zawieszenia zgłaszania błędów jest użycie słowa kluczowego IGNORE w zapytaniach INSERT i UPDATE. Po dodaniu słowa kluczowego IGNORE do zapytania w przypadku nieprawidłowej wartości zamiast zgłoszenia błędu nastąpi jedynie wygenerowanie komunikatu ostrzeżenia.

Dostępne opcje dają elastyczność w zakresie wyboru odpowiedniego poziomu sprawdzania poprawności danych w aplikacji.

3.4. Praca z sekwencjami

Wiele aplikacji musi generować unikalne liczby w celach identyfikacyjnych: numery członków organizacji, próbek, lotów, identyfikatory klientów, zgłoszeń błędów itd.

Oferowanym przez MySQL mechanizmem generowania unikalnych liczb jest atrybut AUTO_INCREMENT dla kolumny pozwalający na automatyczne wygenerowanie kolejnych liczb. Jednak sposób obsługi kolumn AUTO_INCREMENT zależy od silnika bazy danych zastosowanego w MySQL. W tym podrozdziale przedstawiono ogólny sposób działania kolumn z atrybutem AUTO_INCREMENT, a także w poszczególnych silnikach bazy danych. Dzięki temu będziesz wiedział, jak efektywnie z nich korzystać i unikać pułapek, które czasami zaskakują użytkowników. W podrozdziale zaprezentowano również sposoby generowania sekwencji bez użycia kolumn z atrybutem AUTO_INCREMENT. Więcej informacji na temat silników bazy danych dostępnych w MySQL znajdziesz w punkcie 2.6.1, zatytułowanym „Cechy charakterystyczne silników bazy danych”.

3.4.1. Ogólne właściwości AUTO_INCREMENT

Kolumny AUTO_INCREMENT muszą być zdefiniowane zgodnie z poniższymi warunkami:

- W tabeli może znajdować się tylko jedna kolumna oznaczona atrybutem AUTO_INCREMENT i powinna być typu liczby całkowitej. (Atrybut AUTO_INCREMENT można zastosować także w kolumnie typu liczby zmiennoprzecinkowej, ale to bardzo rzadko spotykana sytuacja).
- Kolumna musi być indeksowa. Najczęściej używany jest indeks PRIMARY KEY lub UNIQUE, ale dozwolone jest również użycie indeksu nieunikalnego.
- Kolumna musi mieć zdefiniowane ograniczenie NOT NULL. MySQL nałoży ograniczenie NOT NULL, nawet jeśli wyraźnie nie zdefiniujesz wymienionego ograniczenia.

Po utworzeniu kolumna AUTO_INCREMENT zachowuje się w następujący sposób:

- Wstawienie wartości NULL do kolumny AUTO_INCREMENT powoduje, że MySQL wygeneruje kolejną liczbę w sekwencji i wstawi ją do kolumny. Sekwencje AUTO_INCREMENT normalnie zaczynają się od 1 i otrzymują kolejne wartości

(1, 2, 3 itd.) podczas wstawiania kolejnych rekordów do tabeli. W zależności od silnika bazy danych istnieje możliwość ustawienia lub wyzerowania kolejnego numeru sekwencji bądź też ponownego użycia wartości sekwencji przypisanych usuniętym rekordom.

- Aby pobrać ostatnio wygenerowany numer sekwencji, należy wywołać funkcję `LAST_INSERT_ID()`. W ten sposób w kolejnych zapytaniach możesz się odwołać do ostatniej wartości `AUTO_INCREMENT`, nawet nie znając jej dokładnej wartości. Funkcja `LAST_INSERT_ID()` zwraca wartość 0, jeśli w trakcie aktualnej sesji nie została wygenerowana żadna wartość `AUTO_INCREMENT`.

Funkcja `LAST_INSERT_ID()` jest powiązana jedynie z wartościami `AUTO_INCREMENT` wygenerowanymi w trakcie bieżącej sesji z serwerem. Istnieje możliwość wygenerowania liczby sekwencji, a następnie wywołania funkcji `LAST_INSERT_ID()` w celu jej późniejszego pobrania w trakcie tej samej sesji, nawet jeśli w międzyczasie inne klienty wygenerowały własne wartości sekwencji.

W przypadku obejmujących wiele rekordów zapytań `INSERT` i generujących wiele wartości `AUTO_INCREMENT`, funkcja `LAST_INSERT_ID()` zwraca ostatnią z nich. Jeżeli użyjesz `INSERT DELAYED`, wartość `AUTO_INCREMENT` nie będzie wygenerowana aż do chwili faktycznego wstawienia rekordu. W takim przypadku nie można polegać na funkcji `LAST_INSERT_ID()`.

- Wstawienie rekordu bez podania wartości dla kolumny `AUTO_INCREMENT` ma taki sam efekt jak podanie wartości `NULL` dla tej kolumny. Jeżeli `ai_col` jest kolumną ze zdefiniowanym atrybutem `AUTO_INCREMENT`, wtedy poniższe zapytania mają taki sam efekt:

```
INSERT INTO t (ai_col,name) VALUES(NULL,'abc');
INSERT INTO t (name) VALUES('abc');
```

- Domyślnie, wstawienie 0 do kolumny `AUTO_INCREMENT` ma taki sam efekt jak wstawienie wartości `NULL`. Po włączeniu trybu SQL o nazwie `NO_AUTO_VALUE_ON_ZERO` przypisanie wartości 0 powoduje wstawienie zera do kolumny, a nie kolejnej liczby w sekwencji.
- Jeżeli wstawiasz rekord i przypiszesz wartość inną niż `NULL` lub inną niż zero kolumnie `AUTO_INCREMENT`, która ma unikalny indeks, wówczas są dwa możliwe scenariusze. Pierwszy: jeśli istnieje już rekord o podanej wartości dla kolumny, nastąpi zgłoszenie błędu na skutek powielenia klucza. Drugi: rekord zostanie wstawiony, a kolumna `AUTO_INCREMENT` będzie miała wskazaną wartość. Jeżeli wskazana wartość jest większa niż kolejna liczba sekwencji, sekwencja będzie kontynuowana od następnej liczby po przypisanej wstawionemu rekordowi. Innymi słowy, można „zwiększyć” licznik przez wstawienie rekordu z liczbą sekwencji większą niż bieżąca wartość licznika.
- W przypadku niektórych silników bazy danych wartości usunięte we wcześniejszej części sekwencji można ponownie wykorzystać. Jeżeli usuniesz rekord zawierający największą wartość w kolumnie `AUTO_INCREMENT`, ta wartość zostanie ponownie

użyta podczas kolejnego generowania nowej wartości sekwencji. Po usunięciu wszystkich rekordów w tabeli wszystkie wartości będą ponownie użyte, a sekwencja ponownie rozpocznie się od wartości 1.

- Jeżeli zapytania UPDATE używasz do ustawienia kolumnie AUTO_INCREMENT wartości już istniejącej w innym rekordzie, jeśli kolumna ma zdefiniowany unikalny indeks, to nastąpi zgłoszenie błędu wynikającego z powielenia klucza. Po uaktualnieniu kolumny wartością większą niż jakakolwiek istniejąca, sekwencja będzie kontynuowana od kolejnej liczby po wstawionej do rekordu. Jeżeli uaktualnisz kolumnę, przypisując jej wartość 0, nastąpi wstawienie wartości 0 (niezależnie od tego, czy włączony jest tryb NO_AUTO_VALUE_ON_ZERO).
- Jeżeli użyjesz zapytania REPLACE do uaktualnienia rekordu na podstawie wartości kolumny AUTO_INCREMENT, wspomniana wartość nie ulegnie zmianie. Jeśli zapytanie REPLACE jest używane do uaktualnienia rekordu na podstawie wartości innej kolumny wraz ze zdefiniowanym kluczem podstawowym lub unikalnym indeksem, kolumna AUTO_INCREMENT będzie uaktualniona nową liczbą sekwencji, o ile została ustawiona jako NULL lub jeśli została ustawiona jako 0 i włączono tryb NO_AUTO_VALUE_ON_ZERO.

3.4.2. Właściwości AUTO_INCREMENT charakterystyczne dla silnika bazy danych

Ogólne cechy charakterystyczne AUTO_INCREMENT przedstawione powyżej stanowią podstawy pozwalające na zrozumienie zachowania sekwencji w poszczególnych silnikach bazy danych. Większość silników implementuje zachowanie, które w większości sytuacji pokrywa się z zaprezentowanym powyżej. Warto więc zapamiętać przedstawione tam informacje. Silnik MyISAM oferuje największą elastyczność podczas obsługi sekwencji, a więc rozpoczniemy od niego.

3.4.2.1. Atrybut AUTO_INCREMENT w tabelach MyISAM

Silnik MyISAM charakteryzuje się następującymi cechami w zakresie obsługi AUTO_INCREMENT:

- Sekwencje MyISAM są jednostajne. Wartości w automatycznie wygenerowanej serii są ściśle zwiększające się, a ponadto nie są ponownie używane po usunięciu rekordów. Jeżeli wartość maksymalna wynosi 143 i nastąpi usunięcie rekordu zawierającego tę wartość, po wstawieniu nowego rekordu MySQL wygeneruje kolejną wartość 144. Istnieją dwa wyjątki od przedstawionego zachowania:
 - ◆ Jeżeli opróżnisz tabelę zapytaniem TRUNCATE TABLE, licznik zostaje wyzerowany i ponownie rozpoczyna się od 1.
 - ◆ Wartości usunięte wcześniej w sekwencji są ponownie używane, jeśli korzystasz z indeksu złożonego do wygenerowania wielu sekwencji w tabeli. (Ta technika zostanie wkrótce omówiona).

- Sekwencje MyISAM domyślnie rozpoczynają się od 1, ale możesz wskazać wartość początkową za pomocą opcji `AUTO_INCREMENT = n` w zapytaniu `CREATE TABLE`. Poniższy przykład pokazuje utworzenie tabeli MyISAM wraz z kolumną `AUTO_INCREMENT` o nazwie `seq` zawierającą wartości, począwszy od 1 000 000:

```
CREATE TABLE mytbl
(
    seq INT UNSIGNED NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (seq)
) ENGINE=MYISAM AUTO_INCREMENT=1000000;
```

Tabela może zawierać tylko jedną kolumnę `AUTO_INCREMENT`, zawsze więc wiadomo, do której kolumny ma zastosowanie opcja `AUTO_INCREMENT = n`.

- Za pomocą zapytania `ALTER TABLE` można zmienić bieżący licznik sekwencji dla istniejącej tabeli MyISAM. Jeżeli aktualna wartość liczby w sekwencji to 1000, poniższe zapytanie powoduje, że kolejną liczbą wygenerowaną w sekwencji będzie 2000:

```
ALTER TABLE mytbl AUTO_INCREMENT=2000;
```

Jeżeli chcesz ponownie użyć wartości usuniętych we wcześniejszej części sekwencji, to oczywiście możesz to zrobić. Poniższe zapytanie powoduje ustawienie najniższej możliwej wartości licznika i powoduje, że kolejna liczba będzie o jeden większa od aktualnej maksymalnej wartości sekwencji:

```
ALTER TABLE mytbl AUTO_INCREMENT=1;
```

Nie można użyć opcji `AUTO_INCREMENT` do ustawienia bieżącemu licznikowi wartości mniejszej niż obecna maksymalna wartość w tabeli. Jeżeli kolumna `AUTO_INCREMENT` zawiera wartości 1 i 10, użycie `AUTO_INCREMENT = 5` spowoduje ustawienie licznikowi wartości 11, a nie 5.

Silnik bazy danych MyISAM obsługuje użycie indeksów złożonych (wielokolumnowych) podczas tworzenia wielu niezależnych sekwencji w tej samej tabeli. Aby użyć takiej funkcji, należy utworzyć wielokolumnowy indeks `PRIMARY KEY` lub `UNIQUE` zawierający kolumnę `AUTO_INCREMENT` jako ostatnią. Dla każdego odmiennego klucza w lewej kolumnie lub kolumnach indeksu w kolumnie `AUTO_INCREMENT` zostanie wygenerowana oddzielna sekwencja wartości. Na przykład, tabeli o nazwie `bugs` możesz używać do śledzenia zgłoszeń błędów dla kilku różnych projektów oprogramowania, o ile tabela została zdefiniowana w następujący sposób:

```
CREATE TABLE bugs
(
    proj_name VARCHAR(20) NOT NULL,
    bug_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    description VARCHAR(100),
    PRIMARY KEY (proj_name, bug_id)
) ENGINE = MYISAM;
```

W powyższej definicji kolumna `proj_name` wskazuje nazwę projektu, natomiast w kolumnie `description` znajduje się opis zgłoszonego błędu. Kolumna `bug_id` jest kolumną `AUTO_INCREMENT`, dzięki utworzeniu indeksu wiążącego ją z kolumną `proj_name`

istnieje możliwość wygenerowania niezależnych sekwencji dla poszczególnych projektów. Przyjmujemy założenie, że ponownie rekordy wstawiono do tabeli bugs i tym samym zarejestrowano trzy błędy dla projektu SuperBrowser i dwa dla SpamSquisher:

```
mysql> INSERT INTO bugs (proj_name,description)
-> VALUES('SuperBrowser','ulega awarii podczas wyświetlania skomplikowanych tabel');
mysql> INSERT INTO bugs (proj_name,description)
-> VALUES('SuperBrowser','skalowanie obrazów nie działa');
mysql> INSERT INTO bugs (proj_name,description)
-> VALUES('SpamSquisher','nie blokuje domen wskazanych na liście');
mysql> INSERT INTO bugs (proj_name,description)
-> VALUES('SpamSquisher','nie uwzględnia adresów wskazanych na liście dozwolonych');
mysql> INSERT INTO bugs (proj_name,description)
VALUES('SuperBrowser','wzorce tła nie są wyświetlane');
```

Po wstawieniu rekordów otrzymujemy następującą tabelę:

```
mysql> SELECT * FROM bugs ORDER BY proj_name, bug_id;
```

proj_name	bug_id	description
SpamSquisher	1	nie blokuje domen wskazanych na liście
SpamSquisher	2	nie uwzględnia adresów wskazanych na liście dozwolonych
SuperBrowser	1	ulega awarii podczas wyświetlania skomplikowanych tabel
SuperBrowser	2	skalowanie obrazów nie działa
SuperBrowser	3	wzorce tła nie są wyświetlane

Liczby w kolumnie bug_id są generowane oddzielnie dla każdego projektu, niezależnie od kolejności wstawiania rekordów poszczególnych projektów. Nie ma konieczności wstawiania najpierw wszystkich rekordów jednego projektu, a następnie dopiero kolejnego.

Jeżeli używasz indeksu złożonego do utworzenia wielu sekwencji, wartości usunięte we wcześniejszej części każdej sekwencji są ponownie używane. To różnica w stosunku do standardowego zachowania MyISAM, gdzie wartości nie są ponownie używane.

3.4.2.2. Atrybut AUTO_INCREMENT w tabelach InnoDB

Silnik InnoDB charakteryzuje się następującymi cechami w zakresie obsługi AUTO_INCREMENT:

- Wartość początkowa sekwencji może być ustawiona za pomocą opcji `AUTO_INCREMENT = n` w zapytaniu `CREATE TABLE`. Po utworzeniu tabeli można ją zmodyfikować za pomocą zapytania `ALTER TABLE`.
- Wartości usunięte we wcześniejszej części sekwencji normalnie nie są ponownie używane. Jednak po opróżnieniu tabeli zapytaniem `TRUNCATE TABLE` licznik sekwencji zostaje wyzerowany i ponownie rozpoczyna się od 1. Ponowne użycie liczb sekwencji zachodzi również po spełnieniu wymienionych dalej warunków. W trakcie pierwszego wygenerowania wartości sekwencji dla tabeli `AUTO_INCREMENT`, silnik InnoDB używa wartości o jeden większej niż aktualna wartość maksymalna w kolumnie (lub 1 w przypadku gdy tabela jest pusta). Silnik InnoDB zachowuje ten licznik w pamięci i używa podczas generowania kolejnych wartości, licznik nie jest przechowywany w tabeli. Oznacza to, że po

usunięciu wartości z początkowej części sekwencji i ponownym uruchomieniu serwera usunięte wartości są ponownie używane. Ponowne uruchomienie serwera niweluje także efekt użycia `AUTO_INCREMENT` w zapytaniu `CREATE TABLE` lub `ALTER TABLE`.

- Przerwy w sekwencji mogą wystąpić, gdy zostanie wycofana transakcja generująca wartości `AUTO_INCREMENT`.
- Indeksy złożone nie mogą być używane do wygenerowania wielu niezależnych sekwencji w tabeli.

3.4.2.3. Atrybut `AUTO_INCREMENT` w tabelach `MEMORY`

Silnik `MEMORY` charakteryzuje się następującymi cechami w zakresie obsługi `AUTO_INCREMENT`:

- Wartość początkowa sekwencji może być ustawiona za pomocą opcji `AUTO_INCREMENT = n` w zapytaniu `CREATE TABLE`. Po utworzeniu tabeli można ją zmodyfikować za pomocą zapytania `ALTER TABLE`.
- Wartości usunięte we wcześniejszej części sekwencji normalnie nie są ponownie używane. Jednak po opróżnieniu tabeli zapytaniem `TRUNCATE TABLE` licznik sekwencji zostaje wyzerowany i ponownie rozpoczyna się od 1.
- Indeksy złożone nie mogą być używane do wygenerowania wielu niezależnych sekwencji w tabeli.

3.4.3. Kwestie dotyczące kolumn `AUTO_INCREMENT`

Pamiętaj o przedstawionych poniżej kwestiach, aby uniknąć zaskoczenia zachowaniem kolumny `AUTO_INCREMENT`:

- Podstawowym przeznaczeniem mechanizmu `AUTO_INCREMENT` jest umożliwienie wygenerowania sekwencji dodatnich liczb całkowitych. Użycie w kolumnie `AUTO_INCREMENT` liczb innych niż dodatnie nie jest obsługiwane. Dlatego też kolumnę `AUTO_INCREMENT` można zdefiniować jako `UNSIGNED`. W przypadku kolumn liczb całkowitych użycie atrybutu `UNSIGNED` ma tę zaletę, że podwaja ilość dozwolonych liczb w sekwencji, zanim nastąpi osiągnięcie górnego zakresu wskazanego typu danych.
- Dodanie atrybutu `AUTO_INCREMENT` do definicji kolumny nie jest magicznym rozwiązaniem pozwalającym na otrzymanie nieograniczonej ilości liczb w sekwencji. Sekwencje `AUTO_INCREMENT` zawsze są ograniczone zastosowanym typem danych. Na przykład, użycie typu `TINYINT` dla kolumny oznacza, że maksymalna liczba w sekwencji to 127. Po osiągnięciu tego pułapu w aplikacji będą zgłaszane błędy informujące o powieleniu kluczy. Z kolei po użyciu `TINYINT UNSIGNED` górna granica zostaje podniesiona do 255, ale to nadal jest pewne ograniczenie.

- Całkowite opróżnienie zawartości tabeli za pomocą zapytania `TRUNCATE TABLE` może wyzerować sekwencję, która ponownie będzie rozpoczynała się od 1, nawet jeśli silnik bazy danych normalnie nie używa ponownie wartości `AUTO_INCREMENT`. Wyzerowanie sekwencji następuje ze względu na sposób, w jaki MySQL próbuje zoptymalizować operację całkowitego opróżnienia tabeli. Gdy tylko istnieje możliwość, wyrzucane są wszystkie rekordy danych i indeksy, a następnie tabela tworzona jest zupełnie od początku, zamiast przeprowadzać operacje usuwania poszczególnych rekordów. Dlatego też tracone są informacje dotyczące liczb w sekwencji. W celu usunięcia wszystkich rekordów i jednoczesnego zachowania informacji o sekwencji konieczne jest zawieszenie wspomnianej optymalizacji przez użycie zapytania `DELETE` wraz z klauzulą `WHERE` zawsze zwracającą wartość `TRUE`. W takim przypadku wymusisz na MySQL sprawdzenie warunku dla każdego rekordu, a tym samym ich kolejne usunięcie:

```
DELETE FROM nazwa_tabeli WHERE TRUE;
```

3.4.4. Wskazówki pomocne podczas pracy z kolumnami `AUTO_INCREMENT`

W tym punkcie znajdziesz użyteczne techniki, które można zastosować podczas pracy z kolumnami `AUTO_INCREMENT`.

3.4.4.1. Dodanie do tabeli kolumny liczby sekwencji

Przyjmujemy założenie, że utworzyłeś i wypełniłeś tabelę, wykonując przedstawione poniżej zapytania:

```
mysql> CREATE TABLE t (c CHAR(10));
mysql> INSERT INTO t VALUES('a'),('b'),('c');
mysql> SELECT * FROM t;
```

c
a
b
c

Następnie postanowiłeś, że w tabeli chcesz umieścić kolumnę przechowującą liczbę sekwencji. W tym celu należy wykonać zapytanie `ALTER TABLE` i dodać kolumnę wraz z atrybutem `AUTO_INCREMENT`, używając tego samego rodzaju definicji jak w przypadku zapytania `CREATE TABLE`. Kolumnie `AUTO_INCREMENT` MySQL automatycznie przypisuje wartości sekwencji:

```
mysql> ALTER TABLE t ADD i INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY;
mysql> SELECT * FROM t;
```

c	i
a	1

c	i
b	2
c	3

3.4.4.2. Ponowne utworzenie sekwencji w istniejącej kolumnie

Jeżeli tabela ma już kolumnę `AUTO_INCREMENT`, ale chcesz ją ponownie ponumerować w celu eliminacji przerw w sekwencji powstałych na skutek usunięcia rekordów, najłatwiejszym rozwiązaniem jest usunięcie kolumny, a następnie jej ponowne dodanie. Kiedy MySQL dodaje kolumnę, automatycznie przypisuje jej nowe liczby sekwencji.

Przyjmujemy założenie, że tabela `t` została utworzona w poniższy sposób, natomiast `i` to kolumna zdefiniowana wraz z atrybutem `AUTO_INCREMENT`:

```
mysql> CREATE TABLE t (c CHAR(10), i INT UNSIGNED AUTO_INCREMENT
-> NOT NULL PRIMARY KEY);
mysql> INSERT INTO t (c)
-> VALUES('a'),('b'),('c'),('d'),('e'),('f'),('g'),('h'),('i'),('j'),
('k');
mysql> DELETE FROM t WHERE c IN('a','d','f','g','j');
mysql> SELECT * FROM t;
```

c	i
b	2
c	3
e	5
h	8
i	9
k	11

Poniższe zapytanie `ALTER TABLE` spowoduje usunięcie kolumny `i` i jej ponowne dodanie, co skutkuje ponownym przypisaniem liczb sekwencji:

```
mysql> ALTER TABLE t
-> DROP PRIMARY KEY,
-> DROP i,
-> ADD i INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
-> AUTO_INCREMENT=1;
mysql> SELECT * FROM t;
```

c	i
b	1
c	2
e	3
h	4
i	5
k	6

Klauzula `AUTO_INCREMENT = 1` powoduje wyzerowanie sekwencji, aby ponownie rozpoczynała się od wartości 1. W przypadku tabel `MyISAM`, `InnoDB` i `MEMORY` można użyć wartości innej niż 1, rozpoczynając tym samym sekwencję od wskazanej liczby.

W innych silnikach bazy danych można pominąć klauzulę `AUTO_INCREMENT`, ponieważ nie zezwalają one na określenie wartości początkowej sekwencji w taki właśnie sposób. Sekwencja rozpocznie się wówczas od wartości 1.

Wprawdzie ponowne przypisanie liczb sekwencji jest łatwe, a pytanie „Jak można to zrobić?” pojawia się często, jednak potrzeba zmiany sekwencji rzadko się zdarza. Dla MySQL przerwy w sekwencji nie mają znaczenia, podobnie jak wszelka zmiana wydajności na skutek ponownego przypisania liczb sekwencji. Ponadto, jeśli w innej tabeli znajdują się rekordy odwołujące się do wartości w kolumnie `AUTO_INCREMENT`, wówczas ponowne przypisanie liczb sekwencji powoduje usunięcie relacji między tabelami.

3.4.5. Generowanie sekwencji bez `AUTO_INCREMENT`

MySQL zapewnia metodę generowania liczb sekwencji zupełnie niewykorzystującą kolumny `AUTO_INCREMENT`. Zamiast tego używana jest alternatywna forma w postaci funkcji `LAST_INSERT_ID()` pobierającej argument. Jeżeli wstawisz lub uaktualnisz kolumnę za pomocą `LAST_INSERT_ID(wyrażenie)`, wtedy kolejne wywołanie funkcji `LAST_INSERT_ID()` bez argumentu zwraca wartość *wyrażenia*. Innymi słowy, MySQL traktuje *wyrażenie*, jakby zostało wygenerowane jako wartość `AUTO_INCREMENT`. W ten sposób zyskujesz możliwość tworzenia liczb sekwencji, a następnie pobierania ich w dalszej części sesji bez obaw, że wygenerowana wartość została zmodyfikowana na skutek działalności innych klientów.

Jednym ze sposobów wykorzystania tej strategii jest utworzenie tabeli składającej się z pojedynczego rekordu i zawierającej wartość uaktualnianą za każdym razem, gdy chcesz otrzymać kolejną wartość sekwencji. Na przykład, tabelę możesz utworzyć i zainicjalizować w poniższy sposób:

```
CREATE TABLE seq_table (seq INT UNSIGNED NOT NULL);
INSERT INTO seq_table VALUES(0);
```

Powyższe zapytania tworzą tabelę `seq_table` z pojedynczym rekordem zawierającym wartość `seq` wynoszącą 0. Aby wygenerować kolejną liczbę w sekwencji oraz ją pobrać, należy wykonać poniższe zapytania:

```
UPDATE seq_table SET seq = LAST_INSERT_ID(seq+1);
SELECT LAST_INSERT_ID();
```

Zapytanie `UPDATE` pobiera wartość bieżącą kolumny `seq`, a następnie zwiększa ją o 1 w celu wygenerowania kolejnej wartości w sekwencji. Wygenerowanie nowej wartości za pomocą funkcji `LAST_INSERT_ID(seq+1)` powoduje traktowanie tej wartości jak `AUTO_INCREMENT`. Dlatego też może być później pobierana za pomocą wywołania `LAST_INSERT_ID()` bez argumentów. Funkcja `LAST_INSERT_ID()` jest charakterystyczna dla klienta, a więc otrzymasz prawidłową wartość, nawet jeśli inne klienty generują liczby sekwencji pomiędzy wykonaniem zapytań `UPDATE` i `SELECT`.

Przedstawiona metoda może wygenerować wartości sekwencji zwiększane o wartość inną niż jeden, a także ujemne. Na przykład, poniższe zapytanie może być nieustannie wywoływane w celu wygenerowania kolejnych liczb sekwencji o wartości zwiększanej za każdym razem o 100:

```
UPDATE seq_table SET seq = LAST_INSERT_ID(seq+100);
```

Kolejne wywołanie poniższego zapytania powoduje generowanie coraz mniejszych liczb sekwencji:

```
UPDATE seq_table SET seq = LAST_INSERT_ID(seq-1);
```

Istnieje również możliwość wygenerowania sekwencji rozpoczynającej się od dowolnej wartości; wystarczy po prostu przypisać kolumnie seq odpowiednią wartość początkową.

Powyżej przedstawiono sposób konfiguracji licznika za pomocą tabeli zawierającej pojedynczy rekord. Jeżeli chcesz użyć wielu liczników, do tabeli dodaj kolejną kolumnę służącą jako identyfikator licznika, a następnie nowy rekord dla każdego licznika. Przyjmujemy założenie, że masz witrynę internetową i chcesz zaimplementować licznik odwiedzin stron w stylu „Ta strona została wyświetlona *n* razy”. Utwórz tabelę z dwiema kolumnami. Jedna kolumna będzie przechowywać nazwę unikalnie identyfikującą każdy licznik. Z kolei druga kolumna przechowuje wartość bieżącą licznika. Nadal możesz skorzystać z funkcji `LAST_INSERT_ID()`, ale konieczne jest ustalenie właściwego rekordu na podstawie nazwy licznika. Na przykład, wspomnianą tabelę można utworzyć za pomocą poniższego zapytania:

```
CREATE TABLE counter
(
  name VARCHAR(255) CHARACTER SET latin1 COLLATE latin1_general_cs NOT NULL,
  value INT UNSIGNED,
  PRIMARY KEY (name)
);
```

Kolumna `name` jest typu ciągu tekstowego, można jej nadać dowolną nazwę. Została również zdefiniowana jako `PRIMARY KEY`, aby uniemożliwić powielanie nazw. Przyjęto założenie, że aplikacja używająca tabeli zastosuje zdefiniowane w niej nazwy. W przypadku licznika sieciowego unikalność nazwy licznika jest gwarantowana przez zastosowanie nazwy licznika w postaci po prostu ścieżki dostępu do każdej strony w drzewie dokumentów. Kolejność sortowania w kolumnie `name` rozróżnia wielkość liter, aby wartości w postaci ścieżek dostępu również były traktowane jako rozróżniające wielkość liter. Jeżeli w używanym systemie operacyjnym wielkość liter w nazwach ścieżek nie ma znaczenia, wtedy możesz zastosować kolejność sortowania nierozróżniającą wielkości liter.

Aby użyć tabeli `counter`, użyteczne będzie zapytanie `INSERT ... ON DUPLICATE KEY UPDATE`, ponieważ pozwala na wstawienie nowego rekordu dla strony, która nie zawiera jeszcze licznika, oraz uaktualnienie licznika dla istniejącej strony. Ponadto, dzięki użyciu funkcji `LAST_INSERT_ID(wyrażenie)` do wygenerowania wartości licznika wartość licznika możesz bardzo łatwo pobrać po jego uaktualnieniu. Na przykład, w celu zainicjalizowania lub inkrementacji licznika dla strony głównej witryny, a następnie wyświetlenia wartości bieżącej licznika użyj poniższego zapytania:

```
INSERT INTO counter (name, value)
VALUES('index.html', LAST_INSERT_ID(1))
ON DUPLICATE KEY UPDATE value = LAST_INSERT_ID(value+1);
SELECT LAST_INSERT_ID();
```

Alternatywne rozwiązanie polega na inkrementacji liczników istniejących stron bez użycia do tego celu funkcji `LAST_INSERT_ID()`:

```
UPDATE counter SET value = value+1 WHERE name = 'index.html';
SELECT value FROM counter WHERE name = 'index.html';
```

Jednak takie rozwiązanie nie będzie działało prawidłowo, jeśli inny klient przeprowadzi inkrementację licznika po wykonaniu zapytania `UPDATE`, ale jeszcze przed wykonaniem zapytania `SELECT`. Rozwiązaniem tego problemu może być zastosowanie zapytań `LOCK TABLES` i `UNLOCK TABLES` wokół dwóch powyższych zapytań. Ewentualnie, tabelę można utworzyć z użyciem transakcyjnego silnika bazy danych, a następnie operację uaktualnienia tabeli przeprowadzić w transakcji. Wymienione powyżej metody blokują jednak innych klientów podczas używania licznika, ale oparta na funkcji `LAST_INSERT_ID()` pozwala na znacznie łatwiejsze wykonanie tego samego zadania. Ponieważ jej wartość jest charakterystyczna dla klienta, to zawsze otrzymasz wstawioną wartość, a nie pochodzącą od innego klienta. Ponadto, nie ma potrzeby komplikowania kodu przez nakładanie blokad lub stosowanie transakcji, aby nie mieszać wartości poszczególnych klientów.

3.5. Obliczanie wyrażeń i konwersja typu

Wyrażenia zawierają komponenty i operatory oraz są obliczane w celu wygenerowania wyniku. Komponentami mogą być wartości takie jak stałe, wywołania funkcji, odniesienia do kolumn tabel, a także skalarne podzapytania. Wymienione wartości można łączyć za pomocą różnego rodzaju operatorów, na przykład arytmetycznych lub porównania, a komponenty wyrażenia mogą być grupowane przy użyciu nawiasów. Wyrażenia najczęściej występują na liście kolumn w danych wyjściowych oraz w klauzulach `WHERE` zapytań `SELECT`. Na przykład, poniżej przedstawiono zapytanie podobne do użytego w celu obliczenia wieku w podpunkcie 1.4.9.6, zatytułowanym „Praca z datami”:

```
SELECT
    CONCAT(last_name, ' ', first_name),
    TIMESTAMPDIFF(YEAR, birth, death)
FROM president
WHERE
    birth > '1900-1-1' AND DEATH IS NOT NULL;
```

Każda wybrana wartość przedstawia wyrażenie, podobnie jak zawartość klauzuli `WHERE`. Wyrażenia występują również w klauzulach `WHERE` zapytań `DELETE` i `UPDATE`, klauzulach `VALUES()` zapytań `INSERT` itd.

Kiedy MySQL napotyka wyrażenie, oblicza je w celu wygenerowania wyniku. Na przykład, wartością wyrażenia $(4 \cdot 3) \text{ DIV } (4 - 2)$ jest 6. W trakcie obliczania wartości wyrażenia może wystąpić konieczność przeprowadzenia konwersji typu, na przykład podczas konwersji przez MySQL liczby 960821 na datę '1996-08-21', o ile liczba została użyta w kontekście wymagającym daty.

W tym podrozdziale dowiesz się, jak tworzyć wyrażenia w MySQL oraz jakie reguły rządzą operacjami konwersji typów, które mogą wystąpić w trakcie obliczania wyrażeń.

Wszystkie operatory zostały tutaj wymienione, ale ponieważ MySQL oferuje dużą liczbę funkcji, tylko kilka z nich zajęliśmy się dokładniej. Więcej informacji na ten temat znajdziesz w dodatku C, zatytułowanym „Przewodnik po operatorach i funkcjach”.

3.5.1. Tworzenie wyrażeń

Wyrażenie może być proste i składać się z pojedynczej stałej, takiej jak wartość liczbową 0 lub ciąg tekstowy 'abc'.

Wyrażenia mogą używać wywołań funkcji. Pewne funkcje pobierają argumenty (wartości w nawiasach), podczas gdy inne nie. W przypadku akceptowania przez funkcję wielu argumentów powinny być one rozdzielone przecinkami. Kiedy wywołujesz funkcję wbudowaną, wokół argumentów mogą znajdować się spacje. Jednak jeśli spacja znajdzie się między nazwą funkcji i nawiasem otwierającym, analizator MySQL może nieprawidłowo zinterpretować nazwę funkcji, co zwykle prowadzi do zgłoszenia błędu składni. Włączenie trybu SQL o nazwie IGNORE_SPACE powoduje, że dozwolone jest użycie spacji po nazwach funkcji wbudowanych, ale jednocześnie nazwy funkcji są traktowane jako słowa zarezerwowane.

Wyrażenia mogą zawierać odniesienia do kolumn tabeli. W najprostszym przypadku, gdy tabela, do której należy kolumna, jest wolna od kontekstu, odniesienie do kolumny może składać się po prostu z samej nazwy kolumny. W poniższych zapytaniach SELECT wymieniona jest tylko jedna kolumna, a więc nazwy kolumn są jednoznaczne pomimo tego, że w obu zapytaniach są używane te same:

```
SELECT last_name, first_name FROM president;  
SELECT last_name, first_name FROM member;
```

Gdy nie jest całkowicie jasne, która tabela powinna zostać użyta, trzeba stosować kwalifikowane nazwy kolumn przez poprzedzenie ich prawidłowymi nazwami tabel. Jeżeli nie jest jasne, która baza danych powinna zostać użyta, nazwę tabeli należy poprzedzić kwalifikowaną nazwą bazy danych. Istnieje również możliwość użycia dokładnych kwalifikowanych form w niewymagających tego sytuacjach, aby na przykład wyraźnie podać wszystkie informacje:

```
SELECT  
    president.last_name, president.first_name,  
    member.last_name, member.first_name  
FROM president INNER JOIN member  
WHERE president.last_name = member.last_name;  
  
SELECT sampdb.student.name FROM sampdb.student;
```

Więcej informacji szczegółowych na temat kwalifikatorów znajdziesz w podrozdziale 2.2, zatytułowanym „Identyfikatory składni MySQL i reguły nadawania nazw”.

Podzapytania skalarne mogą być używane w celu dostarczenia pojedynczej wartości w wyrażeniu. Podzapytanie musi być ujęte w nawias:

```
SELECT * FROM president WHERE birth = (SELECT MAX(birth) FROM president);
```

Wreszcie, wszystkie wymienione rodzaje wartości (stałe, wywołania funkcji, odniesienia do kolumn i podzapytania) można ze sobą łączyć, tworząc w ten sposób bardziej skomplikowane wyrażenia.

3.5.1.1. Typy operatorów

Komponenty wyrażen mogą być łączone za pomocą wielu rodzajów operatorów. W tym podpunkcie przedstawiono działanie operatorów, natomiast w podpunkcie 3.5.1.2, zatytułowanym „Kolejność operatorów”, omówimy kolejność, w jakiej są one obliczane.

Wymienione w tabeli 3.16 operatory arytmetyczne zawierają standardowe operatory dodawania, odejmowania, mnożenia i dzielenia, a także operator modulo. Dla operatorów +, - i * działania arytmetyczne są przeprowadzane za pomocą liczb całkowitych typu BIGINT (64-bitowych), o ile oba operandy są liczbami całkowitymi. W takim przypadku wynik będzie bez znaku, jeśli operandy również nie mają zdefiniowanego znaku. Dla każdego operandu innego niż DIV, jeśli operand ma wartość przybliżoną, operacje arytmetyczne są przeprowadzane na liczbach zmiennoprzecinkowych o podwójnej precyzji. Dotyczy to również ciągów tekstowych skonwertowanych na liczby, ponieważ ciągi tekstowe są konwertowane na liczby o podwójnej precyzji. Musisz mieć świadomość, że jeśli operacja na liczbach całkowitych będzie wymagała użycia ogromnych wartości i wynik wykroczy poza 64-bitowy zakres, wtedy otrzymasz zupełnie nieprzewidywalne wyniki. (W rzeczywistości powinienś spróbować unikać przekraczania wartości 63-bitowych, ponieważ jeden bit jest konieczny do określenia znaku).

Tabela 3.16. Operatory arytmetyczne

Operator	Składnia	Opis
+	a + b	Dodawanie; suma operandów
-	a - b	Odejmowanie; różnica operandów
-	-a	Minus jednoargumentowy; negacja operandu
*	a * b	Mnożenie; połączenie operandów
/	a / b	Dzielenie; współczynnik operandów
DIV	a DIV b	Dzielenie; wyrażony w postaci liczby całkowitej współczynnik operandów
%	a % b	Modulo; reszta z dzielenia

Wymienione w tabeli 3.17 operatory logiczne obliczają wyrażenie w celu określenia, czy wynikiem jest prawda (wartość niezerowa), czy fałsz (zero). Istnieje możliwość, że wynikiem wyrażenia będzie NULL, jeśli jego wartości nie będzie można ustalić. Na przykład, wartości wyrażenia 1 AND NULL nie można ustalić.

Tabela 3.17. Operatory logiczne

Operator	Składnia	Opis
AND, &&	a AND b, a && b	Iloczyn logiczny; true, jeśli oba operandy zwracają true
OR,	a OR b, a b	Suma zbiorów; true, jeśli którykolwiek operand zwraca true
XOR	a XOR b	Wykluczające się OR; true, jeśli tylko jeden operand zwraca true
NOT, !	NOT a, !a	Logiczna negacja; true, jeśli operand zwraca false

Jako formę alternatywną dla operatorów AND, OR i NOT MySQL obsługuje również operatory odpowiednio &&, | | i !, ponieważ są używane w języku programowania C. W szczególności zwróć uwagę na operator | |. Standard SQL wskazuje | | jako operator łączenia ciągów tekstowych, ale w MySQL oznacza on logiczną operację OR. Jeżeli użyjesz poniższego wyrażenia, oczekując przeprowadzenia operacji łączenia ciągów tekstowych, możesz być zaskoczony zwróconą wartością w postaci liczby 0:

```
'abc' | | 'def' → 0
```

Dzieje się tak, ponieważ ciągi tekstowe 'abc' i 'def' zostają w trakcie operacji skonwertowane na postać liczb całkowitych i oba przyjmują wartość 0. W celu połączenia ciągów tekstowych w MySQL należy użyć funkcji CONCAT('abc', 'def') lub umieścić je obok siebie:

```
CONCAT('abc', 'def') → 'abcdef'
'abc' 'def' → 'abcdef'
```

Aby włączyć standardowe zachowanie SQL dla operatora | |, należy włączyć tryb SQL o nazwie PIPES_AS_CONCAT.

Wymienione w tabeli 3.18 operatory bitowe przeprowadzają operacje bitowe: iloczyn logiczny, suma zbiorów i wykluczające się OR, w których każdy bit wyniku jest ustalany na podstawie logicznej operacji AND, OR i XOR odpowiednich bitów operandów. Istnieje również możliwość przesunięcia bitów w lewo lub w prawo. Operacje bitowe są przeprowadzane za pomocą liczb całkowitych typu BIGINT (64-bitowe).

Wymienione w tabeli 3.19 operatory porównania obejmują operatory przeznaczone do testowania względnej wielkości oraz leksykalnej kolejności liczb i ciągów tekstowych. Obejmują także operatory odpowiedzialne za przeprowadzanie operacji dopasowania wzorca i sprawdzania względem wartości NULL. Operator <=> jest charakterystyczny dla MySQL. Szczegółowe omówienie właściwości porównywania ciągów tekstowych znajdziesz w punkcie 3.1.2, zatytułowanym „Wartości ciągu tekstowego”.

Tabela 3.18. Operatory bitowe

Operator	Składnia	Opis
&	a & b	Bitowe AND (iloczyn logiczny): każdy bit wyniku jest ustawiony, jeśli odpowiadające sobie bity w obu operandach są ustawione.
	a b	Bitowe OR (suma zbiorów): każdy bit wyniku jest ustawiony, jeśli odpowiadający mu bit w którymkolwiek operandzie został ustawiony.
^	a ^ b	Bitowe XOR: każdy bit wyniku jest ustawiony, jeśli ustawiony jest tylko jeden odpowiadający mu bit w operandach.
<<	a << b	Przesunięcie a w lewo o b bitów.
>>	a >> b	Przesunięcie a w prawo o b bitów.

Tabela 3.19. Operatory porównania

Operator	Składnia	Opis
=	a = b	Prawda, jeśli operandy są równe.
<=>	a <=> b	Prawda, jeśli operandy są równe (nawet jeśli NULL).
<>, !=	a <> b, a != b	Prawda, jeśli operandy nie są równe.
<	a < b	Prawda, jeśli a jest mniejsze niż b.
<=	a <= b	Prawda, jeśli a jest mniejsze niż lub równe b.
>=	a >= b	Prawda, jeśli a jest większe niż lub równe b.
>	a > b	Prawda, jeśli a jest większe niż b.
IN	a IN (b1, b2, ...)	Prawda, jeśli a jest równe dowolnemu b1, b2, ...
BETWEEN	a BETWEEN b AND c	Prawda, jeśli b <= a <= c
NOT BETWEEN	a NOT BETWEEN b AND c	Prawda, o ile nie b <= a <= c
LIKE	a LIKE b	Dopasowanie wzorca SQL: prawda, jeśli a jest dopasowane do b.
NOT LIKE	a NOT LIKE b	Dopasowanie wzorca SQL: prawda, jeśli a nie jest dopasowane do b.
REGEXP	a REGEXP b	Dopasowanie wyrażenia regularnego: prawda, jeśli a jest dopasowane do b.
NOT REGEXP	a NOT REGEXP b	Dopasowanie wyrażenia regularnego: prawda, jeśli a nie jest dopasowane do b.
IS NULL	a IS NULL	Prawda, jeśli wartość operandu wynosi NULL.
IS NOT NULL	a IS NOT NULL	Prawda, jeśli wartość operandu nie wynosi NULL.

Dopasowanie wzorca umożliwia wyszukiwanie wartości bez konieczności jej dokładnego podania. MySQL oferuje dopasowanie wzorca SQL za pomocą operatora LIKE i znaków wieloznacznych % (dopasowanie dowolnej sekwencji znaków) oraz _ (dopasowanie dowolnego pojedynczego znaku). Ponadto, MySQL oferuje dopasowanie wzorca za pomocą operatora REGEXP i wyrażeń regularnych podobnych do stosowanych w programach systemu UNIX, na przykład grep, sed i vi. W celu przeprowadzenia dopasowania wzorca musisz użyć jednego z wymienionych operatorów; nie można stosować operatora =. Aby odwrócić sens dopasowania wzorca, użyj NOT LIKE lub NOT REGEXP.

Poza jedynie użyciem innych operatorów i znaków wzorca dwa dostępne rodzaje dopasowania wzorca różnią się pod względem ważnych aspektów:

- Dopasowanie wzorca SQL w postaci operatora LIKE powoduje dopasowanie jedynie całego ciągu tekstowego. Z kolei dopasowanie wyrażenia regularnego REGEXP stwierdza dopasowanie po znalezieniu szukanego wyrażenia w dowolnym miejscu ciągu tekstowego.
- Klauzula LIKE jest bezpieczna w przypadku znaków zapisywanych za pomocą wielu bajtów. Z kolei REGEXP działa prawidłowo jedynie dla kodowań znaków, w których poszczególne znaki są opisywane tylko jednym bajtem. Ponadto, nie bierze pod uwagę kolejności sortowania.

Wzorce używane wraz z operatorem LIKE mogą zawierać znaki wieloznaczne % i _ . Na przykład, wzorzec 'Frank%' powoduje dopasowanie dowolnego ciągu tekstowego rozpoczynającego się od znaków 'Frank':

```
'Franklin' LIKE 'Frank%'      → 1
'Frankfurter' LIKE 'Frank%'    → 1
```

Znak wieloznaczny % powoduje dopasowanie dowolnej sekwencji znaków, także pustej sekwencji, a więc 'Frank%' dopasowuje 'Frank':

```
'Frank' LIKE 'Frank%'        → 1
```

Wzorzec % dopasowuje także dowolny ciąg tekstowy, również pusty. Jednak % nie dopasowuje wartości NULL. W rzeczywistości, jeśli jeden z operandów ma wartość NULL, dopasowanie wzorca zakończy się niepowodzeniem:

```
'Frank' LIKE NULL            → NULL
NULL LIKE '%'                → NULL
```

Oferowany przez MySQL operator LIKE porównuje operandy jako binarne ciągi tekstowe, jeśli jeden z nich będzie binarnym ciągiem tekstowym. Jeśli operandy są niebinarnymi ciągami tekstowymi, operator LIKE porównuje je, uwzględniając ich kolejność sortowania:

```
'Frankly' LIKE 'Frank%'      → 1
'frankly' LIKE 'Frank%'      → 1
BINARY 'Frankly' LIKE 'Frank%' → 1
BINARY 'frankly' LIKE 'Frank%' → 0
'Frankly' COLLATE latin1_general_cs LIKE 'Frank%' → 1
'frankly' COLLATE latin1_general_cs LIKE 'Frank%' → 0
```

To zachowanie różni się od standardowego operatora SQL LIKE, który rozróżnia wielkość liter.

Inny znak wieloznaczny możliwy do użycia wraz z operatorem LIKE to `_`, który powoduje dopasowanie dowolnego pojedynczego znaku. Wzorec `'__'` powoduje dopasowanie dowolnego ciągu tekstowego składającego się z dokładnie trzech znaków. Z kolei `'c_t'` dopasowuje `'cat'`, `'cot'`, `'cut'`, a nawet `'c_t'` (ponieważ `'_'` dopasowuje również sam znak podkreślenia).

Znaki wieloznaczne mogą być użyte w dowolnym miejscu wzorca. Wzorec `'%bert'` powoduje dopasowanie `'Englebert'`, `'Bert'` i `'Albert'`. Z kolei wzorec `'%bert%'` dopasowuje wszystkie wymienione wcześniej ciągi tekstowe, a także `'Berthold'`, `'Bertram'` i `'Alberta'`. Natomiast wzorec `'b%t'` dopasowuje `'Bert'`, `'bent'` i `'burnt'`.

Aby dopasować dosłowne wystąpienie znaku `%` lub `_`, należy wyłączyć jego znaczenie specjalne przez poprzedzenie znaku ukośnikiem (`\%` lub `_`):

```
'abc' LIKE 'a%c'           → 1
'abc' LIKE 'a\%c'         → 0
'a%c' LIKE 'a\%c'         → 1
'abc' LIKE 'a_c'          → 1
'abc' LIKE 'a\_c'         → 0
'a_c' LIKE 'a\_c'         → 1
```

Inne formy dopasowania wzorca w MySQL opierają się na wyrażeniach regularnych. W takim przypadku zamiast operatora LIKE stosowany jest REGEXP. Przedstawione poniżej przykłady pokazują ogólne cechy charakterystyczne wzorca wyrażen regularnych.

Kropka jest znakiem wieloznacznym i dopasowuje dowolny pojedynczy znak:

```
'abc' REGEXP 'a.c'        → 1
```

Konstrukcja `[...]` powoduje dopasowanie dowolnych znaków wymienionych w nawiasie kwadratowym:

```
'e' REGEXP '[aeiou]'      → 1
'f' REGEXP '[aeiou]'      → 0
```

Aby wskazać zakres znaków, należy podać pierwszy i ostatni znak zakresu, rozdzielając je myślnikiem. W celu zanegowania sensu klasy (czyli dopasowania niewymienionych znaków) trzeba podać `^` jako pierwszy znak klasy:

```
'abc' REGEXP '[a-z]'      → 1
'abc' REGEXP '[^a-z]'     → 0
```

Gwiazdka oznacza „dopasuj dowolną liczbę poprzednio wymienionego znaku”, a więc wzorec `'x*'` powoduje dopasowanie dowolnej liczby wystąpień znaku `x`:

```
'abcdef' REGEXP 'a.*f'    → 1
'abc' REGEXP '[0-9]*abc'  → 1
'abc' REGEXP '[0-9][0-9]*' → 0
```

Wspomniana wcześniej „dowolna liczba” obejmuje także zero wystąpień i dlatego wykonanie drugiego z powyższych wyrażen jest możliwe. Aby dopasować jedno lub więcej wystąpień poprzedniego znaku, należy zamiast gwiazdki użyć znaku plus:

'abc' REGEXP 'cd*'	→ 1
'abc' REGEXP 'cd+'	→ 0
'abcd' REGEXP 'cd+'	→ 1

Wzorce '^wzorec' i 'wzorec\$' powodują dopasowanie tylko wtedy, gdy podany wzorec występuje odpowiednio na początku lub końcu ciągu tekstowego. Natomiast wzorec '^wzorec\$' będzie dopasowany tylko wtedy, gdy wzorec jest całym ciągiem tekstowym:

'abc' REGEXP 'b'	→ 1
'abc' REGEXP '^b'	→ 0
'abc' REGEXP 'b\$'	→ 0
'abc' REGEXP '^abc\$'	→ 1
'abcd' REGEXP '^abc\$'	→ 0

Oferowane przez MySQL dopasowanie wzorca wyrażenia regularnego obsługuje także inne specjalne elementy wzorca. Więcej informacji na ten temat znajdziesz w dodatku C, zatytułowanym „Przewodnik po operatorach i funkcjach”.

Wzorce takie jak LIKE i REGEXP mogą być pobierane z kolumn tabeli, choć jeśli kolumna zawiera wiele różnych wartości, zapytanie będzie wykonywane wolniej niż przy użyciu stałej wzorca. Wzorec musi być przeanalizowany i skonwertowany na wewnętrzną formę po każdej zmianie wartości kolumny.

3.5.1.2. Kolejność operatorów

Kiedy MySQL oblicza wyrażenie, operatory określają kolejność, w której grupowane są komponenty wyrażenia. Pewne operatory mają większe pierwszeństwo, to znaczy są „silniejsze” niż inne pod tym względem, że są obliczane przed innymi. Na przykład, mnożenie i dzielenie ma większe pierwszeństwo niż dodawanie i odejmowanie. Poniższe zapytania mają taki sam efekt, ponieważ operatory * i DIV są obliczane przed operatorami + i -:

3 + 4 * 2 - 10 DIV 2	→ 6
3 + 8 - 5	→ 6

Poniższa lista pokazuje pierwszeństwo operatorów, od najwyższego do najniższego. Operatory wymienione w tym samym wierszu mają takie samo pierwszeństwo. Obliczenie operatorów o wyższym pierwszeństwie w wyrażeniu następuje przed obliczeniem operatorów o niższym pierwszeństwie. Z kolei obliczanie operatorów o takim samym pierwszeństwie następuje w wyrażeniu od lewej do prawej strony, za wyjątkiem przypisania obliczanego od prawej do lewej strony:

```

INTERVAL
BINARY COLLATE
!
- (minus jednoargumentowy) ~ (jednoargumentowa negacja bitowa)
^
* / DIV % MOD
+ -
<< >>
&

```

```

|
< <= = <=> <> != >= > IN IS LIKE REGEXP RLIKE
BETWEEN CASE WHEN THEN ELSE
NOT
AND &&
XOR
OR ||
:=

```

Pewne operatory mają różne pierwszeństwo w zależności od trybu SQL lub wersji serwera MySQL. Więcej informacji na ten temat znajdziesz w dodatku C, zatytułowanym „Przewodnik po operatorach i funkcjach”.

Aby zmienić kolejność operatorów, a tym samym obliczeń komponentów wyrażenia, należy zastosować nawiasy i zgrupować komponenty:

```

1 + 2 * 3 - 4 / 5           → 6.2000
(1 + 2) * (3 - 4) / 5       → -0.6000

```

3.5.1.3. Wartość NULL w wyrażeniach

Zachowaj ostrożność podczas używania wartości NULL w wyrażeniach, ponieważ wynik nie zawsze będzie taki, jakiego oczekujesz. Poniższe wskazówki pomogą w uniknięciu zaskoczenia.

Użycie wartości NULL jako operandu w dowolnym operatorze arytmetycznym lub bitowym powoduje, że wynikiem również jest NULL:

```

1 + NULL           → NULL
1 | NULL           → NULL

```

W przypadku operatorów logicznych wynikiem jest NULL, o ile wyniku nie będzie można obliczyć z całą pewnością:

```

1 AND NULL         → NULL
1 OR NULL          → 1
0 AND NULL         → 0
0 OR NULL          → NULL

```

Wartość NULL użyta jako operand w dowolnym operatorze porównania lub dopasowania wzorca powoduje, że wynikiem również jest NULL. Wyjątkiem są operatory `<=>`, `IS NULL` i `IS NOT NULL`, które z założenia są przeznaczone do pracy z wartościami NULL:

```

1 = NULL           → NULL
NULL = NULL        → NULL
1 <=> NULL          → 0
NULL LIKE '% '     → NULL
NULL REGEXP '.*'   → NULL
NULL <=> NULL       → 1
1 IS NULL          → 0
NULL IS NULL       → 1

```

Funkcje ogólnie zwracają wartość NULL po otrzymaniu argumentów NULL, za wyjątkiem funkcji przeznaczonych do pracy z argumentami NULL. Na przykład, funkcja `IFNULL()` ma możliwość obsługi wartości NULL i zwraca odpowiednio `true` lub `false`. Z drugiej strony,

funkcja `STRCMP()` oczekuje argumentów o wartościach innych niż `NULL`. Przekazanie jej argumentu `NULL` powoduje, że wartością zwrótną będzie `NULL` zamiast `true` lub `false`.

W operacjach sortowania wartości `NULL` są uwzględniane. Pojawiają się jako pierwsze podczas sortowania w kolejności rosnącej i jako ostatnie w sortowaniu w kolejności malejącej.

3.5.2. Konwersja typu

Kiedy wartość jednego typu pojawia się w kontekście wymagającym wartości innego typu, MySQL automatycznie przeprowadza odpowiedni rodzaj operacji konwersji typu. Wspomniana konwersja może nastąpić z dowolnego z wymienionych poniżej powodów:

- Konwersja operandu na typ odpowiedni dla operatora.
- Konwersja argumentu funkcji na typ oczekiwany przez funkcję.
- Konwersja wartości w celu jej przechowywania w innego typu kolumnie tabeli.

Istnieje możliwość wyraźnego przeprowadzenia konwersji typu za pomocą operatora rzutowania lub funkcji.

Liczby są konwertowane na ciągi tekstowe lub wartości daty i godziny stosujące kodowanie znaków i kolejność sortowania wskazane przez zmienne systemowe `character_set_connection` i `collation_connection`. Wynikiem jest niebinarny ciąg tekstowy, o ile zmienna `character_set_connection` nie ma przypisanej wartości binary, ponieważ wtedy otrzymujemy binarny ciąg tekstowy. (Przed wersją MySQL 5.5.3 konwersja liczby na ciąg tekstowy zawsze kończyła się utworzeniem binarnego ciągu tekstowego).

Poniższe zapytanie przeprowadza niejawną konwersję typu. Składa się z operatora dodawania (+) oraz dwóch operandów, 1 i '2':

```
1 + '2'                                → 3
```

Operandy są różnych typów (liczba i ciąg tekstowy), więc MySQL skonwertuje jeden z nich, aby oba były tego samego typu. Powstaje pytanie: który operand powinien być skonwertowany? W omawianym przypadku + jest operatorem liczbowym, MySQL oczekuje, że oba operandy będą liczbami i dlatego skonwertuje ciąg tekstowy '2' na liczbę 2. Następnie obliczy wyrażenie i wygeneruje wynik 3.

Poniżej przedstawiono inny przykład. Funkcja `CONCAT()` łączy ciągi tekstowe w celu utworzenia jednego, dłuższego. Funkcja interpretuje więc argumenty jako ciągi tekstowe niezależnie od ich typu. Jeżeli funkcji przekazesz argumenty w postaci liczb, `CONCAT()` skonwertuje je na ciągi tekstowe, a następnie zwróci wynik ich połączenia:

```
CONCAT(1,23,456)                      → '123456'
```

Jeżeli wywołanie `CONCAT()` jest częścią większego wyrażenia, może dojść do kolejnej operacji konwersji typu. Przeanalizuj poniższe wyrażenie i wynik jego działania:

```
REPEAT('X',CONCAT(1,2,3)/10)           → 'XXXXXXXXXXXX'
```

Wywołanie `CONCAT(1,2,3)` powoduje wygenerowanie ciągu tekstowego '123'. Wyrażenie '123'/10 zostaje skonwertowane na 123/10, ponieważ dzielenie jest operatorem arytmetycznym.

Wynikiem wyrażenia jest 12.3, ale funkcja `REPEAT()` oczekuje liczby całkowitej wskazującej ilość powtórzeń, więc wynik zostaje zaokrąglony do 12. Następnie wywołanie `REPEAT('X', 12)` generuje ostateczny wynik w postaci ciągu tekstowego składającego się z dwunastu znaków X.

Jeżeli wszystkie argumenty funkcji `CONCAT()` będą niebinarnymi ciągami tekstowymi, wynik również będzie niebinarnym ciągiem tekstowym. Natomiast jeśli choć jeden argument będzie binarnym ciągiem tekstowym, wynik także będzie binarnym ciągiem tekstowym. Argumenty liczbowe są konwertowane na ciągi tekstowe, zgodnie z przedstawionym wcześniej opisem i regułami.

Warto pamiętać, że domyślnie zamiast zgłaszać błąd, MySQL próbuje skonwertować wartości na typ wymagany przez wyrażenie. W zależności od kontekstu przeprowadzana jest konwersja wartości każdej z trzech kategorii podstawowych (liczby, ciągi tekstowe oraz daty i godziny) na wartości innych kategorii. Jednak wartość nie zawsze można skonwertować z jednego typu na inny. Jeżeli wartość przeznaczona do konwersji na dany typ nie wygląda jak prawidłowa wartość dla tego typu, wtedy konwersja zakończy się niepowodzeniem. Konwersja na liczbę elementu takiego jak 'abc', który nie przypomina liczby, spowoduje wygenerowanie wyniku 0. Konwersja na datę lub godzinę wartości nieprzypominającej daty lub godziny spowoduje wygenerowanie wyniku w postaci wartości „zerowej” dla typu docelowego. Na przykład, konwersja ciągu tekstowego 'abc' na datę powoduje wynik w postaci daty „zerowej”, czyli '0000-00-00'. Z drugiej strony, dowolna wartość może być potraktowana jako ciąg tekstowy, więc ogólnie rzecz biorąc, nie ma problemu podczas konwersji wartości na postać ciągu tekstowego.

Aby w trakcie operacji przeprowadzanych na danych wejściowych uniemożliwić konwersję wartości nieprawidłowych na najbliższą poprawną, można włączyć tryb ścisły, który w takim przypadku spowoduje wygenerowanie błędu. Zapoznaj się z podrozdziałem 3.3, zatytułowanym „Jak MySQL obsługuje nieprawidłowe wartości danych?”.

MySQL przeprowadza również mniejsze operacje konwersji typu. Jeżeli w kontekście liczb całkowitych używana jest wartość zmiennoprzecinkowa, będzie ona skonwertowana (i zaokrąglona). Konwersja w drugim kierunku również jest przeprowadzana; liczba całkowita bez problemów może być używana wraz z liczbą zmiennoprzecinkową.

Stałe szesnastkowe są traktowane jako binarne ciągi tekstowe, o ile kontekst wyraźnie nie wskazuje liczby. W kontekście ciągu tekstowego każda para cyfr szesnastkowych jest konwertowana na znak, a wynik używany w ciągu tekstowym, jak pokazano w poniższych przykładach:

<code>0x61</code>	→ 'a'
<code>0x61 + 0</code>	→ 97
<code>X'61'</code>	→ 'a'
<code>X'61' + 0</code>	→ 97
<code>CONCAT(0x61)</code>	→ 'a'
<code>CONCAT(0x61 + 0)</code>	→ '97'
<code>CONCAT(X'61')</code>	→ 'a'
<code>CONCAT(X'61' + 0)</code>	→ '97'

W operacjach porównania kontekst określa, czy stała szesnastkowa będzie traktowana jako binarny ciąg tekstowy, czy jako liczba:

- Poniższe wyrażenie traktuje operandy jako binarne ciągi tekstowe i przeprowadza porównanie bajt po bajcie.

`0x0d0a = '\r\n'` → 1

- Poniższe wyrażenie porównuje stałą szesnastkową z liczbą, a więc podczas porównania stała szesnastkowa będzie skonwertowana na liczbę.

`0x0a = 10` → 1

- Poniższe wyrażenie przeprowadza operację porównania binarnego ciągu tekstowego. Pierwszy bajt lewego operandu ma mniejszą wartość niż pierwszy bajt prawego operandu, więc wynikiem jest `false`.

`0xee00 > 0xff` → 0

- W poniższym wyrażeniu prawy operand w postaci stałej szesnastkowej jest konwertowany na liczbę, ponieważ używany jest operator arytmetyczny. Następnie w operacji porównania lewy operand jest konwertowany na liczbę. Wynikiem jest `true`, ponieważ wartość `0xee00` (60928) jest liczbowo większa niż `0xff` (255).

`0xee00 > 0xff+0` → 1

Istnieje możliwość wymuszenia potraktowania stałej szesnastkowej jako niebinarnego ciągu tekstowego przez wskazanie kodowania znaków lub użycie funkcji `CONVERT()`:

<code>0x61</code>	→ 'a'
<code>0x61 = 'A'</code>	→ 0
<code>_latin1 0x61 = 'A'</code>	→ 1
<code>CONVERT(0x61 USING latin1) = 'A'</code>	→ 1

Pewne operatory wymuszają konwersję operandów na typ oczekiwany przez operator, niezależnie od typów operandów. Przykładem są tutaj operatory arytmetyczne. Ponieważ oczekują one liczb, to operandy są konwertowane na odpowiedni typ:

<code>3 + 4</code>	→ 7
<code>'3' + 4</code>	→ 7
<code>'3' + '4'</code>	→ 7

W trakcie konwersji ciągu tekstowego na liczbę nie wystarczy, aby w dowolnym miejscu ciągu tekstowego znajdowały się cyfry. MySQL nie przeszukuje całego ciągu tekstowego w celu znalezienia liczby, a jedynie jego początek. Jeżeli na początku ciągu tekstowego nie znajduje się cyfra, wynikiem konwersji jest 0:

<code>'1973-2-4' + 0</code>	→ 1973
<code>'12:14:01' + 0</code>	→ 12
<code>'23-skidoo' + 0</code>	→ 23
<code>'-23-skidoo' + 0</code>	→ -23
<code>'carbon-14' + 0</code>	→ 0

Stosowana przez MySQL reguła konwersji ciągu tekstowego na liczbę powoduje przeprowadzenie konwersji przypominających liczby ciągów tekstowych na wartości liczbowe:

<code>'-428.9' + 0</code>	→ -428.9
<code>'3E-4' + 0</code>	→ 0.0003

Jednak taki sposób konwersji nie działa w przypadku stałych wyglądających jak szesnastkowe. Użyte zostaje tylko początkowe zero:

'0xff' + 0 → 0

Operatory bitowe są jeszcze bardziej restrykcyjne od arytmetycznych. W przypadku operatorów bitowych nie wystarczy, aby operandy były liczbami, muszą być liczbami całkowitymi. Jeśli będą innego typu, nastąpi odpowiednia konwersja. Oznacza to, że liczba taka jak 0.3 nie przyjmuje wartości true, choć jest niezerowa — po jej konwersji na liczbę całkowitą wynikiem jest 0. W poniższych wyrażeniach operandy nie przyjmują wartości true, jeśli ich wartość nie wyniesie przynajmniej 1:

0.3 .04	→ 0
1.3 .04	→ 1
0.3 & .04	→ 0
1.3 & .04	→ 0
1.3 & 1.04	→ 1

Operatory dopasowania wzorca oczekują, że będą działały na ciągach tekstowych. Oznacza to możliwość użycia oferowanych przez MySQL operatorów dopasowania wzorca wraz z liczbami, ponieważ będą one skonwertowane na ciągi tekstowe w celu znalezienia dopasowania.

```
12345 LIKE '1%'           → 1
12345 REGEXP '1.*5'       → 1
```

Operatory wielkości (<, <=, = itd.) zależą od kontekstu i są obliczane zgodnie z typami operandów. Poniższe wyrażenie przeprowadza liczbowe porównanie operandów, ponieważ oba operandy są liczbami:

$$2 < 11 \rightarrow 1$$

Poniższe wyrażenie wykorzystuje operandy w postaci ciągów tekstowych i dlatego przeprowadzane jest porównanie leksykalne:

$$'2' < '11' \rightarrow 0$$

Z kolei w poniższych porównaniach stosowane są różne typy, więc MySQL porównuje je jako liczby. Dlatego też wynikiem obu wyrażeń jest true:

'2' < 11	→ 1
2 < '11'	→ 1

Podczas przeprowadzania operacji porównania MySQL konwertuje operandy, gdy zajdzie potrzeba, i stosuje przy tym wymienione poniżej reguły:

- W przypadku operatorów innych niż `<=>` porównanie obejmujące wartość `NULL` zwróci wynik `NULL`. (Operator `<=>` jest jak `=`, za wyjątkiem faktu, że wyrażenie `NULL <=> NULL` zwraca wartość `true`, podczas gdy `NULL = NULL` zwraca wartość `NULL`).
- Jeżeli oba operandy są ciągami tekstowymi, wówczas będą porównywane leksykalnie jako ciągi tekstowe. Binarne ciągi tekstowe są porównywane bajt po bajcie za pomocą liczbowych wartości poszczególnych bajtów. Operacje

porównania niebinarnych ciągów tekstowych są przeprowadzane znak po znaku, używając kolejności sortowania kodowania znaków stosowanego w tych ciągach tekstowych. Jeżeli ciągi tekstowe mają różne kodowania znaków, porównanie może być błędne lub zakończyć się niepowodzeniem i zwrócić bezsensowne wyniki. Z kolei porównanie binarnego i niebinarnego ciągu tekstowego jest traktowane jako porównanie binarnych ciągów tekstowych.

- Jeżeli oba operandy są liczbami całkowitymi, są porównywane liczbowo jako liczby całkowite.
- Stałe szesnastkowe, które nie są porównywane z liczbami, są porównywane jako binarne ciągi tekstowe.
- Poza funkcją `IN()`, jeśli jeden operand jest wartością `TIMESTAMP` lub `DATETIME`, a drugi inną stałą, wtedy operandy są porównywane jako wartości typu `TIMESTAMP`. Dzięki temu operacja porównania lepiej działa w aplikacjach ODBC.
- Jeżeli jeden z operandów jest wartością typu `DECIMAL`, wtedy oba operandy są porównywane jako `DECIMAL`, jeśli drugi z nich jest wartością `DECIMAL` lub liczbą całkowitą. Gdy drugi jest inną wartością, wtedy operandy są porównywane jako podwójnej precyzji wartości zmiennoprzecinkowe.
- W pozostałych przypadkach operandy są porównywane liczbowo jako podwójnej precyzji wartości zmiennoprzecinkowe. Obejmuje to również przypadek porównania ciągu tekstowego i liczby. Ciąg tekstowy jest konwertowany na liczbę o podwójnej precyzji, co skutkuje wartością 0, jeśli ciąg tekstowy nie wygląda jak liczba. Na przykład, ciąg tekstowy `'14.3'` jest konwertowany na `14.3`, ale `'L4.3'` na 0.

3.5.2.1. Reguły interpretacji wartości daty i godziny

MySQL bezproblemowo konwertuje ciągi tekstowe i liczby na wartości daty i godziny, gdy wymaga tego kontekst wyrażenia. Oczywiście, konwersja w drugą stronę również jest możliwa. Wartości daty i godziny są w kontekście liczbowym konwertowane na liczby, w kontekście daty i godziny liczby są konwertowane na wartości daty i godziny. Wspomniana konwersja na wartości daty i godziny jest przeprowadzana podczas przypisywania wartości kolumnie daty lub godziny bądź jeśli funkcja wymaga tego rodzaju wartości. W operacjach porównania ogólna reguła stanowi, że wartości daty i godziny są porównywane jako ciągi tekstowe.

Jeżeli tabela `mytbl` zawiera kolumnę typu `DATE` o nazwie `date_col`, poniższe zapytania działają dokładnie tak samo:

```
INSERT INTO mytbl SET date_col = '2025-04-13';
INSERT INTO mytbl SET date_col = '20250413';
INSERT INTO mytbl SET date_col = 20250413;
```

W poniższych przykładach argument funkcji `TO_DAYS()` jest interpretowany jako taka sama wartość we wszystkich trzech wyrażeniach:

```

TO_DAYS('2025-04-13')      → 739719
TO_DAYS('20250413')        → 739719
TO_DAYS(20250413)           → 739719

```

3.5.2.2. Sprawdzanie lub wymuszanie konwersji typu

Aby przekonać się, jak konwersja typu zostanie obsługowana w wyrażeniu, wykonaj zapytanie SELECT obliczające wyrażenie, co pozwoli Ci na przeanalizowanie wyniku:

```

mysql> SELECT X'41', X'41' + 0;
+-----+-----+
| X'41' | X'41' + 0 |
+-----+-----+
| A     | 65         |
+-----+-----+

```

Jeżeli na podstawie analizy wyrażenia nie jesteś w stanie określić jego typu, umieść je w nowej tabeli, a następnie sprawdź definicję tej tabeli:

```

mysql> CREATE TABLE t SELECT X'41' AS col1, X'41' + 0 AS col2;
mysql> DESCRIBE t;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| col1  | varbinary(1)  | NO   |     |          |       |
| col2  | double(17,0)  | NO   |     | 0        |       |
+-----+-----+-----+-----+-----+-----+

```

Sprawdzenie sposobu obliczenia wyrażenia jest szczególnie użyteczne w przypadku zapytań modyfikujących rekordy (takich jak DELETE i UPDATE), ponieważ chcesz mieć wtedy pewność, że zapytanie obejmie jedynie odpowiednie rekordy. Jednym ze sposobów sprawdzenia wyrażenia jest wykonanie wcześniej zapytania SELECT wraz z klauzulą WHERE, która ma być użyta w zapytaniu DELETE lub UPDATE. W ten sposób przekonasz się, czy zapytanie obejmuje właściwe rekordy. Przyjmujemy założenie, że tabela mytbl1 ma kolumnę typu CHAR o nazwie char_col zawierającą poniższe wartości:

```

'abc'
'00'
'def'
'00'
'ghi'

```

Biorąc pod uwagę powyższe wartości, zastanów się, jaki będzie efekt wykonania następującego zapytania:

```
DELETE FROM mytbl1 WHERE char_col = 00;
```

Oczekiwanym efektem jest prawdopodobnie usunięcie dwóch rekordów zawierających wartość '00'. Jednak w rzeczywistości zostaną usunięte wszystkie rekordy, co jest nieprzyjemną niespodzianką. To konsekwencja stosowanych przez MySQL reguł porównania. Kolumna char_col jest kolumną ciągu tekstowego, ale wartość 00 w zapytaniu nie została ujęta w apostrofy, więc jest traktowana jako liczba. Według stosowanych przez MySQL reguł porównania, w przypadku operacji porównania obejmującej ciąg tekstowy i liczbę

następuje porównanie dwóch liczb. Gdy MySQL wykonuje zapytanie DELETE, każdą wartość kolumny char_col konwertuje na liczbę, którą następnie porównuje do zera. Niestety, wartość '00' zostaje skonwertowana na 0, podobnie jak wszystkie ciągi tekstowe, które nie wyglądają jak liczby. Dlatego też klauzula WHERE przyjmuje wartość true dla każdego rekordu, a zapytanie DELETE opróżnia tabelę. To jest przypadek, w którym roztropność nakazywałaby sprawdzenie klauzuli WHERE za pomocą zapytania SELECT jeszcze przed wykonaniem zapytania DELETE. Wtedy można odkryć, że użyte wyrażenie powoduje wybór zbyt dużej liczby rekordów:

```
mysql> SELECT char_col FROM mytbl WHERE char_col = 00;
```

```
+-----+
| char_col |
+-----+
| abc      |
| 00       |
| def      |
| 00       |
| ghi      |
+-----+
```

Kiedy nie pewności co do sposobu, w jaki zostanie użyta wartość, wtedy można wykorzystać konwersję typów w MySQL w celu wymuszenia w wyrażeniu wykorzystania wartości o wskazanym typie. Alternatywą będzie wywołanie funkcji przeprowadzającej żadaną konwersję. Poniżej znajdziesz kilka przykładów użytecznych technik konwersji.

Dodanie +0 lub +0.0 do komponentu powoduje wymuszenie konwersji na wartość liczbową:

```
0x65          → 'e'
0x65 + 0      → 101
0x65 + 0.0    → 101.0
```

W celu usunięcia części ułamkowej liczby należy użyć funkcji FLOOR() lub CAST(). Z kolei dodanie części ułamkowej do liczby całkowitej wymaga dodania dokładnej wartości zero wraz z wymaganą liczbą cyfr dziesiętnych:

```
FLOOR(13.3)      → 13
CAST(13.3 AS SIGNED) → 13
13 + 0.0         → 13.0
13 + 0.0000      → 13.0000
```

Aby zaokrąglić liczbę, trzeba użyć funkcji ROUND() zamiast CAST().

Użycie funkcji CAST() lub CONCAT() powoduje zamianę wartości na ciąg tekstowy:

```
14          → 14
CAST(14 AS CHAR) → '14'
CONCAT(14)   → '14'
```

Użycie funkcji HEX() przeprowadza konwersję liczby na postać szesnastkowego ciągu tekstowego:

```
HEX(255)      → 'FF'
HEX(65535)    → 'FFFF'
```

Istnieje również możliwość użycia funkcji `HEX()` wraz z wartością ciągu tekstowego, aby skonwertować ją na postać szesnastkowego ciągu tekstowego par cyfr przedstawiających kolejne bajty w ciągu tekstowym:

```
HEX('abcd');                                → '61626364'
```

Użycie funkcji `ASCII()` powoduje konwersję znaku opisywanego jednym bajtem na jego wartość ASCII:

```
'A'                                          → 'A'
ASCII('A')                                → 65
```

Aby przeprowadzić konwersję w drugą stronę, z kodu ASCII na znak, trzeba użyć funkcji `CHAR()`:

```
CHAR(65)                                   → 'A'
```

Funkcja `DATE_ADD()` lub `INTERVAL` powoduje wymuszenie potraktowania ciągu tekstowego lub liczby jako daty:

```
DATE_ADD(20130101, INTERVAL 0 DAY)         → '2013-01-01'
20130101 + INTERVAL 0 DAY                  → '2013-01-01'
DATE_ADD('20130101', INTERVAL 0 DAY)       → '2013-01-01'
'20130101' + INTERVAL 0 DAY                → '2013-01-01'
```

Ogólnie rzecz ujmując, wartość daty można skonwertować na liczbę, po prostu dodając zero:

```
CURDATE()                                 → '2013-01-09'
CURDATE()+0                              → 20130109
```

Wartości daty i godziny wraz z częścią godziny można skonwertować na postać zawierającą również mikrosekundy:

```
NOW()                                     → '2012-06-25 09:35:17'
NOW()+0                                   → 20120625093517.000000
CURTIME()                                → '09:35:17'
CURTIME()+0                              → 93517.000000
```

Aby pozbyć się części ułamkowej, wartość należy rzutować na postać liczby całkowitej:

```
CAST(NOW() AS UNSIGNED)                  → 20130109153603
CAST(CURTIME() AS UNSIGNED)              → 153603
```

W celu konwersji ciągu tekstowego ze stosującego jedno kodowanie znaków na inne trzeba użyć funkcji `CONVERT()`. Aby się przekonać, czy wynik stosuje żądane kodowanie znaków, użyj funkcji `CHARSET()`:

```
'abcd'                                    → 'abcd'
CONVERT('abcd' USING ucs2)               → 'abcd'
CHARSET('abcd')                          → 'latin1'
CHARSET(CONVERT('abcd' USING ucs2))       → 'ucs2'
```


Poprzedzenie ciągu tekstowego nazwą kodowania znaków nie powoduje konwersji ciągu tekstowego. MySQL jedynie interpretuje ten ciąg tekstowy jako stosujący wskazane kodowanie znaków:

```
CHARSET(_ucs2 'abcd') → 'ucs2'
```

Aby określić wartość szesnastkową znaku UTF-8 odpowiadającego znakowi UCS-2 o podanej wartości szesnastkowej, należy połączyć funkcje `CONVERT()` i `HEX()`. Poniższe wyrażenie pozwala na ustalenie wartości UTF-8 symbolu przedstawiającego znak zastrzeżony:

```
HEX(CONVERT(_ucs2 0x2122 USING utf8)) → 'E284A2'
```

W celu zmiany kolejności sortowania w ciągu tekstowym należy użyć operatora `COLLATE`. Aby się przekonać, czy wynik stosuje żadaną kolejność sortowania, użyj funkcji `COLLATION()`:

```
COLLATION('abcd') → 'latin1_swedish_ci'
COLLATION('abcd' COLLATE latin1_bin) → 'latin1_bin'
```

Kodowanie znaków i kolejność sortowania muszą być zgodne ze sobą. Jeżeli nie są, użyj funkcji `CONVERT()` w celu konwersji kodowania znaków, a następnie operatora `COLLATE` do zmiany kolejności sortowania:

```
CONVERT('abcd' USING latin2) COLLATE latin2_bin
```

W celu konwersji binarnego ciągu tekstowego na niebinarny o wskazanym kodowaniu znaków trzeba użyć funkcji `CONVERT()`:

```
0x61626364 → 'abcd'
0x61626364 = 'ABCD' → 0
CONVERT(0x61626364 USING latin1) = 'ABCD' → 1
```

Ewentualnie, w przypadku cytowanych binarnych ciągów tekstowych lub wartości szesnastkowych, zmianę interpretacji binarnego ciągu tekstowego można wymusić przez jego poprzedzenie nazwą kodowania znaków:

```
_latin1 0x61626364 = 'ABCD' → 1
```

Aby rzutować niebinarny ciąg tekstowy na binarny, należy użyć słowa kluczowego `BINARY`:

```
'abcd' = 'ABCD' → 1
BINARY 'abcd' = 'ABCD' → 0
'abcd' = BINARY 'ABCD' → 0
```

3.6. Wybór typu danych

W podrozdziale 3.2, zatytułowanym „Typy danych w MySQL”, omówiono dostępne typy danych, ich ogólne właściwości, takie jak rodzaj przechowywanych wartości, a także wymaganą przez poszczególne typy ilość pamięci masowej. W jaki sposób jednak wybierasz typ danych podczas tworzenia tabeli? W tym podrozdziale zostaną przedstawione czynniki, których rozważenie pomoże w dokonaniu odpowiedniego wyboru.

Najbardziej „ogólnym” typem danych jest ciąg tekstowy. Niemalże wszystko można przechowywać w postaci ciągu tekstowego, nawet liczby i daty mogą być wyrażane w tej postaci. Rodzi się więc pytanie: czy wszystkie kolumny powinny zostać zdefiniowane jako typu ciągu tekstowego? Nie. Przeanalizujmy prosty przykład. Przyjmujemy założenie, że masz wartości wyglądające jak liczby. Wprawdzie możesz je przedstawić w postaci ciągów tekstowych, ale czy na pewno powinieneś? Co się wówczas zdarzy?

Z jednej strony, prawdopodobnie zużyjesz większą ilość pamięci masowej na przechowywanie takich danych, ponieważ liczby mogą być przechowywane znacznie efektywniej za pomocą kolumn liczbowych zamiast ciągu tekstowego. Zauważysz także pewne różnice w wynikach zapytania wynikające z odmiennego sposobu obsługi liczb i ciągów tekstowych. Na przykład, kolejność sortowania liczb nie jest taka sama jak ciągów tekstowych. Liczba 2 jest mniejsza od liczby 11, ale pod względem leksykalnym ciąg tekstowy '2' jest większy niż ciąg tekstowy '11'. Możesz sobie z tym poradzić, używając kolumn w kontekście liczbowym, na przykład:

```
SELECT nazwa_kolumny + 0 as num ... ORDER BY num;
```

Dodanie zera do kolumny wymusza przeprowadzanie sortowania w sposób liczbowy, ale czy to jest rozsądne rozwiązanie? Czasami to użyteczna technika, ale nie powinieneś jej stosować za każdym razem, gdy chcesz używać sortowania liczbowego. Wymuszenie na MySQL traktowania kolumny ciągu tekstowego jako liczbowej wiąże się z kilkoma poważnymi implikacjami. Przede wszystkim, wymusza konwersję ciągu tekstowego na liczbę dla każdej wartości kolumny, co samo w sobie jest bardzo nieefektywne. Ponadto, użycie kolumny w obliczeniach uniemożliwia MySQL zastosowanie jakiegokolwiek indeksu dla tej kolumny, co jeszcze bardziej spowalnia wykonywanie zapytań. Jeżeli wartości będą przechowywane w postaci liczb, wspomniana degradacja wydajności nie będzie miała miejsca.

Powyższy przykład wskazał kilka kwestii, które trzeba rozważyć podczas wyboru typu danych. Prosty wybór polegający na użyciu jednego typu zamiast innego ma implikacje w postaci ilości pamięci masowej wymaganej do przechowywania danych, zmiany sposobu obsługi zapytań, a także wydajności ich przetwarzania. Przedstawiona poniżej lista zawiera omówienie czynników, które należy wziąć pod uwagę podczas wyboru typu danych dla kolumny.

Jakie wartości będą przechowywane w kolumnie? Liczby? Ciągi tekstowe? Daty? Dowolnego typu wartość można przedstawić w postaci ciągu tekstowego, ale jak się już przekonałeś, jeśli inny typ będzie bardziej odpowiedni, wtedy prawdopodobnie jego użycie zapewni lepszą wydajność. Jednak wcześniejsze ustalenie rodzaju wartości, jakie będą przechowywane w kolumnie, nie zawsze jest łatwe, zwłaszcza gdy dane będą wstawiane przez innych użytkowników. Jeżeli tabelę przygotowujesz dla kogoś innego, koniecznie trzeba dowiedzieć się, jakie dane będą w niej przechowywane. Do podjęcia dobrej decyzji konieczne jest zadanie pytań i otrzymanie wymaganych informacji.

Czy przechowywane wartości pochodzą z określonego zakresu? Jeżeli przechowywane będą liczby całkowite, warto wiedzieć, czy zawsze będą dodatnie. W takim przypadku można użyć typu UNSIGNED. Jeśli danymi będą ciągi tekstowe, warto sprawdzić, czy zawsze będą wybierane ze stałego, ograniczonego zestawu wartości. W takim przypadku użyteczne może być zastosowanie typu ENUM lub SET.

Istnieje pewna zależność między zakresem typu i ilością wymaganej przez niego pamięci masowej. Jak „dużego” typu danych potrzebujesz? W przypadku liczb można wybrać mały typ danych, oferujący ograniczony zakres wartości, lub ogromny typ o znacznie większym zakresie. Z kolei ciągi tekstowe mogą być krótsze lub dłuższe, więc nie ma sensu definiować typu CHAR(255), jeśli wszystkie przechowywane wartości będą miały poniżej 10 znaków.

Czy występują jakiegokolwiek kwestie związane z wydajnością i efektywnością?

Pewne typy mogą być przetwarzane znacznie efektywniej niż inne. Ogólnie rzecz biorąc, operacje liczbowe mogą być przeprowadzane znacznie szybciej niż operacje na ciągach tekstowych. Porównanie krótszych ciągów tekstowych odbywa się znacznie szybciej niż dłuższych, a ponadto wiąże się z mniejszym obciążeniem dysku. W przypadku tabel MyISAM użycie rekordów o stałej wielkości zapewnia lepszą wydajność niż dla rekordów o zmiennej wielkości.

W poniższych punktach znajdziesz dokładniejsze omówienie wymienionych kwestii za wyjątkiem kwestii wydajności, poruszonych w podrozdziale 5.3, zatytułowanym „Wybór typu danych zapewniającego efektywne wykonywanie zapytań”.

Wprawdzie podczas tworzenia tabeli chcesz dokonać najlepszego wyboru typu danych, ale jeśli wybór nie okaże się optymalny, to naprawdę nic złego się nie stało. Zawsze możesz użyć zapytania ALTER TABLE i zmienić typ na lepszy. Wspomniana operacja może być prosta i sprowadzać się do zmiany typu SMALLINT na MEDIUMINT po odkryciu, że zbiór danych zawiera więcej wartości, niż pierwotnie sądziłeś. Oczywiście, operacja może okazać się bardziej skomplikowana i wymagać na przykład zmiany typu CHAR na ENUM wraz z pewnym zbiorem dozwolonych wartości.

3.6.1. Jakie wartości będą przechowywane w kolumnie?

Podczas wyboru typu danych pierwszym czynnikiem, który należy wziąć pod uwagę, jest rodzaj wartości, jakie będą przechowywane w kolumnie, ponieważ to ma największy wpływ na wybierany typ. Ogólnie rzecz biorąc, dokonujesz oczywistych wyborów: liczby przechowujesz w kolumnach liczbowych, ciągi tekstowe w kolumnach ciągów tekstowych, natomiast daty i godziny w kolumnach daty i godziny. Jeżeli liczby mają część ułamkową, wtedy zamiast typu liczb całkowitych wybierasz typ DECIMAL lub zmiennoprzecinkowy. Czasami zdarzają się pewne wyjątki. Najważniejsze jest zrozumienie natury danych, aby móc dokonać świadomego wyboru. Jeżeli będziesz przechowywał własne dane, wtedy prawdopodobnie potrafisz je dobrze scharakteryzować. Kiedy ktoś inny prosi Cię o przygotowanie tabeli, to czasami zupełnie inna sytuacja. Nie zawsze w łatwy sposób można ustalić dane, które będą przechowywane w bazie danych. Upewnij się, że zadałeś wystarczającą liczbę pytań, aby trafnie ustalić rodzaj wartości, jakie zostaną umieszczone w tabeli.

Przyjmujemy założenie, że otrzymałeś następujące informacje: tabela potrzebuje kolumny przeznaczonej do przechowywania danych o „poziomie opadów”. Czy to będą dane liczbowe? A może „w większości” liczbowe, to znaczy zwykle, ale nie zawsze wyrażone w postaci liczby? Na przykład, gdy oglądasz prognozę pogody w telewizji, najczęściej

podawana jest wówczas informacja o poziomie opadów. Czasami może być w postaci liczbowej (na przykład 10 mm deszczu), innym razem w postaci opisowej, na przykład „niewielkie opady”. Takie podejście jest odpowiednie podczas prezentacji prognozy pogody widzom, ale w jaki sposób te informacje umieścić w bazie danych? Trzeba będzie wyrażenie „niewielkie opady” zdefiniować jako konkretną liczbę, aby móc ją przechowywać w kolumnie typu liczbowego, lub użyć ciągu tekstowego w celu przechowywania całego wyrażenia „niewielkie opady”. Inna możliwość to jeszcze bardziej skomplikowane rozwiązanie, oparte na połączeniu kolumny liczbowej z kolumną ciągu tekstowego, gdzie wypełniana będzie jedna kolumna, a druga otrzyma wówczas wartość NULL. Jeśli istnieje taka możliwość, unikaj takiego rozwiązania, ponieważ tabela staje się trudniejsza do zrozumienia, a same zapytania trudne do przygotowania.

Prawdopodobnie najlepiej spróbować przechowywać wszystkie rekordy w postaci liczbowej, a następnie wartości konwertować na inne typy w celach prezentacji. Na przykład, jeśli dowolna niezerowa ilość opadów mniejsza niż 10 mm oznacza niewielkie opady, wtedy wartość prezentacyjną dla kolumny można wygenerować za pomocą poniższego zapytania

```
SELECT IF(precip>0 AND precip<.01,'niewielkie opady',precip) FROM ... ;
```

Pewne wartości są niewątpliwie liczbowe, ale konieczne jest ustalenie, czy powinien być użyty typ liczb całkowitych, czy innych. W takim przypadku należy poznać wymagania w zakresie jednostki danych i ich precyzji. Czy wystarczające jest zastosowanie liczb całkowitych, czy raczej należy użyć jednostki wraz z częścią ułamkową? To może pomóc w rozróżnieniu między typami liczb całkowitych i zmiennoprzecinkowych. Na przykład, jeśli waga zostaje zaokrąglona do najbliższego kilograma, wtedy można użyć kolumny liczb całkowitych. Aby przechowywać jednostki wraz z częścią ułamkową, możesz użyć typu danych o stałej liczbie cyfr lub zmiennoprzecinkowego. W pewnych przypadkach można nawet użyć wielu kolumn, na przykład w celu przechowywania wagi wyrażonej w kilogramach i gramach.

Odległość to liczbowy typ informacji, które można przedstawić na kilka sposobów:

- Za pomocą ciągu tekstowego, takiego jak '1,8m' dla wartości w stylu „metr osiemdziesiąt”. Zaletą takiej formy jest możliwość szybkiego odczytania i łatwiejszego zrozumienia wartości niż na przykład w postaci 1800 mm. Tego rodzaju wartości jednak trudno użyć w operacjach matematycznych, takich jak obliczanie sumy lub wartości średniej.
- Użycie po jednej kolumnie liczbowej do przechowywania metrów i centymetrów. Wprawdzie to nieco ułatwi przeprowadzanie operacji liczbowych, ale dwie kolumny są trudniejsze w użyciu niż jedna.
- Użycie jednej kolumny liczbowej do przechowywania centymetrów. To najłatwiejsze rozwiązanie do zastosowania w bazie danych i jednocześnie najmniej zrozumiałe dla człowieka. Pamiętaj, że nie będziesz prezentować wartości w takiej samej postaci, w jakiej są przechowywane. Wartościom zawsze można nadać inny format, używając do tego wielu dostępnych funkcji MySQL. W ten sposób będziesz mógł wybrać format pozwalający w najlepszy sposób przedstawić odległość.

Inny typ danych liczbowych to waluta, na przykład polskie złote. Obliczenia przeprowadzane na tych wartościach mają komponent złotych i groszy. Wyglądają jak wartości zmiennoprzecinkowe, ale typy `FLOAT` i `DOUBLE` podlegają błędowi zaokrąglenia, a tym samym nie są odpowiednie do przechowywania wartości walutowych, poza rekordami, w których wystarczy tylko wartość przybliżona. Ponieważ ludzie są drażliwi, gdy chodzi o ich pieniądze, należy wybrać typ zapewniający doskonałą dokładność. Masz tutaj kilka możliwości:

- Przedstawić pieniądze jako typ `DECIMAL(M,2)`, gdzie M wskazuje maksymalną liczbę cyfr odpowiednią dla wymaganego zakresu wartości. W ten sposób wartości będą podawane z dokładnością dwóch miejsc po przecinku dziesiętnym. Zaletą użycia typu `DECIMAL` jest to, że wartości nie podlegają błędowi zaokrąglenia i są dokładne.
- Wszystkie wartości pieniężne przedstawiać wewnętrznie jako grosze za pomocą typu liczb całkowitych. Zaletą jest fakt, że wszystkie obliczenia będą wewnętrznie przeprowadzane za pomocą liczb całkowitych, co jest bardzo szybkie. Wadą natomiast jest konieczność konwersji wartości danych wejściowych lub wyjściowych przez ich mnożenie lub dzielenie przez 100.

Pewne wartości liczb nimi nie są. W numerach telefonów, kart kredytowych i NIP-u znajdują się znaki inne niż cyfry, na przykład spacje i myślniki. Takie numery nie mogą być bezpośrednio przechowywane w kolumnie liczbowej, o ile znaki inne niż cyfry nie zostaną usunięte. Jednak nawet po usunięciu znaków innych niż cyfry tego rodzaju wartości lepiej przechowywać w postaci ciągów tekstowych zamiast liczb, aby uniknąć utraty zer na początku.

Jeżeli konieczne jest przechowywanie informacji o dacie, warto sprawdzić, czy będą one zawierały także wartości godziny, a dokładniej czy *kiedykolwiek* wystąpi konieczność przechowywania informacji o godzinie. MySQL nie oferuje typu danych posiadającego opcjonalną część godziny: typ `DATE` nie przechowuje informacji o godzinie, natomiast typ `DATETIME` wymaga jej podania. Jeżeli godzina naprawdę jest opcjonalna, użyj kolumny typu `DATE` do przechowywania daty i oddzielnej kolumny `TIME` dla godziny. Następnie zdefiniuj możliwość wstawienia wartości `NULL` do kolumny typu `TIME`, co będzie interpretowane jako „brak godziny”:

```
CREATE TABLE mytbl
(
    date DATE NOT NULL,      # Data jest wymagana.
    time TIME NULL          # Godzina jest opcjonalna (może mieć wartość NULL).
);
```

Istnieje jedna sytuacja, w której szczególnie ważne staje się ustalenie, czy wartość godziny będzie potrzebna: to złączenie dwóch tabel w relacji główny – szczegółowy połączonych na podstawie informacji daty. Przyjmujemy założenie, że przeprowadzasz badania obejmujące wykonywanie testów. Po przeprowadzeniu standardowego testu początkowego baterii chcesz przeprowadzić kilka dodatkowych testów, których wybór będzie zależał od wyników testów początkowych. Tego rodzaju informacje można przechowywać za pomocą relacji główny – szczegółowy, w której informacje identyfikujące

przedmiot testu i testy standardowe są przechowywane w rekordzie głównym, natomiast wszelkie testy dodatkowe jako rekordy w drugiej tabeli. Połączenie między tabelami opiera się na identyfikatorze przedmiotu testu i dacie jego przeprowadzenia.

Pytanie, na które trzeba sobie odpowiedzieć w przedstawionej sytuacji, brzmi: czy można użyć jedynie daty, czy kiedykolwiek konieczne będzie użycie daty i godziny? To zależy od przedmiotu testu i od tego, czy procedurę testu można przeprowadzić więcej niż tylko jeden raz w ciągu dnia. Jeżeli tak, wtedy zarejestruj również godzinę (na przykład godzinę rozpoczęcia testu) w kolumnie typu DATETIME lub w oddzielnych kolumnach DATE i TIME wymagających wypełnienia. Bez informacji o godzinie nie będzie możliwości powiązania rekordów informacji szczegółowych z odpowiednimi rekordami głównymi, jeśli test zostanie przeprowadzony więcej niż tylko raz w ciągu dnia.

Już słyszę twierdzenia w stylu: „Nie potrzebuję wartości godziny, nigdy nie będę przeprowadzał dwóch takich samych testów jednego dnia”. Czasami użytkownicy mają rację, ale bardzo często zdarza się również, że później zastanawiają się, jak uniemożliwić mieszanie rekordów informacji szczegółowych z nieodpowiednimi rekordami głównymi z powodu kilkukrotnego przeprowadzenia tego samego testu w ciągu dnia. Niestety, teraz jest już za późno!

Czasami rozwiązaniem problemu jest umieszczenie kolumny typu TIME w tabeli. Niestety, uaktualnienie istniejących rekordów jest trudne, o ile nie dysponujesz niezależnym źródłem danych, takim jak zapiski na papierze. W przeciwnym razie nie ma możliwości ujednoliczenia rekordów informacji szczegółowych przez ich powiązanie z poprawnymi rekordami głównymi. Nawet jeśli masz niezależne źródło informacji, prawdopodobnie będzie ono powodowało problemy dla aplikacji, którą utworzyłeś w celu używania przygotowanych tabel. Najlepszym rozwiązaniem jest dokładne wyjaśnienie problemów właścicielom tabel i upewnienie się o otrzymaniu prawidłowej charakterystyki danych przed przystąpieniem do tworzenia dla nich tabel.

Czasami masz niekompletne dane i to będzie miało wpływ na wybór typu danych. Możesz zbierać daty urodzenia i śmierci w celu przeprowadzenia badań genealogicznych. Może się zdarzyć, że znajdziesz jedynie miesiąc i rok daty urodzenia bądź śmierci, a nie dokładną datę. Jeżeli używasz kolumny typu DATE, nie możesz wprowadzić niekompletnej daty. Aby mieć możliwość wprowadzenia informacji, nawet niekompletnych, powinieneś przygotować oddzielne kolumny dla roku, miesiąca i dnia. Następnie możesz do nich wstawiać znane informacje, w pozostałych zaś użyć wartości NULL. Inną możliwością jest użycie wartości DATE, w których komponent dnia lub miesiąca będzie miał przypisaną wartość 0. Tego rodzaju wartość może być używana do przedstawiania niekompletnych wartości dat.

3.6.2. Czy przechowywane wartości pochodzą z określonego zakresu?

Jeżeli zdecydowałeś się na ogólną kategorię, z której wybierzesz typ danych dla kolumny, wtedy zastanów się nad zakresem wartości umieszczanych w kolumnie. Dzięki temu będziesz mógł zawęzić wybór do określonego typu w danej kategorii. Przyjmujemy

założenie, że chcesz przechowywać wartości w postaci liczb całkowitych. Zakres wartości determinuje typ danych. Jeżeli potrzebne są wartości z zakresu od 0 do 1000, wtedy do wyboru masz dowolny typ, od `SMALLINT` do `BIGINT`. W przypadku zakresu wartości do 2 000 000 nie można użyć `SMALLINT` i wybór zawęży się do typów od `MEDIUMINT` do `BIGINT`.

Oczywiście, możesz po prostu zdecydować się na największy typ dla danego rodzaju wartości, które chcesz przechowywać (`BIGINT` w omawianym przykładzie). Jeśli jednak użyjesz najmniejszego typu wystarczającego dla zdefiniowanych potrzeb, zminimalizujesz ilość pamięci masowej wykorzystanej przez tabele. Ponadto, uzyskasz lepszą wydajność, ponieważ mniejsze kolumny najczęściej mogą być przetwarzane znacznie szybciej niż duże. Odczyt mniejszych wartości wymaga mniejszej aktywności dysku, w buforze indeksu umieszczonym w pamięci zmieści się większa liczba kluczy, co zwiększa szybkość operacji wyszukiwania w indeksie.

Jeżeli nie znasz zakresu możliwych wartości, wtedy musisz zgadywać lub użyć typu `BIGINT` w celu zapewnienia obsługi w najgorszym możliwym scenariuszu. Jeśli zgadujesz, ale później typ okaże się zbyt mały, nie wszystko jest stracone. Użyj zapytania `ALTER TABLE` i zmień typ danych.

Czasami możesz odkryć, że kolumna jest zbyt mała. W podpunkcie 1.4.6.2.3, zatytułowanym „Tabela score”, utworzyliśmy tabelę `score` dla projektu ocen uczniów, zawierającą kolumnę `score` przeznaczoną do zapisywania ocen uzyskanych w trakcie sprawdzianów i testów. Wspomniana kolumna została utworzona jako typu `INT`, aby uprościć ówczesną analizę. Skoro przekonał się, że oceny są z zakresu od 0 do 100, to lepszym wynikiem będzie `TINYINT UNSIGNED`, ponieważ oznacza to mniejsze zużycie pamięci masowej.

Zakres wartości w danych wpływa także na atrybuty, które mogą być stosowane wraz z typem danych. Jeżeli wartości nigdy nie są ujemne, wtedy można użyć `UNSIGNED`. W przeciwnym nie należy używać wymienionego atrybutu.

Typy ciągów tekstowych nie mają „zakresu” definiowanego w sposób taki jak w kolumnach liczbowych. Mają jednak zdefiniowaną długość, a maksymalna długość wpływa na typ danych. Jeżeli przechowywane będą ciągi tekstowe krótsze niż 256 znaków, wtedy można użyć typu `CHAR`, `VARCHAR` lub `TINYTEXT`. W przypadku dłuższych ciągów tekstowych do dyspozycji masz typ `VARCHAR`. Dla najdłuższych ciągów tekstowych można wykorzystać typ `TEXT`.

Dla kolumn ciągów tekstowych przeznaczonych do przechowywania na stałe zdefiniowanego zbioru wartości rozważ użycie typu `ENUM` lub `SET`. To dobry wybór, ponieważ wewnętrznie są one przedstawiane w postaci liczb. Ponieważ przeprowadzane na nich operacje odbywają się liczbowo, są znacznie efektywniejsze niż inne typy ciągów tekstowych. Mogą być również znacznie mniejsze, co przekłada się na oszczędność pamięci masowej. Ponadto, po włączeniu trybu ścisłego SQL zyskujesz możliwość odrzucania wartości niewymienionej na liście wartości. Więcej informacji na ten temat znajdziesz w podrozdziale 3.3, zatytułowanym „Jak MySQL obsługuje nieprawidłowe wartości danych?”.

Kiedy charakteryzujesz zakres wartości, z którymi trzeba będzie pracować, najlepsze pojęcia to „zawsze” i „nigdy” (na przykład „zawsze mniejsze niż 1000” i „nigdy ujemne”), ponieważ pozwalają one na ograniczenie wyboru typu danych. Jednak nie zawsze użycie wymienionych pojęć jest uzasadnione. Zachowaj szczególną ostrożność, gdy podczas

prrowadzenia z innymi osobami konsultacji dotyczących ich danych nagle zaczynają one nadużywać wymienionych pojęć. Kiedy ktoś używa słowa „zawsze” i „nigdy”, musisz się upewnić, że naprawdę ma to na myśli. Czasami użytkownicy twierdzą, że ich dane zawsze mają pewną cechę charakterystyczną, gdy tak naprawdę mają na myśli „prawie zawsze”.

Przyjmujemy założenie, że projektujesz tabelę dla grupy nauczycieli, którzy twierdzą: „Oceny w naszych testach zawsze są z zakresu od 0 do 100”. Opierając się na tym twierdzeniu, wybierasz typ TINYINT i używasz atrybutu UNSIGNED, ponieważ wartości nigdy nie są ujemne. Następnie odkrywasz, że osoby wstawiające dane do bazy danych często używają zapisu -1, oznaczającego „uczeń był nieobecny ze względu na chorobę”. Ups. Tego nauczyciele Ci nie powiedzieli. Do przedstawienia tego rodzaju wartości odpowiednie może być użycie NULL. Jeśli nie, to musisz wstawić wartość -1, ale wówczas nie można używać kolumny z atrybutem UNSIGNED. (W takich sytuacjach ratunkiem jest zapytanie ALTER TABLE).

Czasami decyzja dotycząca tego rodzaju sytuacji może być podjęta znacznie łatwiej dzięki zadaniu prostego pytania: czy istnieją jakiegokolwiek wyjątki? Jeżeli istnieje choć jeden wyjątek, musisz go uwzględnić. Rozmawiając z użytkownikami o projekcie bazy danych, przekonasz się o jednym: oni niezmiennie sądzą, że jeśli wyjątek nie pojawia się zbyt często, to nie ma znaczenia. Tworząc tabelę, nie możesz myśleć w taki sposób. Pytanie, które musisz zadać, nie brzmi: jak często występuje wyjątek? Pytanie musi brzmieć: *czy kiedykolwiek* występują wyjątki? Jeśli tak, to na pewno musisz je uwzględnić w projekcie tabeli.

Widoki i programy składowane

MySQL obsługuje wiele typów obiektów po stronie serwera. Wspomniane obiekty tworzysz, a serwer przechowuje je w celu późniejszego wykorzystania.

Widok to jeden z typów obiektu przechowywanego i jest tabelą wirtualną. Oznacza to, że działa jak tabela, ale nie zawiera żadnych danych. Zamiast tego jest zdefiniowany w kategoriach tabel lub innych widoków i zapewnia alternatywny sposób spojrzenia na dane tabeli. Widoki mogą ułatwić tworzenie aplikacji przez dostarczenie prostego sposobu na wykonywanie skomplikowanych zapytań.

Program składowany to inny typ obiektu przechowywanego i występuje w wielu postaciach. Niektóre programy składowane mogą być wywoływane na żądania. Jeszcze inne są wykonywane automatycznie po wykryciu modyfikacji tabeli lub wedle harmonogramu:

- Funkcja składowana zwraca wynik obliczeń używany następnie w wyrażeniach.
- Procedura składowana nie zwraca wyniku bezpośrednio, ale jest używana do przeprowadzania ogólnych obliczeń lub generowania zbiorów danych przekazywanych z powrotem klientowi.
- Wywołanie jest powiązane z tabelą i następuje po jej modyfikacji za pomocą zapytania INSERT, DELETE lub UPDATE.
- Zdarzenie jest wykonywane o konkretnej godzinie, zgodnie z harmonogramem.

Programy składowane oferują wiele zalet i możliwości:

- Część wykonywalna obiektu może używać zapytań złożonych rozszerzających składnię SQL o bloki, pętle i zapytania warunkowe.
- Cały kod wymagany do zdefiniowania programu składowanego jest wysyłany przez sieć tylko jeden raz w chwili tworzenia programu, a nie w trakcie jego każdego wywołania. W ten sposób następuje zmniejszenie obciążenia w chwili wywołania programu składowanego.
- Program składowany pozwala na hermetyzację skomplikowanych obliczeń w komponencie programu, który można łatwo wywoływać za pomocą jego nazwy.

- Program składowany pozwala na standaryzację operacji obliczeniowych, ponieważ wszystkie używające go aplikacje będą wspomniane operacje przeprowadzały w taki sam sposób.
- Program składowany zapewnia mechanizm służący do obsługi błędów.
- Następuje poprawa bezpieczeństwa bazy danych, ponieważ można włączyć kontrolę dostępu do danych wrażliwych przez odpowiedni wybór uprawnień, jakie program składowany będzie miał w chwili jego wykonywania.

W tym rozdziale stosowana jest poniższa terminologia:

- „Programy składowane” odnosi się ogólnie do funkcji i procedur składowanych, wyzwalaczy oraz zdarzeń.
- „Procedury składowane” to dokładniejsze pojęcie odnoszące się jedynie do funkcji i procedur składowanych. Oba typy obiektów są definiowane przy użyciu podobnej składni, więc często naturalne jest jednocześnie ich omawianie. W rzeczywistości w sporej części literatury poświęconej bazom danych pojęcie „procedury składowane” odnosi się zarówno do funkcji, jak i procedur. Według mnie to niejednoznaczne i nie będziemy używać pojęcia w taki sposób.

Ten rozdział pokazuje, jak tworzyć widoki oraz programy składowane i ich używać. Znajdziesz tutaj również omówienie klauzuli `DEFINER`, wykorzystywanej przez widoki i programy składowane w celu zapewnienia bezpieczeństwa i kontroli dostępu do danych.

Przedstawiony tutaj opis składni dla zapytań `CREATE` prezentuje jedynie najważniejsze klauzule. Pełną składnię znajdziesz w dodatku E, zatytułowanym „Przewodnik po składni SQL”.

4.1. Używanie widoków

Widok jest tabelą wirtualną zdefiniowaną za pomocą zapytania `SELECT` obejmującego tabele lub inne widoki. Wybór danych z widoku odpowiada wyborowi danych za pomocą zapytania definiującego ten widok, ale sam widok ukrywa szczegóły zapytania. Definicja może zawierać operacje takie jak obliczanie wyrażeń i złączenia. Dlatego też pobranie danych z widoku znacznie ułatwia tworzenie prostych zapytań pobierających żądane informacje.

Prosty widok to nic więcej niż jedynie wybór pewnych kolumn tabeli. Przyjmujemy założenie, że z tabeli `president` najczęściej wybierane są jedynie kolumny `last_name`, `first_name`, `city` i `state`, ale nie chcesz w każdym zapytaniu zapisywać ich w postaci:

```
SELECT last_name, first_name, city, state FROM president;
```

Nie chcesz również używać zapytania `SELECT *`. To, oczywiście, łatwiejsze do zapisania, ale gwiazdka powoduje pobranie wszystkich kolumn, także tych, których nie potrzebujesz. Rozwiązaniem jest zdefiniowanie widoku pobierającego jedynie żądane kolumny:

```
CREATE VIEW vpres AS
SELECT last_name, first_name, city, state FROM president;
```

Teraz widok działa w charakterze „okna”, w którym widzisz jedynie wskazane kolumny. Oznacza to, że po użyciu zapytania `SELECT *` z widoku otrzymasz jedynie kolumny wymienione w definicji widoku:

```
mysql> SELECT * FROM vpres;
+-----+-----+-----+-----+
| last_name | first_name | city | state |
+-----+-----+-----+-----+
| Washington | George | Wakefield | VA |
| Adams | John | Braintree | MA |
| Jefferson | Thomas | Albemarle County | VA |
| Madison | James | Port Conway | VA |
| Monroe | James | Westmoreland County | VA |
...
```

W celu utworzenia widoku trzeba mieć uprawnienie `CREATE VIEW`, pewne uprawnienia dla każdej kolumny wskazanej w zapytaniu `SELECT` oraz uprawnienie `SELECT` dla każdej kolumny, do której odwołujesz się w tym zapytaniu.

Jeżeli użyjesz klauzuli `WHERE`, MySQL dołączy ją do definicji widoku podczas wykonywania zapytania i tym samym dalej ograniczy zbiór wynikowy:

```
mysql> SELECT * FROM vpres WHERE last_name = 'Adams';
+-----+-----+-----+-----+
| last_name | first_name | city | state |
+-----+-----+-----+-----+
| Adams | John | Braintree | MA |
| Adams | John Quincy | Braintree | MA |
+-----+-----+-----+-----+
```

To samo dotyczy użytych klauzul `ORDER BY`, `LIMIT` itd.

Kiedy używasz widoku, możesz odwoływać się jedynie do kolumn wymienionych w definicji widoku. Oznacza to brak możliwości odnoszenia się do kolumny niebędącej częścią widoku, nawet jeśli ta kolumna znajduje się w tabeli:

```
mysql> SELECT * FROM vpres WHERE suffix <> '';
ERROR 1054 (42S22): Unknown column 'suffix' in 'where clause'
```

Nazwy kolumn w widoku domyślnie są nazwami kolumn w danych wyjściowych zapytania `SELECT` wykonywanego do widoku. Aby wyraźnie zdefiniować nazwy, trzeba je w definicji widoku umieścić w nawiasie po nazwie widoku:

```
mysql> CREATE VIEW vpres2 (ln, fn) AS
-> SELECT last_name, first_name FROM president;
```

W celu odwołania się do powyższego widoku użyj wskazanych nazw kolumn zamiast nazw podanych w zapytaniu `SELECT` definicji widoku:

```
mysql> SELECT last_name, first_name FROM vpres2;
ERROR 1054 (42S22): Unknown column 'last_name' in 'field list'
mysql> SELECT ln, fn FROM vpres2;
+-----+-----+
| ln | fn |
+-----+-----+
| Washington | George |
| Adams | John |
+-----+-----+
```

```

| Jefferson | Thomas |
| Madison  | James  |
| Monroe   | James  |
...

```

Poprzednie definicje widoków były całkiem proste, ale zapytanie SELECT dla widoku może być znacznie bardziej rozbudowane. W ten sposób zyskujesz możliwość ukrycia poziomu skomplikowania i łatwego pobrania informacji, których otrzymanie wcale nie jest proste. W podpunkcie 1.4.9.10, zatytułowanym „Pobieranie informacji z wielu tabel”, opracowaliśmy zapytanie dla projektu ocen uczniów przeznaczone do pobierania danych statystycznych dotyczących testów i sprawdzianów. Opracowanego tam zapytania możemy użyć jako definicji widoku pozwalającego na znacznie łatwiejsze pobranie tych samych informacji:

```

CREATE VIEW grade_stats AS
SELECT
grade_event.date, grade_event.category,
MIN(score.score) AS minimum,
MAX(score.score) AS maximum,
MAX(score.score)-MIN(score.score)+1 AS span,
SUM(score.score) AS total,
AVG(score.score) AS average,
COUNT(score.score) AS count
FROM score INNER JOIN grade_event
ON score.event_id = grade_event.event_id
GROUP BY grade_event.date;

```

Wybór danych z widoku powoduje przeprowadzenie operacji złączenia i pobiera wyniki obliczeń:

```

mysql> SELECT * FROM grade_stats;
+-----+-----+-----+-----+-----+-----+-----+-----+
| date       | category | minimum | maximum | span | total | average | count |
+-----+-----+-----+-----+-----+-----+-----+
| 2012-09-03 | Q        | 9        | 20       | 12    | 439    | 15.1379  | 29    |
| 2012-09-06 | Q        | 8        | 19       | 12    | 425    | 14.1667  | 30    |
| 2012-09-09 | T        | 60       | 97       | 38    | 2425   | 78.2258  | 31    |
| 2012-09-16 | Q        | 7        | 20       | 14    | 379    | 14.0370  | 27    |
| 2012-09-23 | Q        | 8        | 20       | 13    | 383    | 14.1852  | 27    |
| 2012-10-01 | T        | 62       | 100      | 39    | 2325   | 80.1724  | 29    |
+-----+-----+-----+-----+-----+-----+-----+

```

Aby wyświetlić jedynie wybrane kolumny, należy podać ich nazwy. Z kolei w celu wyświetlenia informacji jedynie dla określonego zdarzenia wystarczy po prostu podać datę, a podczas wykonywania widoku MySQL dołączy do jego definicji odpowiednią klauzulę WHERE. Dzięki przygotowanemu widokowi można w bardzo prosty sposób pobrać żądane informacje:

```

mysql> SELECT date, category, count, average
-> FROM grade_stats WHERE date = '2012-09-16';
+-----+-----+-----+-----+
| date       | category | count | average |
+-----+-----+-----+-----+
| 2012-09-16 | Q        | 27    | 14.0370 |
+-----+-----+-----+-----+

```

Pewne widoki można uaktualniać, co oznacza możliwość wstawiania, uaktualniania i usuwania rekordów w tabelach wymienionych w definicji widoku przez wykonanie na nich odpowiednich operacji. Oto przykład:

```
mysql> CREATE TABLE t (i INT);
mysql> INSERT INTO t (i) VALUES(1),(2),(3);
mysql> CREATE VIEW v AS SELECT i FROM t;
mysql> SELECT i FROM v;
+-----+
| i     |
+-----+
| 1     |
| 2     |
| 3     |
+-----+
mysql> INSERT INTO v (i) VALUES(4);
mysql> DELETE FROM v WHERE i < 3;
mysql> SELECT i FROM v;
+-----+
| i     |
+-----+
| 3     |
| 4     |
+-----+
mysql> UPDATE v SET i = i + 1;
mysql> SELECT i FROM v;
+-----+
| i     |
+-----+
| 4     |
| 5     |
+-----+
```

Aby widok można było uaktualnić, musi być mapowany bezpośrednio na pojedynczą tabelę, musi wybierać jedynie kolumny będące prostymi odniesieniami do kolumn tabeli (nie wyrażeń), a każda operacja na rekordzie widoku musi odpowiadać operacji na pojedynczym rekordzie tabeli wymienionej w definicji widoku. Na przykład, jeżeli widok obejmuje podsumowanie wygenerowane za pomocą funkcji agregującej, każdy rekord widoku może opierać się na wielu rekordach tabeli. W takim przypadku nie można uaktualnić widoku, ponieważ nie ma możliwości jasnego określenia, który z rekordów tabeli wymienionych w definicji widoku powinien być uaktualniony.

4.2. Używanie programów składowanych

W tym podrozdziale dowiesz się, jak tworzyć i stosować wszystkie rodzaje programów składowanych: funkcje składowane, procedury składowane, wyzwalacze i zdarzenia. Zanim zapoznasz się z informacjami szczegółowymi dotyczącymi poszczególnych rodzajów programów składowanych, najpierw przekonasz się, jak tworzyć zapytania złożone.

4.2.1. Zapytania złożone i ograniczniki zapytań

Program składowany zawierający tylko pojedyncze zapytanie SQL może być zapisany bez żadnych specjalnych wymagań. Przedstawiona poniżej procedura używa zapytania SELECT do wyświetlenia nazw tabel w bazie danych sampdb:

```
CREATE PROCEDURE sampdb_tables ()

SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'sampdb' ORDER BY TABLE_NAME;
```

Jednak program składowany nie musi ograniczać się tylko do pojedynczego zapytania. Kod może składać się z wielu poleceń SQL oraz zawierać konstrukcje takie jak zmienne lokalne, zapytania warunkowe, pętle i zagnieżdżone bloki. Zapoznaj się z podrozdziałem E.2, zatytułowanym „Składnia zapytań SQL (zapytania złożone)”.

Aby utworzyć program składowany wykorzystujący wymienione powyżej funkcje, konieczne jest użycie zapytania złożonego składającego się z komponentów BEGIN i END definiujących blok, w którym można umieścić dowolną liczbę zapytań. Poniższa procedura wyświetla komunikat i wita Cię po imieniu (lub nazywa Ziemiianinem, jeśli jesteś użytkownikiem anonimowym):

```
CREATE PROCEDURE greetings ()
BEGIN
# 77 = 16 for username + 60 for hostname + 1 for '@'
DECLARE user CHAR(77) CHARACTER SET utf8;
SET user = (SELECT CURRENT_USER());
IF INSTR(user, '@') > 0 THEN
    SET user = SUBSTRING_INDEX(user, '@', 1);
END IF;
IF user = '' THEN          # Użytkownik anonimowy.
    SET user = 'Ziemiianin';
END IF;
SELECT CONCAT('Witaj, ', user, '!') AS greeting;
END;
```

W przypadku zapytań złożonych zapytania znajdujące się w bloku muszą być rozdzielone średnikami. Ponieważ średnik to również domyślny ogranicznik zapytania w programie mysql, wystąpi konflikt, jeśli spróbujesz zdefiniować programy składowane za pomocą mysql. Rozwiązaniem jest użycie zapytania DELIMITER w celu zmiany domyślnego ogranicznika zapytania w mysql na inny znak, który nie pojawia się w definicji procedury. Dzięki temu mysql nie będzie interpretować średników jako końca zapytania i przekaże serwerowi całą definicję obiektu jako pojedyncze zapytanie. Po zakończeniu definiowania programu składowanego możesz przywrócić średnik jak ogranicznik zapytania. Przedstawiony poniżej program pokazuje, jak tymczasowo zmienić ogranicznik zapytania w mysql na znak \$ na czas definiowania procedury składowanej. Dalej następuje przywrócenie ogranicznika domyślnego i wykonanie procedury:

```
mysql> DELIMITER $
mysql> CREATE PROCEDURE show_times()
-> BEGIN
->     SELECT CURRENT_TIMESTAMP AS 'Local Time';
->     SELECT UTC_TIMESTAMP AS 'UTC Time';
```

```

-> ENDS
mysql> DELIMITER ;
mysql> CALL show_times();
+-----+
| Local Time |
+-----+
| 2012-05-03 18:18:19 |
+-----+
| UTC Time   |
+-----+
| 2012-05-03 23:18:19 |
+-----+

```

Ogranicznikiem wcale nie musi być znak \$, to także nie musi być pojedynczy znak:

```

mysql> DELIMITER EOF
mysql> CREATE PROCEDURE show_times()
-> BEGIN
->   SELECT CURRENT_TIMESTAMP AS 'Local Time';
->   SELECT UTC_TIMESTAMP AS 'UTC Time';
-> END EOF
mysql> DELIMITER ;

```

Warto pamiętać o następującej zasadzie: jeśli kod programu składowanego zawiera jakiegokolwiek średniki, podczas jego definiowania powinieneś zmienić ogranicznik zapytania w `mysql`.

Zapytanie złożone nie musi być używane jedynie w przypadku skomplikowanych programów składowanych. Można z niego skorzystać także wtedy, gdy kod programu składowanego zawiera tylko jedno zapytanie lub nawet żadnego:

```

CREATE PROCEDURE do_little ()
BEGIN
  DO SLEEP(1);
END;

CREATE PROCEDURE do_nothing ()
BEGIN
END;

```

W celu zachowania spójności możesz preferować użycie słów kluczowych `BEGIN` i `END` wokół każdej definicji programu składowanego, nawet jeśli nie jest to ściśle wymagane.

4.2.2. Procedury i funkcje składowane

Funkcje składowane obliczają i zwracają wartość do późniejszego użycia w wyrażeniach, podobnie jak funkcje wbudowane, takie jak `COS()` lub `HEX()`. Procedury składowane wykonywane jako samodzielne operacje są wywoływane za pomocą zapytania `CALL` zamiast w wyrażeniach. Użyj procedury, jeśli musisz jedynie przeprowadzić obliczenia w celu wygenerowania efektu lub akcji bez zwracania wartości lub jeśli obliczenie powoduje wygenerowanie zbioru wynikowego (takiej możliwości funkcja nie oferuje).

W celu utworzenia procedury lub funkcji składowanej użyj zapytania `CREATE FUNCTION` lub `CREATE PROCEDURE`. Ich podstawowa składnia przedstawia się następująco:

```
CREATE FUNCTION nazwa_funkcji ([lista_parametrów])
  RETURNS typ
  zapytania_procedury

CREATE PROCEDURE nazwa_procedury ([lista_parametrów])
  zapytania_procedury
```

W przedstawionym poniżej przykładzie utworzono funkcję pobierającą parametr w postaci liczby całkowitej przedstawiającej rok. (Używam prefiksu `p_` do odróżnienia nazw parametrów od innych nazw, na przykład tabel lub kolumn). Funkcja wykorzystuje podzapytanie w celu ustalenia liczby prezydentów urodzonych we wskazanym roku i zwraca tę liczbę:

```
mysql> DELIMITER $
mysql> CREATE FUNCTION count_born_in_year(p_year INT)
-> RETURNS INT
-> READS SQL DATA
-> BEGIN
-> RETURN (SELECT COUNT(*) FROM president WHERE YEAR(birth) = p_year);
-> END$
mysql> DELIMITER ;
```

Funkcja zawiera klauzulę `RETURNS`, wskazującą typ danych wartości zwrotnej, a także kod obliczający tę wartość. W kodzie funkcji musi znajdować się przynajmniej jedno zapytanie `RETURN` zwracające wartość komponentowi wywołującemu funkcję. Dzięki zdefiniowaniu obliczeń jako funkcji zyskujesz prosty sposób ich przeprowadzenia bez konieczności podawania logiki obliczeń za każdym razem. Same obliczenia mogą być wywołane jak funkcja wbudowana:

```
mysql> SELECT count_born_in_year(1908);
+-----+
| count_born_in_year(1908) |
+-----+
| 1 |
+-----+
mysql> SELECT count_born_in_year(1913);
+-----+
| count_born_in_year(1913) |
+-----+
| 2 |
+-----+
```

Wymienione zapytania wywołują funkcję, ale funkcje składowane mogą być używane w wyrażeniach o dowolnym poziomie skomplikowania.

Funkcja nie może zwracać wielu wartości. Możesz utworzyć wiele funkcji i wywoływać je z poziomu pojedynczego zapytania. Inne podejście polega na utworzeniu procedury składowanej „zwracającej” wartości za pomocą parametrów `OUT`. Tego rodzaju procedura powinna obliczać żądane wartości i przypisywać je parametrom, które następnie będą dostępne dla wywołującego po zakończeniu działania procedury. Więcej informacji

szczegółowych na ten temat znajdziesz w podpunkcie 4.2.2.2, zatytułowanym „Typy parametrów procedury składowanej”.

Jeżeli zdefiniujesz funkcję składowaną o takiej samej nazwie jak funkcja wbudowana, podczas wywoływania funkcji musisz podać nazwę bazy danych. Na przykład, po zdefiniowaniu w bazie danych sampdb funkcji składowanej o nazwie PI() wywołujesz ją, podając nazwę sampdb.PI() i wyraźnie wskazując wywołanie funkcji składowanej, a nie wbudowanej. Lepszym rozwiązaniem jest unikanie niejednoznaczności i nieużywanie dla funkcji składowanych nazw funkcji wbudowanych.

Procedura składowana jest podobna do funkcji składowanej, ale pozbawiona wartości zwrotnej. Dlatego też nie ma klauzuli RETURN i żadnych poleceń RETURN. Przedstawiona poniżej prosta procedura jest podobna do funkcji count_born_in_year(), ale zamiast obliczoną liczbę przekazywać jako wartość zwrótną, wyświetla zbiór wynikowy zawierający rekord informacji dla każdego prezydenta urodzonego we wskazanym roku:

```
mysql> delimiter $
mysql> CREATE PROCEDURE show_born_in_year(p_year INT)
-> BEGIN
->     SELECT first_name, last_name, birth, death
->     FROM president
->     WHERE YEAR(birth) = p_year;
-> END$
mysql> delimiter ;
```

W przeciwieństwie do funkcji składowanej, procedura składowana nie jest używana w wyrażeniach. Zamiast tego jest wywoływana za pomocą zapytania CALL:

```
mysql> CALL show_born_in_year(1908);
+-----+-----+-----+-----+
| first_name | last_name | birth      | death      |
+-----+-----+-----+-----+
| Lyndon B.  | Johnson  | 1908-08-27 | 1973-01-22 |
+-----+-----+-----+-----+
mysql> CALL show_born_in_year(1913);
+-----+-----+-----+-----+
| first_name | last_name | birth      | death      |
+-----+-----+-----+-----+
| Richard M. | Nixon    | 1913-01-09 | 1994-04-22 |
| Gerald R.  | Ford     | 1913-07-14 | 2006-12-26 |
+-----+-----+-----+-----+
```

W powyższym przykładzie kod procedury wykonuje zapytanie SELECT. Jak pokazano w przykładzie, zbiór wynikowy zapytania nie jest zwracany w postaci wartości procedury, ale wysyłany klientowi. Procedura może wygenerować wiele rekordów wynikowych, z których każdy może być przekazany klientowi.

Przedstawione dotąd przykłady jedynie pobierały informacje, ale procedury składowane mogą również modyfikować tabele. Kolejna procedura, o nazwie update_expiration(), pobiera identyfikator członka Ligi Historycznej i uaktualnia odpowiedni rekord wskazaną datą wygaśnięcia członkostwa:

```
CREATE PROCEDURE update_expiration (p_id INT UNSIGNED, p_date DATE)
BEGIN
```

```
UPDATE member SET expiration = p_date WHERE member_id = p_id;
END;
```

Poniższe wywołania `update_expiration()` powodują przedłużenie członkostwa o jeden rok od bieżącego dnia oraz „dożywotnio” (wartość NULL oznacza „członkostwo dożywotnie”).

```
mysql> CALL update_expiration(61, CURDATE() + INTERVAL 1 YEAR);
mysql> CALL update_expiration(87, NULL);
```

Funkcje składowane mają pewne ograniczenia: nie mogą modyfikować tabeli odczytywanej lub zapisywanej przez zapytanie, które wywołało daną funkcję składowaną. Procedury składowane normalnie nie mają takiego ograniczenia, ale wspomniane ograniczenie ma zastosowanie, jeśli procedura składowana będzie wywołana w funkcji składowanej. Na przykład, nie można wywołać `update_expiration()` z poziomu funkcji składowanej, która została z kolei wywołana w zapytaniu pobierającym informacje z tabeli `member`.

4.2.2.1. Uprawnienia procedur i funkcji składowanych

Procedury i funkcje składowane należą do bazy danych. W celu utworzenia procedury lub funkcji składowanej konieczne jest posiadanie uprawnienia `CREATE ROUTINE` dla tej bazy danych. Domyślnie, podczas tworzenia procedury składowanej serwer automatycznie nadaje uprawnienia `EXECUTE` i `ALTER ROUTINE`, jeśli ich nie masz, aby w ten sposób umożliwić wykonanie lub usunięcie procedury. Jeżeli usuniesz procedurę składowaną, serwer automatycznie odbiera Ci wymienione uprawnienia. Aby wyłączyć automatyczne nadawanie i odbieranie uprawnień, należy przypisać wartość 0 zmiennej systemowej o nazwie `automatic_sp_privileges`.

Jeśli serwer ma włączone binarne rejestrowanie dziennika zdarzeń, wtedy funkcje składowane muszą spełnić dodatkowe warunki zapewniające bezpieczeństwo binarnemu dziennikowi zdarzeń podczas tworzenia kopii zapasowej i przeprowadzania replikacji. Wspomniane warunki polegają na ograniczeniu zezwalającemu na tworzenie jedynie funkcji niedeterministycznych lub modyfikujących dane. (Jeśli funkcja generuje różne wyniki dla konkretnych wartości danych wejściowych, przywrócenie danych za pomocą ponownego wykonania zapytań z binarnego dziennika zdarzeń może zakończyć się niepowodzeniem. W takim przypadku nie ma możliwości przywrócenia oryginalnych danych, a funkcja może inaczej się replikować w serwerach głównym i podległym). Wspomniane warunki (omówione poniżej) mają również zastosowanie w trakcie tworzenia wyzwalaczy:

- Jeżeli zmienna systemowa `log_bin_trust_function_creators` nie jest włączona, konieczne jest posiadanie uprawnienia `SUPER` w celu utworzenia funkcji składowanych. Ponadto, każda tworzona funkcja powinna być deterministyczna i nie powinna modyfikować danych. W tym celu należy ją zadeklarować z jednym z wymienionych atrybutów: `DETERMINISTIC`, `NO SQL` lub `READS SQL DATA`. Na przykład:

```
CREATE FUNCTION half (p_value DOUBLE)
RETURNS DOUBLE
DETERMINISTIC
```

```
BEGIN
    RETURN p_value / 2;
END;
```

- Jeżeli zmienna systemowa `log_bin_trust_function_creators` jest włączona, wtedy nie są wymuszane żadne ograniczenia. Najczęściej zdarza się to w sytuacjach, gdy masz pewność, że żaden użytkownik serwera MySQL nie będzie definiował niebezpiecznych funkcji przechowywanych.

4.2.2.2. Typy parametrów procedury składowanej

Każdy parametr procedury składowanej może mieć jeden z trzech typów. Dla parametru IN wywołujący przekazuje wartość do procedury. Ta wartość może być zmodyfikowana wewnątrz procedury, ale wprowadzone zmiany nie są widoczne dla wywołującego po zakończeniu działania procedury. Parametr OUT jest przeciwieństwem IN. Procedura przypisuje wartość parametrowi, który jest dostępny dla wywołującego po zakończeniu działania procedury. Z kolei parametr INOUT pozwala wywołującemu na przekazanie i otrzymanie wartości.

Aby wyraźnie wskazać typ parametru, należy użyć słowa kluczowego IN, OUT lub INOUT przed nazwą parametru na liście parametrów. Jeżeli nie będzie podany żaden typ, to domyślnie stosowany jest IN.

W celu użycia parametru OUT lub INOUT konieczne jest podanie nazwy zmiennej podczas wywołania procedury. Procedura może ustawić wartość parametru, a odpowiadająca mu zmienna będzie miała tę wartość po zakończeniu działania procedury. Typy parametrów OUT i INOUT są szczególnie użyteczne, gdy zachodzi potrzeba przeprowadzenia obliczeń generujących wiele wartości. (Funkcja przechowywana zwraca tylko pojedynczą wartość i nie może być używana w tego rodzaju sytuacjach).

Przedstawiona poniżej procedura prezentuje użycie parametrów OUT. Zadaniem procedury jest zliczenie uczniów obu płci w tabeli `student` i zwrot liczb w parametrach, aby wywołujący miał do nich dostęp:

```
CREATE PROCEDURE count_students_by_sex (OUT p_male INT, OUT p_female INT)
BEGIN
    SET p_male = (SELECT COUNT(*) FROM student WHERE sex = 'M');
    SET p_female = (SELECT COUNT(*) FROM student WHERE sex = 'F');
END;
```

Aby wywołać procedurę, po prostu podaj dla parametrów zmienne zdefiniowane przez użytkownika. Obliczone wartości procedura przypisze parametrom. Po zakończeniu działania procedury wspomniane zmienne będą miały odpowiednie wartości:

```
mysql> CALL count_students_by_sex(@male_count, @female_count);
mysql> SELECT @male_count, @female_count;
+-----+-----+
| @male_count | @female_count |
+-----+-----+
|          16 |           15 |
+-----+-----+
```

Czy jesteś ograniczony do przekazywania jako parametrów jedynie zmiennych zdefiniowanych przez użytkownika? Nie. Jeśli wywołasz procedurę `count_students_by_sex()` z poziomu innego programu składowanego, zmienne lokalne lub parametry zdefiniowane w tym programie mogą być przekazane jako parametry `count_students_by_sex()`.

Bardziej zaawansowane przykłady mogą wymagać parametrów dodatkowych. Na przykład, możesz utworzyć procedurę posiadającą parametr `IN` wskazujący identyfikator testu lub sprawdzianu w tabeli `score`. Następnie procedura może obliczać dane statystyczne (odchylenie standardowe, zakres itd.) dla powiązanych ze sobą wyników i za pomocą parametrów `OUT` przekazywać obliczone wartości wywołującemu.

Słowa kluczowe `IN`, `OUT` i `INOUT` nie mają zastosowania w funkcjach składowanych, wyzwalaczach i zdarzeniach. W przypadku funkcji składowanych wszystkie parametry działają jak parametry typu `IN`. Z kolei wyzwalacze i zdarzenia w ogóle nie mają parametrów.

4.2.3. Wyzwalacze

Wyzwalacz to program składowany powiązany z określoną tabelą i zdefiniowany do uaktywnienia się podczas wykonywania zapytania `INSERT`, `DELETE` lub `UPDATE` dla tej tabeli. Wyzwalacz może być aktywowany przed przetworzeniem lub po przetworzeniu każdego rekordu przez zapytanie. Definicja wyzwalacza zawiera zapytania wykonywane po jego aktywowaniu.

Wyzwalacz charakteryzuje się następującymi zaletami:

- Wyzwalacz może analizować lub zmieniać nowe wartości danych przeznaczone do wstawienia lub użycia podczas uaktualnienia rekordu. W ten sposób można zapewnić zachowanie spójności danych, na przykład przez sprawdzenie, czy wartość procentowa została podana w zakresie od 0 do 100. Istnieje również możliwość przeprowadzenia filtrowania danych wejściowych.
- Wyzwalacz może dostarczać wartości domyślne dla kolumny na podstawie wyrażenia, nawet dla typów kolumn, które mogą być zdefiniowane jedynie z wartością domyślną w postaci stałej.
- Wyzwalacz może przeanalizować bieżącą zawartość rekordu przed jego usunięciem lub uaktualnieniem. Taka możliwość może być wykorzystana na przykład do zarejestrowania zmian wprowadzanych w istniejących rekordach.

W celu utworzenia wyzwalacza używane jest zapytanie `CREATE TRIGGER`. Definicja wskazuje określony typ zapytania powodującego aktywację wyzwalacza (`INSERT`, `DELETE` lub `UPDATE`) oraz moment aktywacji: przed modyfikacją lub po modyfikacji rekordów. Podstawowa składnia tworzenia wyzwalacza przedstawia się następująco:

```
CREATE TRIGGER nazwa_wyzwalacza      # Nazwa wyzwalacza.
{BEFORE | AFTER}                    # Moment aktywacji wyzwalacza.
{INSERT | UPDATE | DELETE}           # Typ zapytania aktywującego wyzwalacz.
ON nazwa_tabeli                     # Tabela powiązana z wyzwalaczem.
FOR EACH ROW zapytania_wyzwalacza    # Operacje wykonywane przez wyzwalacz.
```

W powyższym zapytaniu *nazwa_wyzwalacza* wskazuje nazwę nadaną wyzwalaczowi, natomiast *nazwa_tabeli* to powiązana z nim tabela. W przypadku nazwy wyzwalacza warto stosować konwencję polegającą na tym, aby nazwa jasno wskazywała przeznaczenie wyzwalacza i powiązaną z nim tabelę, na przykład *bi_nazwa_tabeli* lub *ai_nazwa_tabeli* dla wyzwalacza BEFORE INSERT lub AFTER INSERT działającego w tabeli o podanej nazwie.

Z kolei *zapytania_wyzwalacza* to jego kod, czyli zapytania wykonywane po aktywacji wyzwalacza. W kodzie wyzwalacza składni *NEW.nazwa_kolumny* można użyć w celu odwołania się do kolumn w nowym rekordzie przeznaczonym do wstawienia lub uaktualnienia w wyzwalaczu INSERT lub UPDATE. Podobnie, składni *OLD.nazwa_kolumny* można użyć w celu odwołania się do kolumn w starym rekordzie przeznaczonym do usunięcia lub uaktualnienia w wyzwalaczu DELETE lub UPDATE. Aby zmienić wartość kolumny w wyzwalaczu BEFORE, zanim ta wartość będzie umieszczona w tabeli, należy użyć zapytania *SET NEW.nazwa_kolumny = wartość*.

W poniższym przykładzie pokazano wyzwalacz *bi_t* dla zapytań INSERT w tabeli *t* zawierającej kolumnę typu INT o nazwie *percent*, przeznaczoną do przechowywania wartości procentowych (od 0 do 100), i kolumnę typu DATETIME. Wyzwalacz używa BEFORE, a więc ma możliwość przeanalizowania i ewentualnie zmodyfikowania wartości danych wstawianych do kolumny.

```
mysql> CREATE TABLE t (percent INT, dt DATETIME);
mysql> DELIMITER $
mysql> CREATE TRIGGER bi_t BEFORE INSERT ON t
->   FOR EACH ROW BEGIN
->     IF NEW.percent < 0 THEN
->       SET NEW.percent = 0;
->     ELSEIF NEW.percent > 100 THEN
->       SET NEW.percent = 100;
->     END IF;
->     SET NEW.dt = CURRENT_TIMESTAMP;
->   END$
mysql> DELIMITER ;
```

Przedstawiony wyzwalacz wykonuje dwie operacje:

- W przypadku próby wstawienia wartości procentowej spoza zakresu od 0 do 100 wyzwalacz konwertuje tę wartość do najbliższego punktu końcowego.
- Wyzwalacz automatycznie dostarcza wartość CURRENT_TIMESTAMP dla kolumny typu DATETIME. W efekcie stanowi rozwiązanie ograniczenia polegającego na tym, że wartość domyślna kolumny musi być stałą i implementuje automatyczną inicjalizację kolumny typu DATETIME wartością typu TIMESTAMP. (Począwszy od wersji MySQL 5.6.5, kolumna typu DATETIME może być zainicjalizowana bieżącym znacznikiem czasu, a tym samym ta część wyzwalacza nie jest konieczna. Jednak działa zgodnie z założeniami w wersji 5.6 oraz wszystkich wcześniejszych. W ten sposób masz zapewnioną zgodność w sposób niezależny od wersji serwera).

Aby przekonać się, jak działa ten wyzwalacz, do tabeli wstaw pewne rekordy, a następnie pobierz jej zawartość:

```
mysql> INSERT INTO t (percent) VALUES(-2); DO SLEEP(2);
mysql> INSERT INTO t (percent) VALUES(30); DO SLEEP(2);
mysql> INSERT INTO t (percent) VALUES(120);
mysql> SELECT * FROM t;
```

```
+-----+-----+
| percent | dt                |
+-----+-----+
|      0  | 2012-05-19 11:14:54 |
|      30 | 2012-05-19 11:14:56 |
|     100 | 2012-05-19 11:14:58 |
+-----+-----+
```

Wyzwalacz należy do tabeli i dlatego musisz mieć uprawnienie TRIGGER dla tej tabeli, aby móc w niej tworzyć i usuwać wyzwalacze. (Jeżeli usuniesz tabelę, to MySQL usunie wszystkie powiązane z nią wyzwalacze).

Tworzenie wyzwalaczy jest objęte pewnymi ograniczeniami, podobnie jak w przypadku funkcji składowanych. Zapoznaj się z podpunktem 4.2.2.1, zatytułowanym „Uprawnienia procedur i funkcji składowanych”.

4.2.4. Zdarzenia

MySQL ma harmonogram zdarzeń pozwalający na przeprowadzanie w bazie danych operacji o określonych godzinach. Zdarzenie to program składowany powiązany z harmonogramem. Wspomniany harmonogram definiuje godzinę lub godziny wykonania zdarzenia oraz opcjonalnie moment zakończenia zdarzenia. Zdarzenia są szczególnie użyteczne podczas przeprowadzania operacji administracyjnych, takich jak okresowe uaktualnienia raportów podsumowań, usunięcie starych danych lub rotacji tabeli dzienników zdarzeń. W tym punkcie zademonstrujemy utratę ważności starych danych. Przykład pokazujący przeprowadzenie opartej na zdarzeniach rotacji tabeli dzienników zdarzeń znajdziesz w podpunkcie 12.8.7.4, zatytułowanym „Utrata ważności, czyli rotacja tabel dzienników zdarzeń”.

Harmonogram zdarzeń nie jest uruchamiany domyślnie, a więc musisz go włączyć, aby móc używać zdarzeń. W pliku opcji odczytywanym przez serwer w chwili jego uruchamiania umieść następujące wiersze:

```
[mysqld]
event_scheduler=ON
```

Aby w trakcie działania sprawdzić stan harmonogramu zdarzeń, wykonaj następujące zapytanie:

```
SHOW VARIABLES LIKE 'event_scheduler';
```

W celu zatrzymania lub uruchomienia harmonogramu zdarzeń w trakcie działania serwera należy zmienić wartość zmiennej systemowej event_scheduler (to jest zmienna globalna, więc musisz mieć uprawnienia SUPER):

```
SET GLOBAL event_scheduler = OFF;      # Lub 0.
SET GLOBAL event_scheduler = ON;       # Lub 1.
```

Jeżeli zatrzymasz harmonogram, wtedy żadne zdarzenia nie będą uruchamiane. Istnieje również możliwość pozostawienia działającego harmonogramu, ale wyłączenia poszczególnych zdarzeń. Takie rozwiązanie zostanie przedstawione w dalszej części rozdziału.

Uwaga

Jeżeli zmiennej `event_scheduler` przypiszesz wartość `DISABLED` podczas uruchamiania serwera, w trakcie działania serwera możesz tylko sprawdzić jej stan, ale już nie zmienić. Tworzenie zdarzeń będzie możliwe, ale ich wykonywanie nie.

Harmonogram zdarzeń zapisuje informacje do dziennika błędów serwera, tam znajdziesz informacje o działaniach podejmowanych przez harmonogram. W trakcie wykonywania zdarzeń informacje o nich są rejestrowane w dzienniku, podobnie jak wszelkie ewentualne błędy. Jeżeli harmonogram zdarzeń nie działa, choć powinien, sprawdź dziennik błędów w poszukiwaniu komunikatów wskazujących przyczynę takiego zachowania.

Aby utworzyć zdarzenie, trzeba użyć zapytania `CREATE EVENT`, którego składnia przedstawia się następująco:

```
CREATE EVENT nazwa_zdarzenia
ON SCHEDULE
  {AT datetime | EVERY expr okres_czasu [STARTS datetime] [ENDS datetime]}
DO zapytania_zdarzenia
```

Zdarzenie należy do bazy danych, a więc musisz mieć uprawnienia `EVENT` dla tej bazy danych, aby móc tworzyć i usuwać w niej zdarzenia.

Poniższy przykład pokazuje, jak utworzyć proste zdarzenie usuwające stare rekordy z tabeli. Przyjmujemy założenie, że masz tabelę o nazwie `web_session` przechowującą informacje o stanie dla sesji powiązanych z użytkownikami odwiedzającymi witrynę internetową. We wspomnianej tabeli znajduje się kolumna typu `DATETIME` o nazwie `last_visit`, wskazująca datę i godzinę ostatniej odsłony strony przez użytkownika. Aby uniemożliwić tabeli przechowywanie starych rekordów, konfigurujemy zdarzenie, które okresowo będzie się ich pozbywało. W celu wykonania zdarzenia co cztery godziny i usunięcia rekordów starszych niż jeden dzień musisz utworzyć następujące zdarzenie:

```
CREATE EVENT expire_web_session
ON SCHEDULE EVERY 4 HOUR
DO
  DELETE FROM web_session
  WHERE last_visit < CURRENT_TIMESTAMP - INTERVAL 1 DAY;
```

Klauzula `n okres_czasu` wskazuje, że zdarzenie będzie uruchamiane w ustalonych odstępach czasu. Wartość `okres_czasu` będzie podobna do używanej w funkcji `DATE_ADD()`, na przykład `hour`, `day` lub `month`. Po słowie kluczowym `EVERY` można podać opcjonalne klauzule `STARTS datetime` i `ENDS datetime`, wskazujące początkową i końcową datę oraz godzinę wykonania zdarzenia. Domyślnie, zdarzenie `EVERY` wykonywane jest po raz pierwszy natychmiast po jego utworzeniu i nie ma zdefiniowanej daty końcowej.

Klauzula `DO` definiuje kod zdarzenia, czyli zapytania SQL wykonywane w trakcie aktywacji zdarzenia. Podobnie jak w przypadku pozostałych typów programów składowanych, to może być zapytanie zarówno proste, jak i złożone, utworzone za pomocą słów kluczowych `BEGIN` i `END`.

W celu utworzenia zdarzenia uruchamianego jednokrotnie należy użyć typu `AT` zamiast `EVERY`. Definicja taka jak przedstawiona poniżej tworzy zdarzenie uruchamiane tylko jednokrotnie — godzinę po jego utworzeniu:

```
CREATE EVENT one_shot
ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
DO ... ;
```

Aby wyłączyć zdarzenie i zaprzestać jego wykonywania lub włączyć zdarzenie, należy użyć zapytania `ALTER EVENT`:

```
ALTER EVENT nazwa_zdarzenia DISABLE;
ALTER EVENT nazwa_zdarzenia ENABLE;
```

4.3. Zapewnienie bezpieczeństwa widokom i programom składowanym

Zdefiniowanie widoku powoduje konfigurację zapytania `SELECT` przeznaczonego do późniejszego wykonania. To samo dotyczy definicji programu składowanego: tworzony jest obiekt, który będzie wykonywany w późniejszym terminie. Aspekt „wykonywany później” tego rodzaju obiektów oznacza, że użytkownik uruchamiający obiekt może nie być użytkownikiem, który go zdefiniował. Z tego powodu rodzi się ważne pytanie: jakie środki bezpieczeństwa powinien podjąć serwer, sprawdzając uprawnienia w trakcie wykonywania wspomnianego obiektu? A dokładniej, uprawnienia którego konta powinny być wykorzystane?

Domyślnie, serwer używa konta użytkownika, który zdefiniował dany obiekt. Przyjmujemy założenie, że definiujesz procedurę składowaną `p()` uzyskującą dostęp do Twoich tabel. Jeżeli uprawnienia `EXECUTE` do `p()` dasz innemu użytkownikowi, to będzie mógł on wykonać zapytanie `CALL p()` w celu wywołania procedury składowanej i uzyskać dostęp do Twoich tabel, ponieważ wspomniana procedura składowana jest uruchamiana w ramach Twoich uprawnień. Tego rodzaju kontekst bezpieczeństwa może być dobry lub zły:

- Jest dobry w tym sensie, że pozwala na konfigurację starannie przygotowanych programów składowanych w sposób umożliwiający im kontrolowany dostęp do tabel dla użytkowników, którzy nie mogą bezpośrednio uzyskać dostępu do wspomnianych tabel.
- Jest zły, ponieważ jeśli użytkownik zdefiniuje program składowany uzyskujący dostęp do danych wrażliwych, ale o tym zapomni, inni użytkownicy uruchamiający ten obiekt będą mieli taki sam dostęp do danych jak użytkownik definiujący ten program składowany.

Aby wyraźnie zdefiniować odbiorcę dla widoku lub programu składowanego, należy w zapytaniu CREATE umieścić klauzulę `DEFINER = konto_użytkownika`. Powoduje ona, że konto wskazane w zapytaniu będzie traktowane jako definiujący i używane w celu sprawdzenia uprawnień dostępu podczas uruchomienia obiektu. Na przykład:

```
CREATE DEFINER = 'sampadm'@'localhost' PROCEDURE count_students()  
  SELECT COUNT(*) FROM student;
```

W klauzuli `DEFINER` podana wartość może być kontem użytkownika w postaci `'nazwa_użytkownika'@'nazwa_hosta'` stosowanym w zapytaniach zarządzania kontami, na przykład `CREATE USER`. (Więcej informacji na ten temat znajdziesz w podpunkcie 13.2.1.1, zatytułowanym „Określanie nazw kont”). W przypadku wymienionego formatu konieczne jest podanie obu komponentów, czyli *nazwa_użytkownika* i *nazwa_hosta*. Ewentualnie, wartością może być `CURRENT_USER` lub `CURRENT_USER()` w celu wskazania konta użytkownika wykonującego zapytanie `SELECT`. To będzie dokładnie to samo konto, które jest używane domyślnie w przypadku braku klauzuli `DEFINER`.

Jeżeli masz uprawnienie `SUPER`, jako wartość `DEFINER` możesz podać dowolną poprawną nazwę konta. Jeżeli konto o podanej nazwie nie istnieje, zostanie wyświetlone ostrzeżenie. Gdy nie masz uprawnienia `SUPER`, jako wartość klauzuli `DEFINER` możesz podać tylko własne konto użytkownika w postaci jego dosłownej nazwy lub jako `CURRENT_USER`.

Dla widoków, procedur i funkcji składowanych można użyć atrybutu `SQL SECURITY`, dającego dodatkową kontrolę nad sprawdzaniem uprawnień dostępu w trakcie wykonywania obiektu. Atrybut `SQL SECURITY` pobiera wartość `DEFINER` (uruchomienie wraz z uprawnieniami użytkownika definiującego obiekt) lub `INVOKER` (uruchomienie wraz z uprawnieniami użytkownika, który uruchamia dany obiekt).

Atrybut `SQL SECURITY INVOKER` to preferowane rozwiązanie w sytuacjach, gdy nie chcesz, aby widok, procedura lub funkcja składowana zostały uruchomione z większymi uprawnieniami niż posiadane przez użytkownika. Przedstawiony poniżej widok uzyskuje dostęp do tabeli w bazie danych `mysql`, ale jest uruchamiany wraz z uprawnieniami użytkownika. Dlatego też, jeśli użytkownik nie ma uprawnień dostępu do tabeli `mysql.user`, widok nie udzieli użytkownikowi większych uprawnień.

```
CREATE SQL SECURITY INVOKER VIEW v  
  AS SELECT CONCAT(User, '@', Host) AS Account, Password FROM mysql.user;
```

Wyzwalacze i zdarzenia są uruchamiane automatycznie przez serwer, więc koncepcja „użytkownika wywołującego” nie ma tutaj zastosowania. Dlatego też nie mają atrybutu `SQL SECURITY` i zawsze są wywoływane z uprawnieniami użytkownika, który je zdefiniował.

Jeżeli widok lub program składowany działa z uprawnieniami definiującego go użytkownika, ale konto tego użytkownika nie istnieje, wtedy wystąpi błąd.

W widoku lub programie składowanym funkcja `CURRENT_USER()` domyślnie zwraca konto odpowiadające atrybutowi `DEFINER` obiektu. W przypadku widoków, procedur i funkcji składowanych zdefiniowanych wraz z atrybutem `SQL SECURITY INVOKER`, funkcja `CURRENT_USER()` zwraca nazwę konta użytkownika wywołującego dany obiekt.

Optymalizacja zapytań

Teoria świata relacyjnej bazy danych jest zdominowana przez tabele i zbiory oraz przeprowadzane na nich operacje. Baza danych jest zbiorem tabel, która z kolei jest zbiorem rekordów i kolumn. Kiedy wykonujesz zapytanie `SELECT` w celu pobrania rekordów z tabeli, w wyniku otrzymujesz inny zbiór rekordów i kolumn, to znaczy inną tabelę. To jest abstrakcyjny zapis, który nie ma żadnego odniesienia do rzeczywistego sposobu ułożenia danych używanego przez system bazy danych do prowadzenia operacji na danych znajdujących się w tabelach. Kolejna abstrakcja w teorii polega na tym, że operacje na tabelach są przeprowadzane natychmiast, zapytania są zestawem operacji, w których nie istnieje koncepcja czasu.

Rzeczywisty świat jest inny. Systemy zarządzania bazami danych implementują abstrakcyjne koncepcje, ale w rzeczywistym sprzęcie, w którym trzeba się liczyć z fizycznymi ograniczeniami. Dlatego też wykonanie zapytań wymaga czasu, niekiedy nawet irytująco długiego. Ponieważ ludzie z natury są niecierpliwi i nie lubią czekać, musimy porzucić abstrakcyjny świat natychmiastowych operacji matematycznych na zbiorach i zacząć szukać sposobu przyśpieszenia zapytań. Na szczęście, istnieje kilka technik skracających czas wykonywania zapytań:

- Utworzenie indeksów w tabelach, aby pozwolić serwerowi bazy danych na szybsze wyszukiwanie rekordów.
- Zmiana sposobu tworzenia zapytań, aby w maksymalnym stopniu wykorzystać wspomniane indeksy. Ponadto, użycie zapytania `EXPLAIN` w celu sprawdzenia, co tak naprawdę robi serwer MySQL.
- Zmiana typów danych i sposobu przechowywania pamięci na formaty, które mogą być efektywniej przetwarzane przez serwer.

W tym rozdziale skoncentrujemy się na wymienionych powyżej kwestiach. Naszym celem jest przeprowadzenie optymalizacji wydajności systemu bazy danych, aby zapytania były przetwarzane z maksymalną szybkością. Serwer MySQL jest całkiem szybki, ale nawet najszybsza baza danych może jeszcze szybciej wykonywać zapytania, jeśli trochę jej się w tym pomoże.

5.1. Użycie indeksowania

Z wielu dostępnych technik przyspieszania zapytań indeksowanie jest najważniejsze. Ogólnie rzecz biorąc, prawidłowe użycie indeksów to rozwiązanie, którego zastosowanie powoduje największą zmianę na korzyść. Bardzo często zdarza się, że w przypadku wolno wykonywanych zapytań dodanie indeksów natychmiast rozwiązuje problem. To jednak nie zawsze się sprawdza, ponieważ nie w każdym przypadku optymalizacja jest prostym zadaniem. Niezależnie od tego, jeśli nie używasz indeksów, to w wielu przypadkach po prostu marnujesz czas, próbując za pomocą innych rozwiązań poprawić wydajność. Zastosowanie indeksowania to podstawowa technika skrócenia czasu wykonywania zapytań. Po jej użyciu możesz zastanowić się, jakie inne techniki mogą być jeszcze użyteczne.

W tym podrozdziale zostaną omówione indeksy oraz oferowane przez nie sposoby poprawienia wydajności wykonywania zapytań. Przeanalizowane będą również warunki, w których użycie indeksów może przyczynić się do spadku wydajności. Dowiesz się także, jak dobrze wybrać indeksy dla używanych tabel. W kolejnym podrozdziale przejdziemy do optymalizatora zapytań w MySQL, który próbuje odnaleźć najefektywniejszy sposób wykonywania zapytań. Oprócz umiejętności tworzenia indeksów warto zrozumieć sposób działania optymalizatora, ponieważ pozwala to na lepsze wykorzystanie przygotowanych indeksów. Pewne sposoby tworzenia zapytań w rzeczywistości uniemożliwiają użyteczne zastosowanie indeksów i dlatego warto wiedzieć, jak unikać takich sytuacji.

5.1.1. Zalety wynikające z indeksowania

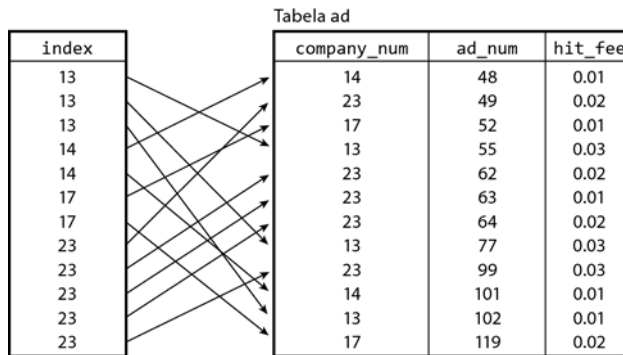
Przeanalizujemy sposób działania indeksu, rozpoczynając od pozbawionej go tabeli. Nieindeksowana tabela to po prostu nieuporządkowana kolekcja rekordów. Na rysunku 5.1 pokazano tabelę o nazwie `ad`, która została omówiona w rozdziale 1., zatytułowanym „Rozpoczęcie pracy z MySQL”. Ponieważ w wymienionej tabeli nie ma żadnych indeksów, wyszukanie rekordów dla danej firmy wymaga przeanalizowania wszystkich w celu sprawdzenia, czy zawierają dopasowanie żądanej wartości. Oznacza to pełne skanowanie tabeli, co jest operacją zarówno powolną, jak i ogromnie nieefektywną, w przypadku gdy tabela jest ogromna i zawiera jedynie kilka rekordów dopasowujących kryteria wyszukiwania.

Na rysunku 5.2. pokazano tę samą tabelę, ale zawierającą indeks obejmujący kolumnę `company_num`. Indeks zawiera wpis dla każdego rekordu tabeli, ale jego wartości są posortowane według `company_num`. Teraz, zamiast przeszukiwać rekordy tabeli w celu znalezienia dopasowań, można wykorzystać indeks. Przyjmujemy założenie, że szukamy wszystkich rekordów dla firmy o identyfikatorze 13. Rozpoczynamy skanowanie indeksu i znajdujemy trzy wartości dla tej firmy, po czym dochodzimy do firmy o identyfikatorze 14, czyli wyższym od szukanego. Wartości indeksu są posortowane, więc po odczytaniu rekordu zawierającego 14 mamy pewność, że nie znajdziemy więcej dopasowania, i możemy zakończyć operację wyszukiwania. Dlatego też efektywność indeksu polega na możliwości określenia, kiedy nie dopasujemy większej liczby rekordów, co pozwala na pominięcie analizy pozostałych. Inna korzyść wynika z możliwości użycia algorytmów pozycjonujących

Tabela ad

company_num	ad_num	hit_fee
14	48	0.01
23	49	0.02
17	52	0.01
13	55	0.03
23	62	0.02
23	63	0.01
23	64	0.02
13	77	0.03
23	99	0.03
14	101	0.01
13	102	0.01
17	119	0.02

Rysunek 5.1. Niezindeksowana tabela ad



Rysunek 5.2. Zindeksowana tabela ad

w celu odnalezienia pierwszego dopasowania bez konieczności przeprowadzania skanowania liniowego od początku indeksu (np. skanowanie binarne jest znacznie szybsze niż zwykle). W ten sposób można szybko ustalić położenie pierwszego dopasowania i zaoszczędzić dużą ilość czasu. Bazy danych stosują różne techniki szybkiego pozycjonowania wartości indeksu, ale w tym momencie konkretne techniki nie są najważniejsze. Najważniejsze jest, że wspomniane techniki działają, a indeksowanie przynosi korzyść, ponieważ pozwala na wykorzystanie tych technik.

Dlaczego nie można po prostu posortować rekordów danych i przydzielić im indeksów? Czy to nie spowoduje takiego samego zwiększenia wydajności wyszukiwania? Tak, mogłoby, jeśli tabela miałaby pojedynczy indeks. Być może do tabeli będziesz chciał dodać drugi indeks; w takim przypadku nie ma możliwości jednoczesnego posortowania rekordów danych na dwa różne sposoby. Na przykład, jeden indeks może obejmować nazwiska klientów, a drugi identyfikatory lub numery telefonów klientów. Użycie indeksów jako jednostek oddzielonych od rekordów danych rozwiązuje problem i pozwala na utworzenie wielu indeksów. Ponadto, rekordy w indeksach są z reguły mniejsze niż rekordy danych.

Po wstawieniu lub usunięciu wartości, w celu zapewnienia odpowiedniej kolejności sortowania znacznie łatwiej jest poruszać się po krótszych wartościach indeksu niż po dłuższych rekordach danych.

Konkretne szczegóły implementacji indeksu różnią się dla poszczególnych silników bazy danych MySQL. Na przykład, dla tabeli MyISAM rekordy danych tabeli są przechowywane w pliku danych, natomiast wartości indeksu w pliku indeksu. W tabeli może znajdować się więcej niż tylko jeden indeks, ale wszystkie będą przechowywane w tym samym pliku indeksu. Każdy indeks w pliku indeksu składa się z posortowanej tabeli kluczy rekordów, które są używane w celu uzyskania szybkiego dostępu do pliku danych.

W przypadku silnika InnoDB rekordy danych i wartości indeksu nie są oddzielone w ten sam sposób, choć indeksy dostępne są w postaci zbiorów posortowanych wartości. Domyślnie, silnik InnoDB używa pojedynczej przestrzeni tabel, w której zarządza danymi i indeksami dla wszystkich tabel InnoDB. Silnik InnoDB można skonfigurować w taki sposób, aby każda tabela miała własną przestrzeń tabeli. Jednak nawet w takim przypadku dane tabeli i jej indeksy będą przechowywane w tym samym pliku przestrzeni tabel.

Powyżej przedstawiono zalety indeksu w kontekście zapytań wykonywanych do pojedynczej tabeli, gdzie użycie indeksu znacząco przyspiesza wyszukanie dzięki eliminacji konieczności przeprowadzenia pełnego skanowania tabeli. Indeksy stają się jeszcze użyteczniejsze podczas wykonywania zapytań przeprowadzających złączenia wielu tabel. W przypadku pełnego skanowania pojedynczej tabeli liczba rekordów, które trzeba przeanalizować, równa się liczbie rekordów danej tabeli. Jeśli zapytanie obejmuje wiele tabel, liczba możliwych kombinacji gwałtownie rośnie, ponieważ jest iloczynem liczby rekordów w tabelach.

Przyjmujemy założenie, że mamy trzy niezindeksowane tabele t1, t2 i t3, zawierające kolumny odpowiednio i1, i2 i i3, a każda ma po tysiąc rekordów przechowujących liczby od 1 do 1000. Zapytanie wyszukujące wszystkie możliwe połączenia rekordów tabel, w których wartości są jednakowe, przedstawia się następująco:

```
SELECT t1.i1, t2.i2, t3.i3
FROM t1 INNER JOIN t2 INNER JOIN t3
WHERE t1.i1 = t2.i2 AND t2.i2 = t3.i3;
```

Wynikiem zapytania powinno być 1000 rekordów, każdy zawierający po trzy unikalne wartości. Jeżeli zapytanie będzie przetwarzane bez indeksów, wtedy nie wiadomo, jakie wartości zawierają rekordy. Dlatego też konieczne jest wypróbowanie wszystkich kombinacji w celu znalezienia tych dopasowanych do klauzuli WHERE. Liczba możliwych kombinacji wynosi $1000 \times 1000 \times 1000$ (miliard!), czyli milion razy więcej niż liczba dopasowań. Oznacza to mnożstwo zmarnowanego wysiłku. Co gorsza, jeśli tabele będą większe, ilość czasu potrzebnego na przetworzenie złączeń wymienionych tabel bez indeksów zwiększy się jeszcze bardziej, co prowadzi do jeszcze gorszej wydajności. Wykonywanie zapytań można znacznie przyspieszyć przez indeksowanie tabel, ponieważ indeksy pozwalają na przetworzenie omawianego zapytania w następujący sposób:

1. Wybór pierwszego rekordu z tabeli t1 i sprawdzenie znajdujących się w nim wartości.

2. Użycie indeksu w tabeli t2 w celu bezpośredniego przejścia do rekordu dopasowującego wartość z t1. Podobnie, użycie indeksu w tabeli t3 w celu bezpośredniego przejścia do rekordu dopasowującego wartość z t2.
3. Przejście do kolejnego rekordu tabeli t1 i powtórzenie powyższej procedury. Operacja będzie kontynuowana aż do przeanalizowania wszystkich rekordów w tabeli t1.

W takim przypadku nadal przeprowadzane jest pełne skanowanie tabeli t1, ale jednocześnie wykorzystujemy oparte na indeksie wyszukiwanie w tabelach t2 i t3 w celu bezpośredniego pobrania odpowiednich rekordów. Zapytanie jest wykonywane dosłownie milion razy szybciej niż wcześniej. Przedstawiony przykład jest nieco naciągany na potrzeby przedstawienia zalet indeksów, ale ilustrowany przez niego problem jest jak najbardziej rzeczywisty. Dodanie indeksu do tabeli niezawierającej jeszcze żadnego bardzo często przekłada się na ogromny wzrost wydajności.

MySQL używa indeksów na wiele sposobów:

- Jak przedstawiono powyżej, indeksy są używane w celu przyspieszenia wyszukiwania dopasowanych rekordów w zapytaniach posiadających klauzulę WHERE oraz rekordów dopasowujących rekordy w innych tabelach podczas przeprowadzania złączeń.
- Dla zapytań używających funkcji MIN() lub MAX() serwer MySQL może szybko i bez konieczności indeksowania wszystkich rekordów znaleźć wartość najmniejszą lub największą.
- MySQL często używa indeksów w celu efektywnego przeprowadzenia operacji sortowania i grupowania dla klauzul ORDER BY i GROUP BY.
- Czasami MySQL może używać indeksu do odczytania wszystkich informacji wymaganych przez zapytanie. Przyjmujemy założenie, że wartości są pobierane tylko z jednej, zindeksowanej kolumny liczbowej w tabeli InnoDB lub MyISAM. W takim przypadku, gdy serwer MySQL odczytuje wartość indeksu, pobiera tę samą wartość, jaką pobierałby z rekordu danych. Nie ma konieczności dwukrotnego odczytu wartości, więc rekord danych w ogóle nie został użyty.

5.1.2. Koszt indeksowania

Ogólnie rzecz biorąc, jeśli MySQL potrafi określić, jak używać indeksów w celu szybkiego wykonania zapytania, to zastosuje to rozwiązanie. Oznacza to, że w większości przypadków, jeśli nie indeksujesz tabel, to sam sobie szkodzisz. Być może sądzisz, że przedstawiam zbyt różowy obraz zalet indeksowania. Czy indeksowanie ma jakiekolwiek wady? Oczywiście, indeksowanie wiąże się z kosztem wyrażonym zarówno w czasie, jak i pamięci masowej. W praktyce zalety indeksowania znacznie przewyższają jego wady, ale mimo tego i tak warto je znać.

Po pierwsze, indeksowanie przyspiesza pobieranie danych, ale spowalnia operacje wstawiania i usuwania, jak również uaktualniania wartości zindeksowanych kolumn. Jak widzisz, indeksowanie spowalnia większość operacji obejmujących zapis danych. Wynika to z prostego faktu: tego rodzaju operacja wymaga nie tylko zapisu rekordu danych, ale także wprowadzenia zmian w indeksach. Im więcej tabel istnieje w indeksach, tym więcej zmian trzeba wprowadzić, co przekłada się na większy spadek wydajności operacji. W przypadku większości tabel liczba operacji odczytu znacznie przewyższa liczbę operacji zapisu. Jednak w sytuacji, gdy liczba operacji zapisu jest duża, koszt uaktualnienia indeksu może być znaczący. W podrozdziale 5.5, zatytułowanym „Efektywne wczytywanie danych”, dowiesz się, jak można obniżyć wspomniany koszt.

Po drugie, indeks wymaga pewnej ilości miejsca na dysku, a wiele indeksów odpowiednio większej ilości miejsca. Z tego powodu maksymalną wielkość tabeli można osiągnąć znacznie szybciej niż w przypadku tabeli pozbawionej indeksów:

- W przypadku tabeli MyISAM indeksowanie może spowodować, że plik indeksu osiągnie maksymalną dozwoloną wielkość znacznie szybciej niż plik danych.
- Wszystkie tabele InnoDB umieszczone w systemie przestrzeni tabel InnoDB konkurują ze sobą o tę samą wspólną pulę przestrzeni, a dodanie indeksów znacznie szybciej uszczupla dostępną ilość pamięci masowej. Jednak w przeciwieństwie do plików używanych w tabelach MyISAM, system przestrzeni tabel InnoDB nie jest ograniczony maksymalną wielkością pliku w danym systemie operacyjnym, ponieważ może być skonfigurowany na bazie wielu plików. Dopóki masz wolne miejsce w pamięci masowej, dopóty możesz rozszerzać przestrzeń tabel przez dodawanie do niej nowych komponentów.

Tabele InnoDB używające własnych, oddzielnych przestrzeni tabel przechowują dane i indeksy razem w tym samym pliku. Dlatego też dodanie indeksów powoduje, że plik przechowujący tabelę znacznie szybciej osiągnie maksymalną dozwoloną wielkość.

Jakie są praktyczne implikacje wymienionych czynników? Jeżeli nie potrzebujesz określonego indeksu w celu zwiększenia wydajności wykonywania zapytań, to najlepiej go nie twórz.

5.1.3. Wybór indeksów

Składnia tworzenia indeksów została omówiona w podpunkcie 2.6.4.2, zatytułowanym „Tworzenie indeksów”. Zakładam, że zapoznałeś się z wymienionym podpunktem. Jednak znajomość składni nie pomaga w wyborze odpowiedniego indeksu. Wybór najlepszego z możliwych wymaga przeanalizowania sposobu użycia tabel. W tym punkcie zostaną przedstawione pewne wskazówki pomagające w wytypowaniu kolumn do zindeksowania.

Indeksuj kolumny używane do wyszukiwania, sortowania lub grupowania, a nie kolumny wybierane dla danych wyjściowych. Innymi słowy, najlepszymi kandydatami do zindeksowania są kolumny pojawiające się w klauzulach WHERE, kolumnach stosowanych

w klauzulach złączeń lub kolumnach używanych w klauzulach ORDER BY i GROUP BY. Kolumny używane jedynie na liście danych wyjściowych podanej po słowie kluczowym SELECT nie są dobrymi kandydatami do zindeksowania:

```
SELECT
    col_a                                to nie jest dobry kandydat
FROM
    tb11 LEFT JOIN tb12
    ON tb11.col_b = tb12.col_c          dobrzy kandydaci
WHERE
    col_d = expr;                      dobry kandydat
```

Kolumny użyte do wyświetlenia oraz wymienione w klauzuli WHERE oczywiście mogą być tymi samymi kolumnami. Trzeba pamiętać o jednym: kolumna umieszczona na liście kolumn danych wyjściowych nie jest dobrym kandydatem do zindeksowania.

Kolumny stosowane w klauzulach złączeń oraz wyrażeniach w postaci *kolumna1* = *kolumna2* w klauzulach WHERE są szczególnie dobrymi kandydatami do zindeksowania. W powyższym zapytaniu col_b i col_c to przykłady tego rodzaju kolumn. Jeżeli MySQL potrafi zoptymalizować zapytanie używające złączonych kolumn, wówczas znacznie zmniejsza liczbę potencjalnych kombinacji tabela-rekord przez eliminację operacji pełnego skanowania tabel.

Rozważ liczebność kolumny. Liczebność kolumny to liczba znajdujących się w niej odmiennych wartości. Na przykład, dla kolumny zawierającej wartości 1, 3, 7, 4, 7 i 3 liczebność wynosi 4. Indeksy najlepiej sprawdzają się w przypadku kolumn charakteryzujących się wysoką liczebnością względem całkowitej liczby rekordów tabeli (to znaczy kolumn mających wiele unikalnych wartości i niewiele powtarzających się). Dla kolumny zawierającej wiele różnych wartości wieku indeks łatwo rozróżnia poszczególne rekordy. Z kolei w przypadku kolumny przechowującej informacje o płci (czyli tylko dwie różne wartości) zastosowanie indeksu nie przyniesie pożytku. Jeżeli wartości rozłożone są mniej więcej równo, otrzymasz połowę rekordów niezależnie od szukanej wartości. W takich przypadkach indeks w ogóle może być nieużywany, ponieważ ogólnie rzecz biorąc, optymalizator zapytania pomija indeks i wybiera pełne skanowanie tabeli po wykryciu wysokiego współczynnika procentowego występowania wartości w rekordach tabeli.

Aby określić liczbę unikalnych wartości w kolumnie względem liczby rekordów tabeli, należy użyć zapytania takiego jak poniższe:

```
mysql> SELECT COUNT(*), COUNT(DISTINCT state) FROM member;
+-----+-----+
| COUNT(*) | COUNT(DISTINCT state) |
+-----+-----+
|      102 |                   46 |
+-----+-----+
```

Indeksuj krótkie wartości. Gdy tylko istnieje możliwość, używaj mniejszych typów danych. Na przykład, nie używaj kolumny typu BIGINT, jeśli typ MEDIUMINT jest wystarczający do przechowywania wszystkich wymaganych wartości. Nie używaj typu CHAR(100), jeśli żadna z wartości nie ma więcej niż 25 znaków. Mniejsze wartości na wiele sposobów poprawiają przetwarzanie indeksu:

- Krótsze wartości mogą być znacznie szybciej porównywane, co się przekłada na szybsze działanie indeksu.
- Mniejsze wartości oznaczają także mniejsze indeksy wymagające mniejszej liczby dyskowych operacji wejścia-wyjścia.
- W przypadku krótszych wartości kluczy bloki indeksu w buforze kluczy mogą przechowywać więcej wartości. Serwer MySQL będzie mógł jednorazowo umieścić w pamięci większą liczbę kluczy, co zwiększa prawdopodobieństwo wyszukania wartości klucza bez konieczności wczytywania z dysku dodatkowych bloków indeksu.

Dla silnika InnoDB używającego indeksów klastrowanych szczególnie korzystne jest zapewnienie jak najkrótszych kluczy podstawowych. Indeks klastrowany przechowuje rekordy danych wraz z wartościami klucza podstawowego (to znaczy klastrowane). Pozostałe indeksy są drugorzędne, przechowują wartości klucza podstawowego wraz z wartościami indeksu drugorzędnego. Wyszukiwanie w indeksie drugorzędnym odbywa się przez wartość klucza podstawowego, która jest używana do zlokalizowania rekordu danych. Implikacja polega na tym, że wartości klucza podstawowego są powielane w każdym indeksie drugorzędnym. Dlatego też im dłuższe wartości kluczy podstawowych, tym większa ilość dodatkowej pamięci masowej jest wymagana dla każdego indeksu drugorzędnego.

Indeksuj prefiksy wartości ciągu tekstowego. W celu zindeksowania kolumny ciągu tekstowego podaj długość prefiksu, gdy ma to sens. Na przykład, jeśli masz kolumnę typu CHAR(200), to nie indeksuj całej kolumny, gdy większość wartości jest unikalnych w pierwszych dziesięciu lub dwudziestu znakach. Zindeksowanie jedynie pierwszych dziesięciu lub dwudziestu znaków pozwoli na oszczędzenie dużej ilości miejsca i prawdopodobnie przyspieszy również wykonywanie zapytań. (Indeksowanie krótszych wartości daje omówione wcześniej zalety związane z wydajnością i zmniejszeniem dyskowych operacji wejścia-wyjścia). Jak zawsze, warto zachować zdrowy rozsądek. Zindeksowanie jedynie pierwszego znaku w kolumnie prawdopodobnie nie będzie użyteczne, ponieważ powstały w ten sposób indeks nie będzie zawierał zbyt wielu unikalnych wartości.

Istnieje możliwość indeksowania prefiksów kolumn typu CHAR, VARCHAR, BINARY, VARBINARY, TEXT i BLOB za pomocą składni przedstawionej w podpunkcie 2.6.4.2, zatytułowanym „Tworzenie indeksów”.

Wykorzystaj zalety prefiksów po lewej stronie. Podczas tworzenia indeksu złożonego obejmującego n kolumn tak naprawdę tworzysz n indeksów, które mogą być używane przez MySQL. Indeks złożony działa jako kilka indeksów, ponieważ dowolny zbiór kolumn po lewej stronie indeksu może być używany w celu dopasowania rekordów. Wspomniany zbiór jest nazywany „lewym prefiksem”. (To jest zupełnie co innego niż operacja indeksowania prefiksu kolumny, która powoduje utworzenie indeksu opartego na pierwszych n znakach lub bajtach wartości kolumny).

Przyjmujemy założenie, że mamy tabelę wraz z indeksem złożonym opartym na kolumnach o nazwie country, state i city. Rekordy w indeksie są sortowane w kolejności country/state/city, a więc automatycznie są sortowane także w kolejności country/state

i country. Oznacza to, że MySQL może wykorzystać zalety indeksu nawet wtedy, gdy w zapytaniu użyjesz jedynie wartości country lub wartości country i state. Dlatego też indeks można wykorzystać do wyszukiwania następujących kombinacji kolumn:

```
country, state, city
country, state
country
```

Podczas operacji wyszukiwania MySQL nie używa indeksu pozbawionego lewego prefiksu, na przykład jeśli wyszukujesz wartości kolumny state lub city. W trakcie wyszukiwania wartości kolumny country i city (kolumny 1. i 3. w indeksie) indeks nie może być użyty dla wspomnianej kombinacji wartości, choć MySQL potrafi zawęzić wyszukiwanie, używając indeksu znajdującego rekordy dopasowujące wartość kolumny country.

Nie przesadzaj z indeksami. Nie indeksuj wszystkiego w zasięgu wzroku na podstawie założenia „im więcej, tym lepiej”. Każdy dodatkowy indeks oznacza zwiększenie ilości wymaganej pamięci masowej dla tabeli i negatywnie wpływa na operacje zapisu, o czym już wcześniej wspomniano. Indeksy muszą być uaktualniane i czasem również reorganizowane po modyfikacji tabeli — im większa liczba indeksów, tym dłużej to trwa. Indeks, który jest używany rzadko lub w ogóle, niepotrzebnie spowalnia modyfikacje tabeli. Ponadto, MySQL uwzględnia indeksy podczas generowania planu wykonywania zapytania pobierającego dane. Utworzenie dodatkowych indeksów zwiększa ilość pracy wykonywanej przez optymalizator zapytań. Może się również zdarzyć (to wcale nie jest takie nieprawdopodobne), że w przypadku dostępności wielu indeksów MySQL nie wybierze najlepszego. Aby pomóc optymalizatorowi zapytań, zachowaj tylko te indeksy, które są potrzebne.

Jeżeli rozważasz dodanie indeksu do już zindeksowanej tabeli, sprawdź, czy ewentualny nowy indeks nie jest lewym prefiksem już istniejącego. Jeżeli tak jest, to nie dodawaj nowego indeksu, ponieważ tak naprawdę już go masz. Na przykład, jeśli masz indeks obejmujący kolumny country, state i city, to nie ma sensu dodawanie indeksu obejmującego kolumnę country. Wyjątkiem jest indeks typu FULLTEXT, ponieważ wymaga on oddzielnego indeksu dla każdego zestawu kolumn, które mają być przeszukiwane.

Dopasuj typ indeksu do operacji porównania. Kiedy tworzysz indeks, większość silników bazy danych pozwala na wybór używanej implementacji. Na przykład, InnoDB zawsze używa indeksów typu B-tree. MyISAM używa indeksów typu B-tree, za wyjątkiem przestrzennych typów danych, dla których używane są indeksy typu R-tree. Z kolei silnik MEMORY domyślnie używa indeksów typu hash, choć obsługuje również indeksy B-tree i pozwala na wybór typu indeksu. Aby wybrać typ indeksu, przeanalizuj rodzaje operacji porównań, jakie mają być przeprowadzane na zindeksowanej kolumnie:

- Dla indeksu typu hash funkcja hash będzie zastosowana względem każdej wartości indeksu. Wynikowe wartości hash są przechowywane w indeksie i używane podczas operacji wyszukiwania. (Funkcja hash implementuje algorytm generujący różne wartości hash dla różnych wartości danych wejściowych. Zaletą użycia wartości hash jest ich znacznie efektywniejsze porównywanie niż wartości początkowych). Indeksy typu hash są bardzo szybkie podczas porównań dokładnego dopasowywania

przeprowadzanych za pomocą operatorów `=` i `<=>`. Natomiast słabo spisują się w trakcie wyszukiwania zakresu wartości, na przykład w wyrażeniach takich jak poniższe:

```
id < 30
weight BETWEEN 100 AND 150
```

- Indeksy typu B-tree mogą być używane do efektywnego przeprowadzania operacji porównywania dokładnego lub opartego na zakresie, wykonywanego za pomocą operatorów `<`, `<=`, `=`, `>=`, `>`, `<>`, `!=` i `BETWEEN`. Indeksy B-tree mogą być również używane w dopasowaniach wzorca `LIKE`, o ile wzorec rozpoczyna się dosłownym ciągiem tekstowym, a nie znakiem wieloznacznym.

Jeżeli tabela `MEMORY` jest używana jedynie do operacji wyszukiwania dokładnych wartości, indeks typu hash jest dobrym wyborem. To domyślny typ indeksu dla tabel `MEMORY`, więc nie trzeba podejmować żadnych specjalnych kroków. Jeżeli zwykle przeprowadzane są porównania oparte na zakresie, należy użyć indeksu B-tree. Aby wskazać typ indeksu, trzeba dodać słowa kluczowe `USING BTREE` do definicji indeksu, na przykład:

```
CREATE TABLE lookup
(
    id      INT NOT NULL,
    name    CHAR(20),
    PRIMARY KEY (id) USING BTREE
) ENGINE=MEMORY;
```

Jeżeli rodzaje przeprowadzanych operacji wyszukiwania dają odpowiednie podstawy, pojedyncza tabela `MEMORY` może zawierać indeksy typu B-tree i hash, nawet w tej samej kolumnie.

Pewne typy operacji porównania nie używają indeksów. Jeżeli przeprowadzasz porównanie jedynie przez przekazanie wartości kolumny do funkcji takiej jak `STRCMP()`, zindeksowanie kolumny nie przynosi pożytku. Serwer musi obliczyć wartość funkcji dla każdego rekordu, co uniemożliwia użycie indeksu kolumny.

Użyj dziennika zdarzeń wolno wykonywanych zapytań do ustalenia, które zapytania są wykonywane niezwykle wolno. Wspomniany dziennik zdarzeń może pomóc w wyszukaniu zapytań, w przypadku których użycie indeksu może zwiększyć wydajność ich wykonywania. (Ogólne omówienie dzienników zdarzeń serwera MySQL znajdziesz w podrozdziale 12.8, zatytułowanym „Dzienniki zdarzeń serwera”). Dziennik zdarzeń wolno wykonywanych zapytań ma postać zwykłego tekstu, a więc jego treść można przejrzeć za pomocą dowolnego programu wyświetlającego zawartość pliku. Ewentualnie użyj narzędzia `mysql_dumps_low` do wygenerowania podsumowania wspomnianego dziennika. Jeżeli pewne zapytanie często powtarza się w dzienniku, prawdopodobnie jest przygotowane nieoptymalnie, a jego modyfikacja może spowodować, że będzie wykonywane szybciej. Podczas przeglądania dziennika zdarzeń wolno wykonywanych zapytań pamiętaj, że „wolno” jest mierzone w czasie rzeczywistym, więc w trakcie dużego obciążenia serwera we wspomnianym dzienniku zapisuje on więcej zapytań niż w chwilach mniejszego obciążenia.

5.2. Optymalizator zapytań MySQL

Po otrzymaniu zapytania do wykonania serwer MySQL przystępuje do jego analizy w celu ustalenia, jakie optymalizacje mogą przyspieszyć wykonanie danego zapytania. W tym podrozdziale przekonasz się, jak działa optymalizator zapytań. Więcej informacji szczegółowych na temat pomiarów w zakresie optymalizacji przeprowadzanych przez MySQL znajdziesz w podręczniku użytkownika MySQL.

Optymalizator zapytań w MySQL oczywiście wykorzystuje zalety indeksów, ale używa także innych informacji. Na przykład, po wykonaniu poniższego zapytania MySQL wykona je bardzo szybko, niezależnie od wielkości tabeli:

```
SELECT * FROM nazwa_tabeli WHERE FALSE;
```

W omawianym przypadku MySQL sprawdza klauzulę WHERE, wykrywa, że żaden rekord nie spełni warunków zapytania, a więc w ogóle nie przystępuje do przeszukiwania tabeli. Możesz się o tym przekonać, wykonując zapytanie EXPLAIN, które nakazuje MySQL wyświetlenie informacji o sposobie wykonania zapytania SELECT bez jego faktycznego wykonania. (Począwszy od MySQL 5.6.3, zapytania EXPLAIN można również używać wraz z DELETE, INSERT, REPLACE i UPDATE).

Aby użyć zapytania EXPLAIN, wystarczy po prostu umieścić słowo kluczowe EXPLAIN przed zapytaniem SELECT:

```
mysql> EXPLAIN SELECT * FROM nazwa_tabeli WHERE FALSE\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
          table: NULL
           type: NULL
possible_keys: NULL
           key: NULL
        key_len: NULL
           ref: NULL
          rows: NULL
     Extra: Impossible WHERE
```

Normalnie zapytanie EXPLAIN wyświetla więcej informacji, niż przedstawiono powyżej, i na dodatek są one znacznie dokładniejsze niż widoczne w przykładzie wartości NULL. Wyświetlane normalnie dane zawierają informacje między innymi o indeksach używanych do przeskanowania tabel, typach złączeń oraz oszacowaną liczbę rekordów, które będą musiały być przeanalizowane w każdej tabeli. Przykłady znajdziesz w punkcie 5.2.2, zatytułowanym „Użycie zapytania EXPLAIN do sprawdzenia operacji optymalizatora”.

W pewnych przypadkach zapytanie EXPLAIN może wykonać część zapytania, na przykład jeśli zawiera podzapytania w klauzuli FROM — zapytanie EXPLAIN musi wykonać podzapytanie, aby poznać jego wartość zwrótną, zanim przystąpi do analizy głównego zapytania SELECT. (Począwszy od MySQL 5.6.3, to zachowanie dłużej nie występuje).

5.2.1. Jak działa optymalizator zapytań?

Optymalizator zapytań ma wiele celów, ale podstawowym jest maksymalne wykorzystanie indeksów, gdy tylko jest to możliwe, i użycie jak najbardziej restrykcyjnego indeksu w celu eliminacji maksymalnej liczby rekordów z puli przeznaczonych do przetworzenia. Ostatnia część poprzedniego zdania może wydawać się dziwna. Przecież celem użytkownika wykonującego zapytanie SELECT jest *wyszukanie* rekordów, a nie ich odrzucanie. Optymalizator próbuje *odrzuć* rekordy, ponieważ im szybciej wyeliminuje pewne rekordy, tym szybciej znajdzie rekordy dopasowane do kryteriów wyszukiwania. Zapytania można przetwarzać znacznie szybciej, jeśli najpierw zostaną przeprowadzone najbardziej restrykcyjne testy. Przyjmujemy założenie, że zapytanie sprawdza dwie kolumny, z których każda jest zindeksowana:

```
SELECT col1 FROM mytable
WHERE col1 = 'dowolna wartość' AND col2 = 'inna dowolna wartość';
```

Przyjmujemy również założenie, że operacja sprawdzenia kolumny col1 dopasowała 900 rekordów, natomiast kolumny col2 dopasowała 300 rekordów, a oba połączone testy dopasowują 30 rekordów. Sprawdzenie kolumny col1 jako pierwszej oznacza konieczność przeanalizowania 900 rekordów w celu znalezienia 30 dopasowanych do wartości kolumny col2. To oznacza 870 operacji sprawdzenia zakończonych niepowodzeniem. Z kolei sprawdzenie kolumny col2 jako pierwszej w celu wyszukania 30 wartości dopasowanych do col1 oznacza jedynie 270 testów zakończonych niepowodzeniem, a więc mniej obliczeń i mniej dyskowych operacji wejścia-wyjścia. Dlatego też optymalizator zdecyduje o sprawdzeniu najpierw kolumny col2, ponieważ oznacza to mniejszą ilość ogólnej pracy do wykonania.

Aby pomóc optymalizatorowi zapytań wykorzystać zalety indeksów, pamiętaj o przedstawionych poniżej wskazówkach.

Przeanalizuj table. To powoduje wygenerowanie danych statystycznych dotyczących rozkładu wartości klucza, co pomaga optymalizatorowi w lepszym oszacowaniu efektywności indeksu. Domyślnie, podczas porównywania wartości zindeksowanych kolumn ze stałą optymalizator przyjmuje założenie, że wartości kluczy są rozłożone równomiernie w indeksie. Optymalizator przeprowadza także szybkie sprawdzenie indeksu w celu oszacowania liczby rekordów, jakie będą musiały być użyte w trakcie określania, czy dla danego porównania ze stałą wykorzystać indeks. W przypadku tabel InnoDB i MyISAM można za pomocą zapytania ANALYZE TABLE nakazać serwerowi przeprowadzenie analizy kluczy.

Tabela, która po wypełnieniu pozostaje statyczna, musi być przeanalizowana tylko raz po jej wczytaniu. Z kolei tabela regularnie uaktualniana powinna być okresowo ponownie analizowana (z częstotliwością odpowiadającą częstoci uaktualnień).

Użyj zapytania EXPLAIN do weryfikacji operacji optymalizatora. Zapytanie EXPLAIN może poinformować, czy w trakcie wykonywania zapytania będą użyte indeksy. Tego rodzaju informacja jest użyteczna, gdy próbujesz utworzyć zapytanie na różne sposoby lub kiedy sprawdzasz, czy dodanie indeksu faktycznie ma wpływ na wydajność wykonywania zapytania.

Udziel optymalizatorowi wskazówek lub pomiń, gdy zajdzie potrzeba. Istnieje możliwość użycia słów kluczowych `FORCE INDEX`, `USE INDEX` lub `IGNORE INDEX` po nazwie tabeli na liście tabel do złączenia, aby wskazać serwerowi preferowane indeksy. Zapoznaj się z opisem zapytania `SELECT` w dodatku E, zatytułowanym „Przewodnik po składni SQL”.

Można również użyć słowa kluczowego `STRAIGHT_JOIN` w celu wymuszenia na optymalizatorze użycia tabel we wskazanej kolejności. Normalnie optymalizator MySQL samodzielnie ustala kolejność skanowania tabel pozwalającą na szybkie pobranie rekordów. Jednak czasami nie dokonuje on optymalnych wyborów. Jeżeli wykryjesz tego rodzaju sytuację, możesz nadpisać kolejność wybraną przez optymalizator, używając słowa kluczowego `STRAIGHT_JOIN`. Złączenie przeprowadzane przez `STRAIGHT_JOIN` wymusza, aby tabele były złączane w kolejności, w jakiej je wymieniono w klauzuli `FROM`.

Jeżeli zdecydujesz się na tego rodzaju rozwiązanie, spróbuj najpierw zastosować kolejność tabel nakładającą najbardziej restrykcyjny wybór rekordów. Jako pierwsze powinny być wymienione tabele, z których pobierana jest najmniejsza liczba rekordów. Zapytanie jest wykonywane lepiej, jeśli na początku zawęży się liczbę potencjalnych rekordów.

W zapytaniu `SELECT` słowo kluczowe `STRAIGHT_JOIN` można podać w dwóch miejscach. Pierwsze, między słowem kluczowym `SELECT` i listą tabel — w tym przypadku ma efekt globalny, wpływając na wszystkie złączenia w zapytaniu. Drugie znajduje się w klauzuli `FROM`. Przedstawione poniżej zapytania działają w taki sam sposób:

```
SELECT STRAIGHT_JOIN ... FROM t1 INNER JOIN t2 INNER JOIN t3 ... ;
SELECT ... FROM t1 STRAIGHT_JOIN t2 STRAIGHT_JOIN t3 ... ;
```

Upewnij się o wypróbowaniu wariantu zapytania wraz ze słowem kluczowym `STRAIGHT_JOIN` oraz bez niego. Być może serwer MySQL ma dobry powód nieużywania indeksów, o których sądzisz, że są najlepszym wyborem, i wtedy słowo kluczowe `STRAIGHT_JOIN` prawdopodobnie nie pomoże. Za pomocą zapytania `EXPLAIN` sprawdź plan wykonywania i przekonaj się, jak MySQL obsługuje poszczególne zapytania.

Porównuj kolumny o takim samym typie danych. Podczas porównywania zindeksowanych kolumn identyczne typy danych oferują lepszą wydajność niż różne. Na przykład, typ `INT` różni się od `BIGINT`, a więc porównania `INT/INT` lub `BIGINT/BIGINT` są szybsze niż `INT/BIGINT`. Typ `CHAR(10)` jest uznawany za taki sam jak `VARCHAR(10)`, ale inny niż `CHAR(12)` lub `VARCHAR(12)`. Jeżeli często porównywane kolumny są różnych typów danych, rozważ użycie zapytania `ALTER TABLE` do zmiany jednego z nich, aby typy stały się identyczne.

Zindeksowane kolumny powinny być samodzielne w wyrażeniach porównania.

Jeżeli kolumna jest używana w wywołaniu funkcji lub jako część bardziej skomplikowanego składnika w wyrażeniu arytmetycznym, wtedy MySQL nie może wykorzystać indeksu, ponieważ konieczne jest obliczenie wartości wyrażenia dla każdego rekordu. Czasami nie da się tego uniknąć, innym zaś razem można zmodyfikować zapytanie i wyizolować zindeksowaną kolumnę.

Poniższa klauzula `WHERE` pokazuje, jak to działa. Pod względem arytmetycznym wynik działania jest taki sam, ale przedstawione klauzule są zupełnie inaczej traktowane podczas fazy optymalizacji:

```
WHERE mycol * 2 < 4
WHERE mycol < 4 / 2
```

W pierwszym wierszu MySQL musi pobrać wartość `mycol` dla każdego rekordu, pomnożyć ją przez dwa, a następnie porównać wynik z 4. W takim przypadku nie będzie użyty żaden indeks. Każda wartość w kolumnie musi być pobrana, aby umożliwić obliczenie wyrażenia znajdującego się po lewej stronie porównania. Z kolei w drugim wierszu optymalizator upraszcza wyrażenie `4/2` do wartości 2, a następnie używa indeksu w `mycol` do szybkiego znalezienia wartości mniejszej niż 2. Dlatego też druga wersja klauzuli jest lepsza od pierwszej.

Przeanalizujmy inny przykład. Przyjmujemy założenie, że mamy zindeksowaną kolumnę typu `DATE` o nazwie `date_col`. Jeżeli wykonane zostanie zapytanie podobne do poniższego, indeks nie będzie użyty:

```
SELECT * FROM mytbl WHERE YEAR(date_col) < 1990;
```

Wyrażenie nie będzie porównywało wartości 1990 z zindeksowaną kolumną; porówna 1990 z wartością obliczoną na podstawie kolumny, a wspomniana wartość musi być obliczona dla każdego rekordu. Dlatego też indeks obejmujący kolumnę `date_col` nie będzie używany; w trakcie wykonywania zapytania konieczne będzie przeprowadzenie pełnego skanowania tabeli. Jak to naprawić? Rozwiązaniem jest użycie dosłownej daty. W takim przypadku optymalizator może użyć indeksu w kolumnie `date_col` w celu znalezienia dopasowanych wartości kolumny:

```
WHERE date_col < '1990-01-01'
```

Przyjmujemy założenie, że nie mamy konkretnej daty. Możesz być przecież zainteresowany wyszukianiem rekordów posiadających datę w pewnym zakresie dni, począwszy od dnia bieżącego. Istnieje kilka sposobów wyrażenia tego rodzaju porównania i nie wszystkie są równie efektywne. Poniżej przedstawiono kilka przykładów:

```
WHERE TO_DAYS(date_col) - TO_DAYS(CURDATE()) < granica
WHERE TO_DAYS(date_col) < granica + TO_DAYS(CURDATE())
WHERE date_col < DATE_ADD(CURDATE(), INTERVAL granica DAY)
```

W pierwszym wierszu nie będzie użyty żaden indeks, ponieważ kolumna musi być pobrana dla każdego rekordu, aby było możliwe obliczenie wartości `TO_DAYS(date_col)`. Wiersz drugi zawiera lepszą wersję. Zarówno *granica*, jak i `TO_DAYS(CURDATE())` są stałymi i dlatego prawa strona wyrażenia może być obliczona przez optymalizator jednokrotnie przed rozpoczęciem przetwarzania wyrażenia zamiast dla każdego rekordu. Jednak kolumna `date_col` nadal występuje w wywołaniu funkcji, uniemożliwiając użycie indeksu. Wersja przedstawiona w wierszu trzecim jest najlepsza. Ponownie, prawa strona wyrażenia może być obliczona jednokrotnie jako stała jeszcze przed wykonaniem zapytań, ale teraz wartość jest datą. Tę wartość można bezpośrednio porównywać z wartościami kolumny `date_col`, które nie muszą być konwertowane na postać dni. W takim przypadku można użyć indeksu.

Nie używaj znaków wieloznacznych na początku wzorca LIKE. Pewne ciągi tekstowe wyszukiwania używają klauzuli `WHERE` w następującej postaci:

```
WHERE nazwa_kolumny LIKE '%ciąg_tekstowy%'
```


To jest poprawne, jeśli chcesz znaleźć ciąg tekstowy niezależnie od miejsca jego występowania w kolumnie. Nie umieszczaj jednak znaku % po obu stronach ciągu tekstowego jedynie z przyzwyczajenia. Jeżeli naprawdę szukasz ciągu tekstowego występującego jedynie na początku kolumny, znak % pozostaw jedynie na końcu. Przyjmujemy założenie, że szukasz kolumny zawierającej nazwiska takie jak MacGregor i MacDougall, czyli rozpoczynające się przedrostkiem Mac. W takim przypadku powinieneś utworzyć następującą klauzulę WHERE:

```
WHERE last_name LIKE 'Mac%'
```

Optymalizator analizuje dosłowną część wzorca i używa indeksu w celu znalezienia rekordów dopasowanych do poniższego wyrażenia, które jest formą pozwalającą na użycie indeksu obejmującego kolumnę last_name:

```
WHERE last_name >= 'Mac' AND last_name < 'Mad'
```

Ta optymalizacja nie jest stosowana względem wzorców dopasowania używających operatora REGEXP. Wyrażenia REGEXP nigdy nie są optymalizowane.

Wykorzystaj zalety obszarów, na których optymalizator jest dopracowany. MySQL może wykonywać złączenia i podzapytania, ale obsługa podzapytań została wprowadzona później. Dlatego też optymalizator jest w pewnych przypadkach bardziej dopracowany pod kątem złączeń niż podzapytań. Wiąże się to z pewnym praktycznym skutkiem, gdy masz podzapytanie wykonywane bardzo wolno. Jak przedstawiono w punkcie 2.9.7, zatytułowanym „Przepisywanie podzapytań na postać złączeń”, pewne podzapytania można zmodyfikować, tworząc z nich logiczne odpowiedniki złączeń. Jeżeli wolno wykonywane podzapytanie można tak zmodyfikować, spróbuj je utworzyć w postaci złączenia i sprawdź, czy jest wykonywane szybciej. (Tego rodzaju strategia zapewnia najlepsze efekty w wersjach wcześniejszych niż 5.6, ponieważ programiści włożyli ogromną ilość pracy w poprawę wydajności zapytań w MySQL 5.6).

Przetestuj alternatywne formy zapytań, ale wykonuj je więcej niż tylko jeden raz.

Podczas testowania alternatywnych form zapytania (na przykład podzapytanie kontra odpowiadające mu złączenie) każde z nich wykonuj kilkukrotnie. Jeżeli zapytanie wykonasz tylko po jednym razie na dwa różne sposoby, bardzo często okaże się, że drugie zapytanie jest szybsze, ponieważ informacje pierwszego nadal pozostają w buforze i nie muszą być odczytywane z dysku. Powinieneś próbować wykonywać zapytania w standardowo obciążonym systemie, aby uniknąć wpływu innych operacji przeprowadzanych akurat w systemie.

Unikaj automatycznej konwersji typu. MySQL przeprowadza automatyczną konwersję typu, ale jeśli można uniknąć konwersji, wtedy skutkiem może być lepsza wydajność. Na przykład, jeśli num_col to kolumna typu liczb całkowitych, oba poniższe zapytania zwrócą ten sam wynik:

```
SELECT * FROM mytbl WHERE num_col = 4;  
SELECT * FROM mytbl WHERE num_col = '4';
```

Jednak w przypadku drugiego z wymienionych zapytań przeprowadzana jest konwersja typu. Operacja konwersji sama w sobie wiąże się z pewnym spadkiem wydajności na skutek konwersji liczby całkowitej i ciągu tekstowego na postać liczb zmiennoprzecinkowych

o podwójnej precyzji używanych w trakcie porównania. Znacznie poważniejszy problem polega na tym, że jeśli kolumna `num_col` jest indeksowana, porównanie obejmujące konwersję typu może uniemożliwić użycie indeksu.

Odwrotny rodzaj porównania (kolumny ciągu tekstowego z wartością liczbową) również może uniemożliwić użycie indeksu. Przyjmujemy założenie, że zapytanie zostało utworzone w następujący sposób:

```
SELECT * FROM mytbl WHERE str_col = 4;
```

W takim przypadku nie ma możliwości użycia indeksu w kolumnie `str_col`, ponieważ kolumna `str_col` może zawierać wiele różnych ciągów tekstowych, które po konwersji będą miały wartość 4 (na przykład '4', '4.0', '4th'). Jedynym sposobem przekonania się, czy wartość kwalifikuje się, jest jej odczytanie, konwersja i porównanie. Aby uniknąć tego rodzaju problemu, gdy szukana jest konkretna wartość, na przykład '4', zapytanie należy utworzyć następująco:

```
SELECT * FROM mytbl WHERE str_col = '4';
```

5.2.2. Użycie zapytania EXPLAIN do sprawdzenia operacji optymalizatora

Zapytanie EXPLAIN można wykorzystać do zapoznania się z wygenerowanymi przez optymalizator planami wykonania danego zapytania. W tym punkcie zostaną przedstawione dwa użycia zapytania EXPLAIN:

- W celu określenia, czy zapisanie zapytania w innej postaci wpływa na użycie indeksu.
- W celu sprawdzenia wpływu dodania indeksów na możliwość wygenerowania przez optymalizator efektywnych planów wykonania zapytania.

Analiza przedstawia jedynie te pola danych wyjściowych zapytania EXPLAIN, które mają znaczenie dla omawianych przykładów. Szczegółowe omówienie danych wyjściowych zapytania EXPLAIN znajdziesz w dodatku E, zatytułowanym „Przewodnik po składni SQL”. Dane wyjściowe są w postaci, w jakiej zostały wyświetlone w moim systemie. W zależności od wersji serwera i jego konfiguracji możesz otrzymać inne efekty.

W poprzednim punkcie dowiedziałeś się, że sposób utworzenia zapytania może wpływać na to, czy optymalizator będzie mógł używać indeksów. Przedstawiona tam analiza dotyczyła między innymi przykładu trzech zaprezentowanych poniżej zapytań z logicznie odpowiadającymi sobie klauzulami WHERE, z których tylko jedno pozwala na użycie indeksu.

```
WHERE TO_DAYS(date_col) - TO_DAYS(CURDATE()) < granica
WHERE TO_DAYS(date_col) < granica + TO_DAYS(CURDATE())
WHERE date_col < DATE_ADD(CURDATE(), INTERVAL granica DAY)
```

Zapytanie EXPLAIN pozwala na sprawdzenie, czy jeden sposób utworzenia zapytania jest lepszy od innych. Aby się o tym przekonać, wypróbujemy wszystkie trzy wymienione powyżej klauzule WHERE w celu wyszukania wartości kolumny `expiration` w tabeli `member`, używając *granicy* zdefiniowanej jako 30 dni. Jak już wcześniej wspomniano, tabela `member`

nie ma indeksu dla kolumny `expiration`. W pierwszej kolejności musimy więc utworzyć indeks obejmujący kolumnę `expiration`, co pozwoli nam na sprawdzenie związku między użyciem indeksu i sposobem utworzenia wyrażenia:

```
mysql> ALTER TABLE member ADD INDEX (expiration);
```

Następnie wykonujemy zapytanie `EXPLAIN` dla każdego wyrażenia i sprawdzamy, jakie plany wykonania wybrał optymalizator:

```
mysql> EXPLAIN SELECT * FROM member
-> WHERE TO_DAYS(expiration) - TO_DAYS(CURDATE()) < 30\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: member
        type: ALL
possible_keys: NULL
      key: NULL
     key_len: NULL
        ref: NULL
       rows: 102
  Extra: Using where
mysql> EXPLAIN SELECT * FROM member
-> WHERE TO_DAYS(expiration) < 30 + TO_DAYS(CURDATE())\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: member
        type: ALL
possible_keys: NULL
      key: NULL
     key_len: NULL
        ref: NULL
       rows: 102
  Extra: Using where
mysql> EXPLAIN SELECT * FROM member
-> WHERE expiration < DATE_ADD(CURDATE(), INTERVAL 30 DAY)\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: member
        type: range
possible_keys: expiration
      key: expiration
     key_len: 4
        ref: NULL
       rows: 6
  Extra: Using where
```

Wynik dwóch pierwszych zapytań pokazuje, że indeks nie został użyty. W wierszu `type` podawana jest liczba wartości, które zostaną odczytane z tabeli. Wartość `ALL` w wymienionym wierszu oznacza „wszystkie rekordy zostaną przeanalizowane”, czyli przeprowadzenie pełnego skanowania tabeli i niewykorzystanie zalet indeksu. Wartość `NULL` w każdej kolumnie powiązanej z indeksem także wskazuje na nieużywanie indeksu.

Natomiast w wynikach trzeciego zapytania widać, że klauzula `WHERE` została utworzona w sposób pozwalający optymalizatorowi na użycie indeksu w kolumnie `expiration`:

- Wiersz `type` wskazuje, że optymalizator może użyć indeksu do wyszukania określonego zakresu wartości (mniejszych niż data podana po prawej stronie wyrażenia).
- Wiersze `possible_key` i `key` wskazują, że optymalizator uznał indeks w kolumnie `expiration` za kandydata na indeks i faktycznie go wykorzystał.
- Wiersz `row` wskazuje, że optymalizator oszacował konieczność przeanalizowania sześciu rekordów w celu przetworzenia zapytania. To znacznie lepiej niż 102 rekordy w dwóch pierwszych planach wykonywania.

Drugi sposób użycia zapytania `EXPLAIN` ma na celu sprawdzenie, czy dodanie indeksów pomoże optymalizatorowi w efektywniejszym wykonaniu zapytania. Na potrzeby przykładu wykorzystamy dwie tabele, które początkowo są niezindeksowane. To będzie wystarczające do pokazania efektu utworzenia indeksów dla prostego złączenia. Te same zasady mają zastosowanie w bardziej skomplikowanych złączeniach obejmujących wiele tabel.

Przykładowe dane wyjściowe pochodzą z tabel `MyISAM`, co skutkuje dokładniejszym oszacowaniem liczby rekordów na 1000 w przypadku pełnego skanowania tabeli, a tym samym upraszcza narrację. W przypadku tabel `InnoDB` wynik jest w dużej mierze podobny, ale liczba oszacowana dla pełnego skanowania tabeli nie wynosi 1000, ponieważ `InnoDB` używa innego mechanizmu oszacowania.

Przyjmujemy założenie, że mamy dwie tabele `t1` i `t2`, każda po 1000 rekordów zawierających wartości od 1 do 1000. Analizowane przez nas zapytanie szuka rekordów, które w obu tabelach mają takie same wartości:

```
mysql> SELECT t1.i1, t2.i2 FROM t1 INNER JOIN t2
-> WHERE t1.i1 = t2.i2;
```

```
+-----+-----+
| i1 | i2 |
+-----+-----+
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
...
```

Gdy żadna z używanych w zapytaniu tabel nie ma indeksów, zapytanie `EXPLAIN` generuje następujące wyniki:

```
mysql> EXPLAIN SELECT t1.i1, t2.i2 FROM t1 INNER JOIN t2
-> WHERE t1.i1 = t2.i2\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: t1
         type: ALL
possible_keys: NULL
          key: NULL
         key_len: NULL
```

```

      ref: NULL
      rows: 1000
      Extra:
***** 2. row *****
      id: 1
      select_type: SIMPLE
      table: t2
      type: ALL
possible_keys: NULL
      key: NULL
      key_len: NULL
      ref: NULL
      rows: 1000
      Extra: Using where; Using join buffer

```

W powyższych danych wyjściowych wartość ALL w wierszu type oznacza pełne skanowanie tabeli analizujące wszystkie rekordy. Wartość NULL w wierszu possible_keys wskazuje brak kandydatów na indeksy, które mogłyby przyspieszyć zapytanie. Ze względu na brak odpowiedniego indeksu, w wierszach key, key_len i ref również mamy wartości NULL. Z kolei komunikat Using where informuje, że do identyfikacji odpowiednich rekordów zostanie użyta klauzula WHERE.

Wymienione informacje wskazują, że optymalizator nie znalazł użytecznych danych pozwalających na efektywniejsze wykonanie zapytania i będzie kontynuował pracę następująco:

- Przeprowadzi pełne skanowanie tabeli t1.
- Dla każdego rekordu tabeli t1 przeprowadzi pełne skanowanie tabeli t2 i spróbuje dopasować odpowiednie rekordy na podstawie informacji znajdujących się w klauzuli WHERE.

Wiersz rows wskazuje oszacowaną przez optymalizator liczbę rekordów, które trzeba będzie przeanalizować na każdym etapie zapytania. Dla tabeli t1 wspomniana liczba wynosi 1000, ponieważ przeprowadzone zostanie pełne skanowanie tabeli. Podobnie, dla tabeli t2 ta liczba także wynosi 1000, ale *dla każdego rekordu* w tabeli t1. Innymi słowy, oszacowana przez optymalizator liczba rekordów koniecznych do przeanalizowania w celu przetworzenia zapytania wynosi 1000 x 1000, czyli milion. To ogromne marnotrawstwo, ponieważ tak naprawdę jedynie 1000 kombinacji spełnia warunek zdefiniowany w klauzuli WHERE.

Aby zapytanie zostało wykonane bardziej efektywnie, należy dodać indeks do jednej ze złączanych tabel, a następnie ponownie wykonać zapytanie EXPLAIN:

```

mysql> ALTER TABLE t2 ADD INDEX (i2);
mysql> EXPLAIN SELECT t1.i1, t2.i2 FROM t1 INNER JOIN t2
-> WHERE t1.i1 = t2.i2\G
***** 1. row *****
      id: 1
      select_type: SIMPLE
      table: t1
      type: ALL
possible_keys: NULL
      key: NULL
      key_len: NULL
      ref: NULL

```

```

        rows: 1000
Extra:
***** 2. row *****
        id: 1
select_type: SIMPLE
        table: t2
        type: ref
possible_keys: i2
        key: i2
        key_len: 5
        ref: sampdb.t1.i1
        rows: 10
Extra: Using where; Using index

```

Powyższe wyniki pokazują ogromną poprawę. Dane wyjściowe dla tabeli t1 nadal wskazują pełne skanowanie, ale optymalizator może inaczej przetworzyć tabelę t2:

- Wartość wiersza type zmieniła się z ALL na ref i oznacza, że odniesienie do wartości (wartość z t1) może być użyte do przeprowadzenia wyszukiwania indeksu w celu znalezienia kwalifikujących się rekordów w t2.
- Wartość, do której można się odnieść, została podana w wierszu ref: sampdb.t1.i1.
- Wartość wiersza rows spadła z 1000 do 10, co oznacza, że optymalizator przewiduje konieczność przeanalizowania tylko 10 rekordów w tabeli t2 dla każdego rekordu w tabeli t1. (To jest pesymistyczne przewidywanie. W rzeczywistości tylko jeden rekord w t2 będzie dopasowany do rekordu w t1. W dalszej części rozdziału przekonasz się, jak pomóc optymalizatorowi w poprawie tych danych szacunkowych). Całkowita liczba oszacowanych rekordów wynosi 1000 x 10, czyli 10 000. To znacznie lepszy wynik niż poprzedni milion rekordów przy braku jakiegokolwiek indeksu.

Czy istnieje jakkolwiek korzyść ze indeksowania tabeli t1? Mimo wszystko w tym konkretnym złączeniu konieczne jest przeskanowanie jednej z tabel, a do tego przecież nie jest wymagany indeks. Aby się przekonać, czy dodanie indeksu będzie miało jakkolwiek efekt, zindeksuj t1.i1 i ponownie wykonaj zapytanie EXPLAIN:

```

mysql> ALTER TABLE t1 ADD INDEX (i1);
mysql> EXPLAIN SELECT t1.i1, t2.i2 FROM t1 INNER JOIN t2
-> WHERE t1.i1 = t2.i2\G
***** 1. row *****
        id: 1
select_type: SIMPLE
        table: t1
        type: index
possible_keys: i1
        key: i1
        key_len: 5
        ref: NULL
        rows: 1000
Extra: Using index
***** 2. row *****
        id: 1
select_type: SIMPLE

```

```

        table: t2
        type: ref
possible_keys: i2
  key: i2
  key_len: 5
    ref: sampdb.t1.i1
    rows: 10
  Extra: Using where; Using index

```

Dane wyjściowe są podobne do otrzymanych w poprzednio wykonanym zapytaniu EXPLAIN, ale dodanie indeksu spowodowało pewne zmiany w danych wyjściowych tabeli t1. Wiersz type ma wartość index zamiast NULL, natomiast w wierszu Extra pojawił się komunikat Using index. Wymienione zmiany wskazują, że wprawdzie przeprowadzone będzie pełne skanowanie zindeksowanych wartości, ale optymalizator może je odczytać bezpośrednio z indeksu bez konieczności odwoływania się do pliku danych. Przedstawiony wynik dotyczy tabel MyISAM, ponieważ optymalizator wie, że wszystkie potrzebne informacje może pobrać jedynie z pliku indeksu. Tego rodzaju wynik może dotyczyć także tabel InnoDB, o ile optymalizator będzie mógł użyć informacji pochodzących jedynie z indeksu, bez konieczności szukania informacji w rekordach danych.

Istnieje jeszcze jeden krok, który może pomóc optymalizatorowi w lepszym oszacowaniu kosztu: to wykonanie zapytania ANALYZE TABLE. Wymienione zapytanie powoduje, że serwer wygeneruje dane statystyczne dotyczące rozkładu wartości klucza. Analizując tabele i ponownie wykonując zapytanie EXPLAIN, można otrzymać dokładne oszacowanie liczby rekordów, które będą musiały być przetworzone:

```

mysql> ANALYZE TABLE t1, t2;
mysql> EXPLAIN SELECT t1.i1, t2.i2 FROM t1 INNER JOIN t2
-> WHERE t1.i1 = t2.i2\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: t1
        type: index
possible_keys: i1
      key: i1
      key_len: 5
        ref: NULL
        rows: 1000
      Extra: Using index
***** 2. row *****
      id: 1
  select_type: SIMPLE
        table: t2
        type: ref
possible_keys: i2
      key: i2
      key_len: 5
        ref: sampdb.t1.i1
        rows: 1
      Extra: Using where; Using index

```

W powyższym przykładzie optymalizator oszacował, że każdej wartości z tabeli t1 zostanie dopasowany tylko jeden rekord tabeli t2.

5.3. Wybór typu danych zapewniającego efektywne wykonywanie zapytań

Wybrany typ danych może na wiele sposobów mieć wpływ na wydajność wykonywania zapytania. W tym podrozdziale przedstawione zostaną wskazówki dotyczące wyboru typu danych, który może pomóc w szybszym wykonywaniu zapytań.

Używaj operacji liczbowych zamiast na ciągach tekstowych. Ogólnie rzecz biorąc, obliczenia na liczbach są przeprowadzane szybciej niż na ciągach tekstowych. Rozważ na przykład operację porównania. Liczby mogą być porównane w pojedynczej operacji, natomiast porównanie ciągów tekstowych może obejmować wiele porównań bajt po bajcie lub znak po znaku; im dłuższy ciąg tekstowy, tym więcej wspomnianych operacji.

Jeżeli kolumna ciągu tekstowego ma ograniczoną liczbę wartości, wtedy możesz użyć typu ENUM lub SET, aby wykorzystać zalety operacji liczbowych. Wymienione typy są wewnątrznie przedstawiane w postaci liczb i mogą być znacznie efektywniej przetwarzane.

Rozważ alternatywne sposoby przedstawiania ciągów tekstowych. Czasami wydajność można poprawić przez przedstawienie ciągu tekstowego w postaci liczb. Na przykład, adres IP taki jak 192.168.0.4, czyli w formacie z użyciem kropek, może być przedstawiony w postaci ciągu tekstowego. Ewentualnie adres IP można skonwertować na postać liczby całkowitej przez przechowywanie każdej z jego czterech części jako jednego bajta w czterobajtowym typie INT UNSIGNED. Przechowywanie liczb całkowitych zaoszczędzi pamięć masową, a ponadto przyspieszy wyszukiwanie. Z drugiej strony, przedstawianie adresu IP jako wartości typu INT może utrudniać przeprowadzanie operacji dopasowania wzorca, takich jak wyszukiwanie liczb w danej podsieci. To samo zadanie można prawdopodobnie wykonać za pomocą operacji bitowych. Wspomniane kwestie jasno pokazują, że nie możesz brać pod uwagę jedynie wymagań dotyczących pamięci masowej. Na podstawie przeznaczenia wartości musisz zdecydować, który rodzaj ich przedstawiania będzie najlepszy. (Niezależnie od dokonanego wyboru funkcje INET_ATON() i INET_NTOA() mogą pomóc w konwersji pomiędzy dwoma wymienionymi reprezentacjami).

Nie używaj ogromnych typów danych, gdy mniejsze w zupełności by wystarczyły. MySQL przetwarza mniejsze typy danych znacznie szybciej niż większe. Szczególnie w przypadku ciągów tekstowych czas przetwarzania jest bezpośrednio związany z długością ciągu tekstowego. Ponadto, stosowanie mniejszych typów zmniejsza wielkość tabel i przekłada się na mniejsze obciążenie związane z operacjami dyskowymi. Dla zindeksowanej kolumny krótsze wartości oferują jeszcze większy przyrost wydajności. Indeks nie tylko przyspiesza wykonywanie zapytań, ale krótsze wartości klucza mogą być przetwarzane znacznie szybciej niż dłuższe.

Dla kolumn używających typów danych o stałej wielkości wybieraj najmniejszy typ pozwalający na przechowywanie żadanego zakresu wartości. Nie używaj typu BIGINT, jeśli MEDIUMINT będzie wystarczający. Nie używaj DOUBLE, gdy wymagana jest jedynie precyzja na poziomie FLOAT. Jeżeli używasz kolumn CHAR o stałej wielkości, nie definiuj ich jako niepotrzebnie dużych. Jeśli najdłuższa wartości przechowywana w kolumnie ma 40 znaków, kolumnę zdefiniuj jako CHAR(40) zamiast na przykład CHAR(255).

Dla typów o zmiennej wielkości wybór mniejszego również pozwala na oszczędność miejsca w pamięci masowej. Do zapisania informacji o wielkości wartości typ BLOB używa 2 bajtów, natomiast typ LONGBLOB już 4. Gdy przechowujesz wartości, które zawsze są mniejsze niż 64 KB, użycie typu BLOB przynosi oszczędność 2 bajtów dla każdej wartości. (Podobne reguły mają zastosowanie względem typów TEXT).

Definiuj kolumny jako NOT NULL. Jeżeli kolumna jest zdefiniowana jako NOT NULL, MySQL może obsłużyć ją znacznie szybciej, ponieważ w trakcie przetwarzania zapytania nie trzeba sprawdzać, czy wartością kolumny jest NULL. To pozwala na tworzenie prostszych zapytań, ponieważ nie trzeba sprawdzać NULL jako przypadku specjalnego, a prostsze zapytania mogą być przetwarzane znacznie szybciej.

Rozważ użycie kolumn typu ENUM. Jeżeli masz kolumnę ciągu tekstowego o niskim poziomie liczebności (zawierającą jedynie niewielką liczbę różnych wartości), rozważ jej konwersję na kolumnę typu ENUM. Wartości ENUM mogą być przetwarzane szybko, ponieważ wewnątrznie są przedstawiane w postaci wartości liczbowych.

Używaj PROCEDURE ANALYSE(). Uruchom PROCEDURE ANALYSE() i przekonaj się, jakie informacje będą wyświetlone o kolumnach w tabeli:

```
SELECT * FROM nazwa_tabeli PROCEDURE ANALYSE();
SELECT * FROM nazwa_tabeli PROCEDURE ANALYSE(16,256);
```

Jedna z kolumn danych wyjściowych sugeruje optymalny typ danych dla każdej kolumny w tabeli. Drugi przykład nakazuje zapytaniu PROCEDURE ANALYSE() niesugerowanie typów ENUM zawierających więcej niż 16 wartości lub pobierających więcej niż 256 bajtów (wymienione wartości możesz zmienić). Bez tego rodzaju ograniczenia dane wyjściowe mogą być bardzo długie; definicje ENUM są często trudne w odczycie.

Analizując dane wyjściowe PROCEDURE ANALYSE(), możesz dojść do wniosku, że warto zmodyfikować tabelę, aby wykorzystać zalety efektywniejszego typu danych. W celu zmiany typu kolumny należy użyć zapytania ALTER TABLE.

Defragmentuj table, które tego wymagają. Często modyfikowane table, zwłaszcza zawierające kolumny danych o zmiennej wielkości, ulegają fragmentacji. Wspomniana fragmentacja jest niekorzystna, ponieważ prowadzi do marnotrawstwa miejsca w blokach na dysku używanych do przechowywania tabel. Wraz z upływem czasu konieczne staje się odczytanie większej liczby bloków, aby pobrać odpowiednie rekordy, co negatywnie wpływa na wydajność. Wprawdzie dotyczy to każdej tabeli zawierającej rekordy o zmiennej wielkości, ale w szczególności kolumn typu BLOB i TEXT, ponieważ one w ogromnym stopniu mogą się różnić wielkością.

Regularnie wykonywane zapytanie OPTIMIZE TABLE pozwala na eliminację lub ograniczenie ilości zmarnowanego miejsca we fragmentowanych tabelach MyISAM lub InnoDB i chroni tabelę przed spadkiem wydajności podczas jej używania. Metodą defragmentacji sprawdzającą się w każdym silniku bazy danych jest zrzucenie zawartości tabeli za pomocą narzędzia mysql dump, a następnie jej usunięcie i ponowne utworzenie na podstawie wygenerowanego wcześniej pliku z zawartością tabeli:

```
% mysql dump nazwa_bazy_danych nazwa_tabeli > dump.sql
% mysql nazwa_bazy_danych < dump.sql
```

Pakuj dane w kolumnie BLOB lub TEXT. Użycie kolumny typu BLOB lub TEXT do przechowywania danych pakowanych lub rozpakowywanych w aplikacji pozwala na pobranie całości za pomocą jednej operacji zamiast kilku. To może być również użyteczne w przypadku wartości danych, których nie można łatwo przedstawić w standardowej strukturze tabeli lub ulegających zmianie na przestrzeni czasu. W analizie zapytania ALTER TABLE w rozdziale 2., zatytułowanym „Użycie MySQL do zarządzania danymi”, jeden z przykładów dotyczył tabeli używanej do przechowywania wartości pochodzących z pól formularza sieciowego. We wspomnianym przykładzie pokazano, jak można użyć zapytania ALTER TABLE w celu dodania kolumn do tabeli za każdym razem, gdy dodawane są nowe pytania do formularza sieciowego.

Innym sposobem rozwiązania powyższego problemu jest zastosowanie aplikacji przetwarzającej dane formularza sieciowego na postać pewnego rodzaju struktury danych, która następnie będzie wstawiana do pojedynczej kolumny typu BLOB lub TEXT. Na przykład, odpowiedzi wprowadzone w formularzu można przedstawiać za pomocą XML, a następnie ciąg tekstowy XML umieszczać w kolumnie typu TEXT. To niewątpliwie zwiększa obciążenie aplikacji działającej po stronie klienta, która musi kodować dane (i później je dekodować po pobraniu rekordów z tabeli), ale jednocześnie upraszcza strukturę tabeli i eliminuje konieczność jej zmiany po każdej zmianie formularza.

Z drugiej strony, wartości typu BLOB i TEXT powodują własne problemy, zwłaszcza w przypadku dużej liczby zapytań typu DELETE lub UPDATE. Usunięcie tego rodzaju wartości tworzy ogromne dziury w tabeli, które później będą wypełnione rekordem lub rekordami o prawdopodobnie różnych wielkościach. Przedstawione wcześniej informacje dotyczące defragmentacji sugerują, jak można sobie z tym poradzić.

Używaj syntetycznego indeksu. Czasami pomocne mogą być kolumny indeksu syntetycznego. Jedną z metod polega na utworzeniu wartości hash na podstawie innych kolumn i przechowywaniu ich w oddzielnej kolumnie. Następnie wyszukiwanie rekordów można przeprowadzać przez znalezienie odpowiednich wartości hash. Jednak ta technika jest dobra jedynie w zapytaniach dokładnego dopasowania. (Wartości hash są bezużyteczne w przypadku wyszukiwania zakresu za pomocą operatora takiego jak < lub >=). Wartości hash mogą być wygenerowane za pomocą funkcji MD5(). Inne opcje to użycie funkcji SHA2() lub CRC32(). Ewentualnie możesz obliczać własne wartości hash za pomocą logiki zdefiniowanej w aplikacji. Pamiętaj, że liczbowe wartości hash mogą być przechowywane bardzo efektywnie. Jeżeli algorytm hash może wygenerować wartości ciągu tekstowego zawierającego spacje na końcu, nie przechowuj ich w kolumnie typu danych, w którym następuje usunięcie spacji z końca ciągu tekstowego.

Indeks syntetyczny może być szczególnie użyteczny wraz z kolumnami BLOB i TEXT. Wyszukanie tego rodzaju wartości za pomocą wartości hash używanej w charakterze klucza będzie znacznie szybsze niż bezpośrednie wyszukiwanie kolumny typu BLOB lub TEXT.

Unikaj pobierania ogromnych wartości BLOB lub TEXT, o ile nie ma absolutnej konieczności. Na przykład, zapytanie SELECT * pobierające cały rekord nie jest najlepszym rozwiązaniem, o ile nie masz pewności, czy dodanie klauzuli WHERE spowoduje ograniczenie wyniku dożądanego zbioru rekordów. W przeciwnym razie przez sieć możesz pobierać

potencjalnie ogromne wartości zupełnie bez sensu. To jest kolejny przykład pokazujący, jak użyteczne mogą być identyfikatory informacji kolumn typu BLOB lub TEXT przechowywane w kolumnie indeksu syntetycznego. Wspomnianą kolumnę identyfikatorów można przeszukiwać w celu ustalenia rekordu lub rekordów koniecznych do pobrania, a następnie pobrać wartości BLOB lub TEXT z wytypowanych rekordów.

Kolumny BLOB lub TEXT umieszczaj w oddzielnej tabeli. W pewnych sytuacjach sensowne może być przeniesienie wymienionych kolumn do innej tabeli, jeśli pozwoli to na konwersję pozostałych kolumn na format rekordów o stałej wielkości. W ten sposób zmniejsza się fragmentacja kolumny głównej i można wykorzystać większą wydajność wynikającą z używania rekordów o stałej wielkości. Ponadto, zyskujesz możliwość wykonywania zapytań `SELECT *` w tabeli głównej bez pobierania przez sieć ogromnych wartości typu BLOB lub TEXT.

5.4. Wybór formatu tabeli dla efektywnych zapytań

Pewne silniki bazy danych implementują wiele formatów przechowywania danych, z których każdy ma określone cechy charakterystyczne. Przedstawione poniżej sugestie zawierają ogólne wskazówki dotyczące wyboru formatu, ale najlepszym rozwiązaniem jest przeprowadzenie testów i przekonanie się, czy konkretny format naprawdę przewyższa inne w danej aplikacji.

Silnik MyISAM domyślnie używa rekordów o stałej wielkości, jeśli wszystkie kolumny mają stałą wielkość, oraz rekordów o zmiennej wielkości, gdy rekord ma jakąkolwiek kolumnę o zmiennej wielkości. Masz na to wpływ, wybierając między typami danych CHAR i VARCHAR (lub BINARY i VARBINARY) dla kolumn ciągu tekstowego. Do rozważenia jest zależność przestrzeni kontra czas. W przypadku kolumn o maksymalnej długości n typ CHAR wymaga n znaków dla każdego rekordu. W przypadku typu VARCHAR przeciętnie wymagana jest połowa n , więc jeśli chcesz zaoszczędzić pamięć masową, to wybierz kolumny typu VARCHAR. Z kolei jeśli zależy Ci na szybkości i możesz pozwolić sobie na zużycie większej ilości pamięci masowej, użyj kolumn typu CHAR (o stałej wielkości), ponieważ MyISAM przetwarza rekordy o stałej wielkości znacznie szybciej niż rekordy o zmiennej wielkości. Dotyczy to przede wszystkim tabel często modyfikowanych i tym samym ulegających fragmentacji:

- W przypadku rekordów o zmiennej wielkości większa fragmentacja występuje w tabelach, w których przeprowadzana jest duża liczba operacji usunięcia lub uaktualnienia danych, i wiąże się ze zróżnicowaniem wielkości rekordów. Aby zachować dobrą wydajność takiej tabeli, należy regularnie wykonywać zapytanie `OPTIMIZE TABLE`. Wspomniany problem nie dotyczy rekordów o stałej wielkości.
- Tabele z rekordami o stałej wielkości są łatwiejsze do odtworzenia w przypadku wystąpienia awarii tabeli. Początek każdego rekordu można łatwo ustalić, ponieważ we wszystkich to położenie będzie wielokrotnością wielkości rekordu. To nie

dotyczy rekordów o zmiennej wielkości. Ponadto, wprowadzie to nie jest problem z perspektywy przetwarzania zapytania, ale przyspiesza proces naprawy tabeli.

Silnik MEMORY używa rekordów o stałej wielkości, gdzie kolumny CHAR i VARCHAR są traktowane jako CHAR, więc nie ma znaczenia, który z wymienionych typów wybierzesz.

Silnik InnoDB nie traktuje inaczej kolumn o wielkości stałej i zmiennej (wszystkie rekordy używają nagłówka zawierającego wskaźniki do wartości kolumn). Dlatego też użycie kolumny typu CHAR nie jest wewnętrznie prostsze niż kolumny typu VARCHAR. Podstawowym czynnikiem wpływającym na wydajność jest więc ilość pamięci masowej wymaganej do przechowywania rekordów. Dla kolumn o maksymalnej wielkości n typ VARCHAR wymaga mniejszej przestrzeni niż CHAR oraz minimalizuje ilość pamięci masowej i liczbę dyskowych operacji wejścia-wyjścia koniecznych do przetworzenia rekordów. Typ CHAR pobiera n znaków nawet dla wartości NULL, a więc korzyści z użycia typu VARCHAR są jeszcze większe w przypadku kolumn zawierających wiele wartości NULL.

Kiedy stworzysz tabelę InnoDB dla rekordów, wybierz format pamięci masowej, którego cechy charakterystyczne są najlepiej dopasowane do danych przechowywanych w tabeli:

- Domyślnie InnoDB używa formatu COMPACT, który jest rozsądnym wyborem ogólnego przeznaczenia.
- Dla tabel zawierających powtarzające się dane format COMPRESSED bardzo często przynosi korzyści. Tego rodzaju tabela zabiera mniejszą ilość pamięci masowej (mniejsza liczba operacji wejścia-wyjścia koniecznych do przeprowadzenia), a czas zaoszczędzony na odczycie danych będzie większy niż czas procesora wymagany do dekompresji danych. Format skompresowany nie jest użyteczny w przypadku tabel przechowujących losowo wygenerowane lub już skompresowane dane.
- Dla tabel zawierających długie wartości BLOB lub TEXT najefektywniejszym formatem rekordu jest DYNAMIC.

Aby wskazać format rekordu dla nowej tabeli InnoDB, należy użyć opcji ROW_FORMAT, na przykład:

```
CREATE TABLE t1 ( ... ) ENGINE=InnoDB ROW_FORMAT=COMPRESSED;
```

Aby sprawdzić format rekordu używany w istniejącej tabeli, należy wykonać zapytanie SHOW TABLE STATUS. Do zmiany formatu tabeli trzeba użyć zapytania ALTER TABLE:

```
ALTER TABLE t1 ROW_FORMAT=DYNAMIC;
```

Formaty COMPRESSED i DYNAMIC wymagają formatu pliku Barracuda, który z kolei wymaga odpowiedniego ustawienia zmiennych systemowych innodb_file_per_table i innodb_file_format. Zapoznaj się z podpunktem 12.5.3.1.4, zatytułowanym „Używanie oddzielnych przestrzeni tabel dla każdej tabeli InnoDB”.

5.5. Efektywne wczytywanie danych

Większość czasu będziesz prawdopodobnie spędzał na optymalizacji zapytań SELECT, ponieważ są one najczęściej wykorzystywane, a ponadto nie zawsze można znaleźć odpowiedni sposób ich optymalizacji. Z kolei wczytywanie danych do bazy danych jest proste. Wymienione poniżej strategie możesz zastosować w celu poprawienia wydajności operacji wczytania danych. Podstawowe reguły są następujące:

- Im bardziej można zredukować liczbę operacji zapisu na dysku, tym szybciej nastąpi wczytanie danych. Dlatego też jednoczesne wczytywanie dużej ilości danych jest znacznie efektywniejsze niż pojedynczych rekordów, ponieważ wczytane rekordy mogą być buforowane, a następnie zapisane na dysku na końcu operacji wstawiania danych. Jednoczesny zapis wszystkich danych na końcu operacji zamiast po każdym rekordzie znacząco zmniejsza liczbę dyskowych operacji wejścia-wyjścia.
- Wczytywanie jest szybsze, gdy tabela zawiera mniejszą liczbę indeksów. W przypadku indeksów nie tylko trzeba w tabeli wstawić rekordy z danymi, ale każdy indeks musi również zostać zmodyfikowany, aby odzwierciedlał wstawienie nowego rekordu. Z tego powodu nie należy tworzyć zbędnych indeksów i usunąć takie, jeśli istnieją.
- Krótsze zapytania SQL są szybsze niż dłuższe, ponieważ ich przetworzenie przez serwer wymaga wykonania mniejszej ilości pracy. Ponadto, są znacznie szybciej przekazywane klientowi przez sieć.

Niektóre z wymienionych powyżej czynników mogą wydawać się błahe (jeden w szczególności), ale jeśli wczytujesz duże ilości danych, nawet niewielki wzrost efektywności może oznaczać dużą różnicę. Na podstawie przedstawionych powyżej ogólnych reguł można sformułować kilka praktycznych wskazówek, których zastosowanie pomaga w szybszym wczytaniu danych.

Zapytanie `LOAD DATA` (we wszelkich postaciach) jest znacznie efektywniejsze niż `INSERT`, ponieważ wczytuje dane hurtem. Serwer musi przetworzyć i zinterpretować tylko jedno zapytanie zamiast wielu. Ponadto, indeks musi być uaktualniony i zapisany dopiero po przetworzeniu wszystkich rekordów, a nie po każdym.

Zapytanie `LOAD DATA` działa efektywniej bez atrybutu `LOCAL` niż z nim. Bez użycia wymienionego atrybutu plik musi znajdować się w serwerze, a użytkownik musi mieć uprawnienia `FILE`, ale serwer może odczytać plik bezpośrednio z dysku. W przypadku zapytania `LOAD DATA LOCAL` klient odczytuje plik, a następnie przez sieć przekazuje jego zawartość serwerowi, co jest wolniejsze.

Jeżeli musisz używać zapytania `INSERT` do wstawiania danych, spróbuj zastosować składnię pozwalającą na wstawienie wielu rekordów za pomocą pojedynczego zapytania:

```
INSERT INTO nazwa_tabeli VALUES(...),(...),... ;
```

Im więcej rekordów podasz w zapytaniu, tym lepiej. W ten sposób zredukujesz całkowitą liczbę wymaganych zapytań oraz zminimalizujesz liczbę operacji modyfikacji

indeksu. Może się wydawać, że ta rada jest sprzeczna z jedną z wcześniejszych reguł (krótsze zapytania są przetwarzane znacznie szybciej niż dłuższe). Jednak tutaj nie ma żadnej sprzeczności. Po prostu pojedyncze zapytanie INSERT wstawiające wiele rekordów i tak jest krótsze niż odpowiadający mu zbiór zapytań INSERT wstawiających po jednym rekordzie. Ponadto, wstawiające wiele rekordów zapytanie może być przetworzone przez serwer z dużo mniejszą liczbą operacji modyfikacji i zapisu indeksu.

Jeżeli do wygenerowania plików kopii zapasowej bazy danych używasz narzędzia `mysql dump`, domyślnie generuje ono zapytania INSERT wstawiające wiele rekordów. Włączona jest opcja `--opt` (optymalizacja), która z kolei włącza opcję `--extended-insert`, powodującą wygenerowanie zapytań INSERT wstawiających wiele rekordów. Ponadto, włączone zostają także inne opcje powodujące znacznie efektywniejsze przetworzenie pliku kopii zapasowej podczas jego ponownego wczytywania.

W narzędziu `mysql dump` staraj się unikać opcji `--complete-insert`. Powoduje ona wygenerowanie zapytań INSERT wstawiających pojedyncze rekordy. Przetwarzanie tego rodzaju zapytań trwa dłużej i wymaga większej ilości pracy do wykonania niż w przypadku zapytań INSERT wstawiających wiele rekordów.

Jeżeli musisz użyć wielu zapytań INSERT, grupuj je, jeśli istnieje możliwość, aby zmniejszyć liczbę operacji zapisu na dysku. W przypadku transakcyjnych silników bazy danych odbywa się to przez wykonanie zapytań INSERT w ramach pojedynczej transakcji zamiast w trybie automatycznego zatwierdzania:

```
START TRANSACTION;
INSERT INTO nazwa_tabeli... ;
INSERT INTO nazwa_tabeli... ;
INSERT INTO nazwa_tabeli... ;
COMMIT;
```

W przypadku nietransakcyjnych silników bazy danych należy nałożyć blokadę na tabelę, a następnie wykonać zapytania INSERT, gdy tabela pozostaje zablokowana:

```
LOCK TABLES nazwa_tabeli WRITE;
INSERT INTO nazwa_tabeli... ;
INSERT INTO nazwa_tabeli... ;
INSERT INTO nazwa_tabeli... ;
UNLOCK TABLES;
```

Niezależnie od zastosowanego rozwiązania otrzymasz tę samą korzyść: MySQL zapisze indeks tylko jeden raz po wykonaniu wszystkich zapytań zamiast po każdym zapytaniu INSERT. Ta druga sytuacja występuje w przypadku używania trybu automatycznego zatwierdzania lub jeśli nie nałożono blokady na tabelę.

Dla tabel MyISAM inną strategią pozwalającą na redukcję operacji zapisu indeksu jest użycie opcji `DELAY_KEY_WRITE`. Użycie wymienionej opcji powoduje, że rekordy danych są, jak dotąd, zapisywane w pliku danych natychmiast, ale bufor kluczy jedynie co pewien czas, a nie po każdej operacji wstawiania. Aby użyć opóźnionego zapisu indeksu w serwerze, należy uruchomić `mysqld` wraz ze zmienną systemową `delay_key_write`, której przypisano wartość `ALL`. W takim przypadku serwer opóźnia zapis bloków indeksu tabeli aż do chwili, gdy konieczne będzie opróżnienie wszystkich bloków, aby zrobić miejsce dla innych wartości indeksu; gdy zostanie wykonane zapytanie `FLUSH TABLES` lub gdy tabela zostanie zamknięta.

Jeżeli używasz opóźnionego zapisu kluczy dla tabeli MyISAM, wtedy awaria serwera i jego zamknięcie może spowodować utratę wartości indeksu. To nie jest problem o znaczeniu krytycznym, ponieważ indeksy MyISAM mogą być naprawione na podstawie rekordów danych. Aby mieć gwarancję przeprowadzenia naprawy, trzeba uruchomić serwer wraz ze zmienną systemową `mysam_recover_options`, której należy przypisać wartość zawierającą opcję `FORCE`. Wspomniana opcja powoduje wymuszenie na serwerze sprawdzenia tabel MyISAM podczas ich otwierania i automatyczną naprawę, gdy istnieje potrzeba.

Dla serwera podległego replikacji zmiennej `delay_key_write` można przypisać wartość `ALL`, aby opóźnić zapis indeksu na dysku we wszystkich tabelach MyISAM niezależnie od tego, czy zostały początkowo utworzone w serwerze głównym replikacji.

Użycie kompresowanego protokołu klient-serwer pozwala na redukcję ilości danych przekazywanych przez sieć. Dla większości klientów MySQL użycie wspomnianego protokołu można nakazać za pomocą opcji `--compress` podanej w wierszu poleceń. Ogólnie rzecz biorąc, z tego rozwiązania należy korzystać jedynie w wolnych sieciach, ponieważ kompresja wymaga całkiem sporej ilości czasu procesora.

Pozwól serwerowi MySQL na wstawianie wartości domyślnych. W zapytaniu `INSERT` nie podawaj kolumn, którym i tak przypisujesz wartości domyślne. W ten sposób zapytania będą krótsze i zmniejszy się liczba znaków przekazywanych przez sieć do serwera. Ponadto, skoro zapytania będą zawierały mniej wartości, serwer będzie przeprowadzał mniejszą liczbę operacji przetwarzania i konwersji wartości.

W przypadku tabel MyISAM, gdy zachodzi potrzeba wczytania dużej ilości danych do nowej tabeli, utwórz ją bez indeksów, wczytaj dane i dopiero wtedy utwórz indeksy. Znacznie szybsze jest jednorazowe utworzenie wszystkich indeksów niż ich modyfikacja po wstawieniu każdego rekordu. Jeśli tabela ma już indeksy, wczytanie danych może odbyć się szybciej po wcześniejszym ich usunięciu lub wyłączeniu, a następnie ponownym utworzeniu lub aktywacji już po wczytaniu danych.

Aby usunąć i ponownie utworzyć indeksy, należy użyć zapytań `DROP INDEX` i `CREATE INDEX` lub powiązanych z indeksem form zapytania `ALTER TABLE`. W celu dezaktywacji i ponownej aktywacji indeksów użyj słów kluczowych `DISABLE KEYS` i `ENABLE KEYS` w zapytaniu `ALTER TABLE`, co powoduje wyłączenie i uaktualnienie wszystkich nieunikalnych indeksów w tabeli:

```
ALTER TABLE nazwa_tabeli DISABLE KEYS;  
... zapytania wczytujące zawartość tabeli ...  
ALTER TABLE nazwa_tabeli ENABLE KEYS;
```

Jeżeli używasz zapytania `LOAD DATA` w celu wczytania danych do pustej tabeli MyISAM, serwer automatycznie przeprowadza dezaktywację i aktywację indeksu. Ponadto, narzędzie `mysql dump` domyślnie dodaje zapytania `ALTER TABLE`.

W przypadku tabel InnoDB usunięcie i dodanie później indeksów drugorzędnych przyspieszy operację, więc rozważ taki krok przed rozpoczęciem wczytywania ogromnej ilości danych. To jednak nie dotyczy indeksu podstawowego (klastrowanego), nie musisz więc usuwać i później dodawać tego indeksu.

Jeżeli rozważasz zastosowanie strategii usunięcia lub dezaktywacji indeksów na czas wczytywania danych do tabeli, weź pod uwagę całą ogólną sytuację i zastanów się, czy taki krok przyniesie korzyści. W przypadku wczytywania niewielkiej ilości danych do ogromnej tabeli ponowne utworzenie indeksów będzie prawdopodobnie trwało dłużej niż sama operacja wczytywania danych bez żadnych szczególnych przygotowań.

Przedstawione wcześniej reguły dotyczące wczytywania danych mają zastosowanie także w środowiskach mieszanych, w których klienci przeprowadzają różnego rodzaju operacje. Na przykład, ogólnie rzecz biorąc, powinieneś unikać długo wykonywanych zapytań SELECT w tabelach, które ulegają częstym zmianom (to znaczy często są w nich zapisywane dane). W przeciwnym razie dojdzie do rywalizacji między operacjami odczytu i zapisu, a także spadku wydajności operacji zapisu. Dla tego typu sytuacji istnieje pewne rozwiązanie: jeśli operacje zapisu w większości składają się z zapytań SELECT, nowe rekordy wstawiaj do tabeli pomocniczej, a następnie okresowo do głównej. To nie jest odpowiednia strategia, jeśli trzeba mieć natychmiastowy dostęp do nowo wstawionych rekordów. Jednak jeśli mogą pozostać niedostępne przez krótki okres, użycie tabeli pomocniczej jest korzystne z co najmniej dwóch powodów. Po pierwsze, zmniejsza rywalizację z zapytaniami SELECT wykonywanymi w tabeli głównej, co powoduje skrócenie czasu ich wykonywania. Po drugie, wstawienie ogromnej liczby rekordów do tabeli głównej z tabeli pomocniczej wymaga mniejszej ilości czasu niż w przypadku przeprowadzania operacji wstawiania pojedynczych rekordów do tabeli głównej, ponieważ w tym pierwszym przypadku operacja hurtowego wczytywania danych jest szybka.

Użycie aplikacji to rozwiązanie, gdy masz dostęp do bazy danych MySQL za pomocą strony internetowej działającej w Twoim w serwerze WWW. W takim przypadku umieszczenie wszystkich rekordów w tabeli głównej może nie mieć najwyższego priorytetu.

5.6. Harmonogram, blokady i współbieżność

W poprzednich podrozdziałach skoncentrowaliśmy się na przyspieszeniu wykonywania poszczególnych zapytań. W tym podrozdziale spojrzymy na prowadzoną przez MySQL politykę harmonogramu oraz ogólny wpływ, jaki wywierają na współbieżność między klientami blokady nakładane na poziomie silnika bazy danych. Przedstawione tutaj informacje wykorzystaj podczas wyboru silnika bazy danych dla aplikacji. Typ zapytań dominujących w danej aplikacji może być w jednym silniku obsługiwany lepiej niż w innym. Na potrzeby zaprezentowanej tutaj analizy przyjmujemy założenie, że klient przeprowadzający operacje odczytu (zapytania SELECT) jest odczytującym. Natomiast klient przeprowadzający operacje modyfikujące tabele (zapytania DELETE, INSERT, REPLACE lub UPDATE) jest zapisującym.

Stosowaną przez MySQL politykę harmonogramu można podsumować w następujący sposób:

- Operacje zapisu mają większy priorytet niż odczytu.
- W danej chwili może być przeprowadzana tylko jedna operacja zapisu w tabeli, a żądania zapisu są przetwarzane w kolejności ich zgłaszania.
- Jednocześnie można przetwarzać wiele operacji odczytu danych z tabeli.

Silnik InnoDB implementuje przedstawioną politykę harmonogramu za pomocą blokad nakładanych na poziomie rekordu, ale InnoDB blokuje rekordy tylko w przypadku absolutnej konieczności. W wielu sytuacjach, na przykład w trakcie jedynie operacji odczytywania, InnoDB może w ogóle nie korzystać z blokad.

Silniki MyISAM, MERGE i MEMORY implementują nakładanie blokad na zupełnie innym poziomie i używają blokad tabeli. Dlatego też mają inne cechy charakterystyczne w zakresie zarządzania rywalizacją. Kiedy klient chce uzyskać dostęp do tabeli, najpierw musi być nałożona blokada na tę tabelę. Po zakończeniu przez klienta pracy z tabelą następuje zniesienie blokady. Istnieje możliwość wyraźnego nakładania i znoszenia blokad przez wykonanie zapytań LOCK TABLES i UNLOCK TABLES. Jednak menedżer blokad serwera automatycznie nakłada i znosi blokady wedle aktualnych potrzeb. Typ wymaganej blokady zależy od operacji przeprowadzanej przez klienta (odczyt czy zapis):

- Aby przeprowadzić operację zapisu w tabeli, klient musi nałożyć blokadę zapewniającą wyłączny dostęp do tabeli. W trakcie operacji tabela pozostaje w niespójnym stanie, ponieważ rekord danych jest usuwany, dodawany lub zmieniany, a wszystkie indeksy tabeli mogą wymagać uaktualnienia w celu odzwierciedlenia wprowadzonych zmian. Umożliwienie innym klientom dostępu do modyfikowanej tabeli może doprowadzić do problemów. Dlatego też poważnym błędem jest umożliwienie dwóm klientom przeprowadzania jednoczesnych operacji zapisu, ponieważ to szybko doprowadzi do uszkodzenia tabeli. Umożliwienie klientowi odczytu danych z modyfikowanej tabeli również nie jest dobrym rozwiązaniem, ponieważ może ulec zmianie aktualne położenie odczytywanych danych i otrzymany wynik będzie nieprawidłowy.
- Aby przeprowadzić operację odczytu z tabeli, klient musi nałożyć blokadę uniemożliwiającą innym klientom przeprowadzenie operacji zapisu i modyfikacji tabeli w trakcie odczytu z niej danych. Blokada nie musi gwarantować dostępu na wyłączność. Operacja odczytu nie powoduje zmiany tabeli, więc nie ma powodu, aby uniemożliwiać innym klientom uzyskanie dostępu do tabeli. Dlatego też blokada nakładana podczas odczytu zezwala innym klientom na odczyt danych z tabeli w tym samym czasie.

Stosowany przez silnik bazy danych poziom nakładania blokad ma istotny wpływ na współbieżność między klientami. Przyjmujemy założenie, że dwa klienty chcą uaktualnić rekord w tabeli. W celu przeprowadzenia operacji uaktualnienia oba klienty muszą nałożyć blokadę zapisu. Większy poziom współbieżności można uzyskać w przypadku tabeli InnoDB niż MyISAM. W tabeli InnoDB obie operacje uaktualnienia mogą przebiegać jednocześnie, o ile klienty nie uaktualniają tego samego rekordu. Natomiast w MyISAM silnik nakłada blokadę dla pierwszego klienta, co powoduje, że drugi musi poczekać aż do chwili zakończenia operacji zainicjowanej przez pierwszego klienta.

Ogólnie rzecz biorąc, nakładanie blokad na dokładniejszym poziomie zapewnia lepszą współbieżność, ponieważ większa liczba klientów może jednocześnie korzystać z tabeli, o ile używają jej różnych fragmentów. Praktyczną implikację można przedstawić następująco: poszczególne silniki bazy danych są na różnym poziomie przystosowane do obsługi różnych połączeń zapytań:

- Tabele InnoDB mogą zapewnić lepszą wydajność podczas przeprowadzania wielu operacji uaktualniania. Ponieważ blokady są nakładane na poziomie rekordu zamiast tabeli, zablokowana zostaje tylko niewielka część tabeli. W ten sposób zmniejsza się rywalizację blokad i poprawia współbieżność.
- Tabele MyISAM są niezwykle szybkie podczas pobierania danych. Jednak stosowane przez nie blokady na poziomie tabeli mogą być problemem w środowiskach, w których występują zapytania pobierania i uaktualniania, zwłaszcza gdy wykonanie zapytań pobierających dane zabiera więcej czasu. W takim przypadku operacje uaktualniania mogą być bardzo opóźnione.

W porównaniu do nakładania blokad na poziomie rekordów blokowanie całej tabeli ma przewagę w zakresie ochrony przed zakleszczeniami. W przypadku zablokowania tabeli zakleszczenie nigdy nie występuje. Analizując zapytanie, serwer może określić tabele wymagane do jego wykonania, a następnie nałożyć na nie blokady. W przypadku tabel InnoDB zakleszczenia mogą wystąpić, ponieważ na początku transakcji silnik bazy danych nie nakłada wszystkich wymaganych blokad. Zamiast tego blokady są nakładane w trakcie trwania transakcji, gdy wystąpi potrzeba. Istnieje możliwość, że dwa zapytania nałożą blokady, a następnie będą próbowały nałożyć kolejne blokady, z których każda zależy od zwolnienia już nałożonych blokad. W takim przypadku klienty wstrzymują działanie w oczekiwaniu na zwolnienie blokad. Wynikiem jest zakleszczenie i konieczność przerwania przez serwer jednej z transakcji.

Wprowadzenie do programowania MySQL

Ten rozdział przedstawia powody tworzenia własnych aplikacji opartych na MySQL zamiast używania standardowych klientów dostarczanych wraz z dystrybucją MySQL. Ponadto, zaprezentowany będzie ogólny opis interfejsów, które wykorzystamy w trzech językach programowania omówionych w kolejnych rozdziałach (C, Perl i PHP). Przeanalizowane zostaną także czynniki warte rozważenia w trakcie wyboru języka programowania dla tworzonej aplikacji.

6.1. Dlaczego tworzyć własne aplikacje MySQL?

Dystrybucje MySQL zawierają pewien zestaw programów klienckich, które pozwalają na komunikację z serwerem. Na przykład, `mysqldump` eksportuje definicje tabel i ich zawartość, `mysqladmin` pozwala na przeprowadzanie operacji administracyjnych, a `mysql` umożliwia wykonywanie dowolnych zapytań SQL.

Standardowe programy klienckie obsługują wiele z zadań najczęściej wykonywanych przez użytkowników MySQL, ale aplikacje czasami mają pewne wymagania wykraczające poza możliwości standardowych programów klienckich. Na szczęście, MySQL wyposażono w interfejs programowania aplikacji (API) pozwalający na spełnienie wszelkich specjalnych wymagań tworzonej aplikacji. Wspomniane API zapewnia dostęp do serwera MySQL oraz otwiera możliwości ograniczone jedynie przez wyobraźnię programisty.

W tym rozdziale dowiesz się, jak tworzyć aplikacje oparte na MySQL i uzyskujące dostęp do bazy danych. Aby dokładnie zrozumieć, jakie można odnieść z tego korzyści, zastanów się, co będziesz mógł zrobić za pomocą samodzielnie utworzonych aplikacji, a co wykorzystując możliwości oferowane przez standardowego klienta `mysql` i jego pozbawiony wszelkich bajerów interfejs:

- **Dostosowana do własnych potrzeb obsługa danych wejściowych.** W przypadku programu mysql wpisujesz tekst zapytań. We własnych programach możesz zaoferować użytkownikowi łatwiejsze i bardziej intuicyjne metody wprowadzania danych wejściowych. Program może nawet wyeliminować konieczność znajomości języka SQL przez użytkownika. Mało tego — użytkownik może nawet nie znać roli odgrywanej przez bazę danych w wykonaniu danego zadania. Pobieranie danych wejściowych może odbywać się za pomocą prostego interfejsu, takiego jak w wierszu poleceń, lub w bardziej zaawansowanej postaci, na przykład w formularzu wyświetlanym w środowisku graficznym. Dla większości osób znacznie łatwiejsze okazuje się podawanie parametrów wyszukiwania za pomocą formularza niż przez wykonywanie zapytań SELECT.
 - **Weryfikacja danych wejściowych użytkownika.** Można wymusić, aby pewne pola były wypełniane w określony sposób. To znacznie zwiększa bezpieczeństwo aplikacji.
 - **Dostosowane do własnych potrzeb dane wyjściowe.** Dane wyjściowe generowane przez klienta mysql są praktycznie niesformatowane: do wyboru masz dane rozdzielone tabulatorami lub wyświetlone w postaci tabeli. W celu otrzymania elegancko sformatowanych danych wyjściowych formatowanie musisz przeprowadzić samodzielnie. Formatowanie może być proste, na przykład zastąpienie wartości NULL komunikatem Brak, lub bardziej skomplikowane, na przykład wygenerowanie rachunku, w którym trzeba umieścić informacje o kliencie oraz zamówionych produktach. Tego rodzaju dokumenty bardzo łatwo wykraczają poza oferowane przez mysql możliwości w zakresie formatowania.
 - **Uniezależnienie od ograniczeń nakładanych przez naturę samego języka SQL.** W większości przypadków skrypt SQL składa się z zestawu zapytań wykonywanych po kolei od początku do końca wraz z minimalnym sprawdzaniem błędów. Po uruchomieniu w trybie wsadowym programu mysql pliku zawierającego zapytania SQL wymienione narzędzie zakończy działanie po napotkaniu pierwszego błędu; jeśli natomiast użyto opcji --force, będzie wykonywało wszystkie zapytania, niezależnie od liczby znalezionych błędów. W samodzielnie utworzonym programie można selektywnie decydować o kontynuacji lub przerwaniu wykonywania zapytań i tym samym zachować większą kontrolę nad przeprowadzanymi operacjami. Istnieje możliwość zdefiniowania zapytania wykonywanego po sukcesie lub niepowodzeniu innego zapytania lub też podejmowania decyzji o kolejnych krokach na podstawie wyniku poprzedniego zapytania.
- MySQL obsługuje programy składowane zapewniające dodatkową elastyczność na poziomie SQL dzięki konstrukcjom kontroli przebiegu działania programu i obsługi błędów. Jednak wymienione konstrukcje nie są elastyczne, jak dostarczane przez języki programowania ogólnego przeznaczenia.

Ogólnie rzecz biorąc, do wykonania zadań obejmujących relacje główny-szczegółowy i wymagających skomplikowanego formatowania danych wyjściowych wymagane jest narzędzie inne niż `mysql`. Program stanowi wówczas „spoiwo” łączące zapytania i pozwalające na użycie danych wyjściowych jednego zapytania jako danych wejściowych innego.

- **Integracja MySQL w dowolnej aplikacji.** Wiele programów zostało przygotowanych do wykorzystania możliwości bazy danych w zakresie dostarczania informacji. Interfejs programowania klienta daje szansę użycia wspomnianych możliwości. Aplikacja, która musi sprawdzić identyfikator klienta lub dostępność produktu w magazynie, może wykonać takie zadanie za pomocą szybkiego zapytania do bazy danych. Aplikacja sieciowa pozwalająca klientowi na wyszukanie wszystkich książek danego autora może je znaleźć w bazie danych, a następnie wynik przekazać do przeglądarki internetowej klienta.

W rozdziale 1., zatytułowanym „Rozpoczęcie pracy z MySQL”, wymieniono kilka zadań w kontekście przykładowej bazy danych `sampdb`, których wykonanie wymagałoby utworzenia programów prowadzących komunikację z serwerem MySQL. Niektóre z wspomnianych zadań wymieniono na poniższej liście:

- sformatowanie katalogu członków Ligi Historycznej w celu wydruku;
- umożliwienie wyświetlenia i przeszukiwania listy członków Ligi na stronie internetowej;
- wysyłanie wiadomości e-mail przypominających o konieczności odnowienia członkostwa;
- łatwe wprowadzanie ocen uczniów za pomocą przeglądarki internetowej.

Szczegółowo (do pewnego stopnia) przeanalizujemy sposób integracji w środowisku sieciowym możliwości oferowanych przez MySQL. Baza danych MySQL nie obsługuje bezpośrednio aplikacji sieciowych, ale połączenie MySQL z odpowiednimi narzędziami daje możliwość wykonywania zapytań z poziomu serwera WWW w imieniu użytkownika klienta i przekazywanie wyniku do przeglądarki internetowej użytkownika. W ten sposób dostęp do bazy danych odbywa się łatwo przez internet.

Istnieją dwie uzupełniające się nawzajem perspektywy połączenia MySQL i sieci:

- **Użycie serwera WWW w celu zapewnienia rozbudowanego dostępu do MySQL.** W takim przypadku interesuje Cię przede wszystkim baza danych i chcesz użyć sieci jako narzędzia pozwalającego na uzyskanie łatwego dostępu do danych. To jest punkt widzenia administratora MySQL. Miejsce bazy danych w tego rodzaju scenariuszu jest jasno i wyraźnie nakreślone, ponieważ stanowi ona podstawowy obiekt zainteresowania. Na przykład, możesz tworzyć strony internetowe pozwalające na wyświetlanie listy tabel w bazie danych i analizę struktury bądź zawartości każdej z nich.

- **Użycie MySQL w celu zwiększenia możliwości oferowanych przez serwer WWW.** W takim przypadku podstawowym obiektem zainteresowania jest Twoja witryna internetowa, a MySQL chcesz użyć w charakterze narzędzia zwiększającego wartość treści witryny internetowej dla jej użytkowników. To jest punkt widzenia programisty sieciowego. Na przykład, jeżeli witryna internetowa zawiera forum lub listę dyskusyjną, wtedy bazę danych można wykorzystać do przechowywania postów. Rola MySQL pozostaje tutaj bardziej subtelna: użytkownicy witryny mogą nawet nie wiedzieć, że baza danych odgrywa pewną rolę w usługach dostarczanych przez witrynę.

Wymienione perspektywy nie są wzajemnie wykluczające się. Na przykład, w scenariuszu projektu Ligi Historycznej sieć jest używana jako narzędzie pozwalające członkom Ligi na uzyskanie łatwego dostępu do zawartości katalogu członków Ligi przez jego udostępnienie w internecie. W ten sposób sieć zapewnia dostęp do bazy danych. Jednocześnie, dodanie zawartości katalogu do witryny internetowej Ligi Historycznej zwiększa wartość tej witryny dla jej użytkowników. W ten sposób użycie bazy danych spowodowało rozszerzenie usług dostarczanych przez witrynę.

Niezależnie od tego, jak postrzegasz integrację MySQL z siecią, implementacja pozostaje podobna. Projekt graficzny witryny internetowej łączy z bazą danych, używając serwera WWW w charakterze pośrednika. Serwer WWW zbiera informacje od użytkownika, przekazuje je serwerowi MySQL w postaci zapytania, a następnie pobiera wynik zapytania i zwraca go przeglądarce internetowej użytkownika w celu jego wyświetlenia.

Danych, oczywiście, nie trzeba udostępniać w sieci, ale taki krok przynosi wiele korzyści w porównaniu do uzyskania dostępu do danych za pomocą standardowych programów klienckich MySQL:

- Osoby uzyskujące dostęp do danych przez sieć mogą użyć ulubionej przeglądarki internetowej na dowolnej platformie. Nie są ograniczone do systemów, w których można uruchomić standardowe programy klienckie MySQL. Niezależnie od tego, jak szeroko są rozpowszechnione programy klienckie MySQL, przeglądarki internetowe są powszechniejsze.
- Interfejs aplikacji sieciowej może być przygotowany jako prostszy niż oferowany przez samodzielnego klienta MySQL działającego w wierszu poleceń.
- Interfejs sieciowy można dostosować do wymagań określonej aplikacji. Programy klienckie MySQL to narzędzia ogólnego przeznaczenia z niezmiennym interfejsem.
- Dynamiczne strony internetowe rozszerzają możliwości MySQL do zadań, które są trudne lub niemożliwe do wykonania za pomocą jedynie standardowych klientów MySQL. Na przykład, tego rodzaju klientów naprawdę nie możesz użyć do przygotowania aplikacji wykorzystującej koszyk na zakupy.

Dowolny język programowania może być użyty w celu utworzenia aplikacji sieciowej, ale niektóre z nich są do tego lepiej przygotowane niż inne. Powrócimy do tej kwestii w podrozdziale 6.3, zatytułowanym „Wybór API”.

6.2. API dostępne dla MySQL

W celu ułatwienia programowania MySQL oferuje bibliotekę klienta utworzoną w języku C i umożliwiającą uzyskanie dostępu do baz danych MySQL z poziomu dowolnego programu w C. Wspomniana biblioteka klienta implementuje API składające się z zestawu struktur danych i funkcji.

Oferowane przez MySQL interfejsy dla innych języków mają możliwość połączenia utworzonej w języku C biblioteki klienta z językiem procesora. Tym samym biblioteka klienta zapewnia możliwość przygotowania wiązań MySQL z innymi językami na podstawie wspomnianego API C. Ten typ interfejsu istnieje dla języków Perl, PHP, Python, Ruby, C++, Tcl i innych.

Każde wiązanie z językiem programowania definiuje własny interfejs wskazujący reguły uzyskania dostępu do MySQL. Dla MySQL jest dostępnych wiele API, ale skoncentrujemy się jedynie na trzech najpopularniejszych:

- **API biblioteki klienta utworzonej w języku C.** To jest podstawowy interfejs programowania w MySQL. To API zostało wykorzystane do implementacji standardowych klientów dostarczanych wraz z dystrybucją MySQL, takich jak `mysql`, `mysqladmin` i `mysqldump`.
- **API DBI (Database Interface) dla języka Perl.** API DBI zostało zaimplementowane w postaci modułu języka Perl działającego z innymi modułami na poziomie DBD (ang. *Database Driver*), z których każdy zapewnia dostęp do określonego serwera bazy danych. Użyty tutaj moduł DBD to jeden z oferujących obsługę MySQL. Wykorzystamy MySQL wraz z DBI do utworzenia samodzielnych skryptów wywoływanych z poziomu wiersza poleceń, jak również skryptów wywoływanych przez serwer WWW zapewniający dostęp sieciowy do MySQL.
- **API PHP.** PHP to język skryptowy zapewniający wygodny sposób osadzania konstrukcji programistycznych na stronach internetowych. Wspomniane strony są przetwarzane przez PHP w serwerze WWW przed ich wysłaniem klientowi, co pozwala skryptom na generowanie treści dynamicznej, na przykład zawierającej wynik zapytania MySQL. Podobnie jak DBI, PHP zapewnia dostęp do wielu serwerów baz danych, a nie tylko do MySQL. Posiada interfejsy zarówno przeznaczone dla konkretnych serwerów baz danych, jak i bardziej niezależne od serwera. W tej książce wykorzystamy interfejs PHP Data Objects (PDO).

W pozostałej części rozdziału znajduje się porównawczy opis trzech wymienionych API. W pierwszej kolejności poznasz ich ogólne cechy charakterystyczne, a dopiero później powody ich wyboru zamiast innych dla konkretnej aplikacji. Natomiast dokładne omówienie wymienionych powyżej interfejsów przedstawiono w trzech kolejnych rozdziałach książki.

Oczywiście, nie ma powodu, by ograniczać się tylko do pojedynczego API. Znajomość różnych API zapewnia Ci wiedzę, którą możesz wykorzystać do rozsądnego wyboru API odpowiedniego w danej sytuacji. W ogromnym projekcie, składającym się z wielu komponentów, można wykorzystać wiele API i utworzyć pewne komponenty w jednym języku, a pozostałe w innym, w zależności od tego, który język będzie najbardziej odpowiedni do realizacji konkretnego zadania.

Jeżeli potrzebujesz oprogramowania wymaganego do użycia dowolnego z wymienionych API, zajrzyj do dodatku A, zatytułowanego „Oprogramowanie wymagane do użycia tej książki”.

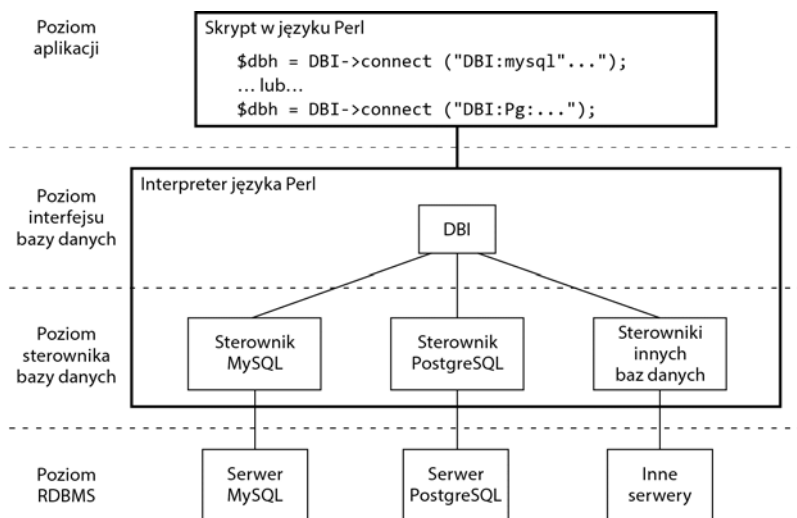
6.2.1. API C

API C jest używane w kontekście skompilowanych programów C. To jest biblioteka klienta oferująca interfejs pozwalający na komunikację z serwerem MySQL, oferujący funkcje niezbędne do nawiązania połączenia i prowadzenia komunikacji z serwerem.

Programy klienckie C dostarczone wraz z dystrybucją MySQL zostały zbudowane właśnie na bazie tego API. Utworzona w języku C biblioteka klienta służy również jako podstawa dla większości wiązań MySQL z innymi językami programowania. Na przykład, opracowany dla MySQL sterownik dla modułu Perl DBI umożliwia Perlowi współpracę z MySQL dzięki połączeniu z kodem w utworzonej w języku C bibliotece klienta.

6.2.2. API Perl DBI

API DBI jest używane do tworzenia aplikacji bazodanowych w języku skryptowym Perl. Wymienione API próbuje działać z wieloma serwerami baz danych i jednocześnie ukrywa przed twórcą skryptu szczegóły dotyczące serwera. Stało się to możliwe, ponieważ DBI używa modułów Perl współpracujących ze sobą na dwóch poziomach architektury (patrz rysunek 6.1):



Rysunek 6.1. Architektura DBI

- Poziom DBI (ang. *Database Interface*) zapewnia interfejs ogólnego przeznaczenia dla skryptów klientów. Ten poziom stanowi warstwę abstrakcji, która nie odwołuje się do konkretnego serwera bazy danych.
- Poziom DBD (ang. *Database Driver*) zapewnia obsługę różnych serwerów baz danych za pomocą sterowników przygotowanych dla konkretnych serwerów. Dla MySQL moduł na poziomie DBD nosi nazwę DBD::mysql.

Architektura DBI pozwala na tworzenie aplikacji we względnie ogólny sposób. Kiedy stworzysz skrypt DBI, wykorzystujesz standardowy zestaw wywołań pozwalających na uzyskanie dostępu do bazy danych. Warstwa DBI wywołuje odpowiedni sterownik na poziomie warstwy DBD w celu obsługi żądań, a sterownik zajmuje się obsługą komunikacji z używanym serwerem bazy danych. Dane zwrócone przez serwer są przez warstwę DBD przekazywane na poziom warstwy DBI, która z kolei udostępnia je aplikacji. Forma danych pozostaje stała niezależnie od systemu bazy danych, z którego one pochodzą.

Z punktu widzenia twórcy aplikacji, wynikiem jest interfejs ukrywający różnice między serwerami baz danych, ale jednocześnie umożliwiający obsługę wielu serwerów — tytu, dla ilu powstały odpowiednie sterowniki. Warstwa DBI zapewnia stały interfejs klienta zwiększający przenośność aplikacji, ponieważ dostęp do każdego serwera bazy danych można uzyskać w ten sam sposób.

Podczas tworzenia skryptu jedynym aspektem ściśle związanym z konkretnym serwerem jest nawiązanie połączenia z serwerem bazy danych, ponieważ wtedy trzeba wskazać sterownik używany w trakcie nawiązywania połączenia. Na przykład, w celu użycia bazy danych MySQL połączenie należy nawiązać w następujący sposób:

```
$dbh = DBI->connect ("DBI:mysql:...");
```

Aby użyć bazy danych PostgreSQL lub Oracle, połączenie trzeba nawiązać następująco:

```
$dbh = DBI->connect ("DBI:Pg:...");  
$dbh = DBI->connect ("DBI:Oracle:...");
```

Po nawiązaniu połączenia nie trzeba wykonywać jakichkolwiek konkretnych odniesień do sterownika. Warstwa DBI i sterownik razem zajmują się obsługą szczegółów współpracy. Tak przedstawia się to rozwiązanie w teorii. Warto jednak pamiętać o dwóch kwestiach związanych z przenośnością skryptu DBI:

- Implementacje SQL różnią się w poszczególnych systemach baz danych i istnieje niebezpieczeństwo, że zapytania SQL działające w jednym serwerze nie będą zrozumiałe dla innego. Jeżeli stworzysz rozsądnie ogólne zapytania SQL, skrypty powinny być przenośne między różnymi serwerami. Jednak jeśli kod SQL został przygotowany pod kątem konkretnego serwera, będzie to dotyczyło również skryptu. Na przykład, jeśli skrypt używa charakterystycznego dla MySQL zapytania `SHOW VARIABLES`, nie będzie ono działało w innych serwerach baz danych.
- Moduły MySQL bardzo często dostarczają informacje charakterystyczne dla serwera, aby umożliwić twórcom skryptów użycie funkcji opracowanych

specjalnie dla danego systemu bazy danych. Na przykład, DBD dla MySQL zapewnia w wynikach zapytania dostęp do właściwości kolumn takich jak maksymalna wielkość wartości, sprawdzenie, czy kolumny są typu liczbowego, itd. Inne serwery baz danych niekoniecznie muszą udostępniać tego rodzaju informacje. Charakterystyczne dla DBD funkcje są sprzeczne z przenośnością kodu. Dlatego jeśli będziesz z nich korzystał, skrypt utworzony dla MySQL będzie znacznie trudniejszy do użycia w innych systemach baz danych.

Pomimo dwóch wymienionych powyżej czynników wiążących skrypty z konkretnymi bazami danych, mechanizm DBI zapewniający dostęp do bazy danych w sposób niezależny od jej rodzaju pozwala na osiągnięcie rozsądnego poziomu przenośności skryptów. Tylko od Ciebie zależy, ile umieścisz w skrypcie funkcji utrudniających jego przenośność. Jak się przekonasz w rozdziale 8., zatytułowanym „Tworzenie programów MySQL przy użyciu Perl DBI”, włożyłem niewiele wysiłku w celu uniknięcia charakterystycznych dla MySQL konstrukcji oferowanych przez MySQL DBD. Powód tego jest prosty: powinienś znać dostępne konstrukcje i samodzielnie podejmować decyzje o ich ewentualnym użyciu.

6.2.3. API PHP

Podobnie jak Perl, PHP to język skryptowy. Jednak w przeciwieństwie do Perla, PHP został zaprojektowany mniej jako język programowania ogólnego przeznaczenia, a bardziej pod kątem tworzenia aplikacji sieciowych. API PHP jest używane przede wszystkim do osadzania na stronach internetowych wykonywanych skryptów. To znacznie ułatwia programistom sieciowym tworzenie stron wraz z dynamicznie generowaną zawartością. Kiedy przeglądarka internetowa klienta wysyła do serwera WWW żądanie pobrania strony PHP, interpreter PHP wykonuje wszystkie skrypty znalezione na stronie i zastępuje je danymi wyjściowymi tych skryptów. Następnie wynik jest przekazywany do przeglądarki internetowej klienta. W ten sposób strona wyświetlana w przeglądarce internetowej zmienia się w zależności od okoliczności, w których wykonane zostało żądanie pobrania tej strony. Na przykład, po osadzeniu na stronie internetowej poniższego krótkiego skryptu PHP wyświetla on adres IP komputera klienta żądającego danej strony:

```
<?php echo $_SERVER["REMOTE_ADDR"]; ?>
```

Nieco trudniejszą i bardziej interesującą aplikacją może być skrypt zapewniający odwiedzającym informacje oparte na treści pochodzącej z bazy danych. Przedstawiony poniżej przykład prezentuje prosty skrypt, którego można użyć w witrynie internetowej Ligi Historycznej. Poniższy skrypt wykonuje zapytanie ustalające aktualną liczbę członków Ligi i wyświetlające tę liczbę odwiedzającemu:

```
<html>
<head>
<title>Liga Historyczna USA</title>
</head>
<body bgcolor="white">
<h2>Liga Historyczna USA</h2>
<p>Witamy w witrynie internetowej Ligi Historycznej USA.</p>
```

```

<?php
# Strona główna Ligi Historycznej USA.
try
{
    $dbh = new PDO("mysql:host=localhost;dbname=sampdb", "sampadm", "secret");
    $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $sth = $dbh->query("SELECT COUNT(*) FROM member");
    $count = $sth->fetchColumn(0);
    print("<p>Liga Historyczna liczy aktualnie $count członków.</p>");
    $dbh = NULL; # Zamknięcie połączenia.
}
catch(PDOException $e) { } # Pusta procedura obsługi (przechwytuje błędy, ale je ignoruje).
?>
</body>
</html>

```

Skrypty PHP z reguły wyglądają jak zwykłe strony HTML, ale zawierają kod wykonywalny osadzony w znacznikach `<?php i ?>`. Strona może zawierać dowolną liczbę fragmentów kodu PHP. To zapewnia niezwykle elastyczne podejście podczas tworzenia skryptu. Na przykład, początkowo możesz utworzyć skrypt PHP jak zwykłą stronę HTML w celu przygotowania ogólnej konfiguracji frameworka, a dopiero później dodać kod odpowiedzialny za generowanie dynamicznej treści strony.

PHP w rzeczywistości ma wiele typów interfejsów bazy danych. Wśród nich znajduje się zestaw bibliotek niskiego poziomu, z których każda działa z pojedynczym serwerem bazy danych. Nie podjęto wysiłku w celu ujednolicenia interfejsu dla różnych serwerów, tak jak ma to miejsce w przypadku DBI. Zamiast tego interfejs dla każdego serwera przypomina raczej interfejs dla odpowiedniej biblioteki utworzonej w języku C i implementującej niskiego poziomu API dla danego serwera. Na przykład, nazwy funkcji niskiego poziomu pozwalających na uzyskanie dostępu do MySQL z poziomu skryptów PHP są bardzo podobne do nazw funkcji w utworzonej w języku C bibliotece klienta MySQL.

Przypominające podejście DBI dla PHP oferuje rozszerzenie PDO (ang. *PHP Data Objects*). Wymienione rozszerzenie jest rodzajem abstrakcyjnego interfejsu dla systemów baz danych i używa dwupoziomowej architektury podobnej do DBI. Skrypty PHP przedstawione w rozdziale 9., zatytułowanym „Tworzenie programów MySQL przy użyciu języka PHP”, wykorzystują rozszerzenie PDO w celu uzyskania dostępu do bazy danych.

6.3. Wybór API

W tym podrozdziale zostaną przedstawione ogólne wskazówki pomagające w wyborze API dla różnego rodzaju aplikacji. Porównane będą możliwości oferowane przez API C, DBI i PHP, co pozwoli Ci na poznanie ich wad i zalet. Ponadto, znajdziesz tutaj odpowiedzi, kiedy wybrać jedno API zamiast innego.

Nie mam zamiaru narzucać Ci żadnego z wymienionych języków programowania, choć oczywiście mam swoje preferencje. Ty masz swoje, podobnie jak recenzenci techniczni niniejszej książki. Jeden z recenzentów uważał, że nieco bardziej powinienem podkreślić wagę języka C podczas programowania MySQL, podczas gdy drugi zasugerował,

aby odradzać używanie C. Jeśli weźmiesz pod uwagę czynniki przedstawione w tym rozdziale, samodzielnie wyciągniesz wnioski.

Podczas wyboru najodpowiedniejszego API do wykonania określonego zadania pod uwagę należy wziąć kilka czynników:

- **Przewidywane środowisko działania.** Kontekst, w którym będzie używana dana aplikacja.
- **Wydajność.** Jaka będzie wydajność aplikacji, gdy zostanie utworzona w języku wybranego API?
- **Łatwość programowania.** Jak wygodne jest wybrane API i sam język, w którym tworzona będzie aplikacja?
- **Przenośność.** Czy aplikacja będzie używana w systemach baz danych innych niż MySQL?

Musisz mieć świadomość, że wymienione współczynniki wpływają na siebie. Na przykład, celem może być opracowanie aplikacji o doskonałej wydajności, ale równie dobrze ważnym czynnikiem może być wybór języka pozwalającego na szybkie opracowanie aplikacji, nawet jeśli nie będzie działała wyjątkowo sprawnie.

6.3.1. Przewidywane środowisko działania

Podczas tworzenia aplikacji trzeba uwzględnić środowisko, w którym będzie używana. Na przykład, aplikacja może być programem generującym raporty i wywoływanym z poziomu powłoki lub też programem tworzącym podsumowania kont i uruchamianym przez mechanizm cron na koniec każdego miesiąca. Ogólnie rzecz biorąc, polecenia uruchamiane z poziomu powłoki lub mechanizmu cron są samodzielne i wymagają niewielkiej ilości informacji pochodzących ze środowiska uruchomieniowego. Ewentualnie możesz tworzyć aplikację przeznaczoną do uruchamiania w serwerze WWW. Tego rodzaju program może wymagać pobrania określonego rodzaju informacji ze środowiska uruchomieniowego, takich jak typ używanej przeglądarki internetowej, parametry podane w formularzu zapisu na listę dyskusyjną lub poprawne hasło pozwalające na uzyskanie dostępu do informacji o personelu.

API poszczególnych języków różnią się poziomem dostosowania do tworzenia aplikacji działających w wymienionych powyżej odmiennych środowiskach:

- C to język programowania ogólnego przeznaczenia, więc można go użyć do wykonania właściwie każdego zadania. W praktyce język C jest używany najczęściej do tworzenia samodzielnych programów, a nie do programowania sieciowego. Jednym z tego powodów jest brak łatwej możliwości przetwarzania tekstu i zarządzania pamięcią w C, jak ma to miejsce w językach Perl i PHP, a wymienione możliwości są często używane w aplikacjach sieciowych.
- Perl, podobnie jak C, jest odpowiedni do tworzenia samodzielnych programów. Jednak okazuje się, że jest także całkiem użyteczny w programowaniu sieciowym, na przykład dzięki wykorzystaniu modułu CGI.pm. W ten sposób Perl staje się

wygodnym językiem tworzenia aplikacji łączących MySQL z siecią. Tego rodzaju aplikacje mogą łączyć się z siecią za pomocą modułu CGLpm, natomiast z MySQL za pomocą modułu DBI.

- PHP został zaprojektowany do tworzenia aplikacji sieciowych i najlepiej sprawdza się w tym środowisku. Co więcej, dostęp do bazy danych to jedna z największych zalet języka PHP, który w ten sposób staje się naturalnym wyborem dla aplikacji sieciowych wykonujących zadania powiązane z MySQL. Istnieje możliwość tworzenia w PHP skryptów powłoki, ale to rzadko wykorzystywana funkcja.

Biorąc pod uwagę powyższe czynniki, języki C i Perl to kandydaci podczas tworzenia samodzielnych aplikacji. Dla aplikacji sieciowych najlepszym wyborem będzie język Perl lub PHP. Jeżeli zachodzi potrzeba tworzenia obu rodzajów aplikacji, a Ty nie znasz żadnego z wymienionych języków i nie chcesz uczyć się wszystkich, najlepszym wyborem może okazać się Perl.

6.3.2. Wydajność

Kiedy wszystkie inne czynniki są takie same, preferowane są aplikacje, które działają jak najszybciej. Jednak rzeczywista waga wydajności jest powiązana z częstotliwością używania danego programu. Jeśli program jest uruchamiany raz w miesiącu w nocy przez mechanizm cron, jego wydajność nie ma aż tak dużego znaczenia. Z kolei w przypadku programu uruchamianego wielokrotnie w ciągu sekundy w bardzo obciążonej witrynie internetowej każdy bit efektywności jest na wagę złota i może oznaczać dużą różnicę. W takim przypadku wydajność odgrywa ważną rolę w użyteczności i postrzeganiu witryny internetowej. Wolno działająca witryna jest irytująca dla odwiedzających ją użytkowników, niezależnie od jej zawartości. Jeśli witryna internetowa stanowi źródło dochodu, mniejsza wydajność jej działania bezpośrednio przekłada się na mniejszy dochód. Nie będziesz mógł obsłużyć wszystkich chętnych w danej chwili, a użytkownicy zniechęceni oczekiwaniem po prostu przejdą do innej witryny.

Zapewnienie wydajności to skomplikowana kwestia. Najlepszym sposobem przekonania się, jaka będzie wydajność aplikacji utworzonej dla określonego API, jest jej przygotowanie dla danego API i wypróbowanie. Najlepszym testem porównawczym jest implementacja wielu wersji za pomocą różnych API i zestawienie ich ze sobą. Oczywiście, prace programistyczne nie zawsze są prowadzone w taki właśnie sposób. Najczęściej po prostu tworzysz program. Kiedy masz działającą aplikację, wtedy zaczynasz się zastanawiać nad jej optymalizacją w celu szybszego działania, zużywania mniejszej ilości pamięci lub pod innym kątem, który ma dla Ciebie specjalne znaczenie. Jednak istnieją przynajmniej dwa ogólne czynniki, które w dość konsekwentny sposób wpływają na wydajność:

- Programy kompilowane są wykonywane o wiele szybciej niż interpretowane skrypty. Działają znacznie efektywniej, używają mniejszej ilości pamięci i są wykonywane szybciej niż odpowiadające im wersje programów w językach skryptowych. Wynika to z faktu ich bezpośredniego uruchamiania bez obciążenia

związanego z interpreterem języka wykonującego skrypt. C jest językiem kompilowanym, Perl i PHP są interpretowane, a więc programy C, ogólnie rzecz biorąc, będą działały szybciej niż skrypty w Perlu lub PHP. Dlatego też język C może być najlepszym wyborem dla często używanego programu. Jednak nadal warto rozważyć przeznaczenie aplikacji. Różnica między programami kompilowanymi i interpretowanymi zmniejsza się, gdy w aplikacjach skryptowych większość czasu wykonywania jest poświęcana na uruchamianie skompilowanych procedur biblioteki klienta MySQL powiązanych z silnikiem interpretera.

- W przypadku języków interpretowanych używanych w kontekście sieciowym wydajność jest lepsza, gdy interpreter jest wywoływany jako moduł serwera WWW niż jako oddzielny proces. Na przykład, jeśli używasz serwera WWW Apache, to możesz go skonfigurować do wywoływania interpretera skryptu jako oddzielnego procesu. W takim trybie działania, gdy Apache będzie musiał wykonać skrypt Perl lub PHP, uruchomi odpowiedni interpreter i nakaże mu wykonanie skryptu. Oznacza to, że Apache używa interpreterów jako programów CGI, czyli komunikuje się z nimi za pomocą protokołu CGI (ang. *Common Gateway Interface*). Z drugiej strony, interpreter może być używany jako moduł połączony bezpośrednio z plikiem binarnym Apache i działać jako część procesu samego serwera Apache. W kontekście Apache interpretery Perl i PHP przybierają postać modułów `mod_perl` i `mod_php`.

6.3.3. Czas potrzebny na opracowanie aplikacji

Wymienione w poprzednich punktach czynniki wpływają na wydajność działania aplikacji, ale efektywność wykonywania programu nie musi być Twoim jedynym celem. Czas programisty jest równie ważny jak łatwość programowania, a więc czas potrzebny na opracowanie aplikacji to kolejny czynnik, który warto wziąć pod uwagę podczas wyboru API dla programowania MySQL. Jeżeli opracowanie skryptu w Perlu lub PHP wymaga połowy czasu potrzebnego na opracowanie odpowiadającego mu programu w języku C, możesz postanowić o niewybraniu API C, nawet pomimo faktu, że skrypt będzie działał wolniej niż program w C. Często warto mniejszą wagę przykładать do efektywności wykonywania programu niż do czasu potrzebnego na jego utworzenie, zwłaszcza jeśli aplikacja będzie rzadko uruchamiana. Godzina pracy programisty jest warta znacznie więcej niż godzina pracy komputera!

Ogólnie rzecz biorąc, języki skryptowe pozwalają na szybkie utworzenie działającego programu, zwłaszcza na etapie prototypowania. Mają na to wpływ przynajmniej dwa czynniki. Po pierwsze, języki skryptowe zapewniają wyższego poziomu konstrukcje niż języki kompilowane. W ten sposób możesz myśleć na wyższym poziomie abstrakcji, a więc o tym, co chcesz uzyskać, a nie jak to zrobić. Na przykład, tablice asocjacyjne w PHP i wartości hash w Perlu zapewniają ogromną oszczędność czasu podczas obsługi danych obejmujących relacje typu klucz-wartość (takie jak para identyfikator-nazwisko). Język C nie oferuje tego

rodzaju konstrukcji. Aby ją zaimplementować w C, musisz utworzyć kod obsługujący na niskim poziomie wiele szczegółów związanych między innymi z zarządzaniem pamięcią, a później zająć się usuwaniem błędów z kodu. To wszystko wymaga czasu. Inną zaletą Perla i PHP są ich możliwości w zakresie dopasowania wzorca i przeprowadzania operacji na tekście. Na tych obszarach możliwości języka C są znacznie mniejsze.

Po drugie, w przypadku języków skryptowych cykl tworzenia aplikacji ma mniej kroków niż w języku kompilowanym. Tworząc aplikację w języku C, stosujesz cykl edycja-kompilacja-testowanie. Po wprowadzeniu każdej modyfikacji programu musisz go ponownie skompilować przed przetestowaniem. Z kolei w językach Perl i PHP cykl tworzenia sprowadza się do edycja-test, ponieważ skrypt można uruchomić natychmiast po wprowadzeniu modyfikacji i nie ma potrzeby jego kompilacji. Z drugiej strony, kompilator C nakłada znacznie więcej ograniczeń na program, na przykład w postaci ścisłego sprawdzania typu. Większa dyscyplina narzucana przez kompilator może pomóc w uniknięciu błędów, których wychwycenie nie jest wcale takie łatwe w luźniejszych językach, takich jak Perl i PHP. Jeśli błędnie zapiszesz nazwę zmiennej w C, to kompilator wygeneruje odpowiednie ostrzeżenie. PHP i Perl nie będą nawet o tym fakcie informować. Wspomniane większe ograniczenia mogą być szczególnie użyteczne, gdy aplikacja stanie się większa i znacznie trudniejsza w obsłudze.

Ogólnie rzecz biorąc, kompromis między językami kompilowanymi i interpretowanymi zwykle sprowadza się do czasu potrzebnego na opracowanie aplikacji kontra wydajność jej działania. Czy chcesz opracować program w języku kompilowanym, który będzie działał znacznie szybciej, ale wymaga poświęcenia dłuższego czasu na jego przygotowanie? Czy raczej chcesz utworzyć program w postaci skryptu, który przygotujesz szybko, ale kosztem mniejszej wydajności działania?

6.4.4. Przenośność

Przenośność określa, jak łatwo program utworzony do pracy z MySQL można przystosować do użycia z innym systemem bazy danych. To jest czynnik, który może w ogóle Cię nie interesować. Jednak, o ile nie zajmujesz się przewidywaniem przyszłości, nieco ryzykowne będzie stwierdzenie w stylu: „Nigdy nie będę używał tego programu z systemem bazy danych innym niż MySQL”. Przyjmujemy założenie, że masz inną pracę, ale chcesz wykorzystać wcześniej utworzone programy. Jednak nowy pracodawca używa innego systemu bazy danych. Co możesz wtedy zrobić? Jeżeli przenośność kodu jest ważna, pamiętaj o wymienionych poniżej różnicach między poszczególnymi API:

- API C w ogóle nie jest przenośne. Z natury zostało przygotowane specjalnie dla MySQL.
- API Perl DBI jest niezwykle przenośne, ponieważ niezależność od bazy danych to cel, który przyświecał utworzeniu DBI.
- Przenośność skryptu PHP jest na poziomie podobnym do DBI, o ile używasz wspomnianego wcześniej rozszerzenia PDO zapewniającego dostęp do bazy danych.

Przenośność w postaci niezależności od bazy danych jest szczególnie ważna, gdy w ramach pojedynczej aplikacji trzeba zachować dostęp do wielu systemów baz danych. Na przykład, może wystąpić konieczność eksportu danych z jednego systemu bazy danych, a następnie ich importu w innym lub wygenerowania raportu na podstawie informacji pochodzących z różnych systemów baz danych.

Tworzenie programów MySQL przy użyciu języka C

MySQL oferuje utworzoną w języku programowania C bibliotekę klienta, którą można wykorzystać do opracowywania programów klienckich uzyskujących dostęp do baz danych MySQL. Wspomniana biblioteka definiuje interfejs programowania aplikacji (API) oferujący następujące możliwości:

- Procedury zarządzania połączeniem pozwalające na otwieranie i zamykanie sesji pracy z serwerem.
- Procedury pozwalające na tworzenie zapytań SQL, przekazywanie ich do serwera i przetwarzanie wyników.
- Funkcje odpowiedzialne za sprawdzanie stanu i informowanie o błędach. Pozwalają one na ustalenie przyczyny błędu, gdy wywołanie API zakończy się niepowodzeniem.
- Procedury przetwarzające opcje podane w pliku opcji lub wierszu poleceń.

W tym rozdziale dowiesz się, jak używać utworzonej w języku C biblioteki klienta do opracowywania własnych programów, używając konwencji sensownie spójnych z zastosowanymi w programach klienckich rozprowadzanych wraz z dystrybucją MySQL.

W pierwszej części programu utworzymy serie krótkich programów. Seria zakończy się prostym programem działającym w charakterze frameworka dla klienta i jego celem będzie jedynie nawiązanie i zamknięcie połączenia z serwerem. Powód jest prosty: wprawdzie programy klienckie MySQL są tworzone z różnych powodów, ale na pewno jedno je łączy — konieczność nawiązania połączenia z serwerem.

Przygotowany przez nas program frameworka będzie prosty, ale wyposażony w kod pozwalający na przetwarzanie opcji i obsługę błędów. Stanowi więc solidną podstawę dla wielu różnych programów klienckich. Po jego opracowaniu odetchniemy na chwilę od języka C i przekonamy się, jak wykonywać różnego rodzaju zapytania SQL. Początkowo omówione będą zapytania na stałe zdefiniowane w kodzie, a dopiero później przejdziemy do kodu pozwalającego na przetwarzanie różnych zapytań. Następnie do tworzonego

frameworka dodamy kod odpowiedzialny za przetwarzanie zapytań i tym samym otrzymamy program podobny do klienta `mysql`, którego będzie można używać do interaktywnego wykonywania zapytań.

W dalszej części rozdziału zademonstrowanych będzie wiele innych zadań obsługiwanych przez bibliotekę klienta:

- Tworzenie programów klienckich, które komunikują się z serwerem za pomocą bezpiecznego połączenia używającego protokołu SSL (ang. *Secure Sockets Layer*).
- Jednorazowe wysyłanie do serwera wielu zapytań i przetwarzanie otrzymanego zbioru wynikowego.
- Używanie zapytań przygotowanych po stronie serwera.

W tym rozdziale przeanalizowano jedynie funkcje i typy danych z biblioteki klienta, które są potrzebne w przykładowych programach.

Przykładowe programy są dostępne w postaci plików, co pozwala na ich wypróbowanie bez konieczności samodzielnego wpisywania. Informacje dotyczące pobrania dystrybucji `sampdb` zawierającej te programy znajdziesz w dodatku A, zatytułowanym „Oprogramowanie wymagane do użycia tej książki”. Wspomniane programy umieszczono w katalogu *cap* dystrybucji.

Gdzie znajdę przykładowe programy?

Na listach dyskusyjnych poświęconych MySQL bardzo często pojawia się pytanie w stylu: gdzie znajdę przykłady klientów utworzonych w języku C? Odpowiedź brzmi: oczywiście w tej książce. Ponadto, dystrybucja kodu źródłowego MySQL zawiera kilka programów klienckich utworzonych w języku C (`mysql`, `mysqladmin` i `myqldump`). Ponieważ wspomniana dystrybucja jest powszechnie dostępna, znajdziesz w niej całkiem sporą ilość przykładowego kodu. Jeśli jeszcze tego nie zrobiłeś, pobierz kod źródłowy dystrybucji i zajrzyj do katalogów *client* oraz *tests*.

7.1. Kompilacja i linkowanie programów klienckich

W tym podrozdziale dowiesz się, jak skompilować i linkować program używający biblioteki klienckiej MySQL. Polecenia używane do utworzenia klientów różnią się w poszczególnych systemach operacyjnych i może wystąpić potrzeba modyfikacji przedstawionych tutaj poleceń. Jednak sam opis jest ogólny i powinieneś bez problemów móc go zastosować w większości tworzonych programów klienckich.

Kiedy w języku C tworzysz program klienta MySQL, do pracy potrzebujesz oczywiście kompilatora C. Zaprezentowane tutaj przykłady opierają się na gcc, czyli najczęściej używanym kompilatorze C w środowisku UNIX. Będziesz również potrzebował plików nagłówkowych MySQL oraz biblioteki klienta.

Pliki nagłówkowe i biblioteka klienta stanowią podstawy wymagane do rozpoczęcia obsługi programowania klienta MySQL. Jeżeli baza danych MySQL została zainstalowana

w systemie ze źródeł lub dystrybucji binarnej, pliki wymagane do zapewnienia obsługi programowania klienta również powinny znajdować się w systemie. Jeśli instalację przeprowadziłeś z pakietów RPM, wspomniana obsługa będzie niedostępna aż do chwili zainstalowania pakietu RPM dla programistów MySQL. Jeśli nie wiesz, jak pobrać pliki nagłówkowe MySQL i bibliotekę klienta, zajrzyj do dodatku A, zatytułowanego „Oprogramowanie wymagane do użycia tej książki”.

W celu przeprowadzenia kompilacji i linkowania programu klienta musisz wskazać położenie plików nagłówkowych MySQL i biblioteki klienta, o ile nie są zainstalowane w katalogach domyślnie sprawdzanych przez kompilator i linkera. Dla przedstawionych poniżej przykładów przyjęto założenie, że pliki nagłówkowe i biblioteka klienta znajdują się w katalogach odpowiednio `/usr/local/include/mysql` i `/usr/local/lib/mysql`. Jeśli wystąpi potrzeba, to zmodyfikuj ścieżki dostępu, aby odpowiadały używanym w Twoim systemie.

Podczas kompilacji pliku źródłowego na plik obiektu możesz podpowiedzieć kompilatorowi, gdzie znajdują się pliki nagłówkowe MySQL; wystarczy użyć opcji `-I` wraz z nazwą odpowiedniego katalogu. Na przykład, aby skompilować plik `myclient.c` i otrzymać `myclient.o`, wydaj przedstawione poniżej polecenie:

```
% gcc -c -I/usr/local/include/mysql myclient.c
```

Aby wskazać linkerowi miejsce położenia biblioteki klienta oraz jej nazwę, należy użyć argumentów `-L/usr/local/lib/mysql` i `-lmysqlclient` podczas linkowania pliku obiektu i generowania pliku wykonywalnego:

```
% gcc -o myclient myclient.o -L/usr/local/lib/mysql -lmysqlclient
```

Jeżeli klient składa się z wielu plików, w poleceniu trzeba wymienić wszystkie.

Operacja linkowania może skutkować wygenerowaniem komunikatu błędu, jeśli nie zostaną znalezione użyte funkcje. W takich przypadkach trzeba skorzystać z dodatkowych opcji `-l` i wskazać biblioteki zawierające funkcje. Jeżeli zobaczysz komunikat informujący o funkcjach `compress()` lub `uncompress()`, spróbuj dodać `-lz` lub `-lgz`, nakazując tym samym linkerowi znalezienie biblioteki z `lib` odpowiedzialnej za kompresję:

```
% gcc -o myclient myclient.o -L/usr/local/lib/mysql -lmysqlclient -lz
```

Jeżeli wiadomość zawiera wymienioną funkcję `floor()`, dodaj `-lm`, powodując dołączenie biblioteki matematycznej. Może wystąpić konieczność dołączenia również innych bibliotek.

Alternatywą dla samodzielnego ustalania odpowiednich flag potrzebnych podczas kompilacji i linkowania programów MySQL jest użycie narzędzia `mysql_config`, które automatycznie ustali wymagane opcje. Na przykład, wymienione narzędzie może wskazać konieczność użycia pewnych opcji:

```
% mysql_config --include  
-I'/usr/local/mysql/include'  
% mysql_config --libs  
-L'/usr/local/mysql/lib' -lmysqlclient -lpthread -lz -lm -lrt -ldl
```

Aby narzędzia `mysql_config` użyć bezpośrednio w poleceniu kompilacji lub linkowania, należy je ująć w odwrotne apostrofy:

```
% gcc -c `mysql_config --include` myclient.c
% gcc -o myclient myclient.o `mysql_config --libs`
```

Powłoka wywoła narzędzie `mysql_config` i jego dane wyjściowe umieści w poleceniu kompilacji lub linkowania, co automatycznie dostarczy odpowiednie flagi kompilatorowi `gcc`.

Jeżeli podczas tworzenia programów nie używasz `make`, to sugeruję poznanie wymienionego narzędzia, ponieważ w ten sposób możesz zaoszczędzić sobie ręcznego wpisywania dużej liczby poleceń. Przyjmujemy założenie, że masz program klienta o nazwie `myclient`, składający się z dwóch plików kodu źródłowego `main.c` i `lib.c` oraz pliku nagłówkowego `myclient.h`.

Prosty plik `Makefile`, pozwalający na kompilację programu, został przedstawiony poniżej. Wcięcia muszą być tabulatorami, spacje nie działają.

```
CC = gcc
INCLUDES = -I/usr/local/include/mysql
LIBS = -L/usr/local/lib/mysql -lmysqlclient

all: myclient

main.o: main.c myclient.h
    $(CC) -c $(INCLUDES) main.c
lib.o: lib.c myclient.h
    $(CC) -c $(INCLUDES) lib.c

myclient: main.o lib.o
    $(CC) -o myclient main.o lib.o $(LIBS)

clean:
    rm -f myclient main.o lib.o
```

Po modyfikacji dowolnego pliku źródłowego za pomocą pliku `Makefile` można ponownie skompilować program przez wydanie po prostu polecenia `make`, które wyświetli i wykona niezbędne polecenia:

```
% make
gcc -c -I/usr/local/mysql/include/mysql myclient.c
gcc -o myclient myclient.o -L/usr/local/mysql/lib/mysql -lmysqlclient
```

Takie rozwiązanie jest łatwiejsze i bardziej odporne na błędy niż wpisywanie długich poleceń `gcc`. Plik `Makefile` ułatwia także zmianę procesu kompilacji. Na przykład, jeśli w trakcie operacji linkowania używany przez Ciebie system wymaga dodatkowych bibliotek, np. matematycznej lub kompresji, wtedy przeprowadź edycję wiersza `LIBS` w pliku `Makefile` i dodaj opcję `-lm` lub `-lz`:

```
LIBS = -L/usr/local/lib/mysql -lmysqlclient -lm -lz
```

Jeżeli konieczne będą inne biblioteki, dodaj je do wiersza `LIBS`. Po wywołaniu `make` wartości `LIBS` zostaną automatycznie uaktualnione.

Sposobem zmiany zmiennych `make` w inny sposób niż przez edycję pliku `Makefile` jest ich podanie w wierszu poleceń. Na przykład, jeśli kompilator C ma nazwę `cc` zamiast `gcc`, wówczas możesz wydać poniższe polecenie:

```
% make CC=cc
```

Jeżeli narzędzie `mysql_config` jest dostępne, możesz go użyć w celu uniknięcia wprowadzania w `Makefile` dosłownych ścieżek dostępu do nazw katalogów plików nagłówkowych i biblioteki. Wiersze `INCLUDES` i `LIBS` zapisz w następujący sposób:

```
INCLUDES = ${shell mysql_config --include}
LIBS = ${shell mysql_config --libs}
```

Po uruchomieniu `make` nastąpi wykonanie `mysql_config` i użycie danych wyjściowych tego narzędzia do ustawienia wartości odpowiednim zmiennym. Konstrukcja `${shell}` jest obsługiwana przez GNU `make`. Jeżeli używana przez Ciebie wersja `make` nie jest oparta na GNU, może wystąpić potrzeba zastosowania innej składni.

Jeżeli używasz zintegrowanego środowiska programistycznego (ang. *Integrated Development Environment*, IDE), wtedy w ogóle nie musisz korzystać z `make`. Szczegóły zależą od konkretnego środowiska IDE.

7.2. Nawiązanie połączenia z serwerem

Nasz pierwszy program klienta MySQL będzie najprostszy z możliwych: nawiąże połączenie z serwerem, zamknie połączenie, a następnie zakończy działanie. Sam w sobie to nie jest zbyt użyteczny program, ale musisz wiedzieć, jak przeprowadzać wymienione operacje, ponieważ przed wykonaniem jakiegokolwiek zapytania do bazy danych MySQL konieczne jest nawiązanie połączenia z serwerem. Nawiązywanie połączenia z serwerem to tak często wykonywana operacja, że opracowany tutaj kod będzie używany w każdym tworzonym programie klienta. Poza tym, wymienione zadanie pozwala nam na rozpoczęcie pracy od czegoś naprawdę prostego. Sam kod można później rozbudować, aby wykonywał bardziej użyteczne operacje.

Nasz pierwszy program klienta, `connect1`, składa się z pojedynczego pliku źródłowego o nazwie `connect1.c`:

```
/*
 * connect1.c - Nawiązanie połączenia z serwerem MySQL, a następnie jego zamknięcie.
 */

#include <my_global.h>
#include <my_sys.h>
#include <mysql.h>

static char *opt_host_name = NULL; /* Komputer serwera (domyślnie = localhost). */
static char *opt_user_name = NULL; /* Nazwa użytkownika (domyślnie = nazwa logowania). */
static char *opt_password = NULL; /* Hasło (domyślnie = brak). */
static unsigned int opt_port_num = 0; /* Numer portu (użyj wbudowanej wartości). */
static char *opt_socket_name = NULL; /* Nazwa gniazda (użyj wbudowanej wartości). */
static char *opt_db_name = NULL; /* Nazwa bazy danych (domyślnie = brak). */
```

```

static unsigned int opt_flags = 0;      /* Flagi połączenia (brak). */

static MYSQL *conn;                      /* Wskaźnik do uchwytu połączenia. */

int main (int argc, char *argv[])
{
    MY_INIT (argv[0]);
    /* Inicjalizacja biblioteki klienta. */
    if (mysql_library_init (0, NULL, NULL))
    {
        fprintf (stderr, "Wywołanie mysql_library_init() zakończyło się niepowodzeniem\n");
        exit (1);
    }
    /* Inicjalizacja uchwytu połączenia. */
    conn = mysql_init (NULL);
    if (conn == NULL)
    {
        fprintf (stderr, "Wywołanie mysql_init() zakończyło się niepowodzeniem
(prawdopodobnie z powodu braku pamięci)\n");
        exit (1);
    }
    /* Nawiązanie połączenia z serwerem. */
    if (mysql_real_connect (conn, opt_host_name, opt_user_name, opt_password,
        opt_db_name, opt_port_num, opt_socket_name, opt_flags) == NULL)
    {
        fprintf (stderr, "Wywołanie mysql_real_connect() zakończyło się niepowodzeniem\n");
        mysql_close (conn);
        exit (1);
    }
    /* Zamknięcie połączenia z serwerem i zamknięcie biblioteki klienta. */
    mysql_close (conn);
    mysql_library_end ();
    exit (0);
}

```

Plik kodu źródłowego rozpoczyna się od dołączenia plików nagłówkowych *my_global.h*, *my_sys.h* i *mysql.h*. W zależności od działania programu klienta MySQL może wystąpić konieczność dołączenia jeszcze innych plików nagłówkowych. Trzy wymienione to z reguły absolutne minimum:

- *my_global.h* zawiera wiele plików nagłówkowych (na przykład *stdio.h*), które prawdopodobnie będą użyteczne. Ponadto, zawiera informacje o zgodności z Windows, jeśli program jest kompilowany w systemie Windows. (Wprawdzie możesz nie przewidywać używania systemu Windows, ale jeśli zamierzasz rozprowadzać kod, to użycie pliku nagłówkowego *my_global.h* pomoże wszystkim, którzy planują kompilację klienta w Windows).
- *my_sys.h* zawiera makra przenośności oraz definicje struktur i funkcji używanych przez bibliotekę klienta.
- *mysql.h* definiuje podstawowe ograniczenia i struktury danych powiązane z MySQL.

Kolejność dołączania plików jest ważna; *my_global.h* powinien być dołączony przed wszystkimi plikami nagłówkowymi powiązanymi z MySQL.

Następnie w programie znajdują się deklaracje zmiennych odpowiadających parametrom podawanym podczas nawiązywania połączenia z serwerem. W omawianym kliencie parametry mają na stałe zdefiniowane wartości domyślne. Nieco później zastosujemy znacznie elastyczniejsze podejście, pozwalające na nadpisanie wartości domyślnych wartościami podanymi w pliku opcji lub wierszu poleceń. (Dlatego też wszystkie nazwy rozpoczynają się od `opt_`, wartości tych zmiennych później będą ustawiane za pomocą opcji polecenia). Program ma także zadeklarowany wskaźnik do struktury MySQL, która będzie działała w charakterze uchwytu połączenia.

Funkcja `main()` programu nawiązuje połączenie z serwerem, a następnie je zamyka. Nawiązanie połączenia to proces składający się z dwóch kroków:

1. Wywołanie funkcji `mysql_init()` w celu pobrania uchwytu połączenia.
2. Wywołanie funkcji `mysql_real_connect()` w celu nawiązania połączenia z serwerem.

Przekazanie wartości NULL funkcji `mysql_init()` automatycznie powoduje alokację struktury MySQL, jej inicjalizację i zwrot prowadzącego do niej wskaźnika. Typ danych MySQL jest strukturą zawierającą informacje o połączeniu. Zmienne tego typu są nazywane „uchwytemi połączenia”.

Inne podejście polega na przekazaniu wskaźnika do istniejącej struktury MySQL. W takim przypadku funkcja `mysql_init()` zainicjalizuje strukturę i zwróci prowadzący do niej wskaźnik, ale nie zaalokuje struktury.

Funkcja `mysql_real_connect()` pobiera mnóstwo parametrów:

- Wskaźnik do uchwytu połączenia. To powinna być wartość zwrócona przez funkcję `mysql_init()`.
- Komputer serwera. Interpretacja tej wartości zależy od platformy. W systemach UNIX, jeśli podasz ciąg tekstowy zawierający nazwę komputera lub jego adres IP, klient nawiąże połączenie ze wskazanym komputerem za pomocą połączenia TCP/IP. Jeśli podasz NULL lub nazwę `localhost`, klient nawiąże połączenie z serwerem działającym w komputerze lokalnym i użyje do tego pliku gniazda systemu UNIX. W systemie Windows zachowanie jest podobne, choć w przypadku wartości `localhost` do połączenia zostanie użyta pamięć współdzielona lub połączenie TCP/IP zamiast pliku gniazda systemu UNIX. W Windows, jeśli nazwa komputera to `.` lub NULL i serwer obsługuje połączenia nazwanych potoków, wtedy próba połączenia z serwerem lokalnym odbywa się za pomocą nazwanego potoku.
- Nazwa użytkownika i hasło konta MySQL, które ma zostać użyte. Jeśli nazwą będzie NULL, biblioteka klienta użyje Twojej nazwy logowania (lub ODBC w Windows). Jeśli hasło będzie NULL, wtedy żadne hasło nie zostanie wysłane.
- Nazwa bazy danych do wybrania jako domyślnej po nawiązaniu połączenia. Użycie wartości NULL powoduje, że nie zostanie wybrana żadna baza danych.
- Numer portu. Ten port jest używany w połączeniach TCP/IP. Wartość 0 informuje bibliotekę klienta, że ma być użyty domyślny numer portu.

- Nazwa pliku gniazda. W systemach UNIX to będzie nazwa pliku gniazda UNIX służącego do nawiązywania połączeń. Z kolei w Windows ta nazwa jest interpretowana jako nazwa dla połączenia przez potok. Wartość NULL nakazuje bibliotece klienta użycie nazwy gniazda (lub potoku) domyślnego.
- Wartość flag. Program `connect1` przekazuje wartość 0, ponieważ nie używa żadnych specjalnych opcji połączenia.

Aby zamknąć połączenie, należy wywołać funkcję `mysql_close()` i przekazać jej wskaźnik do uchwytu połączenia. Jeśli uchwyt został zaalokowany automatycznie na skutek przekazania wartości NULL funkcji `mysql_init()`, wtedy funkcja `mysql_close()` automatycznie usunie uchwyt po zamknięciu połączenia. Po wywołaniu funkcji `mysql_close()` uchwyt nie może być używany do dalszej komunikacji z serwerem.

Poza kodem przeznaczonym do nawiązania połączenia, w pliku `connect1.c` znajdują się jeszcze trzy inne wywołania:

- `MY_INIT()` to jest makro inicjalizacyjne. Powoduje przypisanie zmiennej globalnej nazwy programu (przekazywanej mu jak argument) do użycia przez biblioteki MySQL w komunikatach błędów. Ponadto, wywołuje funkcję `my_init()` w celu przeprowadzenia pewnych operacji konfiguracyjnych.
- Funkcja `mysql_library_init()` inicjalizuje bibliotekę klienta MySQL. Tę funkcję trzeba wywołać przed wywołaniem jakiejkolwiek innej funkcji `mysql_XXX()`.
- Funkcja `mysql_library_end()` powoduje zakończenie używania biblioteki klienta i przeprowadza wszelkie niezbędne operacje czyszczące. Tę funkcję trzeba wywołać po zakończeniu pracy z biblioteką klienta.

Aby wypróbować program `connect1`, musisz przeprowadzić kompilację i linkowanie, opierając się na informacjach przedstawionych w poprzednim podrozdziale, a następnie uruchomić go. W systemach UNIX program uruchamiasz w następujący sposób:

```
% ./connect1
```

Znaki `./` na początku są niezbędne w systemie UNIX, jeśli powłoka nie ma katalogu bieżącego (`.`) podanego w ścieżce wyszukiwania. Jeśli katalog bieżący znajduje się w ścieżce wyszukiwania lub używasz systemu Windows, wtedy możesz pominąć część `./` w nazwie polecenia:

```
% connect1
```

Jeżeli program `connect1` nie wygeneruje żadnych danych wyjściowych, oznacza to, że nawiązanie połączenia zakończyło się powodzeniem. W przeciwnym razie zostanie wyświetlony komunikat błędów:

```
% ./connect1
```

Wywołanie `mysql_real_connect()` zakończyło się niepowodzeniem

Powyższe dane wyjściowe informują o nienawiązaniu połączenia, ale bez podania przyczyny. Bardzo prawdopodobną przyczyną niepowodzenia są nieodpowiednie

parametry domyślne połączenia (nazwa komputera, nazwa użytkownika itd.). Przyjmując założenie, że tak jest faktycznie, jedynym sposobem usunięcia tego problemu jest ponowna kompilacja programu po wcześniejszym poprawieniu poleceń inicjalizujących zmienne parametrów połączenia i przypisanie im prawidłowych wartości pozwalających na uzyskanie dostępu do serwera. Będzie to miało tę zaletę, że przynajmniej uzyskasz możliwość nawiązania połączenia. Jednak program nadal zawiera zdefiniowane na stałe wartości parametrów połączenia, co nie jest zbyt elastyczne, jeśli klient ma być używany przez innych użytkowników. To także niebezpieczne, ponieważ ujawnia hasło dostępu do bazy danych. Być może sądzisz, że hasło stanie się ukryte po kompilacji programu na postać binarnego pliku wykonywalnego, ale wcale nie będzie ukryte dla użytkownika, który uruchomi narzędzie strings względem pliku binarnego. Ponadto, każdy, kto uzyska uprawnienia odczytu pliku źródłowego, bez problemów odczyta hasło.

W powyższym akapicie wymieniono dwie główne wady programu connect1:

- Komunikat błędu nie podaje przyczyny problemu.
- Brakuje elastycznego sposobu podawania parametrów przez użytkownika uruchamiającego tego klienta. Wspomniane parametry są na stałe zdefiniowane w kodzie źródłowym. Znacznie lepiej byłoby umożliwić użytkownikowi nadpisanie parametrów w wierszu poleceń lub pliku opcji.

W kolejnym podrozdziale usuniemy wymienione błędy.

7.3. Obsługa błędów i przetwarzanie opcji polecenia

Nasz kolejny klient, connect2, jest podobny do connect1 w tym sensie, że nawiązuje połączenie z MySQL, zamyka połączenie i kończy pracę. Jednak w programie connect2 wprowadzono dwa ważne usprawnienia:

- Znacznie lepiej radzi sobie z informowaniem o błędach, ponieważ używa funkcji biblioteki klienta MySQL zwracających konkretne informacje o przyczynie błędów.
- Dostarcza domyślne parametry połączenia, ale pozwala użytkownikowi na ich nadpisanie opcjami podanymi w wierszu poleceń lub pliku opcji.

7.3.1. Sprawdzanie pod kątem błędów

Przeanalizujmy najpierw temat obsługi błędów. Na początku warto podkreślić wagę operacji sprawdzania, czy nie wystąpił błąd podczas wywoływania funkcji MySQL, której działanie może zakończyć się niepowodzeniem. Dość często zdarza się, że komunikaty przygotowane przez programistów mają następujące znaczenie: „sprawdzanie, czy nie wystąpił błąd, pozostawiono jako ćwiczenie dla użytkowników”. Prawdopodobna przyczyna tego jest prosta: sprawdzanie błędów jest nudne. Nieważne, konieczne jest przeprowadzenie

sprawdzenia pod kątem wystąpienia błędów i podjęcie odpowiednich działań. Funkcje biblioteki klienta zwracające wartości o stanie są przeznaczone właśnie do wspomnianego sprawdzenia; możesz je zignorować na własne ryzyko. Na przykład, jeśli funkcja zwraca wskaźnik do struktury danych lub NULL w celu wskazania błędu, lepiej będzie sprawdzić jej wartość zwrótną. Próba użycia w dalszej części programu wartości NULL, gdy oczekiwana jest struktura danych, może prowadzić do nieoczekiwanych wyników lub awarii aplikacji.

Brak sprawdzania wartości zwrótniej to przyczyna wielu niepotrzebnych trudności podczas programowania i fenomen często pojawiający się na listach dyskusyjnych poświęconych MySQL. Typowe pytania, z którymi można się spotkać, to: „Dlaczego program ulega awarii, gdy wykonuję to zapytanie?” i „Jak to możliwe, że to zapytanie nie zwraca żadnych wyników?”. W wielu przypadkach program, którego dotyczy pytanie, przed wykonaniem zapytania nie przeprowadza operacji sprawdzenia, czy połączenie zostało nawiązane, lub przed pobraniem wyników zapytania nie przeprowadza operacji sprawdzenia, czy wykonanie zapytania przez serwer zakończyło się powodzeniem.

Nie rób błędu, przyjmując założenie, że każde wywołanie biblioteki klienta kończy się powodzeniem. Jeśli nie sprawdzisz wartości zwrótnych, wtedy możesz skończyć na wyszukiwaniu niejasnych problemów występujących w programie lub użytkownicy będą się zastanawiać, dlaczego program zachowuje się w nieprzewidywalny sposób.

Znajdujące się w bibliotece klienta MySQL procedury zwracające wartość z reguły wskazują powodzenie lub niepowodzenie operacji na jeden z dwóch sposobów, w zależności od typu wartości zwrótniej (wskaźnik lub liczba całkowita).

Funkcje zwracające wskaźnik w przypadku sukcesu zwracają wskaźnik inny niż NULL, natomiast w przypadku niepowodzenia wartością zwrótną jest NULL. (W tym kontekście NULL oznacza „wskaźnik NULL języka C”, a nie „wartość NULL kolumny MySQL”).

Z użytych dotąd procedur biblioteki klienta, `mysql_init()` i `mysql_real_connect()` zwracają wskaźnik do uchwytu połączenia (sukces) lub wartość NULL (niepowodzenie).

Funkcje zwracające liczby całkowite najczęściej zwracają 0 w przypadku sukcesu i wartość niezerową w przypadku niepowodzenia. Bardzo ważne jest, aby nie przeprowadzać sprawdzenia pod kątem określonej wartości niezerowej, na przykład -1. Nie ma gwarancji, że biblioteka klienta zwróci konkretną wartość w przypadku niepowodzenia operacji. Czasami można się spotkać z kodem nieprawidłowo sprawdzającym wartość zwrótną z funkcji API C `mysql_XXX()`:

```
if (mysql_XXX () == -1)      /* Ten test jest nieprawidłowy. */
    fprintf (stderr, "Wystąpił błąd.\n");
```

Powyższy test może działać, ale nie musi. API MySQL nie określa, że konkretna wartość niezerowa zostanie zwrócona w przypadku wystąpienia błędu; to będzie po prostu wartość inna niż 0. Test lepiej zapisać w poniższej postaci:

```
if (mysql_XXX () != 0)      /* Ten test jest prawidłowy. */
    fprintf (stderr, "Wystąpił błąd.\n");
```

Ewentualnie, test można zapisać w jeszcze nieco prostszej postaci, ale odpowiadającej poprzedniej:

```
if (mysql_XXX ()) /* Ten test jest prawidłowy. */
    fprintf (stderr, "Wystąpił błąd.\n");
```

Jeżeli zajrzysz do kodu źródłowego samego serwera MySQL, przekonasz się, że w operacjach sprawdzania, ogólnie rzecz biorąc, jest stosowana druga postać testu.

Nie każde wywołanie API zwraca wartość. Z innych użytych dotąd procedur klienta funkcja `mysql_close()` nie zwraca wartości. (Czy jej działanie może zakończyć się niepowodzeniem? Jeśli tak, co wówczas? Pracę z połączeniem i tak już zakończyłeś).

Kiedy wywołanie biblioteki klienta zakończy się niepowodzeniem, jego przyczynę możesz poznać dzięki trzem wywołaniom w API:

- Wartością zwrótną funkcji `mysql_error()` jest ciąg tekstowy zawierający komunikat błędu.
- Wartością zwrótną funkcji `mysql_errno()` jest zdefiniowany w MySQL numer kodu błędu.
- Wartością zwrótną funkcji `mysql_sqlstate()` jest kod SQLSTATE. Wartość SQLSTATE jest bardziej niezależna od producenta serwera bazy danych, ponieważ opiera się na standardach ANSI SQL i ODBC.

Argumentem wszystkich wymienionych powyżej funkcji jest wskaźnik do uchwytu połączenia. Wywołaj je po wystąpieniu błędu. Jeśli wykonasz inne wywołanie API zwracające informacje o stanie, wszelkie komunikaty błędu wygenerowane przez funkcje `mysql_error()`, `mysql_errno()` lub `mysql_sqlstate()` będą miały zastosowanie do późniejszego wywołania.

Ogólnie rzecz biorąc, dla użytkownika programu komunikat błędu jest bardziej zrozumiały niż jego kod. Dlatego też jeśli zgłaszasz tylko jedną wartość, najlepiej, aby był to komunikat błędu. Przedstawione w tym rozdziale przykłady zgłaszają wszystkie trzy wartości. Jednak żmudne jest tworzenie trzech funkcji wywoływanych w każdym miejscu, w którym może wystąpić błąd. Zamiast tego warto utworzyć funkcję narzędziową, na przykład o nazwie `print_error()`, wyświetlającą przygotowane przez nas komunikaty błędów oraz wartości błędów dostarczane przez procedury biblioteki klienta MySQL. Innymi słowy, w celu sprawdzenia, czy wystąpił błąd, za każdym razem nie będziemy umieszczać w kodzie wywołań trzech funkcji: `mysql_error()`, `mysql_errno()` i `mysql_sqlstate()`.

```
if (...działanie pewnej funkcji MySQL zakończy się niepowodzeniem...)
{
    fprintf (stderr, "...komunikat błędu...\nBłąd %u (%s): %s\n",
            mysql_errno (conn), mysql_sqlstate (conn), mysql_error (conn));
}
```

Zgłaszanie błędów za pomocą funkcji pomocniczej jest proste i sprowadza się do jej wywołania:

```
if (...działanie pewnej funkcji MySQL zakończy się niepowodzeniem...)
{
    print_error (conn, "...komunikat błędu...");
}
```

Funkcja `print_error()` wyświetla komunikat błędu i wywołuje funkcje błędów MySQL. Wywołanie funkcji `print_error()` jest prostsze niż `fprintf()`, więc łatwiej ją utworzyć, a sam kod programu jest czytelniejszy. Ponadto, jeżeli funkcja `print_error()` zostanie przygotowana do wykonania sensownej operacji, nawet jeśli `conn` ma wartość `NULL`, wtedy można jej użyć w sytuacjach takich jak zakończone niepowodzeniem wywołanie funkcji `mysql_init()`. Unikniemy wówczas stosowania różnych wywołań zgłaszania błędów — części `fprintf()`, a części w postaci `print_error()`.

Już słyszę, jak ktoś może mieć pewne wątpliwości: „Nie trzeba wywoływać wszystkich funkcji błędu za każdym razem, gdy chcesz zgłosić błąd. Celowo wyolbrzymiasz zadanie zgłaszania błędów, aby zaproponowana funkcja narzędziowa sprawiała wrażenie jeszcze bardziej użytecznej. I naprawdę nie trzeba całego kodu wyświetlającego błędy pisać za każdym razem, wystarczy to zrobić raz, a następnie skorzystać z techniki kopiuj i wklej”. To są rozsądne wątpliwości, ale odpowiadam na nie w następujący sposób:

- Nawet jeśli używasz techniki „kopiuj i wklej”, to łatwiej ją stosować w przypadku krótszych fragmentów kodu.
- Jeżeli zgłaszanie błędów okaże się łatwiejsze, to prawdopodobnie będziesz konsekwentnie je stosował wszędzie tam, gdzie powinienieś.
- Niezależnie od tego, czy preferujesz wywołanie wszystkich funkcji błędów za każdym razem, gdy chcesz zgłosić błąd, czy nie, tworzenie tego samego kodu obsługi błędów rodzi pokusę zastosowania skrótu, a tym samym braku spójności w sposobie zgłaszania błędów. Umieszczenie całego kodu obsługi błędów w funkcji pomocniczej, którą można łatwo wywołać, zmniejsza wspomnianą pokusę i poprawia spójność kodu.
- Jeżeli kiedykolwiek zajdzie potrzeba zmiany formatu komunikatów błędów, znacznie łatwiej jest wprowadzić zmianę tylko w jednym miejscu niż w całym programie. Ewentualnie, jeśli zdecydujesz o zapisie błędów w pliku dziennika zdarzeń zamiast (lub równocześnie) przekazywania ich do `stderr`, wtedy będziesz musiał zmodyfikować jedynie funkcję `print_error()`. Takie podejście jest mniej podatne na błędy i zmniejsza pokusę stosowania skrótów, a tym samym wprowadzenia niekonsekwencji.
- Jeżeli podczas testowania programów korzystasz z debuggera, umieszczenie punktu kontrolnego w funkcji zgłaszającej błędy jest wygodnym sposobem wstrzymania działania programu i przejścia do debuggera po wykryciu błędu.

Z wymienionych powodów programy przedstawione w pozostałej części rozdziału, które sprawdzają błędy powiązane z MySQL, do ich zgłaszania wykorzystują funkcję `print_error()`.

Przedstawiony poniżej kod zawiera definicję funkcji `print_error()`, która oferuje omówione już wcześniej korzyści:

```
static void
print_error (MYSQL *conn, char *message)
{
    fprintf (stderr, "%s\n", message);
    if (conn != NULL)
    {
        fprintf (stderr, "Błąd %u (%s): %s\n",
                 mysql_errno (conn), mysql_sqlstate (conn), mysql_error (conn));
    }
}
```

W pliku kodu źródłowego *connect2.c* fragment wymagający sprawdzenia pod kątem błędów jest podobny do odpowiadającego mu kodu w *connect1.c* i wygląda następująco po użyciu `print_error()`:

```
/* Inicjalizacja uchwytu połączenia. */
conn = mysql_init (NULL);
if (conn == NULL)
{
    print_error (NULL, "Wywołanie mysql_init() zakończyło się niepowodzeniem
(prawdopodobnie z powodu braku pamięci).");
    exit (1);
}
/* Nawiązanie połączenia z serwerem. */
if (mysql_real_connect (conn, opt_host_name, opt_user_name, opt_password,
                        opt_db_name, opt_port_num, opt_socket_name, opt_flags) == NULL)
{
    print_error (conn, "Wywołanie mysql_real_connect() zakończyło się niepowodzeniem.");
    mysql_close (conn);
    exit (1);
}
```

Logika sprawdzania błędów opiera się na tym, że zarówno funkcja `mysql_init()`, jak i `mysql_real_connect()` zwraca wartość `NULL` w przypadku niepowodzenia. Jeżeli wywołanie `mysql_init()` zakończy się niepowodzeniem, wartość `NULL` jest przekazywana jako pierwszy argument funkcji `print_error()`. To nie spowoduje wywołania oferowanej przez MySQL funkcji zgłaszania błędów, ponieważ uchwyt połączenia przekazywany wspomnianym funkcjom nie zawiera żadnych sensownych informacji. Z kolei, jeśli działanie funkcji `mysql_real_connect()` zakończy się niepowodzeniem, uchwyt połączenia zostaje przekazany funkcji `print_error()`. Wprawdzie ten uchwyt nie będzie zawierał informacji odpowiadających poprawnemu połączeniu, ale zawiera informacje diagnostyczne, które mogą być wyodrębnione przez funkcje zgłaszające błędy. Wspomniany uchwyt może być również przekazany funkcji `mysql_close()` w celu zwolnienia pamięci automatycznie zaalokowanej przez `mysql_init()`. (Tego uchwytu nie przekazuj innym procedurom klienta! Większość z nich przyjmuje założenie, że otrzyma poprawny uchwyt połączenia. Gdy otrzymają nieprawidłowy, program może ulec awarii).

Pozostałe programy w rozdziale przeprowadzają sprawdzanie pod kątem błędów; to samo powinieneś stosować we własnych programach. Na dłuższą metę przynosi to korzyść, ponieważ poświęcisz mniej czasu na wyszukiwanie niejasnych problemów w kodzie.

7.3.2. Pobieranie parametrów połączenia w trakcie działania programu

Możemy teraz przystąpić do wykonania zadania polegającego na umożliwieniu użytkownikom podania parametrów połączenia w trakcie działania programu, zamiast używać parametrów domyślnych na stałe zdefiniowanych w kodzie. Program `connect1` miał poważne ograniczenie polegające właśnie na tym, że parametry połączenia były na stałe zdefiniowane w kodzie źródłowym. Aby zmienić te wartości, konieczne jest przeprowadzenie edycji pliku źródłowego, a następnie jego ponownej kompilacji. To nie jest zbyt wygodne rozwiązanie, zwłaszcza jeśli program ma zostać udostępniony innym użytkownikom. Jednym z powszechnie stosowanych sposobów podania parametrów w trakcie działania programu jest użycie opcji wiersza poleceń. Na przykład, programy dostarczane w dystrybucji MySQL akceptują parametry w dwóch postaciach wymienionych w tabeli 7.1.

Tabela 7.1. Dwie formy parametrów akceptowanych przez standardowe klienty MySQL

Parametr	Długa forma opcji	Krótką forma opcji
Nazwa komputera	<code>--host=nazwa_komputera</code>	<code>-h nazwa_komputera</code>
Nazwa użytkownika	<code>--user=nazwa_użytkownika</code>	<code>-u nazwa_użytkownika</code>
Hasło	<code>--password</code> lub <code>--password=hasło</code>	<code>-p</code> lub <code>-phasło</code>
Numer portu	<code>--port=numer_portu</code>	<code>-P numer_portu</code>
Nazwa gniazda	<code>--socket=nazwa_gniazda</code>	<code>-S nazwa_gniazda</code>

W celu zachowania zgodności ze standardowymi klientami MySQL nasz program `connect2` również będzie akceptował te same formaty. To jest bardzo łatwe do wykonania, ponieważ biblioteka klienta obsługuje przetwarzanie opcji. Ponadto, program `connect2` ma możliwość wyodrębniania informacji z plików opcji. W ten sposób zyskujesz możliwość umieszczenia parametrów połączenia w pliku `~/.my.cnf` (to znaczy w pliku `.my.cnf` w katalogu domowym) lub w dowolnym globalnym pliku opcji. Wtedy nie będzie potrzeby podawania opcji w wierszu poleceń w trakcie każdego uruchamiania programu. Biblioteka klienta znacznie ułatwia sprawdzanie plików opcji MySQL i pobieranie z nich wymaganych informacji. Dzięki dodaniu do programu jedynie kilku wierszy kodu możesz zapewnić mu obsługę pliku opcji bez konieczności wyważania otwartych drzwi i samodzielnego tworzenia kodu do tego celu. (Omówienie składni pliku opcji znajdziesz w punkcie F.2.2, zatytułowanym „Pliki opcji”).

Zanim przejdziemy do omawiania sposobu przetwarzania opcji w programie `connect2`, utworzymy dwa programy prezentujące ogólnie koncepcje, których będziemy używać. Wspomniane programy pokażą, że obsługa opcji działa względnie łatwo i bez nadmiernego komplikowania operacji połączenia z serwerem MySQL i przetwarzania zapytań.

Uwaga

MySQL oferuje dwie opcje mające związek z operacją nawiązywania połączenia. Pierwsza z nich, `--protocol`, określa protokół połączenia (TCP/IP, plik gniazda systemu UNIX itd.), natomiast druga, `--shared-memory-base-name`, określa nazwę pamięci współdzielonej używanej w Windows dla połączeń współdzielących pamięć. W tym rozdziale wymienione opcje nie będą omówione, ale jeśli jesteś nimi zainteresowany, to dystrybucja `sampdb` zawiera kod źródłowy programu `protocol`, pokazujący, jak można ich używać.

7.3.2.1. Uzyskanie dostępu do pliku opcji

W celu odczytania pliku opcji i pobrania wartości parametru połączenia konieczne jest wywołanie funkcji `load_defaults()`. Wymieniona funkcja szuka pliku opcji, przetwarza jego zawartość pod kątem dowolnej grupy opcji oraz modyfikuje wektor argumentów programu (tablicę `argv[]`). Informacje pobrane dla wskazanych grup opcji są umieszczane na początku tablicy `argv[]` w formie opcji wiersza poleceń. W ten sposób opcje pojawiają się tak, jak podane w wierszu poleceń. Podczas przetwarzania opcji polecenia parametry połączenia będą dostępne dla zwykłego kodu przetwarzającego opcje. Opcje są dodawane do `argv[]` tuż po nazwie polecenia i przed innymi argumentami (a nie na końcu). Dlatego też wszelkie parametry połączenia podane w wierszu poleceń pojawiają się później na liście opcji, a tym samym nadpisują opcje dodane przez funkcję `load_defaults()`.

Poniżej przedstawiono niewielki program `show_argv`, pokazujący użycie funkcji `load_defaults()` oraz sposób modyfikacji wspomnianego wcześniej wektora argumentów:

```
/*
 * show_argv.c - Program pokazuje użycie funkcji load_defaults() i jej wpływ na tablicę argumentów.
 */

#include <my_global.h>
#include <my_sys.h>
#include <mysql.h>

static const char *client_groups[] = { "client", NULL };

int main (int argc, char *argv[])
{
    int i;

    printf ("Początkowa tablica argumentów:\n");
    for (i = 0; i < argc; i++)
        printf ("arg %d: %s\n", i, argv[i]);

    MY_INIT (argv[0]);
    load_defaults ("my", client_groups, &argc, &argv);

    printf ("Zmodyfikowana tablica argumentów:\n");
    for (i = 0; i < argc; i++)
        printf ("arg %d: %s\n", i, argv[i]);

    exit (0);
}
```

Kod przetwarzający plik opcji składa się z kilku komponentów:

- `client_groups[]` to tablica ciągów tekstowych wskazujących nazwy grup w pliku opcji, z których chcesz pobrać opcje. Program klienta zwykle zawiera co najmniej `client` na tej liście (odpowiada grupie `[client]`), ale możesz wymienić dowolną liczbę grup. Ostatnim elementem tablicy musi być wartość `NULL`, wskazująca koniec listy.
- `MY_INIT()` to makro inicjalizujące, którego już wcześniej używaliśmy. Ważne jest to, że makro `MY_INIT()` wywołuje funkcję `my_init()` w celu przeprowadzenia pewnych operacji konfiguracyjnych wymaganych do działania funkcji `load_defaults()`.
- Funkcja `load_defaults()` odczytuje plik opcji. Pobiera cztery argumenty: prefiks używany w nazwach pliku opcji (to zawsze powinien być `my`), tablicę zawierającą nazwy interesujących Cię grup opcji oraz adresy liczby argumentów programu i wektora argumentów. Nie przekazuj wartości liczby argumentów i wektora argumentów, zamiast tego przekaz ich adresy, ponieważ funkcja `load_defaults()` będzie musiała zmienić wartości wymienionych argumentów. W szczególności dotyczy to `argv` — wprawdzie jest wskaźnikiem, ale mimo tego powinienśes przekazać `&argv`, czyli adres wskaźnika.

Program `show_argv` dwukrotnie wyświetla argumenty, aby w ten sposób pokazać wpływ, jaki na tablicę argumentów ma wywołanie funkcji `load_defaults()`. Na początku zostają wyświetlone argumenty w postaci, w jakiej podano je w wierszu poleceń. Następnie program wywołuje funkcję `load_defaults()` i ponownie wyświetla tablicę argumentów.

Aby przekonać się, jak działa funkcja `load_defaults()`, upewnij się o umieszczeniu pliku `.my.cnf` w katalogu domowym. Oczywiście, wymieniony plik powinien mieć pewne opcje w grupie `[client]`. (W systemie Windows można użyć pliku `C:\my.ini`). Przyjmujemy założenie, że zawartość pliku przedstawia się następująco:

```
[client]
user=sampadm
password=secret
host=pewien_komputer
```

W takim przypadku uruchomienie programu `show_argv` powinno wygenerować przedstawione poniżej dane wyjściowe:

```
% ./show_argv a b
Początkowa tablica argumentów:
arg 0: ./show_argv
arg 1: a
arg 2: b
Zmodyfikowana tablica argumentów:
arg 0: ./show_argv
arg 1: --user=sampadm
arg 2: --password=secret
arg 3: --host=pewien_komputer
arg 4: a
arg 5: b
```


W trakcie drugiego wyświetlenia tablicy argumentów wartości pobrane z pliku opcji są pokazane jako część listy argumentów. Istnieje możliwość, że zobaczysz pewne opcje, które nie zostały zdefiniowane w wierszu poleceń lub pliku `~/my.cnf`. W takim przypadku wspomniane opcje zostały prawdopodobnie zdefiniowane w grupie `[client]` globalnego pliku opcji. Dzieje się tak, ponieważ działanie funkcji `load_defaults()` polega na wyszukiwaniu pliku opcji w wielu miejscach. (Dokładną listę wspomnianych miejsc znajdziesz w punkcie F.2.2, zatytułowanym „Pliki opcji”).

Programy klientów używających funkcji `load_defaults()` zwykle zawierają `client` na liście nazw grup opcji (a więc z plików opcji pobierają wszystkie ogólne ustawienia klienta). Kod odpowiedzialny za przetwarzanie pliku opcji możesz przygotować do przetwarzania także opcji pochodzących z innych grup. Przyjmujemy założenie, że program `show_argv` ma odczytać opcje z grup `[client]` i `[show_argv]`. Aby wykonać to zadanie, w pliku kodu źródłowego `show_argv.c` odszukaj poniższy wiersz:

```
const char *client_groups[] = { "client", NULL };
```

i zamień go na następujący:

```
const char *client_groups[] = { "show_argv", "client", NULL };
```

a następnie ponownie skompiluj program `show_argv`. Zmodyfikowany program będzie odczytywał opcje z obu wymienionych grup. Jeśli chcesz się o tym przekonać, dodaj grupę `[show_argv]` do pliku `~/my.cnf`:

```
[client]
user=sampadm
password=secret
host=pewien_komputer
```

```
[show_argv]
host=innny_komputer
```

Po wprowadzeniu powyższych zmian ponowne uruchomienie programu `show_argv` powoduje wygenerowanie innych danych wyjściowych niż poprzednio:

```
% ./show_argv a b
Początkowa tablica argumentów:
arg 0: ./show_argv
arg 1: a
arg 2: b
Zmodyfikowana tablica argumentów:
arg 0: ./show_argv
arg 1: --user=sampadm
arg 2: --password=secret
arg 3: --host=pewien_komputer
arg 4: --host=innny_komputer
arg 5: a
arg 6: b
```

Na kolejność wartości opcji w tablicy argumentów wpływ ma kolejność, w jakiej zostały podane w pliku opcji, a nie kolejność wymienienia nazw grup w tablicy `client_groups[]`. Oznacza to, że w pliku opcji grupy charakterystyczne dla programu prawdopodobnie będziesz umieszczał po grupie `[client]`. W ten sposób, jeśli opcja zostanie wymieniona

w obu grupach, wartości przeznaczone dla programu będą miały pierwszeństwo przed wartościami zdefiniowanymi w ogólnej grupie `[client]`. Możesz się o tym przekonać w przedstawionym powyżej przykładzie: opcja `host` została wymieniona w obu grupach (`[client]` i `[show_argv]`), ale ponieważ grupa `[show_argv]` znajduje się na końcu pliku opcji, to jej ustawienie opcji `host` umieszczone jest na dalszym miejscu w tablicy argumentów, a tym samym ma pierwszeństwo.

Funkcja `load_defaults()` nie pobiera wartości z ustawień środowiska. Aby użyć wartości zmiennych środowiskowych, takich jak `MYSQL_TCP_PORT` lub `MYSQL_UNIX_PORT`, konieczne jest ich samodzielne pobranie za pomocą funkcji `getenv()`. Tego rodzaju funkcjonalności nie dodamy do omawianych tutaj klientów, ale poniżej przedstawiono krótki fragment kodu pokazujący, jak sprawdzić wartość dwóch zmiennych środowiskowych powiązanych z MySQL:

```
extern char *getenv();
char *p;
int port_num = 0;
char *socket_name = NULL;

if ((p = getenv ("MYSQL_TCP_PORT")) != NULL)
    port_num = atoi (p);
if ((p = getenv ("MYSQL_UNIX_PORT")) != NULL)
    socket_name = p;
```

W standardowych klientach MySQL wartości zmiennych środowiskowych mają mniejsze pierwszeństwo niż wartości podane w plikach opcji lub wierszu poleceń. Aby sprawdzić zmienne środowiskowe we własnych programach i zachować spójność z konwencją, wspomniane sprawdzenie środowiska przeprowadzaj przed wywołaniem funkcji `load_defaults()` (ale nie po) lub przed przetwarzaniem opcji wiersza poleceń.

Funkcja `load_defaults()` i bezpieczeństwo

W systemach wielodostępnych narzędzia takie jak program `ps` mogą wyświetlać listę argumentów dowolnych procesów, w tym także procesów uruchomionych przez innych użytkowników. Z tego powodu możesz się zastanawiać, czy istnieją jakiegokolwiek implikacje związane z przetwarzaniem przez funkcję `load_defaults()` haseł znalezionych w plikach opcji i umieszczanych później na liście argumentów. W rzeczywistości to nie stanowi problemu, ponieważ program `ps` wyświetla początkową zawartość tablicy `argv[]`. Jakikolwiek argument hasła utworzony przez wywołanie funkcji `load_defaults()` prowadzi do obszaru pamięci zaalokowanego przez funkcję. Ten obszar pamięci nie jest częścią oryginalnej tablicy, a więc program `ps` nawet go nie widzi.

Z drugiej strony, hasło podane w wierszu poleceń **będzie** wyświetlone przez program `ps` i dlatego podawanie haseł w taki sposób nie jest dobrym rozwiązaniem. Jedynym zabezpieczeniem, które może zastosować program w celu zmniejszenia niebezpieczeństwa, jest usunięcie hasła z listy argumentów tuż po uruchomieniu. W kolejnym podpunkcie dowiesz się, jak można to zrobić.

7.3.2.2. Przetwarzanie argumentów wiersza poleceń

Za pomocą funkcji `load_defaults()` można pobrać wszystkie parametry połączenia i umieścić je w tablicy argumentów. Potrzebujemy jednak sposobu na przetworzenie wspomnianej tablicy. Do tego celu została zaprojektowana funkcja `handle_options()`, która jest częścią biblioteki klienta MySQL. Jeśli masz dostęp do biblioteki klienta, wówczas masz również dostęp do wymienionej funkcji.

Poniżej wymieniono pewne cechy charakterystyczne oferowanych przez bibliotekę klienta procedur przetwarzania opcji:

- Precyzyjna specyfikacja typu opcji i zakresu dozwolonych wartości. Na przykład, można wskazać nie tylko, że opcja ma mieć wartości w postaci liczb całkowitych, ale również ich zakres (na przykład liczby dodatnie będące wielokrotnością 1024).
- Integracja z tekstem pomocy w celu łatwego wyświetlania komunikatu pomocy podczas wywołania funkcji biblioteki. Nie musisz tworzyć własnego kodu przeznaczonego do wyświetlania komunikatów pomocy.
- Wbudowana obsługa standardowych opcji `--no-defaults`, `--print-defaults`, `--defaults-file` i `--defaults-extra-file`. (Wymienione opcje zostały omówione w punkcie F.2.2, zatytułowanym „Pliki opcji”).
- Obsługa standardowego zestawu prefiksów opcji, takich jak `--disable-`, `--enable-` i `--loose-`, w celu ułatwienia implementacji opcji boolowskich (włączona/wyłączona) i ignorowanych. (Ta możliwość nie zostanie omówiona w tym rozdziale, ale odpowiednie informacje znajdziesz w podrozdziale F.2, zatytułowanym „Określanie opcji programu”).

Aby zademonstrować użycie oferowanych przez MySQL możliwości w zakresie obsługi opcji, w tym miejscu utworzymy program `show_opt`, używający funkcji `load_defaults()` do wczytania pliku opcji i konfiguracji tablicy argumentów, a następnie przetwarzający wynik za pomocą funkcji `handle_options()`.

Program `show_opt` pozwala na eksperymenty z różnymi sposobami podawania parametrów połączenia (za pomocą plików opcji lub wiersza poleceń), a także zobaczenie wyniku tych eksperymentów. Program wyświetla wartości, jakie zostały użyte do nawiązania połączenia z serwerem MySQL. Omawiany program jest użyteczny, ponieważ pokazuje, jak będzie działał nasz kolejny program klienta (`connect2`) po umieszczeniu w nim kodu przetwarzającego opcje i połączeniu z kodem faktycznie nawiązującym połączenie z serwerem.

W celu prezentacji zdarzeń zachodzących na każdym etapie przetwarzania argumentu program `show_opt` wykonuje następujące operacje:

1. Konfiguracja wartości domyślnych dla nazwy komputera, nazwy użytkownika, hasła oraz innych parametrów połączenia.
2. Wyświetlenie wartości początkowego parametru połączenia i tablicy argumentów.
3. Wywołanie funkcji `load_defaults()` i zmiana wektora połączenia w celu odzwierciedlenia zawartości pliku opcji, a następnie wyświetlenie zmodyfikowanej tablicy.

4. Wywołanie funkcji `handle_options()` odpowiedzialnej za przetwarzanie opcji i przetworzenie tablicy argumentów, a następnie wyświetlenie otrzymanych w wyniku zmian wartości parametrów połączenia i zawartości tablicy argumentów.

Wkrótce dowiesz się, jak działa program `show_opt`, ale najpierw zapoznaj się z jego kodem źródłowym umieszczonym w pliku `show_opt.c`:

```
/*
 * show_opt.c - Program demonstruje przetwarzanie opcji za pomocą funkcji load_defaults() i handle_options().
 */

#include <my_global.h>
#include <my_sys.h>
#include <mysql.h>
#include <my_getopt.h>

static char *opt_host_name = NULL;    /* Komputer serwera (domyślnie= localhost). */
static char *opt_user_name = NULL;    /* Nazwa użytkownika (domyślnie = nazwa logowania). */
static char *opt_password = NULL;     /* Hasło (domyślnie = brak). */
static unsigned int opt_port_num = 0;  /* Numer portu (użyj wbudowanej wartości). */
static char *opt_socket_name = NULL;  /* Nazwa gniazda (użyj wbudowanej wartości). */

static const char *client_groups[] = { "client", NULL };

static struct my_option my_opts[] = /* Struktury informacji opcji. */
{
    {"help", '?', "Wyświetlenie tego komunikatu pomocy i zakończenie programu.",
     NULL, NULL, NULL,
     GET_NO_ARG, NO_ARG, 0, 0, 0, 0, 0, 0},
    {"host", 'h', "Komputer, z którym chcesz się połączyć.",
     (uchar **) &opt_host_name, NULL, NULL,
     GET_STR, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
    {"password", 'p', "Hasło.",
     (uchar **) &opt_password, NULL, NULL,
     GET_STR, OPT_ARG, 0, 0, 0, 0, 0, 0},
    {"port", 'P', "Numer portu.",
     (uchar **) &opt_port_num, NULL, NULL,
     GET_UINT, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
    {"socket", 'S', "Ścieżka dostępu do gniazda.",
     (uchar **) &opt_socket_name, NULL, NULL,
     GET_STR, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
    {"user", 'u', "Nazwa użytkownika.",
     (uchar **) &opt_user_name, NULL, NULL,
     GET_STR, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
    { NULL, 0, NULL, NULL, NULL, NULL, GET_NO_ARG, NO_ARG, 0, 0, 0, 0, 0, 0 }
};

static my_bool
get_one_option (int optid, const struct my_option *opt, char *argument)
{
    switch (optid)
    {
    case '?':
        my_print_help (my_opts); /* Wyświetlenie komunikatu pomocy. */
        exit (0);
    }
    return (0);
}
```

```

}

int main (int argc, char *argv[])
{
    int i;
    int opt_err; printf ("Początkowe parametry połączenia:\n");
    printf ("nazwa komputera: %s\n", opt_host_name ? opt_host_name : "(null)");
    printf ("nazwa użytkownika: %s\n", opt_user_name ? opt_user_name : "(null)");
    printf ("hasło: %s\n", opt_password ? opt_password : "(null)");
    printf ("numer portu: %u\n", opt_port_num);
    printf ("nazwa pliku gniazda: %s\n",
            opt_socket_name ? opt_socket_name : "(null)");

    printf ("Początkowa tablica argumentów:\n");
    for (i = 0; i < argc; i++)
        printf ("arg %d: %s\n", i, argv[i]);

    MY_INIT (argv[0]);
    load_defaults ("my", client_groups, &argc, &argv);
    printf ("Tablica argumentów po wywołaniu funkcji load_defaults():\n");
    for (i = 0; i < argc; i++)
        printf ("arg %d: %s\n", i, argv[i]);

    if ((opt_err = handle_options (&argc, &argv, my_opts, get_one_option)))
        exit (opt_err);

    printf ("Parametry połączenia po wywołaniu funkcji handle_options():\n");
    printf ("nazwa komputera: %s\n", opt_host_name ? opt_host_name : "(null)");
    printf ("nazwa użytkownika: %s\n", opt_user_name ? opt_user_name : "(null)");
    printf ("hasło: %s\n", opt_password ? opt_password : "(null)");
    printf ("numer portu: %u\n", opt_port_num);
    printf ("nazwa pliku gniazda: %s\n",
            opt_socket_name ? opt_socket_name : "(null)");

    printf ("Tablica argumentów po wywołaniu funkcji handle_options():\n");
    for (i = 0; i < argc; i++)
        printf ("arg %d: %s\n", i, argv[i]);

    exit (0);
}

```

Zastosowane w kodzie źródłowym *show_opt.c* podejście w zakresie przetwarzania opcji obejmuje wiele aspektów, które są wspólne dla wszystkich programów używających biblioteki klienta MySQL w celu obsługi opcji polecenia. W samodzielnie tworzonych programach powinieneś stosować te same rozwiązania:

1. Oprócz innych plików, które dołączaliśmy już we wcześniejszych programach, tutaj dołączany jest także *my_getopt.h*. Wymieniony plik definiuje interfejs możliwości MySQL w zakresie przetwarzania opcji.
2. Definicja struktur *my_option*. W kodzie źródłowym *show_opt.c* ta tablica ma nazwę *my_opts*. Tablica powinna mieć po jednej strukturze dla danej opcji rozpoznawanej przez program. Każda struktura zapewnia informacje takie jak krótka i długa nazwa opcji, jej wartość domyślna, typ opcji (ciąg tekstowy lub liczba) itd.
3. Po wywołaniu funkcji *load_defaults()* w celu odczytania pliku opcji i konfiguracji tablicy argumentów następuje wywołanie funkcji *handle_options()* w celu

przetworzenia opcji. Pierwsze dwa argumenty funkcji `handle_options()` to adresy wartości określających liczbę argumentów i tablicę argumentów programu. (Podobnie jak w przypadku funkcji `load_defaults()`, także tutaj należy przekazać adresy zmiennych, a nie ich wartości). Trzeci argument prowadzi do tablicy struktur `my_option`. Czwarty argument jest wskaźnikiem do funkcji pomocniczej. Funkcja `handle_options()` i struktury `my_option` zostały zaprojektowane w celu umożliwienia automatycznego wykonania przez bibliotekę klienta większości operacji związanych z przetwarzaniem opcji. Jednak aby pozwolić na wykonanie operacji specjalnych nieobsługiwanych przez bibliotekę, program powinien zawierać także funkcję pomocniczą dla wywołania `handle_options()`. W kodzie `show_opt.c` wspomniana funkcja nosi nazwę `get_one_option()`.

Struktura `my_option` definiuje typy informacji, które muszą być podane dla każdej opcji rozpoznawanej przez program:

```
struct my_option
{
    const char *name;           /* Długa nazwa opcji. */
    int id;                     /* Krótka nazwa opcji lub jej kod. */
    const char *comment;        /* Opis opcji wyświetlany w komunikacie pomocy. */
    void *value;                /* Wskaźnik do zmiennej przechowującej wartość. */
    void *u_max_value;          /* Maksymalna wartość zmiennej zdefiniowanej przez użytkownika. */
/*
    struct st_typelib *typelib; /* Wskaźnik do dozwolonych wartości (nieużyty). */
    ulong var_type;             /* Typ wartości opcji. */
    enum get_opt_arg_type arg_type; /* Określenie, czy wartość opcji jest wymagana. */
    longlong def_value;         /* Wartość domyślna opcji. */
    longlong min_value;         /* Minimalna dozwolona wartość opcji. */
    longlong max_value;         /* Maksymalna dozwolona wartość opcji. */
    longlong sub_size;          /* Wielkość przesunięcia wartości. */
    long block_size;            /* Opcjonalny mnożnik wartości. */
    void *app_type;             /* Zarezerwowane dla specyficznych potrzeb aplikacji. */
};
```

Poniżej wymieniono elementy struktury `my_option`:

- `name` to długa nazwa opcji. To jest forma opcji w postaci `--name`, ale bez myślników na początku. Na przykład, jeżeli długa forma opcji to `--user`, wtedy w strukturze `my_option` zostanie zapisana jako `user`.
- `id` to krótka (w postaci pojedynczej litery) nazwa opcji lub kod przypisany opcji, jeśli nie podano wspomnianej nazwy. Na przykład, jeśli krótka nazwa opcji to `-u`, wtedy w strukturze `my_option` zostanie zapisana jako `u`. Dla opcji posiadających jedynie długie nazwy i pozbawionych odpowiadających im nazw w postaci pojedynczej litery powinienś ustawić kod opcji, który będzie wewnętrznie używany jako jej krótka nazwa. Wspomniany kod musi być unikalny i różnić się od wszystkich nazw w postaci pojedynczej litery. (Aby spełnić to drugie wymaganie, upewnij się, że kod będzie miał wartość większą niż 255, czyli niż największa wartość dozwolona dla pojedynczego znaku. Odpowiednie przykłady znajdziesz w podrozdziale 7.6, zatytułowanym „Utworzenie klienta z obsługą SSL”).

- `comment` to ciąg tekstowy opisujący przeznaczenie opcji. Zdefiniowany tutaj tekst będzie wyświetlony w komunikacie pomocy.
- `value` to adres ogólnego wskaźnika, zadeklarowany jako `uchar **`. Jeżeli opcja pobiera argument, `value` prowadzi wtedy do zmiennej, w której można przechowywać ten argument. Po przetworzeniu opcji można sprawdzić wartość tej zmiennej i przekonać się, jak była ustawiona opcja. Typ danych zmiennej musi być zgodny z wartością elementu `var_type`. Jeżeli opcja nie pobiera argumentu, wówczas wartością `value` jest `NULL`.
- `u_max_value` to kolejny adres ogólnego wskaźnika, ale używany jedynie przez serwer. Dla programów klienckich wartość `u_max_value` powinna wynosić `NULL`.
- `type1ib` to aktualnie nieużywany element. W przyszłych wydaniach MySQL może być używany do zdefiniowania listy dozwolonych wartości. W takim przypadku podana wartość opcji będzie musiała być dopasowana do jednej z wartości podanych w tym elemencie.
- `var_type` wskazuje rodzaj wartości; musi się znajdować po nazwie opcji w wierszu poleceń. W tabeli 7.2 wymieniono te typy, ich opis oraz odpowiadające im typy w języku C.

Tabela 7.2. Typy wartości podawanych opcji w wierszu poleceń

Wartość <code>var_type</code>	Opis	Typ C
<code>GET_NO_ARG</code>	Brak wartości.	
<code>GET_BOOL</code>	Wartość boolowska.	<code>my_bool</code>
<code>GET_INT</code>	Wartość w postaci liczby całkowitej.	<code>int</code>
<code>GET_UINT</code>	Wartość w postaci liczby całkowitej bez znaku.	<code>unsigned int</code>
<code>GET_LONG</code>	Wartość w postaci długiej (<code>long</code>) liczby całkowitej.	<code>long</code>
<code>GET_ULONG</code>	Wartość w postaci długiej (<code>long</code>) liczby całkowitej bez znaku.	<code>unsigned long</code>
<code>GET_LL</code>	Wartość w postaci długiej (<code>long long</code>) liczby całkowitej.	<code>long long</code>
<code>GET_ULL</code>	Wartość w postaci długiej (<code>long long</code>) liczby całkowitej bez znaku.	<code>unsigned long long</code>
<code>GET_STR</code>	Wartość w postaci ciągu tekstowego.	<code>char *</code>
<code>GET_STR_ALLOC</code>	Wartość w postaci ciągu tekstowego.	<code>char *</code>
<code>GET_DISABLED</code>	Opcja jest wyłączona.	
<code>GET_ENUM</code>	Wartość w postaci typu wyliczeniowego (aktualnie nieużywana).	
<code>GET_SET</code>	Wartość w postaci zbioru (aktualnie nieużywana).	
<code>GET_DOUBLE</code>	Wartość w postaci liczby o podwójnej precyzji (zmiennoprzecinkowej).	<code>double</code>

Różnica między `GET_STR` i `GET_STR_ALLOC` polega na tym, że w przypadku `GET_STR` biblioteka klienta ustawia zmienną opcji w taki sposób, aby prowadziła bezpośrednio do wartości w tablicy argumentów. Z kolei w przypadku `GET_STR_ALLOC` tworzona jest kopia argumentu, do której prowadzi później zmienna opcji.

Typ `GET_DISABLED` może być użyty do wskazania, że opcja nie jest już dostępna lub jest dostępna jedynie po utworzeniu programu w określony sposób (na przykład z włączoną obsługą debuggowania). Przykład użycia tego typu możesz zobaczyć w pliku *mysql.cc* znajdującym się w dystrybucji kodu źródłowego MySQL.

- `arg_type` wskazuje, czy po nazwie opcji znajduje się wartość oraz czy jest dowolną z wartości wymienionych w tabeli 7.3. Jeżeli wartością `arg_type` jest `NO_ARG`, wtedy wartością `var_type` powinno być `GET_NO_ARG`.

Tabela 7.3. Wartości `arg_type`

Wartość <code>arg_type</code>	Opis
<code>NO_ARG</code>	Opcja nie pobiera argumentu.
<code>OPT_ARG</code>	Opcja może pobrać argument.
<code>REQUIRED_ARG</code>	Opcja wymaga argumentu.

- `def_value` — ten element jest przeznaczony dla opcji liczbowych. To jest wartość domyślna przypisywana opcji, jeśli w tablicy argumentów nie zostanie wyrażnie podana wartość dla tej opcji.
- `min_value` — ten element jest przeznaczony dla opcji liczbowych. To jest najmniejsza dozwolona wartość. Mniejsze wartości będą automatycznie podniesione do zdefiniowanej tutaj minimalnej. Wartość 0 oznacza „brak minimum”.
- `max_value` — ten element jest przeznaczony dla opcji liczbowych. To jest największa dozwolona wartość. Większe wartości będą automatycznie zmniejszone do zdefiniowanej tutaj maksymalnej. Wartość 0 oznacza „brak maksimum”.
- `sub_sizes` — ten element jest przeznaczony dla opcji liczbowych. Wskazuje wartość przesunięcia używaną podczas konwersji wartości z zakresu podanego w tablicy argumentów na zakres używany wewnętrznie. Na przykład, jeśli wartości podawane w wierszu poleceń są z zakresu od 1 do 256, natomiast program chce wewnętrznie używać zakresu od 0 do 255, wtedy wartość `sub_size` powinna wynieść 1.
- `block_size` — ten element jest przeznaczony dla opcji liczbowych. Ta wartość wskazuje wielkość bloku, jeśli jest niezerowy. Podawane przez użytkownika wartości opcji są zaokrąglane w dół do najbliższej wielokrotności tej wielkości, o ile zachodzi potrzeba. Na przykład, jeśli wartości muszą być parzyste, wtedy

ustaw wielkość bloku na 2. Funkcja `handle_options()` zaokrągli wartość nieparzystą do najbliższej wartości parzystej.

- `app_type` — ten element jest zarezerwowany na specjalne potrzeby aplikacji.

Tablica `my_opts` powinna mieć strukturę `my_option` dla każdej prawidłowej opcji, a na końcu strukturę skonfigurowaną w poniższy sposób, wskazujący na koniec tablicy:

```
{ NULL, 0, NULL, NULL, NULL, NULL, GET_NO_ARG, NO_ARG, 0, 0, 0, 0, 0 }
```

Kiedy wywołujesz funkcję `handle_options()` do przetworzenia tablicy argumentów, funkcja pomija pierwszy argument, jakim jest nazwa programu, a następnie przetwarza argumenty opcji, to znaczy argumenty rozpoczynające się od znaku minus. Następnie kontynuuje działanie aż do przetworzenia wszystkich elementów tablicy lub napotkania specjalnego argumentu w postaci dwóch znaków minus (--), oznaczającego „koniec opcji”. Gdy funkcja `handle_options()` porusza się po tablicy argumentów, wtedy dla każdej opcji jednokrotnie wywołuje funkcję pomocniczą w celu umożliwienia jej przeprowadzenia wszelkich operacji specjalnych. Funkcja `handle_options()` przekazuje trzy argumenty funkcji pomocniczej: krótką nazwę opcji, wskaźnik do struktury `my_option` danej opcji oraz wskaźnik do argumentu znajdującego się po opcji w tablicy argumentów (ten wskaźnik będzie miał wartość `NULL`, jeśli opcja zostanie użyta bez żadnej wartości).

Po zakończeniu działania funkcji `handle_options()` liczba argumentów i tablica argumentów są odpowiednio zerowane, aby zawierały jedynie listę argumentów bez opcji.

Poniżej przedstawiono przykładowe wywołanie programu `show_opt` i wygenerowane przez niego dane wyjściowe (przyjęto założenie, że plik `~/my.cnf` ma taką samą zawartość jak w ostatnim przykładzie programu `show_argv` w podpunkcie 7.3.2.1, zatytułowanym „Uzyskanie dostępu do pliku opcji”):

```
% ./show_opt -h jeszcze_inny_komputer --user=bartek x
Początkowe parametry połączenia:
nazwa komputera: (null)
nazwa użytkownika: (null)
hasło: (null)
numer portu: 0
nazwa pliku gniazda: (null)
Początkowa tablica argumentów:
arg 0: ./show_opt
arg 1: -h
arg 3: jeszcze_inny_komputer
arg 3: --user=bartek
arg 4: x
Tablica argumentów po wywołaniu funkcji load_defaults():
arg 0: ./show_opt
arg 1: --user=sampadm
arg 2: --password=secret
arg 3: --host=pewien_komputer
arg 4: -h
arg 5: jeszcze_inny_komputer
arg 6: --user=bartek
arg 7: x
Parametry połączenia po wywołaniu funkcji handle_options():
nazwa komputera: jeszcze_inny_komputer
```

```

nazwa_uzytkownika: bartek
haslo: secret
numer_portu: 0
nazwa_pliku_gniazda: (null)
Tablica argumentów po wywołaniu funkcji handle_options():
arg 0: x

```

Dane wyjściowe pokazują, że nazwa komputera została pobrana z wiersza poleceń (nadpisując tym samym wartość z pliku opcji), natomiast nazwa użytkownika i hasło pochodzą z pliku opcji. Funkcja `handle_options()` prawidłowo przetwarza opcje podane zarówno w krótkiej formie (na przykład `-h jeszcze_inny_komputer`), jak i w długiej (na przykład `--user=bartek`).

Funkcja pomocnicza `get_one_option()` jest używana w połączeniu z `handle_options()`. W programie `show_opt` jej działanie jest minimalne i dotyczy jedynie opcji `--help` lub `-?` (dla której funkcja `handle_options()` przekazuje elementowi `optid` wartość `?`):

```

static my_bool
get_one_option (int optid, const struct my_option *opt, char *argument)
{
    switch (optid)
    {
        case '?':
            my_print_help (my_opts); /* Wyświetlenie komunikatu pomocy. */
            exit (0);
        }
    return (0);
}

```

Funkcja `my_print_help()` to oferowana przez bibliotekę klienta procedura, która automatycznie generuje komunikat pomocy na podstawie nazwy opcji i ciągu tekstowego zdefiniowanych w tablicy `my_opts`. Aby przekonać się, jak działa, wydaj poniższe polecenie:

```
% ./show_opt --help
```

Istnieje możliwość dodania kolejnych elementów do polecenia `switch()` w funkcji `get_one_option()`, o ile wystąpi potrzeba (takie rozwiązanie wkrótce zastosujemy w programie `connect2`). Na przykład, funkcja `get_one_option()` jest użyteczna do obsługi opcji hasła. Po podaniu tego rodzaju opcji wartość hasła może, choć nie musi być podana, na co wskazuje `OPT_ARG` w strukturze informacyjnej opcji. Oznacza to możliwość użycia opcji w formie `--password` lub `--password=haslo` (jeśli stosujesz długie nazwy opcji) bądź też w formie `-p` lub `-phaslo` (jeśli stosujesz krótkie nazwy opcji). Klient MySQL zwykle pozwala na pominięcie wartości hasła w wierszu poleceń, co uniemożliwia innym użytkownikom poznanie Twojego hasła. W późniejszych programach użyjemy funkcji `get_one_option()` w celu sprawdzenia, czy użytkownik podał hasło. Jeśli tak, wartość zostanie zapisana, natomiast w przeciwnym razie nastąpi ustawienie flagi wskazującej, że program powinien wyświetlić użytkownikowi pytanie o hasło, zanim podejmie próbę nawiązania połączenia z serwerem.

Pouczające może być, jeśli zmodyfikujesz struktury opcji w pliku kodu źródłowego `show_opt.c`, aby przekonać się, jaki wpływ na zachowanie programu będą miały te zmiany.

Na przykład, jeśli dla opcji `--port` ustawisz wartości minimalną, maksymalną oraz wielkość bloku jako odpowiednio 100, 1000 i 25, wtedy po ponownej kompilacji programu nie będziesz mógł podać numeru portu spoza zakresu od 100 do 1000. Ponadto, numer portu będzie zaokrąglony do najbliższej wielokrotności liczby 25.

Procedury przetwarzania opcji automatycznie obsługują opcje `--no-defaults`, `--print-defaults`, `--defaults-file` i `--defaults-extra-file`. Przekonaj się sam, jaki będzie efekt wywołania programu `show_opt` wraz z każdą z wymienionych opcji.

7.3.3. Implementacja przetwarzania opcji w programie klienta

Teraz jesteśmy gotowi do utworzenia pliku kodu źródłowego `connect2.c`. Zdefiniowany w nim program charakteryzuje się następującymi cechami:

- Nawiązuje połączenie z serwerem MySQL, zamyka połączenie, a następnie kończy działanie. Zachowanie jest więc podobne do programu zdefiniowanego w pliku `connect1.c`, ale do zgłoszenia ewentualnych błędów używana jest funkcja `print_error()`.
- Przetwarza opcje podane w wierszu poleceń lub w plikach opcji. Odbywa się to za pomocą kodu podobnego do przedstawionego w pliku `show_opt.c`, ale wzbogaconego o pytanie użytkownika o hasło, jeśli zachodzi potrzeba.

Poniżej przedstawiono zawartość pliku kodu źródłowego `connect2.c`:

```
/*
 * connect2.c - Program nawiązujący połączenie z serwerem MySQL za pomocą parametrów połączenia
 * podanych w pliku opcji lub w wierszu poleceń.
 */

#include <my_global.h>
#include <my_sys.h>
#include <m_string.h> /* Dla strdup(). */
#include <mysql.h>
#include <my_getopt.h>

static char *opt_host_name = NULL; /* Komputer serwera (domyślnie= localhost). */
static char *opt_user_name = NULL; /* Nazwa użytkownika (domyślnie = nazwa logowania). */
static char *opt_password = NULL; /* Hasło (domyślnie = brak). */
static unsigned int opt_port_num = 0; /* Numer portu (użyj wbudowanej wartości). */
static char *opt_socket_name = NULL; /* Nazwa gniazda (użyj wbudowanej wartości). */
static char *opt_db_name = NULL; /* Nazwa bazy danych (domyślnie = brak). */
static unsigned int opt_flags = 0; /* Flagi połączenia (brak). */

static int ask_password = 0; /* Czy program ma pytać o hasło? */

static MYSQL *conn; /* Wskaźnik do uchwytu połączenia. */

static const char *client_groups[] = { "client", NULL };

static struct my_option my_opts[] = /* Struktury informacji opcji. */
{
```

```

{"help", '?', "Wyświetlenie tego komunikatu pomocy i zakończenie programu.",
NULL, NULL, NULL,
GET_NO_ARG, NO_ARG, 0, 0, 0, 0, 0, 0},
{"host", 'h', "Komputer, z którym chcesz się połączyć.",
(uchar **) &opt_host_name, NULL, NULL,
GET_STR, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
{"password", 'p', "Hasło.",
(uchar **) &opt_password, NULL, NULL,
GET_STR, OPT_ARG, 0, 0, 0, 0, 0, 0},
{"port", 'P', "Numer portu.",
(uchar **) &opt_port_num, NULL, NULL,
GET_UINT, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
{"socket", 'S', "Ścieżka dostępu do gniazda.",
(uchar **) &opt_socket_name, NULL, NULL,
GET_STR, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
{"user", 'u', "Nazwa użytkownika.",
(uchar **) &opt_user_name, NULL, NULL,
GET_STR, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
{ NULL, 0, NULL, NULL, NULL, NULL, GET_NO_ARG, NO_ARG, 0, 0, 0, 0, 0, 0 }
};

static void print_error (MYSQL *conn, char *message)
{
    fprintf (stderr, "%s\n", message);
    if (conn != NULL)
    {
        fprintf (stderr, "Błąd %u (%s): %s\n",
                mysql_errno (conn), mysql_sqlstate (conn), mysql_error (conn));
    }
}

static my_bool get_one_option (int optid, const struct my_option *opt, char *argument)
{
    switch (optid)
    {
    case '?':
        my_print_help (my_opts); /* Wyświetlenie komunikatu pomocy. */
        exit (0);
    case 'p':
        /* Hasło. */
        if (!argument)
            /* Brak podanej wartości; należy poprosić o nią później. */
            ask_password = 1;
        else
            /* Skopiowanie hasła i nadpisanie początkowego. */
            {
                opt_password = strdup (argument);
                if (opt_password == NULL)
                {
                    print_error (NULL, "Nie udało się alokować bufora hasła.");
                    exit (1);
                }
                while (*argument)
                    *argument++ = 'x';
                ask_password = 0;
            }
        break;
    }
    return (0);
}

```

```

int main (int argc, char *argv[])
{
    int opt_err;

    MY_INIT (argv[0]);
    load_defaults ("my", client_groups, &argc, &argv);

    if ((opt_err = handle_options (&argc, &argv, my_opts, get_one_option)))
        exit (opt_err);

    /* Wyświetlenie pytania o hasło, o ile zachodzi potrzeba. */
    if (ask_password)
        opt_password = get_tty_password (NULL);

    /* Pobranie nazwy bazy danych, jeśli została podana w wierszu poleceń. */
    if (argc > 0)
    {
        opt_db_name = argv[0];
        --argc; ++argv;
    }

    /* Inicjalizacja biblioteki klienta. */
    if (mysql_library_init (0, NULL, NULL))
    {
        print_error (NULL, "Wywołanie mysql_library_init() zakończyło się niepowodzeniem.");
        exit (1);
    }

    /* Inicjalizacja uchwytu połączenia. */
    conn = mysql_init (NULL);
    if (conn == NULL)
    {
        print_error (NULL, "Wywołanie mysql_init() zakończyło się niepowodzeniem
(prawdopodobnie z powodu braku pamięci).");
        exit (1);
    }

    /* Nawiązanie połączenia z serwerem. */
    if (mysql_real_connect (conn, opt_host_name, opt_user_name, opt_password,
        opt_db_name, opt_port_num, opt_socket_name, opt_flags) == NULL)
    {
        print_error (conn, "Wywołanie mysql_real_connect() zakończyło się niepowodzeniem.");
        mysql_close (conn);
        exit (1);
    }

    /* Kod odpowiedzialny za wykonanie zapytań i przetworzenie ich wyników. */

    /* Zamknięcie połączenia z serwerem i zamknięcie biblioteki klienta. */
    mysql_close (conn);
    mysql_library_end ();
    exit (0);
}

```

W porównaniu do utworzonych wcześniej programów connect1 i show_opt program connect2 różni się pod kilkoma względami:

- Pozwala na podanie domyślnej bazy danych jako argumentu w wierszu poleceń. Takie zachowanie jest zgodne ze standardowymi klientami dostarczonymi wraz z dystrybucją MySQL.
- Jeżeli hasło będzie znajdowało się w tablicy argumentów, funkcja `get_one_option()` utworzy jego kopię i nadpisze hasło początkowe. To skraca do minimum czas, przez który hasło podane w wierszu poleceń jest widoczne dla ps lub innych narzędzi systemowych wyświetlających informacje o stanie. (Wspomniany czas jest jedynie *zminimalizowany*, a nie wyeliminowany. Podawanie hasła w wierszu poleceń nadal jest ryzykowne).
- Jeżeli opcja hasła zostanie podana bez wartości, wtedy funkcja `get_one_option()` ustawi flagę wskazującą, że program powinien poprosić użytkownika o podanie hasła. Odbywa się to za pomocą funkcji `get_tty_password()` wywoływanej w funkcji `main()` po przetworzeniu wszystkich opcji. Funkcja `get_tty_password()` jest oferowana przez bibliotekę klienta funkcją narzędziową, która prosi użytkownika o podanie hasła, ale nie wyświetla go na ekranie. Mógłbyś w tym miejscu zapytać: dlaczego po prostu nie wywołać funkcji `getpass()`? Odpowiedź jest prosta: ponieważ nie wszystkie systemy mają tę funkcję (na przykład Windows nie ma). Funkcja `get_tty_password()` jest przenośna między systemami, ponieważ może być skonfigurowana w celu dopasowania do specyficznych cech systemu.

Skompiluj program `connect2`, a następnie spróbuj go uruchomić w poniższy sposób:

```
% ./connect2
```

Jeżeli program `connect2` nie wygeneruje żadnych danych wyjściowych (jak wcześniej pokazano), oznacza to, że nawiązanie połączenia zakończyło się sukcesem. Ewentualnie możesz zobaczyć komunikat podobny do poniższego:

```
% ./connect2
```

```
Wywołanie mysql_real_connect() zakończyło się niepowodzeniem:
Error 1045 (28000): Access denied for user 'sampadm'@'localhost'
(using password: NO)
```

Powyższy komunikat informuje, że połączenie nie zostało nawiązane, i podaje przyczynę. W przedstawionej sytuacji błąd `Access denied` oznacza konieczność podania odpowiednich parametrów połączenia. W przypadku programu `connect1` konieczna była edycja kodu źródłowego i jego ponowna kompilacja. Z kolei program `connect2` nawiązuje połączenie z serwerem MySQL za pomocą podanych opcji. Aby nie komplikować przykładu, przyjmujemy założenie o braku pliku opcji. Jeśli program `connect2` zostanie wywołany bez argumentów, wtedy spróbuje nawiązać połączenie z `localhost` i przekaże serwerowi nazwę użytkownika i hasło z systemu UNIX. Natomiast wywołanie programu `connect2`, jak pokazano poniżej, powoduje wyświetlenie pytania o hasło (ponieważ po opcji `-p` nie podano hasła), nawiązanie połączenia z komputerem o nazwie `pevien_komputer` i przekazanie serwerowi nazwy użytkownika `pevien_uzytkownik` oraz hasła podanego po wyświetleniu odpowiedniego komunikatu:

```
% ./connect2 -h pevien_komputer -p -u pevien_uzytkownik pewna_baza_danych
```

Program `connect2` przekazuje nazwę bazy danych (`pewna_baza_danych`) funkcji `mysql_real_connect()` w celu wskazania jej jako domyślnej. Jeżeli istnieje plik opcji, program przetworzy jego zawartość i odpowiednio zmodyfikuje parametry połączenia.

Zróbmy na chwilę krok wstecz i rozważmy, co zostało dotąd zrobione. Praca wykonana podczas tworzenia programu `connect2` pokazuje zadanie konieczne do wykonania w przypadku każdego klienta MySQL: nawiązanie połączenia z serwerem za pomocą odpowiednich parametrów. Przygotowane rozwiązanie dobrze sprawdza się podczas zgłaszania błędów, jeśli próba nawiązania połączenia zakończy się niepowodzeniem. W ten sposób przygotowaliśmy coś na kształt frameworka, który może być używany jako punkt wyjścia dla wielu różnych programów klienckich. Aby utworzyć nowego klienta, należy wykonać poniższe kroki:

1. Utwórz kopię pliku źródłowego `connect2.c`.
2. Jeżeli program akceptuje opcje dodatkowe inne niż standardowo zdefiniowane w pliku `connect2.c`, dodaj je do tablicy `my_opts` i zmodyfikuj pętlę przetwarzania opcji.
3. Dodaj własny odpowiedni dla aplikacji kod między wywołaniami nawiązywania i zamykania połączenia.

I to już wszystko.

Wszystkie rzeczywiste akcje dla aplikacji zachodzą między wywołaniami funkcji `mysql_real_connect()` i `mysql_close()`. Przygotowanie możliwego do ponownego użycia szkieletu oznacza, że możesz się bardziej skoncentrować na celach, które chcesz osiągnąć, czyli na uzyskaniu dostępu do zawartości baz danych.

7.4. Przetwarzanie zapytań SQL

Celem nawiązania połączenia z serwerem jest prowadzenie z nim komunikacji za pomocą zapytań wykonywanych, gdy połączenie jest otwarte. W tym podrozdziale dowiesz się, jak to zrobić. Wykonanie każdego zapytania obejmuje wymienione poniżej kroki:

1. Przygotowanie zapytania. Konkretny sposób zależy od treści zapytania, w szczególności od tego, czy zawiera dane binarne.
2. Wykonanie zapytania przez jego wysłanie do serwera. Serwer wykona otrzymane zapytanie i wygeneruje wynik.
3. Przetworzenie wyników zapytania. To zależy od typu wykonanego zapytania. Na przykład, zapytanie `SELECT` zwraca rekordy danych do przetworzenia. Z kolei zapytanie `INSERT` nie zwraca rekordów.

Biblioteka klienta zawiera dwa zestawy procedur przeznaczonych do obsługi wykonywania zapytań. Pierwszy zestaw każde zapytanie wysyła do serwera w postaci ciągu tekstowego i dla wszystkich kolumn zwraca wyniki w formacie ciągu tekstowego. Z kolei drugi zestaw używa protokołu binarnego, co pozwala na wysyłanie i otrzymywanie danych innych niż w formacie ciągu tekstowego w ich rodzimej postaci bez konieczności przeprowadzania konwersji na format ciągu tekstowego oraz z niego.

W tym podrozdziale zostanie omówiona oryginalna metoda przetwarzania zapytań SQL. W podrozdziale 7.8, zatytułowanym „Używanie zapytań preinterpretowanych”, znajdziesz omówienie protokołu binarnego.

Podczas tworzenia zapytań szczególnie ważny jest jeden czynnik: wybór procedury używanej w celu wysłania zapytania do serwera. Ogólną procedurą przeznaczoną do wysyłania zapytań jest `mysql_real_query()`. Za jej pomocą podajesz zapytanie w postaci ciągu tekstowego o określonej długości (to znaczy sam ciąg tekstowy i jego długość). Wspomnianą długość ciągu tekstowego zapytania musisz śledzić i przekazywać funkcji `mysql_real_query()` wraz z samym ciągiem. Ponieważ zapytanie jest traktowane jako ciąg tekstowy o wskazanej długości, a nie zakończony znakiem null, może zawierać cokolwiek, nawet dane binarne i bajty null.

Inna funkcja przeznaczona do wysyłania zapytań nosi nazwę `mysql_query()` i jest bardziej restrykcyjna pod względem zawartości ciągu tekstowego zapytania, ale jednocześnie zwykle łatwiejsza w użyciu. Każde zapytanie przekazane funkcji `mysql_query()` powinno być ciągiem tekstowym zakończonym znakiem null. Oznacza to, że tekst zapytania nie może zawierać bajtów null, ponieważ spowodują one interpretację krótszego fragmentu zapytania. Ogólnie rzecz ujmując, jeśli zapytanie może zawierać dowolne dane binarne, wtedy może zawierać bajty null, a więc nie powinieneś używać funkcji `mysql_query()`. Z drugiej strony, podczas pracy z ciągami tekstowymi ograniczonymi znakami null zyskujesz luksus tworzenia zapytań za pomocą funkcji standardowej biblioteki C przeznaczonej do obsługi ciągów tekstowych. Wspomniane funkcje prawdopodobnie doskonale znasz, zaliczają się do nich między innymi `strcpy()` i `sprintf()`.

Inny czynnik, który warto wziąć pod uwagę podczas tworzenia zapytań, to potrzeba przeprowadzania jakichkolwiek operacji cytowania znaków. To jest konieczne, jeśli chcesz przygotowywać zapytania za pomocą wartości zawierających dane binarne lub innych kłopotliwych znaków, takich jak znaki cytowania i ukośniki. To zagadnienie poruszono w podpunkcie 7.4.7.1, zatytułowanym „Praca z ciągami tekstowymi zawierającymi znaki specjalne”.

Prosty szkielet procedury obsługi zapytania przedstawia się następująco:

```
if (mysql_query (conn, stmt_str) != 0)
{
    /* Niepowodzenie; należy zgłosić błąd. */
}
else
{
    /* Sukces; należy sprawdzić, jaki efekt miało wykonanie zapytania. */
}
```

Funkcje `mysql_query()` i `mysql_real_query()` zwracają wartość zero w przypadku sukcesu i wartość niezerową w przypadku niepowodzenia. Słowo „sukces” oznacza tutaj akceptację zapytania przez serwer i możliwość jego wykonania. Natomiast nie zawiera żadnych informacji odnośnie efektu wykonania zapytania. Na przykład, nie informuje, czy zapytanie `SELECT` pobrało jakiegokolwiek rekordy lub czy zapytanie `DELETE` usunęło jakiegokolwiek rekordy. Sprawdzenie efektu wykonania zapytania tak naprawdę oznacza konieczność przeprowadzenia dodatkowego przetwarzania.

Wykonanie zapytania może z wielu powodów zakończyć się niepowodzeniem. Najczęstsze przyczyny awarii są następujące:

- Zapytanie zawiera błąd składni.
- Zapytanie jest semantycznie niepoprawne, na przykład odwołuje się do nieistniejącej tabeli.
- Nie masz wystarczających uprawnień w celu uzyskania dostępu do tabeli użytej w zapytaniu.

Zapytania mogą zaliczać się do dwóch szeroko zdefiniowanych kategorii: modyfikujących rekordy i zwracających zbiór wynikowy (zestaw rekordów). Zapytania takie jak INSERT, DELETE i UPDATE modyfikują rekordy i zwracają liczbę wskazującą ilość rekordów, których dotyczyło zapytanie.

Z kolei zapytania takie jak SELECT i SHOW zwracają zbiór wynikowy. W API C oferowanym przez MySQL zbiór wynikowy zwracany przez wymienione zapytania jest przedstawiany przez typ danych MYSQL_RES. To struktura zawierająca wartości danych dla rekordów, a także metadane opisujące wspomniane wartości (przykłady metadanych to nazwy kolumn lub wielkość wartości danych). Zbiór wynikowy może być pusty (to znaczy nie zawiera żadnego rekordu).

7.4.1. Obsługa zapytań modyfikujących rekordy

W celu przetworzenia zapytania modyfikującego rekordy należy wywołać funkcję `mysql_query()` lub `mysql_real_query()`. Jeżeli wykonanie zapytania zakończy się powodzeniem, liczbę wstawionych, usuniętych lub uaktualnionych rekordów można poznać za pomocą wywołania `mysql_affected_rows()`.

Poniższy przykład pokazuje, jak obsługiwać zapytanie modyfikujące rekordy:

```
if (mysql_query (conn, "INSERT INTO my_tbl SET name = 'My Name'") != 0)
{
    print_error (conn, "Wykonanie zapytania INSERT zakończyło się niepowodzeniem.");
}
else
{
    printf ("Wykonanie zapytania INSERT zakończyło się sukcesem; liczba rekordów,
           na które wpłynęło zapytanie: %lu\n", (unsigned long) mysql_affected_rows (conn));
}
```

Zwróć uwagę na rzutowanie wyniku wykonania funkcji `mysql_affected_rows()` na wartość typu `unsigned long`. Wymieniona funkcja zwraca wartość typu `my_ulonglong`, ale próba bezpośredniego wyświetlenia wartości tego typu może nie działać we wszystkich systemach. Rzutowanie wartości na typ `unsigned long` i użycie specyfikatora formatu `%lu` rozwiązuje ten problem. Taka sama zasada ma zastosowanie względem wszystkich pozostałych funkcji zwracających wartości typu `my_ulonglong`, na przykład `mysql_num_rows()` i `mysql_insert_id()`. Powinieneś o tym pamiętać, jeżeli chcesz zapewnić przenośność programów klienckich między różnymi systemami.

Wartością zwrótną funkcji `mysql_affected_rows()` jest liczba rekordów, na które wpłynęło wykonanie zapytania. Jednak znaczenie słowa „wpłynąć” zależy od typu zapytania. W przypadku zapytań `INSERT`, `REPLACE` lub `DELETE` będzie to liczba rekordów wstawionych, zastąpionych lub usuniętych. Z kolei dla zapytania `UPDATE` oznacza liczbę rekordów uaktualnionych, czyli faktycznie zmodyfikowanych przez MySQL. Serwer MySQL nie uaktualnia rekordu, jeśli jego zawartość jest taka sama, jaką miałby zostać uaktualniony. Tak więc wprowadzić rekord może być wybrany do uaktualnienia (przez klauzulę `WHERE` zapytania `UPDATE`), ale w rzeczywistości pozostanie niezmienny.

W rzeczywistości znaczenie wyrażenia „rekordów, na które wpłynęło zapytanie” w przypadku zapytania `UPDATE` jest nieco kontrowersyjne. Dla części osób oznacza „dopasowanych rekordów”, czyli liczbę rekordów wybranych do uaktualnienia, nawet jeśli operacja uaktualnienia nie spowoduje zmiany ich wartości. Jeżeli w tworzonej przez Ciebie aplikacji wymagane jest takie znaczenie, żądane zachowanie osiągniesz, jeśli podczas nawiązania połączenia z serwerem przekażesz wartość `CLIENT_FOUND_ROWS` we flagach parametru funkcji `mysql_real_connect()`.

7.4.2. Obsługa zapytań zwracających zbiór wyników

Zapytania zwracające dane robią to w postaci zbioru danych otrzymywanego po wykonaniu zapytania za pomocą wywołania funkcji `mysql_query()` lub `mysql_real_query()`. Trzeba koniecznie pamiętać, że w MySQL zapytanie `SELECT` nie jest jedynym, które zwraca rekordy. Zapytania takie jak `SHOW`, `DESCRIBE`, `EXPLAIN` i `CHECK TABLE` również zwracają rekordy. W przypadku wszystkich wymienionych zapytań po wykonaniu każdego z nich konieczne jest przeprowadzenie dodatkowych operacji przetwarzania rekordów:

1. Wygenerowanie zbioru wynikowego za pomocą wywołania `mysql_store_result()` lub `mysql_use_result()`. Wymienione funkcje zwracają wskaźnik `MYSQL_RES` w przypadku sukcesu lub `NULL` w przypadku niepowodzenia. W dalszej części rozdziału zostaną omówione różnice między funkcjami `mysql_store_result()` i `mysql_use_result()`, a także warunki wpływające na wybór jednej zamiast drugiej. Na razie w przykładach będziemy używali funkcji `mysql_store_result()`, która natychmiast pobiera rekordy z serwera i buforuje je w pamięci po stronie klienta.
2. Wywołanie funkcji `mysql_fetch_row()` dla każdego rekordu zbioru wynikowego. Wartością zwrótną tej funkcji jest wartość `MYSQL_ROW` lub `NULL`, gdy nie ma już więcej rekordów do pobrania. Wartość `MYSQL_ROW` jest wskaźnikiem do tablicy ciągów tekstowych przedstawiających wartości dla poszczególnych kolumn w rekordzie. Sposób użycia rekordów zależy od aplikacji. Na przykład, możesz wyświetlić wartości rekordów lub przeprowadzić na nich pewne obliczenia statystyczne.
3. Po zakończeniu pracy ze zbiorem wynikowym konieczne jest wywołanie funkcji `mysql_free_result()` w celu zwolnienia używanej przez niego pamięci. Jeżeli zapomnisz o tym kroku, w aplikacji wystąpi zjawisko wycieku pamięci. Prawidłowe usuwanie zbiorów wynikowych z pamięci jest szczególnie ważne w przypadku

długo działających aplikacji. W przeciwnym razie aplikacja będzie zużywała coraz większą ilość zasobów systemowych, a wydajność systemu zacznie powoli spadać.

Poniższy przykładowy szkielet pokazuje, jak przetwarzać zapytanie zwracające zbiór wyników:

```
MYSQL_RES *res_set;
if (mysql_query (conn, "SHOW TABLES FROM sampdb") != 0)
    print_error (conn, "Wywołanie mysql_query() zakończyło się niepowodzeniem.");
else
{
    res_set = mysql_store_result (conn); /* Wygenerowanie zbioru wynikowego. */
    if (res_set == NULL)
        print_error (conn, "Wywołanie mysql_store_result() zakończyło się niepowodzeniem.");
    else
    {
        /* Przetworzenie zbioru wynikowego, a następnie jego usunięcie. */
        process_result_set (conn, res_set);
        mysql_free_result (res_set);
    }
}
```

Przedstawiony przykład ukrywa szczegóły związane z przetwarzaniem zbioru wynikowego w innej funkcji, o nazwie `process_result_set()`, która nie została jeszcze zdefiniowana. Ogólnie rzecz biorąc, operacje obsługujące zbiór wyników opierają się na pętli przypominającej poniższą konstrukcję:

```
MYSQL_ROW row;

while ((row = mysql_fetch_row (res_set)) != NULL)
{
    /* Dowolne operacje na zawartości rekordów. */
}
```

Wartością zwrótną funkcji `mysql_fetch_row()` jest wartość `MYSQL_ROW`, będąca wskaźnikiem do tablicy wartości. Jeżeli wartość zwrótna zostanie przypisana zmiennej o nazwie `row`, dostęp do każdej wartości w rekordzie można uzyskać za pomocą `row[i]`, gdzie *i* ma zakres od 0 do liczby o jeden mniejszej od liczby kolumn w danym rekordzie. Istnieje kilka ważnych kwestii dotyczących typu danych `MYSQL_ROW`, o których należy pamiętać:

- `MYSQL_ROW` to wskaźnik i dlatego zmienną tego typu należy zadeklarować jako `MYSQL_ROW row`, a nie jako `MYSQL_ROW *row`.
- Wartości dla wszystkich typów danych (nawet liczbowych) są zwracane w tablicy `MYSQL_ROW` w postaci ciągów tekstowych. Aby wartość traktować jak liczbę, trzeba samodzielnie przeprowadzić konwersję ciągu tekstowego.
- Ciągi tekstowe w tablicy `MYSQL_ROW` są zakończone znakami null. Jednak jeśli kolumna zawiera dane binarne, to może mieć bajty null, a więc nie można traktować jej wartości jako ciągu tekstowego zakończonego znakiem null. Konieczne jest sprawdzenie wielkości kolumny w celu ustalenia wielkości

wartości. (W punkcie 7.4.6, zatytułowanym „Używanie metadanych zbioru wynikowego”, dowiesz się, jak ustalić wielkość kolumny).

- Wartości SQL NULL są w tablicy MYSQL_ROW przedstawiane w postaci wskaźników C NULL. Jeżeli nie jesteś pewien, że kolumna została zadeklarowana jako NOT NULL, zawsze powinieneś sprawdzać, czy wartością kolumny nie jest NULL. W przeciwnym razie może dojść do awarii aplikacji w przypadku próby odwołania się do wskaźnika NULL.

Akcje podejmowane na poszczególnych rekordach zależą od przeznaczenia aplikacji. Na potrzeby przykładu przyjmujemy założenie, że każdy rekord zostanie wyświetlony jako zestaw wartości rozdzielonych tabulatorami. W tym celu konieczne jest ustalenie liczby kolumn wartości znajdujących się w rekordzie. Wspomnianą liczbę zwraca inna funkcja biblioteki klienta, o nazwie `mysql_num_fields()`.

Poniżej przedstawiono kod funkcji `process_result_set()`:

```
void
process_result_set (MYSQL *conn, MYSQL_RES *res_set)
{
    MYSQL_ROW    row;
    unsigned int i;

    while ((row = mysql_fetch_row (res_set)) != NULL)
    {
        for (i = 0; i < mysql_num_fields (res_set); i++)
        {
            if (i > 0)
                fputc ('\t', stdout);
            printf ("%s", row[i] != NULL ? row[i] : "NULL");
        }
        fputc ('\n', stdout);
    }
    if (mysql_errno (conn) != 0)
        print_error (conn, "Wywołanie mysql_fetch_row() zakończyło się niepowodzeniem.");
    else
        printf ("Liczba zwróconych rekordów: %lu\n",
                (unsigned long) mysql_num_rows (res_set));
}
```

Funkcja `process_result_set()` wyświetla zawartość każdego rekordu w formacie ograniczonym tabulatorami (wartości NULL są wyświetlane jako słowa „NULL”) oraz liczbę pobranych rekordów. Wspomniana liczba jest otrzymywana za pomocą wywołania funkcji `mysql_num_rows()`. Podobnie jak w przypadku `mysql_affected_rows()`, także funkcja `mysql_num_rows()` zwraca wartość `my_ulonglong`. Dlatego rzutujemy ją na typ `unsigned long`, a następnie do wyświetlenia używamy specyfikatora formatu `%lu`. Jednak w przeciwieństwie do `mysql_affected_rows()`, pobierającej argument w postaci uchwytu połączenia, argumentem pobieranym przez `mysql_num_rows()` jest wskaźnik zbioru wynikowego.

Kod znajdujący się za pętlą zapewnia obsługę błędów. W przypadku utworzenia zbioru wynikowego za pomocą funkcji `mysql_store_result()`, wartość zwrotna NULL funkcji `mysql_fetch_row()` zawsze oznacza „brak kolejnych rekordów”. Jednak po utworzeniu

zbioru wynikowego za pomocą `mysql_use_result()` wartość zwrrotna `NULL` funkcji `mysql_fetch_row()` może oznaczać „brak kolejnych rekordów” lub błąd. Ponieważ funkcja `process_result_set()` nie wie, za pomocą jakiej funkcji został utworzony zbiór wynikowy, przeprowadza test, co pozwala jej na prawidłowe zapewnienie obsługi błędów.

Przedstawiona powyżej wersja funkcji `process_result_set()` prezentuje minimalne podejście w zakresie wyświetlania wartości kolumn, co oczywiście wiąże się z pewnymi wadami. Przyjmujemy założenie, że zostało wykonane poniższe zapytanie:

```
SELECT last_name, first_name, city, state FROM president
ORDER BY last_name, first_name
```

W wyniku otrzymujemy następujące dane wyjściowe, które nie są łatwe w odczycie:

```
Adams   John   Braintree  MA
Adams   John   Quincy    Braintree  MA
Arthur  Chester A. Fairfield  VT
Buchanan James  Mercersburg PA
Bush    George H.W. Milton    MA
Bush    George W.  New Haven  CT
Carter  James E.   Plains     GA
...
```

Istnieje możliwość ładniejszego sformatowania danych wyjściowych przez podanie informacji takich jak etykiety kolumn i pionowe wyrównanie wartości. W tym celu potrzebne są etykiety oraz ustalenie najdłuższej wartości w każdej kolumnie. Wspomniane informacje są dostępne, ale nie jako część wartości danych kolumny — raczej jako część metadanych zbioru wynikowego. (Warto pamiętać, że metadane to po prostu dane o danych). Po ogólnym przygotowaniu procedury obsługi zapytania w punkcie 7.4.6, zatytułowanym „Używanie metadanych zbioru wynikowego”, zastosujemy nieco ładniejszy sposób wyświetlenia danych.

Wyświetlanie danych binarnych

Kolumny zawierające dane binarne wraz z bajtami `null` nie będą prawidłowo wyświetlane za pomocą specyfikatora formatu `%s` funkcji `printf()`. Funkcja `printf()` oczekuje ciągów tekstowych zakończonych znakami `null` i wyświetla wartość kolumny jedynie do pierwszego napotkanego znaku `null`. W przypadku danych binarnych najlepszym rozwiązaniem jest użycie funkcji akceptujących argument w postaci długości kolumny, co pozwala na wyświetlenie pełnej wartości kolumny. Na przykład, możesz użyć funkcji `fwrite()`.

7.4.3. Procedury obsługi zapytań ogólnego przeznaczenia

Poprzednie przykłady obsługi zapytań oparte były na wiedzy, czy zapytanie powinno zwrócić jakiegokolwiek dane. Było to możliwe z powodu umieszczenia zapytań na stałe w kodzie aplikacji: użyliśmy zapytania `INSERT`, niezwracającego zbioru wynikowego, oraz zapytania `SHOW TABLES`, zwracającego zbiór wynikowy.

Jednak nie zawsze będzie wiadomo, do której kategorii zalicza się dane zapytanie. Na przykład, jeśli wykonujesz zapytanie wpisane przez użytkownika z klawiatury lub

pobrane z pliku, to może być ono dowolne. Nie będziesz wcześniej wiedział, czy oczekiwać zbioru wynikowego ani nawet czy zapytanie będzie prawidłowe. Co można zrobić w takiej sytuacji? Na pewno nie należy próbować przetworzyć zapytania w celu ustalenia jego rodzaju. To jest przecież zadaniem serwera.

Na szczęście, nie musisz wcześniej znać kategorii zapytania, aby móc je prawidłowo przetworzyć. API C w MySQL pozwala na tworzenie procedur obsługi zapytań ogólnego przeznaczenia, które przetwarzają zapytania dowolnego rodzaju: zwracające zbiór wynikowy i niezwracające go, wykonane z sukcesem lub niepowodzeniem. Zanim przystąpimy do tworzenia kodu tego rodzaju procedury, wcześniej warto naszkicować sposób jej implementacji:

1. Wykonanie zapytania. Jeśli zakończy się niepowodzeniem, to już koniec pracy.
2. Jeżeli wykonanie zapytania zakończy się sukcesem, należy wywołać funkcję `mysql_store_result()` w celu pobrania rekordów z serwera i utworzenia zbioru wynikowego.
3. Jeżeli wykonanie funkcji `mysql_store_result()` zakończy się powodzeniem, wartością zwrótną będzie zbiór wynikowy. Rekordy trzeba przetworzyć za pomocą wywoływania funkcji `mysql_fetch_row()` aż do chwili, gdy jej wartością zwrótną będzie NULL, a następnie usunąć zbiór wynikowy z pamięci.
4. Jeżeli wykonanie funkcji `mysql_store_result()` zakończy się niepowodzeniem, zapytanie może nie zwracać żadnego zbioru wynikowego lub powinno, ale wystąpił błąd podczas pobierania wspomnianego zbioru wynikowego. Ustalenie właściwej przyczyny niepowodzenia jest możliwe przez przekazanie uchwytu połączenia funkcji `mysql_field_count()` i sprawdzenie jej wartości zwrótniej.

Jeśli wartością zwrótną funkcji `mysql_field_count()` jest zero, wtedy zapytanie nie zwróciło żadnych kolumn i tym samym nie ma zbioru wynikowego. (To wskazuje na wykonanie zapytania typu INSERT, DELETE lub UPDATE).

Jeśli wartością zwrótną funkcji `mysql_field_count()` jest wartość niezerowa, wtedy mamy do czynienia z błędem, ponieważ zapytanie powinno zwrócić zbiór wynikowy, ale tego nie zrobiło. Błąd mógł wystąpić z wielu przyczyn. Na przykład, zbiór wynikowy może być na tyle ogromny, że nie ma wystarczającej ilości pamięci na jego alokację. Inna przyczyna to awaria sieci między klientem i serwerem w trakcie pobierania rekordów.

Poniższy fragment kodu pokazuje funkcję przetwarzającą dowolne zapytanie. Funkcja pobiera uchwyt połączenia oraz zakończony znakiem null ciąg tekstowy zapytania:

```
void process_statement (MYSQL *conn, char *stmt_str)
{
    MYSQL_RES *res_set;

    if (mysql_query (conn, stmt_str) != 0) /* Zapytanie zakończyło się niepowodzeniem. */
    {
        print_error (conn, "Nie można było wykonać zapytania.");
        return;
    }
}
```

```

}

/* Wykonanie zapytania zakończone sukcesem. Należy ustalić, czy zapytanie powinno zwrócić dane. */
res_set = mysql_store_result (conn);
if (res_set) /* Zbiór wynikowy został zwrócony. */
{
    /* Przetworzenie rekordów i usunięcie zbioru wynikowego z pamięci. */
    process_result_set (conn, res_set);
    mysql_free_result (res_set);
}
else /* Zbiór wynikowy nie został zwrócony. */
{
    /*
     * Czy brak zbioru wynikowego oznacza brak danych wyjściowych zapytania,
     * czy raczej wskazuje na wystąpienie błędu?
     */
    if (mysql_field_count (conn) == 0)
    {
        /*
         * Zapytanie nie wygenerowało zbioru wynikowego (to nie było zapytanie typu SELECT,
         * SHOW, DESCRIBE itd.); wystarczy jedynie ustalić liczbę rekordów, których dotyczyło.
         */
        printf ("Liczba rekordów, których dotyczyło zapytanie: %lu\n",
                (unsigned long) mysql_affected_rows (conn));
    }
    else /* Wystąpił błąd. */
    {
        print_error (conn, "Nie można było pobrać zbioru wynikowego.");
    }
}
}

```

7.4.4. Alternatywne podejścia do przetwarzania zapytań

Przedstawiona wcześniej wersja funkcji `process_statement()` ma trzy wymienione poniżej właściwości:

- Używa funkcji `mysql_query()` do wykonania zapytania.
- Używa funkcji `mysql_store_query()` do pobrania zbioru wynikowego.
- W przypadku braku zbioru wynikowego używa funkcji `mysql_field_count()` w celu ustalenia przyczyny jego braku: zapytanie nie generuje zbioru wynikowego lub wystąpił błąd.

Alternatywne podejścia są dostępne dla wszystkich trzech wymienionych powyżej aspektów obsługi zapytań:

- Zapytanie można wykonać za pomocą ciągu tekstowego o określonej wielkości i funkcji `mysql_real_query()` zamiast ciągu tekstowego zakończonego znakiem null o funkcji `mysql_query()`.
- Zbiór wynikowy można utworzyć za pomocą wywołania funkcji `mysql_use_result()` zamiast `mysql_store_result()`.

- Wywołanie funkcji `mysql_error()` lub `mysql_errno()` zamiast `mysql_field_count()` pozwala na ustalenie, czy pobranie zbioru wynikowego zakończyło się niepowodzeniem, czy po prostu zapytanie nie wygenerowało żadnego.

Wszystkie powyższe podejścia lub dowolne z nich można zastosować zamiast użytych w funkcji `process_statement()`. Poniżej przedstawiono kod funkcji `process_real_statement()`, działającej analogicznie do `process_statement()`, ale używającej wszystkich trzech wymienionych podejść alternatywnych:

```
void process_real_statement (MYSQL *conn, char *stmt_str, unsigned int len)
{
    MYSQL_RES *res_set;

    if (mysql_real_query (conn, stmt_str, len) != 0) /* Zapytanie zakończyło się niepowodzeniem. */
    {
        print_error (conn, "Nie można było wykonać zapytania.");
        return;
    }
    /* Wykonanie zapytania zakończone sukcesem. Należy ustalić, czy zapytanie powinno zwrócić dane. */
    res_set = mysql_use_result (conn);
    if (res_set) /* Zbiór wynikowy został zwrócony. */
    {
        /* Przetworzenie rekordów i usunięcie zbioru wynikowego z pamięci. */
        process_result_set (conn, res_set);
        mysql_free_result (res_set);
    }
    else /* Zbiór wynikowy nie został zwrócony. */
    {
        /*
         * Czy brak zbioru wynikowego oznacza brak danych wyjściowych zapytania,
         * czy raczej wskazuje na wystąpienie błędu?
         */
        if (mysql_errno (conn) == 0)
        {
            /*
             * Zapytanie nie wygenerowało zbioru wynikowego (to nie było zapytanie typu SELECT,
             * SHOW, DESCRIBE itd.); wystarczy jedynie ustalić liczbę rekordów, których dotyczyło.
             */
            printf ("Liczba rekordów, których dotyczyło zapytanie: %lu\n",
                    (unsigned long) mysql_affected_rows (conn));
        }
        else /* Wystąpił błąd. */
        {
            print_error (conn, "Nie można było pobrać zbioru wynikowego.");
        }
    }
}
```

7.4.5. Funkcja `mysql_store_result()` kontra `mysql_use_result()`

Funkcje `mysql_store_result()` i `mysql_use_result()` są podobne pod tym względem, że obie pobierają argument w postaci uchwytu połączenia i zwracają zbiór wynikowy.

Jednak między wymienionymi funkcjami istnieją znaczne różnice. Podstawowa polega na sposobie, w jaki zbiór wynikowy jest pobierany z serwera. W przypadku funkcji `mysql_store_result()` wszystkie rekordy są pobierane natychmiast po jej wywołaniu. Z kolei funkcja `mysql_use_result()` inicjuje pobieranie, ale w rzeczywistości nie pobiera żadnych rekordów. Wymieniona różnica w pobieraniu rekordów spowodowała powstanie wszystkich pozostałych różnic między dwiema omawianymi funkcjami. W tym punkcie porównamy je, dzięki czemu zdobędziesz wiedzę pozwalającą na wybór odpowiedniej z nich w tworzonej aplikacji.

Funkcja `mysql_store_result()` pobiera zbiór wynikowy z serwera, wszystkie jego rekordy, alokuje dla nich pamięć i buforuje je po stronie klienta. Kolejne wywołania funkcji `mysql_fetch_row()` nigdy nie powodują powstania błędu, ponieważ funkcja po prostu pobiera rekord ze struktury danych przechowującej zbiór wynikowy. Dlatego też wartość zwrótna wywołania `mysql_fetch_row()` zawsze oznacza koniec zbioru wynikowego.

Z kolei funkcja `mysql_use_result()` nie pobiera żadnych rekordów. Zamiast tego inicjuje proces pobierania rekord po rekordzie, który trzeba przeprowadzić samodzielnie za pomocą wywołania funkcji `mysql_fetch_row()` dla każdego rekordu. W takim przypadku wartość zwrótna `NULL` funkcji `mysql_fetch_row()`, normalnie oznaczająca dotarcie do końca zbioru wynikowego, może oznaczać także wystąpienie błędu w trakcie komunikacji z serwerem. Rozróżnienie między dwiema wymienionymi sytuacjami jest możliwe dzięki wywołaniu funkcji `mysql_error()` lub `mysql_errno()`.

Funkcja `mysql_store_result()` ma większe wymagania w zakresie pamięci i przetwarzania niż `mysql_use_result()`, ponieważ cały zbiór wynikowy znajduje się po stronie klienta. Obciążenie związane z alokacją pamięci i konfiguracją struktury danych jest większe, a klient pobierający ogromny zbiór wynikowy ryzykuje wystąpieniem sytuacji braku pamięci. Jeżeli przewidujesz pobieranie w pojedynczym zbiorze wynikowym ogromnej liczby rekordów, rozważ użycie funkcji `mysql_use_result()`.

Funkcja `mysql_use_result()` ma mniejsze wymagania dotyczące pamięci, ponieważ jednorazowo potrzebuje do alokacji ilości pamięci niezbędnej do obsługi tylko jednego rekordu. Takie rozwiązanie działa także szybciej, ponieważ nie ma potrzeby konfiguracji skomplikowanej struktury danych dla zbioru wynikowego. Z drugiej strony, funkcja `mysql_use_result()` oznacza większe obciążenie dla serwera, który musi przechowywać rekordy zbioru wynikowego aż do chwili, gdy klient pobierze je wszystkie. Z tego powodu funkcja `mysql_use_result()` jest kiepskim wyborem dla określonego typu klientów:

- Klientów interaktywnych poruszających się z rekordu do rekordu na żądanie klienta. (Nie chcesz przecież, aby serwer czekał z wysłaniem kolejnego rekordu, ponieważ użytkownik postanowił zrobić przerwę na kawę).
- Klientów przeprowadzających intensywne operacje przetwarzania między kolejnymi operacjami pobierania rekordów.

W obu typach sytuacji klient nie będzie w stanie szybko pobrać wszystkich rekordów zbioru wynikowego. To stanowi obciążenie dla serwera oraz może mieć negatywny wpływ na inne klienty, zwłaszcza w przypadku używania silnika takiego jak MyISAM, który stosuje blokady na poziomie tabeli. Tabele, z których są pobierane dane, mają

nałożoną blokadę odczytu na czas wykonywania zapytania. Inne klienty, które próbują uaktualnić te tabele, zostaną zablokowane.

Zwiększone wymagania w zakresie pamięci narzucane przez `mysql_store_result()` przynoszą też korzyść w postaci dostępu od razu do całego zbioru wynikowego. Wszystkie rekordy zbioru są dostępne i można używać dowolnych z nich. Funkcje `mysql_data_seek()`, `mysql_row_seek()` i `mysql_row_tell()` pozwalają na uzyskanie dostępu do rekordów w dowolnej kolejności. Bez użycia funkcji `mysql_use_result()` dostęp do rekordów odbywa się w kolejności, w której zostały pobrane przez `mysql_fetch_row()`. Jeżeli rekordy chcesz przetwarzać w dowolnej kolejności innej niż sekwencja zwrócona przez serwer, to musisz użyć funkcji `mysql_store_result()`. Na przykład, jeśli aplikacja pozwala użytkownikowi na poruszanie się do przodu i wstecz po rekordach pobranych przez zapytanie, najlepszym rozwiązaniem będzie użycie funkcji `mysql_store_result()`.

W przypadku funkcji `mysql_store_result()` możesz uzyskać dostęp do pewnego rodzaju informacji o kolumnach, które są niedostępne podczas używania `mysql_use_result()`. Liczba rekordów jest pobierana za pomocą wywołania funkcji `mysql_num_rows()`. Maksymalne długości wartości w poszczególnych kolumnach są przechowywane w elemencie `max_width` kolumny `MYSQL_FIELD` struktur informacyjnych. W przypadku `mysql_use_result()` funkcja `mysql_num_rows()` nie zwraca poprawnej wartości aż do chwili pobrania wszystkich rekordów. Podobnie, element `max_width` jest niedostępny, ponieważ może być obliczony dopiero po analizie danych wszystkich rekordów.

Ponieważ funkcja `mysql_use_result()` wykonuje mniejszą ilość pracy niż `mysql_store_result()`, nakłada wymaganie nieistniejące w przypadku użycia `mysql_store_result()`: klient musi wywoływać funkcję `mysql_fetch_row()` dla każdego rekordu istniejącego w zbiorze wynikowym. Jeżeli nie zostanie to zrobione przed wykonaniem kolejnego zapytania, wszystkie pozostałe rekordy w bieżącym zbiorze wynikowym staną się częścią zbioru wynikowego następnego zapytania i wystąpi błąd „utruty synchronizacji”. (Aby tego uniknąć, przed wykonaniem kolejnego zapytania należy wywołać funkcję `mysql_free_result()`, która usuwa z pamięci wszystkie oczekujące rekordy). Jedną z implikacji zastosowania tego rodzaju modelu przetwarzania jest to, że za pomocą `mysql_use_result()` można jednocześnie pracować z tylko jednym zbiorem wynikowym.

Błędy utraty synchronizacji nie występują w przypadku używania funkcji `mysql_store_result()`, ponieważ po zakończeniu jej działania w serwerze nie ma żadnego rekordu oczekującego na pobranie. W rzeczywistości, używając funkcji `mysql_store_result()`, nie trzeba wywoływać `mysql_fetch_row()`. Czasami taka możliwość jest bardzo użyteczna, na przykład jeśli chcesz się tylko przekonać, czy uzyskasz niepusty zbiór wynikowy, a nie jaka jest jego zawartość. Na przykład, aby sprawdzić, czy tabela `mytbl` istnieje, można wykonać poniższe zapytanie:

```
SHOW TABLES LIKE 'mytbl'
```

Jeśli po wywołaniu funkcji `mysql_store_result()` wartość zwrótna `mysql_num_rows()` będzie niezerowa, to tabela istnieje. Nie ma potrzeby wywoływania funkcji `mysql_fetch_row()`.

Zbiory wynikowe generowane przez funkcję `mysql_store_result()` powinny być w pewnym momencie usuwane z pamięci za pomocą wywołania `mysql_free_result()`, ale niekoniecznie przed wykonaniem kolejnego zapytania. Oznacza to możliwość wygenerowania wielu zbiorów wynikowych i jednoczesnej pracy z nimi, co jest przeciwieństwem dla ograniczenia „tylko jeden zbiór wynikowy w danej chwili”, nakładanego podczas pracy z funkcją `mysql_use_result()`.

Aby zapewnić maksymalną elastyczność, daj użytkownikom możliwość wyboru przetwarzania zbioru wynikowego. Programy `mysql` i `mysqldump` dają taką możliwość. Domyślnie używają funkcji `mysql_store_result()`, ale wykorzystanie `mysql_use_result()` następuje po użyciu opcji `--quick`.

7.4.6. Używanie metadanych zbioru wynikowego

Zbiór wynikowy zawiera nie tylko wartości kolumn dla rekordów danych, ale także pewne informacje o danych. Wspomniane informacje są nazywane metadanymi zbioru wynikowego i obejmują następujące dane:

- Liczbę rekordów i kolumn w zbiorze wynikowym, dostępną za pomocą wywołania `mysql_num_rows()` i `mysql_num_fields()`.
- Długość każdej wartości kolumny w bieżącym rekordzie, dostępną za pomocą wywołania `mysql_fetch_lengths()`.
- Informacje o poszczególnych kolumnach, między innymi nazwę i typ, maksymalną długość wartości każdej kolumny, a także nazwę tabeli zawierającej tę kolumnę. Wymienione informacje są przechowywane w strukturach `MYSQL_FIELDS`, które najczęściej można pobrać za pomocą wywołania `mysql_fetch_field()`.

Dostępność metadanych po części zależy od metody przetwarzania zbioru wynikowego. Jak wspomniano w poprzednim punkcie, jeśli chcesz używać wartości maksymalnej długości kolumn, konieczne jest utworzenie zbioru wynikowego za pomocą funkcji `mysql_store_result()` zamiast `mysql_use_result()`.

Metadane zbioru wynikowego są użyteczne podczas podejmowania decyzji o sposobie przetwarzania danych zbioru wynikowego:

- Nazwy kolumn i ich długości są użyteczne podczas generowania ładnie sformatowanych danych wyjściowych wyrównanych pionowo i zawierających tytuły kolumn.
- Licznik kolumny wskazuje na liczbę iteracji pętli przetwarzającej kolejne wartości kolumny dla rekordów danych.
- Liczniki rekordu i kolumny pomagają w alokacji struktur danych zależnych od wielkości zbioru wynikowego.
- Typ danych kolumny pozwala na określenie, czy kolumna przedstawia liczbę, dane binarne itd.

Wcześniej w rozdziale, a dokładnie w punkcie 7.4.2, zatytułowanym „Obsługa zapytań zwracających zbiór wyników”, utworzyliśmy wersję funkcji `process_result_set()`, która wyświetlała rekordy zbioru wynikowego w formacie wartości rozdzielonych tabulatorami. Takie rozwiązanie świetnie sprawdza się w pewnych sytuacjach (na przykład, gdy dane chcesz zaimportować do arkusza stylów), ale to nie jest format pozwalający na ładne wyświetlenie danych. Przypomnij sobie, że wspomniana wersja funkcji `process_result_set()` wygenerowała następujące dane wyjściowe:

```
Adams John Braintree MA
Adams John Quincy Braintree MA
Arthur Chester A. Fairfield VT
Buchanan James Mercersburg PA
Bush George H.W. Milton MA
Bush George W. New Haven CT
Carter James E. Plains GA
...
```

Utworzymy teraz inną wersję funkcji `process_result_set()`, która będzie generowała dane wyjściowe w postaci tabeli i umieszczała nagłówki oraz linie dla każdej kolumny. Nowa wersja funkcji wyświetli te same dane, ale w formacie łatwiejszym do interpretacji:

```
+-----+-----+-----+-----+
| last_name | first_name | city | state |
+-----+-----+-----+-----+
| Adams | John | Braintree | MA |
| Adams | John Quincy | Braintree | MA |
| Arthur | Chester A. | Fairfield | VT |
| Buchanan | James | Mercersburg | PA |
| Bush | George H.W. | Milton | MA |
| Bush | George W. | New Haven | CT |
| Carter | James E. | Plains | GA |
...
+-----+-----+-----+-----+
```

Algorytm wyświetlania wykonuje następujące kroki:

1. Określenie szerokości każdej wyświetlanej kolumny.
2. Wyświetlenie otoczonego liniami (poziomymi i pionowymi) wiersza nagłówków kolumn.
3. Wyświetlenie wartości każdego rekordu ze zbioru wynikowego wraz z odpowiednimi liniami i pionowe wyrównanie wartości. Liczby są wyrównane do prawej kolumny, natomiast dla wartości NULL będzie wyświetlone słowo „NULL”.
4. Na końcu wyświetlenie liczby pobranych rekordów.

To ćwiczenie stanowi dobry przykład pokazujący możliwość wykorzystania metadanych zbioru wynikowego, ponieważ wymaga umiejętności wykonania względem zbioru wynikowego całkiem sporej liczby zadań, innych niż jedynie pobranie wartości danych znajdujących się w rekordach.

Być może myślisz sobie: „Opis ćwiczenia brzmi bardzo podobnie do sposobu, w jaki klient `mysql` wyświetla dane wyjściowe”. Tak, masz rację i śmiało możesz porównać kod

źródłowy klienta mysql z kodem tworzonej tutaj funkcji `process_result_set()`. Kod źródłowy nie jest taki sam, a porównanie dwóch podejść do rozwiązania tego samego problemu może okazać się pouczające.

Przede wszystkim, konieczne jest ustalenie szerokości każdej wyświetlanej kolumny. Przedstawiony poniżej fragment kodu pokazuje rozwiązanie tego problemu. Zwróć uwagę, że obliczenia są oparte całkowicie na metadanych zbioru wynikowego i nie odnoszą się w żaden sposób do wartości rekordu:

```
MYSQL_FIELD *field;
unsigned long col_len;
unsigned int i;

/* Określenie szerokości wyświetlanych kolumn; zbiór wynikowy musi być */
/* wygenerowany za pomocą funkcji mysql_store_result(), a nie mysql_use_result(). */
mysql_field_seek (res_set, 0);
for (i = 0; i < mysql_num_fields (res_set); i++)
{
    field = mysql_fetch_field (res_set);
    col_len = strlen (field->name);
    if (col_len < field->max_length)
        col_len = field->max_length;
    if (col_len < 4 && !IS_NOT_NULL (field->flags))
        col_len = 4; /* 4 = długość słowa "NULL". */
    field->max_length = col_len; /* Wyzercowanie informacji o kolumnie. */
}
```

Powyższy kod oblicza szerokość kolumny za pomocą iteracji przez struktury `MYSQL_FIELD` dla kolumn w zbiorze wynikowym. Przejście do pierwszej następuje przez wywołanie funkcji `mysql_field_seek()`. Kolejne wywołania funkcji `mysql_fetch_field()` zwracają wskaźniki do struktur dla kolejnych kolumn. Szerokość kolumny dla celów jej wyświetlenia to maksimum trzech wartości, z których każda zależy od metadanych w strukturze informacyjnej kolumny:

- długość `field->name`, czyli tytułu kolumny;
- `field->max_length`, wielkość najdłuższej wartości w kolumnie;
- długość ciągu tekstowego „NULL”, o ile `field->flags` wskazuje, że kolumna może zawierać wartości NULL.

Zauważ, że kiedy szerokość kolumny jest znana, tę wartość przypisujemy `max_length`, czyli elementowi struktury pobranej z biblioteki klienta. Czy to jest dozwolone? Czy zawartość struktury `MYSQL_FIELD` nie powinna być uznawana za jedynie do odczytu? Normalnie powiedziałbym „tylko do odczytu”, ale skoro niektóre programy klienckie w dystrybucji MySQL zmieniają wartość `max_length` w podobny sposób, to można przyjąć założenie, że jest to dozwolone. (Jeżeli preferujesz podejście alternatywne, bez modyfikacji `max_length`, to musisz zaalokować tablicę wartości `unsigned long` i przechowywać w niej obliczone szerokości kolumn).

Obliczenia szerokości kolumny wiążą się z jednym zastrzeżeniem. Przypomnij sobie, że `max_length` nie ma znaczenia podczas tworzenia zbioru wynikowego za pomocą funkcji

`mysql_use_result()`. Ponieważ wartości `max_length` potrzebujemy w celu obliczenia długości wartości kolumny, poprawne działanie algorytmu wymaga, aby zbiór wynikowy został wygenerowany za pomocą funkcji `mysql_store_result()`. W programach używających funkcji `mysql_use_result()` zamiast `mysql_store_result()` jednym z możliwych rozwiązań jest użycie elementu `length` struktury `MYSQL_FIELD`, który wskazuje maksymalną długość, jaką może mieć wartość kolumny.

Kiedy szerokości kolumn są już znane, jesteśmy gotowi do ich wyświetlenia. Obsługa tytułów jest łatwa. Dla danej kolumny trzeba użyć struktury informacyjnej wskazywanej przez `field` i wyświetlić element `name`, używając obliczonej wcześniej szerokości kolumny:

```
printf (" %-*s |", (int) field->max_length, field->name);
```

W przypadku danych konieczne jest przeprowadzenie iteracji pętli przez rekordy zbioru wynikowego i wyświetlenie wartości kolumn bieżącego rekordu w trakcie każdej iteracji. Wyświetlenie wartości z rekordu jest nieco trudne, ponieważ wartością może być `NULL` lub liczba (w takim przypadku należy ją wyrównać do prawej strony). Wartości kolumny są wyświetlane za pomocą przedstawionego poniżej fragmentu kodu, w którym `row[i]` przechowuje wartość danych, natomiast `field` prowadzi do informacji o kolumnie:

```
if (row[i] == NULL)           /* Wyświetlenie słowa "NULL". */
    printf (" %-*s |", (int) field->max_length, "NULL");
else if (IS_NUM (field->type)) /* Wyświetlenie wartości wyrównanej do prawej strony. */
    printf (" %-*s |", (int) field->max_length, row[i]);
else                          /* Wyświetlenie wartości wyrównanej do lewej strony. */
    printf (" %-*s |", (int) field->max_length, row[i]);
```

Wartością makro `IS_NUM()` jest `true`, jeśli typ danych wskazywany przez `field->type` to jeden z typów liczbowych, na przykład `INT`, `FLOAT` lub `DECIMAL`.

Poniżej przedstawiono pełny kod przeznaczony do wyświetlenia zbioru wynikowego. Ponieważ linie są wyświetlane wielokrotnie, znacznie łatwiejsze jest utworzenie funkcji `print_dashed()`, przeznaczonej do tego celu, zamiast powtarzanie odpowiedniego kodu w wielu miejscach:

```
void print_dashes (MYSQL_RES *res_set)
{
    MYSQL_FIELD *field;
    unsigned int i, j;

    mysql_field_seek (res_set, 0);
    fputc ('+', stdout);
    for (i = 0; i < mysql_num_fields (res_set); i++)
    {
        field = mysql_fetch_field (res_set);
        for (j = 0; j < field->max_length + 2; j++)
            fputc ('-', stdout);
        fputc ('+', stdout);
    }
    fputc ('\n', stdout);
}

void process_result_set (MYSQL *conn, MYSQL_RES *res_set)
```

```

{
    MYSQL_ROW    row;
    MYSQL_FIELD  *field;
    unsigned long col_len;
    unsigned int  i;

    /* Określenie szerokości wyświetlanych kolumn; zbiór wynikowy musi być */
    /* wygenerowany za pomocą funkcji mysql_store_result(), a nie mysql_use_result(). */
    mysql_field_seek (res_set, 0);
    for (i = 0; i < mysql_num_fields (res_set); i++)
    {
        field = mysql_fetch_field (res_set);
        col_len = strlen (field->name);
        if (col_len < field->max_length)
            col_len = field->max_length;
        if (col_len < 4 && !IS_NOT_NULL (field->flags))
            col_len = 4; /* 4 = długość słowa "NULL". */
        field->max_length = col_len; /* Wyzercowanie informacji o kolumnie. */
    }

    print_dashes (res_set);
    fputc ('|', stdout);
    mysql_field_seek (res_set, 0);
    for (i = 0; i < mysql_num_fields (res_set); i++)
    {
        field = mysql_fetch_field (res_set);
        printf (" %-*s |", (int) field->max_length, field->name);
    }
    fputc ('\n', stdout);
    print_dashes (res_set);

    while ((row = mysql_fetch_row (res_set)) != NULL)
    {
        mysql_field_seek (res_set, 0);
        fputc ('|', stdout);
        for (i = 0; i < mysql_num_fields (res_set); i++)
        {
            field = mysql_fetch_field (res_set);
            if (row[i] == NULL) /* Wyświetlenie słowa "NULL". */
                printf (" %-*s |", (int) field->max_length, "NULL");
            else if (IS_NUM (field->type)) /* Wyświetlenie wartości wyrównanej do prawej strony. */
                printf (" %-*s |", (int) field->max_length, row[i]);
            else /* Wyświetlenie wartości wyrównanej do lewej strony. */
                printf (" %-*s |", (int) field->max_length, row[i]);
        }
        fputc ('\n', stdout);
    }
    print_dashes (res_set);
    printf ("Liczba zwróconych rekordów: %lu\n",
           (unsigned long) mysql_num_rows (res_set));
}

```

Biblioteka klienta MySQL zapewnia wiele sposobów na uzyskanie dostępu do struktur informacji o kolumnach. Na przykład, kod przedstawiony w poprzednim przykładzie wielokrotnie uzyskuje dostęp do wspomnianych struktur za pomocą pętli o następującej ogólnej postaci:

```
mysql_field_seek (res_set, 0);
for (i = 0; i < mysql_num_fields (res_set); i++)
{
    field = mysql_fetch_field (res_set);
    ...
}
```

Jednak połączenie funkcji `mysql_field_seek()` i `mysql_fetch_field()` to tylko jeden z wielu sposobów na pobranie struktur `MYSQL_FIELD`.

Użycie programu `metadata` do wyświetlenia metadanych zbioru wynikowego

Dystrybucja `sampdb` zawiera kod źródłowy programu o nazwie `metadata`, który możesz skompilować i uruchomić, aby się przekonać, jakie metadane generują różnego rodzaju zapytania. Program prosi o podanie zapytania, uruchamia je i wyświetla metadane zbioru wynikowego zamiast jego zawartości. Dla porównania, uruchom program `mysql` wraz z opcją `--column-type-info`, a następnie wykonaj te same zapytania.

7.4.7. Kodowanie znaków specjalnych i danych binarnych

Programy wykonujące zapytania muszą zachować ostrożność podczas obsługi określonych znaków. Na przykład, w celu umieszczenia znaku cytowania w cytowanym ciągu tekstowym konieczne jest jego dwukrotne podanie lub poprzedzenie ukośnikiem:

```
'Mc' 'Donalds'
'Mc\'Donalds'
```

W tym punkcie dowiesz się, jak rozwiązać kwestie związane z cytowaniem w ciągach tekstowych oraz jak pracować z danymi binarnymi.

7.4.7.1. Praca z ciągami tekstowymi zawierającymi znaki specjalne

Jeżeli zostaną umieszczone dosłownie w zapytaniu, wartości zawierające znaki cytowania, bajty null lub ukośniki mogą spowodować problemy w trakcie próby wykonania takiego zapytania. W tym podpunkcie poznasz naturę problemu i dowiesz się, jak go rozwiązać.

Przyjmujemy założenie, że konieczne jest przygotowanie zapytania `SELECT` na podstawie zawartości zakończonego bajtem null ciągu tekstowego wskazywanego przez zmienną `name_val`:

```
char stmt_buf[1024];
sprintf (stmt_buf, "SELECT * FROM mytbl WHERE name='%s'", name_val);
```

Jeżeli wartość `name_val` będzie w postaci takiej jak `O'Malley, Brian`, przygotowane zapytanie jest nieprawidłowe, ponieważ znak cytowania został umieszczony w innych znakach cytowania:

```
SELECT * FROM mytbl WHERE name='O'Malley, Brian'
```


Wewnętrzny znak cytowania musisz potraktować w sposób specjalny, aby serwer nie zinterpretował go jako końcowego znaku cytowania dla `name`. Standardową konwencją SQL służącą do tego celu jest poprzedzenie znaku cytowania drugim znakiem cytowania. MySQL rozpoznaje tę konwencję, a także pozwala na poprzedzanie znaków cytowania ukośnikiem. Dlatego też zapytanie można zapisać w dwóch poniższych formach:

```
SELECT * FROM mytbl WHERE name='0''Malley, Brian'  
SELECT * FROM mytbl WHERE name='0\'\'Malley, Brian'
```

Aby rozwiązać problem, użyj funkcji `mysql_real_escape_string()`, która koduje znaki specjalne i pozwala na ich stosowanie w cytowanych ciągach tekstowych. Znaki uznawane za specjalne przez funkcję `mysql_real_escape_string()` to bajt null, apostrof, cudzysłów, ukośnik, znak nowego wiersza, znak powrotu do początku wiersza i *Control+Z*. (Ostatni z wymienionych ma znaczenie specjalne w systemie Windows, gdzie czasami oznacza koniec pliku).

Kiedy powinno się używać funkcji `mysql_real_escape_string()`? Najbezpieczniejsza odpowiedź brzmi: zawsze. Jednak jeśli masz pewność w zakresie formatu danych i ich poprawności — na przykład z powodu wcześniejszego przeprowadzenia ich weryfikacji — to nie trzeba kodować znaków specjalnych. Na przykład, jeśli pracujesz z ciągami tekstowymi składającymi się wyłącznie z cyfr i znaków minus, czyli przedstawiającymi poprawne numery telefonów, wtedy nie musisz wywoływać funkcji `mysql_real_escape_string()`. W pozostałych przypadkach prawdopodobnie powinieś.

Funkcja `mysql_real_escape_string()` koduje problematyczne znaki przez zastąpienie ich dwuznakową sekwencją rozpoczynającą się ukośnikiem. Na przykład, bajt null staje się `\0`, gdzie 0 to możliwa do wyświetlenia cyfra zero w standardzie ASCII, a nie null. Ukośnik, apostrof i cudzysłów stają się odpowiednio `\\`, `\'` i `\"`.

Aby użyć funkcji `mysql_real_escape_string()`, należy ją wywołać w następujący sposób:

```
to_len = mysql_real_escape_string (conn, to_str, from_str, from_len);
```

Funkcja `mysql_real_escape_string()` koduje `from_str` i zapisuje wynik w `to_str`. Ponadto, dodaje znak oznaczający koniec ciągu tekstowego, co jest wygodne, ponieważ tak wygenerowanego ciągu tekstowego można używać wraz z funkcjami takimi jak `strcpy()`, `strlen()` lub `printf()`.

Argument `from_str` prowadzi do bufora typu `char`, zawierającego ciąg tekstowy przeznaczony do zakodowania. Wspomniany ciąg tekstowy może zawierać cokolwiek, nawet dane binarne. Z kolei argument `to_str` prowadzi do istniejącego bufora typu `char`, w którym ma zostać zapisany ciąg tekstowy. Nie przekazuje wskaźnika `NULL` lub niezainicjalizowanego, oczekując, że funkcja `mysql_real_escape_string()` zaalokuje wymaganą pamięć. Wielkość bufora wskazywanego przez `to_str` musi wynosić przynajmniej $(\text{from_len} * 2) + 1$ bajtów. (Istnieje możliwość, że wszystkie znaki w `from_str` będą wymagały zakodowania za pomocą dwóch znaków, dodatkowy bajt jest wymagany dla znaku null, wskazującego koniec ciągu tekstowego).

Argumenty `from_len` i `to_len` to wartości typu `unsigned long`. Argument `from_len` wskazuje wielkość danych w `from_str`. Podanie wielkości jest konieczne, ponieważ `from_str`

może zawierać bajty null i nie może być traktowany jako ciąg tekstowy zakończony znakiem null. Argument `to_len`, wartość zwrótna funkcji `mysql_real_escape_string()`, to rzeczywista długość zakodowanego ciągu tekstowego bez uwzględnienia znaku null oznaczającego koniec ciągu tekstowego.

Po zakończeniu działania funkcji `mysql_real_escape_string()` zakodowany ciąg tekstowy zapisany w `to_str` może być traktowany jako ciąg tekstowy zakończony bajtem null, ponieważ wszystkie bajty null z `from_str` są wyświetlane jako sekwencja `\0`.

Poniżej przedstawiono zmodyfikowany kod tworzący zapytanie SELECT działające prawidłowo nawet z wartościami zawierającymi znaki cytowania:

```
char stmt_buf[1024], *p;

p = strcpy (stmt_buf, "SELECT * FROM mytbl WHERE name='");
p += strlen (p);
p += mysql_real_escape_string (conn, p, name_val, strlen (name_val));
*p++ = '\'';
*p = '\0';
```

To prawda, powyższy kod jest nieco skomplikowany. Można go nieco uprościć, ale kosztem użycia drugiego bufora:

```
char stmt_buf[1024], buf[1024];

(void) mysql_real_escape_string (conn, buf, name_val, strlen (name_val));
sprintf (stmt_buf, "SELECT * FROM mytbl WHERE name='%s'", buf);
```

Bardzo ważne jest upewnienie się, że bufor przekazywany funkcji `mysql_real_escape_string()` naprawdę istnieje. Przeanalizuj poniższy przykład, w którym złamano wymienioną zasadę:

```
char *from_str = "pewny ciąg tekstowy";
char *to_str;
unsigned long len;

len = mysql_real_escape_string (conn, to_str, from_str, strlen (from_str));
```

W czym tkwi problem? Argument `to_str` musi prowadzić do istniejącego bufora. Ponieważ nie prowadzi, to nie zostaje zainicjalizowany, a tym samym może wskazywać losowo wybraną lokalizację. Nie przekazuj niezainicjalizowanego wskaźnika jako argumentu `to_str` funkcji `mysql_real_escape_string()`, o ile nie chcesz radośnie wskazać dowolnie wybranego miejsca w pamięci.

7.4.7.2. Praca z danymi binarnymi

Inna problematyczna sytuacja wiąże się z użyciem dowolnych danych binarnych w zapytaniu. Może się to zdarzyć na przykład w aplikacjach przechowujących obrazy w bazie danych. Ponieważ wartość binarna może przechowywać dowolne znaki (w tym także bajty null, znaki cytowania i ukośniki), nie można jej uznać za bezpieczną do bezpośredniego osadzenia w zapytaniu.

Funkcja `mysql_real_escape_string()` ma istotne znaczenie podczas pracy z danymi binarnymi. W tym podpunkcie na przykładzie danych obrazu odczytywanych z pliku dowiesz się, jak pracować z danymi binarnymi. Przedstawione tutaj koncepcje i rozwiązania można również stosować względem innych form danych binarnych.

Przyjmujemy założenie, że chcesz odczytywać obrazy z plików i przechowywać je w tabeli o nazwie `picture` wraz z unikalnym identyfikatorem. Typ `MEDIUMBLOB` jest dobrym wyborem dla wartości binarnych mniejszych niż 16 MB, a więc można użyć tabeli utworzonej w następujący sposób:

```
CREATE TABLE picture
(
    pict_id INT NOT NULL PRIMARY KEY,
    pict_data MEDIUMBLOB
);
```

W celu faktycznego pobrania obrazu z pliku i umieszczenia go w tabeli `picture` przedstawiona poniżej funkcja `load_image()` wykonuje to zadanie, opierając się na podanym identyfikatorze i wskaźniku do otwartego pliku zawierającego dane obrazu:

```
int load_image (MYSQL *conn, int id, FILE *f)
{
    char          stmt_buf[1024*1024], buf[1024*10], *p;
    unsigned long from_len;
    int           status;

    /* Rozpoczęcie tworzenia zapytania INSERT, dodanie wartości id. */
    sprintf (stmt_buf,
        "INSERT INTO picture (pict_id,pict_data) VALUES (%d,''",
            id);
    p = stmt_buf + strlen (stmt_buf);
    /* Odczyt danych z pliku fragmentami, zakodowanie każdego fragmentu */
    /* i umieszczenie na końcu zapytania. */
    while ((from_len = fread (buf, 1, sizeof (buf), f)) > 0)
    {
        /* Nie przepelnij bufora zapytania! */
        if (p + (2*from_len) + 3 > stmt_buf + sizeof (stmt_buf))
        {
            print_error (NULL, "Obraz jest zbyt duży.");
            return (1);
        }
        p += mysql_real_escape_string (conn, p, buf, from_len);
    }
    *p++ = '\\';
    *p++ = ')';
    status = mysql_real_query (conn, stmt_buf, (unsigned long) (p - stmt_buf));
    return (status);
}
```

Funkcja `load_image()` nie alokuje zbyt dużego bufora zapytania (1 MB), a więc działa ze względnie niewielkimi obrazami. W rzeczywistej aplikacji lepszym rozwiązaniem jest dynamiczna alokacja bufora na podstawie wielkości pliku obrazu.

Pobranie danych obrazu (lub wartości binarnej) z bazy danych nie jest aż takim problemem, jak ich wstawienie do bazy danych. Dane są dostępne w postaci niezmodyfikowanej w zmiennej `MYSQL_ROW`, a ich wielkość można sprawdzić za pomocą wywołania `mysql_fetch_lengths()`. Po prostu upewnij się, że wartość traktujesz jak ciąg tekstowy o określonej długości, a nie jak ciąg tekstowy zakończony bajtem null.

7.5. Program do interaktywnego wykonywania zapytań

Na tym etapie możemy zebrać opracowane dotąd fragmenty kodu i utworzyć prostego klienta o nazwie `exec_stmt`, pozwalającego na interaktywne wykonywanie zapytań. Wspomniany program umożliwi użytkownikowi wprowadzenie zapytania, a następnie wykona je za pomocą procedury obsługi ogólnego przeznaczenia zapytań `process_statement()`. Wyniki zostaną wyświetlone za pomocą funkcji `process_result_set()`, również przygotowanej już wcześniej.

Pod wieloma względami program `exec_stmt` jest podobny do `mysql`, choć oczywiście oferuje znacznie mniejszą funkcjonalność. Istnieje wiele ograniczeń w zakresie danych wejściowych akceptowanych przez `exec_stmt`:

- Każdy wiersz danych wejściowych musi zawierać pojedyncze, pełne zapytanie.
- Zapytania nie mogą kończyć się średnikiem lub znakami `\g`.
- Poza zapytaniami SQL jedyne polecenia rozpoznawane przez klienta to `quit` i `\q`, które powodują zakończenie jego działania. Ten sam skutek ma również naciśnięcie klawiszy `Ctrl+D`.

Okazuje się, że utworzenie programu `exec_stmt` jest niezwykle łatwe (to jedynie około dziesięciu nowych wierszy kodu). Wszystko, czego potrzebujemy, dostarcza nam szkielet programu klienta (`connect2.c`), a także inne opracowane dotąd funkcje. Jedyne nowe fragmenty konieczne do dodania to pętla pobierająca wiersze danych wejściowych i wykonująca je.

W celu przygotowania programu `exec_stmt` pracę rozpoczynamy od skopiowania kodu szkieletu klienta (`connect2.c`) do pliku `exec_stmt.c`. Następnie dodajemy kod funkcji `process_statement()`, `process_result_set()` i `print_dashes()`. Na końcu w pliku `exec_stmt.c` wyszukujemy wiersz `main()` o następującej treści:

```
/* ... wykonanie zapytań i przetworzenie wyników ... */
```

Powyższy wiersz zastępujemy poniższym kodem pętli:

```
while (1)
{
    char buf[10000];

    fprintf(stderr, "query> ");
    if (fgets(buf, sizeof(buf), stdin) == NULL) /* Wyświetlenie znaku zachęty. */
        break; /* Odczyt polecenia. */
    if (strcmp(buf, "quit\n") == 0 || strcmp(buf, "\\q\n") == 0)
```

```

        break;
    process_statement (conn, buf);          /* Wykonanie zapytania. */
}

```

Kompilacja pliku *exec_stmt.c* powoduje wygenerowanie pliku *exec_stmt.o*; po jego połączeniu z biblioteką klienta otrzymasz program *exec_stmt*. W ten sposób przygotowałeś interaktywnego klienta MySQL, za pomocą którego możesz wykonać dowolne zapytanie i wyświetlić jego wynik. Poniższy przykład pokazuje sposób działania programu z zapytaniami typu SELECT i innymi niż SELECT, a także zawierającymi błędy:

```

% ./exec_stmt
query> USE sampdb
Liczba rekordów, których dotyczyło zapytanie: 0
query> SELECT DATABASE(), USER()
+-----+-----+
| DATABASE() | USER() |
+-----+-----+
| sampdb     | sampadm@localhost |
+-----+-----+
Liczba zwróconych rekordów: 1
query> SELECT COUNT(*) FROM president
+-----+
| COUNT(*) |
+-----+
|         43 |
+-----+
Liczba zwróconych rekordów: 1
query> SELECT last_name, first_name FROM president ORDER BY last_name LIMIT 3
+-----+-----+
| last_name | first_name |
+-----+-----+
| Adams    | John      |
| Adams    | John Quincy |
| Arthur   | Chester A. |
+-----+-----+
Liczba zwróconych rekordów: 3
query> CREATE TABLE t (i INT)
Liczba rekordów, których dotyczyło zapytanie: 0
query> SELECT j FROM t
Nie można było wykonać zapytania
Error 1054 (42S22): Unknown column 'j' in 'field list'
query> USE mysql
Nie można było wykonać zapytania
Error 1044 (42000): Access denied for user 'sampadm'@'localhost' to
database 'mysql'

```

7.6. Utworzenie klienta z obsługą SSL

MySQL zapewnia obsługę protokołu SSL, a więc można utworzyć własne programy uzyskujące dostęp do serwera za pomocą bezpiecznego połączenia. Aby przekonać się, jak to zrobić, w tym podrozdziale przedstawiono proces modyfikacji klienta *exec_stmt* i utworzenia na jego podstawie klienta o nazwie *exec_stmt_ssl*, który działa w taki sam

sposób, ale pozwala na nawiązywanie szyfrowanych połączeń. W celu zapewnienia prawidłowego działania klienta `exec_stmt_ssl` serwer MySQL musi być utworzony wraz z obsługą SSL i uruchomiony wraz z opcjami identyfikującymi pliki certyfikatu oraz kluczy. Konieczne będzie także uzyskanie plików certyfikatu i kluczy oraz ich konfiguracja po stronie klienta. Więcej informacji na ten temat znajdziesz w podrozdziale 13.5, zatytułowanym „Konfiguracja bezpiecznych połączeń za pomocą SSL”.

Dystrybucja `sampdb` zawiera plik źródłowy o nazwie `exec_stmt_ssl.c`, na podstawie którego tworzony jest program klienta `exec_stmt_ssl`. Przedstawiona poniżej procedura opisuje sposób utworzenia pliku `exec_stmt_ssl.c` na podstawie `exec_stmt.c`:

1. Zawartość pliku `exec_stmt.c` skopiuj do `exec_stmt_ssl.c`. Pozostałe kroki należy przeprowadzać w pliku `exec_stmt_ssl.c`.
2. Aby umożliwić kompilatorowi wykrycie dostępnej obsługi protokołu SSL, plik nagłówkowy MySQL o nazwie `my_config.h` definiuje symbol `HAVE_OPENSSL`. Oznacza to, że w trakcie tworzenia kodu powiązanego z SSL stosujesz poniższą konstrukcję, aby kod był zignorowany, jeśli nie można użyć protokołu SSL:

```
#ifndef HAVE_OPENSSL
...miejsce na kod powiązany z SSL...
#endif
```

Nie ma potrzeby dołączania pliku `my_config.h`, ponieważ jest on dołączany przez `my_global.h`, który z kolei jest dołączany przez `exec_stmt_ssl.c`.

3. Zmodyfikuj tablicę `my_opts`, zawierającą struktury informacji opcji, aby zawierała także struktury przeznaczone do obsługi opcji powiązanych z SSL (`--ssl-ca`, `--ssl-key` itd.). Najłatwiejszym sposobem będzie dołączenie zawartości pliku `sslopt-longopts.h` do tablicy za pomocą dyrektywy `#include`. Po wprowadzeniu zmian tablica `my_opts` przedstawia się następująco:

```
static struct my_option my_opts[] = /* Struktury informacji opcji. */
{
    {"help", '?', "Wyświetlenie tego komunikatu pomocy i zakończenie programu.",
     NULL, NULL, NULL,
     GET_NO_ARG, NO_ARG, 0, 0, 0, 0, 0, 0},
    {"host", 'h', "Komputer, z którym chcesz się połączyć.",
     (uchar **) &opt_host_name, NULL, NULL,
     GET_STR, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
    {"password", 'p', "Hasło.",
     (uchar **) &opt_password, NULL, NULL,
     GET_STR, OPT_ARG, 0, 0, 0, 0, 0, 0},
    {"port", 'P', "Numer portu.",
     (uchar **) &opt_port_num, NULL, NULL,
     GET_UINT, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
    {"socket", 'S', "Ścieżka dostępu do gniazda.",
     (uchar **) &opt_socket_name, NULL, NULL,
     GET_STR, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
    {"user", 'u', "Nazwa użytkownika.",
     (uchar **) &opt_user_name, NULL, NULL,
     GET_STR, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
#include <sslopt-longopts.h>
    { NULL, 0, NULL, NULL, NULL, NULL, GET_NO_ARG, NO_ARG, 0, 0, 0, 0, 0, 0 }
};
```

Plik *sslopt-longopts.h* jest publicznym plikiem nagłówkowym MySQL.

Jego (nieco inaczej sformatowana) zawartość przedstawia się następująco:

```
#ifndef HAVE_OPENSSL
{"ssl", OPT_SSL_SSL,
 "Włączenie SSL dla połączenia (automatycznie włączenie z innymi flagami).
 Pominiecie za pomocą opcji --skip-ssl.",
 (uchar **) &opt_use_ssl, (uchar **) &opt_use_ssl, 0,
 GET_BOOL, NO_ARG, 0, 0, 0, 0, 0, 0},
{"ssl-ca", OPT_SSL_CA,
 "Plik CA w formacie PEM (sprawdź dokumentację OpenSSL, oznacza użycie --ssl).",
 (uchar **) &opt_ssl_ca, (uchar **) &opt_ssl_ca, 0,
 GET_STR, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
{"ssl-capath", OPT_SSL_CAPATH,
 "Katalog CA (sprawdź dokumentację OpenSSL, oznacza użycie --ssl).",
 (uchar **) &opt_ssl_capath, (uchar **) &opt_ssl_capath, 0,
 GET_STR, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
{"ssl-cert", OPT_SSL_CERT, "Certyfikat X509 w formacie PEM (oznacza użycie --ssl).",
 (uchar **) &opt_ssl_cert, (uchar **) &opt_ssl_cert, 0,
 GET_STR, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
{"ssl-cipher", OPT_SSL_CIPHER, "Użycie szyfrowania SSL (oznacza użycie --ssl).",
 (uchar **) &opt_ssl_cipher, (uchar **) &opt_ssl_cipher, 0,
 GET_STR, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
{"ssl-key", OPT_SSL_KEY, "Klucz X509 w formacie PEM (oznacza użycie --ssl).",
 (uchar **) &opt_ssl_key, (uchar **) &opt_ssl_key, 0,
 GET_STR, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
#endif

#ifndef MYSQL_CLIENT
{"ssl-verify-server-cert", OPT_SSL_VERIFY_SERVER_CERT,
 "Sprawdzenie \"Common Name\" serwera w jego pliku względem nazwy komputera
 użytej podczas nawiązywania połączenia. Ta opcja jest domyślnie wyłączona.",
 (uchar **) &opt_ssl_verify_server_cert,
 (uchar **) &opt_ssl_verify_server_cert, 0,
 GET_BOOL, NO_ARG, 0, 0, 0, 0, 0, 0},
#endif

#endif /* HAVE_OPENSSL */
```

4. Struktury opcji zdefiniowane przez *sslopt-longopts.h* odwołują się do wartości `OPT_SSL`, `OPT_SSL_KEY` itd. Są one używane dla krótkich wersji opcji lub ich kodów i muszą być zdefiniowane w programie, co odbywa się przez umieszczenie poniższych wierszy przed definicją tablicy `my_opts`:

```
#ifndef HAVE_OPENSSL
enum options_client
{
    OPT_SSL_SSL=256,
    OPT_SSL_KEY,
    OPT_SSL_CERT,
    OPT_SSL_CA,
    OPT_SSL_CAPATH,
    OPT_SSL_CIPHER,
    OPT_SSL_VERIFY_SERVER_CERT
};
#endif
```

Podczas tworzenia własnych aplikacji definiujących kody dla innych opcji upewnij się, że wartości wspomnianych kodów są inne niż wartości symboli `OPT_SSL_XXX`.

5. Zdefiniowane w pliku *sslopt-longopts.h* struktury opcji powiązanych w SSL odwołują się do zestawu zmiennych używanych do przechowywania wartości opcji. Aby je zadeklarować, należy użyć dyrektywy `#include` i dołączyć do programu zawartość pliku *sslopt-vars.h* przed definicją tablicy `my_opts`.

Zawartość pliku *sslopt-vars.h* przedstawia się następująco:

```
#ifndef HAVE_OPENSSL
static my_bool opt_use_ssl = 0;
static char *opt_ssl_ca = 0;
static char *opt_ssl_capath = 0;
static char *opt_ssl_cert = 0;
static char *opt_ssl_cipher = 0;
static char *opt_ssl_key = 0;
#endif
#ifdef MYSQL_CLIENT
static my_bool opt_ssl_verify_server_cert = 0;
#endif
#endif
```

6. W kodzie funkcji `get_one_option()` trzeba tuż przy końcu dodać wiersz dołączający plik nagłówkowy o nazwie *sslopt-case.h*:

```
static my_bool get_one_option (int optid, const struct my_option *opt, char
*argument)
{
    switch (optid)
    {
        case '?':
            my_print_help (my_opts); /* Wyświetlenie komunikatu pomocy. */
            exit (0);
        case 'p': /* Hasło. */
            if (!argument) /* Brak podanej wartości; należy poprosić o nią później. */
                ask_password = 1;
            else /* Skopiowanie hasła i nadpisanie początkowego. */
            {
                opt_password = strdup (argument);
                if (opt_password == NULL)
                {
                    print_error (NULL, "Nie udało się alokować bufora hasła.");
                    exit (1);
                }
                while (*argument)
                    *argument++ = 'x';
                ask_password = 0;
            }
            break;
    }
    #include <sslopt-case.h>
    return (0);
}
```

Plik *sslopt-case.h* zawiera dodatkowe składniki konstrukcji `switch()`, pozwalające na wykrycie użycia jakiegokolwiek opcji SSL i ustawienie wówczas odpowiedniej zmiennej `opt_use_ssl`. Wspomniany kod przedstawia się następująco:


```

#ifdef HAVE_OPENSSL
    case OPT_SSL_KEY:
    case OPT_SSL_CERT:
    case OPT_SSL_CA:
    case OPT_SSL_CAPATH:
    case OPT_SSL_CIPHER:
    /*
        Włączenie użycia SSL, jeśli została użyta dowolna opcja ssl.
        SSL można później wyłączyć za pomocą opcji --skip-ssl lub --ssl=0.
    */
    opt_use_ssl = 1;
    break;
#endif

```

Efektom wprowadzonej zmiany jest to, że po przetworzeniu opcji istnieje możliwość ustalenia, czy użytkownik chce użyć bezpiecznego połączenia. Wystarczy po prostu sprawdzić wartość zmiennej `opt_use_ssl`.

Jeżeli używasz przedstawionej procedury, zaprezentowane wcześniej zwykłe funkcje `load_defaults()` i `handle_options()` automatycznie zajmą się przetwarzaniem opcji powiązanych z SSL i konfiguracją ich wartości. Twoim jedynym zadaniem jest przekazanie bibliotece klienta informacji opcji SSL jeszcze przed nawiązaniem połączenia z serwerem, jeśli opcje wskazują, że użytkownik chce skorzystać z połączenia SSL.

W tym celu wywołaj funkcję `mysql_ssl_set()` po `mysql_init()`, ale przed wywołaniem `mysql_real_connect()`. Sekwencja przedstawia się następująco:

```

/* Inicjalizacja uchwytu połączenia. */
conn = mysql_init (NULL);
if (conn == NULL)
{
    print_error (NULL, "Wywołanie mysql_init() zakończyło się niepowodzeniem
        (prawdopodobnie z powodu braku pamięci).");
    exit (1);
}
#ifdef HAVE_OPENSSL
/* Przekazanie bibliotece klienta informacji SSL. */
if (opt_use_ssl)
    mysql_ssl_set (conn, opt_ssl_key, opt_ssl_cert, opt_ssl_ca,
        opt_ssl_capath, opt_ssl_cipher);
mysql_options (conn, MYSQL_OPT_SSL_VERIFY_SERVER_CERT,
    (char*)&opt_ssl_verify_server_cert);
#endif
/* Nawiązanie połączenia z serwerem. */
if (mysql_real_connect (conn, opt_host_name, opt_user_name, opt_password,
    opt_db_name, opt_port_num, opt_socket_name, opt_flags) == NULL)
{
    print_error (conn, "Wywołanie mysql_real_connect() zakończyło się niepowodzeniem.");
    mysql_close (conn);
    exit (1);
}

```

Powyższy kod nie testuje funkcji `mysql_ssl_set()` w celu sprawdzenia, czy wystąpił błąd. Wszelkie problemy z informacjami dostarczonymi do wymienionej funkcji powodują zgłoszenie błędu w trakcie wywołania funkcji `mysql_real_connect()`.

Przeprowadź kompilację pliku `exec_stmt_ssl.c` i wygeneruj program `exec_stmt_ssl`, a następnie go uruchom. Przy założeniu, że wywołanie funkcji `mysql_real_connect()` zakończy się powodzeniem, będziesz mógł wykonywać zapytania. Jeżeli program `exec_stmt_ssl` zostanie uruchomiony wraz z odpowiednimi opcjami SSL, komunikacja z serwerem będzie odbywała się przez szyfrowane połączenie. Możesz się o tym przekonać za pomocą przedstawionego poniżej zapytania:

```
SHOW STATUS LIKE 'Ssl_cipher'
```

Jeśli używane jest szyfrowanie, wartość `Ssl_cipher` będzie niepusta. (Aby ułatwić całą procedurę, wersja programu `exec_stmt_ssl` w dystrybucji `sampdb` automatycznie wykonuje wymienione zapytanie i wyświetla jego wynik).

7.7. Jednoczesne wykonywanie wielu zapytań

Biblioteka klienta MySQL zapewnia możliwość jednoczesnego wykonywania wielu zapytań. Dzięki temu do serwera możesz wysłać ciąg tekstowy zawierający wiele zapytań rozdzielonych średnikami, a następnie kolejno otrzymać zbiory wynikowe.

Możliwość jednoczesnego wykonywania zapytań nie jest domyślnie włączona, a więc trzeba poinformować serwer o chęci skorzystania z niej. Można to zrobić na dwa sposoby. Pierwszy polega na dodaniu opcji `CLIENT_MULTI_STATEMENTS` do flag argumentu funkcji `mysql_real_connect()` w trakcie jej wywoływania:

```
opt_flags |= CLIENT_MULTI_STATEMENTS;
if (mysql_real_connect (conn, opt_host_name, opt_user_name, opt_password,
    opt_db_name, opt_port_num, opt_socket_name, opt_flags) == NULL)
{
    print_error (conn, "Wywołanie mysql_real_connect() zakończyło się niepowodzeniem.");
    mysql_close (conn);
    exit (1);
}
```

Drugi polega na wywołaniu funkcji `mysql_set_server_option()` i włączeniu wspomnianej możliwości dla istniejącego połączenia, na przykład:

```
if (mysql_set_server_option (conn, MYSQL_OPTION_MULTI_STATEMENTS_ON) != 0)
    print_error (conn, "Nie można było włączyć trybu wykonywania wielu zapytań.");
```

Która z wymienionych metod jest preferowana? Jeżeli program nie używa procedur składowanych, wtedy obie są odpowiednie. Natomiast jeśli program używa procedur składowanych i wykonuje wywołania `CALL` zwracające zbiór wynikowy, wtedy skorzystaj z pierwszego sposobu. Powód jest prosty: opcja `CLIENT_MULTI_STATEMENTS` powoduje również włączenie opcji `CLIENT_MULTI_RESULTS`, która musi być włączona, ponieważ w przeciwnym razie nastąpi błąd, gdy procedura składowana spróbuje zwrócić wynik. (Lepszym rozwiązaniem może być dodanie `CLIENT_MULTI_RESULT` do flag argumentu funkcji `mysql_real_connect()`, ponieważ w ten sposób wyraźnie włączasz wymienioną opcję).

Dwie wymienione poniżej funkcje tworzą podstawę mechanizmu sprawdzania bieżącego stanu pobierania wyniku podczas jednoczesnego przetwarzania wielu zbiorów wynikowych:

- Wartość zwrrotna funkcji `mysql_more_results()` jest niezerowa, jeśli dostępny jest więcej niż jeden zbiór wynikowy. W przeciwnym razie funkcja zwraca zero.
- Wartością zwrrotną funkcji `mysql_next_result()` są informacje o stanie. Ponadto, funkcja inicjuje pobranie kolejnego zestawu, gdy dostępnych jest więcej zbiorów wynikowych. Wartość zero dla stanu oznacza dostępność kolejnych wyników, -1 brak wyników, natomiast wartość większa niż zero oznacza błąd.

Wymienione funkcje możesz wykorzystać przez umieszczenie w pętli kodu pobierającego wynik. Po pobraniu wyniku za pomocą zwykłego kodu przeznaczonego do tego celu należy sprawdzić, czy istnieją jakiegokolwiek inne wyniki do pobrania. Jeśli tak, wtedy wykonuje się kolejną iterację pętli. Jeśli nie ma więcej wyników, następuje opuszczenie pętli. W zależności od struktury pętli może w ogóle nie wystąpić konieczność wywołania funkcji `mysql_more_results()`, ponieważ dostępność kolejnych wyników można określić na podstawie wartości zwrótej funkcji `mysql_next_result()`.

W punkcie 7.4.3, zatytułowanym „Procedury obsługi zapytań ogólnego przeznaczenia”, utworzyliśmy funkcję o nazwie `process_statement()`, odpowiedzialną za wykonanie zapytania i pobieranie jego wyniku lub wyświetlenie liczby rekordów, których dotyczyło zapytanie. Dzięki umieszczeniu w pętli kodu pobierającego wynik i wykorzystaniu funkcji `mysql_next_result()` możemy opracować podobną funkcję do wcześniej wymienionej. Nowa funkcja nosi nazwę `process_multi_statement()` i pozwala na pobranie wielu zbiorów wynikowych:

```
void process_multi_statement (MYSQL *conn, char *stmt_str)
{
    MYSQL_RES *res_set;
    int      status;
    int      keep_going = 1;

    if (mysql_query (conn, stmt_str) != 0) /* Wykonanie zapytania/zapytań zakończyło się niepowodzeniem. */
    {
        print_error (conn, "Nie można było wykonać zapyta(nia/ń).");
        return;
    }

    /* Wykonanie zapytania/zapytań zakończone powodzeniem; wejście do pętli pobrania wyniku. */
    do {
        /* Ustalenie, czy bieżące zapytanie zwraca dane. */
        res_set = mysql_store_result (conn);
        if (res_set) /* Zbiór wynikowy został zwrócony. */
        {
            /* Przetworzenie rekordów i usunięcie zbioru wynikowego z pamięci. */
            process_result_set (conn, res_set);
            mysql_free_result (res_set);
        }
        else /* Zbiór wynikowy nie został zwrócony. */
        {
            /*
             * Czy brak zbioru wynikowego oznacza brak danych wyjściowych zapytania,

```

```

        /* czy raczej wskazuje na wystąpienie błędu?
        */
        if (mysql_field_count (conn) == 0)
        {
            /*
            * Zapytanie nie wygenerowało zbioru wynikowego (to nie było zapytanie typu SELECT,
            * SHOW, DESCRIBE itd.); wystarczy jedynie ustalić liczbę rekordów, których dotyczyło.
            */
            printf ("Liczba rekordów, których dotyczyło zapytanie: %lu\n",
                    (unsigned long) mysql_affected_rows (conn));
        }
        else /* Wystąpił błąd. */
        {
            print_error (conn, "Nie można było pobrać zbioru wynikowego.");
            keep_going = 0;
        }
    }
    /* Ustalenie, czy dostępnych jest więcej wyników. */
    /* 0 = tak, -1 = nie, >0 = błąd */
    status = mysql_next_result (conn);
    if (status != 0) /* Brak więcej wyników lub wystąpił błąd. */
    {
        keep_going = 0;
        if (status > 0) /* Błąd. */
            print_error (conn, "Nie można było wykonać zapytania.");
    }
} while (keep_going);
}

```

Jeżeli chcesz, możesz sprawdzić, czy wartością zwrótną funkcji `mysql_next_result()` jest zero, a następnie opuścić pętlę, gdy wartość jest niezerowa. Wadą takiej uproszczonej strategii jest to, że w przypadku braku kolejnych wyników nie wiadomo, czy po prostu pobrane zostały już wszystkie wyniki, czy raczej wystąpił błąd. Innymi słowy, nie wiadomo, czy należy wyświetlić komunikat błędu.

7.8. Używanie zapytań preinterpretowanych

We wcześniejszych podrozdziałach kod odpowiedzialny za przetwarzanie zapytań SQL był oparty na zestawie dostarczanych przez bibliotekę klienta funkcji, które wysyłały i pobierały wszystkie informacje w postaci ciągu tekstowego. W tym podrozdziale dowiesz się, jak używać binarnego protokołu klient-serwer. Wspomniany protokół binarny zapewnia obsługę zapytań preinterpretowanych i pozwala na transmisję wartości danych w ich rodzimym formacie.

Nie wszystkie zapytania mogą być preinterpretowane przez serwer. Początkowa implementacja zapytań preinterpretowanych zapewniała jedynie obsługę zapytań `CREATE TABLE`, `DELETE`, `DO`, `INSERT`, `REPLACE`, `SELECT`, `SET`, `UPDATE` oraz większości odmian `SHOW`. Od tamtej chwili lista obsługiwanych zapytań systematycznie się wydłuża. Bieżącą listę znajdziesz w podręczniku użytkownika MySQL.

W celu użycia protokołu binarnego konieczne jest utworzenie uchwytu zapytania. Za pomocą wspomnianego uchwytu zapytanie będzie wysyłane do serwera w celu jego „preinterpretowania”, czyli wstępnego przetworzenia. Serwer przeanalizuje zapytanie, zapamięta je i informacje o tym przekaże bibliotece klienta, która z kolei będzie te informacje przechowywać w uchwycie zapytania. Dalsze przetwarzanie zapytania będzie się odbywało za pomocą wspomnianego uchwytu.

Zapytanie preinterpretowane może zostać sparametryzowane przez umieszczenie znaków ?, wskazujących miejsce późniejszego wstawienia wartości danych podczas wykonywania zapytania. Na przykład, zapytanie preinterpretowane może mieć następującą postać:

```
INSERT INTO score (event_id,student_id,score) VALUES(?,?,?)
```

W powyższym zapytaniu znajdują się trzy znaki ? działające w charakterze miejsc zarezerwowanych. Później wystarczy po prostu podać wartości danych, które zostaną umieszczone w tych miejscach. W ten sposób zapytanie będzie uzupełnione w chwili jego wykonywania. Dzięki sparametryzowaniu zapytania można je ponownie wykorzystać: to samo zapytanie może być wykonane wielokrotnie, za każdym razem wraz z nowym zestawem danych. Oznacza to, że tekst zapytania jest wysłany do serwera tylko jednokrotnie, natomiast w trakcie jego każdego wykonywania do serwera przekazywane są tylko odpowiednie dane. Dlatego też używanie zapytań preinterpretowanych wiąże się ze wzrostem wydajności działania serwera:

- Serwer analizuje zapytanie tylko jednokrotnie, a nie w trakcie jego każdego wykonywania.
- Zmniejsza się obciążenie sieci, ponieważ w trakcie każdego wykonywania zapytania do serwera są przekazywane jedynie dane, a nie całe zapytanie.
- Dane są przekazywane bez ich wcześniejszej konwersji na postać ciągu tekstowego, co wiąże się ze spadkiem obciążenia. Na przykład, trzy kolumny podane w poprzednim zapytaniu SELECT są typu INT. Podczas użycia funkcji `mysql_query()` lub `mysql_real_query()` do wykonania podobnego zapytania INSERT konieczne będzie przeprowadzenie konwersji danych na postać ciągów tekstowych, które będą mogły być umieszczone w tekście zapytania. W przypadku używania zapytań preinterpretowanych dane są wysyłane oddzielnie w formacie binarnym.
- Podczas pobierania wyników również nie trzeba przeprowadzać konwersji. W zbiorze wynikowym zwróconym przez zapytanie preinterpretowane wartości inne niż ciągi tekstowe są zwracane w formacie binarnym, bez ich konwersji na postać ciągu tekstowego.

Protokół binarny ma również pewne wady w porównaniu do początkowego protokołu niebinarnego:

- Pozostaje trudniejszy w użyciu, ponieważ w celu transmisji i pobrania danych trzeba przeprowadzić znacznie dłuższy etap konfiguracji.
- Protokół binarny nie obsługuje wszystkich zapytań, na przykład nie działają zapytania USE.

- W przypadku programów interaktywnych można również korzystać z protokołu niebinarnego. W takim przypadku każde zapytanie otrzymywane od użytkownika jest wykonywane tylko jednokrotnie. Tutaj zastosowanie zapytań preinterpretowanych przyniesie tylko niewielką korzyść; największą efektywność mają w przypadku nieustannie wykonywanych zapytań.

Ogólna procedura użycia zapytań preinterpretowanych obejmuje wykonanie wymienionych poniżej kroków:

1. Alokacja uchwytu zapytania przez wywołanie funkcji `mysql_stmt_init()`. Wymieniona funkcja zwraca wskaźnik do uchwytu, który jest następnie wykorzystywany w kolejnych krokach.
2. Wywołanie funkcji `mysql_stmt_prepare()` w celu przekazania serwerowi zapytania do interpretacji oraz powiązania go z uchwytem zapytania. Serwer określa pewne cechy charakterystyczne zapytania, takie jak jego rodzaj i liczbę zdefiniowanych miejsc zarezerwowanych, oraz ustala, czy wykonanie zapytania powoduje wygenerowanie zbioru wynikowego.
3. Jeżeli zapytanie zawiera jakiekolwiek miejsca zarezerwowane, przed jego wykonaniem konieczne jest dostarczenie odpowiednich danych dla wszystkich miejsc zarezerwowanych. W tym celu trzeba skonfigurować strukturę `MYSQL_BIND` dla każdego parametru. Poszczególne struktury wskazują typ danych parametru, jego wartość, czy jego wartością jest `NULL` itd. Dołączenie wymienionej struktury do zapytania odbywa się za pomocą wywołania funkcji `mysql_stmt_bind_param()`.
4. Wywołanie funkcji `mysql_stmt_execute()` w celu wykonania zapytania.
5. Jeżeli zapytanie modyfikuje dane, zamiast wygenerować zbiór wynikowy (na przykład jest zapytaniem typu `INSERT` lub `UPDATE`), następuje wywołanie funkcji `mysql_stmt_affected_rows()` w celu ustalenia liczby rekordów, na które wpłynęło zapytanie.
6. W przypadku zapytania generującego zbiór wynikowy, jeśli chcesz pobrać metadane dotyczące wspomnianego zbioru wynikowego, wywołaj funkcję `mysql_stmt_result_metadata()`. Aby pobrać rekordy, konieczne jest ponowne użycie struktur `MYSQL_BIND`, ale tym razem służą one w charakterze zamienników dla danych otrzymywanych z serwera, a nie danych źródłowych wysyłanych do serwera. Dla każdej kolumny w zbiorze wynikowym konieczne jest skonfigurowanie jednej struktury `MYSQL_BIND`. Będą one zawierały informacje o oczekiwanych w poszczególnych rekordach wartościach do pobrania z serwera. Połączenie struktur z uchwytem zapytania następuje za pomocą wywołania funkcji `mysql_stmt_bind_result()`. Następnie, aby pobrać każdy rekord, trzeba wywołać funkcję `mysql_stmt_fetch()`. Po pobraniu każdego rekordu można uzyskać dostęp do wartości jego kolumn.

Przed wywołaniem funkcji `mysql_stmt_fetch()` opcjonalnym krokiem jest wywołanie funkcji `mysql_stmt_store_result()`. Jeżeli zdecydujesz się na ten krok, wszystkie rekordy zbioru wynikowego zostaną pobrane z serwera w jednej operacji i umieszczone w buforze znajdującym się po stronie klienta. Ponadto, liczbę rekordów w zbiorze wynikowym będzie można ustalić za pomocą wywołania funkcji `mysql_stmt_num_rows()`, która wywołana w innym przypadku zwraca wartość zero.

Po pobraniu zbioru wynikowego należy wywołać funkcję `mysql_stmt_free_result()` w celu zwolnienia zajmowanej przez niego pamięci.

7. Aby ponownie wykonać to samo zapytanie, trzeba powrócić do kroku 3. i podać nowe wartości parametrów.
8. W celu przygotowania innego zapytania za pomocą danego uchwytu trzeba powrócić do kroku 2.
9. Po zakończeniu pracy z uchwytym zapytania usuń go z pamięci za pomocą wywołania funkcji `mysql_stmt_close()`. Jeżeli połączenie klienta zostanie zamknięte, gdy serwer nadal ma zapytania preinterpretowane powiązane z połączeniem, wtedy serwer automatycznie je usunie.

Aplikacja klienta może przygotować wiele zapytań, a następnie wykonywać je w kolejności odpowiedniej dla aplikacji.

Poniżej przedstawiono procedurę tworzenia prostego programu, który wstawia rekordy do tabeli, a następnie je pobiera. We fragmencie programu przetwarzającym zapytanie `INSERT` pokazano użycie miejsc zarezerwowanych w zapytaniu i transmisję do serwera danych, które będą dołączone do zapytania preinterpretowanego w chwili jego wykonywania. Z kolei we fragmencie programu przetwarzającym zapytanie `SELECT` pokazano, jak pobierać zbiór wynikowy wygenerowany przez zapytanie preinterpretowane. Kod źródłowy omawianego tutaj programu znajdziesz w plikach *prepared.c* i *process_prepared_statement.c*, umieszczonych w katalogu *capi* dystrybucji *sampdb*. Tutaj nie zaprezentowano kodu odpowiedzialnego za konfigurację połączenia, ponieważ jest podobny do użytego w poprzednich programach.

Główna część programu, przeprowadzająca konfigurację zapytań preinterpretowanych, przedstawia się następująco:

```
void process_prepared_statements (MYSQL *conn)
{
    MYSQL_STMT *stmt;
    char        *use_stmt = "USE sampdb";
    char        *drop_stmt = "DROP TABLE IF EXISTS t";
    char        *create_stmt =
        "CREATE TABLE t (i INT, f FLOAT, c CHAR(24), dt DATETIME)";

    /* Wybór bazy danych i utworzenie tabeli testowej. */
    if (mysql_query (conn, use_stmt) != 0
        || mysql_query (conn, drop_stmt) != 0
        || mysql_query (conn, create_stmt) != 0)
```

```

{
    print_error (conn, "Nie można było skonfigurować tabeli testowej.");
    return;
}

stmt = mysql_stmt_init (conn); /* Alokacja uchwytu zapytania. */
if (stmt == NULL)
{
    print_error (conn, "Nie można było zainicjalizować uchwytu zapytania.");
    return;
}

/* Wstawienie i pobranie pewnych rekordów. */
insert_rows (stmt);
select_rows (stmt);
mysql_stmt_close (stmt);      /* Usunięcie z pamięci uchwytu zapytania. */
}

```

W pierwszej kolejności wybieramy bazę danych i tworzymy tabelę testową. Wspomniana tabela zawiera cztery kolumny różnego typu danych: INT, FLOAT, CHAR i DATETIME. Poszczególne typy danych muszą być obsługiwane w nieco inny sposób, o czym wkrótce się przekonasz.

Po utworzeniu tabeli wywołujemy funkcję `mysql_stmt_init()` w celu alokacji uchwytu zapytania preinterpretowanego, wstawiamy i pobieramy pewne rekordy, a następnie usuwamy uchwyt z pamięci. Wszystkie rzeczywiste operacje są przeprowadzane w funkcjach `insert_rows()` i `select_rows()`, które wkrótce zostaną omówione. Do obsługi błędów program również używa funkcji — `print_stmt_error()`, podobnej do funkcji `print_error()` używanej we wcześniejszych programach. Jednak omawiany tutaj program wywołuje funkcję `print_stmt_error()` w sposób przystosowany dla zapytań preinterpretowanych:

```

static void print_stmt_error (MYSQL_STMT *stmt, char *message)
{
    fprintf (stderr, "%s\n", message);
    if (stmt != NULL)
    {
        fprintf (stderr, "Błąd %u (%s): %s\n",
                 mysql_stmt_errno (stmt),
                 mysql_stmt_sqlstate (stmt),
                 mysql_stmt_error (stmt));
    }
}

```

Funkcja `insert_rows()` jest odpowiedzialna za wstawienie nowych rekordów do tabeli testowej:

```

static void insert_rows (MYSQL_STMT *stmt)
{
    char          *stmt_str = "INSERT INTO t (i,f,c,dt) VALUES(?,?,?,?)";
    MYSQL_BIND    param[4];
    int           my_int;
    float         my_float;
    char          my_str[26]; /* Funkcja ctime() zwraca ciąg tekstowy o długości 26 znaków. */
    MYSQL_TIME    my_datetime;
}

```



```

unsigned long my_str_length;
time_t        clock;
struct tm     *cur_time;
int           i;

printf ("Wstawianie rekordów...\n");
if (mysql_stmt_prepare (stmt, stmt_str, strlen (stmt_str)) != 0)
{
    print_stmt_error (stmt, "Nie można było przygotować zapytania INSERT.");
    return;
}

/*
 * Wyzerowanie struktur parametrów, a następnie przeprowadzenie inicjalizacji
 * wszystkich parametrów, które są stałe i nie zmieniają każdego rekordu.
 */
memset ((void *) param, 0, sizeof (param));

/* Konfiguracja parametru INT. */
param[0].buffer_type = MYSQL_TYPE_LONG;
param[0].buffer = (void *) &my_int;
param[0].is_unsigned = 0;
param[0].is_null = 0;

/* buffer_length, wielkości nie trzeba definiować. */
/* Konfiguracja parametru FLOAT. */
param[1].buffer_type = MYSQL_TYPE_FLOAT;
param[1].buffer = (void *) &my_float;
param[1].is_null = 0;

/* is_unsigned, buffer_length, wielkości nie trzeba definiować. */
/* Konfiguracja parametru CHAR. */
param[2].buffer_type = MYSQL_TYPE_STRING;
param[2].buffer = (void *) my_str;
param[2].buffer_length = sizeof (my_str);
param[2].is_null = 0;

/* is_unsigned wielkości nie trzeba definiować, wielkość będzie ustawiona później. */
/* Konfiguracja parametru DATETIME. */
param[3].buffer_type = MYSQL_TYPE_DATETIME;
param[3].buffer = (void *) &my_datetime;
param[3].is_null = 0;

/* is_unsigned, buffer_length, wielkości nie trzeba definiować. */
if (mysql_stmt_bind_param (stmt, param) != 0)
{
    print_stmt_error (stmt, "Nie można dołączyć parametrów zapytania INSERT.");
    return;
}

for (i = 1; i <= 5; i++)
{
    printf ("Wstawienie rekordu %d...\n", i);
    (void) time (&clock); /* Pobranie bieżącej godziny. */
    /* Konfiguracja zmiennych używanych dla każdego parametru. */
    /* param[0]: ustawienie wartości my_int. */
    my_int = i;
    /* param[1]: ustawienie wartości my_float. */

```

```

my_float = (float) i;
/* param[2]: przypisanie my_str wartości ciągu tekstowego funkcji current ctime() */
/* oraz ustawienie długości jako wartości zmiennej wskazującej długość my_str. */
(void) strcpy (my_str, ctime (&clock));
my_str[24] = '\0'; /* Usunięcie znaku nowego wiersza na końcu. */
my_str_length = strlen (my_str);
param[2].length = &my_str_length;
/* param[3]: przypisanie my_datetime bieżącej daty i godziny. */
cur_time = localtime (&clock);
my_datetime.year = cur_time->tm_year + 1900;
my_datetime.month = cur_time->tm_mon + 1;
my_datetime.day = cur_time->tm_mday;
my_datetime.hour = cur_time->tm_hour;
my_datetime.minute = cur_time->tm_min;
my_datetime.second = cur_time->tm_sec;
my_datetime.second_part = 0;
my_datetime.neg = 0;
if (mysql_stmt_execute (stmt) != 0)
{
    print_stmt_error (stmt, "Nie można było wykonać zapytania.");
    return;
}
sleep (1); /* Krótka przerwa (aby umożliwić zmianę godziny). */
}
}

```

Ogólnym celem funkcji `insert_rows()` jest wstawienie do tabeli testowej pięciu rekordów zawierających wymienione poniżej wartości:

- Wartość INT z zakresu od 1 do 5.
- Wartość FLOAT z zakresu od 1.0 do 5.0.
- Wartość CHAR. W celu wygenerowania tej wartości konieczne jest wywołanie funkcji systemowej `ctime()`, aby otrzymać wartość aktualnej daty i godziny w postaci ciągu tekstowego. Funkcja `ctime()` zwraca wartości w następującym formacie:
Sun Sep 19 16:47:23 CDT 2004
- Wartość DATETIME. To również jest wartość bieżącej daty i godziny, ale przechowywana w strukturze `MYSQL_TIME`. Protokół binarny używa struktur `MYSQL_TIME` w celu transmisji wartości typu DATETIME, TIMESTAMP, DATE i TIME.

Pierwszym zadaniem w funkcji `insert_rows()` jest przygotowanie zapytania INSERT przez jego przekazanie funkcji `mysql_stmt_prepare()`. Zapytanie przedstawia się następująco:

```
INSERT INTO t (i,f,c,dt) VALUES(?,?,?,?)
```

Powyższe zapytanie ma cztery miejsca zarezerwowane. W chwili jego każdego wykonania konieczne jest więc dostarczenie czterech wartości danych. Miejsca zarezerwowane najczęściej przedstawiają wartości danych na liście `VALUES()` lub w klauzuli `WHERE`. Są jednak miejsca, w których nie mogą być stosowane:

- Jako identyfikatory, na przykład nazwa tabeli lub kolumny. Poniższe zapytanie jest nieprawidłowe:

```
SELECT * FROM ?
```

- Miejsce zarezerwowane można umieścić tylko po jednej stronie operatora, ale nie po obu. Poniższe zapytanie jest prawidłowe:

```
SELECT * FROM student WHERE student_id = ?
```

Z kolei poniższe zapytanie jest nieprawidłowe:

```
SELECT * FROM student WHERE ? = ?
```

Wymienione ograniczenia są konieczne, aby serwer mógł określić typ danych parametru.

Kolejnym krokiem jest konfiguracja struktur `MYSQL_BIND`, po jednej dla każdego miejsca zarezerwowanego. Jak pokazano w funkcji `insert_rows()`, konfiguracja wymienionych struktur przebiega w dwóch etapach:

1. Inicjalizacja wszystkich części struktur, które będą takie same dla wszystkich wstawianych rekordów.
2. Wykonanie pętli wstawiania rekordu, która dla każdego rekordu zainicjalizuje części struktur różniące się w każdym rekordzie.

Tak naprawdę całą inicjalizację można przeprowadzić w pętli, ale to będzie mniej efektywne rozwiązanie.

Pierwszy etap inicjalizacji rozpoczyna się od wyzerowania zawartości tablicy `param` zawierającej strukturę `MYSQL_BIND`. Program używa funkcji `memset()`, ale równie dobrze można wykorzystać funkcję `bzero()`, jeśli `memset()` jest niedostępna w Twoim systemie. Oba przedstawione poniżej polecenia działają identycznie:

```
memset((void *) param, 0, sizeof (param));  
bzero((void *) param, sizeof (param));
```

Wyczyszczenie tablicy `param` powoduje wyraźne przypisanie wartości zero wszystkim elementom struktury. Kod, który zostanie wkrótce przedstawiony, przypisuje wartość zero tylko niektórym elementom, aby wyraźniej pokazać, co się dzieje. To jednak nie jest konieczne. W praktyce po wyczyszczeniu struktury żadnemu jej elementowi nie musisz przypisywać wartości zero.

Kolejnym krokiem jest przypisanie prawidłowych informacji każdemu parametrowi w tablicy `MYSQL_BIND`. Dla każdego parametru elementy struktury wymagające konfiguracji zależą od typu przekazywanej wartości:

- Element `buffer_type` zawsze musi być skonfigurowany, wskazuje typ danych wartości.
- Element `buffer` powinien mieć przypisany adres zmiennej zawierającej dane. Funkcja `insert_rows()` deklaruje cztery zmienne przechowujące wartości rekordu: `my_int`, `my_float`, `my_str` i `my_datetime`. Każda wartość `param[i].buffer` została ustawiona w taki sposób, że wskazuje odpowiednią zmienną. Kiedy

trzeba wstawić rekord, wymienionym zmiennym zostają przypisane wartości kolumn tabeli, a następnie są one używane do utworzenia nowego rekordu.

- Element `is_unsigned` ma jedynie zastosowanie do typów danych w postaci liczb całkowitych. Powinien mieć wartość `true` (niezerowa) lub `false` (zero) w celu wskazania, czy parametr odpowiada typowi `UNSIGNED`. W tabeli znajduje się kolumna wartości `INT` ze znakiem, a więc zmiennej `is_unsigned` zostaje przypisana wartość zero. Gdyby kolumna była zdefiniowana jako `INT UNSIGNED`, wtedy zmiennej `is_unsigned` trzeba przypisać wartość 1. Ponadto, zmienna `my_int` musiałaby zostać zadeklarowana jako `unsigned int` zamiast `int`.
- Element `is_null` wskazuje, czy przekazywana będzie wartość `NULL`. Ogólnie rzecz biorąc, temu elementowi przypisywany jest adres zmiennej `my_bool`. Następnie przed wstawieniem rekordu należy wymienionej zmiennej przypisać wartość `true` lub `false`, określającą, czy wstawianą wartością będzie `NULL`. Jeżeli żadne wartości `NULL` nie będą wstawiane (jak w omawianym przypadku), wtedy zmiennej `is_null` można przypisać zero, a zmienna `my_bool` nie będzie potrzebna.
- Dla wartości w postaci ciągu tekstowego lub danych binarnych (wartości `BLOB`) używane są dwa dodatkowe elementy struktury `MYSQL_BIND`. Określają one wielkość bufora przechowującego wartość oraz rzeczywistą wielkość przekazywanej wartości. W wielu przypadkach obie wielkości są takie same, ale będą różne, jeśli używany jest bufor o stałej wielkości, a wysyłane wartości mają różną wielkość w poszczególnych rekordach. Element `buffer_length` określa wielkość bufora, natomiast `length` jest wskaźnikiem. Powinien mieć przypisany adres zmiennej typu `unsigned long` zawierającej rzeczywistą wielkość wartości wysyłanej do bazy danych.
W przypadku danych liczbowych i wskazujących datę oraz godzinę elementy `buffer_length` i `length` nie muszą być ustawiane. Wielkość wymienionych typów jest stała i może być ustalona na podstawie wartości `buffer_type`. Na przykład, `MYSQL_TYPE_LONG` i `MYSQL_TYPE_FLOAT` wskazują na wartości odpowiednio 4- i 8-bajtowe.

Po przeprowadzeniu początkowej konfiguracji tablicy `MYSQL_BIND` należy ją dołączyć do zapytania preinterpretowanego przez przekazanie tablicy funkcji `mysql_stmt_bind_param()`. Następnie trzeba przypisać wartości zmiennym wskazywanym przez struktury `MYSQL_BIND` i wykonać zapytanie. Odbyna się to w pętli o pięciu iteracjach. Każda iteracja pętli powoduje przypisanie wartości parametrom zapytania:

- Dla parametrów w postaci liczb całkowitych i zmiennoprzecinkowych konieczne jest przypisanie wartości jedynie powiązanym z nimi zmiennym `int` i `float`.
- Dla parametru w postaci ciągu tekstowego buforowi typu `char` przypisywana jest aktualna godzina w formacie ciągu tekstowego. Wspomniana wartość jest otrzymywana za pomocą wywołania `ctime()`, a następnie usuwany jest z niej znak nowego wiersza.

- Parametrowi `datetime` również przypisywana jest aktualna godzina, ale odbywa się to przez przypisanie komponentów godziny poszczególnym elementom struktury `MYSQL_TIME`.

Po ustawieniu wartości parametrów następuje wykonanie zapytania za pomocą wywołania `mysql_stmt_execute()`. Wymieniona funkcja powoduje przekazanie wartości bieżących do serwera, który następnie wstawia je do zapytania preinterpretowanego i wykonuje.

Po zakończeniu działania funkcji `insert_rows()` tabela testowa jest już wypełniona i można użyć funkcji `select_rows()` w celu pobrania rekordów:

```
static void select_rows (MYSQL_STMT *stmt)
{
    char          *stmt_str = "SELECT i, f, c, dt FROM t";
    MYSQL_BIND    param[4];
    int           my_int;
    float         my_float;
    char          my_str[24];
    unsigned long my_str_length;
    MYSQL_TIME    my_datetime;
    my_bool       is_null[4];

    printf ("Pobieranie rekordów...\n");
    if (mysql_stmt_prepare (stmt, stmt_str, strlen (stmt_str)) != 0)
    {
        print_stmt_error (stmt, "Nie można było przygotować zapytania SELECT.");
        return;
    }

    if (mysql_stmt_field_count (stmt) != 4)
    {
        print_stmt_error (stmt, "Nieoczekiwana liczba kolumn w zapytaniu SELECT.");
        return;
    }

    /*
     * Inicjalizacja struktur kolumn wyniku.
     */
    memset ((void *) param, 0, sizeof (param)); /* Wyzeroowanie struktur. */
    /* Konfiguracja parametru INT. */
    param[0].buffer_type = MYSQL_TYPE_LONG;
    param[0].buffer = (void *) &my_int;
    param[0].is_unsigned = 0;
    param[0].is_null = &is_null[0];
    /* buffer_length, wielkości nie trzeba definiować. */
    /* Konfiguracja parametru FLOAT. */
    param[1].buffer_type = MYSQL_TYPE_FLOAT;
    param[1].buffer = (void *) &my_float;
    param[1].is_null = &is_null[1];
    /* is_unsigned, buffer_length, wielkości nie trzeba definiować. */
    /* Konfiguracja parametru CHAR. */
    param[2].buffer_type = MYSQL_TYPE_STRING;
    param[2].buffer = (void *) my_str;
    param[2].buffer_length = sizeof (my_str);
    param[2].length = &my_str_length;
```

```

param[2].is_null = &is_null[2];
/* is_unsigned wielkości nie trzeba definiować. */
/* Konfiguracja parametru DATETIME. */
param[3].buffer_type = MYSQL_TYPE_DATETIME;
param[3].buffer = (void *) &my_datetime;
param[3].is_null = &is_null[3];
/* is_unsigned, buffer_length, wielkości nie trzeba definiować. */
if (mysql_stmt_bind_result (stmt, param) != 0)
{
    print_stmt_error (stmt, "Nie można dołączyć parametrów zapytania SELECT.");
    return;
}

if (mysql_stmt_execute (stmt) != 0)
{
    print_stmt_error (stmt, "Nie można było wykonać zapytania SELECT.");
    return;
}

/*
 * Pobranie wyniku i umieszczenie w pamięci klienta; wprawdzie to jest opcjonalne,
 * ale pozwala na wywołanie funkcji mysql_stmt_num_rows() w celu ustalenia liczby
 * rekordów znajdujących się w zbiorze wynikowym.
 */
if (mysql_stmt_store_result (stmt) != 0)
{
    print_stmt_error (stmt, "Nie można było buforować zbioru wynikowego.");
    return;
}
else
{
    /* Wywołanie funkcji mysql_stmt_store_result() pozwala na zliczenie rekordów. */
    printf ("Liczba pobranych rekordów: %lu\n",
            (unsigned long) mysql_stmt_num_rows (stmt));
}

while (mysql_stmt_fetch (stmt) == 0) /* Pobranie każdego rekordu. */
{
    /* Wyświetlenie wartości rekordu. */
    printf ("%d ", my_int);
    printf ("%2f ", my_float);
    printf ("%*.s ", (int) my_str_length, (int) my_str_length, my_str);
    printf ("%04d-%02d-%02d %02d:%02d:%02d\n",
            my_datetime.year,
            my_datetime.month,
            my_datetime.day,
            my_datetime.hour,
            my_datetime.minute,
            my_datetime.second);
}
mysql_stmt_free_result (stmt); /* Usunięcie zbioru wynikowego z pamięci. */
}

```

Funkcja `select_rows()` przygotowuje zapytanie, wykonuje je i pobiera wynik. W takim przypadku zapytanie nie zawiera miejsc zarezerwowanych:

```
SELECT i, f, c, dt FROM t
```

Oznacza to, że nie ma konieczności konfiguracji jakichkolwiek struktur `MYSQL_BIND` przed wykonaniem zapytania. Niestety, nie mamy jeszcze świętego spokoju. Większość pracy wykonywanej w funkcjach `select_rows()` i `insert_rows()` wiąże się z konfiguracją tablicy struktur `MYSQL_BIND`. Różnica polega na tym, że są one używane do pobierania danych z serwera *po* wykonaniu zapytania, a nie do konfiguracji danych wysyłanych do serwera *przed* wykonaniem zapytania.

Procedura konfiguracji tablicy `MYSQL_BIND` jest podobna do odpowiadającego jej kodu w funkcji `insert_rows()`:

1. Wyzerowanie tablicy.
2. Ustawienie odpowiedniego typu kodu elementowi `buffer_type` każdego parametru.
3. Przypisanie elementowi `buffer` każdego parametru zmiennej przeznaczonej do przechowywania wartości po pobraniu rekordów.
4. Przypisanie wartości zero elementowi `is_unsigned` dla parametru w postaci liczby całkowitej.
5. Dla parametru ciągu tekstowego przypisanie elementowi `buffer_length` liczby wskazującej maksymalną liczbę bajtów, które powinny być pobrane, natomiast elementowi `length` adresu zmiennej typu `unsigned long`. W czasie pobierania rekordów wymieniona zmienna będzie miała przypisaną rzeczywistą liczbę pobranych bajtów.
6. Dla każdego parametru elementowi `is_null` powinien być przypisany adres zmiennej `my_bool`. W trakcie pobierania rekordów wspomniana zmienna będzie wskazywała, czy pobraną wartością jest `NULL`. (Nasz program ignoruje te zmienne po pobraniu rekordów, ponieważ wiemy, że tabela testowa nie zawiera wartości `NULL`. W innych przypadkach powinienś sprawdzać wartość zmiennej `my_bool`).

Po skonfigurowaniu parametrów tablicę dołączamy do zapytania przez wywołanie funkcji `mysql_stmt_bind_result()`, a następnie wykonujemy zapytanie.

Na tym etapie można od razu przystąpić do pobierania rekordów za pomocą funkcji `mysql_stmt_fetch()`. W programie zademonstrowano krok opcjonalny, który można wykonać na początku. To wywołanie funkcji `mysql_stmt_store_result()`, która pobiera cały zbiór wynikowy i buforuje go w pamięci klienta. Zaletą takiego rozwiązania jest możliwość wywołania `mysql_stmt_num_rows()` w celu ustalenia liczby rekordów znajdujących się w zbiorze wynikowym. Z kolei wadą jest większe zużycie pamięci po stronie klienta.

Pętla odpowiedzialna za pobieranie rekordów wywołuje funkcję `mysql_stmt_fetch()`, dopóki jej wartość zwrótta jest niezerowa. Po pobraniu każdego rekordu zmienne powiązane z parametrami struktur będą zawierały wartości kolumn dla bieżącego rekordu.

Po pobraniu wszystkich rekordów wywołanie funkcji `mysql_stmt_free_result()` powoduje zwolnienie całej pamięci zaalokowanej dla zbioru wynikowego.

W tym miejscu działanie funkcji `select_rows()` kończy się i następuje wywołanie funkcji `mysql_stmt_close()` w celu usunięcia uchwytu zapytania preinterpretowanego.

Powyższa analiza przedstawiła ogólny opis interfejsu zapytań preinterpretowanych oraz wybrane z ich funkcji kluczowych. Biblioteka klienta zawiera wiele innych funkcji.

7.9. Użycie preinterpretowanego zapytania CALL

Obsługa zapytań preinterpretowanych została znacznie usprawniona w MySQL 5.5, gdzie poprawiono obsługę preinterpretowanych zapytań CALL w celu wywoływania procedur składowanych, dodając między innymi możliwość uzyskania dostępu do wartości zwrótnych parametrów OUT i INOUT procedury. Wcześniej preinterpretowane zapytania CALL nie mogły wygenerować wielu zbiorów wynikowych, a wywołujący nie miał dostępu do zwróconych wartości parametru.

Na potrzeby przykładu przyjmujemy założenie istnienia poniższej procedury składowanej:

```
CREATE PROCEDURE grade_event_stats
  (IN p_event_id INT, OUT p_min INT, OUT p_max INT)
BEGIN
  -- Wyświetlenie wyników dla danego zdarzenia.
  SELECT student_id, score
    FROM score
   WHERE event_id = p_event_id
  ORDER BY student_id;
  -- Wartości minimalnego i maksymalnego wyniku są przechowywane w parametrach OUT.
  SELECT MIN(score), MAX(score)
    FROM score
   WHERE event_id = p_event_id
  INTO p_min, p_max;
END;
```

Powyższa procedura pobiera jeden parametr IN i dwa OUT. Mając podany identyfikator zdarzenia jako parametr, procedura wyświetla wyniki dla danego zdarzenia, a także wynik minimalny i maksymalny danego zdarzenia, przechowywane w dwóch parametrach OUT. Jeżeli z poziomu klienta `mysql` wywołamy funkcję `grade_event_stats()`, zapytanie może mieć następującą postać:

```
mysql> SET @p_min = NULL, @p_max = NULL;
mysql> CALL grade_event_stats(4, @p_min, @p_max);
+-----+-----+
| student_id | score |
+-----+-----+
|          2 |     7 |
|          3 |    17 |
|          4 |    16 |
|          5 |    20 |
...
mysql> SELECT @p_min, @p_max;
```



```
+-----+-----+
| @p_min | @p_max |
+-----+-----+
|      7 |      20 |
+-----+-----+
```

Poniższa analiza prezentuje sposób użycia API C do osiągnięcia tego samego celu. Kod źródłowy programu znajduje się w pliku *prepared_call.c* w katalogu *capi* dystrybucji *sampdb*. W celu utworzenia procedury należy użyć pliku *prepared_call_setup.sql*, który znajduje się w tym samym katalogu:

```
% mysql sampdb < prepared_call_setup.sql
```

Obsługa preinterpretowanego zapytania CALL wymaga serwera MySQL w wersji 5.5.3 lub nowszej. Aby sprawdzić, czy używany serwer oferuje omawianą możliwość, program *prepared_call* sprawdza wersję serwera:

```
if (mysql_get_server_version (conn) < 50503)
{
    print_error (NULL, "Obsługa preinterpretowanego zapytania CALL wymaga serwera MySQL
5.5.3 lub nowszego.");
    mysql_close (conn);
    exit (1);
}
```

Kolejny fragment programu inicjalizuje uchwyt zapytania preinterpretowanego i używa go do wykonania preinterpretowanego zapytania CALL. Jeżeli operacja zakończy się powodzeniem, wtedy nastąpi przetworzenie wyniku działania procedury:

```
stmt = mysql_stmt_init (conn);
if (!stmt)
    print_error (NULL, "Nie można było zainicjalizować uchwytu zapytania.");
else
{
    if (exec_prepared_call (stmt) == 0)
        process_call_result (conn, stmt);
    mysql_stmt_close (stmt);
}
```

Obsługa preinterpretowanego zapytania CALL jest podobna do obsługi innych zapytań preinterpretowanych. Ciąg tekstowy zapytania zostaje przekazany funkcji *mysql_stmt_prepare()* wraz z wartościami danych przedstawianymi w postaci znaków ? jako miejsc zarezerwowanych. W przypadku zapytania CALL miejsce zarezerwowane przedstawia wartość parametru przekazywaną procedurze. Konfiguracja parametrów musi być dokładna w zakresie liczby i typów parametrów przyjmowanych przez procedurę. W celu zachowania prostoty wszystkie parametry funkcji *grade_event_stats()* są liczbami całkowitymi.

```
static int exec_prepared_call (MYSQL_STMT *stmt)
{
    MYSQL_BIND params[3]; /* Bufory parametru. */
    int          int_data[3]; /* Wartości parametru. */
    int          i;
```

```

/* Przygotowanie zapytania CALL. */
if (mysql_stmt_prepare (stmt, "CALL grade_event_stats(?, ?, ?)", 31))
{
    print_stmt_error (stmt, "Nie można przygotować zapytania.");
    return (1);
}

/* Inicjalizacja struktur parametru i dołączenie ich do zapytania. */
memset (params, 0, sizeof (params));
for (i = 0; i < 3; ++i)
{
    params[i].buffer_type = MYSQL_TYPE_LONG;
    params[i].buffer = (char *) &int_data[i];
    params[i].length = 0;
    params[i].is_null = 0;
}

if (mysql_stmt_bind_param (stmt, params))
{
    print_stmt_error (stmt, "Nie można dołączyć parametrów.");
    return (1);
}

/* Przypisanie wartości parametrów i wykonanie zapytania. */
int_data[0] = 4; /* p_event_id */
int_data[1] = 0; /* p_min (Parametr OUT; wartość początkowa ignorowana przez procedurę). */
int_data[2] = 0; /* p_min (Parametr OUT; wartość początkowa ignorowana przez procedurę). */
if (mysql_stmt_execute (stmt))
{
    print_stmt_error (stmt, "Nie można wykonać zapytania.");
    return (1);
}
return (0);
}

```

Po wykonaniu zapytania CALL należy przetworzyć jego wyniki, które mogą składać się z kilku części:

1. Każde zapytanie wykonywane w procedurze może wygenerować zbiór wynikowy. Dotyczy to zapytań takich jak SELECT, SHOW itd.
2. Jeżeli procedura ma jakiegokolwiek parametry OUT lub INOUT, wtedy istnieje dodatkowych zbiór wynikowy składający się z pojedynczego rekordu, który zawiera ostateczne wartości parametru, w kolejności ich pojawiania się w definicji procedury.
3. Ostateczny pakiet stanu. Nie ma przypisanego zbioru wynikowego, więc można go odróżnić od zbioru wynikowego zapytania lub parametru, ponieważ nie posiada żadnej kolumny.

Wszystkie istniejące części wyniku zapytania CALL muszą być przetworzone. Zbiory wynikowe zapytania i parametru są opcjonalne, w zależności od sposobu utworzenia procedury. Procedura zawsze zwraca ostateczny pakiet stanu, niezależnie od istnienia jakichkolwiek zbiorów wynikowych.

Funkcja `process_call_result()` pokazuje ogólnego przeznaczenia pętlę pobierającą wyniki procedury. W kodzie nie poczyniono żadnych założeń dotyczących tego, czy procedura wygeneruje jakikolwiek zbiór wynikowy zapytania lub parametru. Pętla pobierająca wyniki ma następującą logikę:

1. Pobierz liczbę kolumn kolejnego wyniku, aby w ten sposób ustalić, czy do przetworzenia jest kolejny zbiór wynikowy, czy pozostał tylko ostateczny pakiet stanu. Wartość zero wskazuje na pakiet stanu, który nie wymaga przetworzenia.
2. Dodatnia liczba kolumn wskazuje na istnienie zbioru wynikowego zawierającego wskazaną liczbę kolumn. Wspomniany zbiór wynikowy mógł zostać wygenerowany przez zapytanie lub zawiera zwrócone przez procedurę wartości parametru. Uchwył połączenia ma element stanu wraz z flagami, co pozwala na jego użycie do odróżnienia wymienionych przypadków. (Funkcja `process_call_result()` używa flagi jedynie w celach informacyjnych, do wskazania sposobu wygenerowania poszczególnych zbiorów wynikowych). W każdym przypadku należy pobrać zbiór wynikowy.
3. Trzeba wywołać funkcję `mysql_stmt_next_result()` i sprawdzić, czy istnieje więcej wyników. Wartością zwrótną wymienionej funkcji będzie `-1` w przypadku istnienia kolejnych wyników lub wartość większa niż zero w przypadku wystąpienia błędu. Dlatego też, jeśli wartością zwrótną jest zero, wtedy należy wrócić do kroku 1. i pobrać kolejny wynik.

```
static void process_call_result (MYSQL *conn, MYSQL_STMT *stmt)
{
    int status;
    int num_cols;

    /*
     * Trzeba sprawdzić liczbę kolumn w każdym wyniku. Jeżeli wynik nie zawiera żadnej,
     * to jest ostatecznym pakietem stanu i oznacza brak kolejnych danych do przetworzenia.
     * Wartość niezerowa oznacza konieczność pobrania zbioru wynikowego.
     */
    do {
        if ((num_cols = mysql_stmt_field_count (stmt)) > 0)
        {
            /* Poinformowanie, czy zbiór wynikowy zawiera parametry lub dane. */
            if (conn->server_status & SERVER_PS_OUT_PARAMS)
                printf ("Wartości parametrów OUT/INOUT:\n");
            else
                printf ("Wartości zbioru wynikowego zapytania:\n");
            if (process_result_set (stmt, num_cols))
                break; /* Wystąpił błąd. */
        }
        /* Wartość stanu: -1 = koniec pracy, 0 = więcej wyników, >0 = błąd. */
        status = mysql_stmt_next_result (stmt);
        if (status > 0)
            print_stmt_error (stmt, "Błąd w trakcie sprawdzania istnienia kolejnego
zbioru
        wyników.");
    } while (status == 0);
}
```

W celu pobrania zbioru wynikowego zapytania lub parametru pętla wywołuje funkcję `process_result_set()`, której kod nie został tutaj przedstawiony. Znajdziesz go w pliku o nazwie *prepared_call.c*. Aby nie komplikować przykładu, przyjęto założenie, że wszystkie otrzymane wyniki są liczbami całkowitymi. Możesz to zmienić w przypadku programów obsługujących inne typy danych.

Poniżej przedstawiono dane wyjściowe wygenerowane przez uruchomiony program `prepared_call`:

```
% ./prepared_call sampdb
Wartości zbioru wynikowego zapytania:
val[1] = 2; val[2] = 7;
val[1] = 3; val[2] = 17;
val[1] = 4; val[2] = 16;
val[1] = 5; val[2] = 20;
...
Wartości parametrów OUT/INOUT:
val[1] = 7; val[2] = 20;
```

Zachowaj ostrożność i pamiętaj o wszelkich różnicach między położeniami parametrów w definicji procedury oraz ich położeniem w zbiorze wynikowym parametru. Zwróć uwagę, że według powyższych danych wyjściowych zwrócone wartości parametrów mają położenia 1 i 2, choć tak naprawdę to parametry 2 i 3 procedury `grade_event_stats()`. Dlaczego tak się dzieje? Ponieważ ostateczny zbiór wynikowy zwracający wartości parametrów zawiera jedynie parametry OUT i INOUT, a nie parametry IN. Wszelkie zmiany w parametrach IN w procedurze są niewidoczne dla wywołującego. Dlatego też mają takie same wartości po wywołaniu jak przed wywołaniem i nie muszą być zwracane w wynikach procedury.

Tworzenie programów MySQL przy użyciu Perl DBI

W tym rozdziale dowiesz się, jak używać interfejsu Perl DBI wraz z bazą danych MySQL. Nie zostanie omówiona filozofia lub architektura Perl DBI. Więcej informacji na temat wspomnianych aspektów DBI (szczególnie w porównaniu do API C i PHP) znajdziesz w rozdziale 6., zatytułowanym „Wprowadzenie do programowania MySQL”.

Przedstawione tutaj przykłady opierają się na przykładowej bazie danych *sampdb* i tabelach dla projektów ocen uczniów i Ligi Historycznej utworzonych w rozdziale 1., zatytułowanym „Rozpoczęcie pracy z MySQL”.

Przyjęto założenie użycia DBI w wersji minimum 1.50, choć większość przedstawionego tutaj materiału ma zastosowanie także względem wcześniejszych wersji. DBI 1.50 wymaga języka Perl 5.6.0 (preferowana wersja to 5.6.1). Z kolei dla DBI 1.611 minimalną wersją Perl jest 5.8.1. Konieczne jest posiadanie zainstalowanego modułu Perl DBD::mysql, a także utworzonej w języku C biblioteki klienta MySQL i plików nagłówkowych. Jeżeli planujesz tworzenie sieciowych skryptów DBI w przedstawiony tutaj sposób, wymagany jest również moduł CGI.pm. W tym rozdziale moduł CGI.pm jest używany w połączeniu z serwerem WWW Apache. Jeśli musisz pobrać którykolwiek z wymienionych pakietów, zajrzyj do dodatku A, zatytułowanego „Oprogramowanie wymagane do użycia tej książki”. W wymienionym dodatku znajdziesz informacje dotyczące pobrania dystrybucji *sampdb*, zawierającej skrypty opracowane w tym rozdziale. Skrypty znajdziesz w katalogu *perlapi* dystrybucji.

W większości przypadków w rozdziale omówiono metody i zmienne Perl DBI tylko w zakresie, w jakim są one potrzebne w prezentowanych programach. Dokumentacja jest dostępna także pod adresem <http://dbi.perl.org/> oraz po wydaniu poniższych poleceń:

```
% perl doc DBI
% perl doc DBI::FAQ
% perl doc DBD::mysql
```

Na poziomie sterownika bazy danych (DBD) sterownik dla MySQL został zbudowany na bazie utworzonej w języku C biblioteki klienta MySQL, a tym samym współdzieli z nią

pewne cechy charakterystyczne. Więcej informacji na temat wspomnianej biblioteki znajdziesz w rozdziale 7., zatytułowanym „Tworzenie programów MySQL przy użyciu języka C”.

8.1. Cechy charakterystyczne skryptu Perl

Skrypty Perl to zwykle pliki tekstowe, a więc można je tworzyć za pomocą dowolnego edytora tekstów. Wszystkie skrypty Perl przedstawione w tym rozdziale stosują konwencję systemu UNIX i rozpoczynają się od wiersza shebang (`#!`), wskazującego ścieżkę dostępu do programu używanego w celu wykonania skryptu. W przykładach używamy następującego wiersza:

```
#!/usr/bin/perl
```

W systemie UNIX będziesz musiał zmodyfikować ten wiersz, jeśli ścieżka dostępu do Perla jest inna, na przykład `/opt/bin/perl`. W przeciwnym razie skrypty Perla nie będą prawidłowo uruchamiane w systemie.

Skrypt Perl o nazwie *myscript.pl* można w dowolnym systemie uruchomić w poniższy sposób:

```
% perl myscript.pl
```

Istnieje również możliwość wykonania skryptu bez konieczności wcześniejszego podawania nazwy programu perl używanego do jego uruchomienia. W systemie UNIX odbywa się to przez zmianę trybu pliku i nadanie skryptowi uprawnień wykonywania za pomocą narzędzia `chmod`:

```
% chmod +x myscript.pl
```

Następnie w celu uruchomienia skryptu wystarczy podać jego nazwę:

```
% ./myscript.pl
```

Ten styl wywoływania skryptów jest stosowany w tym rozdziale. Znaki `./` na początku nazwy trzeba dodać, jeśli skrypt znajduje się w katalogu bieżącym (`.`), który nie został wymieniony w ścieżce wyszukiwania powłoki. W przeciwnym razie można pominąć `./` w nazwie polecenia:

```
% myscript.pl
```

W systemie Windows można zdefiniować powiązanie między Perlem i nazwami plików z rozszerzeniem *.pl*. Na przykład, jeśli zainstalujesz ActiveState Perl, to jego program instalacyjny pozwoli na ustawienie powiązania plików z rozszerzeniem *.pl* z interpreterem Perl. W takim przypadku wykonanie skryptu Perl wymaga jedynie podania jego nazwy w wierszu poleceń:

```
C:\> myscript.pl
```

8.2. Ogólny opis Perl DBI

W tym podrozdziale zostaną przedstawione ogólne informacje o DBI, które powinieneś znać, tworząc własne skrypty i chcąc zrozumieć skrypty przygotowane przez innych. Jeżeli już znasz DBI, możesz pominąć ten podrozdział i przejść bezpośrednio do podrozdziału 8.3, zatytułowanego „Praca z DBI”.

8.2.1. Typy danych DBI

Pod pewnymi względami API Perl DBI jest podobne do omówionej w rozdziale 7. biblioteki klienta utworzonej w języku C. Aby móc używać wspomnianej biblioteki klienta, wywołujesz funkcje i uzyskujesz dostęp do danych powiązanych z MySQL za pomocą wskaźników do struktur lub tablic. W celu użycia API DBI także wywołujesz funkcje i korzystasz ze wskaźników do struktur, ale wywoływane funkcje są nazywane „metodami”, wskaźniki „odniesieniami”, wskaźniki zmiennych „uchwytyami”, natomiast struktury, do których prowadzą uchwyty, są nazywane „obiektami”.

DBI używa wielu rodzajów uchwytów. W dokumentacji DBI odniesienia do nich odbywają się za pomocą konwencji przedstawionej w tabeli 8.1. Sposób użycia wymienionych uchwytów będzie przedstawiony w rozdziale. Ponadto, stosowanych jest wiele konwencjonalnych nazw dla zmiennych niebędących uchwytami (patrz tabela 8.2). Tak naprawdę, w rozdziale nie użyjemy wszystkich z wymienionych nazw zmiennych, ale warto o nich wiedzieć w trakcie analizy skryptów DBI przygotowanych przez inne osoby.

Tabela 8.1. Konwencje nazw zmiennych będących uchwytami w Perl DBI

Nazwa	Opis
<code>\$dbh</code>	Uchwyt do obiektu bazy danych.
<code>\$sth</code>	Uchwyt do obiektu zapytania.
<code>\$fh</code>	Uchwyt do otwartego pliku.
<code>\$h</code>	Uchwyt „ogólnego przeznaczenia”; jego znaczenie zależy od kontekstu.

Tabela 8.2. Konwencje nazw zmiennych niebędących uchwytami w Perl DBI

Nazwa	Opis
<code>\$rc</code>	Kod zwrotny operacji, która zwraca wartość <code>true</code> lub <code>false</code> .
<code>\$rv</code>	Kod zwrotny operacji, która zwraca wartość w postaci liczby całkowitej.
<code>\$rows</code>	Kod zwrotny operacji, która zwraca wartość określającą liczbę rekordów.
<code>@ary</code>	Tablica (lista) przedstawiająca rekord wartości zwrócony przez zapytanie.

8.2.2. Prosty skrypt DBI

Rozpocniemy od prostego skryptu o nazwie *dump_members.pl*, który ilustruje kilka standardowych koncepcji w programowaniu DBI, takich jak nawiązywanie i zamykanie połączenia z serwerem MySQL, wykonywanie zapytań SQL oraz pobieranie danych. Poniższy skrypt generuje dane wyjściowe składające się z rozdzielonej tabulatorami listy członków Ligi Historycznej. Format sam w sobie nie jest interesujący. Na tym etapie znacznie ważniejsze jest przekonanie się, jak użyć DBI do wygenerowania eleganckich danych wyjściowych. Kod skryptu *dump_members.pl* przedstawia się następująco:

```
#!/usr/bin/perl
# dump_members.pl - wygenerowanie listy członków Ligi Historycznej.

use strict;
use warnings;
use DBI;

# Nazwa źródła danych, nazwa użytkownika, hasło i atrybuty połączenia.
my $dsn = "DBI:mysql:sampdb:localhost";
my $user_name = "sampadm";
my $password = "secret";
my %conn_attrs = (RaiseError => 1, PrintError => 0, AutoCommit => 1);

# Nawiązanie połączenia z bazą danych.
my $dbh = DBI->connect ($dsn, $user_name, $password, \%conn_attrs);

# Wykonanie zapytania.
my $sth = $dbh->prepare ("SELECT last_name, first_name, suffix, email,"
    . " street, city, state, zip, phone FROM member ORDER BY last_name");
$sth->execute ();

# Odczyt i wyświetlenie wyników zapytania.
while (my @ary = $sth->fetchrow_array ())
{
    print join ("\t", @ary), "\n";
}
$sth->finish ();

$dbh->disconnect ();
```

Aby samodzielnie wypróbować ten skrypt, użyj kopii umieszczonej w dystrybucji *sampdb* lub utwórz go samodzielnie za pomocą edytora tekstów. Jeżeli korzystasz z procesora tekstów, upewnij się, że zapisałeś skrypt w postaci zwykłego tekstu. Nie zapisuj go w rodzimym formacie procesora tekstów. Prawdopodobnie będziesz musiał zmienić przynajmniej niektóre parametry połączenia, na przykład nazwę komputera, nazwę użytkownika i hasło. (To dotyczy także pozostałych skryptów w tym rozdziale, w których podano parametry połączenia). W punkcie 8.2.9, zatytułowanym „Określenie parametrów połączenia”, przekonasz się, jak pobierać parametry z pliku opcji, zamiast umieszczać je bezpośrednio w skrypcie.

Przeanalizujemy teraz skrypt fragment po fragmencie. Pierwszy wiersz zawiera standardowy identyfikator wskazujący lokalizację interpretera Perl:

```
#!/usr/bin/perl
```


Powyższy wiersz znajduje się w każdym skrypcie omówionym w rozdziale. Nie będę o tym więcej przypominał.

Dobrym rozwiązaniem jest umieszczenie w skrypcie przynajmniej minimalnego opisu wskazującego jego przeznaczenie. Dlatego też kolejny wiersz jest komentarzem informującym czytającego o przeznaczeniu skryptu:

dump_members.pl - wygenerowanie listy członków Ligi Historycznej.

Tekst znajdujący się po znaku # aż do końca wiersza jest uznawany za komentarz. Dobrze jest wyrobić w sobie nawyk umieszczania komentarzy w skryptach i tym samym objaśniania sposobu ich działania.

Następnie mamy kilka poleceń use:

```
use strict;  
use warnings;  
use DBI;
```

Polecenie `use strict` informuje Perl, że wymagane jest zadeklarowanie zmiennych przed ich użyciem. Wprawdzie można tworzyć skrypty bez wymienionego wiersza, ale jego zastosowanie pomaga w wychwytywaniu pomyłek. Dlatego też zalecam, aby zawsze używać polecenia `use strict`. Na przykład, jeśli zadeklarujesz zmienną `$my_var`, a następnie przez pomyłkę będziesz się do niej odwoływał `$mv_var`, wtedy po uruchomieniu skryptu w trybie ścisłym zostanie wyświetlony poniższy komunikat:

Global symbol "\$mv_var" requires explicit package name at line

Kiedy zobaczysz powyższy komunikat, zastanowisz się: „Przecież nigdy nie używałem żadnej zmiennej o nazwie `$mv_var`”. Następnie przyjrzyj się *n* wierszom skryptu, zobaczysz zmienną `$my_var` błędnie zapisaną jako `$mv_var` i poprawisz to. Bez trybu ścisłego Perl nie wyświetli komunikatu o zmiennej `$mv_var`, ale po prostu ją utworzy wraz z wartością `undef` (niezdefiniowana) i będzie używać bez żadnego narzekania. Natomiast Ty będziesz się zastanawiał, dlaczego skrypt nie działa.

Polecenie `use warnings` nakazuje Perlowi wyświetlanie ostrzeżeń po znalezieniu budzących wątpliwości konstrukcji języka lub wykonaniu operacji takich jak wyświetlenie niezainicjalizowanych zmiennych. To użyteczna możliwość, ponieważ informuje programistę, że powinien ostrożniej tworzyć kod.

Polecenie `use DBI` informuje Perl o konieczności użycia modułu DBI. Bez tego wiersza próba wykonania w skrypcie dowolnej operacji powiązanej z DBI spowoduje wygenerowanie błędu. Nie ma konieczności wskazywania konkretnego sterownika na poziomie modułu DBI, ponieważ DBI aktywuje odpowiedni po nawiązaniu połączenia z serwerem bazy danych.

Ponieważ pracujemy w trybie ścisłym, konieczne jest zadeklarowanie zmiennych używanych w skrypcie, co odbywa się za pomocą słowa kluczowego `my`. W ten sposób wyraźnie wskazujesz skryptowi, że to są Twoje zmienne. W kolejnym fragmencie skryptu następuje przypisanie wspomnianym zmiennym parametrów połączenia, a następnie użycie zmiennych w trakcie operacji nawiązania połączenia z bazą danych.

```
# Nazwa źródła danych, nazwa użytkownika, hasło i atrybuty połączenia.
my $dsn = "DBI:mysql:sampdb:localhost";
my $user_name = "sampadm";
my $password = "secret";
my %conn_attrs = (RaiseError => 1, PrintError => 0, AutoCommit => 1);

# Nawiązanie połączenia z bazą danych.
my $dbh = DBI->connect ($dsn, $user_name, $password, \%conn_attrs);
```

Metoda `connect()` jest wywołana jako `DBI->connect()`, ponieważ to metoda klasy DBI. (Naprawdę nie musisz wiedzieć, co to oznacza. Odrobina żargonu z programowania zorientowanego obiektowo na pewno nie zaszkodzi. Jeżeli koniecznie chcesz wiedzieć, oznacza to, że `connect()` jest funkcją „należącą” do DBI). Metoda `connect()` pobiera wiele argumentów:

- Źródło danych jest określane także mianem „nazwy źródła danych” lub „DSN”. Wspomniane DSN wskazuje moduł DBD do użycia oraz prawdopodobnie również inne parametry.
- Nazwa użytkownika i hasło do konta w MySQL.
- Opcjonalny argument wskazujący dodatkowe atrybuty połączenia. Jeśli zostanie podany, to ten argument powinien odnosić się do tablicy hash zawierającej nazwy atrybutów połączenia oraz ich wartości.

Format źródła danych jest określony przez wymagania konkretnego modułu DBD stosowanego w skrypcie. W przypadku sterownika MySQL dozwolone są następujące formaty DSN:

```
DBI:mysql: nazwa_bazy_danych
DBI:mysql: nazwa_bazy_danych:nazwa_komputera
```

Wielkość liter w członie DBI nie ma znaczenia, ale `mysql` trzeba podać małymi literami. W powyższych wierszach *nazwa_bazy_danych* oznacza używaną bazę danych, natomiast *nazwa_komputera* wskazuje komputer, w którym został uruchomiony serwer. Jeżeli pominiesz nazwę komputera, domyślnie będzie użyta `localhost`. (W punkcie 8.2.9, zatytułowanym „Określenie parametrów połączenia”, omówiono dozwolone formaty źródła danych).

Tablica hash atrybutów połączenia podana w postaci wartości dla `%conn_attrs` włącza atrybut `RaiseError` i wyłącza `PrintError`. Wymienione ustawienia powodują, że DBI sprawdza, czy wystąpiły błędy związane z bazą danych. Po wykryciu błędu następuje wyświetlenie odpowiedniego komunikatu błędu i zakończenie pracy. (Z tego powodu w skrypcie *dump_members.pl* nie znajduje się żaden kod obsługi błędów, tym w całości zajmuje się DBI). W punkcie 8.2.3, zatytułowanym „Obsługa błędów”, przedstawiono alternatywne metody reakcji na błędy.

Tablica hash włącza również atrybut `AutoCommit`. Obecnie nie jest to ściśle konieczne (wymieniony atrybut jest włączony domyślnie), ale wyraźne włączenie `AutoCommit` powoduje włączenie trybu automatycznego zatwierdzania podczas obsługi transakcji. Wprawdzie skrypt nie zawiera żadnych transakcji, ale istnieje możliwość, że w przyszłości DBI będzie wymagać wyraźnego włączania `AutoCommit` w skryptach. Dlatego też zrobienie tego już teraz gwarantuje, że skrypt jest przygotowany na ewentualną zmianę w przyszłości.

Aby wskazać atrybuty połączenia, odniesienie do tablicy hash można podać bezpośrednio w wywołaniu metody `connect()`:

```
my $dbh = DBI->connect ($dsn, $user_name, $password,  
                        { RaiseError => 1, PrintError => 0, AutoCommit => 1 });
```

Dla różnych osób jeden lub drugi styl okazuje się łatwiejszy w odczycie lub edycji, ale pod względem działania oba funkcjonują dokładnie tak samo.

Jeżeli działanie metody `connect()` zakończy się powodzeniem, to zwróci uchwyt do bazy danych, który przypisujemy zmiennej `$dbh`. Domyślnie, metoda `connect()` zwraca wartość `undef` w przypadku niepowodzenia. Jednak ponieważ w skrypcie włączono `RaiseError`, DBI kończy działanie po wyświetleniu komunikatu błędu, gdy działanie metody `connect()` zakończy się niepowodzeniem. (To dotyczy również innych metod DBI. Wkrótce opiszę wartość zwrotną wskazującą na błąd, ale włączenie `RaiseError` powoduje zwrot wartości `undef`, czyli niezdefiniowanej).

Po nawiązaniu połączenia z bazą danych skrypt *dump_members.pl* wykonuje zapytanie `SELECT` w celu pobrania listy członków Ligi. Następnie wykonywana jest pętla odpowiedzialna za przetworzenie wszystkich zwróconych rekordów. Wspomniane rekordy tworzą zbiór wyników. Aby wykonać zapytanie `SELECT`, trzeba je przygotować i później wykonać:

```
# Wykonanie zapytania.  
my $sth = $dbh->prepare ("SELECT last_name, first_name, suffix, email,"  
                        . " street, city, state, zip, phone FROM member ORDER BY last_name");  
$sth->execute ();
```

Metoda `prepare()` jest wywoływana wraz z uchwytom do bazy danych i przekazuje zapytanie SQL sterownikowi w celu jego przetworzenia przed wykonaniem. Na tym etapie pewne sterowniki w rzeczywistości przeprowadzą operacje na zapytaniu. Z kolei inne po prostu zapamiętają je aż do chwili wywołania metody `execute()`, która powoduje faktyczne wykonanie zapytania. Wartością zwrotną metody `prepare()` jest uchwyt zapytania, przypisany tutaj zmiennej `$sth`. Wspomniany uchwyt zapytania jest używany we wszystkich dalszych operacjach związanych z przetwarzaniem zapytania.

Zwróć uwagę na brak średnika na końcu zapytania SQL. Bez wątplenia masz nawyk umieszczania średnika na końcu zapytań SQL (nabyty na skutek długich godzin pracy z klientem `mysql`). Jednak podczas pracy z DBI najlepiej zapomnieć o tym nawyku, ponieważ średnik często jest powodem niepowodzenia wykonania zapytania ze względu na błędy składni. To samo dotyczy dołączania znaków `\g` lub `\G` do ciągów tekstowych zapytań, więc nie rób tego. Wspomniane znaki oznaczające koniec zapytania są konwencjami klienta `mysql` i nie są używane podczas wykonywania zapytań w skryptach DBI. Koniec ciągu tekstowego zapytania wyraźnie kończy zapytanie i nie ma konieczności stosowania specjalnego znaku w tym celu.

Jeżeli wywołujesz metodę bez przekazania jej jakichkolwiek argumentów, to możesz pominąć nawiasy. Dlatego też dwa poniższe wywołania mają taki sam skutek:

```
$sth->execute ();  
$sth->execute;
```

Osobiście preferuję stosowanie nawiasów, ponieważ wtedy wywołanie mniej przypomina odniesienie do zmiennej. Ty możesz na to patrzeć inaczej.

Po wywołaniu metody `execute()` rekordy listy członków Ligi są gotowe do przetworzenia. W skrypcie `dump_members.pl` pętla pobierająca rekord po prostu wyświetla jego zawartość w postaci wartości rozdzielonych tabulatorami:

```
# Odczyt i wyświetlenie wyników zapytania.
while (my @ary = $sth->fetchrow_array ())
{
    print join ("\t", @ary), "\n";
}
$sth->finish ();
```

Metoda `fetchrow_array()` zwraca tablicę zawierającą wartości kolumn bieżącego rekordu lub pustą tablicę, gdy nie ma więcej rekordów. Dlatego też pętla pobiera kolejne rekordy zwrócone przez zapytanie `SELECT` i wyświetla ich zawartość w postaci wartości rozdzielonych tabulatorami.

Wartości `NULL` bazy danych są w skrypcie Perl zwracane w postaci wartości `undef`, ale wyświetlane jako puste ciągi tekstowe, a nie słowo „NULL”. Wartość `undef` ma także jeszcze jeden efekt w trakcie wykonywania skryptu — powoduje wyświetlenie przez interpreter Perl komunikatu ostrzeżenia podobnego do poniższego:

```
Use of uninitialized value in join at dump_members.pl line n.
```

Wspomniane ostrzeżenia są generowane po użyciu polecenia `use warnings` w skrypcie. Po usunięciu tego polecenia i ponownym uruchomieniu skryptu ostrzeżenia nie będą wyświetlane. Jednak tryb ostrzeżeń jest użyteczny i pomaga w odkrywaniu problemów, takich jak próba wyświetlenia wartości niezainicjalizowanej zmiennej. Lepszym sposobem pozbycia się ostrzeżeń jest wykrywanie i zapewnienie obsługi wartości `undef`. Pewne techniki wykonywania tego rodzaju zadań zostaną przedstawione w punkcie 8.2.5, zatytułowanym „Obsługa zapytań zwracających zbiór wyników”.

W poleceniu `print` zwróć uwagę na znaki tabulatora i nowego wiersza (przedstawiane jako sekwencje odpowiednio `\t` i `\n`) ujęte w cudzysłów. W języku Perl sekwencje sterujące są interpretowane tylko po ich ujęciu w cudzysłów, a nie w apostrofy. Jeżeli sekwencja sterująca zostanie ujęta w apostrofy, dane wyjściowe będą zawierały dosłowne wystąpienia `\t` i `\n`.

Po zakończeniu działania pętli pobierającej rekordy wywołanie metody `finish()` wskazuje, że uchwyt zapytania nie jest dłużej potrzebny i wszelkie zaalokowane dla niego zasoby tymczasowe mogą być zwolnione. W omawianym skrypcie wywołanie `finish()` służy jedynie celom ilustracyjnym. W rzeczywistości nie jest tutaj wymagane, ponieważ procedura obsługująca pobieranie rekordu automatycznie wywoła metodę `finish()` po dotarciu do końca zbioru wynikowego. Metoda `finish()` jest znacznie bardziej użyteczna w sytuacjach, gdy pobierana jest jedynie część zbioru wynikowego i nie następuje dotarcie do jego końca (na przykład pobieranych jest tylko kilka pierwszych rekordów). Poczawszy od tego miejsca, przykłady nie zawierają wywołania metody `finish()`, o ile nie jest to konieczne.

Po wyświetleniu listy członków Ligi Historycznej można zamknąć połączenie z serwerem przed zakończeniem działania programu:

```
$dbh->disconnect ();
```

Skrypt *dump_members.pl* ilustruje pewną liczbę koncepcji wspólnych dla większości programów DBI. Na tym etapie prawdopodobnie mógłbyś zacząć tworzenie własnych programów DBI bez konieczności zdobywania dalszej wiedzy. Na przykład, w celu wyświetlenia zawartości innej tabeli musisz jedynie zmienić tekst zapytania SELECT przekazywanego metodzie `prepare()`. Jeśli chcesz zobaczyć inne aplikacje wykorzystujące tę technikę, przejdź do punktu 8.3, zatytułowanego „Praca z DBI”, w którym omówiono sposób wygenerowania listy członków Ligi Historycznej w celu zorganizowania ich dorocznego spotkania, a także katalogu Ligi. Jednak DBI oferuje znacznie więcej innych użytecznych możliwości. W kolejnych punktach niektóre z nich zostaną przedstawione nieco dokładniej. Dzięki temu dowiesz się, jak zrobić więcej, niż tylko wykonywać proste zapytania SELECT w skryptach Perla.

8.2.3. Obsługa błędów

Skrypt *dump_members.pl* włącza atrybut `RaiseError` podczas wywoływania metody `connect()`. Dlatego też błędy automatycznie powodują zakończenie działania skryptu wraz z odpowiednim komunikatem błędu, zamiast jedynie zwracać kod błędu. Błędy można obsługiwać jeszcze na wiele innych sposobów. Na przykład, istnieje możliwość samodzielnego sprawdzania, czy wystąpił błąd, zamiast w tym zakresie całkowicie polegać na DBI.

Aby przekonać się, jak kontrolować zachowanie obsługi błędów w DBI, przyjrzyj się bliżej tablicy hash atrybutu połączenia przekazywanego jako ostatni argument metody `connect()`:

```
# Nazwa źródła danych, nazwa użytkownika, hasło i atrybuty połączenia.
my $dsn = "DBI:mysql:sampdb:localhost";
my $user_name = "sampadm";
my $password = "secret";
my %conn_attrs = (RaiseError => 1, PrintError => 0, AutoCommit => 1);

# Nawiązanie połączenia z bazą danych.
my $dbh = DBI->connect ($dsn, $user_name, $password, \%conn_attrs);
```

Dwa atrybuty powiązane z obsługą błędów to `RaiseError` i `PrintError`:

- Po włączeniu `RaiseError` (ustawienie wartości niezerowej) DBI powoduje zgłoszenie wyjątku po wystąpieniu błędu w metodzie DBI. Domyślnie to powoduje wywołanie metody `die()` w celu wyświetlenia komunikatu i zakończenia działania skryptu.
- Po włączeniu `PrintError`, jeśli wystąpi błąd DBI, wtedy DBI wywołuje metodę `warn()` w celu wyświetlenia odpowiedniego komunikatu, ale wykonywanie skryptu jest kontynuowane.

Domyślnie atrybut `RaiseError` jest wyłączony, a `PrintError` włączony. W takim przypadku, jeśli działanie metody `connect()` zakończy się niepowodzeniem, to DBI wyświetli odpowiedni komunikat, ale działanie skryptu będzie kontynuowane. Dlatego też w domyślnym zachowaniu obsługi błędów po pominięciu czwartego argumentu metody `connect()` możesz stosować poniższe rozwiązanie w celu sprawdzenia, czy wystąpił błąd:

```
my $dbh = DBI->connect ($dsn, $user_name, $password)
    or exit (1);
```

W takim przypadku, jeśli wystąpi błąd, to metoda `connect()` zwróci wartość `undef` wskazującą niepowodzenie i nastąpi wywołanie metody `exit()`. Nie trzeba wyświetlać komunikatu błędu, ponieważ DBI się tym zajmie.

Jeżeli wyraźnie podajesz wartości domyślne dla atrybutów sprawdzania błędów, ustawienia przekazywane metodzie `connect()` prezentują się następująco:

```
my %conn_attrs = (RaiseError => 0, PrintError => 1, AutoCommit => 1);
my $dbh = DBI->connect ($dsn, $user_name, $password, \%conn_attrs)
    or exit (1);
```

To oznacza nieco większą ilość kodu do utworzenia, ale zachowanie w zakresie obsługi błędów jest oczywiste dla osoby czytającej ten kod.

Aby sprawdzić, czy wystąpił błąd, i wyświetlić własny komunikat, należy wyłączyć oba atrybuty — `RaiseError` i `PrintError`:

```
my %conn_attrs = (RaiseError => 0, PrintError => 0, AutoCommit => 1);
my $dbh = DBI->connect ($dsn, $user_name, $password, \%conn_attrs)
    or die "Nie można nawiązać połączenia z serwerem: $DBI::err ($DBI::errstr)\n";
```

Zmienne `$DBI::err` i `$DBI::errstr` użyte w kodzie są użyteczne podczas tworzenia komunikatów błędów. Zawierają kod błędu MySQL i ciąg tekstowy błędu, podobnie jak funkcje `mysql_errno()` i `mysql_error()` dostępne w API C. Jeżeli nie wystąpi żaden błąd, wartością zmiennej `$DBI::err` będzie zero lub `undef`, natomiast `$DBI::errstr` — ciąg tekstowy lub `undef`. (Innymi słowy, obie zmienne przyjmą wartość `false`).

Aby obsługiwać błędów zlecić DBI, to znaczy nie sprawdzać samodzielnie, czy wystąpił błąd, wystarczy włączyć atrybut `RaiseError` i wyłączyć `PrintError`:

```
my %conn_attrs = (RaiseError => 1, PrintError => 0, AutoCommit => 1);
my $dbh = DBI->connect ($dsn, $user_name, $password, \%conn_attrs);
```

To jest podejście najłatwiejsze i stosowane w niemal wszystkich skryptach tworzonych w tym rozdziale. Powodem wyłączenia `PrintError` po włączeniu `RaiseError` jest uniknięcie sytuacji, w której komunikaty błędów będą prezentowane dwukrotnie. (Po włączeniu obu atrybutów w pewnych sytuacjach oba mechanizmy DBI obsługi błędów mogą być wywołane jednocześnie).

Włączenie `RaiseError` może być nieodpowiednie, jeśli chcesz przeprowadzić pewnego rodzaju operacje czyszczące, gdy skrypt kończy działanie. W takim przypadku możesz wykonać żądane operacje przez modyfikację uchwytu `$SIG{__DIE__}`. Innym powodem unikania włączania `RaiseError` jest wyświetlanie przez DBI informacji technicznych w komunikatach, na przykład:

```
disconnect($dbh) invalidates 1 active statement. Either
destroy statement handles or call finish on them before disconnecting.
```

Tego rodzaju informacje są użyteczne dla programisty, ale jednocześnie to rodzaj informacji, których nie chcesz wyświetlać zwykłemu użytkownikowi. W takim przypadku lepiej samodzielnie zająć się sprawdzaniem, czy wystąpił błąd, i wyświetlać komunikaty

czytelniejsze dla osób używających skryptu. Możesz również rozważyć modyfikację uchwytu `$SIG{__DIE__}`. Takie rozwiązanie będzie użyteczne, ponieważ możesz włączyć atrybut `RaiseError` w celu ułatwienia obsługi błędów, ale jednocześnie zastępujesz domyślne komunikaty błędów wyświetlane przez DBI własnymi komunikatami. Aby dostarczyć własny uchwyt `__DIE__`, przed wykonaniem jakiegokolwiek wywołania DBI umieść kod podobny do poniższego:

```
$SIG{__DIE__} = sub { die "Przepraszamy, wystąpił błąd\n"; };
```

Istnieje również możliwość zdefiniowania podprocedury w zwykły sposób i ustawienia wartości uchwytu za pomocą odniesienia do podprocedury:

```
sub die_handler
{
    die "Przepraszamy, wystąpił błąd\n";
}
$SIG{__DIE__} = \&die_handler;
```

Poniższy skrypt *dump_members2.pl* pokazuje sposób utworzenia skryptu sprawdzającego, czy wystąpił błąd, i wyświetlającego własne komunikaty. Skrypt *dump_members2.pl* przetwarza te same polecenia jak w *dump_members.pl*, ale wyraźnie wyłącza `PrintError` i `RaiseError` oraz sprawdza wynik każdego wywołania DBI. Po wystąpieniu błędu skrypt wywołuje podprocedurę `bail_out()` w celu wyświetlenia komunikatu i zawartości zmiennych `DBI::err` i `DBI::errstr` przed zakończeniem działania skryptu:

```
#!/usr/bin/perl
# dump_members2.pl - wygenerowanie listy członków Ligi Historycznej.

use strict;
use warnings;
use DBI;

# Nazwa źródła danych, nazwa użytkownika, hasło i atrybuty połączenia.
my $dsn = "DBI:mysql:sampdb:localhost";
my $user_name = "sampadm";
my $password = "secret";
my %conn_attrs = (RaiseError => 0, PrintError => 0, AutoCommit => 1);

# Nawiązanie połączenia z bazą danych.
my $dbh = DBI->connect ($dsn, $user_name, $password, \%conn_attrs)
    or bail_out ("Nie można nawiązać połączenia z bazą danych.");

# Wykonanie zapytania.
my $sth = $dbh->prepare ("SELECT last_name, first_name, suffix, email,
    . " street, city, state, zip, phone FROM member ORDER BY last_name")
    or bail_out ("Nie można przygotować zapytania.");
$sth->execute ()
    or bail_out ("Nie można wykonać zapytania.");

# Odczyt i wyświetlenie wyników zapytania.
while (my @ary = $sth->fetchrow_array ())
{
    print join ("\t", @ary), "\n";
}
```

```

!$DBI::err
    or bail_out ("Błąd w trakcie pobierania danych.");

$dbh->disconnect ()
    or bail_out ("Nie można zamknąć połączenia z bazą danych.");

# Podprocedura bail_out() - wyświetlenie kodu i komunikatu błędu, a następnie zakończenie działania.
sub bail_out
{
    my $message = shift;
    die "$message\nBłąd $DBI::err ($DBI::errstr)\n";
}

```

Podprocedura `bail_out()` jest podobna do funkcji `print_error()` używanej w programach C tworzonych w rozdziale 7., zatytułowanym „Tworzenie programów MySQL przy użyciu języka C”. Różnica polega na tym, że podprocedura `bail_out()` kończy działanie, zamiast zwracać kontrolę wywołującemu. Dzięki `bail_out()` możesz zaoszczędzić sobie kłopotu związanego z utworzeniem kodu wyświetlającego wartości zmiennych `DBI::err` i `DBI::errstr` za każdym razem, gdy chcesz wyświetlić komunikat błędu. Ponadto, hermetyzacja w podprocedurze operacji wyświetlania komunikatu błędu pozwala na spójną w skrypcie zmianę formatu komunikatów błędów poprzez wprowadzenie odpowiednich modyfikacji w podprocedurze.

Skrypt `dump_members2.pl` przeprowadza operację sprawdzenia po pętli pobierającej rekord. Ponieważ skrypt nie kończy automatycznie działania po wystąpieniu błędu w metodzie `fetchrow_array()`, rozsądne będzie sprawdzenie, czy zakończenie pętli nastąpiło z powodu pobrania całego zbioru wynikowego (normalne zakończenie), czy raczej z powodu wystąpienia błędu. Działanie pętli i tak zakończy się niezależnie od powodu, ale w przypadku błędu nastąpi skrócenie danych wyjściowych skryptu. Bez operacji sprawdzenia, czy wystąpił błąd, osoba używająca skryptu nie będzie wiedziała, że coś poszło źle! Jeżeli samodzielnie zajmujesz się sprawdzaniem, czy wystąpił błąd, nie zapominaj o sprawdzeniu wyniku działania pętli pobierającej dane.

8.2.4. Obsługa zapytań modyfikujących rekordy

Zapytania modyfikujące rekordy, na przykład `DELETE`, `INSERT` i `UPDATE`, są względnie łatwe do przetworzenia w porównaniu do zapytań zwracających rekordy, na przykład `SELECT`, `DESCRIBE` i `SHOW`. W celu przetworzenia zapytania innego niż `SELECT` należy je przekazać metodzie `do()` za pomocą uchwytu bazy danych. Metoda `do()` w jednym kroku przygotowuje i wykonuje zapytanie. Na przykład, aby utworzyć rekord dla nowego członka Ligi, wraz z datą wygaśnięcia członkostwa 3 czerwca 2012 roku, można użyć poniższego kodu:

```

$rows = $dbh->do ("INSERT INTO member (last_name,first_name,expiration)"
    . " VALUES('Brown','Marcia','2012-06-03')");

```

Metoda `do()` zwraca liczbę rekordów, których dotyczyło zapytanie, wartość `undef`, jeśli coś poszło źle, lub `-1`, gdy liczba rekordów jest nieznana. Błędy mogą wystąpić z różnych powodów. (Na przykład, zapytanie może być nieprawidłowe lub nie masz uprawnień dostępu do tabeli). W przypadku wartości zwrotnej innej niż `undef` zwróć uwagę na przypadek, kiedy

zapytanie nie dotyczyło żadnych rekordów. W takiej sytuacji metoda `do()` nie zwraca zero, zamiast tego wartością zwrótną jest ciąg tekstowy `"0E0"` (forma używanej przez Perl notacji naukowej oznaczającej zero). W kontekście liczbowym `"0E0"` przyjmuje wartość 0, ale w poleceniach warunkowych przyjmuje wartość `true` w celu łatwego odróżnienia od `undef`. Jeżeli wartością zwrótną metody `do()` byłoby 0, wtedy znacznie trudniej byłoby odróżnić sytuację wystąpienia błędu (`undef`) od „zapytanie nie dotyczyło żadnych rekordów”. Sprawdzenie, czy wystąpił błąd, można przeprowadzić za pomocą dowolnego z poniższych testów:

```
if (!defined ($rows))
{
    print "Wystąpił błąd\n";
}
if (!$rows)
{
    print "Wystąpił błąd\n";
}
```

W kontekście liczbowym `"0E0"` przyjmuje wartość 0, a więc poniższy kod prawidłowo wyświetla liczbę rekordów, gdy wartość zmiennej `$rows` jest inna niż `undef`:

```
if (!$rows)
{
    print "Wystąpił błąd\n";
}
else
{
    $rows += 0; # Wymuszenie konwersji na liczbę, jeśli wartość wynosi "0E0".
    print "Liczba rekordów, których dotyczyło zapytanie: $rows\n";
}
```

Wartość zmiennej `$rows` można wyświetlić także za pomocą metody `printf()` wraz ze specyfikatorem formatu `%d`, wymuszając w ten sposób konwersję na liczbę:

```
if (!$rows)
{
    print "Wystąpił błąd\n";
}
else
{
    printf "Liczba rekordów, których dotyczyło zapytanie: %d\n", $rows;
}
```

Metoda `do()` jest odpowiednikiem użycia metod `prepare()` i następnie `execute()`. Oznacza to, że przedstawione wcześniej zapytanie `INSERT` może być wykonane w poniższy sposób zamiast użycia metody `do()`:

```
$sth = $dbh->prepare ("INSERT INTO member (last_name,first_name,expiration)"
    . " VALUES('Brown','Marcia','2012-06-03')");
$rows = $sth->execute ();
```

8.2.5. Obsługa zapytań zwracających zbiór wynikowy

W tym punkcie zostaną przedstawione informacje o opcjach dostępnych podczas wykonywania pętli pobierającej rekordy wybrane przez zapytanie SELECT (lub inne niż SELECT zapytania zwracające rekordy, na przykład DESCRIBE, EXPLAIN i SHOW). Dowiesz się również, jak pobrać liczbę rekordów w wyniku, jak obsłużyć zbiór wynikowy bez konieczności użycia pętli oraz jak pobrać jednocześnie cały zbiór wynikowy.

8.2.5.1. Tworzenie pętli pobierania rekordu

W skrypcie *dump_members.pl* dane były pobierane za pomocą sekwencji metod DBI: `prepare()` pozwalała sterownikowi na przygotowanie zapytania, `execute()` rozpoczynała wykonanie zapytania, a `fetchrow_array()` pobierała wszystkie rekordy zbioru wynikowego.

Metody `prepare()` i `execute()` to całkiem standardowe fragmenty procesu przetwarzania dowolnego zapytania zwracającego rekordy. Jednak w rzeczywistości metoda `fetchrow_array()` to tylko jedna z wielu dostępnych metod pobierania rekordów (patrz tabela 8.3).

Tabela 8.3. Oferowane przez DBI metody służące do pobierania rekordów

Nazwa metody	Wartość zwrotna
<code>fetchrow_array()</code>	Tablica wartości.
<code>fetchrow_arrayref()</code>	Odniesienie do tablicy wartości.
<code>fetch()</code>	Taka sama jak metody <code>fetchrow_arrayref()</code> .
<code>fetchrow_hashref()</code>	Odniesienie do tablicy hash wartości rekordów, w której kluczami są nazwy kolumn.

Przedstawione poniżej przykłady pokazują, jak można wykorzystać wymienione metody pobierania rekordów. Wspomniane przykłady wykonują iterację przez rekordy zbioru wynikowego i dla każdego rekordu wyświetlają wartości kolumn rozdzielone przecinkami. W pewnych przypadkach istnieją znacznie efektywniejsze sposoby utworzenia kodu, ale przykłady służą jedynie celom demonstracyjnym (pokazanie składni dostępu do wartości poszczególnych kolumn), a nie do efektywnego działania.

Metody `fetchrow_array()` można użyć następująco:

```
while (my @ary = $sth->fetchrow_array ())
{
    my $delim = "";
    for (my $i = 0; $i < @ary; $i++)
    {
        $ary[$i] = "" if !defined ($ary[$i]); # NULL value?
        print $delim, $ary[$i];
        $delim = ",";
    }
    print "\n";
}
```

Każde wywołanie metody `fetchrow_array()` zwraca tablicę wartości rekordu lub pustą w przypadku, gdy nie ma więcej rekordów do zwrócenia. Pętla wewnętrzna testuje

wartość każdej kolumny, sprawdzając, czy jest zdefiniowana. Jeśli nie jest zdefiniowana, kolumnie przypisywany jest pusty ciąg tekstowy. W ten sposób wartości NULL (przedstawiane przez DBI jako undef) zostają skonwertowane na postać pustych ciągów tekstowych. To może wydawać się zupełnie zbędnym krokiem, w końcu Perl niczego nie wyświetla dla wartości undef i pustego ciągu tekstowego. Jednak jeśli skrypt zostanie uruchomiony z włączonymi ostrzeżeniami, operacja wyświetlenia wartości undef spowoduje wygenerowanie przez Perl odpowiedniego komunikatu ostrzeżenia o użyciu niezainicjalizowanej wartości. Konwersja wartości undef na pusty ciąg tekstowy eliminuje wspomniany komunikat. Podobne konstrukcje spotkasz w innych częściach tego rozdziału.

Jeżeli preferujesz wyświetlenie innej wartości dla undef, na przykład ciągu tekstowego NULL, wtedy wystarczy nieco zmienić pętlę i f:

```
while (my @ary = $sth->fetchrow_array ())
{
    my $delim = "";
    for (my $i = 0; $i < @ary; $i++)
    {
        $ary[$i] = "NULL" if !defined ($ary[$i]); # Wartość NULL?
        print $delim, $ary[$i];
        $delim = ",";
    }
    print "\n";
}
```

Podczas pracy z tablicami wartości kod można nieco skrócić dzięki użyciu metody `map()` do jednoczesnej konwersji wszystkich elementów undef w tablicy:

```
while (my @ary = $sth->fetchrow_array ())
{
    @ary = map { defined ($) ? $_ : "NULL" } @ary;
    print join (",", @ary), "\n";
}
```

Metoda `map()` przetwarza każdy element tablicy za pomocą wyrażenia zdefiniowanego w nawiasie i zwraca tablicę zawierającą wartości wynikowe.

Alternatywą dla przypisania wartości zwrotnej metody `fetchrow_array()` zmiennej tablicy jest pobranie wartości kolumn bezpośrednio do zbioru zmiennych skalarnych. W ten sposób zyskujesz możliwość pracy z nazwami zmiennych znacznie czytelniejszymi niż `$ary[0]`, `$ary[1]` itd. Przyjmujemy założenie, że w zmiennych chcesz umieścić nazwisko członka Ligi oraz jego adres e-mail. Za pomocą metody `fetchrow_array()` możesz wybrać rekordy i pobrać je w następujący sposób:

```
my $sth = $dbh->prepare ("SELECT last_name, first_name, suffix, email"
                        . " FROM member ORDER BY last_name");

$sth->execute ();
while (my ($last_name, $first_name, $suffix, $email)
      = $sth->fetchrow_array ())
{
    # Dowolne operacje na zmiennych.
}
```

Aby w ten sposób użyć listy zmiennych, upewnij się, że kolejność kolumn wybranych przez zapytanie odpowiada kolejności zmiennych, w których mają być umieszczane wartości. DBI nie zna kolejności, w jakiej kolumny zostały podane w zapytaniu SELECT, więc do Ciebie należy prawidłowe przypisanie zmiennych. Wartości kolumn mogą być również automatycznie przypisywane poszczególnym zmiennym w trakcie pobierania rekordu za pomocą techniki o nazwie „dołączania parametru” (patrz punkt 8.2.7, zatytułowany „Miejsca zarezerwowane i zapytania preinterpretowane”).

Jeżeli pobierzesz pojedynczą wartość i umieścisz ją w zmiennej, to uważaj podczas tworzenia przypisania. Po jego umieszczeniu na początku pętli, jak pokazano poniżej, przypisanie będzie działało prawidłowo:

```
while (my ($val) = $sth->fetchrow->array ()) ...
```

Wartość jest pobierana w kontekście listy, więc operacja sprawdzenia zakończy się niepowodzeniem jedynie wtedy, gdy nie ma już więcej rekordów. Jeżeli jednak test zapiszesz w poniższy sposób, niepowodzenia będą pojawiały się niemalże magicznie:

```
while (my $val = $sth->fetchrow->array ()) ...
```

W takim przypadku wartość jest pobierana w kontekście skalarnym, więc jeśli \$val przyjmie wartość zero, undef lub pustego ciągu tekstowego, wtedy cały test przyjmie wartość false i nastąpi wyjście z pętli, nawet jeśli nie został jeszcze pobrany cały zbiór wynikowy.

Druga metoda pobierania rekordu, `fetchrow_arrayref()`, działa podobnie do `fetchrow_array()`, ale zamiast zwracać tablicę zawierającą wartości kolumn dla bieżącego rekordu, zwraca odniesienie do tablicy lub wartość undef, gdy nie ma więcej rekordów. Metody `fetchrow_arrayref()` można użyć w następujący sposób:

```
while (my $ary_ref = $sth->fetchrow_arrayref ())
{
    my $delim = "";
    for (my $i = 0; $i < @{$ary_ref}; $i++)
    {
        $ary_ref->[$i] = "" if !defined ($ary_ref->[$i]); # Wartość NULL?
        print $delim, $ary_ref->[$i];
        $delim = ",";
    }
    print "\n";
}
```

Dostęp do elementów tablicy odbywa się za pomocą odniesienia tablicy `$ary_ref`. Przypomina to odniesienie do wskaźnika, więc można użyć `$ary_ref->[$i]` zamiast `$ary[$i]`. W celu konwersji odniesienia na tablicę należy użyć konstrukcji `@{$ary_ref}`.

Metoda `fetchrow_arrayref()` jest nieodpowiednia dla pobierania zmiennych na liście. Na przykład, przedstawiona poniżej pętla nie działa:

```
while (my ($var1, $var2, $var3, $var4) = @{$sth->fetchrow_arrayref ()})
{
    # Dowolne operacje na zmiennych.
}
```

Pętla działa prawidłowo, dopóki metoda `fetchrow_arrayref()` pobiera rekordy. Kiedy nie ma więcej rekordów do pobrania, metoda `fetchrow_arrayref()` zwraca wartość `undef`, a konstrukcja `@{undef}` jest niedozwolona. (Przypomina to próbę odwołania się do wskaźnika `NULL` w programie C).

Trzecia metoda pobierania rekordu `fetchrow_hashref()` jest używana w następujący sposób:

```
while (my $hash_ref = $sth->fetchrow_hashref ())
{
    my $delim = "";
    foreach my $key (keys (%{$hash_ref}))
    {
        $hash_ref->{$key} = "" if !defined ($hash_ref->{$key}); # Wartość NULL?
        print $delim, $hash_ref->{$key};
        $delim = ",";
    }
    print "\n";
}
```

Każde wywołanie metody `fetchrow_hashref()` zwraca odniesienie do tablicy hash wartości rekordów, w której kluczami są nazwy kolumn, a `undef` oznacza brak kolejnych rekordów. W takim przypadku wartości kolumn nie pojawiają się w żadnej określonej kolejności, ponieważ elementy tablicy hash w Perlu są nieuporządkowane. Jednak klucze DBI elementów tablicy hash używają nazw kolumn, a więc `$hash_ref` daje pojedynczą zmienną, za pomocą której można uzyskać dostęp do dowolnej wartości kolumny przez jej nazwę. Oznacza to możliwość pobrania wartości (lub ich wskazanego podzbioru) w dowolnej kolejności, bez konieczności posiadania wiedzy o kolejności kolumn, w jakiej zostały pobrane przez zapytanie `SELECT`. Na przykład, w celu uzyskania dostępu do kolumn nazwiska i adresu e-mail członka Ligi można użyć poniższego fragmentu kodu:

```
while (my $hash_ref = $sth->fetchrow_hashref ())
{
    my $delim = "";
    foreach my $key ("last_name", "first_name", "suffix", "email")
    {
        $hash_ref->{$key} = "" if !defined ($hash_ref->{$key}); # Wartość NULL?
        print $delim, $hash_ref->{$key};
        $delim = ",";
    }
    print "\n";
}
```

Metoda `fetchrow_hashref()` jest szczególnie użyteczna, gdy zachodzi potrzeba przekazania rekordu wartości do funkcji, która nie zna kolejności wymienienia kolumn w zapytaniu `SELECT`. W takim przypadku metoda `fetchrow_hashref()` pobiera rekordy i przekazuje funkcji, która uzyskuje dostęp do tych wartości za pomocą nazw kolumn.

Podczas używania metody `fetchrow_hashref()` należy pamiętać o wymienionych poniżej kwestiach:

- Jeżeli wymagasz najlepszej możliwej wydajności, to metoda `fetchrow_hashref()` nie jest optymalnym wyborem. Działa ona mniej efektywnie niż `fetchrow_array()` lub `fetchrow_arrayref()`.

- Domyślnie, nazwy kolumn są używane jako klucze zapisane literami o takiej samej wielkości jak nazwy kolumn w zapytaniu SELECT. W bazie danych MySQL nazwy kolumn nie rozróżniają wielkości liter, więc zapytanie będzie działało niezależnie od wielkości liter użytej do zapisania nazw kolumn. Jednak klucze w tablicy hash Perla *rozróżniają* wielkość liter, co może spowodować problemy. Aby uniknąć potencjalnych problemów z wielkością liter, metodzie `fetchrow_hashref()` trzeba nakazać stosowanie określonej wielkości liter w nazwach kolumn. Odbywa się to przez przekazanie atrybutu `NAME_lc` lub `NAME_uc`:

```
$hash_ref = $sth->fetchrow_hashref ("NAME_lc"); # Używaj nazw zapisanych małymi literami.  
$hash_ref = $sth->fetchrow_hashref ("NAME_uc"); # Używaj nazw zapisanych wielkimi literami.
```

- Tablica hash zawiera po jednym elemencie dla unikalnej nazwy kolumny. Jeżeli przeprowadzasz złączenia obejmujące kolumny z wielu tabel i o powtarzających się nazwach, wtedy nie będziesz miał dostępu do wszystkich wartości kolumn. Po wykonaniu poniższego zapytania metoda `fetchrow_hashref()` zwróci tablicę hash zawierającą tylko jeden element `name`:

```
SELECT a.name, b.name FROM a INNER JOIN b WHERE a.name = b.name
```

Aby uniknąć tego rodzaju problemu, należy stosować aliasy w celu zagwarantowania unikalności nazw kolumn. Po modyfikacji powyższego zapytania na postać przedstawioną poniżej metoda `fetchrow_hashref()` zwróci odniesienie do tablicy hash zawierającej dwa elementy — `name` i `name2`:

```
SELECT a.name, b.name AS name2 FROM a INNER JOIN b WHERE a.name = b.name
```

8.2.5.2. Ustalanie liczby rekordów zwróconych przez zapytanie

W jaki sposób można określić liczbę rekordów zwróconych przez zapytanie SELECT lub mu podobne? Jednym z możliwych rozwiązań jest zliczanie rekordów w trakcie ich pobierania. W rzeczywistości to *jedyny* przenośny sposób w DBI pozwalający na ustalenie liczby rekordów zwróconych przez zapytanie SELECT. Sterownik MySQL oferuje metodę `rows()`, którą można wywołać po wykonaniu metody `execute()`. To rozwiązanie jednak nie jest przenośne do innych systemów baz danych, a dokumentacja DBI wyraźnie zniechęca do używania metody `rows()` dla zapytań SELECT. Nawet w przypadku MySQL, jeśli ustawiony został atrybut `mysql_use_result`, to metoda `rows()` nie zwróci prawidłowej liczby aż do chwili pobrania wszystkich rekordów. Dlatego też w tym przypadku również trzeba zliczać rekordy w trakcie ich pobierania.

8.2.5.3. Pobieranie wyniku składającego się z pojedynczego rekordu

Jeżeli zbiór wyników składa się z pojedynczego rekordu, wtedy nie ma potrzeby używania pętli w celu jego pobrania. Przyjmujemy założenie, że chcemy utworzyć skrypt `count_members.pl`, podający liczbę członków Ligi Historycznej. Kod wykonujący odpowiednie zapytanie do bazy danych przedstawia się następująco:

```
# Wykonanie zapytania.
my $sth = $dbh->prepare ("SELECT COUNT(*) FROM member");
$sth->execute ();

# Odczyt i wyświetlenie wyników zapytania.
my $count = $sth->fetchrow_array ();
$sth->finish ();
$count = "Nie można ustalić liczby członków Ligi." if !defined ($count);
print "$count\n";
```

Powyższe zapytanie SELECT zwraca tylko jeden rekord i nie ma potrzeby użycia pętli, metodę `fetchrow_array()` można wywołać tylko raz. Ponadto, ponieważ wybierana jest tylko jedna kolumna, nie ma nawet potrzeby przypisywania wartości zwrotnej do tablicy. Po wywołaniu metody `fetchrow_array()` w kontekście skalarnym (gdzie oczekiwana jest pojedyncza wartość zamiast listy) metoda zwraca jedną kolumnę rekordu lub wartość `undef`, jeśli nie ma dostępnego rekordu. DBI nie definiuje elementu rekordu zwracanego przez metodę `fetchrow_array()` wywołaną w kontekście skalarnym, ale to nie ma znaczenia dla przedstawionego powyżej zapytania. Pobierana jest tylko pojedyncza wartość, więc nie ma niejednoznaczności w określeniu, która wartość została pobrana.

Kod wywołuje metodę `finish()` w celu zwolnienia zasobów zajmowanych przez zbiór wyników, nawet jeśli składa się on tylko z jednego rekordu. (Metoda `fetchrow_array()` zwalnia zasoby zbioru wyników po ustaleniu, że dotarła do jego końca, ale to może nastąpić dopiero po jej drugim wywołaniu).

Inny typ zapytania, który może zwrócić pojedynczy rekord, to zapytanie zawierające klauzulę `LIMIT 1`, ograniczającą do jednego liczbę zwracanych rekordów. Takie rozwiązanie jest najczęściej stosowane do zwrócenia rekordu zawierającego wartość maksymalną lub minimalną danej kolumny. Na przykład, przedstawione poniżej zapytanie wyświetla imię i nazwisko oraz datę urodzenia najmłodszego prezydenta:

```
my $stmt = "SELECT last_name, first_name, birth FROM president"
          . " WHERE birth = (SELECT MAX(birth) FROM president)";
my $sth = $dbh->prepare ($stmt);
$sth->execute ();

my ($last_name, $first_name, $birth) = $sth->fetchrow_array ();
$sth->finish ();
if (!defined ($last_name))
{
    print "Zapytanie nie zwróciło wyniku.\n";
}
else
{
    print "Najmłodszy prezydent: $first_name $last_name ($birth).\n";
}
```

Inne rodzaje zapytań, w których niepotrzebna jest pętla pobierająca rekordy, to zapytanie wykorzystujące funkcję `MAX()` lub `MIN()` do wybrania jednego rekordu. Jednak we wszystkich wspomnianych przypadkach jeszcze łatwiejszym sposobem pobrania wyniku składającego się z pojedynczego rekordu jest użycie metody `selectrow_array()`, która w pojedynczym wywołaniu łączy metody `prepare()`, `execute()` i operację pobrania rekordu. Wartością zwrótną jest tablica (nie odniesienie) lub pusta tablica, jeśli zapytanie nie zwróciło rekordu

bądź wystąpił błąd. Poprzedni przykład można więc zmodyfikować na postać używającą metody `selectrow_array()`:

```
my $stmt = "SELECT last_name, first_name, birth FROM president"
        . " WHERE birth = (SELECT MAX(birth) FROM president)";
my ($last_name, $first_name, $birth) = $dbh->selectrow_array ($stmt);
if (!defined ($last_name))
{
    print "Zapytanie nie zwróciło wyniku.\n";
}
else
{
    print "Najmłodszy prezydent: $first_name $last_name ($birth).\n";
}
```

8.2.5.4. Praca z pełnymi zbiorami wyników

Podczas używania pętli pobierania DBI nie zapewnia sposobu na przetwarzanie rekordów w kolejności innej niż ta, w jakiej są zwracane przez pętlę. Ponadto, po pobraniu rekordu poprzedni zostaje utracony, o ile nie zostały podjęte kroki w celu jego zachowania w pamięci. Takie zachowanie nie zawsze jest pożądane. Na przykład, będzie nieodpowiednie w przypadku, gdy trzeba wielokrotnie przetworzyć rekord podczas przeprowadzania obliczeń statystycznych. (Prawdopodobnie pierwsze przejście przez zbiór wynikowy ma na celu zebranie pewnych ogólnych danych statystycznych dotyczących danych, a dopiero następne służy przeprowadzeniu określonych operacji analitycznych).

Dostęp do zbioru wynikowego jako całości można uzyskać na wiele sposobów. Możesz wykonać zwykłą pętlę pobierającą rekordy i każdy z nich zapisać po pobraniu. Ewentualnie można użyć metody od razu pobierającej cały zbiór wynikowy. Niezależnie od wybranego rozwiązania, skutkiem będzie macierz zawierająca jeden wiersz dla każdego rekordu zbioru wynikowego i wskazaną liczbę kolumn. Następnie, elementy macierzy można przetworzyć w wybranej kolejności oraz dowolną liczbę razy. Poniżej omówiono oba podejścia.

Jednym ze sposobów użycia pętli pobierającej rekordy do przechwycenia zbioru wynikowego jest wywołanie metody `fetchrow_array()` i zapisanie tablicy odniesień do rekordów. Przedstawiony poniżej kod działa tak samo jak pętla „pobierz i wyświetl” użyta w skrypcie `dump_members.pl`. Różnica polega na zapisaniu wszystkich rekordów, a następnie wyświetleniu macierzy. Przykład pokazuje, jak ustalić liczbę wierszy i kolumn w macierzy oraz jak uzyskać dostęp do poszczególnych jej elementów:

```
my @matrix = (); # Tablica odniesień do tablicy.

while (my @ary = $sth->fetchrow_array ()) # Pobranie każdego rekordu.
{
    push (@matrix, [ @ary ]); # Zachowanie odniesienia do właśnie pobranego rekordu.
}

# Określenie wymiarów macierzy.
my $rows = scalar (@matrix);
my $cols = ($rows == 0 ? 0 : scalar (@{$matrix[0]}));
for (my $i = 0; $i < $rows; $i++) # Wyświetlenie wszystkich wierszy.
```



```
{
    my $delim = "";
    for (my $j = 0; $j < $cols; $j++)
    {
        $matrix[$i][$j] = "" if !defined ($matrix[$i][$j]); # Wartość NULL?
        print $delim, $matrix[$i][$j];
        $delim = ",";
    }
    print "\n";
}
```

Podczas sprawdzania wymiarów macierzy liczba wierszy musi być ustalona jako pierwsza, ponieważ obliczenie liczby kolumn zależy od tego, czy macierz jest pusta. Jeżeli wartość \$rows wynosi 0, to macierz jest pusta i wartością \$cols również będzie 0. W przeciwnym razie liczba kolumn zostanie obliczona na podstawie liczby elementów w pierwszym wierszu za pomocą składni @{\$matrix[0]}, pozwalającej na uzyskanie dostępu do wiersza jako całości.

Powyższy fragment kodu pobierał rekord jako tablicę, a następnie zachowywał odniesienie do tego rekordu. Być może sądzisz, że znacznie efektywniejszym rozwiązaniem będzie wywołanie metody fetchrow_arrayref() zamiast bezpośredniego pobierania odniesień do rekordów:

```
my @matrix = (); # Tablica odniesień do tablicy.
while (my $ary_ref = $sth->fetchrow_arrayref ())
{
    push (@matrix, $ary_ref); # Zachowanie odniesienia do właśnie pobranego rekordu.
}
```

Takie rozwiązanie jednak nie działa, ponieważ metoda fetchrow_arrayref() ponownie używa tablicy, do której prowadzi odniesienie. Otrzymana w wyniku macierz jest tablicą odniesień, z których każde prowadzi do tego samego rekordu pobranego jako ostatni. Dlatego też, jeśli chcesz utworzyć macierz przez pobieranie jednorazowo jednego rekordu, użyj metody fetchrow_array() zamiast fetchrow_arrayref().

Alternatywą dla użycia pętli jest wywołanie jednej z metod DBI zwracających cały zbiór wyników. Na przykład, metoda fetchall_arrayref() zwraca odniesienie do tablicy odniesień, z których każde prowadzi do zawartości jednego rekordu zbioru wyników. (Innymi słowy, wartość zwrotna jest odniesieniem do macierzy). Aby użyć metody fetchall_arrayref(), należy wywołać prepare() i execute(), a następnie pobrać zbiór wyników w przedstawiony poniżej sposób:

```
# Pobranie wszystkich rekordów jako odniesień do tablicy odniesień.
my $matrix_ref = $sth->fetchall_arrayref ();
```

Określenie wielkości macierzy i dostęp do jej elementów odbywają się następująco:

```
# Określenie wymiarów macierzy.
my $rows = (!defined ($matrix_ref) ? 0 : scalar (@{$matrix_ref}));
my $cols = ($rows == 0 ? 0 : scalar (@{$matrix_ref->[0]}));

for (my $i = 0; $i < $rows; $i++) # Wyświetlenie wszystkich wierszy.
{
```

```

my $delim = "";
for (my $j = 0; $j < $cols; $j++)
{
    $matrix_ref->[$i][$j] = "" if !defined ($matrix_ref->[$i][$j]); # Wartość NULL?
    print $delim, $matrix_ref->[$i][$j];
    $delim = ",";
}
print "\n";
}

```

Jeżeli zbiór wynikowy jest pusty, wtedy metoda `fetchall_arrayref()` zwraca odniesienie do pustej tablicy. W przypadku wystąpienia błędu wynikiem jest `undef` i dlatego przy wyłączonym atrybucie `RaiseError` konieczne jest sprawdzenie wartości zwrotnej przed jej użyciem.

Na liczbę wierszy i kolumn wpływ ma to, czy macierz jest pusta. W celu uzyskania dostępu do całego wiersza `$i` macierzy, jakby był tablicą, konieczne jest użycie składni `@{$matrix_ref->[$i]}`.

Użycie metody `fetchall_arrayref()` w celu pobrania zbioru wynikowego jest niewątpliwie łatwiejsze niż tworzenie pętli pobierającej rekordy, choć sama składnia uzyskania dostępu do elementów tablicy jest trudniejsza. Metoda podobna do `fetchall_arrayref()`, ale wykonująca jeszcze większą ilość pracy za programistę, to `selectall_arrayref()`. Wykonuje ona za programistę całą sekwencję realizowaną wcześniej jako wywołanie metod `prepare()` i `execute()` oraz przeprowadza iterację pętli odpowiedzialnej za pobieranie rekordów. Aby użyć metody `selectall_arrayref()`, należy zapytanie przekazać jej bezpośrednio za pomocą uchwytu bazy danych:

```

# Pobranie wszystkich rekordów jako odniesień do tablicy odniesień.
my $matrix_ref = $dbh->selectall_arrayref ($stmt);

# Określenie wymiarów macierzy.
my $rows = (!defined ($matrix_ref) ? 0 : scalar (@{$matrix_ref}));
my $cols = ($rows == 0 ? 0 : scalar (@{$matrix_ref->[0]}));
for (my $i = 0; $i < $rows; $i++)      # Wyświetlenie wszystkich wierszy.
{
    my $delim = "";
    for (my $j = 0; $j < $cols; $j++)
    {
        $matrix_ref->[$i][$j] = "" if !defined ($matrix_ref->[$i][$j]); # Wartość NULL?
        print $delim, $matrix_ref->[$i][$j];
        $delim = ",";
    }
    print "\n";
}

```

8.2.5.5. Sprawdzanie pod kątem wartości NULL

Podczas pobierania informacji z bazy danych może wystąpić potrzeba odróżnienia wartości NULL kolumn od wartości zero lub pustego ciągu tekstowego. To jest łatwe zadanie do wykonania, ponieważ DBI zwraca wartości NULL kolumn jako `undef`. Jednak konieczne jest upewnienie się o przeprowadzeniu prawidłowego testu. Jeśli wykonasz poniższy fragment kodu, to komunikat `false!` będzie wyświetlony trzykrotnie:

```
$col_val = undef; if (!$col_val) { print "Fałsz!\n"; }
$col_val = 0;    if (!$col_val) { print "Fałsz!\n"; }
$col_val = "";   if (!$col_val) { print "Fałsz!\n"; }
```

Powyższy fragment kodu pokazuje, że użyta forma testu nie potrafi rozróżnić wartości undef, 0 i pustego ciągu tekstowego. Kolejny fragment kodu wyświetla Fałsz! dwukrotnie, co wskazuje na brak możliwości odróżnienia wartości undef od pustego ciągu tekstowego:

```
$col_val = undef; if ($col_val eq "") { print "Fałsz!\n"; }
$col_val = "";   if ($col_val eq "") { print "Fałsz!\n"; }
```

Z kolei poniższy fragment kodu wyświetla te same dane wyjściowe, pokazując, że drugi test nie potrafi odróżnić wartości 0 od pustego ciągu tekstowego:

```
$col_val = "";
if ($col_val eq "") { print "Fałsz!\n"; }
if ($col_val == 0) { print "Fałsz!\n"; }
```

Aby odróżnić wartość undef (NULL) kolumny od wartości innych niż undef, należy użyć metody defined(). Kiedy już wiesz, że wartość nie przedstawia NULL, rozróżnienie między innymi typami wartości można przeprowadzić za pomocą odpowiednich testów, na przykład:

```
if (!defined ($col_val)) { print "NULL\n"; }
elsif ($col_val eq "") { print "Pusty ciąg tekstowy.\n"; }
elsif ($col_val == 0) { print "Zero.\n"; }
else { print "Inna wartość.\n"; }
```

Bardzo ważne jest przeprowadzanie testów w odpowiedniej kolejności, ponieważ zarówno drugie, jak i trzecie porównanie zwraca wartość true, gdy \$col_val jest pustym ciągiem tekstowym. Jeżeli zamienisz kolejność wymienionych porównań, to nieprawidłowo zinterpretujesz pusty ciąg tekstowy jako zero.

8.2.6. Cytowanie znaków specjalnych w ciągach tekstowych zapytań

Jak dotąd zapytania tworzyliśmy w najprostszy z możliwych sposobów, czyli za pomocą prostych cytowanych ciągów tekstowych. To powoduje problem na poziomie leksykalnym Perla, gdy cytowany ciąg tekstowy zawiera cytowane wartości. Problem może powstać także na poziomie SQL, gdy będziesz chciał wstawić lub wybrać wartości zawierające znaki cytowania, ukośniki lub dane binarne. Jeżeli utworzysz zapytanie w postaci cytowanego ciągu tekstowego Perla, to wszystkie wystąpienia znaków cytowania w samym zapytaniu musisz poprzedzić znakiem specjalnym:

```
$stmt = 'INSERT INTO absence VALUES(14,\'2012-09-16\')';
$stmt = "INSERT INTO absence VALUES(14,\'2012-09-16\')";
```

Zarówno Perl, jak i MySQL pozwalają na cytowanie ciągów tekstowych za pomocą apostrofów i cudzysłowu, więc czasami, stosując w zapytaniu różne znaki cytowania, można uniknąć konieczności poprzedzania znaków cytowania znakami specjalnymi:

```
$stmt = 'INSERT INTO absence VALUES(14,"2012-09-16")';
$stmt = "INSERT INTO absence VALUES(14,'2012-09-16')";
```

Jednak musisz zachować ostrożność, aby ciągi tekstowe były interpretowane zgodnie z oczekiwaniami. Rozważ przedstawione poniżej czynniki:

- Dwa typy cytowania nie są odpowiednikami w Perlu. Odniesienia do zmiennych są interpretowane jedynie po ujęciu w cudzysłów. Dlatego też apostrofy nie są zbyt użyteczne podczas tworzenia zapytań przez osadzanie odniesień do zmiennych w ciągu tekstowym zapytania. Na przykład, jeśli wartość zmiennej `$var` wynosi 14, dwa poniższe ciągi tekstowe nie są interpretowane jednakowo:

```
"SELECT * FROM member WHERE member_id = $var"
'SELECT * FROM member WHERE member_id = $var'
```

Perl interpretuje je w następujący sposób:

```
"SELECT * FROM member WHERE member_id = 14"
'SELECT * FROM member WHERE member_id = $var'
```

Bez wątplenia pierwszy ciąg tekstowy jest tym, który chcesz przekazać do serwera MySQL. W przypadku drugiego serwer zinterpretuje `$var` jako dosłowną nazwę kolumny w tabeli `member`.

- Apostrofy i cudzysłów nie zawsze są odpowiednikami w MySQL. Jeżeli serwer działa z wyłączonym trybem SQL o nazwie `ANSI_QUOTES`, wówczas faktycznie dowolnego z wymienionych znaków można używać do cytowania ciągu tekstowego. Jednak po włączeniu trybu `ANSI_QUOTES` ciągi tekstowe muszą być cytowane za pomocą apostrofów. Cudzysłów może być stosowany jedynie do cytowania identyfikatorów takich jak nazwa bazy danych lub tabeli. Dlatego też najbezpieczniejszy sposób na cytowanie ciągów tekstowych to użycie apostrofów, ponieważ działają one niezależnie od ustawienia trybu `ANSI_QUOTES`.

Na poziomie Perla alternatywą dla cytowania ciągów tekstowych za pomocą cudzysłowu jest użycie konstrukcji `qq{ }`, która nakazuje Perlowi potraktowanie wszystkiego między `qq{ i }` jako ciągu tekstowego ujętego w cudzysłów. Na przykład, dwa przedstawione poniżej polecenia mają taki sam efekt:

```
$date = "2012-09-16";
$date = qq{2012-09-16};
```

Za pomocą konstrukcji `qq{ }` możesz tworzyć zapytania bez przejmowania się kwestiami związanymi z cytowaniem. Po prostu zyskujesz możliwość dowolnego stosowania apostrofów i cudzysłowu w ciągu tekstowym zapytania bez konieczności ich dalszego cytowania. Ponadto, odniesienia do zmiennych są interpretowane. Obie wymienione właściwości konstrukcji `qq{ }` zaprezentowano w poniższym zapytaniu `INSERT`:

```
$id = 14;
$date = "2012-09-16";
$stmt = qq{INSERT INTO absence VALUES($id,'$date')};
```

Nie trzeba używać nawiasów klamrowych jako ograniczników qq. Dozwolone są także inne formy, na przykład qq() lub qq//, o ile zamykający ogranicznik nie występuje w ciągu tekstowym. Osobiście preferuję wariant qq{}, ponieważ wystąpienie nawiasu klamrowego } w ciągu tekstowym jest mniej prawdopodobne niż nawiasu) lub ukośnika /. Wystąpienie znaku takiego jak ogranicznik końcowy spowoduje przedwczesne oznaczenie końca ciągu tekstowego zapytania. Na przykład, nawias) występuje na końcu przedstawionego powyżej zapytania INSERT, a więc wariantu qq() nie można użyć jako konstrukcji cytowania ciągu tekstowego zapytania.

Konstrukcja qq{} może obejmować więcej niż tylko jeden wiersz, co jest użyteczne, jeśli chcesz, aby ciąg tekstowy zapytania odróżniał się od otaczającego go kodu Perl:

```
$id = 14;
$date = "2012-09-16";
$stmt = qq{
    INSERT INTO absence VALUES($id,$date')
};
```

To jest również użyteczne, jeśli po prostu chcesz sformatować zapytanie w kilku wierszach, zwiększając tym samym jego czytelność. Na przykład, zapytanie SELECT w skrypcie *dump_members.pl* przedstawia się następująco:

```
$sth = $dbh->prepare ("SELECT last_name, first_name, suffix, email,"
    . " street, city, state, zip, phone FROM member ORDER BY last_name");
```

Po zastosowaniu konstrukcji qq{} może być zapisane w poniższej postaci:

```
$sth = $dbh->prepare (qq{
    SELECT
        last_name, first_name, suffix, email,
        street, city, state, zip, phone
    FROM member
    ORDER BY last_name
});
```

Ciągi tekstowe ujęte w cudzysłów również mogą obejmować więcej niż tylko jeden wiersz. Jednak uważam, że konstrukcja qq{} jest lepiej widoczna w kodzie niż "", a samo zapytanie staje się łatwiejsze w odczycie. W tej książce stosowane są obie formy; wybierz tę, którą bardziej Ci odpowiada.

Konstrukcja qq{} obsługuje kwestie cytowania na poziomie leksykalnym Perla, więc możesz stosować dowolne cytowanie, a Perl nie będzie zgłaszał żadnych problemów. Jednak trzeba również pamiętać o składni na poziomie SQL. Spójrz na poniższą próbę wstawienia rekordu do tabeli member:

```
$last = "O'Malley";
$first = "Brian";
$expiration = "2013-09-01";
$rows = $dbh->do (qq{
    INSERT INTO member (last_name,first_name,expiration)
    VALUES('$last','$first','$expiration')
});
```

Ciąg tekstowy wysyłany przez metodę `do()` do MySQL przedstawia się następująco:

```
INSERT INTO member (last_name,first_name,expiration)
VALUES('O'Malley','Brian','2013-09-01')
```

To nie jest poprawny kod SQL, ponieważ w `'O'Malley'` apostrof znajduje się w ciągu tekstowym ujętym w apostrofy. Z takim problemem zetknęliśmy się już w rozdziale 7., zatytułowanym „Tworzenie programów MySQL przy użyciu języka C”. Wtedy rozwiązanie polegało na użyciu funkcji `mysql_real_escape_string()`. DBI oferuje podobny mechanizm: dla każdej cytowanej wartości, której chcesz użyć dosłownie w zapytaniu, należy wywołać metodę `quote()`, a następnie wykorzystać jej wartość zwrótną. Poprzedni przykład, zapisany prawidłowo, przedstawia się następująco:

```
$last = $dbh->quote ("O'Malley");
$first = $dbh->quote ("Brian");
$expiration = $dbh->quote ("2013-09-01");
$rows = $dbh->do (qq{
    INSERT INTO member (last_name,first_name,expiration)
    VALUES($last,$first,$expiration)
});
```

Teraz ciąg tekstowy wysyłany przez metodę `do()` do MySQL przypomina przedstawiony poniżej, a znaki cytowania wewnątrz cytowanego ciągu tekstowego są zapisywane poprawnie:

```
INSERT INTO member (last_name,first_name,expiration)
VALUES('O\'Malley','Brian','2013-09-01')
```

Zwróć uwagę, że kiedy odwołujesz się do zmiennych `$last` i `$first` w ciągu tekstowym zapytania, to nie musisz dodawać nowych znaków cytowania, ponieważ metoda `quote()` robi to za Ciebie. Jeśli jednak dodasz znaki cytowania, to polecenie będzie miało ich zbyt wiele, jak pokazano w poniższym przykładzie:

```
$value = "Paweł";
$quoted_value = $dbh->quote ($value);

print "Cytowana wartość to: $quoted_value\n";
print "Cytowana wartość to: '$quoted_value'\n";
```

Powyższe polecenia wygenerują następujące dane wyjściowe:

```
Cytowana wartość to: 'Paweł'
Cytowana wartość to: ''Paweł''
```

W drugim z przedstawionych wierszy ciąg tekstowy zawiera zbyt wiele znaków cytowania.

8.2.7. Miejsca zarezerwowane i zapytania preinterpretowane

W poprzednich punktach przygotowaliśmy zapytania przez umieszczanie wartości danych przeznaczonych do wstawienia lub użycia w charakterze kryteriów wyboru bezpośrednio w ciągu tekstowym zapytania. Nie ma takiej konieczności. DBI pozwala

na użycie w ciągu tekstowym zapytania znaków specjalnych nazywanych „miejscami zarezerwowanymi” i dostarczenie wartości wstawianych w te miejsca w trakcie wykonywania zapytania. Taki mechanizm nosi nazwę „dołączania wartości do zapytania”. Dzięki temu można otrzymać zalety cytowania znaków za pomocą metody `quote()` bez konieczności jej wyraźnego wywołania. Ponadto, jeśli zapytanie jest wielokrotnie wykonywane w pętli, można je wcześniej preinterpretować, a następnie wielokrotnie wykonywać. W ten sposób unika się obciążenia związanego z interpretacją zapytania przed jego każdym wywołaniem.

Jako ilustrację dla sposobu działania miejsc zarezerwowanych rozważmy następujący przykład. Przyjmujemy założenie, że w szkole rozpoczyna się nowy semestr, więc chcesz wyczyścić tabelę `student` i zainicjalizować ją danymi nowych uczniów za pomocą listy nazwisk znajdujących się w pliku. Bez użycia miejsc zarezerwowanych usunięcie zawartości istniejącej tabeli i wczytanie nowych nazwisk można przeprowadzić następująco:

```
$dbh->do (qq{ DELETE FROM student } ); # Usunięcie istniejących rekordów.
while (<>)                             # Odczyt wszystkich wierszy danych wejściowych.
{                                       # Użycie odczytanego wiersza do wstawienia nowego rekordu.
    chomp;
    $_ = $dbh->quote ($_);
    $dbh->do (qq{ INSERT INTO student SET name = $_ });
}
```

Takie podejście wymaga samodzielnego zajęcia się obsługą znaków specjalnych w wartościach danych za pomocą wywołania metody `quote()`. To jednocześnie nieefektywne podejście, ponieważ podstawowa forma zapytania `INSERT` jest taka sama za każdym razem, a metoda `do()` wywołuje metody `prepare()` i `execute()` w trakcie każdej iteracji pętli. Znacznie efektywniejszym rozwiązaniem jest jednokrotne wywołanie metody `prepare()` i konfiguracja zapytania `INSERT` przed wejściem do pętli, a następnie w pętli wywołanie jedynie metody `execute()`. W ten sposób unikamy wszystkich wywołań metody `prepare()` poza pierwszym. DBI pozwala na zastosowanie tego rodzaju rozwiązania:

```
$dbh->do (qq{ DELETE FROM student } ); # Usunięcie istniejących rekordów.
my $sth = $dbh->prepare (qq{ INSERT INTO student SET name = ? });
while (<>)                             # Odczyt wszystkich wierszy danych wejściowych.
{                                       # Użycie odczytanego wiersza do wstawienia nowego rekordu.
    chomp;
    $sth->execute ($_);
}
```

Ogólnie rzecz biorąc, jeśli metoda `do()` jest wywoływana wewnątrz pętli, to znacznie lepsze rozwiązanie polega na wywołaniu metody `prepare()` przed wejściem do pętli, a następnie `execute()` wewnątrz pętli. Zwróć uwagę na znak zapytania umieszczony w zapytaniu `INSERT`. To jest wspomniane wcześniej miejsce zarezerwowane. Kiedy wywołujesz metodę `execute()`, przekazujesz wartość zastępującą miejsce zarezerwowane w trakcie wysyłania zapytania do serwera. DBI automatycznie zajmuje się cytowaniem znaków specjalnych w wartości, a więc nie ma potrzeby wywołania metody `quote()`.

Podczas pracy z miejscami zarezerwowanymi warto pamiętać o kilku kwestiach:

- W ciągu tekstowym zapytania nie cytuj znaku miejsca zarezerwowanego. Jeżeli to zrobisz, DBI nie rozpozna go jako miejsca zarezerwowanego.
- Nie używaj metody `quote()` do wskazywania wartości dla miejsca zarezerwowanego, ponieważ doprowadzi to do umieszczenia dodatkowych znaków cytowania we wstawianych wartościach.
- W ciągu tekstowym zapytania może znajdować się więcej niż tylko jedno miejsce zarezerwowane. Upewnij się o przekazaniu metodzie `execute()` tylu wartości, ile miejsc zarezerwowanych zdefiniowano w zapytaniu.
- Każde miejsce zarezerwowane musi wskazywać pojedynczą wartość, a nie listę wartości. Na przykład, aby podać wiele wartości danych, nie możesz przygotować i wykonać poniższego zapytania:

```
my $sth = $dbh->prepare (qq{
    INSERT INTO member last_name, first_name VALUES(?)
});
$sth->execute ("Adams,Bill,2014-07-19");
```

Zamiast tego wartości musisz podać oddzielnie i zdefiniować po jednym miejscu zarezerwowanym dla każdej z nich:

```
my $sth = $dbh->prepare (qq{
    INSERT INTO member last_name, first_name VALUES(?,?,?)
});
$sth->execute ("Adams","Bill","2014-07-19");
```

- Aby wskazać NULL jako wartość miejsca zarezerwowanego, użyj `undef`.
- Miejsca zarezerwowane i metoda `quote()` są przeznaczone jedynie dla wartości danych. Nie próbuj użyć miejsca zarezerwowanego dla słów kluczowych, takich jak `SELECT`, lub identyfikatorów, na przykład nazwy bazy danych, tabeli lub kolumny. Takie rozwiązanie nie działa, ponieważ słowo kluczowe lub identyfikator zostaną ujęte w znaki cytowania przed wstawieniem do zapytania, a więc wykonanie zapytania zakończy się niepowodzeniem z powodu błędu składni.

W przypadku pewnych serwerów bazy danych użycie miejsc zarezerwowanych wiąże się z jeszcze jedną korzyścią poza poprawioną wydajnością działania pętli. Niektóre serwery buforują zapytania preinterpretowane i prawdopodobnie także plan wykonania zapytania. W ten sposób, jeśli zapytanie jest później przekazane serwerowi, to może on je ponownie wykorzystać i przetworzyć znacznie szybciej bez obciążenia zawiązanego z początkowym przygotowaniem zapytania. Buforowanie zapytań jest szczególnie użyteczne w przypadku skomplikowanych zapytań `SELECT`, ponieważ ich przygotowanie i wygenerowanie dobrego planu wykonania może zabrać nieco czasu. Miejsca zarezerwowane dają większą szansę na umożliwienie buforowania zapytania, ponieważ stają się ono bardziej ogólne niż zapytania skonstruowane przez osadzenie określonych wartości danych bezpośrednio w ciągu tekstowym zapytania.

MySQL nie buforuje planu wykonania zapytania. Serwer MySQL ma bufor zapytania, ale działa, buforując zbiory wynikowe dla ciągów tekstowych zapytań, a nie planów wykonania. Więcej informacji na ten temat znajdziesz w punkcie 12.7.3, zatytułowanym „Używanie bufora zapytań”.

Domyślnie, MySQL nie buforuje również zapytań preinterpretowanych. Dołączanie parametrów do miejsc zarezerwowanych następuje po stronie klienta w module DBD::mysql. Jednak protokół binarny zaimplementowany w utworzonej w języku C bibliotece klienta pozwala na przygotowanie zapytania po stronie serwera i zlecenie serwerowi obsługi dołączania parametrów.

DBD::mysql może wykorzystać tę możliwość. Aby włączyć przygotowywanie zapytań i dołączanie parametrów po stronie serwera, trzeba jedynie włączyć opcję `mysql_server_prepare`. Na przykład, mając uchwyt bazy danych `$dbh`, można to zrobić w następujący sposób:

```
$dbh->{mysql_server_prepare} = 1;
```

Wyłączenie przygotowywania zapytań po stronie serwera następuje po przypisaniu wartości 0 wymienionej opcji.

Nawet jeśli nie używasz możliwości MySQL w zakresie przygotowywania zapytań po stronie serwera, stosowanie miejsc zarezerwowanych nadal może przynosić pewne korzyści. Kiedy przystosowujesz skrypt do użycia w innym systemie bazy danych zapewniającym buforowanie planu wykonywania, wykonanie zapytań z miejscami zarezerwowanymi będzie znacznie efektywniejsze niż bez nich.

Tajemnicze wartości undef

Pewne metody DBI, takie jak `do()` i `selectrow_array()`, wykonujące ciąg tekstowy zapytania, pozwalają na dostarczenie wartości miejsc zarezerwowanych dołączanych do znaków zapytania umieszczonych w zapytaniu. Na przykład, uaktualnienie rekordu można przeprowadzić w następujący sposób:

```
my $rows = $dbh->do (
    "UPDATE member SET expiration = ? WHERE member_id = ?",
    undef, "2007-01-01", 14);
```

Natomiast pobranie rekordu można przeprowadzić następująco:

```
my $ref = $dbh->selectrow_arrayref (
    "SELECT * FROM member WHERE member_id = ?",
    undef, 14);
```

Zwróć uwagę, że w obu przypadkach wartości miejsca zarezerwowanego są poprzedzane tajemniczym argumentem `undef`, który — jak się wydaje — nic nie wnosi do zapytania. Powód jego umieszczenia jest następujący: dla metod wykonujących zapytania i zapewniających obsługę argumentów miejsc zarezerwowanych wspomniane argumenty są poprzedzone innym argumentem, który może być użyty do wskazania atrybutów przetwarzania zapytania. Tego rodzaju atrybuty są niezwykle rzadko (o ile w ogóle) używane, ale mimo tego sam argument nadal musi być obecny, stąd przypisanie mu wartości `undef`.

8.2.8. Dołączanie wyniku zapytania do zmiennych skryptu

Miejsca zarezerwowane pozwalają na zastąpienie wartości w ciągu tekstowym zapytania w trakcie wykonywania zapytania. Innymi słowy, istnieje możliwość sparowania „danych wejściowych” zapytania. DBI zapewnia odpowiadającą temu operację danych wyjściowych, nazywaną „dołączanie parametrów”, która pozwala na sparowanie „danych wyjściowych”. Odbывается to przez automatyczne umieszczenie wartości kolumn w zmiennych podczas pobierania rekordu i bez konieczności wyraźnego przypisywania wspomnianych wartości zmiennym.

Przyjmujemy założenie, że masz zapytanie pobierające dane członków Ligi z tabeli member. DBI możesz nakazać przypisanie wartości wskazanych kolumn do zmiennych Perla. Podczas pobierania rekordu zmienne są automatycznie uaktualniane odpowiednimi wartościami kolumn, co powoduje, że operacja pobierania danych jest niezwykle efektywna. Poniżej przedstawiono przykład pokazujący, jak dołączyć kolumny do zmiennych i uzyskać do nich dostęp w pętli pobierającej dane:

```
my ($last_name, $first_name, $suffix);
my $sth = $dbh->prepare (qq{
    SELECT last_name, first_name, suffix
    FROM member ORDER BY last_name, first_name
});
$sth->execute ();
$sth->bind_col (1, \$last_name);
$sth->bind_col (2, \$first_name);
$sth->bind_col (3, \$suffix);
print "$last_name, $first_name, $suffix\n" while $sth->fetch ();
```

Metodę `bind_col()` należy wywołać po metodzie `execute()`, ale przed pobraniem rekordu. Każde wywołanie powinno podawać numer kolumny oraz odniesienie do zmiennej powiązanej z tą kolumną. Numery kolumn zaczynają się od 1.

Alternatywnym rozwiązaniem dla poszczególnych wywołań metody `bind_col()` jest przekazanie wszystkich odniesień do zmiennych w pojedynczym wywołaniu metody `bind_columns()`:

```
my ($last_name, $first_name, $suffix);
my $sth = $dbh->prepare (qq{
    SELECT last_name, first_name, suffix
    FROM member ORDER BY last_name, first_name
});
$sth->execute ();
$sth->bind_columns (\$last_name, \$first_name, \$suffix);
print "$last_name, $first_name, $suffix\n" while $sth->fetch ();
```

Metodę `bind_columns()` należy wywołać po metodzie `execute()`, ale przed pobraniem rekordów.

8.2.9. Określenie parametrów połączenia

Najbardziej bezpośrednim sposobem nawiązania połączenia z serwerem jest podanie wszystkich parametrów połączenia jako argumentów metody `connect()`:

```
my $dsn = "DBI:mysql:nazwa_bazy_danych:nazwa_komputera";  
my $dbh = DBI->connect ($dsn, nazwa_uzytkownika, haslo);
```

Jeżeli pominiesz parametry połączenia, to DBI spróbuje ustalić ich wartości w następujący sposób:

- Zmienna środowiskowa `DBI_DSN` jest używana, gdy została zdefiniowana, a nazwa źródła danych (DSN) nie została zdefiniowana lub jest pustym ciągiem tekstowym. Zmienne środowiskowe `DBI_USER` i `DBI_PASS` są używane, o ile zostały zdefiniowane, a nazwa użytkownika i hasło nie zostały zdefiniowane (ale nie jeśli są pustymi ciągami tekstowymi). W systemie Windows, jeśli nazwa użytkownika nie została zdefiniowana, to używana jest zmienna `USER`.
- Jeżeli pominiesz nazwę komputera, DBI spróbuje nawiązać połączenie z komputerem lokalnym.
- Jeśli dla nazwy użytkownika podasz `undef` lub pusty ciąg tekstowy, domyślnie zostanie użyta nazwa logowania (UNIX) lub `ODBC` (Windows).
- Jeżeli dla hasła podasz `undef` lub pusty ciąg tekstowy, wtedy hasło nie będzie wysłane do serwera.

Pewne opcje w DSN można określić przez ich dołączenie w początkowej części ciągu tekstowego, poprzedzając każdą średnikiem. Na przykład, opcja `mysql_default_file` pozwala na wskazanie ścieżki dostępu do pliku opcji:

```
my $dsn = "DBI:mysql:sampdb;mysql_read_default_file=/home/paweł/.my.cnf";
```

W trakcie wykonania skryptu nastąpi odczyt wskazanego pliku i pobranie z niego parametrów połączenia. Przyjmujemy założenie, że plik `/home/paweł/.my.cnf` ma następującą zawartość:

```
[client]  
host=localhost  
user=sampadm  
password=secret
```

W takim przypadku metoda `connect()` próbuje nawiązać połączenie z serwerem MySQL w komputerze lokalnym (`localhost`) na konto użytkownika `sampdb` o hasle `secret`. W systemie UNIX można nakazać skryptowi użycie pliku opcji należącego do użytkownika uruchamiającego skrypt. W tym celu trzeba w następujący sposób sparacetyzować nazwę pliku opcji:

```
my $dsn = "DBI:mysql:sampdb;mysql_read_default_file=$ENV{HOME}/.my.cnf";
```

Zmienna `$ENV{HOME}` zawiera ścieżkę dostępu do katalogu domowego użytkownika uruchamiającego skrypt i dlatego też używane parametry połączenia zostaną pobrane

z pliku opcji danego użytkownika. Dzięki utworzeniu skryptu w pokazany sposób parametrów połączenia nie trzeba osadzać bezpośrednio w skrypcie.

Użycie opcji `mysql_read_default_file` powoduje, że skrypt odczytuje jedynie wskazany plik opcji. Takie rozwiązanie może być niepożądane, jeśli chcesz wykorzystać parametry w systemowych plikach opcji (na przykład `/etc/my.cnf` w systemie UNIX lub `C:\my.ini` w Windows). Aby skrypt odczytywał wszystkie standardowe pliki opcji w poszukiwaniu parametrów połączenia, należy użyć opcji `mysql_read_default_group`. Wymieniona opcja powoduje użycie parametrów w grupie `[client]`, a także w grupie wskazanej jako wartość opcji. Na przykład, jeśli przygotowałeś opcje charakterystyczne dla skryptów powiązanych z `sampdb`, to możesz je umieścić w grupie `[sampdb]`, a następnie w poniższy sposób podać wartość źródła danych:

```
my $dsn = "DBI:mysql:sampdb:mysql_read_default_group=sampdb";
```

Aby odczytać jedynie grupę `[client]` ze standardowych plików opcji, należy użyć następującej wartości źródła danych:

```
my $dsn = "DBI:mysql:sampdb:mysql_read_default_group=client";
```

Więcej informacji odnośnie formatu plików opcji MySQL przedstawiono w dodatku F, zatytułowanym „Przewodnik po programie MySQL”.

Jedyna trudność podczas używania opcji `mysql_read_default_file` w Windows polega na tym, że ścieżki dostępu najczęściej rozpoczynają się od litery i dwukropka. To jest problem, ponieważ DBI interpretuje dwukropek jako znak oddzielający poszczególne części ciągu tekstowego DBI. Istnieje rozwiązanie tego problemu, ale trudno określić je mianem eleganckiego:

1. Użyj `chdir()` w celu przejścia do katalogu głównego dysku, w którym znajduje się plik opcji. W ten sposób ścieżki dostępu bez podanej litery będą interpretowane względem bieżącego dysku.
2. Podaj nazwę pliku jako wartości opcji `mysql_read_default_file` w DSN. Pomiń literę dysku i dwukropek.
3. Jeżeli katalog bieżący powinien pozostać nietknięty przez operację połączenia, przed wywołaniem metody `connect()` zapisz ścieżkę dostępu do katalogu bieżącego, a następnie przejdź do niego za pomocą `chdir()`.

Poniższy fragment kodu pokazuje, jak zastosować powyższe rozwiązanie, jeśli chcesz użyć pliku opcji `C:\my.ini`. (Zwróć uwagę, że ukośniki w ścieżkach dostępu Windows są podawane jak ukośniki w ciągach tekstowych Perl).

```
# Zapisanie ścieżki dostępu do katalogu bieżącego.
use Cwd;
my $orig_dir = cwd ();
# Przejście do katalogu głównego dysku, na którym znajduje się plik.
chdir ("C:/") or die "Nie można zmienić katalogu: $!\n";
# Nawiązanie połączenia z użyciem parametrów zdefiniowanych w pliku C:\my.ini.
my $dsn = "DBI:mysql:sampdb:localhost:mysql_read_default_file=/my.ini";
my %conn_attrs = (RaiseError => 1, PrintError => 0, AutoCommit => 1);
```

```
my $dbh = DBI->connect ($dsn, undef, undef, \%conn_attrs);
# Powrót do poprzedniego katalogu.
chdir ($orig_dir) or die "Nie można zmienić katalogu: $!\n";
```

Użycie pliku opcji nie chroni przed podawaniem parametrów połączenia w wywołaniu metody `connect()`, na przykład jeśli skrypt ma nawiązać połączenie jako określony użytkownik. Wszystkie wyraźnie podane w wywołaniu metody `connect()` informacje, takie jak nazwa komputera, nazwa użytkownika i hasło, nadpisują parametry połączenia pobrane z plików opcji. Na przykład, skrypt może przetwarzać opcje takie jak `--host` i `--user` podane w wierszu poleceń i używać ich wartości zamiast znalezionych w plikach opcji. Takie rozwiązanie jest użyteczne, ponieważ w taki właśnie sposób zachowują się standardowe klienty MySQL. Dzięki przedstawionemu rozwiązaniu Twoje skrypty DBI będą spójne z wspomnianym zachowaniem.

W pozostałych skryptach wiersza poleceń opracowanych w tym rozdziale będziemy używać pewnej standardowej konfiguracji połączenia i odpowiedzialnego za to kodu. Poniżej przedstawiono ten kod; później nie będziemy go omawiać, a koncentrować się na głównych zadaniach tworzonych skryptów:

```
#!/usr/bin/perl

use strict;
use warnings;
use DBI;

# Przetworzenie parametrów z wiersza poleceń, o ile zostały podane.
use Getopt::Long;
$Getopt::Long::ignorecase = 0; # Wielkość liter w opcjach ma znaczenie.
$Getopt::Long::bundling = 1;   # -uname = -u name, a nie -u -n -a -m -e

# Parametry domyślne, wszystkie początkowo niezdefiniowane.
my ($host_name, $password, $port_num, $socket_name, $user_name);

GetOptions (
    # =i oznacza, że po opcji trzeba podać wartość w postaci liczby całkowitej.
    # =s oznacza, że po opcji trzeba podać wartość w postaci ciągu tekstowego.
    "host|h=s"      => \$host_name,
    "password|p=s"  => \$password,
    "port|P=i"      => \$port_num,
    "socket|S=s"    => \$socket_name,
    "user|u=s"      => \$user_name
) or exit (1);

# Utworzenie źródła danych.
my $dsn = "DBI:mysql:sampdb";
$dsn .= ";host=$host_name" if $host_name;
$dsn .= ";port=$port_num" if $port_num;
$dsn .= ";mysql_socket=$socket_name" if $socket_name;
$dsn .= ";mysql_read_default_group=client";

# Nawiązanie połączenia z serwerem.
my %conn_attrs = (RaiseError => 1, PrintError => 0, AutoCommit => 1);
my $dbh = DBI->connect ($dsn, $user_name, $password, \%conn_attrs);
```

Przedstawiony powyżej kod inicjalizuje DBI, sprawdza, czy w wierszu poleceń podano parametry połączenia, a następnie nawiązuje połączenie z serwerem MySQL za pomocą parametrów połączenia z wiersza poleceń lub znalezionych w grupie [client] standardowych plików opcji. Jeśli parametry połączenia są zdefiniowane w pliku opcji użytkownika, nie trzeba ich wpisywać podczas uruchamiania skryptu wykorzystującego powyższy kod.

Ostania część każdego skryptu będzie podobna i po prostu zamyka połączenie przed zakończeniem działania skryptu:

```
$dbh->disconnect ();
```

Kiedy przejdziemy do programowania sieciowego w podrozdziale 8.4, zatytułowanym „Użycie DBI w aplikacjach sieciowych”, wtedy nieco zmodyfikujemy kod konfigurujący połączenie, ale podstawowa idea pozostanie bez zmian.

Istnieje jedna różnica między sposobem, w jaki standardowe klienty MySQL i moduł Getopt obsługują opcje wiersza poleceń. Standardowe klienty mają specjalny kod przetwarzania opcji, który pozwala na podanie opcji hasła (--password lub -p) wraz z samym hasłem lub bez niego; w przypadku braku hasła zostanie wyświetlony komunikat z prośbą o podanie hasła.

W przypadku modułu Getopt, jeśli spróbujesz określić wartość hasła (--password i -p) jako opcjonalną, wtedy nie możesz jednoznacznie wskazać użycia opcji bez wartości, o ile dana opcja nie będzie ostatnim argumentem w wierszu poleceń lub nie znajdzie się natychmiast po innej opcji. Przyjmujemy założenie, że masz skrypt oczekujący podania argumentu w postaci nazwy tabeli tuż po opcji. Jeżeli skrypt zostanie wywołany w poniższy sposób, moduł Getopt zinterpretuje mytbl jako wartość hasła, zamiast wyświetlić pytanie z prośbą o podanie hasła:

```
% ./myscript.pl -u paweł -p mytbl
```

Aby uniknąć tego rodzaju problemów, przedstawiony powyżej kod w Perl wymaga, że w przypadku użycia opcji hasła konieczne jest podanie również hasła.

8.2.10. Usuwanie błędów

W celu usunięcia błędów z nieprawidłowo działającego skryptu DBI najczęściej stosowane są dwie techniki, razem lub każda oddzielnie. Pierwsza polega na umieszczeniu w skrypcie poleceń print. W ten sposób można dopasować do własnych potrzeb wyświetlanie danych wyjściowych procesu usuwania błędów, choć wymienione polecenia trzeba dodawać ręcznie. Druga polega na wykorzystaniu wbudowanych w DBI możliwości w zakresie śledzenia. To bardziej ogólne i systematyczne podejście, a ponadto realizowane automatycznie po jego włączeniu. Śledzenie DBI wyświetla także informacje o działaniu sterownika; tych informacji nie uzyskasz w inny sposób.

8.2.10.1. Usuwanie błędów z użyciem poleceń print

Oto najczęściej pojawiające się pytanie: „Mam zapytanie działające doskonale w programie mysql, ale niedziałające z poziomu skryptu DBI. Dlaczego?”. Nierzadko zdarza się, że skrypt

DBI wykonuje inne zapytanie, niż sądzisz. Jeżeli wyświetlisz je tuż przed wykonaniem, to możesz być zaskoczony postacią zapytania wysyłanego do serwera. Przyjmujemy założenie, że zapytanie wprowadzane w programie `mysql` ma następującą postać:

```
mysql> INSERT INTO member (last_name,first_name,expiration)
-> VALUES('Brown','Marcia','2012-06-03');
```

Następnie to samo zapytanie umieszczamy w skrypcie DBI (pomijając oczywiście średnik na końcu):

```
$last = "Brown";
$first = "Marcia";
$expiration = "2012-06-03";
$stmt = qq{
    INSERT INTO member (last_name,first_name,expiration)
    VALUES($last,$first,$expiration)
};
$rows = $dbh->do ($stmt);
```

Powyższe zapytanie w skrypcie DBI nie działa, pomimo że jest takie samo jak wykonane w programie `mysql`. Czy na pewno jest takie samo? Spróbujmy je wyświetlić:

```
print "$stmt\n";
```

Oto wyświetlone zapytanie:

```
INSERT INTO member (last_name,first_name,expiration)
VALUES(Brown,Marcia,2012-06-03)
```

Na podstawie powyższych danych wyjściowych można stwierdzić, że to zapytanie nie jest takie samo jak wykonane w programie `mysql`. Brakuje znaków cytowania wokół wartości na liście `VALUES()`. Jednym ze sposobów prawidłowego zdefiniowania tego rodzaju zapytania jest użycie metody `quote()`:

```
$last = $dbh->quote ("Brown");
$first = $dbh->quote ("Marcia");
$expiration = $dbh->quote ("2012-06-03");
$stmt = qq{
    INSERT INTO member (last_name,first_name,expiration)
    VALUES($last,$first,$expiration)
};
$rows = $dbh->do ($stmt);
```

Alternatywne rozwiązanie polega na użyciu miejsc zarezerwowanych i przekazaniu wartości do wstawienia jako argumentów metody `do()`:

```
$last = "Brown";
$first = "Marcia";
$expiration = "2012-06-03";
$stmt = qq{
    INSERT INTO member (last_name,first_name,expiration)
    VALUES(?,?,?)
};
$rows = $dbh->do ($stmt, undef, $last, $first, $expiration);
```

Niestety, w przypadku zastosowania tego drugiego podejścia nie można wyświetlić pełnego zapytania, ponieważ wartości miejsc zarezerwowanych są wstawiane dopiero w trakcie wywołania metody `do()`. Kiedy używasz miejsc zarezerwowanych, śledzenie jest bardziej pomocną metodą usuwania błędów ze skryptów.

8.2.10.2. Usuwanie błędów z użyciem śledzenia

DBI oferuje mechanizm generujący informacje procesu usuwania błędów, które pomagają w ustaleniu przyczyn nieprawidłowego działania skryptów. Poziom śledzenia jest z zakresu od 0 (brak) do 15 (maksymalna ilość informacji). Ogólnie rzecz biorąc, poziomy od 1 do 4 są najbardziej użyteczne. Na przykład, poziom 2 śledzenia wyświetla tekst wykonywanego zapytania (łącznie z wynikiem zastąpienia miejsc zarezerwowanych), wynik wywołania metody `quote()` itd. To może ogromnie pomóc w znalezieniu przyczyny problemu.

Za pomocą metody `trace()` można określić poziom śledzenia w poszczególnych skryptach. Ewentualnie, ustawienie zmiennej środowiskowej `DBI_TRACE` określa poziom śledzenia we wszystkich uruchamianych skryptach DBI.

Aby użyć metody `trace()`, należy przekazać jej argument w postaci poziomu śledzenia oraz opcjonalnie nazwę pliku. Jeśli nie podasz nazwy pliku, wszystkie dane wyjściowe mechanizmu śledzenia będą kierowane do `STDERR`. W przeciwnym razie zostaną zapisane we wskazanym pliku. Poniższe wywołanie włącza pierwszy poziom śledzenia, a wszystkie komunikaty będą kierowane do `STDERR`.

```
DBI->trace (1);
```

Z kolei poniższe wywołanie włącza drugi poziom śledzenia, a komunikaty zostaną zapisane w pliku o nazwie `trace.out`:

```
DBI->trace (2, "trace.out");
```

W celu wyłączenia śledzenia trzeba ustawić mu poziom zero:

```
DBI->trace (0);
```

Dzięki `DBI->trace()` wszystkie operacje DBI będą śledzone. W celu zapewnienia sobie większej kontroli nad śledzeniem włącz je na określonym poziomie. To użyteczne rozwiązanie, jeśli masz dobre podejrzenie, w którym miejscu skryptu może znajdować się błąd, i nie chcesz śledzić wszystkiego. Na przykład, jeżeli masz problem z określonym zapytaniem `SELECT`, to włącz śledzenie uchwytu zapytania powiązanego z danym zapytaniem:

```
$sth = $dbh->prepare (qq{ SELECT ... }); # Utworzenie uchwytu zapytania.
$sth->trace (1);                        # Włączenie śledzenia dla danego zapytania.
$sth->execute ();
```

Jeżeli podasz nazwę pliku dla dowolnego wywołania metody `trace()`, niezależnie, czy dla DBI jako całości, czy dla konkretnego uchwytu, wtedy wszystkie dane wyjściowe mechanizmu śledzenia zostaną umieszczone we wskazanym pliku.

Atrybut `TraceLevel` jest alternatywą dla metody `trace()`. Wymieniony atrybut pozwala na ustawienie lub sprawdzenie poziomu śledzenia dla danego uchwytu:


```
$dbh->{TraceLevel} = 3;           # Ustawienie poziomu śledzenia dla uchwytu bazy danych.
my $cur_level = $sth->{TraceLevel}; # Sprawdzenie poziomu śledzenia dla uchwytu zapytania.
```

W celu globalnego włączenia śledzenia, aby było stosowane we wszystkich uruchamianych skryptach DBI, należy w powłoce ustawić zmienną środowiskową DBI_TRACE. Składnia zależy od używanej powłoki:

- W przypadku powłoki csh lub tcsh:


```
% setenv DBI_TRACE wartość
```
- W przypadku powłoki sh, bash lub ksh:


```
$ export DBI_TRACE=wartość
```
- W systemie Windows:


```
C:\> set DBI_TRACE=wartość
```

Format *wartości* jest taki sam dla wszystkich powłok: liczba *n* włącza śledzenie na poziomie *n* i kieruje dane wyjściowe do STDERR, nazwa pliku włącza śledzenie na poziomie 2 i kieruje dane wyjściowe do wymienionego pliku, natomiast *n=nazwa_pliku* powoduje włączenie śledzenia na poziomie *n* i skierowanie danych wyjściowych do wymienionego pliku. Poniżej przedstawiono kilka przykładów z użyciem składni tcsh:

- Śledzenie na poziomie 1 i przekierowanie danych wyjściowych do STDERR:


```
% setenv DBI_TRACE 1
```
- Śledzenie na poziomie 1 i przekierowanie danych wyjściowych do pliku *trace.out*:


```
% setenv DBI_TRACE 1=trace.out
```
- Śledzenie na poziomie 2 i przekierowanie danych wyjściowych do pliku *trace.out*:


```
% setenv DBI_TRACE trace.out
```

Użycie zmiennej środowiskowej DBI_TRACE ma zaletę w postaci włączenia śledzenia skryptów DBI bez konieczności wprowadzania jakichkolwiek zmian w skryptach. Jeśli z poziomu powłoki włączysz śledzenie i zapis jego wyników w pliku, to upewnij się o wyłączeniu śledzenia po znalezieniu błędu. Dane wyjściowe śledzenia są dodawane do pliku bez nadpisywania jego zawartości, więc plik może osiągnąć ogromne rozmiary, jeśli nie zachowasz ostrożności. Szczególnie złym pomysłem jest zdefiniowanie zmiennej środowiskowej w pliku startowym powłoki, takim jak *.cshrc*, *.tcshrc*, *.login* lub *.profile*!

W celu wyłączenia DBI_TRACE dla różnych interpreterów powłoki użyj odpowiedniego polecenia z wymienionych poniżej:

- W przypadku powłoki csh lub tcsh:


```
% setenv DBI_TRACE 0
% unsetenv DBI_TRACE
```
- W przypadku powłoki sh, bash lub ksh:


```
$ unset DBI_TRACE
$ export DBI_TRACE=0
```
- W systemie Windows:


```
C:\> unset DBI_TRACE
C:\> set DBI_TRACE=0
```

8.2.11. Używanie metadanych zbioru wynikowego

DBI zapewnia dostęp do metadanych zbioru wynikowego, czyli informacji opisujących rekordy wybrane przez zapytanie. Aby uzyskać wspomniane informacje, trzeba wcześniej uzyskać dostęp do atrybutów uchwytu zapytania powiązanego z zapytaniem, które wygenerowało dany zbiór wynikowy. Niektóre z nich są standardowymi atrybutami DBI dostępnymi we wszystkich sterownikach baz danych (przykładem tego rodzaju atrybutu jest `NUM_OF_FIELDS`, wskazujący liczbę kolumn w zbiorze wynikowym). Z kolei inne są charakterystyczne dla MySQL i dostarczane przez `DBD::mysql`, czyli sterownik MySQL dla DBI. Wspomniane atrybuty (na przykład `mysql_max_length`, podający maksymalną długość wartości w każdej kolumnie) nie są akceptowane w innych systemach baz danych. Im większa w skrypcie liczba użytych atrybutów charakterystycznych dla MySQL, tym większe niebezpieczeństwo, że skryptu nie będzie można przenieść do innych baz danych. Z drugiej strony, łatwiej będzie pobrać żądane informacje.

Metadane trzeba pobrać w odpowiednim momencie. Ogólnie rzecz biorąc, atrybuty zbioru wynikowego są niedostępne dla zapytania `SELECT` aż do chwili wywołania metod `prepare()` i `execute()`. Ponadto, atrybuty mogą być nieprawidłowe po dotarciu do końca zbioru wynikowego za pomocą funkcji pobierającej rekordy lub po wywołaniu metody `finish()`.

Przedstawiony poniżej przykład pokazuje, jak używać jednego z charakterystycznych dla MySQL atrybutów metadanych (`mysql_max_length`) w połączeniu z ogólnymi atrybutami `NUM_OF_FIELDS` (podaje liczbę kolumn w zbiorze wynikowym) i `NAME` (przechowuje nazwy kolumn znajdujących się w zbiorze wynikowym). Informacje dostarczane przez wymienione atrybuty można połączyć w celu utworzenia skryptu *tabular.pl*, wyświetlającego dane wyjściowe zapytań `SELECT` w takiej samej postaci tabeli, jak w przypadku ich wykonania przez program klienta `mysql` działający w trybie interaktywnym. Poniżej przedstawiono kod skryptu *tabular.pl*. Zapytanie `SELECT` można zastąpić innym, procedury odpowiedzialne za wyświetlanie danych wyjściowych są niezależne od zapytań.

```
my $sth = $dbh->prepare (qq{
    SELECT last_name, first_name, suffix, city, state
    FROM president ORDER BY last_name, first_name
});
$sth->execute (); # Atrybuty powinny być dostępne po tym wywołaniu.

# Rzeczywista maksymalna długość kolumny w zbiorze wynikowym.
my @wid = @{$sth->{mysql_max_length}};
# Liczba kolumn w zbiorze wynikowym.
my $ncols = $sth->{NUM_OF_FIELDS};

# Dostosowanie długości kolumn, jeśli wartości są krótsze niż nagłówki kolumn
# lub niż słowo "NULL" dla kolumn, które akceptują wartości NULL.
for (my $i = 0; $i < $ncols; $i++)
{
    my $name_wid = length ($sth->{NAME}->[$i]);
    $wid[$i] = $name_wid if $wid[$i] < $name_wid;
    $wid[$i] = 4 if $sth->{NULLABLE}->[$i] && $wid[$i] < 4;
}
```

```
# Wyświetlenie danych wyjściowych w formacie tabeli.
print_dashes (\@wid, $ncols);      # Wiersz obramowania tabeli.
print_row ($sth->{NAME}, \@wid, $ncols); # Nagłówki kolumn.
print_dashes (\@wid, $ncols);      # Wiersz obramowania tabeli.
while (my $ary_ref = $sth->fetchrow_arrayref ())
{
    print_row ($ary_ref, \@wid, $ncols);    # Dane rekordu.
}
print_dashes (\@wid, $ncols);            # Wiersz obramowania tabeli.
```

Po zainicjowaniu zapytania za pomocą metody `execute()` można pobrać żądane metadane. Zmienna `$sth->{NUM_OF_FIELDS}` przechowuje wartość skalarną oznaczającą liczbę kolumn w zbiorze wynikowym. Z kolei `$sth->{NAME }` i `$sth->{mysql_max_length}` dostarczają informacje o nazwach kolumn i maksymalnej długości wartości poszczególnych kolumn. Wartości obu wymienionych atrybutów są odniesieniami do tablicy zawierającej element dla każdej kolumny zbioru wynikowego, w kolejności, w jakiej kolumny zostały wymienione w zapytaniu.

Pozostałe obliczenia są bardzo podobne do przeprowadzonych w programie `exec_stmt` opracowanym w rozdziale 7., zatytułowanym „Tworzenie programów MySQL przy użyciu języka C”. Na przykład, w celu uniknięcia błędnego sformatowania danych wyjściowych szerokość kolumn zwiększamy, gdy nazwa kolumny jest dłuższa niż jakiegokolwiek znajdujące się w niej dane.

Metody `print_dashes()` i `print_row()` są podobne do odpowiadającego im kodu w programie `exec_stmt`:

```
sub print_dashes
{
    my $wid_ary_ref = shift; # Odniesienie do tablicy zawierającej informacje o szerokości kolumn.
    my $cols = shift;       # Liczba kolumn.

    for (my $i = 0; $i < $cols; $i++)
    {
        print "+", "-" x ($wid_ary_ref->[$i]+2);
    }
    print "+\n";
}

# Wyświetlenie rekordu danych (kolumny wartości liczbowych nie zostają wyrównane do prawej strony).
sub print_row
{
    my $val_ary_ref = shift; # Odniesienie do tablicy zawierającej wartości kolumn.
    my $wid_ary_ref = shift; # Odniesienie do tablicy zawierającej informacje o szerokości kolumn.
    my $cols = shift;       # Liczba kolumn.

    for (my $i = 0; $i < $cols; $i++)
    {
        printf "| %-*s ", $wid_ary_ref->[$i],
            defined ($val_ary_ref->[$i]) ? $val_ary_ref->[$i] : "NULL";
    }
    print "|\n";
}
```

Dane wyjściowe wygenerowane przez skrypt *tabular.pl* przedstawiają się następująco:

The output from *tabular.pl* looks like this:

```
+-----+-----+-----+-----+-----+
| last_name | first_name | suffix | city | state |
+-----+-----+-----+-----+-----+
| Adams | John | NULL | Braintree | MA |
| Adams | John Quincy | NULL | Braintree | MA |
| Arthur | Chester A. | NULL | Fairfield | VT |
| Buchanan | James | NULL | Mercersburg | PA |
| Bush | George H.W. | NULL | Milton | MA |
| Bush | George W. | NULL | New Haven | CT |
| Carter | James E. | Jr. | Plains | GA |
...

```

Nasz kolejny skrypt używa metadanych kolumn do wygenerowania danych wyjściowych w innym formacie. Ten skrypt, o nazwie *show_member.pl*, pozwala na przeglądanie listy członków Ligi Historycznej bez konieczności wykonywania jakichkolwiek dodatkowych zapytań. Po podaniu nazwiska członka Ligi skrypt wyświetla informacje o nim w następującej postaci:

```
% ./show_member.pl artel
member_id: 63
last_name: Artel
first_name: Mike
suffix:
expiration: 2016-04-16
email: mike_artel@venus.org
street: 4264 Lovering Rd.
city: Miami
state: FL
zip: 12777
phone: 075-961-0712
interests: Civil Rights,Education,Revolutionary War

```

Skrypt *show_member.pl* można również wywołać wraz z argumentem w postaci identyfikatora członka Ligi lub wzorcem SQL pozwalającym na dopasowanie wielu nazwisk. Poniższe polecenia pokazują uruchomienie skryptu w celu wyświetlenia informacji o członku Ligi o identyfikatorze 23 oraz o członkach Ligi, których nazwisko rozpoczyna się na literę C:

```
% ./show_member.pl 23
% ./show_member.pl C%

```

Poniżej przedstawiono kod skryptu *show_member.pl*. Używa atrybutu NAME w celu określenia etykiet dla każdego rekordu danych wyjściowych oraz NUM_OF_FIELDS do ustalenia liczby kolumn znajdujących się w zbiorze wynikowym:

```
my $count = 0; # Liczba wyświetlonych dotąd wierszy.
my @label = (); # Tablica etykiet kolumn.
my $label_wid = 0;

while (@ARGV) # Wykonanie zapytania dla każdego argumentu wiersza poleceń.
{
    my $arg = shift (@ARGV);

```

```

# Domyślnym wzorcem jest wyszukiwanie po nazwisku...
my $clause = "last_name LIKE " . $dbh->quote ($arg);
# ...ale po podaniu argumentu liczbowego wyszukiwanie odbywa się po identyfikatorze.
$clause = "member_id = " . $dbh->quote ($arg) if $arg =~ /\^d+$/;

# Wykonanie zapytania.
my $sth = $dbh->prepare (qq{
    SELECT * FROM member
    WHERE $clause
    ORDER BY last_name, first_name
});
$sth->execute ();

# Pobranie nazw kolumn używanych jako etykiety oraz określenie
# maksymalnej długości nazwy kolumny w celu jej sformatowania
# (ta operacja jest przeprowadzana tylko w trakcie pierwszej iteracji pętli).
if ($label_wid == 0)
{
    @label = @{$sth->{NAME}};
    foreach my $label (@label)
    {
        $label_wid = length ($label) if $label_wid < length ($label);
    }
}

# Odczyt i wyświetlenie wyników zapytania.
my $matches = 0;
while (my @ary = $sth->fetchrow_array ())
{
    # Wstawienie znaku nowego wiersza przed drugim i kolejnym wierszem danych wyjściowych.
    print "\n" if ++$count > 1;
    foreach (my $i = 0; $i < $sth->{NUM_OF_FIELDS}; $i++)
    {
        # Wyświetlenie etykiety.
        printf "%-*s", $label_wid+1, $label[$i] . ":";
        # Wyświetlenie wartości, o ile jest dostępna.
        print " ", $ary[$i] if defined ($ary[$i]);
        print "\n";
    }
    ++$matches;
}
print "\nNie znaleziono dopasowań dla \"$arg\"\n" if $matches == 0;
}

```

Przeznaczeniem skryptu `show_member.pl` jest wyświetlenie całej zawartości, niezależnie od składających się na nią pól. Dzięki użyciu zapytania `SELECT *` do pobrania wszystkich kolumn i atrybutu `NAME` do ich określenia, skrypt działa bez modyfikacji, nawet jeśli kolumny zostaną dodane do tabeli `member` lub z niej usunięte.

Jeżeli po prostu chcesz się dowiedzieć, jakie kolumny zawiera tabela, ale bez pobierania jakichkolwiek rekordów, wtedy możesz wykonać poniższe zapytanie:

```
SELECT * FROM tbl_name WHERE FALSE
```

Klauzula `WHERE FALSE` przyjmuje wartość `false` dla wszystkich rekordów, a więc wykonanie powyższego zapytania powoduje wygenerowanie metadanych kolumn, ale nie zwraca żadnych rekordów. Po wywołaniu metod `prepare()` i `execute()` w zwykły

sposób możesz otrzymać nazwy kolumn dzięki konstrukcji `@{$sth->{NAME}}`. Pamiętaj, że przedstawiona tutaj sztuczka użycia „pustego” zapytania działa w MySQL i nie jest przenośna, a także może nie działać w innych systemach baz danych.

Do Ciebie należy decyzja, czy chcesz zachować przenośność skryptów, unikając stosowania atrybutów charakterystycznych dla MySQL, czy wykorzystać ich możliwości kosztem przenośności skryptów.

8.2.12. Przeprowadzanie transakcji

Jednym ze sposobów przeprowadzenia transakcji w skrypcie DBI jest wykonywanie zapytań `SET autocommit`, `START TRANSACTION`, `COMMIT` i `ROLLBACK` (opis wymienionych zapytań znajdziesz w podrozdziale 2.12, zatytułowanym „Przeprowadzanie transakcji”). Jednak DBI zapewnia możliwość abstrakcji transakcji (w postaci metod i atrybutów DBI) automatycznie wykonujących zapytania SQL powiązane z transakcjami. Takie rozwiązanie zapewnia możliwość przeniesienia skryptu do innego systemu bazy danych obsługującego transakcje, podczas gdy zapytania SQL nie dają gwarancji zachowania przenośności.

Aby użyć oferowanego przez DBI mechanizmu transakcji, aplikacja musi stosować tabele obsługujące transakcje. Jeśli tak nie jest, to za pomocą zapytania `ALTER TABLE` zmień tabelę na odpowiedni typ. Na przykład, w celu zmiany tabeli o nazwie *nazwa_tabeli* na używającą silnika InnoDB należy wykonać poniższe zapytanie:

```
ALTER TABLE nazwa_tabeli ENGINE=InnoDB;
```

Przetwarzanie transakcyjne w DBI jest przeprowadzane wedle poniższej ogólnej procedury:

1. Wyłączenie (lub tymczasowe zawieszenie) trybu automatycznego zatwierdzania zapytań, aby zapytania SQL nie były zatwierdzane aż do chwili, gdy zdecydujesz się na taki krok.
2. Wykonanie zapytania będącego częścią transakcji. Należy to zrobić w bloku `eval` wykonywanym wraz z włączonym atrybutem `RaiseError` i wyłączonym `PrintError`, aby wszelkie błędy powodowały zakończenie wykonywania bloku bez wyświetlania jakichkolwiek komunikatów. Jeżeli wykonanie bloku zakończy się powodzeniem, wtedy ostatnią operacją powinno być wywołanie metody `commit()` i zatwierdzenie transakcji.
3. Po zakończeniu działania bloku `eval` należy sprawdzić jego kod wyjścia. Jeżeli wystąpił błąd, trzeba wywołać metodę `rollback()` w celu wycofania transakcji (i zgłoszenia błędu, jeśli to konieczne).
4. Ostatni krok to przywrócenie trybu automatycznego zatwierdzania zapytań i właściwego stanu atrybutu obsługi błędów, o ile zachodzi taka potrzeba.

Przedstawiony poniżej przykład implementuje omówione podejście. Został oparty na scenariuszu z rozdziału 2., zatytułowanego „Użycie MySQL do zarządzania danymi”, w którym pokazano sposób ręcznego wykonywania z poziomu klienta `mysql` zapytań

powiązanych z transakcjami. We wspomnianym scenariuszu odkryliśmy błędnie wpisane oceny uczniów w tabeli score i musieliśmy je zamienić: uczniowi o identyfikatorze 8 wstawiliśmy wynik 18, natomiast uczniowi o identyfikatorze 9 wynik 13. To był błąd, wyniki powinny być zamienione miejscami. Do rozwiązania problemu użyliśmy wówczas poniższych zapytań UPDATE:

```
UPDATE score SET score = 13 WHERE event_id = 5 AND student_id = 8;
UPDATE score SET score = 18 WHERE event_id = 5 AND student_id = 9;
```

Oba rekordy trzeba poprawić i wstawić im prawidłowe wartości, ale operacja uaktualnienia musi być przeprowadzona jako jedna całość. Przykład w rozdziale 2. pokazywał uaktualnienie rekordów za pomocą zapytań SQL: zmiana trybu automatycznego zatwierdzania, wprowadzenie zmian, zatwierdzenie i wycofanie. W przypadku skryptu Perl używającego mechanizmu transakcji oferowanego przez DBI całą operację można przeprowadzić następująco:

```
my $orig_re = $dbh->{RaiseError}; # Zachowanie stanu atrybutów obsługi błędów.
my $orig_pe = $dbh->{PrintError};
my $orig_ac = $dbh->{AutoCommit}; # Zachowanie stanu trybu automatycznego zatwierdzania.

$dbh->{RaiseError} = 1;          # Błąd powinien zgłosić wyjątek.
$dbh->{PrintError} = 0;          # Zawieszenie wyświetlania komunikatów błędów.
$dbh->{AutoCommit} = 0;          # Wyłączenie automatycznego zatwierdzania zapytań.

eval
{
    # Wykonanie zapytań, które zaliczają się do transakcji.
    my $sth = $dbh->prepare (qq{
        UPDATE score SET score = ?
        WHERE event_id = ? AND student_id = ?
    });
    $sth->execute (13, 5, 8);
    $sth->execute (18, 5, 9);
    $dbh->commit();               # Zatwierdzenie transakcji.
};
if ($?)                        # Czy transakcja zakończyła się niepowodzeniem?
{
    print "Wystąpił błąd w trakcie transakcji: $@\n";
    # Wycofanie transakcji, użycie eval do przechwycenia niepowodzenia wycofania.
    eval { $dbh->rollback (); }
}
$dbh->{AutoCommit} = $orig_ac; # Przywrócenie stanu trybu automatycznego zatwierdzania zapytań.
$dbh->{RaiseError} = $orig_re; # Przywrócenie stanu atrybutów obsługi błędów.
$dbh->{PrintError} = $orig_pe;
```

Blok eval zajmuje się przeprowadzeniem transakcji, a informacje o stanie jej zakończenia znajdują się w zmiennej \$? . Jeżeli zapytania UPDATE i metoda commit() zostaną wykonane bez błędów, zmienna \$? będzie pusta. W przeciwnym razie wykonanie bloku eval zakończy się niepowodzeniem i zmienna \$? będzie zawierała komunikat błędu. W takim przypadku kod znajdujący się po bloku eval wyświetli wspomniany komunikat i wywoła metodę rollback() w celu anulowania transakcji. (Operacja wycofania przebiega w bloku eval w celu uniemożliwienia przerwania wykonywania skryptu, gdy wspomniana operacja zakończy się niepowodzeniem).

W tym rozdziale skrypty DBI używają trybu obsługi błędów w postaci włączonego atrybutu `RaiseError` i wyłączzonego `PrintError`. Oznacza to, że mają już odpowiednie wartości wymagane do przeprowadzania transakcji i dlatego nie ma konieczności zachowywania, ustawiania i przywracania dwóch wymienionych atrybutów, jak to pokazano w omówionym przykładzie. Jednak zaprezentowane rozwiązanie jest podejściem ogólnego przeznaczenia, które działa nawet wtedy, gdy nie ma pewności co do ustawionych atrybutów obsługi błędów.

8.3. Praca z DBI

Na tym etapie poznałeś już wiele koncepcji wykorzystywanych podczas programowania DBI. Przejdźmy więc do pewnych zadań, które chcemy wykonać w przykładowej bazie danych. Spośród celów początkowo przedstawionych w rozdziale 1., zatytułowanym „Rozpoczęcie pracy z MySQL”, wybieramy te, które można osiągnąć za pomocą skryptów DBI:

- W przypadku projektu ocen uczniów celem jest pobieranie wyników dla dowolnego sprawdzianu lub testu.
- W przypadku projektu Ligi Historycznej mamy następujące cele:
 - ◆ Wygenerowanie katalogu członków Ligi w różnych formatach. Potrzebujemy listy nazwisk, która będzie użyta w programie rozdawanym w trakcie dorocznego spotkania członków Ligi. Ponadto, potrzebujemy katalogu w formacie przydatnym do wydrukowania go na papierze.
 - ◆ Wyszukanie członków Ligi, których członkostwo wkrótce wygasa, i wysłanie im wiadomości e-mail przypominającej o odnowieniu członkostwa.
 - ◆ Edycja rekordów członków Ligi. (Po odnowieniu członkostwa konieczne jest uaktualnienie daty jego wygaśnięcia).
 - ◆ Wyszukiwanie członków Ligi o wspólnych zainteresowaniach.
 - ◆ Umieszczenie katalogu członków Ligi w internecie.

Do wykonania niektórych z wymienionych zadań utworzymy skrypty działające z poziomu wiersza poleceń. Dla innych przygotujemy skrypty w podrozdziale 8.4, zatytułowanym „Użycie DBI w aplikacjach sieciowych”, które będą używane w połączeniu z serwerem WWW. W ten sposób na końcu rozdziału pozostanie nam kilka zadań do wykonania. Dokończymy je w rozdziale 9., zatytułowanym „Tworzenie programów MySQL przy użyciu języka PHP”.

8.3.1. Generowanie katalogu Ligi Historycznej

Jednym z celów jest wygenerowanie w różnych formatach informacji z katalogu Ligi Historycznej. Najprostszy format do wygenerowania to po prostu lista nazwisk umieszczana w programie rozdawanym w trakcie dorocznego spotkania członków Ligi. Ten format może być prostą listą w postaci zwykłego tekstu, która stanie się częścią większego dokumentu

używanego do przygotowania programu. Potrzebne są jedynie dane do umieszczenia w dokumencie.

W przypadku drukowanego katalogu potrzebujemy nieco ładniejszego formatowania, a więc czegoś więcej niż tylko zwykły tekst. Rozsądnym rozwiązaniem jest tutaj format RTF (ang. *Rich Text Format*), opracowany przez firmę Microsoft i obsługiwany przez wiele procesorów tekstu. Microsoft Word to tylko jeden z programów obsługujących format RTF, ale mamy też wiele innych, na przykład OpenOffice. Poszczególne procesory tekstów obsługują format RTF w różnym stopniu. W omawianym przykładzie wykorzystamy jedynie niewielką część pełnej specyfikacji RTF, więc powinna być ona prawidłowo rozpoznana przez dowolny program obsługujący format RTF. Na przykład, w systemie OS X edytor TextEdit i przeglądarka internetowa Safari bez problemów odczytują dane wyjściowe RTF, które tutaj wygenerujemy.

Procedury generowania listy (w postaci zwykłego tekstu) na doroczne spotkanie członków Ligi i katalogu w formacie RTF są w zasadzie identyczne: wykonanie zapytania, użycie pętli w celu pobrania rekordów i sformatowania danych wyjściowych. Biorąc pod uwagę podobieństwo między procedurami, dobrze jest uniknąć tworzenia oddzielnych skryptów dla każdego formatu. Przygotujemy więc pojedynczy skrypt *gen_dir.pl*, odpowiedzialny za wygenerowanie różnego rodzaju danych wyjściowych. Struktura skryptu przedstawia się następująco:

1. Przed zapisaniem informacji o członkach Ligi przeprowadzana jest cała inicjalizacja wymagana przez format danych wyjściowych. W przypadku listy w postaci zwykłego tekstu nie jest wymagana żadna inicjalizacja, natomiast pewne początkowe znaki kontrolne języka są potrzebne dla dokumentu w wersji RTF.
2. Pobranie każdego rekordu i odpowiednie sformatowanie danych wyjściowych.
3. Po przetworzeniu rekordu należy przeprowadzić niezbędne operacje czyszczące. Ponownie, w przypadku zwykłego tekstu nie ma żadnych specjalnych wymagań, natomiast wersja RTF dokumentu wymaga dodania pewnych zamykających znaczników kontrolnych języka.

Przewidując, że w przyszłości ten skrypt będziemy chcieli wykorzystać do wygenerowania danych wyjściowych w innych formatach, zapewnimy mu możliwość rozbudowy. W tym celu użyjemy tablicy hash, której poszczególne elementy będą odpowiadały innym formatom danych wyjściowych. Każdy element wskazuje metody do wywołania, które będą odpowiedzialne za wygenerowanie danych wyjściowych w konkretnym formacie: metoda inicjalizacyjna, metoda formatująca i zapisująca dane wyjściowe oraz metoda przeprowadzająca niezbędne operacje czyszczące:

```
# Tablica switchbox wymieniająca metody formatowania dla poszczególnych formatów danych wyjściowych.
my %switchbox =
(
    "text" =>                                # Metody do obsługi formatu zwykłego tekstu.
    {
        "init"    => undef,                    # Inicjalizacja nie jest wymagana.
        "entry"   => \&text_format_entry,
```

```

        "cleanup" => undef          # Operacje czyszczące nie są wymagane.
    },
    "rtf" =>                        # Metody do obsługi formatu RTF.
    {
        "init"      => \&rtf_init,
        "entry"     => \&rtf_format_entry,
        "cleanup"   => \&rtf_cleanup
    }
};

```

Każdy element tablicy ma klucz w postaci nazwy formatu (text i rtf). Skrypt utworzymy w taki sposób, aby argument podany w wierszu poleceń wskazywał żądany format danych wyjściowych:

```

% ./gen_dir.pl text
% ./gen_dir.pl rtf

```

Dzięki konfiguracji tablicy w przedstawiony sposób można bardzo łatwo dodać nowy format, gdy pojawi się potrzeba:

1. Utworzenie trzech metod formatujących używanych w fazie generowania danych wyjściowych.
2. Dodanie nowego elementu do tablicy, definiującego nazwę formatu oraz wskazującego utworzone wcześniej metody generowania danych wyjściowych.
3. W celu wygenerowania danych wyjściowych w nowym formacie należy wywołać skrypt *gen_dir.pl* i jako argument w wierszu poleceń podać nazwę formatu.

Kod dokonuje wyboru odpowiedniego elementu na podstawie pierwszego argumentu podanego w wierszu poleceń. Jeżeli nie zostanie podana prawidłowa nazwa wskazująca obsługiwany format, skrypt wygeneruje komunikat błędu i wyświetli listę dozwolonych nazw. Jeśli zaś użytkownik poda nazwę obsługiwanego formatu, zostanie ona zapisana w zmiennej `$func_hashref`:

```

my $formats = join (" ", sort (keys (%switchbox)));
# Sprawdzenie, czy w wierszu poleceń został podany argument.
@ARGV == 1
    or die "Użycie: gen_dir.pl format_type\nObsługiwane formaty: $formats\n";

# Określenie prawidłowego elementu dla argumentu wiersza poleceń.
# Jeżeli element nie zostanie znaleziony, podano nieobsługiwany format.
my $func_hashref = $switchbox{$ARGV[0]};
defined ($func_hashref)
    or die "Nieznany format: $ARGV[0]\nObsługiwane formaty: $formats\n";

```

Kod wybierający format opiera się na tym, że nazwa formatu danych wyjściowych jest kluczem w tablicy `%switchbox`. Dla każdej prawidłowej nazwy formatu element tablicy wskazuje odpowiednie metody generujące dane wyjściowe. Z kolei dla nieprawidłowej nazwy formatu element nie istnieje. Dzięki temu żadnych nazw formatów nie trzeba na stałe umieszczać w kodzie odpowiedzialnym za wybór formatu. Po dodaniu nowego elementu do tablicy kod automatycznie wykrywa obsługę nowego formatu.

Jeżeli w wierszu poleceń zostanie podana prawidłowa nazwa formatu, będzie ona przechowywana w zmiennej `$func_hashref`. Wartość wymienionej zmiennej jest odniesieniem do elementu tablicy wymieniającego metody generujące dane wyjściowe w wybranym formacie. Przedstawiony poniżej kod używa zmiennej `$func_hash` do wywołania metody inicjalizacyjnej, pobrania rekordów i wyświetlenia ich wartości, a także przeprowadzenia niezbędnych operacji czyszczących:

```
# Wywołanie metody inicjalizującej, o ile taka istnieje.
&{$func_hashref->{init}} if defined ($func_hashref->{init});

# Pobranie wartości rekordu, o ile taki istnieje.
if (defined ($func_hashref->{entry}))
{
    my $sth = $dbh->prepare (qq{
        SELECT * FROM member ORDER BY last_name, first_name
    });
    $sth->execute ();
    while (my $entry_ref = $sth->fetchrow_hashref ("NAME_lc"))
    {
        # Przekazanie (przez odniesienie) wartości do funkcji formatującej.
        &{$func_hashref->{entry}} ($entry_ref);
    }
}

# Wywołanie metody czyszczącej, o ile taka istnieje.
&{$func_hashref->{cleanup}} if defined ($func_hashref->{cleanup});
```

Pętla pobierająca rekordy używa metody `fetchrow_hashref()` nie bez powodu. Jeżeli pętla pobrałaby tablicę, wtedy metoda formatująca musiałaby znać kolejność kolumn. Istnieje możliwość ustalenia kolejności kolumn za pomocą atrybutu `$sth->{NAME}` (zawierającego nazwy kolumn w kolejności ich zwrócenia), ale dlaczego mielibyśmy się tym zajmować? Dzięki użyciu odniesień do tablicy każda metoda formatująca może odwoływać się do wybranych wartości kolumn przez ich podanie za pomocą składni `$entry_ref->{col_name}`.

Do zrobienia pozostało utworzenie wymienionych w tablicy hash metod odpowiedzialnych za wygenerowanie danych wyjściowych w poszczególnych formatach.

8.3.1.1. Generowanie listy członków Ligi w postaci zwykłego tekstu

Dane wyjściowe w postaci zwykłego tekstu nie wymagają żadnych kroków inicjalizacji i czyszczenia. Potrzebna jest jedynie metoda formatująca o nazwie `text_format_entry()`, która pobiera odniesienie do imienia i nazwiska członka Ligi, a następnie generuje odpowiednie dane wyjściowe. Najtrudniejszym zadaniem podczas przygotowywania listy członków Ligi jest obsługa przyrostków. Przyrostek taki jak Jr. lub Sr. jest poprzedzony przecinkiem i spacją, podczas gdy przyrostek typu II lub III jest poprzedzony jedynie spacją:

```
Michael Alvis IV
Clarence Elgar, Jr.
Bill Matthews, Sr.
Mark York II
```

I, V i X to jedyne litery używane w cyfrach rzymskich dla oznaczenia generacji od 1 do 39. Istnieje znikome prawdopodobieństwo, że będzie potrzebna liczba wykraczająca poza ten zakres liczbowy. Dlatego też ustalenie, czy dołączyć przecinek, można przeprowadzić za pomocą sprawdzenia wzorca przyrostka:

```
/^[IVX]+$/
```

Kod w metodzie `text_format_entry()` odpowiedzialny za umieszczenie poszczególnych członów imienia i nazwiska w odpowiedniej kolejności jest potrzebny także podczas generowania katalogu w formacie RTF. Dlatego też zamiast powielać ten kod w metodzie `rtf_format_entry()`, umieścimy go w metodzie pomocniczej:

```
sub format_name
{
    my $entry_ref = shift;

    my $name = $entry_ref->{first_name} . " " . $entry_ref->{last_name};
    if (defined ($entry_ref->{suffix}))      # Istnieje przyrostek dla nazwiska.
    {
        # Brak przecinka w przyrostkach typu I, II, III itd.
        $name .= "," unless $entry_ref->{suffix} =~ /^[IVX]+$/;
        $name .= " " . $entry_ref->{suffix}
    }
    return ($name);
}
```

Po przygotowaniu metody `format_name()` implementacja metody `text_format_function()` staje się niezwykle prosta:

```
sub text_format_entry
{
    printf "%s\n", format_name ($_[0]);
}
```

8.3.1.2. Generowanie katalogu w formacie RTF

Generowanie katalogu w formacie RTF jest nieco trudniejsze niż wygenerowanie listy członków Ligi do programu rozdawanego na corocznym spotkaniu członków Ligi. Przede wszystkim konieczne jest wygenerowanie informacji na podstawie każdego rekordu. Ponadto, dla każdego wpisu w dokumencie RTF trzeba umieścić pewne znaczniki kontrolne języka, aby uzyskać efekt formatowania. Poza tym, pewne znaczniki kontrolne muszą znaleźć się na początku i końcu dokumentu. Minimalny kod tworzący dokument RTF przedstawia się następująco:

```
{\rtf0
{\fonttbl {\f0 Times;}}
\plain \f0 \fs24
...Miejsce na zawartość dokumentu...
}
```

Dokument rozpoczyna się i kończy nawiasem klamrowym (odpowiednio { i }). Słowa kluczowe RTF zaczynają się od ukośnika, a pierwszym słowem kluczowym w dokumencie musi być `\rtfn`, gdzie *n* oznacza wersję specyfikacji używaną przez dokument. Wersja 0 jest odpowiednia dla naszych potrzeb.

W dokumencie określamy tabelę czcionek, wskazując tym samym czcionkę dla poszczególnych wpisów. Informacje o tabeli czcionek są umieszczone w grupie składającej się z klamrowego nawiasu otwierającego, słowa kluczowego `\fonttbl` i danych dotyczących czcionki. Powyższy kod definiuje tabelę czcionek składającą się z jednej czcionki o numerze 0, którą jest Times. (W omawianym przykładzie potrzebujemy tylko jednej czcionki, ale możesz ich użyć więcej, jeśli chcesz urozmaicić dokument).

Kolejnych kilka dyrektyw powoduje zdefiniowanie domyślnego stylu formatowania: `\plain` oznacza zwykły format, `\fo` wskazuje czcionkę numer 0 (zdefiniowaną w tabeli czcionek jako Times), a `\fs24` oznacza czcionkę o wielkości 12 punktów (liczba po `\fs` określa wielkość czcionki podaną w półpunktach). Nie ma konieczności definiowania marginesów, ponieważ większość procesorów tekstu stosuje rozsądne wartości domyślne.

Kod inicjalizujący i czyszczący tworzą strukturę dokumentu. Wspomniany kod przedstawiono poniżej (zwróć uwagę na podwójne ukośniki w celu otrzymania dosłownych ukośników w danych wyjściowych):

```
sub rtf_init
{
    print "{\\rtf0\n";
    print "{\\fonttbl {\\f0 Times;}}\n";
    print "\\plain \\f0 \\fs24\n";
}
sub rtf_cleanup
{
    print "}\n";
}
```

Funkcja formatująca wpis tworzy zawartość dokumentu. Zdecydowaliśmy się na bardzo proste podejście; każdy wpis jest generowany jako seria linii wraz z etykietami. Jeżeli informacje dotyczące danego wiersza danych wyjściowych nie istnieją, wiersz będzie pominięty. Na przykład, wiersz E-mail nie jest generowany dla członków Ligi, którzy nie podali adresu e-mail. Pewne wiersze, takie jak Adres, składają się z informacji pochodzących z wielu kolumn (street, city, state, zip), więc skrypt musi sobie poradzić z różnymi kombinacjami brakujących wartości. Poniżej przedstawiono przykład formatu danych wyjściowych:

Imię i nazwisko: Mike Artel

Adres: 4264 Lovering Rd., Miami, FL 12777

Telefon: 075-961-0712

E-mail: mike_artel@venus.org

Zainteresowania: Civil Rights, Education, Revolutionary War

Wpis w formacie RTF przedstawia się następująco:

```
\\b Imię i nazwisko: Mike Artel\\b0\\par
Adres: 4264 Lovering Rd., Miami, FL 12777\\par
Telefon: 075-961-0712\\par
E-mail: mike_artel@venus.org\\par
Zainteresowania: Civil Rights, Education, Revolutionary War\\par
```

Aby wiersz imienia i nazwiska był pogrubiony, na początku znajduje się znacznik `\b` i spacja, wskazując aktywację czcionki pogrubionej, a na końcu znacznik `\b0`, oznaczający dezaktywację czcionki pogrubionej. Imię i nazwisko członka Ligi jest formatowane przez metodę `format_name()`, przedstawioną w poprzednim podpunkcie 8.3.1.1. Każdy wiersz ma na końcu znacznik akapitu (`\par`), który nakazuje procesorowi tekstu przejście do kolejnego wiersza — to nic skomplikowanego. Podstawowa trudność kryje się w sformatowaniu ciągu tekstowego adresu oraz określeniu wierszy danych wyjściowych do wygenerowania:

```
sub rtf_format_entry
{
    my $entry_ref = shift;

    printf "\\b Imię i nazwisko: %s\\b0\\par\\n", format_name ($entry_ref);
    my $address = "";
    $address .= $entry_ref->{street}
        if defined ($entry_ref->{street});
    $address .= ", " . $entry_ref->{city}
        if defined ($entry_ref->{city});
    $address .= ", " . $entry_ref->{state}
        if defined ($entry_ref->{state});
    $address .= " " . $entry_ref->{zip}
        if defined ($entry_ref->{zip});
    print "Adres: $address\\par\\n"
        if $address ne "";
    print "Telefon: $entry_ref->{phone}\\par\\n"
        if defined ($entry_ref->{phone});
    print "E-mail: $entry_ref->{email}\\par\\n"
        if defined ($entry_ref->{email});
    print "Zainteresowania: $entry_ref->{interests}\\par\\n"
        if defined ($entry_ref->{interests});
    print "\\par\\n";
}
```

Nie jesteś ograniczony jedynie do przedstawionego stylu formatowania. Aby zmienić styl generowanego katalogu, wystarczy po prostu zmodyfikować metodę `rtf_format_entry()`. Gdyby katalog znajdował się w jego oryginalnej postaci (dokument procesora tekstu), takie rozwiązanie nie byłoby łatwe do zastosowania.

Skrypt *gen_dir.pl* jest teraz kompletny. Możesz więc wygenerować katalog w postaci zwykłego tekstu lub danych wyjściowych w formacie RTF, wywołując wymienione poniżej polecenia:

```
% ./gen_dir.pl text > names.txt
% ./gen_dir.pl rtf > directory.rtf
```

Na tym etapie można bardzo prosto wstawić zawartość pliku tekstowego w dokumencie dorocznego spotkania członków Ligi lub użyć pliku RTF w dowolnym programie obsługującym format RTF.

DBI bardzo ułatwia wyodrębnianie żądanych informacji z bazy danych MySQL, a oferowane przez Perl możliwości w zakresie przetwarzania tekstu niezwykle ułatwiają umieszczanie tych informacji w wybranym formacie. MySQL nie zapewnia szczególnie

eleganckiego sposobu formatowania danych wyjściowych, ale to nie ma znaczenia z powodu łatwości, z jaką można zintegrować bazę danych MySQL z językiem takim jak Perl, który ma doskonałe możliwości w zakresie przetwarzania tekstu.

8.3.2. Wysyłanie członkom Ligi przypomnień o konieczności przedłużenia członkostwa

Katalog Ligi Historycznej w jego początkowej formie (dokument procesora tekstów) wymaga przeprowadzenia czasochłonnej i podatnej na błędy procedury wyszukania osób, które trzeba poinformować o konieczności odnowienia członkostwa. Skoro informacje przechowujemy teraz w bazie danych, to istnieje możliwość automatyzacji tego procesu, przynajmniej w pewnym stopniu. Można łatwo wyszukać osoby, które powinny odnowić członkostwo, a następnie wysłać im wiadomość e-mail z odpowiednią informacją. W ten sposób unikamy konieczności telefonowania do tych osób lub wysyłania im listów tradycyjną pocztą.

Wyszukać trzeba osoby, którym członkostwo wygasło lub wygaśnie w najbliższym czasie. Odpowiednie zapytanie zawiera prostą operację obliczenia daty:

```
SELECT ... FROM member
WHERE expiration < DATE_ADD(CURDATE(), INTERVAL liczba_dni DAY)
```

Parametr *liczba_dni* określa liczbę dni pozostałego członkostwa. Powyższe zapytanie wyszukuje osoby, które muszą odnowić członkostwo w czasie krótszym niż wskazana liczba dni (lub ich członkostwo już wygasło). Aby wyszukać jedynie osoby, którym członkostwo wygasło, parametrowi *liczba_dni* należy przypisać wartość zero i tym samym wybrać rekordy o dacie wygaśnięcia w przeszłości.

Jaki powinien być następny krok po wyszukaniu osób, które trzeba powiadomić? Jedną z możliwości jest wysłanie wiadomości e-mail bezpośrednio z samego skryptu. Jednak użyteczne może być przejrzanie listy osób przed wysłaniem im jakichkolwiek wiadomości. Zastosujemy więc podejście składające się z dwóch kroków:

1. Uruchomienie skryptu *need_renewal.pl* w celu wygenerowania listy osób, które muszą odnowić członkostwo. Utworzoną listę można przeanalizować w celu weryfikacji lub modyfikacji, a następnie wykorzystać jako dane wejściowe w drugim kroku, czyli wysłaniu odpowiednich wiadomości.
2. Uruchomienie skryptu *renewal_notify.pl*, wysyłającego wskazanym osobom informacje o konieczności odnowienia członkostwa. Ten skrypt powinien informować o osobach, które nie podały adresu e-mail, ponieważ wówczas trzeba do nich zatelefonować.

W celu wykonania pierwszej części zadania skrypt *need_renewal.pl* musi wyszukać osoby, które powinny wkrótce odnowić członkostwo. Główna część skryptu przedstawia się następująco:

```
# Domyślnie, osoba jest informowana na 30 dni przed wygaśnięciem członkostwa...
my $cutoff = 30;
```

```
#...ale w wierszu poleceń można podać inną liczbę.
$cutoff = shift (@ARGV) if @ARGV && $ARGV[0] =~ /\d+$/;

# Poinformowanie użytkownika o liczbie dni używanej przez skrypt.
warn "Liczba dni do upłygnięcia członkostwa: $cutoff\n";

my $sth = $dbh->prepare (qq{
    SELECT
        member_id, email, last_name, first_name, expiration,
        TO_DAYS(expiration) - TO_DAYS(CURDATE()) AS days
    FROM member
    WHERE expiration < DATE_ADD(CURDATE(), INTERVAL ? DAY)
    ORDER BY expiration, last_name, first_name
});
$sth->execute ($cutoff); # Przekazanie liczby dni jako wartości miejsca zarezerwowanego.

while (my $entry_ref = $sth->fetchrow_hashref ())
{
    # Konwersja wartości undef na puste ciągi tekstowe w celu wyświetlenia na ekranie.
    foreach my $key (keys (%{$entry_ref}))
    {
        $entry_ref->{$key} = "" if !defined ($entry_ref->{$key});
    }
    print join ("\\t",
        $entry_ref->{member_id},
        $entry_ref->{email},
        $entry_ref->{last_name},
        $entry_ref->{first_name},
        $entry_ref->{expiration},
        $entry_ref->{days} . " dni"),
        "\\n";
}
}
```

Dane wyjściowe skryptu *need_renewal.pl* będą podobne do przedstawionych poniżej (otrzymasz inne dane, ponieważ wyniki są określane na podstawie bieżącej daty, która będzie inna niż data pisania niniejszego rozdziału):

89	g.steve@pluto.com	Garner	Steve	2012-08-03	-38 dni
18	york_mark@earth.com	York	Mark	2012-08-24	-17 dni
82	john_edwards@venus.org	Edwards	John	2012-09-12	2 dni

Zwróć uwagę, że w przypadku pewnych osób podana została ujemna liczba dni. Oznacza to, że ich członkostwo wygasło! (Tak się może zdarzać w przypadku ręcznej obsługi wierszy tekstu, po prostu można coś przeoczyć. Po umieszczeniu informacji w bazie danych łatwo można wyszukać osoby, które wcześniej przeoczono).

Druga część zadania obejmuje powiadomienie wybranych osób za pomocą skryptu *renewal_notify.pl*, który wysyła odpowiednie wiadomości przez e-mail. Aby nieco ułatwić użycie skryptu *renewal_notify.pl*, można zaimplementować w nim obsługę trzech rodzajów argumentów wiersza poleceń: identyfikator członka Ligi, adres e-mail i nazwę pliku. Argumenty liczbowe to po prostu identyfikatory członków Ligi, z kolei argumenty zawierające znak @ oznaczają adresy e-mail. Wszystko inne jest traktowane jako nazwa pliku do odczytania zawierającego identyfikatory lub adresy e-mail. Takie rozwiązanie pozwala na wskazanie osób za pomocą identyfikatorów członków Ligi lub ich adresów

e-mail, zarówno bezpośrednio w wierszu poleceń, jak i w pliku. (Istnieje możliwość zapisania w pliku wyniku działania skryptu *need_renewal.pl*, a następnie użycia wspomnianego pliku jako danych wejściowych skryptu *renewal_notify.pl*).

Dla każdego członka Ligi, któremu trzeba wysłać wiadomość, skrypt wyszukuje odpowiedni rekord w tabeli *member*, a następnie wyodrębnia z niego adres e-mail i wysyła wiadomość na pobrany adres. Jeżeli rekord nie zawiera adresu e-mail, skrypt *renewal_notify.pl* generuje komunikat informujący o konieczności kontaktu z daną osobą w inny sposób.

Poniżej przedstawiono główną pętlę przetwarzającą argumenty. Jeżeli w wierszu poleceń nie zostanie podany żaden argument, dane wejściowe będą odczytywane ze standardowego wejścia. W przeciwnym razie każdy argument zostanie przetworzony przez jego przekazanie metodzie *interpret_argument()* w celu klasyfikacji jako identyfikatora, adresu e-mail lub nazwy pliku:

```
if (@ARGV == 0)    # Brak argumentów, odczyt wartości z STDIN.
{
    read_file (\*STDIN);
}
else
{
    while (my $arg = shift (@ARGV))
    {
        # Interpretacja argumentu wraz z rekurencją nazwy pliku.
        interpret_argument ($arg, 1);
    }
}
```

Metoda *read_file()* odczytuje zawartość pliku (przyjęto założenie, że jest już otwarty) i przechodzi na początek każdego wiersza. (Jeżeli dane wyjściowe skryptu *need_renewal.pl* zostały użyte jako dane wejściowe *renewal_notify.pl*, każdy wiersz składa się z kilku pól. Tutaj sprawdzany jest tylko pierwszy, to identyfikator członka Ligi).

```
sub read_file
{
    my $fh = shift;    # Uchwyt do już otworzonego pliku.
    my $arg;
    while (defined ($arg = <$fh>))
    {
        # Usunięcie wszystkiego (także znaku nowego wiersza) poza pierwszą kolumną.
        $arg =~ s/\s.*//s;
        # Interpretacja argumentu bez rekurencji nazwy pliku.
        interpret_argument ($arg, 0);
    }
}
```

Metoda *interpret_argument()* klasyfikuje każdy argument w celu ustalenia, czy jest identyfikatorem, adresem e-mail, czy nazwą pliku. W przypadku identyfikatorów i adresów e-mail następuje wyszukanie odpowiedniego rekordu w tabeli, a następnie przekazanie go metodzie *notify_member()*. Należy zachować ostrożność w przypadku osób wskazywanych przez adresy e-mail. Istnieje możliwość, że dwie osoby będą korzystały z tego samego adresu e-mail (na przykład mąż i żona), a nie chcemy przecież wysłać wiadomości do niewłaściwej

osoby. Aby uniknąć takiej sytuacji, należy wyszukać identyfikator członka Ligi odpowiadający adresowi e-mail i upewnić się, że istnieje tylko jeden. Jeżeli adres będzie dopasowany do więcej niż tylko jednego identyfikatora członka Ligi, powstaje niejednoznaczność i trzeba adres zignorować po uprzednim wyświetleniu odpowiedniego ostrzeżenia.

Jeżeli argument nie wygląda na identyfikator członka Ligi lub jego adres e-mail, zostanie uznany za nazwę pliku zawierającego dane wejściowe. W tym przypadku również trzeba zachować ostrożność — nie chcemy odczytać pliku, jeśli już odczytujemy plik, aby uniknąć powstania nieskończonej pętli:

```
sub interpret_argument
{
    my ($arg, $recurse) = @_ ;

    if ($arg =~ /^^\d+$/)      # Liczbowy identyfikator członka Ligi.
    {
        notify_member ($arg);
    }
    elsif ($arg =~ /\@/)      # Adres e-mail.
    {
        # Pobranie identyfikatora członka Ligi powiązanego z adresem
        # (powinien być dokładnie jeden).
        my $stmt = qq{ SELECT member_id FROM member WHERE email = ? };
        my $ary_ref = $dbh->selectcol_arrayref ($stmt, undef, $arg);
        if (scalar (@{$ary_ref}) == 0)
        {
            warn "Dla podanego adresu e-mail $arg nie znaleziono rekordu: będzie zignorowany\n";
        }
        elsif (scalar (@{$ary_ref}) > 1)
        {
            warn "Dla podanego adresu e-mail $arg znaleziono kilka rekordów: będzie zignorowany\n";
        }
        else
        {
            notify_member ($ary_ref->[0]);
        }
    }
    else
        # Nazwa pliku.
    {
        if (!$recurse)
        {
            {
                warn "Nazwa pliku $arg wewnątrz nazwy pliku: będzie zignorowana\n";
            }
            else
            {
                {
                    open (IN, $arg) or die "Nie można otworzyć pliku $arg: $!\n";
                    read_file (\*IN);
                    close (IN);
                }
            }
        }
    }
}
```

Metoda `notify_member()` jest odpowiedzialna za faktyczne wysłanie informacji o konieczności odnowienia członkostwa. Jeżeli okaże się, że dana osoba nie podała adresu e-mail, metoda `notify_member()` nie będzie mogła wysłać żadnej wiadomości, ale wyświetli

komunikat informujący o konieczności skontaktowania się w inny sposób z danym członkiem Ligi. Możesz wywołać skrypt *show_member.pl* wraz identyfikatorem członka Ligi podanego w wiadomości, aby w ten sposób wyświetlić o nim pełne informacje i na przykład odczytać jego numer telefonu. (Skrypt *show_member.pl* przedstawiono i omówiono w punkcie 8.2.11, zatytułowanym „Używanie metadanych zbioru wynikowego”).

Kod metody *notify_member()* przedstawia się następująco:

```
sub notify_member
{
    my $member_id = shift;

    warn "Wysyłanie wiadomości do członka Ligi o identyfikatorze $member_id...\n";
    my $stmt = qq{ SELECT * FROM member WHERE member_id = ? };
    my $sth = $dbh->prepare ($stmt);
    $sth->execute ($member_id);
    my @col_name = @{$sth->{NAME}};
    my $entry_ref = $sth->fetchrow_hashref ();
    $sth->finish ();
    if (!$entry_ref)
    {
        # Nie znaleziono członka Ligi!
        warn "Nie znaleziono rekordu dla członka Ligi o identyfikatorze $member_id!\n";
        return;
    }
    if (!defined ($entry_ref->{email}))
    {
        # Brak podanego adresu e-mail.
        warn "Osoba $member_id nie podała adresu e-mail; wiadomość nie została wysłana\n";
        return;
    }
    open (OUT, "| $sendmail") or die "Nie można wysłać wiadomości.\n";
    print OUT <<EOF;
    To: $entry_ref->{email}
    Subject: Twoje członkostwo w Lidze Historycznej wymaga odnowienia

    Witaj! Twoje członkostwo w Lidze Historycznej wkrótce wygaśnie.
    Mamy nadzieję, że poświęcisz kilka minut na kontakt z biurem
    Ligi Historycznej i odnowisz członkostwo. Poniżej przedstawiono
    informacje, które podałeś o sobie do katalogu członków Ligi. Prosimy
    o zwrócenie szczególnej uwagi na datę wygaśnięcia członkostwa.
    Dziękujemy!
    EOF
    foreach my $col_name (@col_name)
    {
        printf OUT "$col_name:";
        printf OUT " $entry_ref->{$col_name}"
            jest zdefiniowany ($entry_ref->{$col_name});
        printf OUT "\n";
    }
    close (OUT);
}
```

Metoda *notify_member()* wysyła wiadomość e-mail przez otwarcie potoku do programu *sendmail* i umieszczenie w nim wiadomości. Ścieżka dostępu do programu *sendmail* została podana jako parametr na początku skryptu *renewal_notify.pl*. Może wystąpić konieczność zmiany tej ścieżki dostępu, aby odpowiadała położeniu programu *sendmail* w używanym przez Ciebie systemie:

```
# Zmień ścieżkę dostępu, aby była odpowiednia dla Twojego systemu.
my $sendmail = "/usr/sbin/sendmail -t -oi";
```

Jeżeli nie masz programu `sendmail`, skrypt nie będzie działał prawidłowo. (Na przykład, w systemach Windows program `sendmail` najczęściej nie jest zainstalowany). Aby obejść ten problem, w dystrybucji `sampdb` umieszczono zmodyfikowaną wersję skryptu `renewal_notify.pl` o nazwie `renewal_notify2.pl`, która używa modułu `Mail::Sendmail`, działającego bez programu `sendmail`. Po zainstalowaniu wymienionego modułu będziesz mógł używać skryptu `renewal_notify2.pl`.

Istnieje możliwość rozbudowania skryptu. Na przykład, możesz dodać do tabeli `member` kolumnę przeznaczoną do rejestrowania daty ostatniego wysłania przypomnienia, a następnie zmodyfikować skrypt `renewal_notify.pl`, aby uaktualniał tę kolumnę po wysłaniu wiadomości e-mail. W ten sposób unikniesz zbyt częstego wysyłania przypomnień członkom Ligi. W obecnej postaci skryptu upewnij się, że nie uruchamiasz go częściej niż raz w miesiącu.

Na tym etapie oba omówione skrypty są już gotowe. Możesz z nich korzystać w następujący sposób:

1. Uruchom skrypt `need_renewal.pl` w celu wygenerowania listy osób, których członkostwo wygasło lub wkrótce wygaśnie:
2. Spójrz na plik `tmp` i upewnij się, że jest prawidłowy.
3. Jeśli plik `tmp` jest prawidłowy, wskaż go jako dane wejściowe dla skryptu `renewal_notify.pl`:

```
% ./renewal_notify.pl tmp
```

Aby wysłać przypomnienia poszczególnym członkom Ligi, możesz ich wskazać w wierszu poleceń, podając identyfikatory lub adresy e-mail. Na przykład, poniższe polecenie powoduje wysłanie wiadomości do dwóch osób: pierwsza o identyfikatorze 18, druga o adresie e-mail `g.steve@pluto.com`:

```
% ./renewal_notify.pl 18 g.steve@pluto.com
```

8.3.3. Edycja rekordu członka Ligi Historycznej

Po rozpoczęciu wysyłania informacji przypominających o konieczności odnowienia członkostwa można przyjąć założenie, że część z poinformowanych osób odnowi członkostwo. W takim przypadku trzeba będzie uaktualnić ich rekordy i wprowadzić nową datę wygaśnięcia członkostwa. W następnym rozdziale opracujemy sposób pozwalający na edycję rekordów przez internet, natomiast w tym punkcie dowiesz się, jak można to zrobić za pomocą skryptu wiersza poleceń. Utworzymy skrypt o nazwie `edit_member.pl`, pozwalający na uaktualnienie rekordów za pomocą prostego podejścia polegającego na wyświetleniu pytania o nową wartość dla każdej części wpisu o użytkownika. Skrypt będzie działał w następujący sposób:

- Jeżeli zostanie wywołany bez argumentu wiersza polecenia, oznacza to wprowadzenie informacji o nowym członku Ligi. Skrypt poprosi o podanie wymaganych informacji i utworzy nowy rekord.
- Jeżeli zostanie wywołany wraz z identyfikatorem członka Ligi w wierszu poleceń, wtedy skrypt wyszuka zawartość istniejącego rekordu, a następnie uaktualni wszystkie jego kolumny. Po podaniu wartości dla kolumny zastąpi ona jej bieżącą wartość. Naciśnięcie klawisza *Enter* powoduje pozostawienie wartości kolumny bez zmian. Wprowadzenie słowa *brak* usuwa bieżącą wartość kolumny. Z kolei wprowadzenie słowa *koniec* kończy działanie skryptu bez tworzenia rekordu. (Jeśli nie znasz identyfikatora członka Ligi, uruchom skrypt *show_member.pl nazwisko*, aby wyświetlić w ten sposób rekordy dopasowane do podanego nazwiska i wybrać odpowiedni identyfikator).

Prawdopodobnie niepotrzebne jest umożliwianie w ten sposób edycji całego wiersza, jeśli uaktualniona ma być jedynie data wygaśnięcia członkostwa. Z drugiej strony, skrypt zapewnia użytkownikowi łatwą możliwość edycji dowolnych informacji o członku Ligi bez konieczności znajomości jakiegokolwiek kodu SQL. (Istnieje jeden wyjątek: skrypt *edit_member.pl* nie pozwala na zmianę kolumny *member_id*, ponieważ ta wartość jest generowana automatycznie podczas tworzenia rekordu i nie powinna być później modyfikowana).

Skrypt *edit_member.pl* musi przede wszystkim znać nazwy kolumn tabeli *member* i wiedzieć, czy mogą one przyjmować wartości NULL. Druga z wymienionych właściwości jest konieczna podczas usuwania zawartości kolumny (w celu przypisania jej wartości NULL, jeśli je akceptuje, lub pustego ciągu tekstowego, jeśli kolumna nie akceptuje wartości NULL). Wymagane informacje są dostępne w tabeli COLUMNS bazy danych INFORMATION_SCHEMA:

```
my @col_name = ();          # Tablica nazw kolumn.
my %nullable = ();          # Czy kolumna akceptuje wartości NULL?
# Pobranie nazw kolumn tabeli member.
my $sth = $dbh->prepare (qq{
    SELECT COLUMN_NAME, UPPER(IS_NULLABLE)
    FROM INFORMATION_SCHEMA.COLUMNS
    WHERE TABLE_SCHEMA = ? AND TABLE_NAME = ?
});
$sth->execute ("sampdb", "member");
while (my ($col_name, $is_nullable) = $sth->fetchrow_array ())
{
    push (@col_name, $col_name);
    $nullable{$col_name} = ($is_nullable eq "YES");
}
```

Za pomocą informacji o kolumnach skrypt generuje tablicę zawierającą nazwy kolumn wymienione w kolejności. We wspomnianej tablicy nazwy kolumn są kluczami i wskazują, czy akceptuje ona wartości NULL. Następnie skrypt *edit_member.pl* wykonuje pętlę główną:

```
if (@ARGV == 0) # Jeżeli nie podano argumentów, wtedy należy utworzyć nowy rekord.
{
    # Przekazanie odniesienia do tablicy nazw kolumn.
    new_member (\@col_name);
}
```

```

else
{
    # W przeciwnym razie przeprowadzana jest edycja rekordów wskazanych przez argumenty.
    # Zachowanie tablicy @ARGV, a następnie jej opróżnienie, aby kiedy skrypt odczytuje dane wejściowe
    # z STDIN, nie interpretował zawartości @ARGV jako nazw plików danych wejściowych.
    my @id = @ARGV;
    @ARGV = ();
    # Dla każdej wartości ID należy wyszukać odpowiedni rekord, a następnie przeprowadzić jego edycję.
    while (my $id = shift (@id))
    {
        $sth = $dbh->prepare (qq{
            SELECT * FROM member WHERE member_id = ?
        });
        $sth->execute ($id);
        my $entry_ref = $sth->fetchrow_hashref ();
        $sth->finish ();
        if (!$entry_ref)
        {
            warn "Nie znaleziono osoby o identyfikatorze = $id\n";
            next;
        }
        # Przekazanie odniesień do tablicy nazw kolumn i rekordu.
        edit_member (\@col_name, $entry_ref);
    }
}

```

Kod tworzący nowy rekord prosi o podanie wartości dla wszystkich kolumn tabeli member, a następnie wykonuje zapytanie INSERT odpowiedzialne za wstawienie nowego rekordu:

```

sub new_member
{
    my $col_name_ref = shift; # Odniesienie do tablicy nazw kolumn.
    my $entry_ref = { };      # Utworzenie nowego rekordu jako tablicy.
    return unless prompt ("Utworzyć nowy rekord (t/n)? ") =~ /\t/i;
    # Wyświetlenie pytań o nowe wartości; użytkownik podaje nową wartość lub naciska klawisz Enter,
    # pozostawiając wartość niezmienną. Słowo "BRAK" powoduje usunięcie wartości, natomiast
    # "KONIEC" kończy działanie skryptu bez utworzenia rekordu.
    foreach my $col_name (@{$col_name_ref})
    {
        next if $col_name eq "member_id"; # Pominiecie pola klucza.
        my $col_val = col_prompt ($col_name, undef);
        next if $col_val eq "";           # Użytkownik nacisnął klawisz Enter.
        return if uc ($col_val) eq "KONIEC"; # Zakończenie działania skryptu.
        if (uc ($col_val) eq "BRAK")
        {
            # Wstawienie wartości NULL, jeśli kolumna je akceptuje, w przeciwnym razie wstawiany jest pusty ciąg
            # tekstowy.
            $col_val = ($nullable{$col_name} ? undef : "");
        }
        $entry_ref->{$col_name} = $col_val;
    }
    # Wyświetlenie wartości i pytania o potwierdzenie danych, zanim zostaną wstawione.
    show_member ($col_name_ref, $entry_ref);
    return unless prompt ("\nWstawić rekord (t/n)? ") =~ /\t/i;
    # Przygotowanie zapytania INSERT, a następnie jego wykonanie.
    my $stmt = "INSERT INTO member";
    my $delim = " SET "; # Umieszczenie "SET" przed pierwszą kolumną, natomiast ", " przed pozostałymi.
    foreach my $col_name (@{$col_name_ref})

```

```

{
    # Użyte będą tylko te kolumny, dla których podano wartości.
    next if !defined ($entry_ref->{$col_name});
    # Metoda quote() wstawia undef jako słowo NULL (bez znaków cytowania),
    # co jest żądanym zachowaniem. Następuje przypisanie kolumnom typu NOT NULL
    # ich wartości domyślnych.
    $stmt .= sprintf ("%s %s=%s", $delim, $col_name,
                        $dbh->quote ($entry_ref->{$col_name}));
    $delim = ",";
}
$dbh->do ($stmt) or warn "Ostrzeżenie: nie utworzono nowego rekordu!\n"
}

```

Skrypt `edit_member.pl` używa dwóch procedur w celu pobrania informacji. Metoda `prompt()` zadaje pytanie i zwraca odpowiedź:

```

sub prompt
{
    my $str = shift;

    print STDERR $str;
    chomp ($str = <STDIN>);
    return ($str);
}

```

Metoda `col_prompt()` pobiera nazwę kolumny jako argument. Następnie wyświetla tę nazwę jako pytanie o nową wartość kolumny i zwraca wartość wprowadzoną przez użytkownika:

```

sub col_prompt
{
    my ($col_name, $entry_ref) = @_;

    my $prompt = $col_name;
    if (defined ($entry_ref))
    {
        my $cur_val = $entry_ref->{$col_name};
        $cur_val = "NULL" if !defined ($cur_val);
        $prompt .= " [$cur_val]";
    }
    $prompt .= ": ";
    print STDERR $prompt;
    my $str = <STDIN>;
    chomp ($str);
    return ($str);
}

```

Drugim argumentem metody `col_prompt()` jest odniesienie do tablicy przedstawiającej rekord członka Ligi. Podczas tworzenia nowego rekordu to będzie wartość `undef`, ale w przypadku istniejących rekordów wspomniana tablica prowadzi do bieżącej zawartości kolumn rekordu. W tej drugiej sytuacji metoda `col_prompt()` zawiera wartość bieżącej kolumny i wyświetla ją w pytaniu, aby była widoczna dla użytkownika, który w ten sposób może ją zaakceptować przez naciśnięcie klawisza *Enter*.

Kod odpowiedzialny za edycję istniejącego rekordu jest podobny do kodu tworzącego rekord nowego członka Ligi. Jednak mamy już dane istniejącego rekordu, więc procedura wyświetla bieżące wartości rekordu, a metoda `edit_member()` wykonuje zapytanie UPDATE zamiast INSERT:

```
sub edit_member
{
    # Odniesienie do tablicy nazw kolumn i tablicy rekordu.
    my ($col_name_ref, $entry_ref) = @_;
    # Wyświetlenie wartości początkowych i pytania, czy można przejść dalej do edycji rekordu.
    show_member ($col_name_ref, $entry_ref);
    return unless prompt ("Chcesz edytować ten rekord (t/n)? ") =~ /^t/i;
    # Wyświetlenie pytań o nowe wartości; użytkownik podaje nową wartość lub naciska klawisz Enter,
    # pozostawiając wartość niezmienioną. Słowo "BRAK" powoduje usunięcie wartości, natomiast
    # "KONIEC" kończy działanie skryptu bez utworzenia rekordu.
    foreach my $col_name (@{$col_name_ref})
    {
        next if $col_name eq "member_id"; # Pominięcie pola klucza
        my $col_val = col_prompt ($col_name, $entry_ref);
        next if $col_val eq "";           # Użytkownik nacisnął klawisz Enter.
        return if uc ($col_val) eq "KONIEC"; # Zakończenie działania skryptu.
        if (uc ($col_val) eq "BRAK")
        {
            # Wstawienie wartości NULL, jeśli kolumna je akceptuje, w przeciwnym razie wstawiany jest pusty
            # ciąg tekstowy.
            $col_val = ($nullable{$col_name} ? undef : "");
        }
        $entry_ref->{$col_name} = $col_val;
    }
    # Wyświetlenie nowych wartości i pytania o potwierdzenie danych, zanim zostaną wstawione.
    show_member ($col_name_ref, $entry_ref);
    return unless prompt ("\nUaktualnić ten rekord (t/n)? ") =~ /^t/i;

    # Przygotowanie zapytania UPDATE, a następnie jego wykonanie.
    my $stmt = "UPDATE member";
    my $delim = " SET "; # Umieszczenie "SET" przed pierwszą kolumną, natomiast " " przed pozostałymi.
    foreach my $col_name (@{$col_name_ref})
    {
        next if $col_name eq "member_id"; # Pominięcie pola klucza.
        # Metoda quote() wstawia undef jako słowo NULL (bez znaków cytowania),
        # co jest żądanym zachowaniem.
        $stmt .= sprintf ("%s %s=%s", $delim, $col_name,
                               $dbh->quote ($entry_ref->{$col_name}));
        $delim = ", ";
    }
    $stmt .= " WHERE member_id = " . $dbh->quote ($entry_ref->{member_id});
    $dbh->do ($stmt) or warn "Ostrzeżenie: nie uaktualniono rekordu!\n"
}
}
```

Problem ze skrypcem *edit_member.pl* polega na braku operacji weryfikacji danych wejściowych. W przypadku większości kolumn tabeli member zresztą nie ma zbyt wiele do weryfikacji, to po prostu kolumny ciągów tekstowych. Jednak w przypadku kolumny przechowującej datę wygaśnięcia ważności członkostwa trzeba się upewnić, że podana przez użytkownika wartość jest datą. W ogólnego przeznaczenia aplikacji pobierającej

dane prawdopodobnie będziesz chciał wyodrębnić informacje o tabeli, aby określić typ wszystkich jej kolumn. Wówczas można przeprowadzić weryfikację na podstawie ograniczeń dla poszczególnych typów. To jest znacznie bardziej zaawansowane rozwiązanie niż tutaj przedstawione. Jednak minimalną operację sprawdzania daty można zaimplementować w metodzie `col_prompt()`. Wspomniana operacja sprawdza format danych wejściowych dla kolumny `expiration`:

```
sub col_prompt
{
    my ($col_name, $entry_ref) = @_;

    loop:
    my $prompt = $col_name;
    if (defined ($entry_ref))
    {
        my $cur_val = $entry_ref->{$col_name};
        $cur_val = "NULL" if !defined ($cur_val);
        $prompt .= " [$cur_val]";
    }
    $prompt .= ": ";
    print STDERR $prompt;
    my $str = <STDIN>;
    chomp ($str);
    # Przeprowadzenie prostej operacji sprawdzenia daty wygaśnięcia członkostwa.
    if ($str && $col_name eq "expiration") # Sprawdzenie formatu daty dla kolumny expiration.
    {
        if ($str !~ /\d+\d+\d+\d+/)
        {
            warn "$str nie jest poprawną datą, spróbuj ponownie\n";
            goto loop;
        }
    }
    return ($str);
}
```

Procedura sprawdza, czy data składa się z sekwencji trzech liczb rozdzielonych znakiem innym niż cyfra. To jest jedynie prosta operacja sprawdzenia i nie wykrywa wartości takiej jak 1999-14-92, która zostaje uznana za poprawną. Aby poprawić skrypt, możesz zastosować bardziej rygorystyczne sprawdzanie daty, a także przeprowadzać sprawdzanie innych danych wejściowych, na przykład wprowadzić wymóg podania imienia i nazwiska.

Inne usprawnienia możliwe do wprowadzenia:

- Pominięcie operacji uaktualnienia istniejącego rekordu, jeśli użytkownik nie wprowadził w nim żadnych zmian. W tym celu wartości początkowe trzeba zachować, a następnie utworzyć zapytanie `UPDATE` uaktualniające tylko te kolumny, które zostały zmodyfikowane przez użytkownika. Jeśli żadna nie została zmieniona, zapytanie w ogóle nie powinno być wykonane.
- Poinformowanie użytkownika, że kiedy wprowadzał edycję rekordu, to został on zmodyfikowany przez innego użytkownika. W tym celu trzeba dodać klauzulę `WHERE` zawierającą `AND nazwa_kolumny = wartość_kolumny` dla każdej wartości początkowej. W ten sposób zapytanie `UPDATE` zakończy się niepowodzeniem, jeśli

ktokolwiek inny zmodyfikował rekord w międzyczasie. Dzięki temu skrypt poinformuje, że dwóch użytkowników próbuje jednocześnie zmodyfikować ten sam rekord.

- Włączenie trybu ścisłego SQL i ograniczeń dla danych wejściowych użytkownika. Spowoduje to odrzucanie przez MySQL nieprawidłowych wartości i wygenerowanie komunikatu błędu, gdy dane wejściowe nie będą mogły być użyte w podanej postaci:

```
$dbh->do ("SET sql_mode = 'TRADITIONAL'");
```

Skrypt *edit_member.pl* ma także jeszcze jedną wadę, której usunięcie warto rozważyć. W obecnej postaci skrypt nawiązuje połączenie z bazą danych przed wykonaniem pętli wyświetlającej pytania, ale nie zamyka połączenia aż do chwili ukończenia tworzenia zapytania w pętli. Jeżeli użytkownikowi dużą ilość czasu zabiera wprowadzenie danych nowego rekordu, nowych dla istniejącego lub wystąpi inny czynnik opóźniający te operacje, połączenie pozostanie otwarte przez długi czas. W jaki sposób zmodyfikujesz skrypt *edit_member.pl*, aby połączenie pozostało otwarte jedynie przez minimalną wymaganą ilość czasu?

8.3.4. Wyszukiwanie członków Ligi Historycznej o takich samych zainteresowaniach

Jednym z zajęć sekretarki Ligi Historycznej jest przetwarzanie zapytań, w których członkowie Ligi proszą o dostarczenie listy osób współdzielących te same zainteresowania, na przykład Wielkim Kryzysem lub życiem Abrahama Lincolna. Osoby dzielące te same zainteresowania można łatwo wyszukać w przypadku katalogu członków Ligi zapisanego w dokumencie procesora tekstów, wystarczy po prostu użyć funkcji *Znajdź*. Jednak wygenerowanie listy zawierającej *jedynie* znalezionych członków Ligi jest znacznie trudniejsze, ponieważ wymaga dużej liczby operacji „kopiuj i wklej”. Dzięki użyciu bazy danych MySQL wspomniane zadanie staje się znacznie łatwiejsze, ponieważ możemy wykonać zapytanie takie jak:

```
SELECT * FROM member WHERE interests LIKE '%lincoln%'
ORDER BY last_name, first_name
```

Niestety, wykonanie powyższego zapytania w kliencie *mysql* nie powoduje wygenerowania eleganckich danych wyjściowych. Utworzymy więc niewielki skrypt DBI o nazwie *interests.pl*, którego zadaniem będzie wyszukanie odpowiednich osób i przygotowanie czytelniejszych danych wyjściowych. Skrypt *interests.pl* na początku sprawdza, czy w wierszu poleceń został podany przynajmniej jeden argument. W przeciwnym razie nie ma nic do wyszukiwania. Następnie dla każdego podanego argumentu skrypt przeszukuje kolumnę *interests* tabeli *member*:

```
@ARGV or die "Użycie: interests.pl słowo kluczowe\n";
search_members (shift (@ARGV)) while @ARGV;
```

W celu wyszukania podanego przez użytkownika ciągu tekstowego słowa kluczowego po obu stronach zmiennej umieszczono znak wieloznaczny %, aby podany ciąg tekstowy wyszukać w dowolnym miejscu kolumny interests. Następnie dopasowane rekordy zostają wyświetlone:

```
sub search_members
{
    my $interest = shift;

    print "Wyniki wyszukiwania dla słowa kluczowego: $interest\n\n";
    my $sth = $dbh->prepare (qq{
        SELECT * FROM member WHERE interests LIKE ?
        ORDER BY last_name, first_name
    });
    # Wyszukanie ciągu tekstowego w dowolnym miejscu kolumny interest.
    $sth->execute ("% " . $interest . "%");
    my $count = 0;
    while (my $hash_ref = $sth->fetchrow_hashref ())
    {
        format_entry ($hash_ref);
        ++$count;
    }
    print "Liczba dopasowanych rekordów: $count\n\n";
}
```

Metoda `format_entry()` wyświetla każdy rekord w postaci łatwej do wydrukowania. W tym miejscu nie pokazano jej danych wyjściowych, ponieważ są praktycznie takie same jak generowane przez metodę `rtf_format_entry()` zaimplementowaną w skrypcie *gen_dir.pl*, ale pozbawione znaczników kontrolnych RTF. Implementację metody `format_entry()` możesz zobaczyć, analizując skrypt *interests.pl* umieszczony w dystrybucji *sampdb*.

8.3.5. Udostępnienie w internecie katalogu Ligi Historycznej

W podrozdziale 8.4, zatytułowanym „Użycie DBI w aplikacjach sieciowych”, rozpoczniemy tworzenie skryptów nawiązujących połączenie z bazą danych MySQL w celu wyodrębnienia informacji i przygotowania ich w formie stron internetowych wyświetlanych w przeglądarce internetowej klienta. Wspomniane skrypty dynamicznie generują kod HTML na podstawie żądań klienta. Zanim przystąpimy do ich tworzenia, zapoznajmy się z kodem HTML na podstawie skryptu DBI generującego statyczny dokument HTML, który można umieścić w drzewie dokumentów serwera WWW. Dobrym kandydatem dla takiego zadania jest przygotowanie katalogu Ligi Historycznej w formacie HTML (w końcu jednym z naszych celów było udostępnienie katalogu Ligi w internecie).

Prosty dokument HTML ma następującą strukturę:

<code><html></code>	Początek dokumentu.
<code><head></code>	Początek nagłówka dokumentu.
<code><title>Tytuł strony</title></code>	Tytuł dokumentu.
<code></head></code>	Koniec nagłówka dokumentu.
<code><body bgcolor="white"></code>	Początek treści dokumentu.
	(tło w kolorze białym)

```

<h1>Nagłówek pierwszego stopnia</h1>      Wyświetlenie nagłówka pierwszego stopnia.
... zawartość strony ...
</body>                                     Koniec treści dokumentu.
</html>                                     Koniec dokumentu.

```

Nie ma konieczności tworzenia zupełnie nowego skryptu w celu wygenerowania katalogu w formacie HTML. Przypomnij sobie, że podczas tworzenia skryptu *gen_dir.pl* przygotowaliśmy strukturę gotową do dalszej rozbudowy przez wstawianie kodu generującego katalog w kolejnych formatach. Dlatego też teraz wykorzystamy tę możliwość i dodamy kod generujący dane wyjściowe w formacie HTML.

Wymaga to wprowadzenia następujących zmian w skrypcie *gen_dir.pl*:

- Nowe metody inicjalizacji i czyszczenia dokumentu.
- Nowa metoda formatująca poszczególne rekordy.
- Nowy element tablicy identyfikujący nazwę formatu i wiążący tę nazwę z metodami generującymi dane wyjściowe we wskazanym formacie.

Przedstawiony wcześniej dokument HTML można łatwo podzielić na sekcje początkową i końcową obsługiwane przez metodę odpowiednio inicjalizującą i czyszczącą. Ponadto, dokument zawiera sekcję środkową, która będzie wygenerowana przez metodę formatującą dane. Metoda inicjalizująca generuje wszystko od początku dokumentu do początku treści strony, natomiast czyszcząca jest odpowiedzialna za wygenerowanie zamykających znaczników `</body>` i `</html>`.

```

sub html_init
{
    print "<html>\n";
    print "<head>\n";
    print "<title>Katalog członków Ligi Historycznej</title>\n";
    print "</head>\n";
    print "<body bgcolor=\"white\">\n";
    print "<h1>Katalog członków Ligi Historycznej</h1>\n";
}

sub html_cleanup
{
    print "</body>\n";
    print "</html>\n";
}

```

Rzeczywista praca jest jak zwykle wykonywana w metodzie formatującej wiersze danych wyjściowych. Tak naprawdę to nie jest aż tak trudne zadanie. Tworzymy kopię metody *rtf_format_entry()* i zmieniamy jej nazwę na *html_format_entry()*. Następnie modyfikujemy w taki sposób, aby wszystkie znaki specjalne były kodowane, a znaczniki kontrolne RTF zastępujemy znacznikami HTML:

```

sub html_format_entry
{
    my $entry_ref = shift;
    # Konwersja znaków &, ", > i < na odpowiadające im encje HTML,
    # (&quot;, &lt;, &gt;, &lt;).
    foreach my $key (keys (%{$entry_ref}))
    {

```

```

    next unless defined ($entry_ref->{$key});
    $entry_ref->{$key} =~ s/&/&amp;/g;
    $entry_ref->{$key} =~ s/"/&quot;/g;
    $entry_ref->{$key} =~ s/"/&quot;/g;
    $entry_ref->{$key} =~ s/</&lt;/g;
}
printf "<strong>Imię i nazwisko: %s</strong><br />\n", format_name ($entry_ref);
my $address = "";
$address .= $entry_ref->{street}
    if defined ($entry_ref->{street});
$address .= ", " . $entry_ref->{city}
    if defined ($entry_ref->{city});
$address .= ", " . $entry_ref->{state}
    if defined ($entry_ref->{state});
$address .= " " . $entry_ref->{zip}
    if defined ($entry_ref->{zip});
print "Adres: $address<br />\n";
    if $address ne "";
print "Telefon: $entry_ref->{phone}<br />\n";
    if defined ($entry_ref->{phone});
print "E-mail: $entry_ref->{email}<br />\n";
    if defined ($entry_ref->{email});
print "Zainteresowania: $entry_ref->{interests}<br />\n";
    if defined ($entry_ref->{interests});
print "<br />\n";
}

```

Powyższa metoda powoduje wygenerowanie danych wyjściowych w następującej postaci:

```

<strong>Imię i nazwisko: Mike Artel</strong><br />
Adres: 4264 Lovering Rd., Miami, FL 12777<br />
Telefon: 075-961-0712<br />
E-mail: mike_artel@venus.org<br />
Zainteresowania: prawa człowieka, edukacja, wojna o niepodległość<br />
<br />

```

Skrypt generuje znaczniki `
` zamiast `
`, aby zapewnić zgodność dokumentu z doskonale przygotowanym XHTML, który ma znacznie ściślejszą składnię niż HTML. W punkcie 8.4.2.2, zatytułowanym „Generowanie danych wyjściowych w postaci stron internetowych”, pokrótce przedstawiono różnice między HTML i XHTML.

Ostatnią zmianą wymaganą w skrypcie *gen_dir.pl* jest dodanie do tablicy `switchbox` elementu wskazującego metody obsługujące format HTML. Zmodyfikowaną tablicę przedstawiono poniżej. Ostatni element definiuje format o nazwie `html`, prowadzący do metod generujących katalog w formacie dokumentu HTML:

```

# Tablica switchbox wymieniająca metody formatowania dla poszczególnych formatów danych wyjściowych.
my %switchbox =
(
    "text" =>                                # Metody do obsługi formatu zwykłego tekstu.
    {
        "init"    => undef,                    # Inicjalizacja nie jest wymagana.
        "entry"   => \%text_format_entry,
        "cleanup" => undef                      # Operacje czyszczące nie są wymagane.
    },
    "rtf" =>                                    # Metody do obsługi formatu RTF.

```

```

{
    "init"    => \&rtf_init,
    "entry"   => \&rtf_format_entry,
    "cleanup" => \&rtf_cleanup
},
"html" =>                                     # Metody do obsługi formatu HTML.
{
    "init"    => \&html_init,
    "entry"   => \&html_format_entry,
    "cleanup" => \&html_cleanup
}
};

```

Aby wygenerować katalog w formacie HTML, trzeba wydać poniższe polecenie, a następnie otrzymany plik *directory.html* umieścić w katalogu dokumentów serwera WWW:

```
% ./gen_dir.pl html > directory.html
```

Po każdym uaktualnieniu tabeli member w bazie danych możesz ponownie wykonać powyższe polecenie i uaktualnić katalog dostępny w internecie. W celu uniknięcia konieczności ręcznego wydawania powyższego polecenia inną strategią jest zdefiniowanie zadania, które w określonych odstępach czasu automatycznie uaktualnia internetową wersję katalogu. W systemach UNIX można do tego wykorzystać mechanizm cron. Przyjmujemy założenie, że skrypt *gen_dir.pl* znajduje się w katalogu */usr/local/bin*, natomiast katalogiem Ligi Historycznej w serwerze WWW jest */usr/local/apache/htdocs/ushl*. Wpis cron używany do uaktualnienia katalogu każdego dnia o godzinie 04:00 przedstawia się następująco (całe polecenie musi być wpisane w pojedynczym wierszu):

```
0 4 * * * /usr/local/bin/gen_dir.pl
> /usr/local/apache/htdocs/ushl/directory.html
```

Użytkownik uruchamiający to zadanie cron musi mieć uprawnienia zapisu plików w katalogu dokumentów serwera WWW.

8.4. Użycie DBI w aplikacjach sieciowych

Opracowane dotąd skrypty były przeznaczone dla środowiska wiersza poleceń. DBI jest użyteczne także w innych kontekstach, na przykład w trakcie programowania aplikacji sieciowych. Kiedy stworzysz skrypty DBI, które mogą być wywoływane przez serwer WWW w odpowiedzi na żądania wysyłane przez przeglądarki internetowe, tym samym otwierasz przed użytkownikami nowe interesujące możliwości pracy z bazami danych. Na przykład, jeśli stworzysz skrypt wyświetlający dane w postaci tabeli, każdy nagłówek kolumny można bardzo łatwo zamienić na łącze, którego kliknięcie powoduje ponowne sortowanie danych w tej tabeli. W ten sposób za pomocą tylko jednego kliknięcia użytkownik może wyświetlać dane na różne sposoby bez konieczności wprowadzania jakichkolwiek zapytań. Ewentualnie można dostarczyć formularz, w którym użytkownik podaje kryteria wyszukiwania bazy danych, a następnie wyświetlana jest strona zawierająca wyniki danego wyszukiwania.

Wymienione drobne możliwości znacznie zmieniają poziom interaktywności w trakcie dostępu do zawartości baz danych. Ponadto, możliwości przeglądarek internetowych w zakresie wyświetlania danych są znacznie większe niż okna terminala, co pozwala na osiągnięcie ładnie sformatowanych i przyjemnie wyglądających danych wyjściowych.

W tym podrozdziale przygotowujemy następujące skrypty:

- Ogólna przeglądarka tabel znajdujących się w bazie danych sampdb. Ten skrypt nie jest powiązany z żadnym konkretnym zadaniem do wykonania w bazie danych, ale pokazuje kilka koncepcji programowania sieciowego i zapewnia wygodny sposób przeglądania informacji znajdujących się w tabelach.
- Przeglądarka wyników uczniów, pozwalająca na przeglądanie wyników poszczególnych sprawdzianów i testów. To jest użyteczny skrypt, umożliwiający szybkie przeglądanie wyników w projekcie ocen uczniów. Szczególnie przydatny staje się podczas ustalania poziomu trudności testu.
- Skrypt wyszukujący członków Ligi Historycznej o takich samych zainteresowaniach. Odbyna się to przez umożliwienie użytkownikowi podania szukanego wyrażenia, a następnie przeszukanie pod jego kątem kolumny `interests` tabeli `member`. Do tego celu w punkcie 8.3.4, zatytułowanym „Wyszukiwanie członków Ligi Historycznej o takich samych zainteresowaniach”, utworzyliśmy skrypt `interests.pl` działający w wierszu poleceń. Jednak wspomniany skrypt wiersza poleceń może być uruchamiany tylko przez użytkowników posiadających konto w systemie, w którym zainstalowano skrypt. Dostarczenie sieciowej wersji skryptu pozwala na jego wykorzystywanie przez każdego użytkownika mającego dostęp do przeglądarki internetowej. Ponadto, inna wersja skryptu wykonującego to samo zadanie ma walory edukacyjne i pozwala na porównanie różnych podejść stosowanych do rozwiązania danego problemu. (W rzeczywistości opracujemy dwie implementacje sieciowe. Jedna jest oparta na dopasowaniu wzorca, podobnie jak w skrypcie `interests.pl`. Z kolei druga przeprowadza wyszukiwanie pełnego tekstu).

Wymienione skrypty używają modułu `CGI.pm` Perla, zapewniającego łatwy sposób połączenia DBI z siecią. (Informacje dotyczące pobrania modułu `CGI.pm` znajdziesz w dodatku A, zatytułowanym „Oprogramowanie wymagane do użycia tej książki”). Nazwa modułu wzięła się stąd, że pomaga w tworzeniu skryptów używających protokołu Common Gateway Interface, definiującego sposób, w jaki serwer WWW komunikuje się z innymi programami. Moduł `CGI.pm` zajmuje się wszelkimi szczegółami dotyczącymi obsługi zadań czyszczących, takich jak pobieranie wartości parametrów przekazanych serwerowi WWW przez skrypt jako dane wejściowe. Ponadto, moduł `CGI.pm` oferuje wygodne metody generowania danych wyjściowych w formacie HTML, które zmniejszają niebezpieczeństwo wygenerowania nieprawidłowych znaczników HTML, co się może zdarzyć, jeśli zajmujesz się tym samodzielnie.

W tym rozdziale dowiesz się wystarczająco dużo o module `CGI.pm`, aby tworzyć własne aplikacje sieciowe. Oczywiście, nie zostaną omówione wszystkie możliwości oferowane przez wymieniony moduł. Aby dowiedzieć się więcej na jego temat, zapoznaj się

z książką *Official Guide to Programming with CGI.pm* Lincolna Steina (wydawnictwo Wiley, 1998) lub wyszukaj słowo kluczowe CGI na stronie <http://cpan.perl.org/> i zapoznaj się z dostępną tam dokumentacją.

Inny tekst omawiający moduł CGI.pm, a szczególnie współpracę z MySQL i DBI, to książka *MySQL and Perl for the Web* (wydawnictwo New Riders, 2000).

Skrypty przedstawione w pozostałej części rozdziału znajdziesz w katalogu *perlapi/web* dystrybucji *sampledb*.

8.4.1. Konfiguracja Apache w celu obsługi skryptów CGI

Podczas tworzenia skryptów sieciowych poza DBI i CGI.pm potrzebujemy jeszcze jednego komponentu, jakim jest serwer WWW. Przedstawione tutaj informacje dotyczą skryptów uruchamianych w serwerze WWW Apache, ale na ich podstawie możesz być w stanie wykorzystać także inny serwer WWW.

Przyjmujemy założenie, że serwer Apache został zainstalowany w katalogu */usr/local/apache* (UNIX) lub *C:\Apache* (Windows). Na nasze potrzeby najważniejsze podkatalogi instalacji to *htdocs* (zawiera dokumenty HTML), *cgi-bin* (zawiera skrypty oraz programy wywoływane przez serwer WWW) i *conf* (zawiera pliki konfiguracyjne). Wymienione katalogi mogą znajdować się w innych miejscach w Twoim systemie operacyjnym. Jeżeli tak jest, musisz odpowiednio dostosować przedstawione tutaj skrypty.

Powinieneś się upewnić, że katalog *cgi-bin* nie znajduje się w drzewie dokumentów Apache. To jest bezpieczne założenie, ponieważ uniemożliwia klientom żądanie kodu źródłowego skryptów jako zwykłego tekstu. Nie chcesz, aby złośliwy użytkownik miał możliwość przeprowadzenia analizy skryptów w poszukiwaniu ich słabych punktów.

W celu instalacji skryptu CGI przeznaczonego do użycia w serwerze Apache po prostu skopiuj go do katalogu *cgi-bin*. W systemach UNIX skrypt musi rozpoczynać się od wiersza *#!*, a sam plik skryptu mieć nadane uprawnienia wykonywania, podobnie jak w przypadku innych skryptów wiersza poleceń.

```
# chown www myscrip.pl
# chmod u=rx,go-rwx myscrip.pl
```

Wykonanie powyższych poleceń może wymagać uprawnień użytkownika root. Jeżeli nie masz uprawnień do instalacji skryptów w katalogu *cgi-bin*, poproś administratora systemu, aby to zrobił w Twoim imieniu.

W systemie Windows polecenia *chown* i *chmod* są niepotrzebne, ale skrypt nadal powinien rozpoczynać się od wiersza *#!*. Wspomniany wiersz powinien zawierać pełną ścieżkę dostępu do programu Perl. Na przykład, jeśli Perl jest zainstalowany jako *C:\Perl\bin\perl.exe*, wtedy wiersz *#!* należy zapisać w postaci:

```
#!C:/Perl/bin/perl
```

Alternatywnie, w systemie Windows można użyć poniższego wiersza, jeśli zmienna środowiskowa *PATH* zawiera katalog, w którym jest zainstalowany Perl:

```
#!perl
```


Skrypty Perl w dystrybucji *sampdb* w wierszu *#!* zawierają ścieżkę dostępu Perl w postaci */usr/bin/perl*. Jeśli jest to konieczne, zmodyfikuj skrypty, aby zawierały ścieżkę dostępu odpowiednią dla używanego przez Ciebie systemu.

Po zainstalowaniu skryptu w katalogu *cgi-bin* można go uruchomić z poziomu przeglądarki internetowej przez wysłanie odpowiedniego adresu URL do serwera WWW. Na przykład, jeśli serwer WWW działa w komputerze lokalnym, uruchomienie skryptu o nazwie *myscript.pl* odbywa się za pomocą poniższego adresu URL:

```
http://localhost/cgi-bin/myscript.pl
```

Pamiętaj o zmianie adresów URL w tym rozdziale, aby prowadziły do używanego przez Ciebie serwera WWW zamiast do *localhost*.

Żądanie skryptu przez przeglądarkę internetową powoduje jego uruchomienie przez serwer WWW. Dane wyjściowe skryptu są przekazywane z powrotem, a wynik wyświetlany w postaci strony internetowej w przeglądarce.

Kiedy skrypty DBI uruchamiasz z poziomu wiersza poleceń, ostrzeżenia i komunikaty błędów są wyświetlane w terminalu. W środowisku sieciowym nie ma terminala, więc wspomniane komunikaty są umieszczane w dzienniku błędów serwera Apache. Powinieneś ustalić położenie wspomnianego dziennika błędów, ponieważ dostarcza on wielu cennych informacji podczas usuwania błędów ze skryptów. W moim systemie plik *error_log* znajduje się w podkatalogu *logs* katalogu głównego Apache (*/usr/local/apache*). W Twoim systemie może znajdować się w zupełnie innym miejscu. Położenie wspomnianego dziennika błędów określa dyrektywa *ErrorLog* w pliku konfiguracyjnym *httpd.conf*, który znajdziesz w katalogu *conf* serwera Apache.

8.4.2. Krótki opis modułu CGI.pm

W celu utworzenia skryptu Perl używającego modułu *CGI.pm* na początku skryptu należy umieścić polecenie *use CGI* importujące nazwy funkcji modułu. Standardowy zestaw najczęściej używanych funkcji można zaimportować w następujący sposób:

```
use CGI qw(:standard);
```

Teraz można już wywołać różne funkcje modułu *CGI.pm* odpowiedzialne za generowanie różnego rodzaju struktur HTML. Ogólnie rzecz biorąc, funkcje mają nazwy takie jak odpowiadające im elementy HTML. Na przykład, w celu wygenerowania nagłówka pierwszego poziomu oraz akapitu należy wywołać funkcje *h1()* i *p()*:

```
print h1 ("To jest nagłówek");  
print p ("To jest akapit");
```

Moduł *CGI.pm* obsługuje także styl programowania zorientowanego obiektowo, co pozwala na wywoływanie jego funkcji bez konieczności importu ich nazw. Aby skorzystać z takiej możliwości, użyj polecenia *use* i utwórz obiekt *CGI*:

```
use CGI;  
my $cgi = new CGI;
```

Obiekt daje dostęp do funkcji modułu CGI.pm, które są wywoływane jako metody obiektu:

```
print $cgi->h1 ("To jest nagłówek");
print $cgi->p ("To jest akapit");
```

Interfejs zorientowany obiektowo wymaga nieustannego używania prefiksu `$cgi->`; w tej książce stosowany jest prostszy interfejs wywołań funkcji. Jednak jedną z wad interfejsu wywołań funkcji jest to, że jeśli funkcja modułu CGI.pm ma taką samą nazwę jak wbudowana funkcja Perl, wtedy trzeba ją wywołać w sposób unikający konfliktu. Na przykład, moduł CGI.pm ma funkcję o nazwie `tr()` generującą znaczniki `<tr>` i `</tr>` otaczające komórki w wierszu tabeli HTML. Wymieniona nazwa funkcji jest taka sama jak wbudowanej funkcji Perl: `tr()`. Rozwiązanie tego problemu podczas używania interfejsu wywołań funkcji CGI.pm jest następujące: wywołaj `tr()` jako `Tr()` lub `TR()`. Ten problem nie występuje w interfejsie zorientowanym obiektowo, ponieważ funkcja `tr()` jest wywoływana jako metoda obiektu `$cgi` (czyli `$cgi->tr()`). Dzięki temu jest jasne, że wywołanie nie dotyczy wbudowanej funkcji Perla.

8.4.2.1. Sprawdzanie parametrów danych wejściowych

Jednym z zadań, którymi moduł CGI.pm zajmuje się za programistę, jest obsługa szczegółów związanych z pobieraniem danych wejściowych dostarczanych skryptowi przez serwer WWW. Aby pobrać wspomniane informacje, musisz jedynie wywołać funkcję `param()`. Pobranie tablicy nazw wszystkich dostępnych parametrów następuje po wydaniu poniższego polecenia:

```
my @param = param ();
```

W celu pobrania wartości konkretnego parametru przekaz jego nazwę funkcji `param()`. Jeżeli parametr jest ustawiony, funkcja `param()` zwróci jego wartość. W przeciwnym razie wartością zwrótną będzie `undef`:

```
my $my_param = param ("my_param");
print "my_param value: ", (defined ($my_param) ? $my_param : "not set"), "\n";
```

8.4.2.2. Generowanie danych wyjściowych w postaci stron internetowych

Wiele funkcji modułu CGI.pm generuje dane wyjściowe wysyłane do przeglądarki internetowej klienta. Spójrz na poniższy dokument HTML:

```
<html>
<head>
<title>Moja prosta strona</title>
</head>
<body bgcolor="white">
<h1>Nagłówek strony</h1>
<p>Akapit 1.</p>
<p>Akapit 2.</p>
</body>
</html>
```

Przedstawiony poniżej skrypt używa oferowanych przez moduł CGI.pm funkcji generowania danych wyjściowych do przygotowania odpowiadającego mu dokumentu:

```
#!/usr/bin/perl
# simple_doc.pl - Wygenerowanie prostej strony HTML.

use strict;
use warnings;
use CGI qw(:standard);

print header ();
print start_html (-title => "Moja prosta strona", -bgcolor => "white");
print h1 ("Nagłówek strony");
print p ("Akapit 1.");
print p ("Akapit 2.");
print end_html ();
```

Funkcja `header()` generuje nagłówek `Content-Type`: poprzedzający zawartość strony. Podanie wymienionego nagłówka jest konieczne podczas generowania stron internetowych na podstawie skryptów, aby przeglądarka internetowa wiedziała, jakiego rodzaju dokumentu może oczekiwać. (To różnica w stosunku do sposobu tworzenia statycznych stron HTML. W przypadku wspomnianych stron statycznych nagłówków nie jest konieczny, ponieważ serwer WWW automatycznie wysyła go przeglądarce internetowej). Domyślnie funkcja `header()` generuje następujący nagłówek:

```
Content-Type: text/html
```

Po wywołaniu funkcji `header()` mamy wywołania funkcji odpowiedzialnych za wygenerowanie treści strony. Funkcja `start_html()` generuje znaczniki, począwszy od otwierającego `<html>` aż do otwierającego `<body>`, funkcje `h1()` i `p()` generują odpowiednio nagłówki i akapit, natomiast funkcja `end_html()` generuje zamykające znaczniki dokumentu.

Jak przedstawiono w wywołaniu funkcji `start_html()`, wiele funkcji modułu CGI.pm pozwala na stosowanie parametrów w formie `-nazwa=>wartość`. To ma ogromną zaletę w przypadku funkcji pobierających wiele opcjonalnych parametrów, ponieważ można wymienić tylko wymagane, a na dodatek mogą być wymienione w dowolnej kolejności.

Użycie oferowanych przez moduł CGI.pm funkcji generowania danych wyjściowych nie uniemożliwia samodzielnego tworzenia kodu HTML. Oba wymienione podejścia można ze sobą łączyć, podobnie jak wywołania funkcji modułu CGI.pm z poleceniami `print` generującymi dosłowne znaczniki. Jednak jedną z zalet używania funkcji modułu CGI.pm jest możliwość pracy w kategoriach jednostek logicznych zamiast poszczególnych znaczników, a wynikowy dokument HTML prawdopodobnie będzie zawierał mniejszą liczbę błędów. (Użyłem określenia „prawdopodobnie”, ponieważ moduł CGI.pm nie zabrania wykonywania dziwnych operacji, takich jak wygenerowanie listy w nagłówku).

Moduł CGI.pm zapewnia także pewien poziom przenośności, którego nie masz, samodzielnie tworząc polecenia generujące kod HTML. Na przykład, począwszy od wersji 2.69, moduł CGI.pm automatycznie tworzy dane wyjściowe w formacie XHTML. Jeżeli używasz starszej wersji modułu, generującej zwykły kod HTML, to musisz jedynie uaktualnić moduł, a skrypt automatycznie będzie generował dane wyjściowe w postaci XHTML.

Kod XHTML jest podobny do HTML, ale charakteryzuje się ściślejszym zdefiniowanym formatem. Język HTML jest łatwy do nauki oraz użycia, choć jego największy problem polega na tym, że implementacje w poszczególnych przeglądarkach internetowych mogą inaczej interpretować kod HTML.

Na przykład, niektóre wybaczą błędnie przygotowany kod HTML. Oznacza to, że niezbyt dobrze utworzony kod HTML będzie przez jedną przeglądarkę internetową wyświetlony poprawnie, natomiast przez inną już nie. Wymagania narzucane przez XHTML są ściślejsze i mają na celu zapewnienie lepszego przygotowywania dokumentów. Poniżej wymieniono pewne różnice między HTML i XHTML:

- W przeciwieństwie do HTML, każdemu otwierającemu znacznikowi w XHTML musi odpowiadać znacznik zamykający. Na przykład, akapity są tworzone parą znaczników `<p>` i `</p>`, natomiast w HTML znacznik zamykający `</p>` jest często pomijany. W XHTML znacznik `</p>` jest wymagany. W przypadku znaczników HTML nieposiadających żadnej treści, na przykład `
` i `<hr>`, narzucany przez XHTML wymóg stosowania znaczników zamykających oznaczałby tworzenie konstrukcji w stylu `
</br>` i `<hr></hr>`. Aby tego uniknąć, XHTML pozwala na stosowanie skróconej formy w postaci pojedynczego znacznika (`
` i `<hr/>`), który służy w charakterze znacznika jednocześnie otwierającego i zamykającego. Jednak starsze przeglądarki internetowe napotykające tego rodzaju znaczniki czasami błędnie interpretują ich nazwy jako `br/` i `hr/`. Wstawienie spacji między nazwą znacznika i ukośnik (`
` i `<hr />`) pozwala zminimalizować ryzyko wystąpienia takiego problemu.
- W HTML nazwy znacznika i atrybutu nie rozróżniają wielkości liter. Na przykład, `<BODY BGCOLOR="white">` i `<body bgcolor="white">` oznacza to samo. Z kolei w XHTML nazwy znaczników i atrybutów muszą być pisane jedynie małymi literami, a więc poprawna jest tylko forma `<body bgcolor="white">`.
- Wartości atrybutów w HTML nie muszą być cytowane, a nawet może ich brakować. Na przykład, poniższe polecenie tworzące komórkę tabeli jest poprawnym kodem HTML:

```
<td width=40 nowrap>Pewien tekst</td>
```

W XHTML atrybuty muszą mieć cytowane wartości. Powszechnie stosowaną konwencją w przypadku atrybutów normalnie pozbawionych wartości w HTML jest użycie nazwy atrybutu jako jego wartości. Odpowiednik XHTML powyższego elementu `<tr>` przedstawia się następująco:

```
<td width="40" nowrap="nowrap">Pewien tekst</td>
```

Wszystkie skrypty przedstawione w książce generują dane wyjściowe zgodne z regułami XHTML. W tym rozdziale wykorzystujemy moduł `CGI.pm` do wygenerowania prawidłowych znaczników XHTML. Skrypty przedstawione w rozdziale 9., zatytułowanym „Tworzenie programów MySQL przy użyciu języka PHP”, również generują XHTML, choć odbywa się to w inny sposób, ponieważ PHP nie oferuje takich funkcji generowania znaczników, jakie znajdziemy w module `CGI.pm`.

8.4.2.3. Kodowanie znaków w tekście HTML i adresach URL

Jeżeli tekst przeznaczony do wyświetlenia na stronie internetowej zawiera znaki specjalne, należy się upewnić o ich prawidłowym kodowaniu, przetwarzając wspomniany tekst przez funkcję `escapeHTML()`. Dotyczy to również adresów URL zawierających znaki specjalne, choć w tym przypadku stosowana jest funkcja `escape()`. Bardzo ważne jest użycie odpowiedniej funkcji kodującej, ponieważ każda z nich rozpoznaje inny zestaw znaków specjalnych i inaczej je koduje. Funkcja `escapeHTML()` koduje znaki specjalne na postać odpowiadających im encji HTML. Na przykład, znak `<` staje się encją `<`. Z kolei funkcja `escape()` koduje znak specjalny na postać znaku `%` i dwóch cyfr szesnastkowych przedstawiających liczbowy kod znaku. Dlatego też znak specjalny `<` zostaje zakodowany na postać `%3C`. Przeanalizuj poniższy krótki skrypt Perla o nazwie `escape_demo.pl`, który pokazuje obie formy kodowania znaków:

```
#!/usr/bin/perl
# escape_demo.pl - Demonstracja oferowanych przez moduł CGI.pm funkcji kodowania znaków w danych
# wyjściowych.

use strict;
use warnings;
use CGI qw(escapeHTML escape); # Import funkcji escapeHTML() i escape().

# Przypisanie domyślnego ciągu tekstowego, ale użyty będzie argument wiersza poleceń, o ile zostanie podany.
my $s = "1<=2, prawda?";
$s = shift (@ARGV) if @ARGV;
print "Niezakodowany ciąg tekstowy:      ", $s, "\n";
print "Zakodowany do użycia jako tekst HTML: ", escapeHTML($s), "\n";
print "Zakodowany do użycia w adresie URL:   ", escape($s), "\n";
```

Skrypt koduje ciąg tekstowy `$s` za pomocą obu funkcji, a następnie wyświetla wynik. Po jego uruchomieniu skrypt wygeneruje następujące dane wyjściowe, na podstawie których możesz się przekonać, że konwencje kodowania znaków specjalnych dla tekstu HTML nie są takie same jak dla adresu URL:

```
Niezakodowany ciąg tekstowy:      1<=2, prawda?
Zakodowany do użycia jako tekst HTML: 1&lt;=2, prawda?
Zakodowany do użycia w adresie URL:  1%3C%3D2%2C%20prawda%3F
```

Jeżeli skryptowi `escape_demo.pl` podasz argument wiersza poleceń, skrypt powoduje zakodowanie argumentu zamiast domyślnego ciągu tekstowego. W ten sposób możesz przekonać się, jak będzie wyglądał zakodowany dowolny ciąg tekstowy.

W poleceniu `use CGI` skrypt `escape_demo.pl` importuje nazwy funkcji kodujących. W zależności od używanej wersji modułu `CGI.pm`, mogą, choć nie muszą się one znajdować w domyślnym zestawie funkcji. W tym drugim przypadku może wystąpić konieczność ich importu, pomimo wcześniejszego importu zestawu funkcji standardowych:

```
use CGI qw(:standard escapeHTML escape);
```

8.4.2.4. Tworzenie stron o wielu przeznaczeniach

Jednym z podstawowych powodów tworzenia skryptów sieciowych generujących dokumenty HTML zamiast przygotowywania statycznych dokumentów HTML jest to, że skrypt potrafi przygotować różnego rodzaju strony w zależności od sposobu jego wywołania. Wszystkie skrypty CGI, które przygotujemy, będą miały tę właściwość.

Skrypty działają w następujący sposób:

- W trakcie pierwszego żądania skryptu przez przeglądarkę internetową następuje wygenerowanie strony początkowej, pozwalającej użytkownikowi na wybór rodzaju informacji, które chciałby otrzymać.
- Po dokonaniu wyboru przeglądarka internetowa wysyła żądanie do serwera WWW, co powoduje ponowne wywołanie skryptu. Tym razem skrypt pobiera i wyświetla drugą stronę, zawierającą informacje żądane przez użytkownika.

Problem związany z przedstawionym podejściem polega na tym, że chcemy, aby formularz na pierwszej stronie określał rodzaj treści wyświetlanej na drugiej stronie, ale obie strony internetowe pozostają niezależne od siebie, o ile nie zastosujemy specjalnych rozwiązań. Jednym z rozwiązań jest skrypt generujący strony ustawiające parametrowi wartość wskazującą następnemu wywołaniu skryptu żadaną treść. Podczas pierwszego wywołania skryptu parametr nie ma wartości, a więc skrypt wygeneruje stronę początkową. Po wybraniu rodzaju żądanych informacji skrypt zostaje wywołany ponownie, ale tym razem parametr ma przypisaną wartość, która wskazuje skryptowi rodzaj podejmowanej operacji.

Istnieją różne sposoby, na jakie strony internetowe mogą przekazywać informacje do skryptów. Jednym z rozwiązań jest umieszczenie na stronie formularza wypełnianego przez użytkownika. Kiedy użytkownik wyśle formularz, jego zawartość zostaje przekazana do serwera WWW. Następnie serwer przekazuje te informacje skryptowi, który odczytuje parametry za pomocą wywołania funkcji `param()`. W taki sposób zaimplementowaliśmy wyszukiwanie słów kluczowych w katalogu Ligi Historycznej: strona wyszukiwania zawierała formularz, w którym użytkownik podawał szukane słowa kluczowe.

Innym sposobem przekazania informacji do skryptu jest dodanie wartości parametrów na końcu adresu URL wysłanego do serwera WWW w celu żądania skryptu. Takie podejście zastosujemy w skryptach przeglądarki tabel bazy danych `sampdb` oraz ocen uczniów. Działanie tego sposobu polega na tym, że skrypt generuje stronę zawierającą łącza. Po kliknięciu łącza następuje ponowne wywołanie skryptu, ale łącze zawiera wartość parametru wskazującą skryptowi odpowiednie działanie. W efekcie skrypt wywołany na różne sposoby oferuje odmienne rodzaje wyniku działania, w zależności od wybranego łącza.

Skrypt może umożliwić wywołanie samego siebie, co następuje przez wysłanie przeglądarce internetowej strony zawierającej łącze do skryptu. Na przykład, jeżeli skrypt o nazwie `myscript.pl` zostanie zainstalowany w katalogu `cgi-bin` serwera WWW, wtedy może wygenerować stronę zawierającą poniższe łącze:

```
<a href="/cgi-bin/myscript.pl">Kliknij mnie!</a>
```

Kiedy użytkownik kliknie tekst *Kliknij mnie* wyświetlony na stronie, przeglądarka internetowa użytkownika wyśle żądanie do skryptu *myscript.pl*. To, oczywiście, spowoduje ponowne wyświetlenie tej samej strony, ponieważ w adresie URL nie znajdują się żadne inne informacje. Jednak po dodaniu parametru do adresu URL będzie on wysłany do serwera WWW, gdy użytkownik kliknie łącze. Serwer wywoła skrypt, który za pomocą funkcji `param()` może wykryć ustawienie parametru i podjąć odpowiednie działanie zgodnie z wartością parametru.

W celu dodania parametru na końcu adresu URL umieść znak zapytania, a po nim parę *nazwa=wartość* wskazującą nazwę parametru i jego wartość. Na przykład, aby dodać parametr `size` wraz z wartością `large`, należy adres URL zapisać w następującej postaci:

```
/cgi-bin/myscript.pl?size=large
```

Aby dodać wiele parametrów, trzeba je rozdzielić średnikami (;) lub znakami ampersand (&):

```
/cgi-bin/myscript.pl?size=large;color=blue
```

Moduł `CGI.pm` rozpoznaje oba wymienione znaki jako separatory parametrów. W API innych języków programowania sieciowego mogą być stosowane inne konwencje. Dlatego też musisz wiedzieć, jakiego oczekują znaku rozdzielającego, i odpowiednio przygotować adresy URL. W tym rozdziale stosujemy średnik.

Aby przygotować odwołujący się do samego siebie adres URL wraz z dołączonymi parametrami, skrypt powinien rozpoczynać się od wywołania funkcji modułu `CGI.pm` o nazwie `url()`, która pobiera własny adres URL. Następnie można dołączyć parametry, na przykład w poniższy sposób:

```
$url = url ();           # Pobranie adresu URL skryptu.  
$url .= "?size=large";   # Dodanie pierwszego parametru.  
$url .= ";color=blue";   # Dodanie drugiego parametru.
```

Użycie funkcji `url()` w celu pobrania ścieżki dostępu skryptu pozwala uniknąć jej zdefiniowania na stałe w kodzie skryptu.

W celu wygenerowania łącza należy adres URL przekazać funkcji modułu `CGI.pm` o nazwie `a()`:

```
print a ({-href => $url}, "Kliknij mnie!");
```

Wygenerowane polecenie tworzące łącze przedstawia się następująco:

```
<a href="/cgi-bin/url.pl?size=large;color=blue">Kliknij mnie!</a>
```

Poprzednie przykłady tworzą wartość zmiennej `$url` w nieco nonszalancki sposób, ponieważ nie biorą pod uwagę możliwości, że wartość parametru lub etykieta łącza mogą zawierać znaki specjalne. O ile jesteś absolutnie przekonany, że wartości parametru i etykieta nie wymagają kodowania, to najlepiej zastosować funkcje kodowania dostępne w module `CGI.pm`. Funkcja `escape()` koduje wartości dołączane do adresu URL, natomiast `escapeHTML()` koduje zwykły tekst HTML. Na przykład, jeśli tekst etykiety łącza jest przechowywany w zmiennej `$label`, a wartości parametrów `size` i `color` w zmiennych `$size` i `$color`, wtedy prawidłowe kodowanie jest przeprowadzane następująco:

```
$url = sprintf ("%s?size=%s;color=%s",
                url (), escape ($size), escape ($color));
print a ({-href => $url}, escapeHTML ($label));
```

Aby zobaczyć, jak konstrukcja skryptu odwołującego się do samego siebie działa w kontekście aplikacji, przeanalizuj poniższy krótki skrypt o nazwie *flip_flop.pl*. Po jego pierwszym wywołaniu wyświetla stronę nazwaną A, zawierającą pojedyncze łącze. Kliknięcie łącza ponownie wywołuje skrypt, ale wspomniane łącze zawiera parametr *pageb*, nakazujący skryptowi *flip_flop.pl* wyświetlenie strony B. (W omawianym przykładzie wartość parametru nie ma znaczenia, wystarczy jego ustawienie). Strona B również zawiera łącze do skryptu, ale bez parametru *pageb*. Oznacza to, że kliknięcie łącza na stronie B powoduje ponowne wyświetlenie strony początkowej, czyli A. Innymi słowy, kolejne wywołania skryptu na przemian wyświetlają strony A i B:

```
#!/usr/bin/perl
#flip_flop.pl - Prosty skrypt CGI.pm generujący wiele stron.

use strict;
use warnings;
use CGI qw(:standard);

my $url;
my $this_page;
my $next_page;

# Określenie strony do wyświetlenia na podstawie
# obecności lub braku parametru pageb.
if (!defined (param ("pageb"))) # Wyświetlenie strony A wraz z łączem do strony B.
{
    $this_page = "A";
    $next_page = "B";
    $url = url () . "?pageb=1";
}
else # Wyświetlenie strony B wraz z łączem do strony A.
{
    $this_page = "B";
    $next_page = "A";
    $url = url ();
}

print header ();
print start_html (-title => "Flip-Flop: Page $this_page",
                  -bgcolor => "white");
print p ("To jest strona $this_page. Aby wyświetlić stronę $next_page, "
        . a ({-href => $url}, "kliknij tutaj"));
print end_html ();
```

Zainstaluj skrypt w katalogu *cgi-bin*, załaduj jego uruchomienia z poziomu przeglądarki internetowej, podając poniższy adres, ale zamiast *localhost* podaj adres używanego serwera WWW:

```
http://localhost/cgi-bin/flip_flop.pl
```

Kliknij łącze kilkakrotnie i przekonaj się, jak skrypt zmienia generowane strony.

Teraz przyjmujemy założenie, że inny klient zażądał uruchomienia skryptu *flip_flop.pl*. Co się stanie? Czy dwa egzemplarze uruchomionego tego samego skryptu będą ze sobą oddziaływały? Nie, ponieważ początkowe żądanie pochodzące od każdego klienta nie zawiera parametru *pageb* i skrypt odpowiada wygenerowaniem strony początkowej. Dlatego też żądania wysyłane przez klientów zawierają parametr lub go nie zawierają, zależnie od aktualnie wyświetlanej strony. Serię naprzemiennych stron skrypt *flip_flop.pl* generuje prawidłowo dla każdego klienta i niezależnie od akcji podejmowanych przez poszczególne klienty.

8.4.3. Nawiązanie połączenia z serwerem MySQL z poziomu skryptu sieciowego

Opracowane wcześniej w podrozdziale 8.3, zatytułowanym „Praca z DBI”, skrypty wiersza poleceń korzystały z tego samego kodu odpowiedzialnego za nawiązanie połączenia z serwerem MySQL. Większość naszych skryptów CGI również współdzieli ten sam kod, ale jest on nieco inny:

```
#!/usr/bin/perl

use strict;
use warnings;
use DBI;
use CGI qw(:standard);
use Cwd;

# Plik opcji, który w systemie UNIX powinien zawierać parametry połączenia.
my $option_file = "/usr/local/apache/conf/sampdb.cnf";
my $option_drive_root;
# Nadpisanie położenia pliku w przypadku systemu Windows.
if ($^O =~ /^MSWin/i || $^O =~ /^dos/)
{
    $option_drive_root = "C:/";
    $option_file = "/Apache/conf/sampdb.cnf";
}

# Przygotowanie źródła danych i nawiązanie połączenia z serwerem (w systemie Windows trzeba
# najpierw zachować nazwę bieżącego katalogu roboczego, przejść do dysku zawierającego plik
# opcji, nawiązać połączenie, a następnie powrócić do zapisanego katalogu bieżącego).
my $orig_dir;
if (defined ($option_drive_root))
{
    $orig_dir = cwd ();
    chdir ($option_drive_root)
        or die "Nie można przejść do dysku $option_drive_root: $!\n";
}
my $dsn = "DBI:mysql:sampdb:mysql_read_default_file=$option_file";
my %conn_attrs = (RaiseError => 1, PrintError => 0, AutoCommit => 1);
my $dbh = DBI->connect ($dsn, undef, undef, \%conn_attrs);
if (defined ($option_drive_root))
{
    chdir ($orig_dir)
        or die "Nie można przejść do katalogu $orig_dir: $!\n";
}
```

W porównaniu do skryptów uruchamianych w wierszu poleceń powyższy fragment kodu różni się pod następującymi względami:

- Pierwsza sekcja zawiera polecenia `use CGI` i `use Cwd`. Pierwsze jest przeznaczone dla modułu `CGI.pm`. Z kolei drugie jest przeznaczone dla modułu zwracającego ścieżkę dostępu do aktualnego katalogu roboczego. Ten moduł jest używany w przypadku, gdy skrypt zostanie uruchomiony w systemie Windows (patrz opis w dalszej części rozdziału).
- Żadne parametry połączenia nie są przetwarzane na podstawie argumentów wiersza poleceń. Zamiast tego w kodzie przyjęto założenie, że wspomniane parametry będą wymienione w pliku opcji.
- Zamiast użycia `mysql_read_default_group` do odczytania standardowych plików opcji używamy `mysql_read_default_file` w celu odczytania pojedynczego pliku przeznaczonego do przechowywania opcji przeznaczonych dla skryptów sieciowych, które mają dostęp do bazy danych `sampdb`. Jak można zobaczyć, kod szuka opcji w pliku `/usr/local/apache/conf/sampdb.cnf` (UNIX) lub `C:\Apache\conf\sampdb.cnf` (Windows). Zwróć uwagę, że w systemie Windows przed nawiązaniem połączenia z bazą danych kod zmienia położenie na katalog główny dysku zawierającego plik opcji, a po nawiązaniu połączenia powraca do katalogu początkowego. Powody zastosowania takiego rozwiązania przedstawiono w punkcie 8.2.9, zatytułowanym „Określenie parametrów połączenia”.

Dystrybucja `sampdb` zawiera plik `sampdb.cnf`, który możesz zainstalować w celu użycia przez skrypty sieciowe oparte na DBI. Zawartość wymienionego pliku jest następująca:

```
[client]
host=localhost
user=sampadm
password=secret
```

Aby móc używać opracowanych w tym rozdziale skryptów sieciowych we własnym systemie, w przedstawionym powyżej kodzie musisz zmienić położenie pliku opcji, jeśli znajduje się on w innym katalogu. Konieczne jest również zainstalowanie pliku opcji `sampdb.cnf` w odpowiednim katalogu oraz umieszczenie w nim wartości opcji odpowiadających używanemu przez Ciebie serwerowi MySQL. Oznacza to między innymi podanie odpowiedniej nazwy użytkownika w MySQL i hasła.

W systemie UNIX właścicielem pliku opcji powinien być użytkownik uruchamiający serwer Apache, a uprawnienia pliku powinny być zdefiniowane jako 400 lub 600, uniemożliwiając tym samym innym użytkownikom jego edycję. To stanowi pewien rodzaj zabezpieczenia, ponieważ uniemożliwia bezpośredni odczyt pliku opcji przez innych użytkowników, którzy mają konta pozwalające na logowanie się w komputerze serwera WWW.

Niestety, plik opcji nadal może być odczytywany przez innych użytkowników mających uprawnienia do instalacji skryptów w serwerze WWW w celu ich uruchamiania. Skrypty wywoływane przez serwer WWW są uruchamiane wraz z uprawnieniami konta użytko-

do uruchomienia serwera WWW. Oznacza to, że inny użytkownik z uprawnieniami do instalacji skryptów sieciowych może utworzyć skrypt w taki sposób, aby otworzyć plik opcji i wyświetlić jego zawartość na stronie internetowej. Ponieważ skrypt jest uruchamiany przez użytkownika serwera WWW, ma pełne uprawnienia odczytu pliku opcji zawierającego parametry połączenia niezbędne do nawiązania połączenia z MySQL i uzyskania dostępu do bazy danych `sampdb`. Jeżeli jesteś jedyną osobą z możliwością zalogowania się w komputerze serwera WWW, to nie ma żadnego znaczenia. Jeśli jednak taką możliwość mają inni użytkownicy, którym nie ufasz, to powinieneś utworzyć konto MySQL jedynie z uprawnieniem odczytu (SELECT) bazy danych `sampdb`. Następnie w pliku `sampdb.cnf` umieść dane tego konta (nazwa użytkownika i hasło), a nie własnego. W ten sposób nie ryzykujesz umożliwienia skryptom nawiązania połączenia z bazą danych przez konto MySQL posiadające uprawnienia do modyfikacji tabel. W rozdziale 13., zatytułowanym „Bezpieczeństwo i kontrola dostępu”, dowiesz się, jak utworzyć konto użytkownika MySQL z jedynie ograniczonymi uprawnieniami. Wadą przedstawionego rozwiązania polegającego na użyciu konta MySQL jedynie z uprawnieniem odczytu jest możliwość tworzenia skryptów tylko pobierających dane, bez możliwości wstawiania nowych lub modyfikacji istniejących.

Alternatywne rozwiązanie polega na przygotowaniu skryptów do uruchamiania w ramach mechanizmu suEXEC serwera Apache. To pozwala na uruchamianie skryptów jako określony, zaufany użytkownik. Można więc tworzyć skrypty pobierające parametry połączenia z pliku opcji dostępnego do odczytu jedynie dla wymienionego użytkownika.

Kolejną możliwością jest tworzenie skryptów w taki sposób, aby prosiły o podanie nazwy użytkownika i hasła konta w MySQL, a następnie wykorzystały podane wartości do nawiązania połączenia z serwerem MySQL. Tego rodzaju rozwiązanie jest odpowiednie w przypadku skryptów administracyjnych, a nie przeznaczonych do ogólnego użycia. W każdym razie musisz pamiętać, że pewne metody pobierania nazwy użytkownika i hasła są celem ataków, jeśli atakującemu uda się umieścić sniffera pakietów w sieci między serwerem WWW i przeglądarką internetową klienta. Dlatego też warto skonfigurować bezpieczne połączenie. To jednak wykracza poza zakres tematyczny tej książki.

Jak możesz się przekonać na podstawie powyższych akapitów, zapewnienie bezpieczeństwa skryptom sieciowym może być trudnym zadaniem. To na pewno jest zagadnienie, na temat którego samodzielnie powinieneś dowiedzieć się więcej. Sam temat jest bardzo rozległy, a tutaj nie ma miejsca na jego dokładne omówienie. Wspomniana wcześniej książka *MySQL and Perl for the Web* zawiera poświęcony bezpieczeństwu sieciowemu rozdział, w którym zamieszczono informacje dotyczące konfiguracji bezpiecznych połączeń za pomocą protokołu SSL. Inne dobre źródła informacji dotyczących bezpieczeństwa znajdziesz w podręczniku użytkownika serwera Apache oraz w dokumencie FAQ na stronie <http://www.w3.org/Security/Faq/>.

8.4.4. Przeglądarka bazy danych

Pierwszą internetową aplikacją MySQL jest prosty skrypt `db_browse.pl`, pozwalający na interaktywne przeglądanie tabel znajdujących się w bazie danych `sampdb` oraz ich zawartości za pomocą przeglądarki internetowej. Skrypt działa następująco:

- W trakcie pierwszego żądania skryptu *db_browse.pl* z przeglądarki internetowej następuje nawiązanie połączenia z serwerem MySQL, pobranie listy tabel znajdujących się w bazie danych *sampdb* i wysłanie do przeglądarki strony internetowej wyświetlającej nazwę wszystkich tabel jako łączy. Po wybraniu tabeli przez kliknięcie łącza przeglądarka internetowa wysyła do serwera WWW żądanie wyświetlenia przez skrypt *db_browse.pl* zawartości wskazanej tabeli.
- Jeżeli skrypt *db_browse.pl* zostanie wywołany z nazwą tabeli, pobierze jej zawartość i przekaże te informacje przeglądarce internetowej. Nagłówkiem każdej kolumny danych będzie nazwa kolumny tabeli. Nagłówki zostaną wyświetlone w postaci łączy, jego kliknięcie spowoduje wysłanie przez przeglądarkę internetową do serwera WWW żądania ponownego wyświetlenia tej samej tabeli posortowanej względem wybranej kolumny.

Ostrzeżenie

Zanim przejdiesz dalej, musisz wiedzieć o jednym: choć skrypt *db_browser.pl* jest pouczający, ponieważ przedstawia wiele użytecznych koncepcji programowania sieciowego, to stanowi także lukę w zabezpieczeniach. Skrypt wyświetla wszystkie tabele znajdujące się w bazie danych *sampdb*, co może stanowić problem. W rozdziale 9., zatytułowanym „Tworzenie programów MySQL przy użyciu języka PHP”, utworzymy skrypt pozwalający członkom Ligi Historycznej na edycję przez internet daty wygaśnięcia ich członkostwa. Dostęp do rekordów jest kontrolowany przez hasła przechowywane w tabeli *member_pass*. Dzięki skryptowi *db_browse.pl* każdy może zajrzeć do tabeli haseł, a tym samym uzyskać dostęp do informacji niezbędnych do przeprowadzenia edycji dowolnego rekordu tabeli *member*! Dlatego też po wypróbowaniu skryptu i zrozumieniu sposobu jego działania najlepiej usunąć go z katalogu *cgi-bin*. (Alternatywne rozwiązanie polega na umieszczeniu skryptu w prywatnym serwerze WWW niedostępnym dla niezaufanych użytkowników).

Przyjmujemy założenie, że nie wystraszyłeś się powyższym złowieszczym ostrzeżeniem, i przystępujemy do analizy sposobu działania skryptu *db_browse.pl*. Część główna skryptu generuje początkową część strony internetowej, a następnie sprawdza parametr *tbl_name*, aby przekonać się, czy powinna zostać wyświetlona zawartość konkretnej tabeli:

```
#!/usr/bin/perl
# db_browse.pl - Umożliwienie przeglądania bazy danych sampdb za pomocą internetu.

use strict;
use warnings;
use DBI;
use CGI qw (:standard escapeHTML escape);

# ... Konfiguracja połączenia z bazą danych (nie została pokazana) ...

my $db_name = "sampdb";

# Wygenerowanie początkowej części strony internetowej.
my $title = "Przeglądarka bazy danych $db_name";
print header ();
```

```

print start_html (-title => $title, -bgcolor => "white");
print h1 ($title);

# Parametry, które mogą znajdować się w adresie URL.
my $tbl_name = param ("tbl_name");
my $sort_col = param ("sort_col");

# Jeżeli zmienna $tbl_name nie ma wartości, wtedy zostanie wyświetlona lista nazw tabel, które można kliknąć.
# W przeciwnym razie wyświetlana jest zawartość wskazanej tabeli. Zmienna $sort_col, o ile jest ustawiona,
# wskazuje kolumnę, względem której ma być posortowana tabela.
if (!defined ($tbl_name))
{
    display_table_names ($dbh, $db_name)
}
else
{
    display_table_contents ($dbh, $db_name, $tbl_name, $sort_col);
}
print end_html ();

```

Wartość parametru można uzyskać bardzo łatwo, ponieważ moduł CGI.pm zajmuje się obsługą informacji przekazywanych skryptowi przez serwer WWW. Konieczne jest jedynie wywołanie funkcji `param()` wraz z nazwą interesującego nas parametru. W części głównej skryptu *db_browse.pl* wspomniany parametr nosi nazwę `tbl_name`. Jeżeli nie został ustawiony, to oznacza pierwsze wywołanie skryptu i wyświetlenie listy tabel. W przeciwnym razie wyświetlona będzie zawartość tabeli wskazanej w parametrze `tbl_name`, posortowana względem kolumny wymienionej w parametrze `sort_col`.

Funkcja `display_table_names()` generuje stronę początkową. Wymieniona funkcja pobiera listę tabel, a następnie wyświetla ją w postaci nieuporządkowanej listy, której każdy element jest nazwą tabeli w bazie danych `sampdb`:

```

sub display_table_names
{
    my ($dbh, $db_name) = @_;

    print p ("Wybierz tabelę klikając jej nazwę:");
    # Pobranie odniesienia do składającej się z pojedynczej kolumny tablicy zawierającej nazwy tabel.
    my $sth = $dbh->prepare (qq{
        SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
        WHERE TABLE_SCHEMA = ? ORDER BY TABLE_NAME
    });
    $sth->execute ($db_name);

    # Utworzenie nieporządkowanej listy za pomocą wywołania funkcji
    # ul() i li(). Każdy element jest łączem, którego kliknięcie ponownie
    # wywoła skrypt w celu wyświetlenia zawartości wskazanej tabeli.
    my @item;
    while (my ($tbl_name) = $sth->fetchrow_array ())
    {
        my $url = sprintf ("%s?tbl_name=%s", url (), escape ($tbl_name));
        my $link = a ({-href => $url}, escapeHTML ($tbl_name));
        push (@item, li ($link));
    }
    print ul (@item);
}

```

Funkcja `li()` powoduje dodanie znaczników `` i `` wokół każdego elementu listy, natomiast funkcja `ul()` dodaje znaczniki `` i `` wokół zestawu elementów. Każda nazwa tabeli na liście jest łączem, którego kliknięcie powoduje ponowne wywołanie skryptu w celu wyświetlenia zawartości wskazanej tabeli. Lista wygenerowana przez funkcję `display_table_names()` przedstawia się następująco:

```
<ul>
<li><a href="/cgi-bin/db_browse.pl?tbl_name=absence">absence</a></li>
<li><a href="/cgi-bin/db_browse.pl?tbl_name=grade_event">grade_event</a></li>
<li><a href="/cgi-bin/db_browse.pl?tbl_name=member">member</a></li>
...
</ul>
```

Jeżeli parametr `tbl_name` ma przypisaną wartość w trakcie wywoływania skryptu `db_browse.pl`, to skrypt przekazuje tę wartość funkcji `display_table_contents()` wraz z nazwą kolumny (o ile zostanie podana), względem której ma zostać posortowany wynik:

```
sub display_table_contents
{
    my ($dbh, $db_name, $tbl_name, $sort_col) = @_;
    my $sort_clause = "";
    my @rows;
    my @cells;

    # Jeżeli podana została kolumna sortowania, należy jej użyć do posortowania wyników.
    if (defined ($sort_col))
    {
        $sort_clause = " ORDER BY " . $dbh->quote_identifier ($sort_col);
    }

    # Wyświetlenie łącza pozwalającego na powrót do listy tabel.
    print p (a ({-href => url ()}, "Wyświetl listę tabel"));

    print p (strong ("Zawartość tabeli $tbl_name:"));

    my $sth = $dbh->prepare (
        "SELECT * FROM "
        . $dbh->quote_identifier ($db_name, $tbl_name)
        . "$sort_clause LIMIT 200"
    );
    $sth->execute ();

    # Użycie nazw kolumn w tabeli bazy danych jako nagłówków
    # w tabeli HTML. Każda nazwa jest łączem powodującym
    # ponowne wywołanie skryptu w celu wyświetlenia zawartości
    # tabeli posortowanej względem wybranej kolumny.
    foreach my $col_name (@{$sth->{NAME}})
    {
        my $url = sprintf ("%s?tbl_name=%s;sort_col=%s",
            url (),
            escape ($tbl_name),
            escape ($col_name));
        my $link = a ({-href => $url}, escapeHTML ($col_name));
        push (@cells, th ($link));
    }
    push (@rows, Tr (@cells));
}
```

```

# Wyświetlenie rekordów tabeli.
while (my @ary = $sth->fetchrow_array ())
{
    @cells = ();
    foreach my $val (@ary)
    {
        # Wyświetlenie niepustych wartości, w przeciwnym razie wyświetlana jest spacja niełamiąca.
        if (defined ($val) && $val ne "")
        {
            $val = escapeHTML ($val);
        }
        else
        {
            $val = "&nbsp;";
        }
        push (@cells, td ($val));
    }
    push (@rows, Tr (@cells));
}

# Wyświetlenie tabeli wraz z obramowaniem.
print table ({-border => "1"}, @rows);
}

```

Zapytanie zawiera także klauzulę `LIMIT 200` jako proste zabezpieczenie przed wysłaniem przez serwer ogromnej ilości danych do przeglądarki internetowej. (Tabele znajdujące się w bazie danych `sampdb` zawierają niewiele danych, więc wspomniane niebezpieczeństwo praktycznie nie istnieje. Jednak zabezpieczenie przyda się, jeśli skrypt zaadaptujesz do wyświetlania zawartości tabel innych baz danych). Funkcja `display_table_contents()` wyświetla rekordy tabeli w postaci tabeli HTML, używając funkcji `th()` i `td()` do wygenerowania nagłówków oraz komórek tabeli, `Tr()` do zgrupowania komórek w wierszu oraz `table()` do wygenerowania znaczników `<table>` obejmujących tabelę.

Tabela HTML wyświetla nagłówki kolumn w postaci łączy, których kliknięcie powoduje ponowne wyświetlenie danej tabeli. Wspomniane łącza zawierają parametr `sort_col`, wyraźnie wskazujący kolumnę, względem której ma zostać posortowana tabela. Na przykład, jeśli strona wyświetla zawartość tabeli `grade_event`, nagłówki kolumn przedstawiają się następująco:

```

<a href="/cgi-bin/db_browse.pl?tbl_name=grade_event&sort_col=date">
date</a>
<a href="/cgi-bin/db_browse.pl?tbl_name=grade_event&sort_col=category">
category</a>
<a href="/cgi-bin/db_browse.pl?tbl_name=grade_event&sort_col=event_id">
event_id</a>

```

W funkcji `display_table_contents()` zastosowano małą sztuczkę w zakresie zastąpienia pustych wartości spacją niełamiącą (` `). W tabeli wraz z obramowaniem niektóre przeglądarki internetowe nie wyświetlają prawidłowo obramowania dla pustej komórki. Dlatego też umieszczenie w komórce spacji niełamiącej rozwiązuje ten problem.

W celu utworzenia bardziej ogólnego skryptu można zmodyfikować `db_browse.pl` w taki sposób, aby możliwe było przeglądanie wielu baz danych. Na przykład, możesz

opracować skrypt, który na początku wyświetla listę baz danych w serwerze zamiast listy tabel określonej bazy danych. Następnie użytkownik wybiera bazę danych i otrzymuje listę znajdujących się w niej tabel.

8.4.5. Przeglądarka tabel projektu ocen uczniów

Nasz kolejny skrypt sieciowy, o nazwie *score_browse.pl*, jest zaprojektowany do wyświetlania wyników uzyskanych przez uczniów, które zostały zapisane w ramach projektu ocen uczniów. Ujmując rzecz najprościej, chcemy otrzymać możliwość wprowadzania ocen, zanim będzie można je pobierać. Jednak skrypt przeznaczony do wstawiania ocen zostanie zaprezentowany w następnym rozdziale. W międzyczasie tabele projektu ocen uczniów zawierają nieco danych pozwalających na przetestowanie omawianego tutaj skryptu. Skrypt będzie wykorzystywany do wyświetlania ocen, nawet pomimo braku wygodnej metody ich wstawiania do bazy danych. Omawiany tutaj skrypt wyświetla uporządkowaną listę ocen z dowolnego sprawdzianu lub testu, co jest użyteczne podczas określania klasyfikacji wyniku i wystawiania ocen końcowych.

Pod wieloma względami skrypt *score_browse.pl* jest podobny do omówionego wcześniej *score_browse.pl* (oba działają w charakterze przeglądarek), ale został przeznaczony do konkretnego celu, jakim jest analiza ocen otrzymanych przez uczniów w danym sprawdzianie lub teście. Strona początkowa wyświetla listę ocenianych zdarzeń, co pozwala na wybór konkretnego i wyświetlenie uzyskanych w nim ocen. Same oceny dla danego zdarzenia są posortowane od najwyższej, więc na podstawie wyniku można określić klasyfikację.

Skrypt *score_browse.pl* musi przeanalizować tylko jeden parametr, o nazwie *event_id*, aby sprawdzić, czy podano konkretne zdarzenie. Jeśli wymieniony parametr nie ma wartości, wtedy skrypt *score_browse.pl* wyświetla rekordy tabeli *grade_event*, dając w ten sposób użytkownikowi możliwość wyboru zdarzenia. W przeciwnym razie skrypt wyświetla oceny uzyskane przez uczniów w danym zdarzeniu:

```
# ... Konfiguracja połączenia z bazą danych (nie została pokazana) ...

# Wygenerowanie początkowej części strony internetowej.
my $title = "Przeglądarka ocen uczniów";
print header ();
print start_html (-title => $title, -bgcolor => "white");
print h1 ($title);

# Parametr wskazujący zdarzenie, dla którego mają zostać wyświetlone oceny.
my $event_id = param ("event_id");

# Jeżeli zmienna $event_id nie ma wartości, wtedy zostanie wyświetlona lista zdarzeń.
# W przeciwnym razie wyświetlane są oceny dla wskazanego zdarzenia.
if (!defined ($event_id))
{
    display_events ($dbh)
}
else
{

```



```

    display_scores ($dbh, $event_id);
}
print end_html ();

```

Funkcja `display_events()` pobiera informacje z tabeli `grade_event`, a następnie wyświetla je w postaci tabeli HTML, używając jako jej nagłówków nazw kolumn wymienionych w zapytaniu. W każdym wierszu wartość `event_id` jest łączem, którego kliknięcie powoduje wykonanie zapytania pobierającego wyniki uzyskane przez uczniów w danym zdarzeniu. Adres URL zdarzenia to po prostu ścieżka dostępu do skryptu `score_browse.pl` wraz z dołączonym parametrem wskazującym numer zdarzenia:

```
/cgi-bin/score_browse.pl?event_id=
```

Funkcja `display_events()` przedstawia się następująco:

```

sub display_events
{
    my $dbh = shift;
    my @rows;
    my @cells;

    print p ("Wybierz zdarzenie klikając jego numer:");

    # Pobranie listy zdarzeń.
    my $sth = $dbh->prepare (qq{
        SELECT event_id, date, category
        FROM grade_event
        ORDER BY event_id
    });
    $sth->execute ();

    # Użycie nazw kolumn tabeli jako nagłówków w tabeli HTML.
    for (my $i = 0; $i < $sth->{NUM_OF_FIELDS}; $i++)
    {
        push (@cells, th (escapeHTML ($sth->{NAME}->[$i])));
    }
    push (@rows, Tr (@cells));

    # Wyświetlenie informacji o każdym zdarzeniu jako oddzielnego wiersza tabeli HTML.
    while (my ($event_id, $date, $category) = $sth->fetchrow_array ())
    {
        @cells = ();
        # Wyświetlenie identyfikatora zdarzenia jako łącza, którego kliknięcie powoduje
        # ponowne uruchomienie skryptu i wyświetlenie ocen wskazanego zdarzenia.
        my $url = sprintf ("%s?event id=%d", url (), $event_id);
        my $link = a ({-href => $url}, escapeHTML ($event_id));
        push (@cells, td ($link));
        # Wyświetlenie daty i kategorii zdarzenia.
        push (@cells, td (escapeHTML ($date)));
        push (@cells, td (escapeHTML ($category)));
        push (@rows, Tr (@cells));
    }
    # Wyświetlenie tabeli wraz z obramowaniem.
    print table ({-border => "1"}, @rows);
}

```

Kiedy użytkownik wybierze zdarzenie, przeglądarka internetowa wyśle zapytanie skryptu *score_browse.pl* wraz z umieszczonym na końcu adresu URL identyfikatorem zdarzenia. Skrypt *score_browse.pl* odnajdzie ustawiony parametr *event_id* i wywoła funkcję *display_scores()*, która z kolei wyświetli wszystkie wyniki uzyskane we wskazanym zdarzeniu. Wspomniana funkcja zawiera także łącze *Pokaż listę zdarzeń*, pozwalające na powrót do strony początkowej, aby użytkownik mógł łatwo wybrać inne zdarzenie z listy:

```
sub display_scores
{
    my ($dbh, $event_id) = @_ ;
    my @rows;
    my @cells;

    # Wygenerowanie łącza do skryptu niezawierającego parametru
    # event_id. Jeżeli użytkownik kliknie to łącze, skrypt wyświetli
    # listę zdarzeń.
    print p (a ({-href => url ()}, "Pokaż listę zdarzeń"));

    # Pobranie wyników uzyskanych we wskazanym zdarzeniu.
    my $sth = $dbh->prepare (qq{
        SELECT
            student.name,
            grade_event.date,
            score.score,
            grade_event.category
        FROM
            student INNER JOIN score INNER JOIN grade_event
        ON
            student.student_id = score.student_id
            AND score.event_id = grade_event.event_id
        WHERE
            grade_event.event_id = ?
        ORDER BY
            grade_event.date ASC,
            grade_event.category ASC,
            score.score DESC
    });
    $sth->execute ($event_id); # Umieszczenie identyfikatora zdarzenia w miejscu zarezerwowanym
                              # w zapytaniu.

    print p (strong ("Wyniki uzyskane w zdarzeniu $event_id"));

    # Użycie nazw kolumn jako nagłówków w tabeli HTML.
    for (my $i = 0; $i < $sth->{NUM_OF_FIELDS}; $i++)
    {
        push (@cells, th (escapeHTML ($sth->{NAME}->[$i])));
    }
    push (@rows, Tr (@cells));

    while (my @ary = $sth->fetchrow_array ())
    {
        @cells = ();
        foreach my $val (@ary)
        {
            # Wyświetlenie niepustych wartości, w przeciwnym razie wyświetlana jest spacja nielamiąca.
            if (defined ($val) && $val ne "")
```

```

    {
        $val = escapeHTML ($val);
    }
    else
    {
        $val = "&nbsp;";
    }
    push (@cells, td ($val));
}
push (@rows, Tr (@cells));
}

# Wyświetlenie tabeli wraz z obramowaniem.
print table ({-border => "1"}, @rows);
}

```

Polecenie wywołujące funkcję `display_scores()` działa w sposób podobny do przedstawionego w podpunkcie 1.4.9.10, zatytułowanym „Pobieranie informacji z wielu tabel”, w którym pokazano, jak tworzyć złączenia. W wymienionym podpunkcie były pobierane wyniki dla danej daty, ponieważ data jest czytelniejsza niż numer identyfikacyjny zdarzenia. Natomiast tutaj, w skrypcie *score_browse.pl*, znany jest dokładny identyfikator zdarzenia. Powód takiego rozwiązania jest bardzo prosty: skrypt wyświetla listę identyfikatorów zdarzeń wraz z ich datami i kategoriami. Jak możesz się przekonać, taki rodzaj interfejsu minimalizuje potrzebę poznania wszystkich szczegółów. Nie trzeba znać identyfikatora zdarzenia, musisz jedynie rozpoznać datę interesującego Cię zdarzenia. Następnie skrypt wiąże tę datę z odpowiednim identyfikatorem.

8.4.6. Wyszukiwanie wspólnych zainteresowań w projekcie Ligi Historycznej

Skrypty *db_browse.pl* i *score_browse.pl* pozwalają użytkownikowi na dokonanie wyboru spośród listy opcji wyświetlonej na stronie początkowej. Wspomniane opcje są przedstawione w postaci łączy, których kliknięcie powoduje ponowne uruchomienie skryptu wraz z odpowiednimi wartościami parametrów. Innym sposobem umożliwienia użytkownikowi dostarczenia informacji jest wyświetlenie formularza do wypełnienia. Takie rozwiązanie jest lepsze, gdy dostępnych możliwości nie można łatwo przedstawić w postaci zestawu wartości. Nasz kolejny skrypt demonstruje tę metodę pobierania danych wejściowych od użytkownika.

W podrozdziale 8.3, zatytułowanym „Praca z DBI”, utworzyliśmy skrypt wiersza poleceń o nazwie *interests.pl*, pozwalający na wyszukiwanie członków Ligi Historycznej o takich samych zainteresowaniach. Jednak do wymienionego skryptu członkowie Ligi nie mają bezpośredniego dostępu, sekretarka musi uruchamiać go z poziomu wiersza poleceń, a następnie wysyłać wynik jego działania osobie, która poprosiła o listę członków Ligi o podobnych zainteresowaniach. Byłoby dobrze, gdyby członkowie Ligi Historycznej sami mogli wyszukiwać inne osoby o podobnych zainteresowaniach. Rozwiązaniem jest więc przygotowanie odpowiedniego skryptu sieciowego. W pozostałej części tego punktu

przedstawione będą dwa podejścia w zakresie przeszukiwania tabeli. Pierwsze opiera się na dopasowaniu wzorca, natomiast drugie używa oferowanych przez MySQL możliwości wyszukiwania pełnego tekstu.

8.4.6.1. Wyszukiwanie za pomocą dopasowania wzorca

Pierwszy skrypt, o nazwie *ushl_browse.pl*, wyświetla formularz, w którym użytkownik może podać słowo kluczowe. Po wysłaniu formularza skrypt zostanie ponownie uruchomiony i przeprowadzi operację przeszukiwania tabeli *member*, a następnie wyświetli wyniki. Wyszukiwanie odbywa się po dodaniu znaku wieloznacznego % po obu stronach słowa kluczowego. Stosowana jest klauzula *LIKE*, która wyszukuje rekordy zawierające w dowolnym miejscu kolumny *interests* podane słowo kluczowe.

Główna część skryptu wyświetla formularz pozwalający na podanie słowa kluczowego. Sprawdza również, czy słowo kluczowe znajduje się w wysłanych danych formularza. Jeśli tak, przeprowadzana jest operacja wyszukiwania:

```
my $title = "Wyszukiwanie osób o podobnych zainteresowaniach";
print header ();
print start_html (-title => $title, -bgcolor => "white");
print h1 ($title);

# Sprawdzany parametr.
my $keyword = param ("keyword");

# Wyświetlenie formularza. Ponadto, jeśli zmienna $keyword ma przypisaną wartość,
# przeprowadzane jest wyszukiwanie i wyświetlenie znalezionych osób o podobnych zainteresowaniach.
print start_form (-method => "post");
print p ("Podaj szukane słowo kluczowe:");
print textfield (-name => "keyword", -value => "", -size => 40);
print submit (-name => "button", -value => "Szukaj");
print end_form ();

# Nawiązanie połączenia z serwerem i rozpoczęcie wyszukiwania, o ile podano słowo kluczowe.
if (defined ($keyword) && $keyword !~ /\s*/)
{
    # ... Konfiguracja połączenia z bazą danych (niepokazana tutaj) ...
    search_members ($dbh, $keyword);
    # ... Zamknięcie połączenia (niepokazane tutaj) ...
}
```

Skrypt komunikuje się ze sobą w nieco inny sposób niż *db_browse.pl* lub *score_browse.pl*. Parametr nie jest dodawany na końcu adresu URL. Zamiast tego informacje podawane w formularzu są kodowane przez przeglądarkę internetową i wysyłane jako część żądania typu *post*. Jednak dla modułu *CGI.pm* sposób wysyłania informacji nie ma znaczenia, ponieważ funkcja *param()* zwraca wartość parametru niezależnie od sposobu jego wysłania. To kolejna cecha modułu *CGI.pm* ułatwiająca programowanie sieciowe.

Wyszukiwanie słowa kluczowego odbywa się za pomocą funkcji *search_members()*. Wymieniona funkcja pobiera argumenty w postaci uchwytu do bazy danych oraz słowa kluczowego, a następnie wykonuje zapytanie i zwraca listę dopasowanych rekordów:

```

sub search_members
{
    my ($dbh, $interest) = @_ ;

    print p ("Wyniki wyszukiwania dla słowa kluczowego: " . escapeHTML ($interest));
    my $sth = $dbh->prepare (qq{
        SELECT * FROM member WHERE interests LIKE ?
        ORDER BY last_name, first_name
    });

    # Wyszukanie ciągu tekstowego w dowolnym miejscu kolumny interests.
    $sth->execute ("% " . $interest . "%");
    my $count = 0;
    while (my $ref = $sth->fetchrow_hashref ())
    {
        html_format_entry ($ref);
        ++$count;
    }
    print p ("Liczba znalezionych rekordów: $count");
}

```

Po uruchomieniu skryptu *ushl_browse.pl* przekonasz się, że po każdym wysłaniu słowa kluczowego następuje jego ponowne wyświetlenie w formularzu na kolejnej stronie, nawet jeśli skrypt podaje pusty ciąg tekstowy jako wartość pola keyword podczas generowania formularza. Dzieje się tak, ponieważ moduł CGI.pm automatycznie wypełnia pola formularza wartościami ze środowiska wykonania skryptu, o ile wspomniane wartości istnieją. Aby zmienić to zachowanie i wyświetlić puste pole za każdym razem, należy dodać parametr `override` w wywołaniu `textfield()`:

```

print textfield (-name => "keyword",
                 -value => "",
                 -override => 1,
                 -size => 40);

```

Funkcja `search_members()` używa funkcji pomocniczej `html_format_entry()` do wyświetlenia poszczególnych elementów. Wymieniona funkcja pomocnicza jest podobna do funkcji o takiej samej nazwie, którą utworzyliśmy wcześniej w skrypcie *gen_dir.pl* (patrz punkt 8.3.1, zatytułowany „Generowanie katalogu Ligi Historycznej”). Jednak do wygenerowania znaczników HTML funkcja pomocnicza w skrypcie *gen_dir.pl* używała poleceń bezpośrednio wstawiających znaczniki, natomiast w skrypcie *ushl_browse.pl* używane są funkcje dostępne w module CGI.pm:

```

sub html_format_entry
{
    my $entry_ref = shift;

    # Kodowanie znaków, które mają znaczenie specjalne w HTML.
    foreach my $key (keys (%{$entry_ref}))
    {
        next unless defined ($entry_ref->{$key});
        $entry_ref->{$key} = escapeHTML ($entry_ref->{$key});
    }
    print strong ("Imię i nazwisko: " . format_name ($entry_ref)), br ();
}

```

```

my $address = "";
$address .= $entry_ref->{street}
            if defined ($entry_ref->{street});
$address .= ", " . $entry_ref->{city}
            if defined ($entry_ref->{city});
$address .= ", " . $entry_ref->{state}
            if defined ($entry_ref->{state});
$address .= " " . $entry_ref->{zip}
            if defined ($entry_ref->{zip});
print "Adres: $address", br ()
            if $address ne "";
print "Telefon: $entry_ref->{phone}", br ()
            if defined ($entry_ref->{phone});
print "E-mail: $entry_ref->{email}", br ()
            if defined ($entry_ref->{email});
print "Zainteresowania: $entry_ref->{interests}", br ()
            if defined ($entry_ref->{interests});
print br ();
}

```

Funkcja `html_format_entry()` używa `format_name()` do połączenia wartości kolumn `first_name`, `last_name` i `suffix`. To funkcja identyczna jak funkcja o takiej samej nazwie zaimplementowana w skrypcie *gen_dir.pl*.

8.4.6.2. Wyszukiwanie za pomocą indeksu typu FULLTEXT

Członkowie Ligi Historycznej mogą mieć wiele zainteresowań. W takim przypadku są one rozdzielone przecinkami w kolumnie `interests` tabeli `member`, na przykład:

wojna o niepodległość,wojna amerykańsko-hiszpańska,okres kolonialny,gorączka złota,Lincoln

Czy opracowanego wcześniej skryptu *ushl_browse.pl* można użyć do wyszukania rekordów zawierających wiele słów kluczowych? W pewnym sensie, ale praktycznie nie. Wprawdzie w formularzu możesz podać wiele słów, ale rekordy nie zostaną dopasowane, o ile nie utworzymy znacznie bardziej skomplikowanego zapytania, szukającego dopasowania dla poszczególnych słów. Znacznie elastyczniejsze podejście polega na użyciu indeksu typu FULLTEXT. W tym podpunkcie omówimy skrypt *ushl_ft_browse.pl*, wykonujący to zadanie. Więcej informacji na temat oferowanych przez MySQL możliwości w zakresie wyszukiwania pełnego tekstu znajdziesz w podrozdziale 2.14, zatytułowanym „Wyszukiwanie pełnego tekstu”.

Aby można było użyć tabeli `member` do wyszukiwania pełnego tekstu, to musi być tabela typu MyISAM. Jeżeli utworzyłeś tabelę `member` za pomocą innego silnika bazy danych, skonwertuj ją na tabelę typu MyISAM za pomocą zapytania ALTER TABLE:

```
ALTER TABLE member ENGINE=MyISAM;
```

Następnie konieczne jest prawidłowe zindeksowanie tabeli `member`. W tym celu wykonaj poniższe zapytanie:

```
ALTER TABLE member ADD FULLTEXT (interests);
```

Po wprowadzeniu powyższych zmian kolumna `interests` będzie mogła być używana w trakcie wyszukiwania pełnego tekstu. Skrypt `ushl_ft_browse.pl` w dystrybucji `sampdb` został oparty na skrypcie `ushl_browse.pl`, od którego różni się jedynie funkcją `search_members()`, odpowiedzialną za konstrukcję zapytania operacji wyszukiwania. Zmodyfikowana wersja wymienionej funkcji przedstawia się następująco:

```
sub search_members
{
    my ($dbh, $interest) = @_ ;

    print p ("Wyniki wyszukiwania dla słowa kluczowego: " . escapeHTML ($interest));
    my $sth = $dbh->prepare (qq{
        SELECT * FROM member WHERE MATCH(interests) AGAINST(?)
        ORDER BY last_name, first_name
    });
    # Wyszukanie ciągu tekstowego w dowolnym miejscu kolumny interests.
    $sth->execute ($interest);
    my $count = 0;
    while (my $ref = $sth->fetchrow_hashref ())
    {
        html_format_entry ($ref);
        ++$count;
    }
    print p ("Liczba znalezionych rekordów: $count");
}
```

- W powyższej wersji metody `search_members()` wprowadzono następujące zmiany względem wcześniejszej wersji:
- Zapytanie używa `MATCH() ... AGAINST()` zamiast `LIKE`.
- Nie ma potrzeby użycia znaków wieloznacznych `%` w celu konwersji ciągu tekstowego słowa kluczowego na wzorzec.

Po wprowadzeniu powyższych zmian można wywołać skrypt `ushl_ft_browse.pl` z poziomu przeglądarki internetowej i wprowadzić kilka słów kluczowych w formularzu (bez przecinków). Skrypt wyszuka rekordy, w których dopasowano dowolne z podanych słów kluczowych.

Istnieje możliwość dalszej rozbudowy omówionego skryptu. Na przykład, można wykorzystać fakt, że wyszukiwanie pełnego tekstu ma możliwość jednoczesnego przeszukiwania wielu kolumn dzięki utworzeniu indeksu obejmującego wiele kolumn i modyfikacji skryptu `ushl_ft_browse.pl` w taki sposób, aby przeszukiwane były wszystkie kolumny indeksu. Możesz więc usunąć bieżący indeks typu `FULLTEXT` i dodać nowy, obejmujący kolumny `interests`, `last_name` i `first_name`:

```
ALTER TABLE member DROP INDEX interests;
ALTER TABLE member ADD FULLTEXT (interests,last_name,first_name);
```

W celu użycia nowego indeksu zmodyfikuj zapytanie `SELECT` w funkcji `search_members()`, zamieniając `MATCH(interests)` na `MATCH(interests,last_name,first_name)`.

Kolejną zmianą, którą można wprowadzić w skrypcie `ushl_ft_browse.pl`, jest dodanie kilku przycisków opcji do formularza, aby pozwolić użytkownikowi na wybór między

„dopasowaniem dowolnego słowa kluczowego” i „dopasowaniem wszystkich słów kluczowych”. Tryb dopasowania dowolnego słowa kluczowego jest aktualnie zastosowany w skrypcie. Aby zaimplementować tryb dopasowania wszystkich słów kluczowych, należy zastosować wyszukiwanie pełnego tekstu typu `IN BOOLEAN MODE` i poprzedzić każde słowo kluczowe znakiem plus oznaczającym konieczność jego istnienia w dopasowanych rekordach. Więcej informacji na temat boolowskiego trybu wyszukiwania znajdziesz w punkcie 2.14.2, zatytułowanym „Wyszukiwanie pełnego tekstu w trybie boolowskim”.

Tworzenie programów MySQL przy użyciu języka PHP

PHP to język skryptowy przeznaczony do tworzenia stron internetowych wraz z osadzonym kodem wykonywanym w trakcie żądania wyświetlenia strony. Wspomniany kod może dynamicznie generować treść wstawianą jako część danych wyjściowych wysyłanych do przeglądarki internetowej klienta. W tym rozdziale dowiesz się, jak tworzyć aplikacje sieciowe w języku PHP oparte na bazie danych MySQL. W celu porównania PHP z API C i API Perl DBI podczas programowania MySQL zajrzyj do rozdziału 6., zatytułowanego „Wprowadzenie do programowania MySQL”.

W tym rozdziale przyjęto założenie, że język PHP będzie używany w połączeniu z serwerem WWW Apache, choć skrypty prawdopodobnie będą działały także z innym serwerem WWW. Sam PHP można stosować jako moduł Apache lub samodzielny interpreter, używany podobnie do tradycyjnego programu CGI. Stosowanie PHP jako modułu jest zalecane, ponieważ zapewnia wtedy lepszą wydajność działania.

Przykłady przedstawione w rozdziale opierają się na bazie danych *sampdb* i tabelach utworzonych w rozdziale 1., zatytułowanym „Rozpoczęcie pracy z MySQL”, dla projektów ocen uczniów i Ligi Historycznej. Jeśli musisz pobrać jakiekolwiek oprogramowanie używane w tym rozdziale, zajrzyj do dodatku A, zatytułowanego „Oprogramowanie wymagane do użycia tej książki”. W wymienionym dodatku znajdziesz informacje dotyczące pobrania dystrybucji *sampdb* zawierającej skrypty opracowane w tym rozdziale. Skrypty znajdziesz w katalogu *phpapi* dystrybucji.

PHP oferuje kilka interfejsów umożliwiających uzyskanie dostępu do bazy danych MySQL:

- Rozszerzenie *mysql* to oryginalny interfejs dostępu do MySQL. Składa się z funkcji o nazwach w formie *mysql_XXX()*. W większości przypadków bezpośrednio odpowiadają one funkcjom API C o takich samych nazwach.

Ten interfejs nie oferuje dostępu do funkcji wprowadzonych w serwerach MySQL 4.1 i nowszych i obecnie jest uznawany za przestarzały.

- Rozszerzenie `mysql_i` to usprawniona wersja rozszerzenia `mysql`. Oferuje ono dwa style wywoływania. Pierwszy to zestaw funkcji o nazwach w formie `mysql_xxx()`. Drugi to interfejs zorientowany obiektowo.
- Rozszerzenie PDO (ang. *PHP Data Objects*) jest w mniejszym stopniu powiązane z konkretnym silnikiem bazy danych. Zapewnia ono zorientowany obiektowo interfejs o projekcie podobnym do modułu Perl DBI i niezależny od bazy danych. Opiera się na dwupoziomowej architekturze, w której górny poziom przedstawia ujednolicony interfejs, natomiast dolny zawiera sterowniki do różnych silników baz danych. W celu zmiany jednego sterownika na inny wystarczy jedynie zmodyfikować argumenty przekazywane do metody nawiązującej połączenie. Wspomniane argumenty powinny być odpowiednie dla sterownika, który ma zostać użyty.

W tym rozdziale w skryptach PHP będziemy używać PDO. Rozszerzenie PDO działa wraz z PHP w wersji 5.0 i nowszej, ale tutaj przyjęto założenie o użyciu PHP w wersji minimum 5.1, ponieważ to była pierwsza wersja zawierająca wbudowane rozszerzenie PDO. Więcej informacji znajdziesz na stronie <http://www.php.net/pdo>.

Sterownik PDO dla bazy danych MySQL został początkowo opracowany jako dołączany do utworzonej w języku C biblioteki klienta MySQL. Taki projekt wprowadzał zależność PHP od pewnego komponentu dystrybucji MySQL, gdy za pomocą języka PHP chciałeś uzyskać dostęp do baz danych MySQL. Nowsza biblioteka, o nazwie `mysqlnd`, jest rodzimym sterownikiem implementującym ten sam protokół komunikacyjny jak w bibliotece `libmysqlclient` i może ją zastąpić. Dzięki bibliotece `mysqlnd` można z poziomu PHP uzyskać dostęp do baz danych MySQL bez konieczności instalacji biblioteki klienta. Biblioteka `mysqlnd` została dołączona do PHP w wersji 5.3, natomiast od wersji 5.4 jest biblioteką domyślną. Dlatego też, aby używać PDO wraz z `mysqlnd`, potrzebujesz PHP w wersji minimum 5.3.

W większości przypadków w rozdziale omówiono jedynie te obiekty i metody PDO, które są potrzebne dla tworzonych skryptów. Ponadto, przedstawiono jedynie sterownik PDO dla MySQL. Dostępne są również sterowniki dla innych silników baz danych, ale nie będą tutaj omawiane. Warto zajrzeć do podręcznika użytkownika PHP, dostępnego na witrynie <http://www.php.net/>. Omówiono w nim wszystkie możliwości oferowane przez PHP.

Ogólnie rzecz biorąc, nazwy skryptów PHP mają rozszerzenie pozwalające serwerowi WWW na rozpoznanie, że wymagają wywołania interpretera PHP. Jeżeli użyjesz rozszerzenia nierozpoznawanego przez serwer WWW, wtedy skrypt PHP zostanie wyświetlony w postaci zwykłego tekstu. Skrypty przedstawione w rozdziale używają rozszerzenia `.php`. Informacje dotyczące konfiguracji serwera WWW Apache, aby rozpoznawał inne rozszerzenia, znajdziesz w dodatku A, zatytułowanym „Oprogramowanie wymagane do użycia tej książki”. (Jeżeli nie masz dostępu do plików instalacyjnych Apache, skontaktuj się z administratorem

systemu i zapytaj o obsługiwane rozszerzenia plików). W wymienionym dodatku przedstawiono także konfigurację serwera Apache w taki sposób, aby każdy skrypt o nazwie *index.php* był traktowany jako strona domyślna dla katalogu, w którym znajduje się dany plik. W podobny sposób Apache traktuje pliki o nazwie *index.html*.

W celu użycia skryptów opracowanych w tym rozdziale konieczne jest ich umieszczenie w lokalizacji, do której dostęp ma serwer WWW. Przyjęto konwencję, że projekty ocen uczniów i Ligi Historycznej mają własne, dedykowane im katalogi o nazwach odpowiednio *gp* i *ushl*, położone w katalogu głównym drzewa dokumentów Apache. Jeśli chcesz używany serwer WWW skonfigurować w taki sposób, to teraz powinieneś utworzyć wymienione katalogi. W przypadku serwera działającego w komputerze lokalnym adresy URL prowadzące do wymienionych katalogów rozpoczynają się następująco:

```
http://localhost/ushl/...  
http://localhost/gp/...
```

Na przykład, strony główne w poszczególnych katalogach mogą znajdować się w plikach *index.php* i być dostępne w postaci poniższych adresów:

```
http://localhost/ushl/index.php  
http://localhost/gp/index.php
```

Jeżeli skonfigurowałeś serwer Apache do użycia pliku *index.php* jako strony domyślnej dla katalogu, poniższe adresy URL są odpowiednikami przedstawionych powyżej:

```
http://localhost/ushl/  
http://localhost/gp/
```

Pamiętaj, że w przykładach przedstawionych w rozdziale musisz zmienić adresy URL w taki sposób, aby prowadziły do serwera WWW używanego przez Ciebie, a nie do *localhost*.

9.1. Ogólny opis PHP

Podstawowym zadaniem PHP jest interpretacja skryptu w celu wygenerowania strony internetowej wysyłanej klientowi. Skrypt PHP zwykle zawiera połączenie kodu HTML oraz wykonywalnego. Kod HTML jest wysyłany klientowi bez modyfikacji, natomiast kod PHP jest wykonywany i zastępowany wygenerowanymi przez niego danymi wyjściowymi. Dlatego też klient nigdy nie widzi kodu PHP, a jedynie wygenerowaną przez niego stronę HTML. (Skrypty PHP opracowane w tym rozdziale generują strony w postaci doskonale sformatowanego XHTML, a nie zwykłego HTML. Krótkie omówienie XHTML przedstawiono w punkcie 8.4.2.2, zatytułowanym „Generowanie danych wyjściowych w postaci stron internetowych”).

Kiedy interpreter PHP rozpoczyna odczyt pliku, wtedy po prostu napotkane dane kopiuje do danych wyjściowych, o ile zawartość pliku przedstawia dosłowny tekst, na przykład jest treścią w formacie HTML. Po napotkaniu specjalnego znacznika otwierającego następuje przejście z trybu kopiowania tekstu do trybu kodu PHP i rozpoczęcie

interpretacji pliku jako kodu PHP przeznaczonego go wykonania. Interpreter powraca do trybu kopiowania tekstu po napotkaniu innego znacznika specjalnego wskazującego koniec kodu PHP. Dzięki takiemu podejściu można łączyć tekst statyczny (HTML) z dynamicznie wygenerowanymi wynikami (dane wyjściowe kodu PHP) w celu ostatecznego przygotowania strony zawierającej treść zależną od warunków, w jakich nastąpiło wywołanie strony. Na przykład, skrypt PHP można wykorzystać do przetworzenia wyników formularza, w którym użytkownik podał parametry dla operacji wyszukiwania w bazie danych. W zależności od wartości wprowadzonych przez użytkownika, wspomniane parametry wyszukiwania mogą być różne po każdym wysłaniu formularza. Dlatego też, gdy skrypt przeprowadza operację wyszukiwania i wyświetla informacje żądane przez użytkownika, każda strona wyników będzie inna.

Przekonajmy się, jak działa PHP. Na początek spójrzmy na wyjątkowo prosty skrypt:

```
<html>
<body>
<p>Witaj, świecie</p>
</body>
</html>
```

Przedstawiony powyżej skrypt jest *tak* prosty, że nie zawiera żadnego kodu PHP! Mógłbyś zapytać: co w tym dobrego? To rozsądne pytanie. Odpowiedź jest następująca: czasami użyteczne jest przygotowanie skryptu zawierającego jedynie kod HTML strony, która ma zostać wygenerowana, a dopiero później dodanie do niej kodu PHP. To jest jak najbardziej dozwolone rozwiązanie, a interpreter PHP nie ma problemu z obsługą tego rodzaju kodu.

W celu umieszczenia kodu PHP w skrypcie trzeba go odróżnić od pozostałego tekstu za pomocą specjalnych znaczników otwierających i zamykających `<?php` i `?>`. Kiedy interpreter PHP napotka znacznik otwierający `<?php`, to następuje przejście z trybu kopiowania tekstu do trybu kodu PHP. Odtąd, aż do chwili wystąpienia znacznika zamykającego `?>`, cała napotkana treść jest traktowana jako kod wykonywalny. Kod między znacznikami jest interpretowany i zastępowany przez wygenerowane dane wyjściowe. Przedstawiony wcześniej przykład można zastąpić poniższym, w którym umieszczono niewielką sekcję kodu PHP:

```
<html>
<body>
<p><?php print ("Witaj, świecie"); ?></p>
</body>
</html>
```

W powyższym przykładzie sekcja kodu jest naprawdę minimalna i składa się tylko z pojedynczego wiersza. Po wykonaniu kodu nastąpi wygenerowanie danych wyjściowych Witaj, świecie, które stają się częścią danych wyjściowych wysyłanych do przeglądarki internetowej klienta. Dlatego też strona internetowa wygenerowana przez ten skrypt jest odpowiednikiem strony wygenerowanej przez wcześniejszy skrypt, składający się jedynie z kodu HTML.

Kod PHP może wygenerować dowolną część strony internetowej. Spotkaliśmy się już z jednym przypadkiem ekstremalnym, w którym cały skrypt składał się z dosłownego kodu HTML i nie zawierał żadnego kodu PHP. Innym przypadkiem ekstremalnym jest całkowite wygenerowanie kodu HTML za pomocą kodu PHP:

```
<?php
print("<html>\n");
print("<body>\n");
print("<p>Witaj, świecie</p>\n");
print("</body>\n");
print("</html>\n");
?>
```

Powyższe przykłady pokazują, jak ogromną elastyczność daje PHP w zakresie generowania danych wyjściowych. PHP pozostawia programiście decyzję o stopniu połączenia kodu HTML i PHP. Język PHP jest na tyle elastyczny, że nie wymaga umieszczenia całego kodu w jednym miejscu. Możesz dowolnie przełączać między trybami tekstu i kodu PHP w skrypcie, tak często jak to konieczne.

PHP zapewnia obsługę stylu znaczników innego niż `<?php` i `?>`, który jest wykorzystywany w przykładach przedstawionych w tym rozdziale.

Samodzielne skrypty PHP

Przykładowe skrypty przedstawione w tym rozdziale zostały utworzone z założeniem, że będą uruchamiane przez serwer WWW w celu wygenerowania strony internetowej. Jednak jeśli posiadasz samodzielną wersję PHP, możesz ją wykorzystać do uruchamiania skryptów PHP z poziomu wiersza poleceń. Przyjmijmy założenie, że mamy skrypt o nazwie *hello.php* i następującej zawartości:

```
<?php print ("Witaj, świecie\n"); ?>
```

Aby uruchomić ten skrypt z poziomu wiersza poleceń, trzeba wydać poniższe polecenie:

```
% php hello.php
Witaj, świecie
```

Takie rozwiązanie czasami jest użyteczne podczas pracy nad skryptem, ponieważ od razu można się przekonać, czy zawiera jakiegolwiek błędy składni lub inne problemy. Wspomniane sprawdzenie odbywa się bez konieczności żądania skryptu z poziomu przeglądarki internetowej po każdej wprowadzonej zmianie.

9.1.1. Prosty skrypt PHP

Jeżeli język PHP zapewniałby jedynie możliwość wygenerowania kodu będącego w zasadzie statycznym HTML, to nie byłby aż tak bardzo użyteczny. Prawdziwa potęga PHP kryje się w możliwości generowania stron dynamicznych: dane wyjściowe mogą się różnić między kolejnymi wywołaniami skryptu. Skrypt omówiony w tym punkcie stanowi prosty przykład wspomnianych możliwości. To również stosunkowo krótki skrypt, ale nieco bardziej skomplikowany od poprzedniego. Pokazuje, jak łatwo można uzyskać dostęp do bazy danych MySQL z poziomu PHP i użyć na stronie wyników zapytania.

Ten skrypt stanowi podstawę dla strony głównej witryny internetowej Ligi Historycznej. W dalszej części rozdziału rozbudujemy nieco ten skrypt, który w obecnej postaci wyświetla jedynie krótkie powitanie i liczbę członków Ligi:

```
<html>
<head>
<title>Liga Historyczna</title>
</head>
<body bgcolor="white">
<h2>Liga Historyczna</h2>
<p>Witamy na witrynie internetowej Ligi Historycznej.</p>
<?php
# Strona główna Ligi Historycznej.

try
{
    $dbh = new PDO("mysql:host=localhost;dbname=sampdb", "sampadm", "secret");
    $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $sth = $dbh->query("SELECT COUNT(*) FROM member");
    $count = $sth->fetchColumn(0);
    print("<p>Aktualnie Liga ma $count członków.</p>");
    $dbh = NULL; # Zamknięcie połączenia.
}
catch(PDOException $e) { } # Pusty uchwyt (przechwycony, ale błędy są ignorowane).
?>
</body>
</html>
```

Komunikat powitalny jest statycznym tekstem, więc najłatwiej zapisać go w postaci dosłownego kodu HTML. Jednak liczba członków jest obliczana dynamicznie i ulega zmianie, więc musi być ustalona w trakcie generowania strony. Odbyna się to przez wykonanie zapytania do tabeli member w bazie danych sampdb. W celu obliczenia liczby członków kod między otwierającym i zamykającym znacznikiem skryptu wykonuje następujące kroki:

1. Nawiązanie połączenia z serwerem MySQL i wybranie sampdb jako domyślnej bazy danych.
2. Włączenie wyjątków dla kolejnych wywołań PDO, aby ewentualne błędy mogły być łatwo przechwytywane bez konieczności przeprowadzania testów pod ich kątem.
3. Wykonanie zapytania do serwera w celu ustalenia aktualnej liczby członków Ligi Historycznej (to jest po prostu liczba rekordów w tabeli member).
4. Użycie wyniku zapytania do przygotowania komunikatu informującego o liczbie członków.
5. Zamknięcie połączenia z serwerem MySQL.

Przedstawiony powyżej skrypt znajdziesz w dystrybucji sampdb w katalogu *phpapi/ushl* pod nazwą *index.php*. Zmień odpowiednio parametry połączenia, umieść plik skryptu w katalogu *ushl* serwera WWW, a następnie wywołaj go z poziomu przeglądarki

internetowej, podając jeden z poniższych adresów URL (nazwę komputera i ścieżkę dostępu dopasuj do używanego serwera WWW):

```
http://localhost/ushl/  
http://localhost/ushl/index.php
```

Podzielimy teraz skrypt na fragmenty i przeanalizujemy sposób jego działania. Pierwszym krokiem jest nawiązanie połączenia z serwerem:

```
$dbh = new PDO("mysql:host=localhost;dbname=sampdb", "sampadm", "secret");
```

Konstrukcja `new PDO()` powoduje wywołanie konstruktora klasy PDO. Konstruktor próbuje nawiązać połączenie z serwerem bazy danych i zgłasza wyjątek w przypadku niepowodzenia. Jeśli próba zakończy się sukcesem, konstruktor zwróci obiekt PDO działający w charakterze uchwytu do bazy danych.

Pierwszym argumentem `new PDO()` jest ciąg tekstowy określany mianem „nazwy źródła danych”. Drugi i trzeci argument to odpowiednio nazwa użytkownika i hasło używane podczas nawiązywania połączenia z serwerem (w omawianym przykładzie to `sampadm` i `secret`). Ciąg tekstowy DSN wskazuje PDO sterownik do użycia oraz parametry charakterystyczne dla wybranego sterownika. Dla bazy danych MySQL sterownik nosi nazwę `mysql`, natomiast parametrami są nazwa komputera, w którym działa serwer, i baza danych wskazana jako domyślna. Pokazany DSN wskazuje, że komputerem serwera bazy danych MySQL jest `localhost`, natomiast domyślną bazą danych jest `sampdb`. Oba parametry umieszczone po dwukropku są opcjonalne. Wartością domyślną dla `host` jest `localhost`, więc tak naprawdę ten parametr można pominąć. Jeśli pominiesz `dbname`, to nie zostanie wskazana domyślna baza danych. (DSN może przyjąć także inne formy, dozwolone są również inne parametry).

Prawdopodobnie niepokoiś się osadzeniem w skrypcie nazwy użytkownika i hasła, ponieważ są one doskonale widoczne dla wszystkich. Faktycznie, powinieneś się tym niepokoić. Pamiętaj jednak, że nazwa użytkownika i hasło nie pojawiają się na przekazywanej klientowi wygenerowanej stronie internetowej, ponieważ zawartość skryptu jest zastępowana jego danymi wyjściowymi. Jeśli serwer WWW zostanie błędnie skonfigurowany i nie rozpozna, że skrypt powinien być przetworzony przez PHP, wtedy skrypt będzie wysłany w postaci zwykłego tekstu, co spowoduje ujawnienie parametrów połączenia. Ten problem rozwiązaliśmy w punkcie 9.1.2, zatytułowanym „Użycie biblioteki plików PHP w celu hermetyzacji kodu”.

Zwrócony przez `new PDO()` uchwyt do bazy danych jest wykorzystywany w trakcie dalszej pracy z serwerem MySQL, na przykład w celu przekazywania zapytań SQL do wykonania. Po udanym nawiązaniu połączenia z serwerem domyślny tryb błędów dla wywołań PDO to ukrywanie błędów, co wymaga wyraźnego sprawdzania pod kątem wystąpienia błędów. Aby ułatwić obsługę błędów, skrypt włącza zgłaszanie wyjątków po wystąpieniu błędów PDO:

```
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Po włączeniu wyjątków konstrukcja try/catch może być stosowana w celu „przekierowania” błędów do procedury obsługi wyjątków bez konieczności przeprowadzania wyraźnych operacji sprawdzania, czy wystąpił jakikolwiek błąd. Jeśli nie będziesz używał bloku try/catch, wyjątek spowoduje przerwanie wykonywania skryptu.

Następnie skrypt przekazuje do serwera zapytanie ustalające liczbę członków Ligi. Odbывается to za pomocą wywołania metody query() uchwytu bazy danych, a później wyodrębnienia i wyświetlenia wyniku:

```
$sth = $dbh->query ("SELECT COUNT(*) FROM member");
$count = $sth->fetchColumn (0);
print ("<p>Aktualnie Liga ma $count członków.</p>");
```

Metoda query() przekazuje serwerowi zapytanie do wykonania. Zwróć uwagę na brak średnika lub sekwencji \g bądź \G w ciągu tekstowym zapytania. To przeciwieństwo klienta mysql, w którym wymienione znaki są używane. Metoda query() jest stosowana do wykonywania zapytań zwracających rekordy. (W przypadku zapytań modyfikujących rekordy należy użyć metody o nazwie exec()). Wartością zwrótną metody query() jest obiekt PDOStatement, będący uchwytym zapytania przeznaczonym do przeprowadzania operacji na zbiorze wynikowym.

W przypadku przedstawionego powyżej zapytania zbiór wynikowy składa się z pojedynczego rekordu zawierającego jedną kolumnę, której wartość określa aktualną liczbę członków Ligi. Skrypt wywołuje metodę fetchColumn() obiektu \$sth w celu pobrania rekordu i wyodrębnienia pierwszej kolumny (czyli kolumny 0).

Po wyświetleniu wartości skrypt zamyka połączenie z serwerem przez przypisanie wartości NULL uchwytowi bazy danych. To jest krok opcjonalny. Jeśli nie zamkniesz połączenia, PHP zrobi to automatycznie w chwili zakończenia działania skryptu.

Kod odpowiedzialny za współpracę z bazą danych MySQL został umieszczony w bloku try, aby ewentualne wyjątki zgłoszone na skutek błędów mogły zostać przechwycone i obsłużone przez blok catch. Jeżeli próba nawiązania połączenia zakończy się niepowodzeniem, automatycznie nastąpi zgłoszenie wyjątku. Wywołanie setAttribute() włącza zgłaszanie wyjątków dla kolejnych wywołań PDO, które zakończą się niepowodzeniem. W omawianym przykładzie blok catch jest pusty, a więc powoduje przechwytywanie i ignorowanie błędów. Skrypt nie wyświetla żadnych komunikatów w przypadku wystąpienia błędów, ponieważ podanie aktualnej liczby członków Ligi jest tylko dodatkiem do powitania prezentowanego na stronie głównej. Jakikolwiek komunikat błędu w tym kontekście po prostu byłby mylący dla użytkowników odwiedzających witrynę internetową. Inne sposoby obsługi błędów będą omówione w punkcie 9.1.8, zatytułowanym „Obsługa błędów”.

Zmienne w PHP

W PHP zmienne można utworzyć po prostu przez ich użycie. W omawianym skrypcie generującym stronę główną wykorzystano trzy zmienne: \$dbh, \$sth i \$count; żadna z nich nie została wcześniej zadeklarowana. (Zdarzają się sytuacje, w których deklarujesz zmienne, na przykład w procedurze obsługi wyjątków lub podczas odwoływania się wewnątrz funkcji do zmiennej globalnej).

Zmienne są oznaczane znakiem dolara umieszczonym na początku ich nazwy. Znak \$ umieszczany jest niezależnie od rodzaju wartości przechowywanej przez zmienne, choć w przypadku tablic i obiektów trzeba podjąć dodatkowe kroki w celu uzyskania dostępu do wartości poszczególnych elementów. Jeżeli zmienna `$x` przedstawia pojedynczą wartość skalarną, taką jak liczba lub ciąg tekstowy, dostęp do niej odbywa się po prostu za pomocą `$x`. Natomiast jeśli `$x` przedstawia tablicę o indeksach liczbowych, dostęp do poszczególnych elementów odbywa się za pomocą `$x[0]`, `$x[1]` itd. Z kolei jeśli `$x` przedstawia tablicę asocjacyjną o indeksach w postaci ciągów tekstowych, na przykład `żółty` lub `duży`, dostęp do jej elementów odbywa się za pomocą `$x["żółty"]` i `$x["duży"]`. Tablice PHP mogą mieć elementy liczbowe i asocjacyjne, na przykład `$x[1]` i `$x["duży"]` mogą być elementami tej samej tablicy. Jeśli `$x` przedstawia obiekt, do jego właściwości uzyskujesz dostęp za pomocą `$x->nazwa_właściwości`. Na przykład, `$x->żółty` i `$x->duży` mogą być właściwościami `$x`. Liczby nie są poprawnymi nazwami właściwości, o ile nie zostają ujęte w nawiasy klamrowe. Dlatego też `$x->{1}` to prawidłowa konstrukcja w PHP, natomiast `$x->1` już nie. Nawiasy klamrowe mogą być również stosowane w celu odwoływania się do nazw właściwości zawierających spacje lub inne niedozwolone znaki.

9.1.2. Użycie biblioteki plików PHP w celu hermetyzacji kodu

Skrypty PHP różnią się od skryptów DBI pod tym względem, że są umieszczane w drzewie dokumentów serwera WWW, podczas gdy skrypty DBI zwykle są w katalogu `cgi-bin` znajdującym się poza drzewem dokumentów serwera WWW. Z tego powodu powstaje pewne niebezpieczeństwo: błędna konfiguracja serwera może spowodować, że pliki umieszczone w drzewie dokumentów zostaną przekazane klientowi w postaci zwykłego tekstu. W takim przypadku niebezpieczeństwo ujawnienia świata zewnętrznemu nazwy użytkownika i hasła używanych do nawiązania połączenia z serwerem MySQL jest w PHP znacznie większe w przypadku skryptów DBI.

Nasz skrypt generujący stronę główną Ligi Historycznej również cierpi z wymienionego powodu, ponieważ zawiera osadzone wartości w postaci nazwy użytkownika i hasła MySQL. Dlatego też teraz parametry połączenia przeniesiemy poza skrypt, wykorzystując dwie możliwości oferowane przez PHP: funkcje i pliki nagłówkowe. Przygotujemy funkcję o nazwie `sampdb_connect()`, nawiązującą połączenie z bazą danych i zwracającą do niej uchwyt. Wymieniona funkcja zostanie umieszczona w pliku nagłówkowym, czyli pliku biblioteki niebędącym częścią skryptu, ale do którego można się odnieść z poziomu skryptu. Takie podejście ma kilka zalet:

- **Znacznie ułatwia tworzenie kodu odpowiedzialnego za nawiązanie połączenia.** Parametry połączenia są definiowane tylko jeden raz w funkcji pomocniczej `sampdb_connect()`, a nie w każdym skrypcie wymagającym nawiązania połączenia z bazą danych. Przeniesienie tego rodzaju informacji do biblioteki znajdującej się poza skryptem znacznie je upraszcza, ponieważ można skoncentrować się na unikalnych aspektach poszczególnych skryptów bez rozpraszania się kodem odpowiedzialnym za nawiązanie połączenia.

- **Plik nagłówkowy może być wykorzystywany przez wiele skryptów.** W ten sposób promujemy ponowne wykorzystywanie kodu, który na dodatek staje się łatwiejszy w obsłudze. Ponadto, tego rodzaju podejście pozwala na łatwe wprowadzanie globalnych zmian w każdym skrypcie uzyskującym dostęp do biblioteki. Na przykład, po przeniesieniu bazy danych `sampdb` z `localhost` do `boa.example.com` nie trzeba modyfikować wszystkich skryptów nawiązujących połączenie. Zamiast tego zmieniany jest tylko parametr nazwy komputera w pliku nagłówkowym zawierającym implementację funkcji `sampdb_connect()`.
- **Plik nagłówkowy można przenieść poza drzewo dokumentów serwera WWW.** Takie rozwiązanie oznacza, że klient nie może zażądać pliku nagłówkowego bezpośrednio z poziomu przeglądarki internetowej, a więc jego zawartość nie zostanie ujawniona nawet w przypadku błędnej konfiguracji serwera WWW. Wykorzystanie pliku nagłówkowego jest dobrą strategią ukrywania wszelkiego rodzaju informacji wrażliwych, których ujawniania przez serwer WWW nie chcesz. To rozwiązanie na pewno zapewnia większe bezpieczeństwo niż zastosowane w poprzednio omówionym skrypcie, ale nie wolno przyjmować założenia, że nazwa użytkownika i hasło są już absolutnie bezpieczne. Inni użytkownicy posiadający konta w komputerze serwera WWW (a tym samym dostęp do jego systemu plików) będą mogli bezpośrednio odczytać zawartość pliku nagłówkowego, o ile się przed tym nie zabezpieczymy. W punkcie 8.4.3, zatytułowanym „Nawiązanie połączenia z serwerem MySQL z poziomu skryptu sieciowego”, przedstawiono pewne wskazówki dotyczące instalacji plików konfiguracyjnych DBI w sposób chroniący je przed innymi użytkownikami. Zaprezentowane tam informacje mają również zastosowanie w przypadku plików nagłówkowych PHP.

W celu użycia plików nagłówkowych konieczne jest miejsce przeznaczone do ich przechowywania. Ponadto, wspomnianą lokalizację trzeba wskazać PHP. Jeżeli używany przez Ciebie system ma już tego rodzaju lokalizację, możesz ją wykorzystać. W przeciwnym razie zastosuj przedstawioną poniżej procedurę definiowania lokalizacji dla plików nagłówkowych:

1. Utwórz nowy katalog poza drzewem dokumentów serwera WWW; ten katalog będzie przeznaczony do przechowywania plików nagłówkowych PHP.
W omawianym przykładzie użyjemy `/usr/local/apache/lib/php`, znajdującego się poza drzewem dokumentów (`/usr/local/apache/htdocs`).
2. Z poziomu skryptu dostęp do pliku nagłówkowego odbywa się za pomocą jego pełnej ścieżki dostępu lub jedynie nazwy bazowej (ostatni komponent ścieżki dostępu), o ile skonfigurowano ścieżkę wyszukiwania dla PHP. To drugie podejście jest znacznie wygodniejsze, ponieważ PHP samodzielnie zajmie się wyszukiwaniem pliku. Wspomniana ścieżka wyszukiwania używana przez PHP podczas szukania plików nagłówkowych jest wartością opcji konfiguracyjnej `include_path` w pliku inicjalizacyjnym PHP, czyli `php.ini`. Odszukaj ten plik w systemie (najczęściej znajduje się w katalogu `/etc/php` lub `/usr/local/lib`), a następnie wiersz `include_path`.

Jeżeli nie ma wartości, podaj w niej pełną ścieżkę dostępu do nowo utworzonego katalogu dla plików nagłówkowych:

```
include_path = "/usr/local/apache/lib/php"
```

Jeżeli opcja `include_path` ma przypisaną wartość, nowo utworzony katalog dodaj do już istniejącej wartości:

```
include_path = "/usr/local/apache/lib/php:wartość_bieżąca"
```

W systemach UNIX użyj dwukropka w celu oddzielenia katalogów wymienionych w opcji `include_path`. Z kolei w Windows separatorem katalogów jest średnik.

Po przeprowadzeniu modyfikacji pliku *php.ini* trzeba ponownie uruchomić serwer Apache, aby wprowadzone zmiany zostały zastosowane.

Pliki nagłówkowe w PHP są używane analogicznie jak pliki nagłówkowe w języku C. Na przykład, sposób, w jaki PHP może szukać plików nagłówkowych w różnych katalogach, jest podobny do sposobu, w jaki preprocesor C przeszukuje wiele katalogów pod kątem plików nagłówkowych C.

3. Przygotuj plik nagłówkowy, którego chcesz używać, a następnie umieść go w nowo utworzonym katalogu. Plik powinien mieć wyróżniającą go nazwę; w omawianym przykładzie użyto *sampdb_pdo.php*. Ostatecznie, wymieniony plik będzie zawierał wiele funkcji, ale na początek umieścimy w nim jedynie funkcję `sampdb_connect()`:

```
<?php
# sampdb_pdo.php - Funkcje przeznaczone do obsługi bazy danych sampdb za pomocą PDO
# w skryptach PHP.

# Funkcja używająca tajnej nazwy użytkownika i hasła w celu nawiązania
# połączenia z serwerem MySQL i pracy z bazą danych sampdb. Ponadto,
# włącza zgłaszanie wyjątków dla błędów, które wystąpią w kolejnych wywołaniach PDO.
# Wartością zwrótną funkcji jest uchwyt bazy danych utworzony przez new PDO().
function sampdb_connect ()
{
    $dbh = new PDO("mysql:host=localhost;dbname=sampdb",
                  "sampadm", "secret");
    $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    return ($dbh);
}
?>
```

Funkcja `sampdb_connect()` nawiązuje połączenie z serwerem bazy danych przez utworzenie nazwy źródła danych i przekazanie jej do wywołania `new PDO()` wraz nazwą użytkownika i hasłem konta w MySQL. Następnie włączony zostaje tryb obsługi błędów, w którym błąd PDO powoduje zgłoszenie wyjątku. Wartością zwrótną funkcji jest uchwyt do bazy danych wykorzystywany w trakcie późniejszej współpracy z serwerem. Omówionej funkcji można użyć następująco:

```
$dbh = sampdb_connect ();
```

Obsługa wyjątków została włączona w funkcji `sampdb_connect()`, ponieważ to jest znacznie wygodniejsze rozwiązanie niż włączanie ich oddzielnie w każdym skrypcie używającym danego pliku biblioteki.

Zwróć uwagę, że kod PHP w pliku *sampdb_pdo.php* został umieszczony między znacznikami `<?php i ?>`. Interpreter PHP rozpoczyna odczyt plików nagłówkowych w trybie kopiowania tekstu. Dlatego też pominięcie znaczników spowoduje wysłanie pliku w postaci zwykłego tekstu zamiast jego interpretacji jako kodu PHP. To nie ma znaczenia, jeśli plik ma wygenerować dosłowny kod HTML, ale jeśli chcesz, aby zawartość pliku nagłówkowego została wykonana, kod PHP musi być umieszczony między wymienionymi znacznikami.

4. Aby odnieść się do pliku nagłówkowego z poziomu skryptu, należy użyć jednego z poniższych poleceń:

```
include "sampdb_pdo.php";
require "sampdb_pdo.php";
include_once "sampdb_pdo.php";
require_once "sampdb_pdo.php";
```

PHP w następujący sposób obsługuje wymienione polecenia:

- Polecenia `include` i `require` dołączają i przetwarzają zawartość wskazanego pliku. Różnica między nimi polega na tym, że jeśli pliku nie można odnaleźć, to polecenie `include` wyświetla komunikat ostrzeżenia i działanie skryptu jest kontynuowane, natomiast `require` zgłasza błąd i przerywa działanie skryptu.
- Polecenia `include_once` i `require_once` działają podobnie do `include` i `require`, za wyjątkiem faktu, że jeśli PHP już odczytał zawartość wymienionego skryptu, to plik nie będzie ponownie odczytywany. To jest użyteczne rozwiązanie, gdy plik nagłówkowy powoduje dołączenie innych plików. Unikamy w ten sposób wielokrotnego dołączania plików i powstawania błędów na skutek ponownej definicji tych samych funkcji.

Skrypty przedstawione w tym rozdziale używają polecenia `require_once`. Kiedy PHP napotyka polecenie dołączenia pliku, wtedy wyszukuje go, a następnie odczytuje zawartość wskazanego pliku. Zawartość pliku staje się dostępna dla pozostałej części skryptu.

Dystrybucja *sampdb* zawiera plik *sampdb_pdo.php* umieszczony w katalogu *phpapi*. Zmodyfikuj odpowiednio parametry połączenia, aby odpowiadały tym, których używasz do nawiązania połączenia z serwerem MySQL. Następnie skopiuj plik do przygotowanego katalogu plików nagłówkowych i zmień uprawnienia do pliku w taki sposób, aby był możliwy do odczytu jedynie przez serwer WWW (i żadnych innych użytkowników).

Teraz możemy zmodyfikować stronę główną Ligi Historycznej. Umieszczamy tutaj odniesienie do pliku nagłówkowego *sampdb_pdo.php*, a połączenie z serwerem MySQL nawiązujemy za pomocą funkcji `sampdb_connect()`:

```
<html>
<head>
<title>Liga Historyczna</title>
```

```

</head>
<body bgcolor="white">
<h2>Liga Historyczna</h2>
<p>Witamy na witrynie internetowej Ligi Historycznej.</p>
<?php
# Strona główna Ligi Historycznej - wersja 2.

require_once "sampdb_pdo.php";

try
{
    $dbh = sampdb_connect ();
    $sth = $dbh->query ("SELECT COUNT(*) FROM member");
    $count = $sth->fetchColumn (0);
    print ("<p>Aktualnie Liga ma $count członków.</p>");
    $dbh = NULL; # Zamknięcie połączenia.
}
catch (PDOException $e) { } # Pusty uchwyt (przechwycony, ale błędy są ignorowane).
?>
</body>
</html>

```

Powyższy skrypt został zapisany pod nazwą *index2.php* w katalogu *phpapi/ushl* dystrybucji *sampdb*. Skopiuuj go do katalogu *ushl* w serwerze WWW, zmień nazwę na *index.php* w celu zastąpienia wymienionego pliku nową treścią. W ten sposób mniej bezpieczna wersja strony głównej zostaje zastąpiona znacznie bezpieczniejszą, ponieważ nowy plik nie zawiera nazwy użytkownika i hasła do konta MySQL.

Być może sądzisz, że wersja strony głównej wykorzystująca plik nagłówkowy tak naprawdę nie zmniejszyła ilości kodu koniecznego do utworzenia, a więc gdzie wspomniane wcześniej korzyści w tym zakresie? Poczekaj chwilę. W pliku *sampdb_pdo.php* znajdują się jeszcze inne funkcje, a tym samym plik stanie się wygodnym repozytorium dla wszelkich procedur, które mogą być użyteczne w wielu skryptach. W rzeczywistości już teraz możemy utworzyć dwie dodatkowe funkcje i umieścić je w pliku. Każdy skrypt sieciowy tworzony w pozostałej części rozdziału będzie generował całkiem stereotypowy zestaw znaczników HTML na początku i na końcu strony. Zamiast tworzyć te znaczniki w każdym skrypcie, możemy przygotować funkcje *html_begin()* i *html_end()* generujące je automatycznie. Funkcja *html_begin()* może pobierać kilka argumentów, określających tytuł strony i nagłówek. Poniżej przedstawiono kod dwóch wymienionych funkcji:

```

function html_begin ($title, $header)
{
    print ("<html>\n");
    print ("<head>\n");
    if ($title != "")
        print ("<title>$title</title>\n");
    print ("</head>\n");
    print ("<body bgcolor=\"white\">\n");
    if ($header != "")
        print ("<h2>$header</h2>\n");
}

function html_end ()

```

```
{
    print ("</body>\n");
    print ("</html>\n");
}
```

Po umieszczeniu funkcji `html_begin()` i `html_end()` w pliku *sampdb_pdo.php* stronę główną Ligi Historycznej można zmodyfikować w celu wykorzystania tych funkcji. Zmodyfikowany skrypt (*index3.php*) przedstawia się następująco:

```
<?php
# Strona główna Ligi Historycznej - wersja 3.

require_once "sampdb_pdo.php";

$title = "Liga Historyczna";
html_begin ($title, $title);
?>

<p>Witamy na witrynie internetowej Ligi Historycznej.</p>

<?php
try
{
    $dbh = sampdb_connect ();
    $sth = $dbh->query ("SELECT COUNT(*) FROM member");
    $count = $sth->fetchColumn (0);
    print ("<p>Aktualnie Liga ma $count członków.</p>");
    $dbh = NULL; # Zamknięcie połączenia.
}
catch (PDOException $e) { } # Pusty uchwyt (przechwycony, ale błędy są ignorowane).

html_end ();
?>
```

Zwróć uwagę, że kod PHP został podzielony na dwie części, między którymi znajduje się komunikat powitalny w postaci dosłownego tekstu HTML. Plik *index3.php* skopiuj do katalogu *ushl* w serwerze WWW i zmień mu nazwę na *index.php*, zastępując tym samym istniejący plik nową wersją.

Użycie funkcji generujących początkową i końcową część strony to ważna cecha. W celu zmiany wyglądu nagłówka lub stopki strony wystarczy odpowiednio zmodyfikować te funkcje, a zmiany będą automatycznie uwzględnione we wszystkich skryptach korzystających z wymienionych funkcji. Na przykład, przyjmujemy założenie, że chcesz umieścić informację o treści Copyright USHL na dole każdej strony Ligi Historycznej. W takim przypadku tę informację wystarczy umieścić gdzieś w pobliżu końca funkcji `html_end()`.

9.1.3. Prosta strona pobierająca dane

Skrypt osadzony na stronie głównej Ligi Historycznej wykonuje zapytanie zwracające tylko pojedynczy rekord (aktualną liczbę członków). Nasz kolejny skrypt pokaże, jak można przetworzyć zbiór wyników składający się z wielu rekordów (pełną zawartość tabeli *member*). Ten skrypt będzie odpowiednikiem skryptu Perl DBI o nazwie *dump_members.pl*,

opracowanego w punkcie 8.2.2, zatytułowanym „Prosty skrypt DBI”. Dlatego też skryptowi nadamy nazwę *dump_members.php*. Wersja PHP skryptu różni się od wersji DBI pod tym względem, że została przeznaczona do użycia w środowisku sieciowym, a nie wiersza poleceń. Dlatego też dane wyjściowe są wygenerowane w postaci kodu HTML zamiast tekstu rozdzielonego tabulatorami. W celu eleganckiego sformatowania wierszy i kolumn, w skrypcie *dump_members.php* poszczególne rekordy tabeli member będą wyświetlane jako wiersze w tabeli HTML. Kod skryptu przedstawia się następująco:

```
<?php
# dump_members.php - Wyświetlenie członków Ligi Historycznej w tabeli HTML.

require_once "sampdb_pdo.php";

$title = "Lista członków Ligi Historycznej";
html_begin ($title, $title);

$dbh = sampdb_connect ();

# Wykonanie zapytania.
$stmt = "SELECT last_name, first_name, suffix, email,"
        . " street, city, state, zip, phone FROM member ORDER BY last_name";
$stmt = $dbh->query ($stmt);

print("<table>\n");           # Początek tabeli.
# Odczyt wyników zapytania, a następnie usunięcie danych z pamięci.
while ($row = $stmt->fetch (PDO::FETCH_NUM))
{
    print("<tr>\n");           # Początek wiersza tabeli.
    for ($i = 0; $i < $stmt->columnCount (); $i++)
    {
        # Cytowanie wszystkich znaków specjalnych i wygenerowanie komórki tabeli.
        print("<td>" . htmlspecialchars ($row[$i]) . "</td>\n");
    }
    print("</tr>\n");         # Koniec wiersza tabeli.
}
print("</table>\n");         # Koniec tabeli.

$dbh = NULL; # Zamknięcie połączenia.

html_end ();
?>
```

Funkcja *sampdb_connect()* włącza zgłaszanie wyjątków dla błędów PDO, ale skrypt *dump_members.php* nie zawiera konstrukcji *try/catch* pozwalającej na obsługę wyjątków. Co się stanie, gdy wystąpi błąd? W takim przypadku zachowaniem domyślnym w PHP jest przerwanie wykonywania skryptu i wyświetlenie komunikatu informującego o problemie. To przeciwieństwo strony głównej Ligi Historycznej, na której użyto pustej procedury obsługi wyjątków powodującej zignorowanie błędów. W przypadku strony głównej wyświetlenie aktualnej liczby członków było jedynie niewielkim dodatkiem do podstawowego celu skryptu, czyli wyświetlenia komunikatu powitania. Dlatego też wyświetlenie ewentualnego komunikatu błędu byłoby niepotrzebnym odrywaniem uwagi użytkownika. Z kolei w skrypcie *dump_members.php* wyświetlenie zawartości bazy danych jest jedynym powodem

istnienia skryptu, więc jeśli wystąpi jakikolwiek problem uniemożliwiający wyświetlenie wspomnianych danych, rozsądne jest wyświetlenie komunikatu błędu i poinformowanie użytkownika o jego wystąpieniu.

Po wykonaniu zapytania pobierającego rekordy tabeli `member` skrypt wywołuje metodę `fetch()`, odpowiedzialną za zwrócenie kolejnego rekordu ze zbioru wynikowego. W przypadku jego braku wartością zwrotną jest `FALSE`. Argument `PDO::FETCH_NUM` nakazuje metodzie `fetch()` zwrót rekordu wraz z liczbowo zindeksowanymi kolumnami.

W celu zakodowania wartości przeznaczonych do wyświetlenia na stronie internetowej skrypt `dump_members.php` używa funkcji `htmlspecialchars()`, kodującej znaki, które mają znaczenie specjalne w HTML, na przykład `<`, `>` i `&`. (Do zakodowania wartości przeznaczonych do wstawienia w adresie URL należy użyć funkcji `urlencode()`). Wymienione dwie funkcje PHP są podobne do metod kodujących `escapeHTML()` i `escape()` udostępnianych przez moduł `CGI.pm` w Perlu. Wymienione metody omówiono w podpunkcie 8.4.2.3, zatytułowanym „Kodowanie znaków w tekście HTML i adresach URL”.

Jeśli chcesz wypróbować skrypt `dump_members.php`, umieść go w katalogu `ushl` serwera WWW, a następnie w przeglądarce internetowej wpisz adres:

```
http://localhost/ushl/dump_members.php
```

Aby umożliwić użytkownikom skorzystanie ze skryptu `dump_members.php`, na stronie głównej Ligi Historycznej trzeba umieścić odpowiednie łącze. Zmodyfikowana wersja skryptu (`index4.php`) przedstawia się następująco:

```
<?php
# Strona główna Ligi Historycznej - wersja 4.

require_once "sampdb_pdo.php";

$title = "Liga Historyczna";
html_begin ($title, $title);
?>

<p>Witamy na witrynie internetowej Ligi Historycznej</p>

<?php
try
{
    $dbh = sampdb_connect ();
    $sth = $dbh->query ("SELECT COUNT(*) FROM member");
    $count = $sth->fetchColumn (0);
    print ("<p>Aktualnie Liga ma $count członków.</p>");
    $dbh = NULL; # Zamknięcie połączenia.
}
catch (PDOException $e) { } # Pusty uchwyt (przechwycony, ale błędy są ignorowane).
?>

<p>
W <a href="dump_members.php">tym</a> miejscu możesz przejrzeć listę członków Ligi
Historycznej.
</p>
```



```
<?php
html_end ();
?>
```

Podobnie jak w przypadku wcześniejszych wersji strony, plik *index4.php* skopiuj do katalogu *ushl* w serwerze WWW i zmień jego nazwę na *index.php*, zastępując istniejący plik jego nowszą wersją.

Skrypt *dump_members.php* pokazuje, jak skrypt PHP może pobierać informacje z bazy danych MySQL i konwertować je na postać strony internetowej. Jeśli chcesz, skrypt możesz zmodyfikować, aby generował dane wyjściowe w bardziej dopracowanej postaci. Jedną z możliwych modyfikacji jest wyświetlanie wartości kolumny *email* w postaci aktywnych łączy zamiast statycznego tekstu, co ułatwi użytkownikom wysyłanie wiadomości e-mail do innych członków Ligi. Dystrybucja *sampdb* zawiera skrypt o nazwie *dump_members2.php*, w którym wprowadzono wspomnianą modyfikację. Różnice w stosunku do skryptu *dump_members.php* są niewielkie, dotyczą pętli pobierającej i wyświetlającej informacje o członkach. Początkowa wersja pętli przedstawia się następująco:

```
while ($row = $sth->fetch (PDO::FETCH_NUM))
{
    print ("<tr>\n");          # Początek wiersza tabeli.
    for ($i = 0; $i < $sth->columnCount (); $i++)
    {
        # Cytowanie wszystkich znaków specjalnych i wygenerowanie komórki tabeli.
        print ("<td>" . htmlspecialchars ($row[$i]) . "</td>\n");
    }
    print ("</tr>\n");        # Koniec wiersza tabeli.
}
```

Adresy e-mail członków znajdują się w czwartej kolumnie wyników zapytania i dlatego skrypt *dump_member2.php* traktuje tę kolumnę nieco inaczej od pozostałych. Jeśli nie jest pusta, to jej wartość będzie wyświetlona w postaci łącza:

```
while ($row = $sth->fetch (PDO::FETCH_NUM))
{
    print ("<tr>\n");          # Początek wiersza tabeli.
    for ($i = 0; $i < $sth->columnCount (); $i++)
    {
        print ("<td>");
        # Cytowanie wszystkich znaków specjalnych i wygenerowanie komórki tabeli.
        # Kolumna email jest czwartą (index 3) kolumną w zbiorze wynikowym.
        if ($i == 3 && $row[$i] != "")
        {
            printf ("<a href='mailto:%s'>%s</a>",
                    $row[$i],
                    htmlspecialchars ($row[$i]));
        }
        else
        {
            print (htmlspecialchars ($row[$i]));
        }
        print ("</td>\n");
    }
    print ("</tr>\n");        # Koniec wiersza tabeli.
}
```

9.1.4. Przetwarzanie wyników zapytania

Rozszerzenie PDO oferuje kilka sposobów wykonywania zapytań SQL:

- Obiekt PDO ma metody `exec()` i `query()`, pobierające argument w postaci zapytania SQL, a następnie natychmiast wykonujące to zapytanie. Wartością zwrótną jest wynik:
 - ◆ W przypadku zapytań modyfikujących rekordy, na przykład `DELETE`, `INSERT`, `REPLACE` i `UPDATE`, należy wywołać metodę `exec()`. Jej wartością zwrótną jest liczba wskazująca, ile rekordów zostało zmienionych (odpowiednio usuniętych, wstawionych, zastąpionych lub uaktualnionych).
 - ◆ W przypadku zapytań generujących zbiór wynikowy, na przykład `SELECT`, należy wywołać metodę `query()`, której wartością zwrótną jest obiekt `PDOStatement`, będący uchwytym zapytania. Za pomocą wymienionego obiektu można uzyskać dostęp do zbioru wynikowego. Na przykład, pobranie rekordów zbioru wynikowego odbywa się przez wywołanie metody `fetch()`, natomiast liczbę kolumn w zbiorze wynikowym można ustalić za pomocą wywołania metody `columnCount()`.
- Rozszerzenie PDO obsługuje także dwuetapowe wykonanie zapytania, czyli po prostu zapytania preinterpretowane. Obiekt PDO ma metodę `prepare()`, pobierającą jako argument zapytanie SQL. Zamiast natychmiast wykonać zapytanie, metoda `prepare()` przeprowadza pewne wstępne przetwarzanie zapytania i zwraca obiekt `PDOStatement`, będący uchwytym zapytania. Wspomniany obiekt ma metodę `execute()`, powodującą wykonanie zapytania, a także inne metody służące do przetwarzania wyniku zapytania.

Metody `prepare()` i `execute()` mogą być używane dla wszystkich zapytań. Nie rozróżniają zapytań modyfikujących rekordy od jedynie zwracających zbiór wynikowy.

Zapytania preinterpretowane oferują także ważną możliwość, jaką jest ponowne wykonanie zapytania (co zwiększa wydajność działania), i obsługę znaków specjalnych w wartościach danych. Zapoznaj się z punktem 9.1.6, zatytułowanym „Używanie zapytań preinterpretowanych”.

W przedstawionych poniżej podpunktach znacznie dokładniej przeanalizujemy oferowane przez rozszerzenie PDO możliwości w zakresie wykonywania zapytań. *W tych przykładach przyjęto założenie, że włączono zgłaszanie wyjątków dla błędów PDO.*

9.1.4.1. Obsługa zapytań modyfikujących rekordy

Metoda `exec()` uchwytu bazy danych służy do wykonywania zapytań modyfikujących rekordy. Wartością zwrótną metody `exec()` jest liczba rekordów, których dotyczyło zapytanie. Przyjmujemy założenie, że w tabeli `member` chcesz usunąć rekord dotyczący członka 149, a następnie wyświetlić wynik zapytania. Przedstawiony poniżej fragment kodu pokazuje, jak sprawdzić, czy zapytanie faktycznie usunęło jakikolwiek rekord:

```
$count = $dbh->exec ("DELETE FROM member WHERE member_id = 149");
if ($count > 0)
    print ("Usunięto rekord członka o identyfikatorze 149\n");
else
    print ("Nie znaleziono rekordu członka o identyfikatorze 149\n");
```

9.1.4.2. Obsługa zapytań zwracających zbiór wynikowy

Metoda `query()` uchwytu bazy danych służy do wykonywania zapytań generujących zbiór wynikowy. Wartością zwrótną metody `query()` jest obiekt `PDOStatement` będący uchwytym zapytania, za pomocą którego można uzyskać dostęp do zbioru wynikowego. Wspomniany uchwyt zapytania oferuje wiele użytecznych metod:

- Metoda `fetch()` pobiera kolejne rekordy z wyniku zapytania. Gdy nie ma więcej rekordów do pobrania, zwraca wartość `FALSE`.
- Metoda `fetchColumn()` jest podobna, ale zwraca pojedynczą kolumnę z każdego rekordu.
- Metoda `columnCount()` zwraca liczbę kolumn znajdujących się w zbiorze wynikowym. (Nie ma odpowiadającej jej metody przeznaczonej dla rekordów. Konieczne jest pobranie rekordów, a następnie ich zliczenie).

We wcześniejszych przykładach, dotyczących strony głównej Ligi Historycznej, przekonałeś się, jak za pomocą pojedynczego wywołania metody `fetchColumn()` pobrać wynik w postaci pojedynczej wartości. W przypadku gdy oczekiwane jest otrzymanie wyniku składającego się z wielu rekordów obejmujących wiele kolumn, najczęściej stosowane rozwiązanie polega na użyciu metody `fetch()` w pętli i pobraniu zbioru wynikowego. Poniższy fragment kodu prezentuje tego rodzaju rozwiązanie. Kod zlicza pobierane rekordy w celu ustalenia ich ogólnej liczby.

```
$sth = $dbh->query ("SELECT * FROM member");
# Pobranie wszystkich rekordów zbioru wynikowego.
$count = 0;
while ($row = $sth->fetch (PDO::FETCH_NUM))
{
    # Wyświetlenie wartości rekordu rozdzielonych przecinkami.
    for ($i = 0; $i < $sth->columnCount (); $i++)
        print ($row[$i] . ($i < $sth->columnCount () - 1 ? "," : "\n"));
    $count++;
}
printf ("Liczba zwróconych rekordów: %d\n", $count);
```

Metoda `fetch()` pobiera argument wskazujący rodzaj wartości zwrótej. W tabeli 9.1 wymieniono niektóre z najczęściej używanych argumentów metody `fetch()`.

Argument metody `fetch()` jest opcjonalny, w przypadku braku użyty zostanie tryb domyślny. O ile go nie wyzerujesz, to będzie `PDO::FETCH_BOTH`, więc metoda `fetch()` zwróci rekord w postaci tablicy elementów, do których dostęp można uzyskać za pomocą nazw kolumn lub indeksów liczbowych.

Tabela 9.1. Tryby pobierania rekordów przez metodę fetch()

Argument	Wartość zwrotna
PDO::FETCH_ASSOC	Tablica; dostęp do elementów odbywa się za pomocą indeksu asocjacyjnego.
PDO::FETCH_NUM	Tablica; dostęp do elementów odbywa się za pomocą indeksu liczbowego.
PDO::FETCH_BOTH	Tablica; dostęp do elementów odbywa się za pomocą indeksu asocjacyjnego lub liczbowego.
PDO::FETCH_OBJ	Obiekt; dostęp do elementów odbywa się za pomocą właściwości.

Aby ustawić domyślny tryb pobierania, przed rozpoczęciem podbierania rekordów konieczne jest przekazanie dodatkowego argumentu metodzie query() lub wywołanie polecenia obsługującego metodę setFetchMode(). Wymienione poniżej polecenia powodują ustawienie trybu pobierania jako PDO::FETCH_NUM dla następnych operacji pobierania zbioru wynikowego:

```
$sth = $dbh->query ($stmt, PDO::FETCH_NUM);
```

```
$sth = $dbh->query ($stmt);
$sth->setFetchMode (PDO::FETCH_NUM);
```

W przypadku trybu pobierania PDO::FETCH_ASSOC kolejny rekord zbioru wynikowego metoda fetch() zwróci w postaci tablicy asocjacyjnej. Nazwy elementów są nazwami kolumn wskazanych w zapytaniu. Na przykład, jeśli pobierane są kolumny last_name i first_name z tabeli president, dostęp do kolumn można uzyskać w następujący sposób:

```
$stmt = "SELECT last_name, first_name FROM president";
$sth = $dbh->query ($stmt);
while ($row = $sth->fetch (PDO::FETCH_ASSOC))
    printf ("%s %s\n", $row["first_name"], $row["last_name"]);
```

W przypadku trybu pobierania PDO::FETCH_NUM kolejny rekord zbioru wynikowego metoda fetch() zwróci w postaci tablicy, do której elementów można uzyskać dostęp za pomocą indeksu liczbowego rozpoczynającego się od zera. Aby ustalić liczbę kolumn w zbiorze wynikowym, należy wywołać metodę columnCount() uchwytu zapytania. Przedstawiona poniżej prosta pętla pobiera i wyświetla wartości rekordu w formacie wartości rozdzielonych tabulatorami:

```
$stmt = "SELECT * FROM president";
$sth = $dbh->query ($stmt);
while ($row = $sth->fetch (PDO::FETCH_NUM))
{
    for ($i = 0; $i < $sth->columnCount (); $i++)
        print ($row[$i] . ($i < $sth->columnCount () - 1 ? "\t" : "\n"));
}
```

Tryb pobierania PDO::FETCH_BOTH powoduje, że metoda fetch() zwraca tablicę, do której elementów można uzyskać dostęp za pomocą indeksu liczbowego lub nazw kolumn. Ten tryb jest jak połączenie wymienionych wcześniej PDO::FETCH_NUM i PDO::FETCH_ASSOC.

W przypadku trybu `PDO::FETCH_OBJ` kolejny rekord zbioru wynikowego metoda `fetch()` zwraca w postaci obiektu wraz z właściwościami dostępnymi za pomocą składni `$row->nazwa_kolumny`:

```
while ($row = $sth->fetch (PDO::FETCH_OBJ))
    printf ("%s %s\n", $row->first_name, $row->last_name);
```

Co w sytuacji, jeśli zapytanie zawiera obliczane kolumny? Na przykład, mogło zostać wykonane zapytanie zwracające wartości obliczone na podstawie wyrażenia:

```
SELECT CONCAT(first_name, ' ', last_name) FROM president
```

Tego rodzaju zapytanie jest nieodpowiednie podczas pobierania rekordów w postaci obiektów. Nazwa wskazanej kolumny sama jest wyrażeniem, a to nie jest prawidłowa nazwa właściwości. Jednak poprawną nazwą może być alias utworzony dla kolumny. W poniższym zapytaniu nadano kolumnie alias `full_name`:

```
SELECT CONCAT(first_name, ' ', last_name) AS full_name FROM president
```

Jeżeli pobierzesz każdy rekord powyższego zapytania jako obiekt, dzięki aliasowi będziesz mógł uzyskać dostęp do kolumny za pomocą `$row->full_name`.

W poprzednich przykładach używaliśmy pętli pobierania rekordów w poniższej postaci, w której każdy rekord był przypisywany zmiennej `$row`:

```
while ($row = $sth->fetch ([ fetch_mode]))
    ... obsługa rekordu ...
```

Jednak istnieją jeszcze inne sposoby pobierania rekordów. Jednym z nich jest pobranie tablicy i przypisanie wyniku liście zmiennych. Na przykład, aby przypisać kolumny `last_name` i `first_name` zmiennym o nazwie `$ln` i `$fn` oraz wyświetlić imiona i nazwiska w tej właśnie kolejności, należy użyć poniższego fragmentu kodu:

```
$stmt = "SELECT last_name, first_name FROM president";
$sth = $dbh->query ($stmt);
while (list ($ln, $fn) = $sth->fetch (PDO::FETCH_NUM))
    printf ("%s %s\n", $fn, $ln);
```

Zmienne mogą mieć dowolne dozwolone nazwy, ale ich kolejność wymienienia w metodzie `list()` musi odpowiadać wskazanej w zapytaniu kolejności kolumn.

Istnieje również możliwość pobierania poszczególnych kolumn bezpośrednio do zmiennych PHP. W tym celu za pomocą metody `bindColumn()` należy dołączyć kolumny zbioru wynikowego do zmiennych i pobrać rekordy w trybie `PDO::FETCH_BOUND`. W takim przypadku metoda `fetch()` zwraca wartość `TRUE`, jeśli pozostały jeszcze rekordy do pobrania. Aby dla każdego pobranego rekordu przypisać wartości kolumn do dołączonych zmiennych, należy użyć poniższego fragmentu kodu:

```
$stmt = "SELECT last_name, first_name FROM president";
$sth = $dbh->query ($stmt);
$sth->bindColumn (1, $ln);
$sth->bindColumn (2, $fn);
while ($sth->fetch (PDO::FETCH_BOUND))
    printf ("%s %s\n", $fn, $ln);
```

Jednoczesne pobranie wszystkich rekordów do tablicy umożliwia metoda `fetchAll()`:

```
$rows = $sth->fetchAll ();
```

Podobnie jak w przypadku metody `fetch()`, także `fetchAll()` stosuje domyślny tryb pobierania i akceptuje argument wyraźnie wskazujący tryb pobierania.

Uchwyt zapytania może być użyty jako iterator bez konieczności wyraźnego wywołania metody `fetch()`:

```
foreach ($sth as $row)
    printf ("%s %s\n", $row["first_name"], $row["last_name"]);
```

Domyślny tryb pobierania wpływa na sposób zwrócenia rekordów.

9.1.5. Sprawdzanie pod kątem wartości NULL w wynikach zapytania

Wartość SQL NULL w zbiorze wynikowym PHP przedstawia w postaci wartości PHP NULL. Jednym ze sposobów sprawdzenia, czy kolumna zwrócona przez zapytanie SELECT ma wartość NULL, jest użycie funkcji `is_null()`. Przedstawiony poniżej przykład pobiera z tabeli `member`, a następnie wyświetla imię, nazwisko i adres e-mail członka Ligi. Jeżeli adres ma wartość NULL, wtedy wyświetlany jest komunikat Brak adresu e-mail.

```
$stmt = "SELECT last_name, first_name, email FROM member";
$sth = $dbh->query ($stmt);
while (list ($last_name, $first_name, $email) = $sth->fetch (PDO::FETCH_NUM))
{
    printf ("Imię i nazwisko: %s %s, E-mail: ", $first_name, $last_name);
    if (is_null ($email))
        print ("Brak adresu e-mail.");
    else
        print ($email);
    print ("\n");
}
```

Sprawdzenie pod kątem wartości SQL NULL można przeprowadzić również przez porównanie wartości do stałej PHP NULL za pomocą operatora identyczności (`==`):

```
if ($email == NULL)
    print ("Brak adresu e-mail.");
else
    print ($email);
```

Wartość PHP NULL jest taka sama jak niezdefiniowana wartość, a więc funkcja `isset()` dostarcza kolejny sposób na sprawdzenie, czy wartością jest NULL:

```
if (!isset ($email))
    print ("Brak adresu e-mail.");
else
    print ($email);
```

9.1.6. Używanie zapytań preinterpretowanych

Omówione wcześniej metody `exec()` i `query()` natychmiast wykonują zapytania SQL i zwracają wynik. Rozszerzenie PDO może również w oddzielnych krokach przygotować i wykonać zapytanie. Metoda `prepare()` uchwytu bazy danych pozwala na pobranie uchwytu zapytania, który później jest używany do wykonania zapytania:

```
$sth = $dbh->prepare ($stmt);  
$sth->execute ();
```

Po wykonaniu zapytania modyfikującego rekordy, za pomocą metody `rowCount()` można określić liczbę rekordów, których dotyczyło zapytanie:

```
$count = $sth->rowCount ();
```

Jeżeli zapytanie zwraca rekordy, wtedy można używać metod takich jak `fetch()` i `rowCount()`. W celu określenia liczby rekordów należy je zliczać podczas pobierania. (Metoda `rowCount()` ma zastosowanie jedynie w przypadku zapytań modyfikujących rekordy).

Zapytania preinterpretowane oferują pewne ważne możliwości:

- Ciąg tekstowy zapytania może zawierać miejsca zarezerwowane zamiast dosłownych wartości. Po przygotowaniu zapytania, a przed jego wykonaniem następuje wstawienie konkretnych wartości do miejsc zarezerwowanych. Rozszerzenie PDO zajmuje się obsługą zarówno cytowania znaków specjalnych, jak i wartości NULL. Mamy kilka sposobów wstawiania wartości w miejscach zarezerwowanych; zostały one omówione w punkcie 9.1.7, zatytułowanym „Użycie miejsc zarezerwowanych do obsługi kwestii związanych z cytowaniem danych”.
- Zapytanie preinterpretowane może być wielokrotnie wykonywane. W ten sposób unika się obciążenia związanego z analizą zapytania przed jego wykonaniem. To bardzo użyteczne w przypadku zapytań, które mają być wykonane wielokrotnie, ponieważ znacznie skraca czas ich wykonywania. Na przykład, w celu wstawienia wielu rekordów możesz jednokrotnie wywołać metodę `prepare()` dla zapytania INSERT. Następnie metodę `execute()` można wywoływać w każdej iteracji pętli i dostarczać jej dane dla poszczególnych rekordów, wykorzystując miejsca zarezerwowane do dołączania wartości do zapytania.

9.1.7. Użycie miejsc zarezerwowanych do obsługi kwestii związanych z cytowaniem danych

Podobnie jak w innych językach programowania, takich jak C i Perl, także w PHP występuje konieczność cytowania pewnych znaków w ciągach tekstowych zapytań SQL. Przyjmujemy założenie, że tworzone jest zapytanie wstawiające nowy wiersz do tabeli. W ciągu tekstowym zapytania poszczególne wartości przeznaczone do wstawienia mogły zostać ujęte w cudzysłów:

```
$last = "0'Malley";
$first = "Brian";
$expiration = "2013-09-01";
$stmt = "INSERT INTO member (last_name,first_name,expiration)"
    . " VALUES('$last','$first','$expiration')";
```

Problem polega na ujęciu w cudzysłów wartości zawierającej apostrof (0'Malley), czyli inny znak cytowania. Przekazanie takiego zapytania do serwera MySQL spowoduje wygenerowanie błędu składni. Aby rozwiązać ten problem w języku C, można użyć funkcji `mysql_real_escape_string()` lub `mysql_escape_string()`. Z kolei w skrypcie DBI dostępna jest funkcja `quote()`. Natomiast rozszerzenie PDO do tego samego celu udostępnia metodę `quote()` uchwytu bazy danych. Na przykład, wywołanie `quote("0'Malley")` zwraca wartość `'0\'Malley'`. Aby użyć metody `quote()` do utworzenia zapytania, jej wartość zwrrotną należy wstawić bezpośrednio do ciągu tekstowego zapytania bez dodawania jakichkolwiek znaków cytowania:

```
$last = $dbh->quote ("0'Malley");
$first = $dbh->quote ("Brian");
$expiration = $dbh->quote ("2013-09-01");
$stmt = "INSERT INTO member (last_name,first_name,expiration)"
    . " VALUES($last,$first,$expiration)";
```

Niestety, metoda `quote()` ma pewne wady ograniczające jej użyteczność w porównaniu do oferowanego przez DBI jej odpowiednika o takiej samej nazwie:

- Nie jest implementowana dla wszystkich sterowników; w takim przypadku jej wartością zwrrotną jest `FALSE` zamiast cytowanego ciągu tekstowego.
- W przypadku wartości `NULL` do ciągu tekstowego zapytania chcemy wstawić słowo `NULL` nieujęte w cudzysłów. Jednak przekazanie wartości `NULL` metodzie `quote()` powoduje zwrot pustego ciągu tekstowego (`''`). Aby rozwiązać ten problem, konieczne jest zastosowanie testu i zapewnienie odpowiedniej obsługi wartości w zależności od tego, czy przedstawia ona wartość `NULL`.

Z powodu wymienionych ułomności zalecam unikanie stosowania metody `quote()` za wyjątkiem sytuacji, w których masz pewność, że będą używane jedynie wartości inne niż `NULL`. Lepszym podejściem jest wykorzystanie zapytań preinterpretowanych. Takie rozwiązanie pozwala na wskazanie miejsc zarezerwowanych w zapytaniach SQL, a zadanie cytowania jest pozostawione rozszerzeniu PDO. Kiedy przygotowujesz zapytanie SQL, miejsce wstawienia danych wskazujesz za pomocą znaku zapytania; to jest wspomniane miejsce zarezerwowane. Podczas wykonywania zapytania wartość miejsc zarezerwowanych podajesz w postaci tablicy parametrów:

```
$stmt = "INSERT INTO member (last_name,first_name,expiration) VALUES(?,?,?)";
$stmt = $dbh->prepare ($stmt);
$stmt->execute (array ("0'Malley", "Brian", "2013-09-01"));
```

PDO zajmuje się całą obsługą cytowania znaków specjalnych w ciągach tekstowych i prawidłowo przetwarza wszelkie wartości inne niż ciągi tekstowe, na przykład liczby lub wartości `NULL`.

Innym sposobem dostarczenia wartości danych jest ich wstawienie w miejsca zarezerwowane za pomocą metody `bindValue()` jeszcze przed wywołaniem metody `execute()`:

```
$stmt = "INSERT INTO member (last_name,first_name,expiration) VALUES(?,?,?)";
$stmt = $dbh->prepare ($stmt);
$stmt->bindValue (1, "O'Malley");
$stmt->bindValue (2, "Brian");
$stmt->bindValue (3, "2013-09-01");
$stmt->execute ();
```

Przedstawione powyżej przykłady wykorzystują pozycyjne znaczniki miejsc zarezerwowanych, czyli znaki zapytania, które wszystkie są takie same i mogą być rozróżniane jedynie przez ich położenie w ciągu tekstowym zapytania. Rozszerzenie PDO oferuje również obsługę nazwanych miejsc zarezerwowanych. W takim przypadku poszczególne miejsca zarezerwowane mają nazwę poprzedzoną dwukropkiem. Najpierw przygotowujesz zapytanie do wykonania, a następnie metodzie `execute()` przekazujesz tablicę asocjacyjną zawierającą wartości przeznaczone dla wskazanych miejsc zarezerwowanych:

```
$stmt = "INSERT INTO member (last_name,first_name,expiration)
VALUES(:last_name,:first_name,:expiration)";
$stmt = $dbh->prepare ($stmt);
$stmt->execute (array (
    ":last_name" => "O'Malley",
    ":first_name" => "Brian",
    ":expiration" => "2013-09-01"
));
```

Alternatywne rozwiązanie polega na wstawianiu każdej wartości we wskazane miejsce zarezerwowane przed wywołaniem metody `execute()`:

```
$stmt = "INSERT INTO member (last_name,first_name,expiration)
VALUES(:last_name,:first_name,:expiration)";
$stmt = $dbh->prepare ($stmt);
$stmt->bindValue (":last_name", "O'Malley");
$stmt->bindValue (":first_name", "Brian");
$stmt->bindValue (":expiration", "2013-09-01");
$stmt->execute ();
```

Jedną z zalet nazwanych miejsc zarezerwowanych jest znacznie łatwiejsza możliwość śledzenia powiązań między miejscami zarezerwowanymi i wstawianymi w nich wartościami, gdy stosowana jest większa liczba parametrów.

9.1.8. Obsługa błędów

Podczas pracy z bazą danych MySQL nie wolno zapominać o obsłudze błędów. Jeżeli przyjmiesz założenie, że każde wywołanie zakończy się powodzeniem, znacznie trudniej będzie Ci określić przyczynę nieprawidłowego działania skryptu po wystąpieniu błędu.

Jeżeli próba nawiązania połączenia z serwerem bazy danych za pomocą wywołania `new PDO()` zakończy się niepowodzeniem, zostanie zgłoszony wyjątek. W przypadku

udanego nawiązania połączenia otrzymasz poprawny uchwyt do bazy danych, a ewentualne błędy w kolejnych operacjach PDO będzie przetwarzać zgodnie z wybranym trybem obsługi błędów. Ustawienie wspomnianego trybu obsługi błędów odbywa się następująco:

```
$dbh->setAttribute (PDO::ATTR_ERRMODE, wartość_trybu);
```

PDO zapewnia obsługę trzech wartości trybu:

- `PDO::ERRMODE_SILENT`: PDO jedynie przygotowuje informacje o błędzie dla obiektu, który spowodował wystąpienie błędu. To jest domyślny tryb obsługi błędów.
- `PDO::ERRMODE_WARNING`: ten tryb jest podobny do `PDO::ERRMODE_SILENT`, ale oprócz przygotowania informacji o błędzie rozszerzenie PDO wyświetla komunikat ostrzeżenia.
- `PDO::ERRMODE_EXCEPTION`: po przygotowaniu informacji o błędzie PDO zgłasza wyjątek.

We wszystkich przypadkach, jeśli znasz obiekt, dla którego wystąpił błąd, możesz wywołać metodę `errorCode()` lub `errorInfo()` w celu pobrania informacji o błędzie:

- Metoda `errorCode()` zwraca pięciodziankową wartość `SQLSTATE`. Wartość zwrotna w postaci `PDO::ERR_NONE("00000")` oznacza „brak błędu”.
- Metoda `errorInfo()` zwraca trzelementową tablicę zawierającą wartość `SQLSTATE` oraz charakterystyczne dla sterownika kod i komunikat. W przypadku MySQL dwie ostatnie wartości to kod liczbowy oraz opisowy komunikat błędu.

Jeżeli operacja zakończy się powodzeniem, wartością zwrótną może być tablica zawierająca tylko następującą wartość `SQLSTATE`: `PDO::ERR_NONE("00000")`.

W trybach `PDO::ERRMODE_SILENT` i `PDO::ERRMODE_WARNING` procedura obsługi błędów sprawdza wynik każdej operacji PDO, która może zakończyć się niepowodzeniem. Na przykład:

```
if (!$sth = $dbh->prepare ("SELECT * FROM non_existent_table"))
    die ("Nie można przygotować zapytania: " . $dbh->errorCode () . "\n");
else if (!$sth->execute ())
    die ("Nie można wykonać zapytania: " . $sth->errorCode () . "\n");
```

Zwróć uwagę na wywołanie metody `errorCode()` wraz z uchwyttem prowadzącym do błędu. To samo dotyczy metody `errorInfo()`.

Po włączeniu obsługi wyjątków PHP zgłasza wyjątek `PDOException`, gdy podczas wykonywania operacji PDO wystąpi błąd. Jeśli nie wystąpi żaden błąd, operacja zakończy się powodzeniem. W przeciwnym razie wyjątek spowoduje przerwanie wykonywania skryptu, o ile nie zostanie przechwycony. W tym celu kod, którego wykonanie może zakończyć się niepowodzeniem, należy umieścić w bloku `try`, natomiast kod odpowiedzialny za obsługę błędów w bloku `catch`:

```
try
{
    # Przeprowadzenie operacji na bazie danych.
}
```

```
catch (PDOException $e)
{
    # Obsługa błędu.
}
```

Obiekt błędu (\$e w omawianym przykładzie) ma własne metody dostarczające informacje o błędzie:

- Wartością zwrótną metody `getCode()` jest kod błędu.
- Wartością zwrótną metody `getMessage()` jest ciąg tekstowy zawierający komunikat błędu.

W przedstawionym poniżej przykładzie włączono obsługę wyjątków i pokazano, jak wyświetlić informacje błędu, gdy wykonanie zapytania zakończy się niepowodzeniem:

```
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
try
{
    $dbh->exec("DELETE FROM nieistniejąca_tabela");
}
catch (PDOException $e)
{
    # Wyświetlenie informacji o błędzie pochodzących z obiektu wyjątku.
    print("Wartość getCode: " . $e->getCode() . "\n");
    print("Wartość getMessage: " . $e->getMessage() . "\n");
    # Wyświetlenie informacji o błędzie pochodzących z uchwytu do bazy danych.
    print("Wartość errorCode: " . $dbh->errorCode() . "\n");
    print("Wartość errorInfo: " . join(", ", $dbh->errorInfo()) . "\n");
}
```

Powyższy przykład powoduje wyświetlenie informacji z obiektu wyjątku (\$e) oraz z obiektu uchwytu do bazy danych (\$dbh). To jest możliwe, ponieważ jedynym uchwycem PDO używanym w bloku try jest \$dbh. W przypadku użycia wielu uchwytów w bloku try, w bloku catch nie byłoby wiadomo, który z nich spowodował wystąpienie błędu. W takim przypadku trzeba by było polegać jedynie na metodach wyjątku. Alternatywne rozwiązanie polega na przeprowadzeniu restrukturyzacji kodu i odizolowaniu każdego uchwytu PDO we własnej konstrukcji try/catch.

Pewne funkcje PHP i operacje powodują wygenerowanie odpowiedniego komunikatu w przypadku wystąpienia błędu oraz zwrot wartości informującej o stanie. W kontekście aplikacji sieciowej wspomniany komunikat pojawia się na stronie wysyłanej do przeglądarki internetowej klienta, co nie jest oczekiwanym zachowaniem. W celu wyłączenia (prawdopodobnie tajemniczych dla użytkownika) komunikatów błędów standardowo generowanych przez funkcję jej nazwę należy poprzedzić operatorem @. Na przykład, aby zawiesić wyświetlanie komunikatów błędów przez funkcję `dowolna_funkcja()` i ewentualne błędy zgłaszać użytkownikowi w bardziej odpowiedni sposób, trzeba użyć poniższego polecenia:

```
$status = @dowolna_funkcja();
```

9.2. Praca z PHP

W pozostałej części rozdziału zajmiemy się realizacją naszkicowanych w rozdziale 1., zatytułowanym „Rozpoczęcie pracy z MySQL”, celów, które jeszcze pozostały do osiągnięcia:

- Dla projektu ocen uczniów konieczne jest utworzenie skryptu pozwalającego na wprowadzanie i edycję wyników uzyskanych przez uczniów w sprawdzianach i testach.
- Dla odwiedzających witrynę internetową Ligi Historycznej opracujemy internetowy quiz dotyczący prezydentów USA. Wspomniany quiz będzie interaktywny, wraz z generowanymi w locie odpowiedziami na zadawane pytania.
- Członkom Ligi Historycznej umożliwimy także możliwość edycji informacji o sobie. W ten sposób informacje o członkach Ligi będą aktualne, a sekretarka będzie miała mniejszą ilość pracy do wykonania.

Każdy ze wspomnianych skryptów powoduje wygenerowanie wielu stron internetowych. Pomiedzy kolejnymi wywołaniami skryptu informacje są przekazywane przez ich osadzanie w generowanych stronach. Jeżeli nie znasz koncepcji komunikacji między stronami, zapoznaj się z podpunktem 8.4.2.4, zatytułowanym „Tworzenie stron o wielu przeznaczeniach”.

9.2.1. Aplikacja pozwalająca na wprowadzenie wyników uzyskanych przez uczniów

W tym punkcie naszą uwagę skierujemy na projekt ocen uczniów i przygotujemy skrypt *score_entry.php*, przeznaczony do zarządzania wynikami uzyskanymi przez uczniów w trakcie sprawdzianów i testów. Katalog sieciowy dla tego projektu nosi nazwę *gp* i znajduje się w drzewie dokumentów serwera WWW Apache. Wspomnianemu katalogowi odpowiada poniższy adres URL naszej witryny:

`http://localhost/gp/`

Na obecnym etapie wymieniony katalog jest pusty i odwiedzający, którzy wprowadzą powyższy adres URL, otrzymają jedynie komunikat błędu informujący o niezalezieniu strony. Rozwiązaniem tego problemu jest utworzenie krótkiego skryptu o nazwie *index.php* i umieszczenie go w katalogu *gp*. Ten skrypt będzie działał w charakterze strony głównej projektu. Na chwilę obecną przedstawiony poniżej skrypt jest wystarczający i zawiera dwa łącza. Pierwsze prowadzi do skryptu *score_browse.pl*, utworzonego w punkcie 8.4.5, zatytułowanym „Przeglądarka tabel projektu ocen uczniów”, ponieważ należy on do projektu ocen uczniów. Drugie łącze prowadzi do skryptu *score_entry.php*, który tutaj utworzymy:

```
?php
# Strona główna projektu ocen uczniów.
```

```

require_once "sampdb_pdo.php";

$title = "Projekt ocen uczniów";
html_begin ($title, $title);
?>

<p>
<a href="/cgi-bin/score_browse.pl">Wyświetl</a> wyniki testów i sprawdzianów.
</p>
<p>
<a href="score_entry.php">Wprowadź lub edytuj</a> wyniki testów i sprawdzianów.
</p>

<?php
html_end ();
?>

```

Skrypt *index.php* znajduje się w katalogu *phpapi/gp* dystrybucji *sampdb*. Skopiuj go do katalogu *gp* w używanym serwerze WWW.

Zastanówmy się teraz nad projektem i implementacją skryptu *score_entry.php*, które pozwolą nam na wprowadzenie nowego lub edycję istniejącego zestawu wyników sprawdzianu lub testu. Możliwość wprowadzenia jest użyteczna, gdy do bazy danych trzeba wstawić nowy zestaw wyników. Z kolei możliwość edycji jest konieczna podczas późniejszej zmiany wyników, na przykład wstawienia wyniku uczniów, którzy pisali test lub sprawdzian później z powodu choroby, nieobecności lub innego zdarzenia. Kolejnym powodem zmiany wyników jest konieczność ich poprawienia na skutek ich błędnego wprowadzenia do bazy danych. Omawiany tutaj skrypt działa w następujący sposób:

- Strona początkowa wyświetla listę znanych zdarzeń (sprawdzianów i testów), pozwala na wybór jednego istniejącego lub utworzenie nowego zdarzenia.
- Po wybraniu opcji tworzenia nowego zdarzenia skrypt wyświetla stronę pozwalającą na podanie daty i kategorii (sprawdzian lub test) zdarzenia. Po dodaniu zdarzenia do bazy danych skrypt ponownie wyświetla listę zdarzeń. Na tym etapie nowo dodane zdarzenie znajduje się na liście.
- Po wybraniu istniejącego zdarzenia z listy skrypt wyświetla stronę zawierającą identyfikator zdarzenia, datę, kategorię oraz tabelę wraz z listą uczniów w klasie i przycisk *Wyślij*. Każdy wiersz tabeli HTML zawiera informacje o jednym uczniu: jego imię i nazwisko oraz uzyskany wynik. W przypadku nowych zdarzeń wszystkie komórki są puste. Z kolei w przypadku istniejących zdarzeń wyświetlane są wartości, które zostały wcześniej wprowadzone. Możesz wprowadzić nowe lub zmienić istniejące wyniki, a następnie kliknąć przycisk *Wyślij*. Na tym etapie skrypt wstawi wyniki do tabeli *score* lub uaktualni istniejące. Ta operacja musi być przeprowadzona jako transakcja, aby zagwarantować, że wszystkie modyfikacje zakończą się powodzeniem lub zostaną wycofane, jeśli wystąpi błąd.

9.2.1.1. Pobieranie danych wejściowych w PHP

Zanim przystąpimy do implementacji skryptu *score_entry.php*, w pierwszej kolejności musisz dowiedzieć się, jak działają parametry w danych w PHP. Skrypt przeprowadza wiele różnych akcji, co oznacza konieczność przekazywania wartości stanu między poszczególnymi stronami, aby w trakcie każdego wywołania skrypt wykonywał odpowiednie operacje. Jednym z rozwiązań jest przekazywanie parametrów za pomocą adresu URL. Na przykład, w adresie skryptu można umieścić parametr o nazwie *action*:

```
http://localhost/gp/score_entry.php?action=wartość
```

Wartości parametrów mogą pochodzić również z treści formularza wysłanego przez użytkownika. Każde pole wspomnianego formularza zwróconego przez przeglądarkę internetową użytkownika jako część operacji wysłania formularza ma nazwę oraz wartość.

Parametry danych wejściowych są w skryptach PHP dostępne za pomocą tablic specjalnych. Parametry zakodowane na końcu adresu URL są wysyłane jako część żądania typu GET i umieszczane w tablicy globalnej `$HTTP_GET_VARS` oraz superglobalnej `$_GET`. Z kolei parametry przekazywane w żądaniu typu POST (na przykład zawartość formularza, którego atrybut *method* ma wartość *post*) są umieszczane w tablicy globalnej `$HTTP_POST_VARS` oraz superglobalnej `$_POST`.

Tablice globalne muszą być wyraźnie zadeklarowane, jeśli mają być używane w kontekście innym niż najwyższy poziom skryptu PHP, czyli na przykład w definicjach funkcji. Natomiast tablice superglobalne są dostępne w każdym zakresie i nie wymagają specjalnej deklaracji. W celu zachowania prostoty wykorzystamy tablice superglobalne `$_GET` i `$_POST`. (Tablice `$HTTP_GET_VARS` i `$HTTP_POST_VARS` są uznawane za przestarzałe).

Wymienione `$_GET` i `$_POST` to tablice asocjacyjne, kluczami ich elementów są nazwy parametrów. Na przykład, parametr *action* wysyłany w adresie URL staje się dostępny w skrypcie jako wartość `$_GET["action"]`. Przyjmujemy założenie, że formularz zawiera pola o nazwach *name* i *address*. Kiedy użytkownik wyśle formularz, serwer WWW wywołuje skrypt w celu przetworzenia zawartości formularza. Jeśli formularz został wysłany w postaci żądania GET, wartości wprowadzone w formularzu będą znajdowały się w zmiennych `$_GET["name"]` i `$_GET["address"]`. Natomiast jeśli formularz został wysłany jako żądanie typu POST, zmiennymi są odpowiednio `$_POST["name"]` i `$_POST["address"]`.

W przypadku formularzy zawierających wiele pól nadanie im wszystkim unikalnych nazw może być niewygodne. PHP ułatwia przekazywanie tablic formularzy. Jeżeli użyjesz nazw pól w postaci *x[0]*, *x[1]* itd., PHP przechowuje je w postaci `$_GET["x"]` i `$_POST["x"]`, czyli wartości, która sama w sobie jest tablicą. Po przypisaniu tablicy do zmiennej `$x` elementy tablicy są dostępne jako `$x[0]`, `$x[1]` itd.

W większości przypadków nie ma znaczenia, za pomocą jakiej metody (GET lub POST) został wysłany parametr. Możemy więc utworzyć procedurę pomocniczą o nazwie `script_param()`, pobierającą nazwę parametru i sprawdzającą obie wymienione tablice w poszukiwaniu wartości dla tego parametru. Jeżeli parametr nie zostanie znaleziony, procedura zwraca wartość `NULL`:

```
function script_param ($name)
{
    $val = NULL;
    if (isset ($_GET[$name]))
        $val = $_GET[$name];
    else if (isset ($_POST[$name]))
        $val = $_POST[$name];
    if (get_magic_quotes_gpc ())
        $val = remove_backslashes ($val);
    return ($val);
}
```

Procedura `script_param()` pozwala skryptowi na łatwe uzyskanie dostępu do wartości parametrów danych wyjściowych za pomocą ich nazw bez konieczności koncentrowania się na tablicy, w której mogą być przechowywane. Ponadto, procedura przetwarza wartość parametru po jego wyodrębnieniu, przekazując wspomnianą wartość funkcji `remove_backslashes()`. Celem jest dostosowanie do konfiguracji z włączoną opcją `magic_quotes_gpc`, którą znajdziesz w poniższym wierszu pliku inicjalizacyjnego PHP:

```
magic_quotes_gpc = 0n;
```

Jeżeli wymieniona opcja jest włączona, PHP dodaje ukośniki do wartości parametrów, aby cytować znaki specjalne, takie jak cudzysłów lub ukośnik. Te dodatkowe ukośniki znacznie utrudniają sprawdzenie poprawności wartości parametrów, więc funkcja `remove_backslashes()` po prostu je usuwa. Została zaimplementowana za pomocą algorytmu rekurencyjnego, ponieważ w PHP istnieje możliwość tworzenia parametrów w postaci zagnieżdżonych tablic:

```
function remove_backslashes ($val)
{
    if (is_array ($val))
    {
        foreach ($val as $k => $v)
            $val[$k] = remove_backslashes ($v);
    }
    else if (!is_null ($val))
        $val = stripslashes ($val);
    return ($val);
}
```

Parametry danych wejściowych i opcja `register_globals`

Być może znasz opcję konfiguracyjną PHP o nazwie `register_globals`, która obecnie jest uznawana za przestarzałą i pozwalała na zarejestrowanie parametrów danych wejściowych bezpośrednio w skrypcie. Na przykład, pole formularza lub parametr URL o nazwie `x` były przechowywane bezpośrednio w zmiennej o nazwie `$x` w skrypcie. Niestety, włączenie tej opcji umożliwiało klientowi ustawienie w skrypcie zmiennych w sposób całkowicie niezamierzony przez programistę. To wiązało się z niebezpieczeństwem i dlatego twórcy PHP zalecali wyłączenie opcji `register_globals`. W procedurze `script_param()` celowo używamy jedynie tablic przeznaczonych do obsługi parametrów danych wejściowych. To znacznie bezpieczniejsze rozwiązanie, a na dodatek działa w przypadku wyłączonej opcji `register_globals`.

9.2.1.2. Wyświetlanie i wprowadzanie wyników

Po zapewnieniu sobie możliwości wygodnego wyodrębniania parametrów danych wejściowych możemy przystąpić do tworzenia skryptu *score_entry.php*. Wymieniony skrypt musi mieć możliwość przekazywania informacji między jego wywołaniami. W tym celu wykorzystamy parametr o nazwie *action*, który po wykonaniu skryptu może być pobrany w następujący sposób:

```
$action = script_param ("action");
```

Jeżeli wartość parametru nie została ustawiona, oznacza to pierwsze wywołanie skryptu. W przeciwnym razie na podstawie wartości zmiennej *\$action* określane jest działanie, które powinien podjąć skrypt. Ogólny szkielet skryptu *score_entry.php* przedstawia się następująco:

```
<?php
# score_entry.php - Skrypt pozwalający na wstawienie wyników w projekcie ocen uczniów.

require_once "sampdb_pdo.php";

# Zdefiniowanie stałych określających podejmowane działania.
define ("SHOW_INITIAL_PAGE", 0);
define ("SOLICIT_EVENT", 1);
define ("ADD_EVENT", 2);
define ("DISPLAY_SCORES", 3);
define ("ENTER_SCORES", 4);

# Miejsce na funkcje odpowiedzialne za obsługę danych wejściowych.

$title = "Projekt ocen uczniów -- wstawianie wyników";
html_begin ($title, $title);

$dbh = sampdb_connect ();

# Ustalenie akcji, która powinna zostać podjęta (domyślnie to
# wyświetlenie strony początkowej, jeśli nie zostanie podana żadna akcja).

$action = script_param ("action");
if (is_null ($action))
    $action = SHOW_INITIAL_PAGE;
switch ($action)
{
case SHOW_INITIAL_PAGE:    # Wyświetlenie strony początkowej.
    display_events ($dbh);
    break;
case SOLICIT_EVENT:        # Pobranie informacji o nowym zdarzeniu.
    solicit_event_info ();
    break;
case ADD_EVENT:            # Wstawienie nowego zdarzenia do bazy danych.
    add_new_event ($dbh);
    display_events ($dbh);
    break;
case DISPLAY_SCORES:       # Wyświetlenie wyników wskazanego zdarzenia.
    display_scores ($dbh);
    break;
```



```

case ENTER_SCORES:      # Wstawienie nowych lub edycja istniejących wyników.
    enter_scores ($dbh);
    display_events ($dbh);
    break;
default:
    die ("Nieznany kod ($action)\n");
}

$dbh = NULL; # Zamknięcie połączenia.

html_end ();
?>

```

Zmienna `$action` może przyjąć wiele wartości, które są sprawdzane w poleceniu `switch`. W PHP konstrukcja `switch` przypomina tę stosowaną w języku C. W omawianym skrypcie została użyta w celu ustalenia akcji do wykonania i wywołania funkcji implementujących tę akcję. Aby uniknąć konieczności użycia dosłownych wartości akcji, polecenie `switch` odwołuje się do symbolicznych nazw akcji zainicjalizowanych we wcześniejszej części skryptu za pomocą konstrukcji PHP o nazwie `define()`.

Przeanalizujmy funkcję obsługującą wspomniane akcje. Funkcja o nazwie `display_events()` wyświetla listę zdarzeń, pobierając rekordy tabeli `grade_event` z bazy danych MySQL i wyświetlając je. Każdy wiersz tabeli HTML zawiera identyfikator zdarzenia, datę i kategorię (test lub sprawdzian). Identyfikator zdarzenia jest wyświetlany na stronie jako łącze, którego kliknięcie pozwala na edycję wyników danego zdarzenia. Pod wierszami zdarzeń funkcja umieszcza kolejny wiersz, zawierający łącze pozwalające na utworzenie nowego zdarzenia:

```

function display_events ($dbh)
{
    print ("Wybierz zdarzenie klikając jego numer. Ewentualnie utwórz\n");
    print ("nowe klikając łącze Utwórz nowe zdarzenie:<br /><br />\n");
    print ("<table border='1'\>\n");

    # Wyświetlenie wiersza tabeli zawierającego nagłówki.
    print ("<tr>\n");
    display_cell ("th", "Identyfikator zdarzenia");
    display_cell ("th", "Data");
    display_cell ("th", "Kategoria");
    print ("</tr>\n");

    # Wyświetlenie listy zdarzeń. Powiązanie każdego identyfikatora zdarzenia
    # z łączem wyświetlającym wyniki w danym zdarzeniu.
    $stmt = "SELECT event_id, date, category
            FROM grade_event ORDER BY event_id";
    $sth = $dbh->query ($stmt);

    while ($row = $sth->fetch ())
    {
        print ("<tr>\n");
        $url = sprintf ("%s?action=%d&event_id=%d",
                        script_name (),
                        DISPLAY_SCORES,
                        $row["event_id"]);
    }
}

```

```

display_cell ("td",
              "<a href=\"\$url\">"
                . $row["event_id"]
                . "</a>",
              FALSE);
display_cell ("td", $row["date"]);
display_cell ("td", $row["category"]);
print ("</tr>\n");
}

```

Dodanie jeszcze jednego łącza, jego kliknięcie powoduje utworzenie nowego zdarzenia.

```

print ("<tr align=\"center\">\n");
$url = sprintf ("%s?action=%d",
                script_name (),
                SOLICIT_EVENT);
display_cell ("td colspan=\"3\"",
              "<a href=\"\$url\">Utwórz nowe zdarzenie</a>",
              FALSE);
print ("</tr>\n");
print ("</table>\n");
}

```

Adresy URL łączy ponownie wywołujących skrypt *score_entry.php* są tworzone za pomocą *script_name()*, czyli funkcji określającej ścieżkę dostępu do pliku skryptu. Wymieniona funkcja jest użyteczna, ponieważ pozwala uniknąć osadzania na stałe w kodzie ścieżki dostępu do skryptu. (Jeżeli zapiszesz ją dosłownie, skrypt przestanie działać po zmianie jego nazwy pliku). Funkcję *script_name()* znajdziesz w pliku *sampdb_pdo.php*.

Funkcja *script_name()* jest pod pewnymi względami podobna do *script_param()*, ponieważ uzyskuje dostęp do tablicy superglobalnej w PHP. Jednak używa zupełnie innej tablicy. Nazwa skryptu jest częścią informacji dostarczanych przez serwer WWW, a nie częścią parametrów danych wejściowych:

```

function script_name ()
{
    return ($_SERVER["SCRIPT_NAME"]);
}

```

Funkcja *display_cell()* używana przez *display_events()* generuje komórki w tabeli HTML wyświetlającej informacje o zdarzeniach:

```

# Wyświetlenie komórki tabeli HTML. Zmienna $tag to nazwa znacznika ("th" lub "td"
# dla komórki nagłówka lub danych), $value to wartość do wyświetlenia, natomiast zmienna
# $encode powinna mieć wartość true lub false, wskazując tym samym, czy wartość HTML
# ma zostać zakodowana przed jej wyświetleniem. Zmienna $encode jest opcjonalna,
# domyślnie ma przypisaną wartość true.
function display_cell ($tag, $value, $encode = TRUE)
{
    if (strlen ($value) == 0) # Czy wartość jest pusta lub nieustawiona?
        $value = "&nbsp;";
    else if ($encode) # Przeprowadzenie kodowania HTML, jeśli wartością zmiennej jest true.
        $value = htmlspecialchars ($value);
    print ("<$tag>$value</$tag>\n");
}

```

Po kliknięciu łącza *Utwórz nowe zdarzenie* w tabeli HTML wyświetlanej przez funkcję `display_events()` następuje ponowne wywołanie skryptu `score_entry.php` wraz z akcją `SOLICIT_EVENT`. Wymieniona akcja powoduje wywołanie funkcji `solicit_event_info()` wyświetlającej formularz, w którym użytkownik podaje datę i kategorię nowego zdarzenia:

```
function solicit_event_info ()
{
    printf("<form method=\"post\" action=\"%s?action=%d\">\n",
        script_name (),
        ADD_EVENT);
    print ("Podaj informacje o nowym zdarzeniu:<br /><br />\n");
    print ("Data: ");
    print("<input type=\"text\" name=\"date\" value=\"\" size=\"10\" />\n");
    print("<br />\n");
    print ("Kategoria: ");
    print("<input type=\"radio\" name=\"category\" value=\"T\"");
    print(" checked=\"checked\" />Test\n");
    print("<input type=\"radio\" name=\"category\" value=\"Q\" />Sprawdzian\n");
    print("<br /><br />\n");
    print("<input type=\"submit\" name=\"submit\" value=\"Wyślij\" />\n");
    print("</form>\n");
}
```

Formularz wygenerowany przez funkcję `solicit_event_info()` zawiera pole edycji pozwalające na wprowadzenie daty, parę przycisków opcji umożliwiających wskazanie rodzaju zdarzenia (test lub sprawdzian), a także przycisk *Wyślij*. Domyślną kategorią zdarzenia jest T (test). (Skrypt zawiera dosłowny kod HTML tworzący formularz. W przypadku dalszych skryptów opracujemy zestaw metod pomocniczych odpowiedzialnych za generowanie elementów formularza).

Po wypełnieniu i wysłaniu formularza nastąpi ponowne wywołanie skryptu `score_entry.php`, ale tym razem wraz z wartością akcji odpowiadającą `ADD_EVENT`. Następnie wywoływana jest funkcja `add_new_event()`, odpowiedzialna za wstawienie nowego rekordu do tabeli `grade_event`:

```
function add_new_event ($dbh)
{
    $date = script_param ("date");          # Pobranie wprowadzonych przez użytkownika
    $category = script_param ("category");  # daty i kategorii zdarzenia.

    if (empty ($date)) # Wprowadzona data powinna być w formacie ISO 8601.
        die ("Nie podano daty.\n");
    if (!preg_match ('/^\\d{4}\\D\\d{1,2}\\D\\d{1,2}$/', $date))
        die ("Proszę podać datę w formacie ISO 8601 (RRRR-MM-DD).\n");
    if ($category != "T" && $category != "Q")
        die ("Nieprawidłowa kategoria zdarzenia.\n");

    $stmt = "INSERT INTO grade_event (date,category) VALUES(?,?)";
    $sth = $dbh->prepare ($stmt);
    $sth->execute (array ($date, $category));
}
```

Funkcja `add_new_event()` używa procedury `script_param()` w celu uzyskania dostępu do wartości parametrów odpowiadających polom `date` i `category` w formularzu tworzenia nowego zdarzenia. Następnie przeprowadzane są minimalne operacje sprawdzania:

- Data musi być podana oraz być w formacie ISO 8601. Funkcja `preg_match()` sprawdza, czy data została podana w formacie ISO 8601:

```
preg_match ('/^d{4}\D\d{1,2}\D\d{1,2}$/', $date)
```

 W wywołaniu użyto apostrofów, aby uniemożliwić interpretację znaku dolara i ukośnika jako znaków specjalnych. Warunek przyjmuje wartość `true`, jeśli data składa się z trzech sekwencji cyfr rozdzielonych znakami innymi niż cyfry. To na pewno nie jest absolutnie niezawodne rozwiązanie, ale łatwe do implementacji w skrypcie oraz przechwytyjące większość pomyłek.
 Dodatkowym zabezpieczeniem może być zastosowanie ograniczeń dla danych wejściowych, na przykład przez włączenie trybu SQL przed wstawieniem danych:

```
$dbh->exec ("SET sql_mode = 'TRADITIONAL'");
```
- Kategorii zdarzenia musi być przypisana jedna z wartości dozwolonych do wstawienia w kolumnie `category` tabeli `grade_event`. Dozwolonymi wartościami są T i Q.

Jeżeli wartości parametrów wyglądają na prawidłowe, funkcja `add_new_event()` wstawi nowy rekord do tabeli `grade_event`. Kod odpowiedzialny za wykonanie zapytania używa miejsc zarezerwowanych, aby w ten sposób zagwarantować poprawne cytowanie danych umieszczanych w ciągu tekstowym zapytania. Po wykonaniu zapytania następuje zakończenie działania funkcji `add_new_event()` i powrót do części głównej skryptu, czyli konstrukcji `switch`. Oznacza to ponowne wyświetlenie listy zdarzeń, co pozwala użytkownikowi na wybór nowo dodanego i rozpoczęcie wstawiania wyników.

Po wybraniu zdarzenia z listy wyświetlonej przez funkcję `display_events()` skrypt `score_entry.php` wywołuje funkcję o nazwie `display_scores()`. Wszystkie łącza zdarzeń zawierają identyfikator zdarzenia zakodowany jako parametr `event_id`. Dlatego też funkcja `display_scores()` pobiera wartość parametru, upewnia się, że ma on postać liczby całkowitej, a następnie używa tego parametru w zapytaniu pobierającym rekordy wszystkich uczniów i ewentualnie istniejące wyniki dla danego zdarzenia:

```
function display_scores ($dbh)
{
    # Pobranie identyfikatora zdarzenia, który musi być liczbą całkowitą.
    $event_id = script_param ("event_id");
    if (!ctype_digit ($event_id))
        die ("Nieprawidłowy identyfikator zdarzenia.\n");

    # Pobranie wyników dla danego zdarzenia.
    $stmt = "
        SELECT
            student.student_id, student.name, grade_event.date,
            score.score AS score, grade_event.category
        FROM student
        INNER JOIN grade_event
```

```

        LEFT JOIN score ON student.student_id = score.student_id
                        AND grade_event.event_id = score.event_id
    WHERE grade_event.event_id = ?
    ORDER BY student.name";
    $sth = $dbh->prepare ($stmt);
    $sth->execute (array ($event_id));

    # Pobranie rekordów i umieszczenie w tablicy, aby można było je policzyć.
    $rows = $sth->fetchAll ();
    if (count ($rows) == 0)
        die ("Nie znaleziono danych dla wskazanego zdarzenia.\n");

    printf ("<form method='post' action='%s?action=%d&event_id=%d'>\n",
            script_name (),
            ENTER_SCORES,
            $event_id);

    # Wyświetlenie wyników w postaci tabeli HTML.
    for ($row_num = 0; $row_num < count ($rows); $row_num++)
    {
        $row = $rows[$row_num];
        # Wyświetlenie informacji o zdarzeniu i nagłówek tabeli przed pierwszym rekordem.
        if ($row_num == 0)
        {
            printf ("Identyfikator zdarzenia: %d, Data zdarzenia: %s, Kategoria zdarzenia: %s\n",
                    $event_id,
                    $row["date"],
                    $row["category"]);
            print ("<br /><br />\n");
            print ("<table border='1'>\n");
            print ("<tr>\n");
            display_cell ("th", "Imię i nazwisko");
            display_cell ("th", "Wynik");
            print "</tr>\n");
        }
        print ("<tr>\n");
        display_cell ("td", $row["name"]);
        $col_val = sprintf ("<input type='text' name='score[%d]'\n",
                            $row["student_id"]);
        $col_val .= sprintf (" value='%d' size='5' /><br />\n",
                            $row["score"]);
        display_cell ("td", $col_val, FALSE);
        print ("</tr>\n");
    }

    print ("</table>\n");
    print ("<br />\n");
    print ("<input type='submit' name='submit' value='Wyślij' />\n");
    print "</form>\n";
}

```

Zapytanie używane przez funkcję `display_scores()` do pobrania informacji dla wybranego zdarzenia to nie jest proste złączenie między tabelami, ponieważ nie wybierze rekordów uczniów, którzy nie mają wyniku w danym zdarzeniu. W szczególności dotyczy to nowych zdarzeń; wtedy złączenie nie wybierze żadnych rekordów i formularz będzie pusty! Zamiast tego użyjemy złączenia typu `LEFT JOIN` w celu wymuszenia pobrania rekordów dla każdego

ucznia, niezależnie od tego, czy dla niego znajduje się wynik w tabeli score. Jeżeli uczeń nie ma wyniku z danego zdarzenia, wartością pobraną przez zapytanie będzie NULL. (Zapytanie podobne do zastosowanego w funkcji `display_scores()` w celu pobrania rekordów wyników z MySQL stosowaliśmy w punkcie 2.8.3, zatytułowanym „Złączenia typu LEFT i RIGHT (OUTER)”).

Wyniki pobrane przez zapytanie skrypt umieszcza w formularzu, ponieważ nazwy jego pól są w postaci `score[n]`, gdzie *n* oznacza wartość `student_id`. Teraz masz możliwość wprowadzenia lub edycji wyników, a następnie wysłania formularza i tym samym wstawienia wyników do bazy danych. Kiedy przeglądarka internetowa wyśle formularz z powrotem do serwera WWW, PHP skonwertuje wspomniane pola formularza na elementy tablicy powiązanej z nazwą `name`. Tę tablicę będzie można później pobrać za pomocą polecenia:

```
$score = script_param ("score");
```

Kluczami elementów tablicy są identyfikatory uczniów, co pozwala na łatwe powiązanie każdego ucznia z odpowiadającym mu wynikiem wysłanym w formularzu. Przetwarzanie formularza może obejmować wykonanie wielu poleceń (po jednym dla każdego ucznia), a nie chcemy, aby operacja uaktualnienia zakończyła się tylko częściowym sukcesem. W rozdziale 1., zatytułowanym „Rozpoczęcie pracy z MySQL”, tabelę `score` utworzyliśmy jako InnoDB. Dzięki temu możemy wykorzystać możliwości transakcyjne oferowane przez silnik InnoDB. Stosując transakcję, upewniamy się, że cała operacja zostanie przeprowadzona niepodzielnie. W ten sposób wprowadzone zostaną wszystkie zmiany lub żadna. Przetwarzanie transakcyjne w PDO ma przedstawioną poniżej ogólną strukturę (przyjmujemy założenie o włączeniu zgłaszania wyjątków dla błędów PDO):

```
try
{
    $dbh->beginTransaction ();           # Początek transakcji.
    # Przeprowadzenie operacji na bazie danych.
    $dbh->commit ();                     # Transakcja zakończyła się sukcesem.
}
catch (PDOException $e)
{
    $dbh->rollback ();                   # Transakcja zakończyła się niepowodzeniem.
}
```

Skrypt `score_entry.php` korzysta z powyższej struktury, aby zapewnić niepodzielność operacji wstawiania danych. Ponadto, operacja wycofania zmian została umieszczona we własnej konstrukcji try/catch w celu uniemożliwienia przerwania wykonywania skryptu, jeśli wycofanie transakcji zakończy się niepowodzeniem.

Funkcja `enter_scores()` przetwarza zawartość formularza w celu określenia, które wyniki wymagają uaktualnienia lub usunięcia:

```
function enter_scores ($dbh)
{
    # Pobranie identyfikatora zdarzenia i tablicy wyników dla danego zdarzenia.

    $event_id = script_param ("event_id");
    $score = script_param ("score");
```

```

if (!ctype_digit ($event_id)) # Identyfikator zdarzenia musi być liczbą całkowitą.
    die ("Nieprawidłowy identyfikator zdarzenia.\n");

# Przygotowanie zapytań, które później będą wielokrotnie wykonywane.
$sth_del = $dbh->prepare ("DELETE FROM score
    WHERE event_id = ? AND student_id = ?");
$sth_rep1 = $dbh->prepare ("REPLACE INTO score
    (event_id,student_id,score)
    VALUES(?,?,?)");

# Wyniki są wprowadzane w ramach transakcji.
try
{
    $dbh->beginTransaction ();

    $blank_count = 0;
    $nonblank_count = 0;
    foreach ($score as $student_id => $new_score)
    {
        $new_score = trim ($new_score);
        if (empty ($new_score))
        {
            # Jeżeli w formularzu nie znajduje się wynik dla danego ucznia, należy usunąć
            # ewentualny wynik tego ucznia, który mógł się dotychczas znajdować w bazie danych.
            ++$blank_count;
            $sth = $sth_del;
            $params = array ($event_id, $student_id);
        }
        else if (ctype_digit ($new_score)) # Wynik musi być liczbą całkowitą.
        {
            # Jeżeli wynik został podany, należy nim zastąpić wynik,
            # który mógł dotychczas znajdować się w bazie danych.
            ++$nonblank_count;
            $sth = $sth_rep1;
            $params = array ($event_id, $student_id, $new_score);
        }
        else
        {
            throw new PDOException ("Nieprawidłowy wynik: $new_score");
        }
        $sth->execute ($params);
    }
    # Transakcja zakończyła się powodzeniem, trzeba ją zatwierdzić.
    $dbh->commit ();
    printf ("Liczba wprowadzonych wyników: %d<br />\n", $nonblank_count);
    printf ("Liczba brakujących wyników: %d<br />\n", $blank_count);
}
catch (PDOException $e)
{
    printf ("Nie udało się wstawić wyniku: %s<br />\n",
        htmlspecialchars ($e->getMessage ()));
    # Wycofanie transakcji, używany pusty procedury obsługi wyjątków, aby przechwycić niepowodzenie
    # w trakcie wycofywania.
    try
    {
        $dbh->rollback ();
    }
}

```

```

        catch (PDOException $e) { }
    }
    print ("<br />\n");
}

```

Powyższy skrypt pobiera i przetwarza identyfikatory uczniów oraz powiązane z nimi wyniki za pomocą iteracji przez tablicę \$score:

- Jeżeli po usunięciu wszelkich znaków odstępu z obu stron wartości wynik jest pusty, oznacza to, że użytkownik nie wprowadził wyniku. Gdyby jednak wcześniej istniał wynik, skrypt próbuje go usunąć. (Prawdopodobnie wcześniej przez pomyłkę wprowadzono wynik dla ucznia, który był nieobecny, więc teraz należy usunąć ten wynik). Jeżeli uczeń nie miał wyniku, zapytanie DELETE nie znajdzie rekordu, ale jednocześnie nie zaszkodzi.
- Jeżeli wynik został podany, funkcja przeprowadza podstawowe operacje weryfikacji wartości i akceptuje ją, gdy ma postać liczby całkowitej. Zwróć uwagę na przeprowadzenie sprawdzenia za pomocą dopasowania wzorca zamiast użycia funkcji PHP o nazwie `is_int()`. Wymieniona funkcja jest przeznaczona do sprawdzenia, czy zmienna jest typu liczby całkowitej, natomiast wartości formularza są kodowane na postać ciągów tekstowych. Wartością zwrótną funkcji `is_int()` będzie FALSE dla każdego ciągu tekstowego, nawet jeśli składa się on wyłącznie z cyfr. W tym miejscu musimy więc sprawdzić zawartość ciągu tekstowego. Przedstawiona poniżej funkcja zwraca wartość TRUE, jeśli każdy znak od początku do końca ciągu tekstowego \$str jest cyfrą:

```
ctype_digit ($str)
```

Jeżeli wynik wygląda na liczbę całkowitą, zostaje wstawiony do tabeli score. Wykorzystaliśmy zapytanie REPLACE zamiast INSERT, ponieważ może wystąpić potrzeba zastąpienia istniejącego wyniku zamiast wstawienia nowego. Jeżeli dla danego zdarzenia uczeń nie ma wyniku, zapytanie REPLACE spowoduje wstawienie nowego rekordu, podobnie jak INSERT. W przeciwnym razie zapytanie REPLACE zastępuje istniejący wynik nowym.

Przed wejściem do pętli skrypt wywołuje metodę `beginTransaction()`, która wyłącza tryb automatycznego zatwierdzania w bazie danych. Po zakończeniu pętli skrypt zatwierdza transakcję, o ile nie wystąpiły żadne błędy. Jeżeli wydarzyło się cokolwiek nieprzewidzianego, skrypt wycofuje transakcję.

W ten sposób działa skrypt `score_entry.php`. Całą operację wprowadzenia i edycji wyniku możesz teraz przeprowadzić za pomocą przeglądarki internetowej. Jedną z oczywistych wad skryptu jest brak zabezpieczeń; każdy, kto ma dostęp do serwera WWW, będzie mógł przeprowadzić edycję wyników. Skrypt przeznaczony do edycji informacji o członkach Ligi Historycznej utworzony w dalszej części rozdziału pokaże prosty schemat uwierzytelniania, który można zaadaptować także dla omówionego tutaj skryptu `score_entry.php`.

9.2.2. Tworzenie interaktywnego quizu w internecie

Jednym z celów do zrealizowania na witrynie internetowej Ligi Historycznej jest zaoferowanie internetowej wersji quizu, podobnego do publikowanych w sekcji dziecięcej gazetki Ligi, zatytułowanej *Chronicles of U.S. Past*. Skoro utworzyliśmy tabelę `president`, to możemy ją wykorzystać jako źródło pytań dla quizu historycznego. Do tego celu utworzymy teraz skrypt o nazwie `pres_quiz.php`.

Podstawowy mechanizm działania skryptu polega na wybraniu pytania dotyczącego dowolnego prezydenta, pobraniu odpowiedzi od użytkownika i sprawdzeniu, czy jest ona prawidłowa. Rodzaje pytań wyświetlanych przez skrypt mogą być oparte na dowolnej części tabeli `president`. Jednak w celu zachowania prostoty ograniczymy się tutaj jedynie do pytań o miejsce urodzenia prezydenta. Inne uproszczenie polega na wyświetleniu pytania wraz z kilkoma odpowiedziami do wyboru. Użytkownikowi znacznie łatwiej wybrać odpowiedź z wyświetlonych, niż wpisywać ją całkowicie samodzielnie. Ponadto, skrypt będzie mógł łatwiej przeprowadzić porównanie odpowiedzi wybranej przez użytkownika z prawidłową niż w przypadku stosowania dopasowania wzorca sprawdzającego, co użytkownik napisał w odpowiedzi.

Skrypt `pres_quiz.php` musi spełniać dwie funkcje:

- Po początkowym wywołaniu powinien wygenerować i wyświetlić nowe pytanie na podstawie informacji pobranych z tabeli `president`.
- Po udzieleniu odpowiedzi przez użytkownika skrypt powinien ją sprawdzić i poinformować użytkownika, czy odpowiedź była poprawna. W przypadku niepoprawnej odpowiedzi skrypt powinien ponownie wyświetlić to samo pytanie. W przeciwnym razie skrypt ma wygenerować i wyświetlić nowe pytanie.

Ogólny szkielet skryptu jest całkiem prosty. Jeśli formularz nie został jeszcze wysłany, skrypt wyświetla stronę początkową wraz z pytaniem. W przeciwnym razie sprawdza udzieloną odpowiedź:

```
<?php
# pres_quiz.php - Skrypt będący quizem na temat miejsc urodzenia prezydentów USA.

require_once "smpdb_pdo.php";

# Miejsce na funkcje obsługujące quiz.

$title = "Quiz na temat miejsc urodzenia prezydentów USA";
html_begin ($title, $title);

$dbh = smpdb_connect ();

$response = script_param ("response");
if (is_null ($response)) # Skrypt został wywołany po raz pierwszy.
    present_question ($dbh);
else # Użytkownik wysłał formularz z odpowiedzią na pytanie.
    check_response ($dbh);
```

```
$dbh = NULL;    # Zamknięcie połączenia.
```

```
html_end ();
?>
```

W celu wygenerowania pytań używamy `ORDER BY RAND()` w połączeniu z klauzulą `LIMIT 1`, co powoduje losowe wybranie jednego rekordu z tabeli `president`. Na przykład, przedstawione poniżej zapytanie powoduje pobranie imienia i nazwiska oraz miejsca urodzenia losowo wybranego prezydenta:

```
SELECT CONCAT(first_name, ' ', last_name) AS name,
CONCAT(city, ', ', state) AS place
FROM president ORDER BY RAND() LIMIT 1;
```

Imię i nazwisko prezydenta pojawiającego się w pytaniu oraz miejsce jego urodzenia to prawidłowa odpowiedź na pytanie „Gdzie urodził się ten prezydent?”. Konieczne jest wyświetlenie także niepoprawnych odpowiedzi i do tego celu używamy zapytania podobnego do poprzedniego:

```
SELECT DISTINCT CONCAT(city, ', ', state) AS place
FROM president ORDER BY RAND();
```

Z wyników zapytania pobieramy cztery pierwsze wartości inne niż odpowiedź prawidłowa. Zapytanie zawiera klauzulę `LIMIT 5` zamiast `LIMIT 4` na wypadek, gdyby w wynikach zapytania znajdowała się również odpowiedź prawidłowa. Klauzula `DISTINCT` gwarantuje, że dane miejsce urodzenia znajdzie się tylko jednokrotnie na liście. Wymieniona klauzula byłaby niepotrzebna, gdyby wszystkie miejsca urodzenia były unikalne. Jednak nie są, o czym możesz się przekonać, wykonując poniższe zapytanie:

```
mysql> SELECT city, state, COUNT(*) AS count FROM president
-> GROUP BY city, state HAVING count > 1;
```

```
+-----+-----+-----+
| city   | state | count |
+-----+-----+-----+
| Braintree | MA   | 2     |
+-----+-----+-----+
```

Funkcja generująca pytanie oraz zestaw odpowiedzi przedstawia się następująco:

```
function present_question ($dbh)
{
    # Wykonanie zapytania w celu pobrania prezydenta i miejsca jego urodzenia.
    $stmt = "SELECT CONCAT(first_name, ' ', last_name) AS name,
                CONCAT(city, ', ', state) AS place
            FROM president ORDER BY RAND() LIMIT 1";
    $sth = $dbh->query ($stmt);
    $row = $sth->fetch ();
    $name = $row["name"];
    $place = $row["place"];

    # Przygotowanie zbioru miejsc, które będą wyświetlone jako potencjalne odpowiedzi.
    # Tworzona jest tablica $choices zawierająca pięć miejsc, z których jedno
    # jest prawidłową odpowiedzią na wyświetlone pytanie.
    $stmt = "SELECT DISTINCT CONCAT(city, ', ', state) AS place
            FROM president ORDER BY RAND() LIMIT 5";
```

```

$sth = $dbh->query ($stmt);
$choices[] = $place; # Inicjalizacja tablicy wraz z poprawną odpowiedzią.
while (count ($choices) < 5 && $row = $sth->fetch ())
{
    if ($row["place"] != $place)
        $choices[] = $row["place"]; # Dodanie innych, niepoprawnych odpowiedzi.
}
# Wymieszanie odpowiedzi i wyświetlenie formularza.
shuffle ($choices);
display_form ($name, $place, $choices);
}

```

Funkcja `display_form()` wywoływana przez `president_question()` generuje pytanie quizu wyświetlane w formularzu zawierającym imię i nazwisko prezydenta, listę przycisków opcji wraz z odpowiedziami oraz przycisk *Wyslij*. Formularz służy do oczywistego celu, jakim jest wyświetlenie pytania użytkownikowi, ale nie tylko. Formularz musi przedstawić informacje quizu także klientowi, aby po wysłaniu formularza z odpowiedzią udzieloną przez użytkownika dane przekazane z powrotem do serwera WWW pozwoliły skryptowi na sprawdzenie, czy odpowiedź jest poprawna. W przypadku niepoprawnej odpowiedzi pytanie powinno zostać ponownie wyświetlone.

Przedstawienie pytania quizu jest całkiem proste: wyświetlane jest imię i nazwisko prezydenta oraz lista możliwych miejsc urodzenia. Sprawdzenie udzielonej odpowiedzi i ponowne wyświetlenie pytania to już nieco trudniejsze zadanie. Wymaga uzyskania dostępu do poprawnej odpowiedzi oraz wszystkich informacji koniecznych w celu ponownego wygenerowania pytania. Jednym z możliwych rozwiązań jest wykorzystanie zestawu ukrytych pól formularza zawierających wszystkie niezbędne informacje. Wspomniane pola staną się częścią formularza, ale nie będą wyświetlane użytkownikowi i jedynie zwracane do serwera WWW wraz z formularzem wysłanym przez użytkownika.

Ukryte pola formularza noszą nazwę `name`, `place` i `choices`; przedstawiają odpowiednio imię i nazwisko prezydenta, poprawną odpowiedź i zestaw możliwych odpowiedzi. Za pomocą funkcji `implode()` zestaw odpowiedzi można zakodować na postać pojedynczego ciągu tekstowego, łącząc poszczególne wartości specjalnym znakiem ogranicznika. (Ogranicznik umożliwia później prawidłowy podział ciągu tekstowego za pomocą funkcji `explode()`, gdy konieczne będzie ponowne wyświetlenie pytania). Funkcja `display_form()` jest odpowiedzialna za wyświetlenie formularza:

```

function display_form ($name, $place, $choices)
{
    printf("<form method=\"post\" action=\"%s\">\n", script_name ());
    hidden_field ("name", $name);
    hidden_field ("place", $place);
    hidden_field ("choices", implode ("#", $choices));
    printf("Gdzie urodził się prezydent %s?<br /><br />\n", htmlspecialchars ($name));
    for ($i = 0; $i < 5; $i++)
    {
        radio_button ("response", $choices[$i], $choices[$i], FALSE);
        print("<br />\n");
    }
    print("<br />\n");
}

```

```

        submit_button ("submit", "Wyślij");
        print ("</form>\n");
    }

```

Do wygenerowania pól formularza funkcja `display_form()` używa kilku funkcji pomocniczych. Pierwsza z nich nosi nazwę `hidden_field()` i generuje znacznik `<input>` dla ukrytego pola formularza:

```

function hidden_field ($name, $value)
{
    printf ("<input type=\"%s\" name=\"%s\" value=\"%s\" />\n",
           "hidden",
           htmlspecialchars ($name),
           htmlspecialchars ($value));
}

```

Ponieważ `hidden_field()` to procedura ogólnego przeznaczenia, może być użyteczna w wielu skryptach. Dlatego też logiczne jest umieszczenie jej w naszym pliku biblioteki *sampdb_pdo.php*. Zwróć uwagę na użycie funkcji `htmlspecialchars()` do zakodowania atrybutów znacznika `<input>`, gdyby zawierały jakiegokolwiek znaki specjalne, na przykład cudzysłów.

Poniżej przedstawiono implementacje dwóch pozostałych funkcji pomocniczych, o nazwach `radio_button()` i `submit_button()`:

```

function radio_button ($name, $value, $label, $checked)
{
    printf ("<input type=\"%s\" name=\"%s\" value=\"%s\" %s />\n",
           "radio",
           htmlspecialchars ($name),
           htmlspecialchars ($value),
           ($checked ? " checked=\"checked\" " : "" ),
           htmlspecialchars ($label));
}

function submit_button ($name, $value)
{
    printf ("<input type=\"%s\" name=\"%s\" value=\"%s\" />\n",
           "submit",
           htmlspecialchars ($name),
           htmlspecialchars ($value));
}

```

Kiedy użytkownik wybierze odpowiedź spośród dostępnych i wyśle formularz, jego odpowiedź zostaje przekazana serwerowi WWW jako wartość parametru `response`. Wartość wymienionego parametru można pobrać za pomocą wywołania funkcji `script_param()`. W ten sposób określamy także, czy skrypt został wywołany po raz pierwszy, czy też użytkownik udzielił odpowiedzi we wcześniej wyświetlonym formularzu. Jeżeli parametru `response` nie istnieje, oznacza to pierwsze wywołanie skryptu. Tak więc na podstawie istnienia lub braku parametru `response` część główna skryptu ustala, jakie powinno zostać podjęte działanie:

```

$response = script_param ("response");
if (is_null ($response)) # Skrypt został wywołany po raz pierwszy.
    present_question ($dbh);
else # Użytkownik wysłał formularz z odpowiedzią.
    check_response ($dbh);

```

Do utworzenia pozostała funkcja `check_response()`, porównująca odpowiedź udzieloną przez użytkownika z poprawną. W tym celu konieczne są wartości ukrytych pól formularza `name`, `place` i `choices`. Poprawna odpowiedź znajduje się w polu `place` formularza, natomiast udzielona przez użytkownika w polu `response`. Aby sprawdzić odpowiedź, musimy po prostu porównać wartości obu wymienionych pól. Opierając się na wyniku porównania, funkcja `check_response()` wyświetla pewne informacje, a następnie generuje i wyświetla nowe pytanie lub ponownie wyświetla to samo:

```

function check_response ($dbh)
{
    $name = script_param ("name");
    $place = script_param ("place");
    $choices = script_param ("choices");
    $response = script_param ("response");

    # Czy użytkownik podał poprawne miejsce urodzenia prezydenta?
    if ($response == $place)
    {
        print ("Odpowiedź jest poprawna!<br />\n");
        printf ("%s urodził się w %s.<br />\n",
            htmlspecialchars ($name),
            htmlspecialchars ($place));
        print ("Spróbuj odpowiedzieć na kolejne pytanie:<br /><br />\n");
        present_question($dbh);
    }
    else
    {
        printf ("\\"%s\" to niepoprawna odpowiedź. Spróbuj ponownie.<br /><br />\n",
            htmlspecialchars ($response));
        $choices = explode ("#", $choices);
        display_form ($name, $place, $choices);
    }
}

```

I to już koniec. Na stronie głównej Ligi Historycznej umieść łącze prowadzące do skryptu `pres_quiz.php`, a odwiedzający witrynę będą mogli sprawdzić swoją wiedzę. (Możesz skopiować plik `index5.php` z katalogu `phpapi/ushl` dystrybucji `sampdb` do katalogu `ushl` drzewa dokumentów serwera WWW i zmienić jego nazwę na `index.php`, zastępując istniejący plik o tej nazwie nowym).

Ukryte pola formularza są niebezpieczne

Skrypt *pres_quiz.php* wykorzystuje ukryte pola formularza do przekazywania potrzebnych w trakcie kolejnego wywołania skryptu informacji, których użytkownik nie powinien widzieć. Takie rozwiązanie można zastosować w omówionym skrypcie, przeznaczonym jedynie do zapewnienia rozrywki. Jednak ukrytych pól formularza *nie* należy używać do przekazywania informacji, których użytkownik nigdy nie powinien zobaczyć, ponieważ nie są one zabezpieczone w żaden sposób. Aby się o tym przekonać, umieść skrypt *pres_quiz.php* w katalogu *ushl* serwera WWW i uruchom go z poziomu przeglądarki internetowej. Następnie za pomocą opcji menu *Pokaż źródło* spójrz na kod źródłowy HTML strony quizu. W ukrytym polu formularza o nazwie *place* zobaczysz prawidłową odpowiedź na wyświetlone pytanie. Oczywiście jest, że każdy może sprawdzić tę odpowiedź. W ten sposób bardzo łatwo można oszukiwać w trakcie quizu. To nie ma znaczenia w przypadku aplikacji takiej jak opracowana w tym punkcie, ale przykład pokazuje, jak niebezpieczne jest używanie ukrytych pól formularza. W przypadku informacji, które naprawdę nie powinny być ujawnione użytkownikowi, należy skorzystać z innych metod, takich jak mechanizm sesji, ponieważ wtedy informacje są przechowywane po stronie serwera.

9.2.3. Edycja informacji o członkach Ligi Historycznej

Nasz ostatni skrypt, *edit_member.php*, ma umożliwić członkom Ligi Historycznej edycję informacji o sobie wyświetlanych w katalogu internetowym. Za pomocą wymienionego skryptu członkowie Ligi będą mogli poprawiać lub uaktualniać informacje o sobie bez konieczności kontaktowania się w tym celu z sekretariatem Ligi. Dzięki wspomnianej możliwości katalog członków Ligi powinien być w miarę aktualny, a sekretarce odpadnie praca związana z aktualizacją informacji o członkach Ligi.

Konieczne będzie wprowadzenie pewnych zabezpieczeń, aby zagwarantować, że rekord członka Ligi będzie mógł być modyfikowany tylko przez danego członka lub sekretarkę Ligi. Oznacza to potrzebę zastosowania pewnej formy bezpieczeństwa. Prezentując prosty mechanizm uwierzytelniania, wykorzystamy MySQL do przechowywania haseł wszystkich członków Ligi. Podanie prawidłowego hasła będzie konieczne w celu uzyskania możliwości edycji oferowanych przez tworzony tutaj skrypt. Ten skrypt działa w następujący sposób:

- Po początkowym wywołaniu skrypt *edit_script.php* wyświetla formularz logowania wyświetlający pola dla identyfikatora członka Ligi i jego hasła.
- Po wysłaniu formularza logowania skrypt sprawdza tabelę haseł, a następnie powiązuje identyfikator członka Ligi z hasłem. Jeżeli hasło będzie dopasowane, skrypt pobiera z tabeli *member* informacje o członku Ligi i wyświetla te dane do edycji.
- Po wysłaniu formularza z uaktualnionymi danymi skrypt uaktualnia informacje w bazie danych.

Do wykonania wszystkich zadań konieczne jest użycie hasła. Łatwym sposobem jest automatyczne wygenerowanie haseł dla członków Ligi. Przedstawione poniżej zapytania tworzą tabelę o nazwie *member_pass*, a następnie hasło dla każdego członka Ligi. Odbywa się to

przez wygenerowanie sumy kontrolnej MD5 dla losowo wybranej liczby i użycie pierwszych ośmiu znaków wyniku. W rzeczywistych sytuacjach należy pozwolić użytkownikom na samodzielne definiowanie haseł. Przedstawiona technika pozwala na przeprowadzenie szybkiej i łatwej konfiguracji początkowej:

```
CREATE TABLE member_pass
(
    member_id INT UNSIGNED NOT NULL PRIMARY KEY,
    password CHAR(8)
);
INSERT INTO member_pass (member_id, password)
SELECT member_id, LEFT(MD5(RAND()), 8) AS password FROM member;
```

Oprócz haseł dla wszystkich członków Ligi wymienionych w tabeli member do tabeli member_pass dodajemy rekord specjalny wraz z hasłem bigshot, które będzie służyło w charakterze hasła administracyjnego (superużytkownik). Sekretarka Ligi będzie mogła użyć tego hasła w celu uzyskania dostępu do dowolnego rekordu:

```
INSERT INTO member_pass (member_id, password) VALUES(0, 'bigshot');
```

Uwaga

Przed utworzeniem tabeli member_pass z serwera WWW powinieneś usunąć skrypt `db_browse.pl`. Wymieniony skrypt utworzyliśmy w punkcie 8.4.4, zatytułowanym „Przeglądarka bazy danych”; pozwala on każdemu na przeglądanie zawartości dowolnej tabeli w bazie danych sampdb, w tym także tabeli member_pass. Dlatego też za pomocą wymienionego skryptu można poznać hasło każdego członka Ligi Historycznej oraz sekretarki Ligi.

Po przygotowaniu tabeli member_pass można już przystąpić do tworzenia skryptu `edit_member.php`. Ogólny szkielet skryptu przedstawia się następująco:

```
<?php
# edit_member.php - Edycja przez przeglądarkę internetową informacji o członkach Ligi Historycznej.

require_once "sampdb_pdo.php";

# Zdefiniowanie stałych akcji.
define ("SHOW_INITIAL_PAGE", 0);
define ("DISPLAY_ENTRY", 1);
define ("UPDATE_ENTRY", 2);

# Miejsce na funkcje odpowiedzialne za obsługę danych wejściowych.

$title = "Liga Historyczna -- formularz edycji informacji o członku Ligi";
html_begin ($title, $title);

$dbh = sampdb_connect ();

# Ustalenie akcji, która powinna zostać podjęta (domyślnie to
# wyświetlenie strony początkowej, jeśli nie zostanie podana żadna akcja).

$action = script_param ("action");
if (is_null ($action))
```

```

$action = SHOW_INITIAL_PAGE;

switch ($action)
{
case SHOW_INITIAL_PAGE:   # Wyświetlenie strony początkowej
    display_login_page ();
    break;
case DISPLAY_ENTRY:       # Wyświetlenie formularza edycji.
    display_entry ($dbh);
    break;
case UPDATE_ENTRY:       # Uaktualnienie informacji w bazie danych.
    update_entry ($dbh);
    break;
default:
    die ("Nieznany kod akcji ($action)\n");
}

$dbh = NULL; # Zamknięcie połączenia.

html_end ();
?>

```

Funkcja `display_login_page()` wyświetla stronę początkową zawierającą formularz pozwalający na podanie identyfikatora członka Ligi i jego hasła:

```

function display_login_page ()
{
    printf ("<form method=\"post\" action=\"%s?action=%d\">\n",
           script_name (),
           DISPLAY_ENTRY);
    print ("Podaj identyfikator członka Ligi i hasło,\n");
    print ("a następnie kliknij przycisk Wyślij.\n<br /><br />\n");
    print ("<table>\n");
    print ("<tr>");
    print ("<td>Identyfikator członka Ligi</td><td>");
    text_field ("member_id", "", 10);
    print ("</td></tr>");
    print ("<tr>");
    print ("<td>Hasło</td><td>");
    password_field ("password", "", 10);
    print ("</td></tr>");
    print ("</table>\n");
    submit_button ("button", "Wyślij");
    print "</form>\n";
}

```

Formularz w tabeli HTML wyświetla nagłówki i pola wartości, a więc są one ładnie sformatowane. W przypadku jedynie dwóch pól to nie ma znaczenia, ale użycie tabeli jest użyteczną techniką, zwłaszcza podczas tworzenia formularzy o nagłówkach różnej długości, ponieważ eliminuje pionowe nierówności. Wyrównanie elementów formularza powoduje, że staje się on dla użytkownika łatwiejszy do odczytu i zrozumienia.

Funkcja `display_login_form()` używa dwóch funkcji pomocniczych umieszczonych w pliku biblioteki *sampdb_pdo.php*. Zadaniem funkcji `text_field()` jest wyświetlenie możliwego do edycji pola danych wejściowych:


```
function text_field ($name, $value, $size)
{
    printf("<input type=\"%s\" name=\"%s\" value=\"%s\" size=\"%s\" />\n",
        "text",
        htmlspecialchars ($name),
        htmlspecialchars ($value),
        htmlspecialchars ($size));
}
```

Funkcja `password_field()` działa podobnie, ale wartością atrybutu `type` pola jest `password`.

Po wprowadzeniu identyfikatora, hasła i wysłaniu formularza parametr `action` będzie miał wartość `DISPLAY_ENTRY`. Konstrukcja `switch` w kolejnym wywołaniu skryptu *edit_member.php* wywoła funkcję `display_entry()`, odpowiedzialną za sprawdzenie hasła i wyświetlenie informacji o członku Ligi, jeśli podane hasło jest poprawne:

```
function display_entry ($dbh)
{
    # Pobranie parametrów skryptu; usunięcie znaków odstępu z identyfikatora, ale nie
    # z hasła, ponieważ hasło musi być idealnie dopasowane.

    $member_id = trim (script_param ("member_id"));
    $password = script_param ("password");

    if (empty ($member_id))
        die ("Nie podano identyfikatora członka Ligi.\n");
    if (!ctype_digit ($member_id)) # Wartość musi być liczbą całkowitą.
        die ("Nieprawidłowy identyfikator (musi być liczbą całkowitą).\n");
    if (empty ($password))
        die ("Nie podano hasła.\n");
    if (check_pass ($dbh, $member_id, $password)) # Zwykły członek Ligi.
        $admin = FALSE;
    else if (check_pass ($dbh, 0, $password)) # Administrator.
        $admin = TRUE;
    else
        die ("Nieprawidłowe hasło.\n");

    $stmt = "SELECT
        last_name, first_name, suffix, email, street, city,
        state, zip, phone, interests, member_id, expiration
    FROM member WHERE member_id = ?
    ORDER BY last_name";
    $sth = $dbh->prepare ($stmt);
    $sth->execute (array ($member_id));

    if (!$row = $sth->fetch ())
        die ("Nie znaleziono członka Ligi o identyfikatorze $member_id.\n");

    printf("<form method=\"%post\" action=\"%s?action=%d\">\n",
        script_name (),
        UPDATE_ENTRY);

    # Dodanie identyfikatora i hasła członka Ligi jako ukrytych wartości, aby w trakcie kolejnego wywołania
    # skryptu było wiadomo, który rekord odpowiada tym wartościom. Dzięki temu użytkownik
    # nie będzie musiał ponownie podawać danych uwierzytelniających.

    hidden_field ("member_id", $member_id);
```

```

hidden_field ("password", $password);

# Sformatowanie wyników zapytania.

print("<table>\n");

# Wyświetlenie identyfikatora członka Ligi w postaci statycznego tekstu.

display_column ("Identyfikator członka Ligi", $row, "member_id", FALSE);

# Zmienna $admin ma wartość true, jeśli użytkownik podał hasło administracyjne.
# W przeciwnym razie wartością zmiennej jest false. Administrator ma możliwość edycji
# daty wygaśnięcia członkostwa, zwykły użytkownik nie ma tej możliwości.

display_column ("Data wygaśnięcia członkostwa", $row, "expiration", $admin);

# Wyświetlenie pozostałych wartości jako tekstu możliwego do edycji.

display_column ("Nazwisko", $row, "last_name");
display_column ("Imię", $row, "first_name");
display_column ("Przyrostek", $row, "suffix");
display_column ("E-mail", $row, "email");
display_column ("Ulica", $row, "street");
display_column ("Miejscowość", $row, "city");
display_column ("Województwo", $row, "state");
display_column ("Kod pocztowy", $row, "zip");
display_column ("Telefon", $row, "phone");
display_column ("Zainteresowania", $row, "interests");

print("</table>\n");

submit_button ("button", "Wyślij");
print "</form>\n";
}

```

Pierwszym zadaniem funkcji `display_entry()` jest weryfikacja hasła. Jeżeli hasło podane przez użytkownika odpowiada hasłu przechowywanemu w tabeli `member_pass` dla danego identyfikatora członka Ligi lub jest hasłem administracyjnym (czyli hasłem dla specjalnego członka Ligi o identyfikatorze 0), skrypt *edit_member.php* wyświetla formularz pozwalający na edycję informacji o członku Ligi. Funkcja sprawdzająca hasło `check_pass()` wykonuje proste zapytania pobierające rekord z tabeli `member_pass`, a następnie porównuje wartość kolumny `password` z hasłem podanym przez użytkownika w formularzu logowania:

```

function check_pass ($dbh, $id, $pass)
{
    $stmt = "SELECT password FROM member_pass WHERE member_id = ?";
    $sth = $dbh->prepare ($stmt);
    $sth->execute (array ($id));
    # Wartość TRUE w przypadku znalezienia i dopasowania hasła.
    return (($row = $sth->fetch ()) && $row["password"] == $pass);
}

```

Przyjmując założenie, że podane zostało prawidłowe hasło, funkcja `display_entry()` wyszukuje w tabeli `member` rekord odpowiadający wskazanemu identyfikatorowi członka Ligi, a następnie generuje formularz edycji zainicjalizowany z bieżącymi wartościami rekordu.

Większość pól to możliwy do edycji tekst, co pozwala użytkownikowi na jego łatwą zmianę. Istnieją jednak dwa wyjątki. Pierwszy, wartość `member_id`, jest wyświetlany w postaci statycznego tekstu. To jest wartość klucza unikalnie identyfikująca rekord i dlatego nie powinna być zmieniana. Drugi, data wygaśnięcia członkostwa, nie jest wartością, którą użytkownik może zmieniać. (W takim przypadku członek Ligi mógłby dowolnie ustawić datę wygaśnięcia członkostwa, a tym samym odnawiać członkostwo bez konieczności opłacania składek). Z drugiej strony, jeśli w trakcie logowania zostanie użyte hasło administracyjne, skrypt zezwoli na edycję pola określającego datę wygaśnięcia członkostwa. Przyjmując założenie, że sekretarka Ligi Historycznej zna hasło administracyjne, będzie miała możliwość zmiany daty wygaśnięcia członkostwa osobom, które przedłużyły członkostwo, opłacając składki.

Funkcja `display_column()` zajmuje się obsługą wyświetlania etykiet pól oraz ich wartości. Argumentami funkcji są etykieta wyświetlana obok pola, tablica zawierająca rekord do edycji, nazwa kolumny w rekordzie zawierająca wartość danego pola oraz wartość boolowska wskazująca, czy wyświetlane pole pozwala na edycję, czy jest tylko do odczytu. Wspomniana wartość boolowska jest opcjonalna, domyślnie to `TRUE`:

```
function display_column ($label, $row, $col_name, $editable = TRUE)
{
    print "<tr>\n";
    print "<td>" . htmlspecialchars ($label) . "</td>\n";
    print "<td>";
    if ($editable) # Wyświetlenie w postaci pola, którego zawartość można edytować.
        text_field ("row[$col_name]", $row[$col_name], 80);
    else # Wyświetlenie w postaci tekstu tylko do odczytu.
        print (htmlspecialchars ($row[$col_name]));
    print "</td>\n";
    print "</tr>\n";
}
```

W przypadku wartości możliwych do edycji funkcja `display_column()` generuje pola tekstowe, używając nazw w formacie `rekord[nazwa_kolumny]`. W ten sposób po wysłaniu formularza przez użytkownika PHP umieści wszystkie wartości pól w tablicy, a kluczami elementów będą nazwy kolumn. Dzięki temu można je później łatwo wyodrębnić i powiązać wartość każdego pola z odpowiadającą mu kolumną tabeli `member` podczas uaktualniania rekordu w bazie danych. Na przykład, po umieszczeniu tablicy w zmiennej `$row` dostęp do numeru telefonu odbywa się za pomocą `$row["phone"]`.

Funkcja `display_entry()` umieszcza w ukrytych polach formularza wartości `member_id` i `password`, aby zostały przekazane do kolejnego wywołania skryptu `edit_member.php`, gdy użytkownik wyśle formularz z uaktualnionymi danymi. Dzięki identyfikatorowi członka Ligi skrypt wie, który rekord powinien być uaktualniony. Z kolei hasło pozwala na weryfikację, że użytkownik był wcześniej zalogowany. (Zauważ, że przedstawiona tutaj prosta metoda uwierzytelniania opiera się na przekazywaniu hasła w postaci zwykłego tekstu, co ogólnie rzecz biorąc, nie jest dobrym rozwiązaniem. Jednak w przypadku projektu Ligi Historycznej takie rozwiązanie jest w zupełności wystarczające. Podczas przeprowadzania operacji takich jak transakcje finansowe zdecydowanie powinieneś zastosować znacznie bezpieczniejsze formy uwierzytelniania).

Funkcja odpowiedzialna za uaktualnianie danych członka Ligi po wysłaniu formularza przedstawia się następująco:

```
function update_entry ($dbh)
{
    # Pobranie parametrów skryptu; usunięcie znaków odstępu z identyfikatora,
    # ale nie z hasła (ponieważ hasło musi być idealnie dopasowane) oraz
    # nie z rekordu (ponieważ jest tablicą).

    $member_id = trim (script_param ("member_id"));
    $password = script_param ("password");
    $row = script_param ("row");

    if (empty ($member_id))
        die ("Nie podano identyfikatora członka Ligi.\n");
    if (!ctype_digit ($member_id)) # Wartość musi być liczbą całkowitą.
        die ("Nieprawidłowy identyfikator (musi być liczbą całkowitą).\n");
    if (!check_pass ($dbh, $member_id, $password)
        && !check_pass ($dbh, 0, $password))
        die ("Nieprawidłowe hasło.\n");

    # Przeanalizowanie metadanych tabeli member w celu ustalenia, czy
    # kolumny akceptują wartość NULL. (Trzeba się upewnić, że informacje
    # o obsłudze wartości NULL są zapisane wielkimi literami).

    $stmt = "SELECT COLUMN_NAME, UPPER(IS_NULLABLE)
            FROM INFORMATION_SCHEMA.COLUMNS
            WHERE TABLE_SCHEMA = ? AND TABLE_NAME = ?";
    $sth = $dbh->prepare ($stmt);
    $sth->execute (array ("sampdb", "member"));
    $nullable = array ();
    while ($info = $sth->fetch ())
        $nullable[$info[0]] = ($info[1] == "YES");

    # Iteracja przez wszystkie pola formularza, wartości są używane
    # do przygotowania zapytania UPDATE zawierającego miejsca zarezerwowane,
    # do których wstawiane są wartości pochodzące z tablicy.

    $stmt = "UPDATE member ";
    $delim = "SET";
    $params = array ();
    foreach ($row as $col_name => $val)
    {
        $stmt .= "$delim $col_name=?";
        $delim = ",";
        # Jeżeli w formularzu nie podano wartości, należy użyć wartości NULL
        # odpowiedniej kolumny, o ile obsługuje ona wartości NULL. To forma ochrony
        # przed umieszczeniem pustego ciągu tekstowego w kolumnie daty wygaśnięcia członkostwa,
        # gdy powinna mieć wartość na przykład NULL.
        $val = trim ($val);
        if (empty ($val))
        {
            if ($nullable[$col_name])
                $params[] = NULL; # Wstawienie wartości NULL.
            else
                $params[] = ""; # Wstawienie pustego ciągu tekstowego.
        }
    }
}
```

```

        else
            $params[] = $val;
    }
    $stmt .= " WHERE member_id = ?";
    $params[] = $member_id;

    $sth = $dbh->prepare ($stmt);
    $sth->execute ($params);
    printf ("<br /><a href=\"%s\">Edycja rekordu innego członka Ligi</a>\n",
           script_name ());
}

```

W pierwszej kolejności skrypt ponownie weryfikuje hasło, aby upewnić się, że nie następuje próba wysłania fałszywego formularza, a dopiero później uaktualnia informacje o członku Ligi. Samo uaktualnienie wymaga pewnej ostrożności, ponieważ jeśli pole formularza pozostało niewypełnione, to może wystąpić potrzeba wstawienia wartości NULL zamiast pustego ciągu tekstowego. Przykładem może być tutaj kolumna `expiration`. Przyjmujemy założenie, że sekretarka Ligi loguje się za pomocą hasła administracyjnego (a więc ma możliwość zmiany daty wygaśnięcia członkostwa) i usuwa wartość pola, co oznacza członkostwo dożywotnie. W takim przypadku w tabeli bazy danych powinna być wstawiona wartość NULL zamiast pustego ciągu tekstowego, który przecież nie jest prawidłową datą. Dlatego konieczne jest ustalenie, które kolumny akceptują wartość NULL, i wstawianie wtedy wartości NULL zamiast pustego ciągu tekstowego, gdy pole formularza nie zostało wypełnione.

Aby rozwiązać wspomniany problem, funkcja `update_entry()` wyszukuje metadane tabeli `member` i tworzy tablicę asocjacyjną, w której kluczami elementów są nazwy kolumn akceptujących wartości NULL. Te informacje są dostępne w tabeli `COLUMNS` bazy danych `INFORMATION_SCHEMA`. Potrzebne nam wartości z wymienionej tabeli to nazwa kolumny i informacja, czy akceptuje ona wartości NULL (to znaczy wartości `COLUMN_NAME` i `IS_NULLABLE`).

Na tym etapie skrypt *edit_member.php* jest już ukończony. Umieść go w katalogu *ushl* serwera WWW i podaj członkom Ligi ich hasła dostępu. Każdy z nich będzie mógł za pomocą przeglądarki internetowej uaktualnić informacje o sobie.

Wprowadzenie do administracji bazą danych MySQL

Na przestrzeni czasu baza danych zyskała nowe możliwości, choć jednocześnie stała się znacznie bardziej skomplikowana. Jednak w porównaniu do innych systemów baz danych MySQL jest względnie prosta w użyciu i nie wymaga zbyt dużego wysiłku podczas instalacji, konfiguracji i pracy. Wspomniana prostota niewątpliwie miała ogromny wpływ na popularność MySQL, zwłaszcza wśród osób, które nie są i nie chcą stać się administratorami systemu. Zdobyć większej wiedzy z zakresu MySQL oczywiście pomaga w codziennej pracy z nim, ale nie jest warunkiem koniecznym do osiągnięcia sukcesu w pracy z MySQL.

Mimo wszystko, niezależnie od poziomu wiedzy użytkownika, instalacja MySQL nie może być pozostawiona sama sobie. Ktoś musi się nią zajmować, zapewniać bezproblemowe i efektywne działanie serwera, a także rozwiązywać ewentualne problemy, gdy się pojawią. Jeżeli zadanie obsługi bazy danych MySQL spadnie na Ciebie, koniecznie zapoznaj się z tym rozdziałem.

Część III tej książki, zatytułowana „Administracja serwerem MySQL”, przedstawia obowiązki należące do administratora bazy danych MySQL. W tym rozdziale zostaną zaprezentowane ogólne zadania wykonywane w trakcie administracji instalacją MySQL. W kolejnych rozdziałach znajdziesz informacje szczegółowe dotyczące sposobu wykonywania wymienionych tutaj zadań.

Jeżeli jesteś nowym lub niedoświadczonym administratorem bazy danych MySQL, nie pozwól, aby przeraziła Cię przedstawiona w rozdziale długa lista obowiązków. Wszystkie zadania wymienione w kolejnych podrozdziałach są ważne, ale nie musisz ich poznać od razu. Jeśli chcesz, rozdziały w tej części książki potraktuj jak przewodnik i wyszukuj konkretne zagadnienia, gdy będziesz musiał je poznać.

Jeżeli jesteś doświadczonym administratorem innych systemów baz danych, przekonasz się, że obsługa instalacji MySQL jest pod pewnymi względami podobna, a Twoje doświadczenie będzie cennym zasobem. Jednak administracja MySQL ma także własne, unikalne wymagania. Ta część książki ma za zadanie zaznajomić Cię z nimi.

10.1. Komponenty MySQL

System bazy danych składa się z wielu komponentów. Powinieneś znać te komponenty, przeznaczenie każdego z nich, a także rozumieć naturę systemu, którym administrujesz, i narzędzi pomagających w wykonywaniu zadań administracyjnych. Jeżeli poświęcisz czas na zrozumienie nadzorowanego systemu, Twoja praca stanie się o wiele łatwiejsza. Dlatego też powinieneś zaznajomić się z wymienionymi poniżej aspektami bazy danych MySQL.

Serwer MySQL. Serwer (`mysqld`) jest miejscem centralnym instalacji MySQL, przeprowadza wszystkie operacje na bazach danych i tabelach. W systemach UNIX dostępnych jest wiele skryptów pomagających w konfiguracji i uruchamianiu serwera. Na przykład, `mysqld_safe` to program używany do uruchamiania serwera, monitorowania go oraz ponownego uruchamiania w przypadku zamknięcia serwera. Skrypt `mysql.server` jest użyteczny w systemach UNIX używających do uruchamiania usług systemowych tak zwanych katalogów poziomu działania. Jeśli w jednym komputerze uruchomionych jest wiele serwerów, narzędzie `mysqld_multi` znacznie ułatwi zarządzanie nimi. Z kolei w systemie Windows masz możliwość uruchomienia serwera z poziomu wiersza poleceń lub jako usługi Windows.

Klienty i narzędzia MySQL. Podczas komunikacji z serwerem MySQL możesz wykorzystać wiele dostępnych programów. Niektóre z nich bardziej od pozostałych nadają się do wykonywania zadań administracyjnych:

- **mysql** — Program interaktywny pozwalający na wysyłanie zapytań do serwera i wyświetlanie wyników ich wykonania. Programu `mysql` można również używać do wykonywania skryptów wsadowych (pliki tekstowe zawierające polecenia SQL).
- **mysqladmin** — Program administracyjny przeznaczony do wykonywania zadań takich jak zamknięcie serwera, sprawdzenie jego konfiguracji i monitorowanie stanu, jeśli wydaje się, że serwer nie działa prawidłowo.
- **mysqldump** — Narzędzie pozwalające na tworzenie kopii zapasowej bazy danych oraz kopiowanie baz danych do innego serwera.
- **mysqlcheck** i **myisamchk** — Programy pomagające w przeprowadzaniu operacji sprawdzania, analizy i optymalizacji tabel, a także naprawy, jeśli tabele zostały uszkodzone. Program `mysqlcheck` działa z tabelami MyISAM i do pewnego stopnia także z tabelami innych silników bazy danych. Natomiast program `myisamchk` jest przeznaczony do użycia jedynie z tabelami MyISAM.

Język serwera, czyli SQL. Z serwerem powinieneś komunikować się w jego języku. Oto prosty przykład: być może będziesz chciał sprawdzić, dlaczego uprawnienia

użytkownika nie działają w oczekiwany sposób. Nie ma innego rozwiązania jak tylko bezpośrednia komunikacja z serwerem, którą można przeprowadzić za pomocą programu klienta `mysql` i wykonywania zapytań SQL pozwalających na analizę tabel uprawnień.

Jeżeli w ogóle nie znasz języka SQL, to musisz go poznać przynajmniej w podstawowym zakresie. Brak biegłości w SQL jedynie utrudnia wykonywanie zadań administracyjnych, natomiast czas poświęcony na naukę SQL zwróci się wielokrotnie. Faktyczne opanowanie SQL może trochę potrwać, ale nabycie podstawowych umiejętności nie zajmie zbyt dużo czasu. Informacje na temat SQL i używania klienta `mysql` działającego w wierszu poleceń znajdziesz w rozdziale 1., zatytułowanym „Rozpoczęcie pracy z MySQL”.

Katalog danych w MySQL. Pliki baz danych i stanu serwer przechowuje w katalogu danych. Bardzo ważne jest zrozumienie struktury i zawartości tego katalogu, aby wiedzieć, jak serwer używa systemu plików do przedstawiania baz danych i tabel, a także gdzie umieszcza pliki dzienników i co się w nich znajduje. Z powodu zapotrzebowania na pamięć masową i wydajność działania powinieneś również znać opcje dotyczące alokacji dysku.

10.2. Ogólna administracja MySQL

Ogólna administracja oznacza przede wszystkim obowiązki w zakresie zapewnienia działania `mysqld`, czyli serwera MySQL, oraz umożliwienia użytkownikom uzyskania dostępu do serwera. W realizacji wymienionych obowiązków najważniejsze są wymienione poniżej zadania.

Uruchamianie i zamykanie serwera. Powinieneś wiedzieć, jak ręcznie z poziomu wiersza poleceń uruchomić i zatrzymać serwer, a także jak skonfigurować automatyczne uruchamianie i zatrzymywanie serwera wraz z uruchamianiem i zamykaniem systemu. Bardzo ważna jest umiejętność ponownego uruchomienia serwera, jeśli ulegnie on awarii lub nie uruchomi się prawidłowo.

Konto użytkownika do zadań administracyjnych. Powinieneś rozumieć różnicę między kontami użytkowników MySQL a kontami logowania w systemach UNIX i Windows. Musisz potrafić skonfigurować konto użytkownika MySQL, wskazując użytkowników, którzy mogą nawiązać połączenie z serwerem, oraz określając, skąd mogą nawiązać połączenie i co mogą zrobić po jego nawiązaniu. Konieczna jest również umiejętność zerowania zapomnianych haseł użytkowników.

Obsługa dzienników zdarzeń. Powinieneś znać rodzaje dostępnych dzienników zdarzeń, szczególnie użytecznych dla Ciebie, a także wiedzieć, jak je obsługiwać. Rotacja dzienników zdarzeń oraz czas ich wygaśnięcia to ważne zagadnienia, dzięki którym chronisz system plików przed jego zapełnieniem plikami dzienników zdarzeń.

Konfiguracja i dostrajanie serwera. Serwer MySQL oferuje ogromne możliwości w zakresie konfiguracji. Niektóre z możliwości konfiguracji dotyczą kontroli nad obsługiwanymi silnikami bazy danych, domyślnym kodowaniem znaków oraz domyślną strefą czasową.

Inne kwestie związane z konfiguracją dotyczą dostrajania serwera. Na pewno chcesz zapewnić jak najlepsze działanie serwera. Najszybszą metodą poprawienia wydajności jego działania jest zakup większej ilości pamięci oraz szybszych dysków twardych. Jednak wspomniane techniki nie są zamiennikiem dla zrozumienia zasady działania serwera. Powinienes znać parametry dostępne do dostrajania jego działania oraz sposoby ich stosowania w konkretnych sytuacjach. W przypadku niektórych witryn internetowych większość zapytań dotyczy pobierania danych. Z kolei w innych dominują operacje wstawiania i uaktualniania danych. Na wybór modyfikowanych parametrów wpływ będzie miał rodzaj zapytań wykonywanych w danej witrynie internetowej.

Zarządzanie wieloma serwerami. W pewnych sytuacjach użytecznym rozwiązaniem jest uruchamianie wielu serwerów w tym samym komputerze. W ten sposób możesz przetestować nowe wydanie MySQL, pozostawiając nietknięte środowisko produkcyjne, lub zapewnić lepszą ochronę prywatności pewnym grupom użytkowników przez przydzielenie każdej grupie własnego serwera. (Ostatni z wymienionych scenariuszy jest najczęściej stosowany przez dostawców usług internetowych). W wymienionych sytuacjach powinienes wiedzieć, jak skonfigurować wiele jednocześnie działających instalacji.

Uaktualnianie oprogramowania MySQL. Od czasu do czasu pojawiają się nowe wydania MySQL. Powinienes wiedzieć, kiedy przeprowadzić uaktualnienie serwera do nowszych wersji i jak wykorzystać zalety w postaci usuniętych błędów oraz dodanych nowych funkcji. Trzeba też wiedzieć, kiedy rozsądne będzie wstrzymanie się z uaktualnieniem oprogramowania oraz jak wybierać między wersjami stabilnymi i rozwojowymi.

10.3. Kontrola dostępu i zapewnianie bezpieczeństwa

Podczas obsługi instalacji MySQL bardzo ważne jest zagwarantowanie, że dane użytkowników ufających bazie danych pozostaną bezpieczne. Administrator bazy danych jest odpowiedzialny za kontrolę dostępu do katalogu danych i serwera. Ponadto, powinien zrozumieć wymienione poniżej kwestie.

Bezpieczeństwo systemu plików. W komputerze może istnieć wiele kont użytkowników niepowiązanych z zadaniami administracyjnymi w MySQL. Bardzo ważne jest zagwarantowanie, aby tego rodzaju użytkownicy nie mieli dostępu do katalogu danych. W ten sposób nie będą mogli dobrać się do danych na poziomie systemu plików przez skopiowanie lub usunięcie tabel bądź też odczyt plików dzienników zdarzeń, w których mogą znajdować się informacje wrażliwe. Musisz wiedzieć, jak skonfigurować konto użytkownika używanego do uruchamiania serwera MySQL, jak skonfigurować katalog danych, aby jego właścicielem był wspomniany użytkownik, oraz jak uruchamiać serwer, aby działał z uprawnieniami wspomnianego użytkownika.

Bezpieczeństwo serwera MySQL. Konieczne jest zrozumienie sposobu działania systemu bezpieczeństwa MySQL podczas konfiguracji kont użytkowników, aby nadawać im odpowiednie uprawnienia dostępu do serwera MySQL. Użytkownicy nawiązujący połączenie przez sieć powinni mieć uprawnienia jedynie do wykonywania zadań, które muszą wykonywać. Nie chcesz przecież z powodu niezrozumienia sposobu działania systemu bezpieczeństwa przypadkowo udzielić użytkownikowi zbyt dużych uprawnień.

10.4. Obsługa bazy danych, tworzenie kopii zapasowych i replikacja

Każdy administrator MySQL ma nadzieję na uniknięcie sytuacji, w której będzie musiał poradzić sobie z uszkodzonymi lub usuniętymi tabelami bazy danych. Jednak sama nadzieja nie chroni przed problemami. Dlatego też powinieneś podjąć pewne kroki, aby zminimalizować ryzyko wystąpienia tego rodzaju sytuacji, oraz wiedzieć, co zrobić, jeśli już wspomniana sytuacja wystąpi.

Prewencja. Regularna prewencja może zminimalizować ryzyko wystąpienia uszkodzenia bądź zniszczenia bazy danych. To, oczywiście, nie zwalnia z tworzenia kopii zapasowych, ale regularna prewencja zmniejsza niebezpieczeństwo, że będziesz ich potrzebował.

Kopie zapasowe bazy danych. W przypadku awarii systemu posiadanie kopii zapasowej bazy danych ma znaczenie krytyczne. Dzięki nim będziesz miał możliwość przywrócenia baz danych do stanu w chwili awarii, przy jednoczesnym ograniczeniu do minimum utraconych danych. Pamiętaj, że tworzenie kopii zapasowej baz danych nie jest tym samym, czym jest tworzenie ogólnej kopii zapasowej całego systemu (na przykład w systemach UNIX za pomocą programu dump). Podczas wykonywania kopii zapasowej pliki tworzące tabele bazy danych mogą ulegać ciągłym zmianom na skutek aktywności serwera, więc przywrócenie wspomnianych plików nie daje gwarancji zachowania spójności tabel. Narzędzie `mysqldump` generuje pliki kopii zapasowej, które są znacznie bardziej użyteczne podczas przywracania bazy danych, i pozwala na tworzenie kopii zapasowej bez konieczności wyłączania serwera. Kopia zapasowa plików przydaje się również podczas przenoszenia baz danych w inne miejsce, na przykład po wypełnieniu dysku.

Naprawa po awarii. Jeżeli pomimo podjętych wysiłków nastąpi jakakolwiek katastrofa, powinieneś wiedzieć, co zrobić, aby naprawić lub odzyskać tabele. Naprawa po awarii może być potrzebna rzadko, ale kiedy już wystąpi taka konieczność, to będzie nieprzyjemne i bardzo stresujące zadanie (zwłaszcza gdy podczas tej operacji telefon się wręcz urywa, a do drzwi ciągle pukają kolejne osoby). Musisz jednak wiedzieć, jak poradzić sobie z zadaniem naprawy po awarii, ponieważ w przeciwnym razie użytkownicy będą bardzo niezadowoleni! Poznaj więc oferowane przez dystrybucję MySQL programy przeznaczone do sprawdzania i naprawy tabel. Naucz się przywracać dane z plików kopii zapasowej oraz używać binarnego dziennika zdarzeń do odzyskania zmian wprowadzonych w bazie danych po wykonaniu ostatniej kopii zapasowej.

Migracja bazy danych. Jeżeli zdecydujesz się na uruchomienie serwera MySQL w szybszym komputerze, będziesz musiał skopiować pliki baz danych do nowej maszyny. Dlatego też powinieneś znać odpowiednią procedurę i zastosować ją w razie konieczności. Zawartość plików bazy danych może być uzależniona od komputera; w takim przypadku nie możesz po prostu skopiować plików z jednego komputera do drugiego.

Replikacja bazy danych. Tworzenie kopii zapasowej bazy danych to wykonanie migawki jej stanu w pewnym punkcie w czasie. Innym rozwiązaniem jest zastosowanie replikacji, która polega na skonfigurowaniu dwóch powiązanych ze sobą serwerów. W takim przypadku zmiany wprowadzone w bazie danych zarządzanej przez jeden serwer będą na bieżąco przekazywane do odpowiedniej bazy danych zarządzanej przez drugi serwer.

Aby skorzystać z replikacji, konieczna jest umiejętność skonfigurowania serwera jako głównego serwera aplikacji oraz serwerów podległych replikujących główny. Jeśli wystąpią problemy i nastąpi zatrzymanie replikacji, musisz wiedzieć, jak wykryć problem i ponownie uruchomić replikację.

Przedstawiony powyżej ogólny zarys pokazuje obowiązki spoczywające na administratorze bazy danych MySQL. W kolejnych kilku rozdziałach dokładnie omówimy wymienione obowiązki i przedstawimy szczegółowe procedury pozwalające na efektywne wykonywanie zadań administracyjnych. W pierwszej kolejności będzie omówiony katalog danych serwera MySQL, ponieważ stanowi on najważniejszy przedmiot obsługi i doskonale powinieneś znać jego projekt i zawartość. Następnie przejdziemy do ogólnych zadań administracyjnych, przeanalizujemy system bezpieczeństwa w MySQL oraz obsługę i tworzenie kopii zapasowych.

Katalog danych w MySQL

Pod względem koncepcyjnym różne systemy baz danych mają jedną wspólną cechę: zarządzają zestawem baz danych zawierających zestawy tabel. Jednak każdy system bazy danych charakteryzuje się własnym rozwiązaniem w zakresie zarządzania danymi i MySQL nie jest tutaj wyjątkiem. Domyślnie, serwer MySQL (`mysqld`) przechowuje wszystkie informacje w lokalizacji nazywanej katalogiem danych MySQL. We wspomnianym katalogu znajdują się wszystkie bazy danych, pliki stanu i pliki dzienników zdarzeń dostarczających informacje o działalności serwera. Jeżeli jesteś w jakimkolwiek stopniu odpowiedzialny za zadania administracyjne instalacji MySQL, powinieneś zapoznać się z projektem i sposobem używania katalogu danych, ponieważ ta wiedza będzie Ci potrzebna w wypełnianiu obowiązków administracyjnych. Nawet jeśli nie zamierzasz zajmować się administracją MySQL, to lektura niniejszego rozdziału może przynieść pewne korzyści, a wiedza o sposobie działania serwera na pewno nie zaszkodzi.

W tym rozdziale zostaną omówione wymienione poniżej zagadnienia:

- **Położenie katalogu danych.** Ponieważ katalog danych odgrywa ważną rolę w działalności serwera MySQL, powinieneś potrafić wskazać jego położenie, aby efektywnie administrować zawartością katalogu danych.
- **Sposób, w jaki serwer organizuje dostęp do baz danych i tabel oraz nim zarządza.** To bardzo ważne w celu konfiguracji harmonogramu obsługi serwera, a także przeprowadzania procesu odzyskiwania danych w przypadku uszkodzenia tabel.
- **Rodzaje generowanych plików stanu i dzienników zdarzeń oraz ich zawartość.** Wspomniane pliki zawierają użyteczne informacje na temat działania serwera i są wręcz nieocenione po wystąpieniu jakichkolwiek problemów.
- **Sposób zmiany domyślnego położenia i organizacji katalogu danych.** To może być ważne podczas zarządzania alokacją zasobów dyskowych w systemie, na przykład przez zrównoważenie aktywności dyskowej między dostępnymi napędami lub przez przeniesienie danych do systemów plików z większą ilością wolnej przestrzeni. Zdobyta tutaj wiedza jest także użyteczna podczas planowania miejsca położenia nowych baz danych.

W rozdziale przyjęto założenie, że w przypadku systemów UNIX istnieje konto logowania przeznaczone do wykonywania zadań administracyjnych oraz dla działającego serwera. W tej książce nazwa użytkownika i grupy wspomnianego konta to `mysql`. W podpunkcie 12.2.1.1, zatytułowanym „Działanie serwera w ramach pozbawionego uprawnień konta logowania”, przedstawiono powody, dla których warto używać oddzielnego konta logowania do wykonywania zadań administracyjnych MySQL.

11.1. Położenie katalogu danych

Domyślne położenie katalogu danych (*data*) jest skompilowane w serwerze. W systemach UNIX z reguły będzie to katalog `/usr/local/mysql/data` w przypadku instalacji MySQL z dystrybucji binarnej bądź źródłowej lub `/var/lib/mysql` po instalacji z pakietu RPM. Z kolei w systemach Windows najczęściej będzie to katalog `C:\ProgramData\MySQL` lub `C:\Documents and Settings\All Users\Application Data\MySQL`, w zależności od używanej wersji Windows.

Jeżeli kompilujesz MySQL ze źródeł, domyślne położenie katalogu danych możesz wskazać za pomocą opcji wiersza poleceń `-DMYSQL_DATADIR=nazwa_katalogu` podczas uruchamiania `CMake`.

Aby położenie katalogu danych wskazać w chwili uruchamiania serwera, należy użyć opcji `--datadir=nazwa_katalogu`. To użyteczne rozwiązanie w celu wskazania innego katalogu niż domyślny zdefiniowany w trakcie kompilacji. Jeszcze innym sposobem jest podanie katalogu w pliku opcji odczytywanym przez serwer w chwili jego uruchamiania. W takim przypadku nie ma konieczności podawania katalogu danych w wierszu poleceń w trakcie każdego uruchamiania serwera.

Jako administrator MySQL powinieneś znać miejsce położenia katalogu danych serwera, ale jeśli go nie znasz (prawdopodobnie po przejęciu obowiązków po poprzednim administradorze, który nie pozostawił wystarczająco dokładnych informacji o instalacji MySQL), istnieje kilka sposobów na sprawdzenie wspomnianego położenia. Poniżej przedstawiono jedną z metod, stosowaną, gdy serwer nie jest uruchomiony. Kolejna metoda pozwala na sprawdzenie położenia katalogu danych serwera w przypadku działającego serwera.

Spójrz na plik opcji odczytywany w chwili uruchamiania serwera. Na przykład, w systemach UNIX po otwarciu pliku `/etc/my.cnf` w grupie `[mysqld]` możesz znaleźć wiersz `datadir`:

```
[mysqld]
datadir= /ścieżka/dostępu/do/katalogu/data
```

Ścieżka dostępu wskazuje położenie katalogu danych serwera.

Jeżeli nie jesteś pewien, czy serwer odczytuje pliki opcji, wywołaj go w przedstawiony poniżej sposób i sprawdź komunikat pomocy, który wyświetla położenia plików opcji:

```
% mysql --verbose --help
```

W przypadku działającego serwera nawiąż z nim połączenie i sprawdź położenie katalogu danych. Serwer zawiera pewną liczbę zmiennych systemowych dotyczących jego funkcjonowania i może wyświetlić ich wartości. Położenie katalogu danych wskazuje zmienna `datadir`, której wartość można wyświetlić za pomocą zapytania `SHOW VARIABLES` lub polecenia `mysqladmin variables`. Jeżeli zapytanie `SHOW VARIABLES` wykonasz w systemie UNIX, jego wynik może przedstawiać się następująco:

```
mysql> SHOW VARIABLES LIKE 'datadir';
+-----+-----+
| Variable_name | Value                               |
+-----+-----+
| datadir       | /usr/local/mysql/data/           |
+-----+-----+
```

Z poziomu wiersza poleceń użyj narzędzia `mysqladmin`:

```
% mysqladmin variables
+-----+-----+
| Variable_name | Value                               |
+-----+-----+
...
| datadir       | /usr/local/mysql/data/           |
...
```

W systemie Windows położeniem katalogu danych może być na przykład `C:\ProgramData\MySQL`.

Jeżeli masz kilka działających serwerów, będą one nasłuchiwały na różnych interfejsach sieciowych (porty TCP/IP, pliki gniazd systemu UNIX, nazwane potoki w Windows lub pamięć współdzielona). Informacje o położeniu katalogu danych możesz otrzymać po nawiązaniu połączenia z każdym serwerem po kolei za pomocą odpowiednich opcji parametrów połączenia.

W przypadku istnienia katalogu danych i konieczności jego przeniesienia do innego położenia zapoznaj się z podrozdziałem 11.3, zatytułowanym „Przeniesienie zawartości katalogu danych”, w którym omówiono techniki przenoszenia katalogu danych MySQL.

11.2. Struktura katalogu danych

Katalog danych MySQL zawiera wszystkie bazy danych, którymi zarządza serwer. Ogólnie rzecz biorąc, mają one postać struktury drzewa zaimplementowanego w bardzo prosty sposób, wykorzystujący hierarchiczną strukturę systemu plików UNIX lub Windows:

- Każda baza danych ma własny podkatalog w katalogu danych MySQL.
- Tabele, widoki i wyzwalacze w bazie danych odpowiadają plikom w podkatalogu danej bazy danych.

Poszczególne silniki bazy danych mogą stosować strukturę pamięci masowej różniącą się od ogólnej hierarchicznej implementacji bazy danych za pomocą katalogów i plików. Na przykład, silnik bazy danych InnoDB może we wspólnej przestrzeni tabel

przechowywać wszystkie tabele InnoDB ze wszystkich baz danych. Wspomniana przestrzeń tabel składa się z jednego lub więcej ogromnych plików traktowanych jako pojedyncza, jednolita struktura danych, w ramach której przedstawiane są tabele i indeksy. Domyślnie silnik InnoDB przechowuje pliki przestrzeni tabel w katalogu danych MySQL.

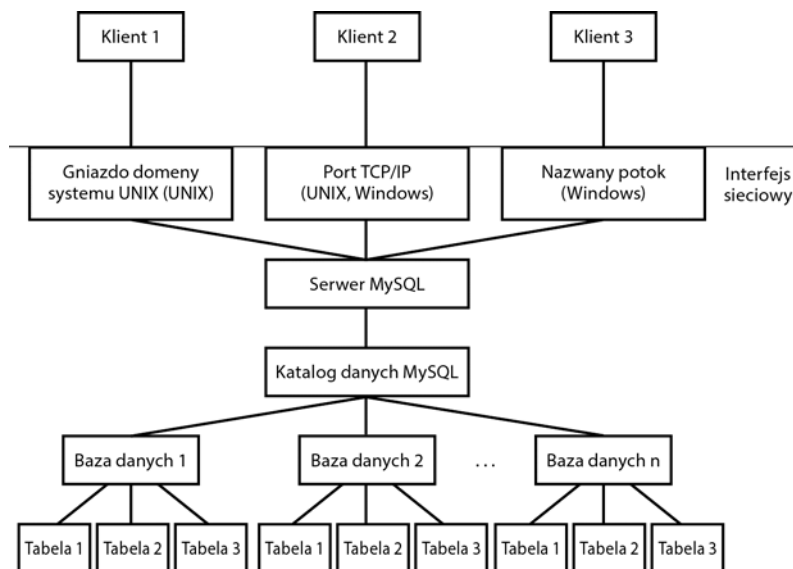
Katalog danych może zawierać także jeszcze inne pliki:

- Plik PID (ang. *Process ID*) serwera. Podczas uruchamiania serwer zapisuje w tym pliku identyfikator procesu, co pozwala innym programom na poznanie wspomnianej wartości, jeśli muszą wysłać jakikolwiek sygnał do serwera.
- Generowane przez serwer pliki stanu i dzienników zdarzeń. Wspomniane pliki zawierają ważne informacje o działalności serwera i są wręcz nieocenione dla administratorów, zwłaszcza w przypadku wystąpienia jakichkolwiek problemów, gdy zachodzi potrzeba znalezienia źródła problemu. Na przykład, jeśli określone zapytanie nie jest z powodzeniem wykonywane w serwerze, problem bardzo często można ustalić przez analizę plików dzienników zdarzeń. (Jeżeli skonfigurujesz serwer w taki sposób, aby rejestracja informacji odbywała się w tabelach bazy danych zamiast w plikach dzienników zdarzeń, to tabele zdarzeń znajdziesz w bazie danych `mysql`).
- Pliki powiązane z serwerem, na przykład plik klucza DES lub pliki kluczy i certyfikatów SSL serwera. Wymienione pliki administratorzy bardzo często umieszczają w katalogu danych MySQL.

11.2.1. W jaki sposób serwer MySQL zapewnia dostęp do danych?

Kiedy baza danych MySQL jest używana w typowej konfiguracji klient-serwer, wszystkie bazy danych znajdujące się w katalogu danych są zarządzane przez pojedynczą jednostkę, czyli serwer MySQL (`mysqld`). Programy klientów nie mają możliwości bezpośredniego przeprowadzania operacji na danych. Zamiast tego serwer zapewnia pojedynczy punkt kontaktu z bazą danych działający w charakterze pośrednika między programem klienta i danymi, których chce użyć. Tę architekturę pokazano na rysunku 11.1.

Podczas uruchamiania serwera następuje otworenie wszystkich obsługiwanych przez niego plików dzienników zdarzeń, a następnie udostępnienie katalogu danych interfejsowi sieciowemu przez nasłuchiwanie różnych rodzajów połączeń sieciowych. (Informacje szczegółowe dotyczące wyboru interfejsu sieciowego znajdziesz w punkcie 12.2.4, zatytułowanym „W jaki sposób serwer nasłuchuje połączeń?”). W celu uzyskania dostępu do danych program klienta nawiązuje połączenie z serwerem i wysyła zapytania SQL odpowiedzialnych za wykonanie określonych operacji, takich jak utworzenie tabeli, wybór rekordów lub uaktualnienie rekordów. Serwer wykonuje wskazane operacje i zwraca klientowi ich wynik. Ponieważ serwer jest wielowątkowy, jednocześnie może obsługiwać wiele połączeń z klientami. Jednak ponieważ jednocześnie można przeprowadzać tylko jedną operację uaktualniania, w efekcie serwer serializuje zapytania, aby dwa klienty nigdy nie miały szansy na jednoczesne uaktualnienie tego samego rekordu.



Rysunek 11.1. Sposób, w jaki serwer MySQL kontroluje dostęp do katalogu danych

W normalnych warunkach serwer działający w charakterze jedyne arbitra dostępu do bazy danych jest gwarancją uniknięcia wszelkiego rodzaju uszkodzeń, które mogłyby powstać na skutek jednoczesnego przetwarzania tabel bazy danych przez wiele klientów. Administratorzy powinni jednak zdawać sobie sprawę, że zdarzają się sytuacje, w których serwer nie ma wyłącznej kontroli nad katalogiem danych. Narzędzia bezpośredniego dostępu, takie jak `myisamchk`, są używane do przeprowadzania zadań administracyjnych względem tabel `MyISAM`, podczas rozwiązywania problemów, operacji naprawy lub kompresji. Tego rodzaju programy działają bezpośrednio na plikach odpowiadających tabelom. Ponieważ te narzędzia mogą zmienić zawartość tabel, używanie w tym samym czasie tabel przez serwer może doprowadzić do ich uszkodzenia.

Najbardziej oczywistym sposobem uniknięcia wymienionego problemu jest zatrzymanie serwera przed uruchomieniem programu narzędziowego. Jeśli nie ma takiej możliwości, konieczne jest poznanie sposobu zakazania serwerowi uzyskiwania dostępu do tabeli, gdy używasz narzędzia bezpośrednio działającego na plikach tej tabeli. W podrozdziale 14.2, zatytułowanym „Obsługa bazy danych w działającym serwerze”, znajdziesz informacje dotyczące współpracy z serwerem podczas używania programów narzędziowych. Alternatywą dla narzędzia `myisamchk` jest wykonanie zapytań takich jak `CHECK TABLE` i `REPAIR TABLE` (lub wykorzystanie narzędzia `mysqlcheck` wykonującego wymienione zapytania). Wymienione zapytania eliminują problem współpracy z serwerem, ponieważ samemu serwerowi nakazują przeprowadzenie operacji na tabelach.

11.2.2. Przedstawienie baz danych w systemie plików

Każda baza danych zarządzana przez serwer MySQL ma własny katalog. Istnieje on jako podkatalog w katalogu danych i ma taką samą nazwę jak baza danych, którą przedstawia. Na przykład, jeżeli *katalog_danych* określa położenie katalogu danych w serwerze, a nazwą bazy danych jest *moja_baza_danych*, wówczas katalogiem bazy danych jest *katalog_danych/moja_baza_danych* w systemach UNIX lub *katalog_danych\moja_baza_danych* w Windows.

Zapytanie `SHOW DATABASES` po prostu generuje listę podkatalogów znajdujących się w katalogu danych MySQL.

Zapytanie `CREATE DATABASE nazwa_bazy_danych` tworzy podkatalog o podanej nazwie w katalogu danych MySQL. Utworzony w ten sposób podkatalog będzie katalogiem bazy danych. Ponadto, w katalogu bazy danych wymienione zapytanie tworzy plik *db.opt* zawierający atrybuty bazy danych, na przykład domyślne kodowanie znaków i kolejność sortowania. W systemach UNIX właścicielem katalogu bazy danych jest użytkownik używany do uruchamiania serwera. Katalog bazy danych jest dostępny tylko dla tego użytkownika.

Zapytanie `DROP DATABASE` jest zaimplementowane niemal w taki sam prosty sposób. Zapytanie `DROP DATABASE nazwa_bazy_danych` powoduje usunięcie z katalogu danych MySQL podkatalogu o nazwie *nazwa_bazy_danych* wraz ze znajdującymi się w nim tabelami i innymi obiektami bazy danych, na przykład widokami i wyzwalaczami. Niemalże odpowiada to ręcznemu usunięciu katalogu bazy danych za pomocą polecenia poziomu systemu plików, takiego jak `rm` w systemie UNIX lub `del` w Windows. Jednak między zapytaniem `DROP DATABASE` i wymienionymi poleceniami systemu plików istnieją pewne różnice:

- W przypadku zapytania `DROP DATABASE` serwer używa jedynie plików rozpoznanych na podstawie rozszerzeń pliku jako tabele lub inne obiekty bazy danych. Jeżeli w katalogu bazy danych utworzyłeś inne pliki lub podkatalogi, serwer pozostawi je nietknięte. W takim przypadku katalog bazy danych nie może być usunięty, a zapytanie `DROP DATABASE` wygeneruje komunikat błędu. Jedną z konsekwencji jest, że zapytanie `SHOW DATABASES` nadal będzie wyświetlało nazwę tej bazy danych. Rozwiązanie takiego problemu polega na ręcznym usunięciu wszystkich plików w podkatalogu bazy danych, a następnie ponownym wykonaniu zapytania `DROP DATABASE`.
- Nie można bezpiecznie usunąć tabel InnoDB w bazie danych przez usunięcie jej katalogu. Dla każdej tabeli InnoDB silnik InnoDB ma odpowiedni wpis w systemowej przestrzeni tabel, która może również przechowywać zawartość tabeli. Jeżeli baza danych zawiera tabele InnoDB, konieczne jest wykonanie zapytania `DROP DATABASE`, aby silnik InnoDB mógł uaktualnić katalog danych i usunąć całą zawartość tabeli z przestrzeni tabel.

11.2.3. Przedstawienie tabel w systemie plików

MySQL obsługuje wiele silników bazy danych, między innymi InnoDB, MyISAM i MEMORY. Na dysku każda tabela MySQL jest przedstawiana przynajmniej za pomocą jednego pliku: to plik w formacie *.frm*, zawierający opis struktury tabeli. Serwer tworzy plik *.frm*, a poszczególne silniki bazy danych mogą tworzyć dodatkowe pliki zawierające dane rekordów i informacje o indeksach. Nazwy i struktura wspomnianych plików zależą od konkretnego silnika bazy danych.

Poniżej przedstawiono ogólne cechy charakterystyczne pewnych silników bazy danych w zakresie przechowywania plików na dysku. Więcej informacji szczegółowych dotyczących różnic między omawianymi silnikami bazy danych znajdziesz w punkcie 2.6.1, zatytułowanym „Cechy charakterystyczne silników bazy danych”.

Domyślnym silnikiem bazy danych jest InnoDB. W katalogu bazy danych dla każdej tabeli InnoDB znajduje się plik w formacie *.frm*, zawierający definicję struktury tabeli. Z kolei dla danych InnoDB stosuje dwa sposoby ich przedstawienia, oba oparte na przestrzeniach tabel:

- **Systemowa przestrzeń tabel.** Ta przestrzeń tabel składa się z jednego lub więcej ogromnych plików umieszczonych w katalogu danych. Wspomniane pliki przestrzeni tabel tworzą logiczną, ciągłą przestrzeń o wielkości równej sumie wielkości poszczególnych plików. Domyślnie, InnoDB przechowuje tabele w systemowej przestrzeni tabel. Dla tego rodzaju tabel jedynym plikiem charakterystycznym dla tabeli jest plik w formacie *.frm*.
- **Poszczególne przestrzenie tabel.** Istnieje możliwość konfiguracji silnika InnoDB w taki sposób, aby dla każdej tabeli tworzona była oddzielna przestrzeń tabel. W takim przypadku każda tabela InnoDB ma dwa charakterystyczne dla siebie pliki w katalogu bazy danych: plik *.frm* oraz plik *.ibd*, zawierający dane tabeli i jej indeksy.

Systemowa przestrzeń tabel jest używana także w innym celu. InnoDB zawiera wewnętrzny katalog danych przechowujący informacje o wszystkich tabelach. Wspomniany katalog znajduje się w systemowej przestrzeni tabel, która tym samym jest konieczna nawet w przypadku używania poszczególnych przestrzeni tabel do przechowywania zawartości pojedynczych tabel.

Każda tabela MyISAM jest przez MySQL przedstawiana w postaci trzech plików umieszczonych w katalogu bazy danych zawierającej daną tabelę. Nazwa bazowa każdego pliku odpowiada nazwie tabeli, natomiast jego rozszerzenie wskazuje przeznaczenie pliku. Na przykład, tabela MyISAM o nazwie *moja_tabela* ma trzy następujące pliki:

- *moja_tabela.frm* to plik formatu zawierający opis struktury tabeli;
- *moja_tabela.myd* to plik danych przechowujący zawartość rekordów tabeli;
- *moja_tabela.myi* to plik zawierający informacje o indeksach dla wszystkich indeksów utworzonych w danej tabeli.

Tabele typu MEMORY są obszarami w pamięci. Dla tabeli typu MEMORY w katalogu bazy danych znajduje się jedynie plik *.frm*, opisujący jej format. Tego rodzaju tabela nie jest w żaden inny sposób przedstawiana w systemie plików, ponieważ wszystkie dane i indeksy tabeli MEMORY serwer przechowuje w pamięci, a nie na dysku. Po zamknięciu serwera cała zawartość tabel typu MEMORY jest bezpowrotnie tracona. Po ponownym uruchomieniu serwera tabela typu MEMORY nadal istnieje (ponieważ w systemie plików jest jej plik *.frm*), ale pozostaje pusta.

11.2.4. Przedstawienie widoków i wyzwalaczy w systemie plików

Widoki i wyzwalacze są obiektami powiązanymi z plikami w katalogu bazy danych zawierającym te obiekty.

Widok składa się z pliku *.frm* zawierającego definicję widoku oraz inne powiązane z nim atrybuty. Bazowa nazwa pliku widoku odpowiada nazwie widoku. Dlatego też widok *mój_widok* jest przedstawiany przez plik *mój_widok.frm*.

Wyzwalacz jest przechowywany w pliku *.trg*, zawierającym definicję wyzwalacza oraz inne powiązane z nim atrybuty. Plik wyzwalacza ma nazwę bazową odpowiadającą tabeli, do której należy. Na przykład, wyzwalacz o nazwie *mój_wyzwalacz* powiązany z tabelą *moja_tabela* jest przechowywany w pliku *moja_tabela.trg*, a nie *mój_wyzwalacz.trg*. Jeżeli tabela zawiera wiele wyzwalaczy, serwer przechowuje ich definicje w tym samym pliku *.trg*. Ponadto, każdy wyzwalacz ma plik *.trn* o nazwie wyzwalacza i zawierający nazwę tabeli, w której został zdefiniowany. Na przykład, wyzwalacz *mój_wyzwalacz* ma plik *mój_wyzwalacz.trn* zawierający nazwę tabeli *moja_tabela*.

11.2.5. Jak zapytania SQL są mapowane na operacje na pliku tabeli?

Każdy silnik bazy danych używa pliku *.frm* do przechowywania formatu (definicji) tabeli, aby dane wyjściowe zapytania `SHOW TABLES nazwa_bazy_danych` były takie same jak nazwy bazowe plików *.frm* w katalogu bazy danych o wskazanej nazwie.

W celu utworzenia tabeli dowolnego typu obsługiwanego przez MySQL należy wykonać zapytanie `CREATE TABLE` definiujące strukturę tabeli i zawierające klauzulę `ENGINE=nazwa_silnika` wskazującą używany silnik bazy danych. W przypadku pominięcia klauzuli `ENGINE` serwer MySQL użyje domyślnego silnika bazy danych (InnoDB, o ile tego nie zmieniłeś). Serwer tworzy plik *.frm* dla nowej tabeli i umieszcza w nim wewnętrznie zakodowaną definicję tabeli oraz nakazuje odpowiedniemu silnikowi bazy danych utworzenie wszelkich plików powiązanych z tabelą. Na przykład, InnoDB tworzy odpowiedni wpis w katalogu danych oraz inicjalizuje w odpowiedniej przestrzeni tabel informacje o danych i indeksach tabeli. Z kolei silnik MyISAM tworzy pliki danych (*.myd*) i indeksów (*.myi*), natomiast silnik CSV tworzy plik danych w formacie *.csv*. W systemach UNIX właścicielem wszystkich utworzonych w ten sposób plików przedstawiających tabelę jest użytkownik, którego konto jest używane do uruchamiania serwera.

Po wykonaniu zapytania `ALTER TABLE` serwer ponownie koduje plik *.frm* tabeli w celu odzwierciedlenia strukturalnych zmian wprowadzonych przez zapytanie i odpowiednio modyfikuje zawartość tabeli (dane i indeksy). To samo dzieje się również w przypadku zapytań `CREATE INDEX` i `DROP INDEX`, ponieważ są one obsługiwane przez serwer jako odpowiedniki zapytań `ALTER TABLE`. Jeżeli zapytanie `ALTER TABLE` powoduje zmianę silnika bazy danych, zawartość tabeli jest transferowana do nowego silnika, który ponownie zapisuje tę zawartość na dysku, używając odpowiedniego typu plików stosowanych do przedstawienia tabeli.

MySQL implementuje zapytanie `DROP TABLE` przez usunięcie plików przedstawiających tabelę. Jeżeli usuniesz tabelę InnoDB, silnik bazy danych InnoDB uaktualnia także jej katalog danych i oznacza jako wolną całą przestrzeń, która w systemowej przestrzeni tabel InnoDB jest powiązana z usuniętą tabelą.

W przypadku innych silników bazy danych, na przykład MyISAM, istnieje możliwość usunięcia tabeli przez ręczne usunięcie w katalogu bazy danych plików odpowiadających danej tabeli. W silnikach takich jak InnoDB lub MEMORY pewne fragmenty tabeli mogą nie być przedstawiane w plikach charakterystycznych dla danej tabeli, a więc zapytanie `DROP TABLE` nie ma odpowiednika w postaci polecenia systemu plików. Na przykład, tabela InnoDB przechowywana w systemowej przestrzeni tabel zawsze jest unikalnie powiązana z plikiem *.frm*, ale usunięcie wymienionego pliku nie powoduje całkowitego usunięcia danej tabeli. Katalog danych InnoDB musi być uaktualniony przez sam silnik InnoDB, a usunięcie pliku *.frm* pozostawia dane i indeksy tabeli „porzucone” w systemowej przestrzeni tabel.

Jeżeli tabela InnoDB znajduje się w oddzielnej przestrzeni tabel, w katalogu bazy danych jest przedstawiana w postaci plików *.frm* i *.ibd*. Jednak nadal nie można „usunąć” tabeli przez usunięcie wymienionych plików, ponieważ wtedy silnik InnoDB nie ma szansy na uaktualnienie katalogu danych. Konieczne jest użycie zapytania `DROP TABLE`, aby silnik InnoDB mógł usunąć pliki *oraz* uaktualnić katalog danych.

11.2.6. Ograniczenia systemu operacyjnego w zakresie nazw obiektów bazy danych

MySQL ma ogólne reguły dotyczące identyfikatorów dla nazw baz danych oraz innych obiektów, takich jak tabele. Wspomniane reguły zostały omówione w podrozdziale 2.2, zatytułowanym „Identyfikatory składni MySQL i reguły nadawania nazw”, ale poniżej pokrótce przedstawiono ich podsumowanie:

- Niecytowane identyfikatory mogą składać się z liter łacińskich od a do z o dowolnej wielkości, cyfr od 0 do 9, znaków dolara i podkreślenia oraz rozszerzonych znaków Unicode z zakresu od U+0080 do U+FFFF.
- Identyfikatory cytowane za pomocą odwróconych apostrofów mogą zawierać także inne znaki, na przykład ``dziwna?nazwa!``. Cytowanie jest również konieczne, gdy jako identyfikator użyte zostało słowo zarezerwowane SQL. Po włączeniu trybu SQL o nazwie `ANSI_QUOTES` identyfikator może być cytowany za pomocą odwróconych apostrofów lub ujęty w cudzysłów.
- Identyfikator może mieć maksymalnie 64 znaki długości.

Poza wymienionymi regułami system operacyjny, w którym został uruchomiony serwer MySQL, może nakładać inne ograniczenia na identyfikatory. Wynikają one z konwencji nazw w systemie plików, ponieważ nazwy baz danych i tabel odpowiadają nazwom katalogów i plików. Każda baza danych jest przedstawiana w systemie plików za pomocą jej katalogu, natomiast każda tabela, niezależnie od używanego silnika bazy danych, jest reprezentowana w systemie plików przynajmniej przez plik *.frm*. Dlatego też zastosowanie mają następujące ograniczenia:

- MySQL pozwala, aby nazwa bazy danych lub tabeli składała się z maksymalnie 64 znaków, ale długość nazwy może być również ograniczona przez używany system plików.
- Rozróżnianie wielkości plików przez system plików wpływa na sposób nadawania nazw bazom danych i plikom oraz odwoływania się do nich. Jeżeli system plików rozróżnia wielkość liter (jak to zwykle ma miejsce w systemach UNIX), nazwy *abc* i *ABC* odwołują się do różnych plików. Jeśli system plików nie rozróżnia wielkości liter (jak ma to miejsce w systemach Windows i Mac OS X Extended), wtedy nazwy *abc* i *ABC* odwołują się do tego samego pliku. Powinieneś o tym pamiętać, tworząc bazę danych w serwerze rozpoznającym wielkość liter w nazwach plików, ponieważ istnieje prawdopodobieństwo, że będziesz ją przenosił lub replikował do serwera, w którym wielkość liter w nazwach plików nie ma znaczenia.

Serwer koduje w identyfikatorach znaki specjalne, które mogą sprawiać problemy w nazwach plików. Wspomniane kodowanie pozwala na użycie znaków takich jak */* i ** w nazwach występujących w zapytaniach SQL. Dowolny znak spoza zakresu cyfr i liter łacińskich jest mapowany w nazwie pliku na znak *@*, po którym znajduje się zakodowana wartość znaku. Na przykład, znaki *?* i *!* mają kody 003f i 0021, a więc struktura tabeli o nazwie *dziwna?nazwa!* będzie zapisana w pliku *.frm* o nazwie *dziwna@003fnazwa@0021.frm*. Pozostałe pliki powiązane z tabelą będą miały podobne nazwy.

Jak wcześniej wspomniano, rozróżnianie wielkości liter przez system plików wpływa na nazewnictwo baz danych i tabel. Jednym z rozwiązań jest używanie zawsze nazw o określonej wielkości liter. Inne rozwiązanie polega na uruchomieniu serwera wraz z przypisaną wartością 1 zmiennej systemowej *lower_case_table_names*, co ma dwa efekty:

- Serwer konwertuje nazwę tabeli na zapisaną małymi literami i dopiero wtedy tworzy odpowiadające jej pliki na dysku.
- Później, podczas odwoływania się do tabeli w zapytaniu, serwer konwertuje jej nazwę na zapisaną małymi literami i dopiero wtedy próbuje odszukać na dysku plik tabeli.

Wynikiem powyższych działań jest traktowanie nazw jako nierozróżniających wielkości liter, niezależnie od rozróżniania wielkości liter przez system plików. Dzięki temu można łatwiej przenosić bazy danych i tabele między systemami. Jednak jeśli planujesz zastosowanie przedstawionej strategii, konfigurację zmiennej *lower_case_table_names* musisz przeprowadzić *przed* rozpoczęciem tworzenia baz danych i tabel, a nie po.

Jeżeli wymienioną zmienną ustawisz już po utworzeniu baz danych lub tabel zawierających w nazwach wielkie litery, zmienna nie przyniesie oczekiwanego efektu, ponieważ nazwy plików na dysku nie są w całości zapisane za pomocą małych liter. Aby uniknąć wspomnianego problemu, nazwy wszystkich tabel, które zawierają wielkie litery, zmień na całkowicie małe i dopiero wtedy ustaw wartość zmiennej `lower_case_table_names`. (W celu zmiany nazwy tabeli można wykonać zapytanie `ALTER TABLE` lub `RENAME TABLE`). W przypadku dużej liczby tabel wymagających zmiany nazwy lub baz danych zawierających w nazwach wielkie litery łatwiejszym rozwiązaniem jest utworzenie kopii zapasowej zawartości tych baz danych, a następnie ich ponowne utworzenie już po ustawieniu zmiennej `lower_case_table_names`:

1. Za pomocą narzędzia `mysql dump` utwórz kopię zapasową zawartości wszystkich baz danych:

```
% mysqldump --databases nazwa_bazy_danych > nazwa_bazy_danych.sql
```

2. Usuń wszystkie bazy danych, wykonując zapytania `DROP DATABASE`.
3. Zatrzymaj serwer, przekonfiguruj go przez przypisanie wartości 1 zmiennej systemowej `lower_case_table_names`, a następnie ponownie uruchom serwer.
4. Za pomocą klienta `mysql` wczytaj zawartość wszystkich utworzonych wcześniej plików kopii zapasowych:

```
% mysql < nazwa_bazy_danych.sql
```

Po ustawieniu zmiennej `lower_case_table_names` wszystkie bazy danych i tabel zostaną ponownie utworzone i zapisane na dysku w plikach o nazwach składających się z małych liter.

Zmiennej `lower_case_table_names` można przypisać wiele wartości, jak to omówiono w dodatku D, zatytułowanym „Przewodnik po zmiennych systemowych, stanu i użytkownika”.

11.2.7. Czynniki wpływające na maksymalną wielkość tabeli

Wielkość tabeli w MySQL jest ograniczona, ale na to wpływa połączenie wielu czynników. Dlatego też nie zawsze można precyzyjnie wskazać te granice.

System operacyjny narzuca ograniczenie maksymalnej wielkości pliku. Wspomniane ograniczenie ma zastosowanie względem plików przedstawiających tabele, na przykład plików `.myd` i `.myi` dla tabeli MyISAM, a także plików tworzących dowolną przestrzeń tabel InnoDB. Jednak ogólne wymagania w zakresie wielkości systemowej przestrzeni tabel InnoDB mogą bardzo łatwo przekroczyć maksymalną dopuszczalną wielkość pliku. Rozwiązaniem jest skonfigurowanie przestrzeni tabel jako składającej się z wielu plików, z których każdy może osiągnąć wielkość maksymalną. Innym sposobem pokonania ograniczenia w postaci maksymalnej wielkości plików jest wykorzystanie niezmodyfikowanych partycji jako przestrzeni tabel InnoDB. Komponenty przestrzeni tabel znajdujące się na niezmodyfikowanych partycjach mogą osiągnąć rozmiary samej partycji. Informacje

dotyczące konfiguracji tego rodzaju rozwiązania znajdziesz w podpunkcie 12.5.3.1, zatytułowanym „Konfiguracja przestrzeni tabel InnoDB”.

Poza ograniczeniami narzucanymi przez system operacyjny, MySQL ma własne, wewnętrzne ograniczenia wielkości tabel. Wspomniane ograniczenia zależą od używanego silnika bazy danych:

- W przypadku InnoDB maksymalna wielkość systemowej przestrzeni tabel InnoDB wynosi 4 miliardy stron o wielkości 16 KB. Maksymalna wielkość przestrzeni tabel jest związana także z wielkością poszczególnych przechowywanych w niej tabel InnoDB. Jeżeli silnik jest zdefiniowany w sposób powodujący użycie oddzielnych przestrzeni tabel, zawartość poszczególnych tabel InnoDB jest przechowywana w plikach *.ibd*. W takim przypadku na maksymalną wielkość tabeli wpływ mają nakładane przez system operacyjny ograniczenia związane z maksymalną wielkością pliku.
- W przypadku MyISAM wielkość plików *.myd* i *.myi* jest domyślnie ograniczona do 256 TB. Jednak jeśli w trakcie tworzenia tabeli zostaną użyte opcje `AVG_ROW_LENGTH` i `MAX_ROWS`, to wymienione pliki mogą mieć wielkość do 65 536 TB. (Zapoznaj się z opisem zapytania `CREATE TABLE` w dodatku E, zatytułowanym „Przewodnik po składni SQL”). Wymienione opcje wpływają na wielkość wewnętrznego wskaźnika, który określa maksymalną liczbę rekordów, które może przechowywać tabela. Gdy tabela MyISAM osiągnie swoją wielkość maksymalną i zacząć pojawiać się błędy o kodzie 135 i 136 dla operacji na tabeli, użyj zapytania `ALTER TABLE` do zwiększenia wartości wymienionych opcji. Aby bezpośrednio zmienić domyślną wielkość wskaźnika MyISAM, należy ustawić zmienną systemową `mysam_data_pointer_size`. Nowa wartość tej zmiennej ma zastosowanie również dla wcześniej utworzonych tabel.

W przypadku silników bazy danych przedstawiających dane i indeksy w oddzielnych plikach maksymalna wielkość tabeli zostaje osiągnięta, gdy dowolny z tworzących ją plików będzie miał maksymalną dopuszczalną wielkość. Dla tabeli MyISAM cechy charakterystyczne indeksowania wpływają na plik, który pierwszy osiągnie limit. Jeżeli tabela ma niewiele lub w ogóle nie ma indeksów, plik danych prawdopodobnie pierwszy osiągnie wielkość maksymalną. Z kolei w przypadku tabeli posiadającej wiele indeksów plik indeksu może być tym, który jako pierwszy osiągnie wielkość maksymalną.

Obecność kolumny `AUTO_INCREMENT` wyraźnie ogranicza liczbę rekordów, które mogą znaleźć się w tabeli. Na przykład, jeśli wymieniona kolumna została zdefiniowana jako typu `TINY UNSIGNED`, jej maksymalną wartością jest 255 i to jednocześnie jest maksymalna liczba rekordów, które mogą być przechowywane przez tabelę. Większe typy liczb całkowitych pozwalają na przechowywanie większej liczby rekordów. Ogólnie rzecz biorąc, zdefiniowanie w tabeli indeksu `PRIMARY KEY` lub `UNIQUE` ogranicza liczbę rekordów tej tabeli do maksymalnej liczby unikalnych wartości, jakie mogą być przechowywane przez utworzony indeks.

W celu określenia rzeczywistej wielkości tabeli, jaką można uzyskać, pod uwagę trzeba wziąć wiele czynników. Efektywna maksymalna wielkość tabeli będzie prawdopodobnie

wartością najmniejszego z wspomnianych czynników. Przyjmujemy założenie, że chcesz utworzyć tabelę typu MyISAM. MySQL określa maksymalną wielkość plików danych i indeksów na 256 TB każdy przy użyciu wskaźnika o domyślnej wielkości. Jednak jeśli system operacyjny nakłada ograniczenie wielkości pliku do 2 GB, to będzie efektywna wielkość maksymalna dla poszczególnych plików tabeli. Z drugiej strony, jeśli system plików obsługuje pliki o wielkości większej niż 256 TB, czynnikiem decydującym o maksymalnej wielkości tabeli będzie czynnik MySQL, a dokładnie wielkość jego wewnętrznego wskaźnika danych. To jest czynnik, nad którym masz kontrolę.

W związku z przechowywaniem tabel InnoDB w systemowej przestrzeni tabel pojedyncza tabela InnoDB może osiągnąć wielkość przestrzeni tabel, która z kolei może być utworzona z wielu plików, aby zapewnić odpowiednią wielkość do pomieszczenia wszystkich tabel. Jeśli (co jest bardzo prawdopodobne) masz wiele tabel InnoDB, wszystkie współdzielą tę samą przestrzeń i tym samym są ograniczone nie tylko wielkością przestrzeni tabel, ale również miejscem zajmowanym w niej przez inne tabele. Pojedyncza tabela InnoDB może zwiększać swoją wielkość, o ile przestrzeń tabel nie jest zapełniona. Po wykorzystaniu całej przestrzeni tabel żadna tabela nie może się zwiększyć aż do chwili dodania do przestrzeni tabel kolejnego komponentu, który w ten sposób ją powiększy. Alternatywne rozwiązanie polega na zastosowaniu automatycznie rozszerzającego się komponentu przestrzeni tabel. W takim przypadku komponent zwiększa swoją wielkość, dopóki nie osiągnie maksymalnej dozwolonej wielkości pliku w systemie operacyjnym lub nie wykorzysta całej dostępnej pamięci masowej. Informacje dotyczące konfiguracji przestrzeni tabel znajdziesz w podpunkcie 12.5.3.1, zatytułowanym „Konfiguracja przestrzeni tabel InnoDB”.

11.2.8. Wpływ struktury katalogu danych na wydajność systemu

Struktura katalogu danych MySQL jest łatwa do zrozumienia, ponieważ w naturalny sposób wykorzystuje hierarchiczną strukturę systemu plików. Jednocześnie wspomniana struktura wiąże się z pewnymi implikacjami w zakresie wydajności, szczególnie podczas wykonywania operacji otwierania plików przedstawiających tabele bazy danych.

W przypadku silników bazy danych przedstawiających poszczególne tabele w postaci własnych plików każde otworzenie tabeli może wymagać deskryptora pliku. Jeżeli tabela jest przedstawiana za pomocą wielu plików, jej otworzenie wymaga wielu deskryptorów plików, a nie tylko jednego. Serwer w sprytny sposób buforuje deskryptory plików, ale bardzo obciążony serwer może bardzo łatwo wykorzystywać wiele z nich podczas obsługi wielu jednoczesnych połączeń z klientami lub w trakcie wykonywania skomplikowanych zapytań odwołujących się do wielu tabel. To może stanowić poważny problem, ponieważ deskryptory plików szybko wyczerpują zasoby w wielu systemach, przede wszystkim w tych, które domyślnie mają ustawioną niską wartość deskryptorów plików. System operacyjny nakładający niskie ograniczenie na liczbę deskryptorów plików i niezwiększający ich liczby nie jest dobrym kandydatem do uruchamiania w nim bardzo obciążonego serwera MySQL.

Innym efektem przedstawiania każdej tabeli w postaci oddzielnych plików jest wydłużenie czasu otwierania tabeli wraz ze wzrostem liczby tabel. Operacje otwarcia tabeli są mapowane

na dostarczane przez system operacyjny operacje otwarcia plików. Efektywność wymienionych operacji zależy od efektywności procedur systemowych odpowiedzialnych za przeszukiwanie katalogu. Normalnie to nie jest żaden problem, ale sytuacja ulega zmianie, jeśli w bazie danych znajduje się ogromna liczba tabel. Na przykład, tabela MyISAM jest przedstawiana przez trzy pliki. Jeśli wymagane jest użycie 10 000 tabel MyISAM, wtedy katalog bazy danych będzie zawierał 30 000 plików. W przypadku tak dużej liczby plików na pewno zauważysz spowolnienie wynikające z czasu potrzebnego na przeprowadzenie operacji otwierania plików. Jeżeli to Cię martwi, rozważ zastosowanie systemu plików charakteryzującego się wysoką wydajnością podczas obsługi ogromnej ilości plików. Na przykład, XFS lub JFS zapewniają dobrą wydajność działania, nawet w przypadku ogromnej liczby małych plików. Jeśli zastosowanie innego systemu plików nie jest możliwe, wtedy konieczne może być ponowne przemyślenie struktury tabel względem wymagań aplikacji i odpowiednia zmiana tej struktury. Przede wszystkim odpowiedz sobie na pytanie, czy naprawdę konieczne jest używanie tak dużej liczby tabel; czasami aplikacja niepotrzebnie zwiększa ich liczbę. Aplikacja może na przykład tworzyć oddzielną tabelę dla każdego wyniku przeprowadzanej przez użytkownika operacji wyszukiwania wielu tabel, a wszystkie wspomniane tabele wynikowe będą miały identyczne struktury. Jeżeli spróbujesz je połączyć w pojedynczą tabelę, może się to okazać możliwe po dodaniu kolejnej kolumny identyfikującej użytkownika, którego dotyczy dany rekord. Jeśli takie rozwiązanie zmniejszy liczbę tabel, wydajność działania aplikacji niewątpliwie wzrośnie.

W przypadku każdego projektu bazy danych trzeba sprawdzić, czy określona strategia jest warta zastosowania w danej aplikacji. Poniżej wymieniono powody, dla których nie powinno się łączyć tabel w przedstawiony wcześniej sposób:

- Większa ilość wymaganego miejsca na dysku. Połączenie tabel zmniejsza całkowitą liczbę wymaganych tabel (skrócenie czasu otwierania tabel), ale wiąże się z dodaniem kolejnej kolumny (zwiększenie ilości wymaganego miejsca na dysku). To jest typowy kompromis — czas otworzenia pliku tabeli kontra ilość wymaganej pamięci masowej; samodzielnie musisz zdecydować, który z wymienionych czynników jest najważniejszy. Jeżeli ważna jest szybkość, to prawdopodobnie poświęcisz nieco dodatkowego miejsca na dysku twardym. Z kolei jeśli ilość pamięci masowej jest ograniczona, wtedy akceptowalne może być użycie większej liczby tabel i nieco większe opóźnienie w działaniu.
- Kwestie dotyczące bezpieczeństwa. Mogą one wpływać na możliwość łączenia tabel. Jednym z powodów stosowania oddzielnych tabel dla poszczególnych użytkowników jest zapewnienie dostępu do tabeli tylko dla pojedynczego konta MySQL i stosowania tym samym uprawnień na poziomie tabeli. Po połączeniu tabel dane wszystkich użytkowników będą znajdowały się w tej samej tabeli. MySQL nie ma systemu ograniczającego danemu użytkownikowi dostęp do określonych rekordów. Dlatego też połączenie tabel nie będzie mogło odbyć się bez złamania reguł kontroli dostępu. Jednym z możliwych rozwiązań jest użycie widoków pobierających rekordy dla bieżącego użytkownika i nadających uprawnienia dostępu poprzez wspomniane widoki. Ewentualnie, jeśli cały

dostęp do danych jest kontrolowany przez aplikację (użytkownik nigdy nie nawiązuje bezpośredniego połączenia z bazą danych), masz możliwość połączenia tabel i użycia logiki aplikacji w celu wymuszenia stosowania reguł dostępu na poziomie rekordów dla zwróconego wyniku.

Innym sposobem utworzenia wielu tabel bez konieczności tworzenia wielu poszczególnych plików jest wykorzystanie tabel InnoDB i przechowywanie ich w systemowej przestrzeni tabel. W takim przypadku silnik InnoDB z każdą tabelą powiązuje jedynie plik *.frm*, natomiast dane i indeksy wszystkich tabel InnoDB są już przechowywane razem. W ten sposób minimalizuje się liczbę plików na dysku wymaganych do przedstawienia tabel, a tym samym znacznie zmniejsza liczbę deskryptorów plików wymaganych do otwarcia tabel. InnoDB potrzebuje tylko jednego deskryptora dla pliku komponentu przestrzeni tabel (który na dodatek nie ulega zmianie w trakcie istnienia procesu serwera) i krótko mówiąc, deskryptor dla otwieranej przez niego tabeli jest odczytywany wraz z plikiem *.frm* tabeli.

11.2.9. Pliki dzienników zdarzeń i stanu MySQL

Poza podkatalogami baz danych, katalog danych MySQL zawiera także pewną liczbę plików dzienników zdarzeń oraz stanu MySQL, które wymieniono w tabeli 11.1. Domyślnym położeniem wszystkich plików wymienionych w tabeli 11.1 jest katalog danych serwera, a nazwa domyślna wielu z nich pochodzi od nazwy komputera, w którym działa serwer, oznaczonej w tabeli jako *HOSTNAME*. Dzienniki zdarzeń binarne i przekazywania są tworzone w postaci ponumerowanej sekwencji plików, co zostało oznaczone jako *nnnnnn*. W tabeli wymieniono jedynie pliki dzienników zdarzeń i stanu używane na poziomie serwera. Poszczególne silniki bazy danych również mogą tworzyć własne dzienniki zdarzeń oraz inne pliki. Na przykład, takie rozwiązanie jest stosowane przez InnoDB.

Tabela 11.1. Pliki dzienników zdarzeń i stanu MySQL

Typ pliku	Nazwa domyślna	Zawartość pliku
Plik PID	<i>HOSTNAME.pid</i>	Identyfikator procesu serwera.
Dziennik błędów	<i>HOSTNAME.err</i>	Zdarzenia podczas uruchamiania i zamykania serwera oraz ewentualne błędy.
Ogólny dziennik zapytań	<i>HOSTNAME.log</i>	Zdarzenia podczas nawiązywania i zamykania połączenia z serwerem oraz informacje o zapytaniach.
Dziennik binarny	<i>HOSTNAME-bin.nnnnnn</i>	Binarny format zapytań modyfikujących dane.
Indeks dziennika binarnego	<i>HOSTNAME-bin.index</i>	Lista aktualnych nazw plików binarnych dzienników zdarzeń.

Tabela 11.1. Pliki dzienników zdarzeń i stanu MySQL (ciąg dalszy)

Typ pliku	Nazwa domyślna	Zawartość pliku
Dziennik przekazywania	<i>HOSTNAME-relay-bin.nnnnnn</i>	Daty modyfikacji otrzymanych przez serwer podległy z serwera głównego.
Indeks dziennika przekazywania	<i>HOSTNAME-relay-bin.index</i>	Lista aktualnych nazw plików dzienników przekazywania.
Plik informacyjny serwera głównego	<i>master.info</i>	Parametry nawiązania połączenia z serwerem głównym.
Plik informacyjny serwera podległego	<i>relay-log.info</i>	Stan przetwarzania dziennika przekazywania.
Dziennik wolnych zapytań	<i>HOSTNAME-slow.log</i>	Tekst zapytań, których przetworzenie wymaga dużej ilości czasu.

W przypadku dzienników zdarzeń ogólnego i wolno wykonywanych zapytań można zdecydować, czy serwer ma zapisywać zdarzenia w pliku dziennika, w tabeli bazy danych `mysql`, czy w obu wymienionych miejscach. Szczegółowe omówienie rejestracji zdarzeń w tabelach przedstawiono w punkcie 12.8.6, zatytułowanym „Rejestracja zdarzeń w tabelach”.

11.2.9.1. Plik PID

W chwili uruchamiania serwera zapisuje on w pliku PID identyfikator procesu (PID), natomiast podczas zamykania serwera MySQL wspomniany plik jest usuwany. Inne procesy mogą wykorzystać plik PID do ustalenia, czy serwer działa oraz jaki jest jego identyfikator procesu. Na przykład, jeśli w trakcie zamykania systemu operacyjnego uruchamia on skrypt `mysql.server` w celu zamknięcia serwera MySQL, skrypt analizuje plik PID i ustala tym samym proces, któremu trzeba wysłać sygnał zakończenia działania.

Jeżeli serwer nie będzie mógł utworzyć pliku PID, w dzienniku zdarzeń umieści odpowiedni komunikat błędu i będzie kontynuował działanie.

11.2.9.2. Dzienniki zdarzeń MySQL

Serwer MySQL może obsługiwać wiele typów plików dzienników zdarzeń. Większość operacji rejestrowania zdarzeń jest opcjonalna. Opcje podczas uruchamiania serwera można wykorzystać do włączenia jedynie potrzebnych dzienników zdarzeń i nadania im nazw, jeśli nie odpowiadają Ci ich nazwy domyślne. Pamiętaj, że wraz z upływem czasu pliki dzienników zdarzeń mogą osiągnąć ogromne rozmiary. Dlatego bardzo ważne jest uniemożliwienie im zapelnienia systemu plików. Aby zachować pod kontrolą ilość miejsca zajmowanego przez pliki dzienników zdarzeń, co pewien czas należy usuwać stare.

W tym punkcie pokrótce zostaną omówione niektóre pliki dzienników zdarzeń. Więcej informacji na temat dzienników zdarzeń oraz opcji kontrolujących zachowanie serwera w zakresie rejestracji zdarzeń i usuwania starych plików dzienników znajdziesz w podrozdziale 12.8, zatytułowanym „Dzienniki zdarzeń serwera”.

Dziennik zdarzeń błędów zawiera informacje diagnostyczne wygenerowane przez serwer po wystąpieniu zdarzenia wyjątkowego. Jeżeli próba uruchomienia serwera zakończy się niepowodzeniem lub serwer niespodziewanie zakończy działanie, ten dziennik zdarzeń okaże się bardzo użyteczny, ponieważ często zawiera informacje o przyczynach problemów.

Ogólny dziennik zdarzeń zawiera ogólne informacje o działaniu serwera: kto i skąd nawiązał połączenie oraz jakie wykonał zapytania. Binarny dziennik zdarzeń również zawiera informacje o zapytaniach, ale jedynie tych modyfikujących zawartość bazy danych. Ponadto, zawiera informacje takie jak znaczniki czasu wymagane do zapewnienia synchronizacji serwerów podległych z głównym po włączeniu replikacji. Zawartość binarnego dziennika zdarzeń jest zapisywana w formacie binarnym, jako „zdarzenia”, które mogą być wykonane i dostarczyć dane wejściowe dla klienta mysql. Towarzyszący dziennikowi binarnemu plik indeksu zawiera listę plików binarnych dzienników zdarzeń aktualnie używanych przez serwer.

Binarny dziennik zdarzeń jest użyteczny w przypadku wystąpienia awarii i konieczności przywrócenia danych z plików kopii zapasowej, ponieważ wtedy pozwala na odtworzenie zmian wprowadzonych po utworzeniu ostatniej kopii zapasowej. W ten sposób bazy danych zostają przywrócone do stanu, w którym znajdowały się w chwili awarii. Binarny dziennik zdarzeń jest używany także po włączeniu replikacji. W takiej konfiguracji działa w charakterze źródła uaktualnień, które muszą być przekazywane z serwera głównego do podległych. Więcej informacji na ten temat znajdziesz w rozdziale 14., zatytułowanym „Obsługa bazy danych, kopie zapasowe i replikacja”.

Dobrym rozwiązaniem jest zagwarantowanie, że pliki dzienników zdarzeń pozostaną bezpieczne i nie są dostępne dla zwykłych użytkowników, ponieważ mogą zawierać tekst zapytań obejmujących informacje wrażliwe, takie jak hasła. Na przykład, przedstawiony poniżej wpis w dzienniku zdarzeń zawiera hasło dla użytkownika root. Bez wątpienia to jest informacja, której lepiej nie udostępniać innym użytkownikom:

```
080412 16:47:24      44 Query      SET PASSWORD FOR  
                    'root'@'localhost'=PASSWORD('secret')
```

Pliki dzienników zdarzeń są przez serwer domyślnie zapisywane w katalogu danych MySQL. Dlatego też dobrym sposobem zabezpieczenia dzienników zdarzeń jest zabezpieczenie samego katalogu danych, do którego dostęp powinien mieć jedynie użytkownik zajmujący się administracją instalacji MySQL. Szczegółowy opis procedury zabezpieczenia katalogu danych przedstawiono w punkcie 13.1.2, zatytułowanym „Zabezpieczenie instalacji MySQL”.

11.3. Przeniesienie zawartości katalogu danych

We wcześniejszej części rozdziału omówiono strukturę katalogu danych MySQL w jego konfiguracji domyślnej, czyli zawierającego wszystkie bazy danych, pliki dzienników zdarzeń oraz stanu. Jednak istnieje możliwość samodzielnego wskazania miejsca położenia zawartości katalogu danych. Serwer MySQL pozwala na zmianę położenia całego katalogu danych lub

tylko jego wybranych elementów. Istnieje kilka powodów, dla których możesz zdecydować się na tego typu rozwiązanie:

- System plików zawierający katalog danych został zapełniony i konieczne jest jego przeniesienie do systemu plików o większej pojemności.
- Jeżeli katalog danych znajduje się na intensywnie używanym dysku, jego przeniesienie na mniej używany pozwala na zrównoważenie aktywności dyskowej między poszczególnymi urządzeniami fizycznymi. Z tych samych powodów w różnych napędach można umieścić pliki baz danych i dzienników zdarzeń lub rozproszyć bazy danych. Podobnie, systemowa przestrzeń tabel InnoDB pod względem koncepcyjnym jest pojedynczym ogromnym blokiem pamięci masowej, ale jej poszczególne komponenty można umieścić w różnych napędach, poprawiając tym samym wydajność działania. Jeśli używane są tabele partycjonowane, takie samo rozwiązanie można zastosować względem poszczególnych partycji tabeli.
- Umieszczenie baz danych na jednym dysku, a dzienników zdarzeń na innym pomaga w minimalizacji uszkodzeń, które mogą powstać na skutek awarii jednego dysku.

W pozostałej części podrozdziału dowiesz się, które elementy katalogu danych mogą być przeniesione oraz jak przeprowadzać tego rodzaju operacje.

11.3.1. Metody przenoszenia katalogu danych lub jego elementów

Mamy dwa sposoby pozwalające na przeniesienie katalogu danych lub znajdujących się w nim elementów.

Pierwszy polega na tym, że na dowolnej platformie można zdefiniować opcje odczytywane przez serwer w chwili jego uruchamiania. Wspomniane opcje można podać w wierszu poleceń lub pliku opcji. Na przykład, aby wskazać położenie katalogu danych, należy uruchomić serwer wraz opcją `--datadir=nazwa_katalogu` w wierszu poleceń lub umieścić poniższe wiersze w pliku opcji:

```
[mysqld]
datadir=nazwa_katalogu
```

Zazwyczaj grupa opcji dla serwera nosi nazwę `[mysqld]`, jak pokazano w powyższym przykładzie. Jednak w zależności od okoliczności odpowiednie mogą być inne nazwy grup opcji. Na przykład, jeśli uruchamiasz wiele serwerów za pomocą `mysqld_multi`, nazwy grup mają postać `[mysqldn]`, gdzie *n* to liczba całkowita przypisana danemu egzemplarzowi serwera. W punkcie 12.2.3, zatytułowanym „Określanie opcji startowych serwera”, dokładnie przedstawiono, które grupy opcji mają zastosowanie do różnych metod uruchamiania serwera, a także zaprezentowano informacje dotyczące uruchamiania wielu serwerów MySQL.

Drugi sposób bazuje na fakcie, że w systemach UNIX można przenieść plik lub cały katalog, a następnie utworzyć dowiązanie symboliczne w położeniu początkowym prowadzące do nowego położenia.

Żadna z wymienionych metod nie jest uniwersalna i nie działa dla każdego elementu, którego położenie można zmienić. W tabeli 11.2 podsumowano możliwości w zakresie przenoszenia katalogu danych lub jego elementów oraz metody możliwe do zastosowania. Jeśli używasz pliku opcji, to opcje możesz podać w globalnym pliku opcji, takim jak */etc/my.cnf* w systemach UNIX i *C:\my.ini* w Windows.

Tabela 11.2. Podsumowanie dostępnych metod przenoszenia katalogu danych lub jego elementów

Element do przeniesienia	Akceptowalne metody przenoszenia
Cały katalog danych	Opcje startowe lub dowiązanie symboliczne.
Poszczególne podkatalogi baz danych	Dowiązanie symboliczne.
Poszczególne tabele bazy danych	Dowiązanie symboliczne.
Pliki przestrzeni tabel InnoDB	Opcje startowe.
Plik PID	Opcje startowe.
Pliki dzienników zdarzeń	Opcje startowe.

11.3.2. Przygotowania do operacji przeniesienia

Przed podjęciem próby przeniesienia czegokolwiek z katalogu danych MySQL konieczne jest utworzenie jego kopii zapasowej, aby móc przywrócić wymieniony katalog, gdy operacja zakończy się niepowodzeniem. Ponadto, przed przeniesieniem należy zatrzymać serwer MySQL, a po przeprowadzeniu operacji ponownie go uruchomić. W przypadku pewnych rodzajów przeniesienia, na przykład katalogu bazy danych, czasami jest *możliwe* pozostawienie uruchomionego serwera, choć takie rozwiązanie nie jest zalecane. Jeśli się jednak zdecydujesz, to koniecznie upewnij się, że serwer nie uzyskuje dostępu do przenoszonych bazy danych. Przed przeniesieniem bazy danych powinieneś również wykonać zapytanie `FLUSH TABLES`, aby serwer zamknął wszystkie otwarte pliki tabel. Niewykonanie wymienionych zadań może doprowadzić do uszkodzenia tabel.

11.3.3. Uzyskanie dostępu do wyniku przeniesienia

Przed podjęciem próby przeniesienia czegokolwiek z katalogu danych MySQL upewnij się, że operacja przyniesie żądany efekt. Na przykład, w systemie UNIX można użyć poleceń `du`, `df` i `ls -l` w celu sprawdzenia ilości dostępnego miejsca na dysku. Konieczne jest przy tym prawidłowe zrozumienie układu systemu plików, aby wymienione polecenia były użyteczne.

Przedstawiony poniżej przykład pokazuje pułapkę, na którą trzeba uważać podczas przenoszenia katalogu danych MySQL. Przyjmujemy założenie, że katalogiem danych jest `/usr/local/mysql/data` i chcemy go przenieść do katalogu `/var/mysql`, ponieważ wynik działania polecenia `df` pokazuje, że system plików `/var` ma większą ilość wolnej przestrzeni:

```
% df -k /usr /var
Filesystem      1K-blocks      Used Available Use% Mounted on
/dev/sda5        28834716K  24078024K   3291968K   88% /usr
/dev/sda6        28834716K   9175456K  18194536K   34% /var
```

Aby przekonać się, ile przestrzeni w systemie plików `/usr` zostanie zwolnione po przeniesieniu katalogu danych, używamy polecenia `du -s`:

```
% du -s /usr/local/mysql/data
3264308K /usr/local/mysql/data
```

Dane wyjściowe pokazują, że przeniesienie katalogu `data` z `/usr` do `/var` spowoduje zwolnienie około 3 GB miejsca w systemie plików `/usr`. Czy na pewno? Aby się tego dowiedzieć, należy użyć polecenia `df` względem katalogu `data`. Załóżmy, że otrzymujemy następujące dane wyjściowe:

```
% df -k /usr/local/mysql/data
Filesystem      1K-blocks      Used Available Use% Mounted on
/dev/sda6        28834716K   9175456K  18194536K   34% /var
```

To dziwne. Dlaczego narzędzie `df` zgłasza użycie miejsca w systemie plików `/var`? Odpowiedź na to pytanie dostarczają dane wyjściowe polecenia `ls -l`:

```
% ls -l /usr/local/mysql/data
lrwxrwxr-x 1 mysql mysql 10 Dec 11 23:46 data -> /var/mysql
```

Powyższe dane wyjściowe pokazują, że katalog `/usr/local/mysql/data` jest dowiązaniem symbolicznym do `/var/mysql`. Innymi słowy, katalog danych MySQL *już* znajduje się w systemie plików `/var` i został zastąpiony przez dowiązanie symboliczne prowadzące do `/var`. Tak więc przeniesienie katalogu danych do systemu plików `/var` nie spowoduje zwolnienia miejsca w systemie `/usr`.

Jeżeli chcesz przenieść bazę danych do innego systemu plików w celu podjęcia próby redystrybucji zajmowanej przez nią przestrzeni na dysku, musisz pamiętać o jednym: w przypadku używania tabel InnoDB przechowywanych w systemowej przestrzeni tabel zawartość tych tabel *nie* jest przechowywana w katalogu bazy danych. Gdy baza danych składa się głównie z tabel InnoDB, zmiana położenia katalogu bazy danych powoduje przeniesienie jedynie plików `.frm` tabel, a nie ich treści. To ma niewielki wpływ na zajmowane miejsce na dysku.

Z przedstawionych przykładów płynie następujący wniosek. Poświęcenie chwili na sprawdzenie efektu przeniesienia katalogu danych MySQL może zapobiec zmarnowaniu znacznie większej ilości czasu na kopiowanie plików tylko po to, aby się przekonać, że żadanego celu nie udało się osiągnąć.

11.3.4. Przeniesienie całego katalogu danych

W celu przeniesienia całego katalogu danych należy zatrzymać serwer MySQL, a następnie przenieść katalog danych do nowego położenia. Po przeniesieniu trzeba uruchomić serwer wraz z opcją `--datadir`, wyraźnie wskazującą nowe położenie. W systemach UNIX alternatywą dla użycia wymienionej opcji jest utworzenie w starym położeniu katalogu danych dowiązania symbolicznego wskazującego nowe położenie.

11.3.5. Przeniesienie poszczególnych baz danych

Serwer zawsze szuka podkatalogów baz danych w katalogu danych MySQL, a więc jedynym sposobem na przeniesienie bazy danych jest utworzenie dowiązania symbolicznego.

Procedura różni się w systemach UNIX i Windows.

W systemie UNIX przeniesienie bazy danych trzeba przeprowadzić w następujący sposób:

1. Zatrzymanie serwera, jeśli działa.
2. Przeniesienie katalogu bazy danych do nowego położenia, czyli jego skopiowanie do nowej lokalizacji i usunięcie w początkowej.
3. Utworzenie w katalogu danych MySQL dowiązania symbolicznego o nazwie takiej samej jak oryginalna baza danych i prowadzącego do nowej lokalizacji.
4. Ponowne uruchomienie serwera.

Poniższy przykład pokazuje, jak przenieść bazę danych `bigdb` z katalogu `/usr/local/mysql/data` do `/var/db`:

```
% mysqladmin -p -u root shutdown
Enter password: *****
% cd /usr/local/mysql/data
% tar cf - bigdb | (cd /var/db; tar xf -)
% rm -rf bigdb
% ln -s /var/db/bigdb bigdb
% mysql_safe &
```

Powyższe polecenie trzeba wykonać po zalogowaniu się jako administrator MySQL.

W systemie Windows procedura przeniesienia bazy danych jest nieco inna:

1. Zatrzymanie serwera, jeśli działa.
2. Przeniesienie katalogu bazy danych do nowego położenia, czyli jego skopiowanie do nowej lokalizacji i usunięcie w początkowej.
3. Utworzenie w katalogu danych MySQL pliku działającego w charakterze dowiązania symbolicznego, które pozwala serwerowi MySQL na znalezienie przeniesionego katalogu danych. Wspomniany plik powinien mieć rozszerzenie `.sym` i nazwę bazową odpowiadającą nazwie bazy danych. Na przykład, jeśli przenosisz bazę danych `sampdb` z katalogu `C:\mysql\data\sampdb` do `E:\mysql-book\sampdb`, utwórz plik o nazwie `E:\mysql\data\sampdb.sym` i umieść w nim następujący wiersz:

```
E:\mysql-book\sampdb\
```

4. Upewnij się o włączonej obsłudze dowiązania symbolicznego podczas ponownego uruchamiania serwera. Serwery Windows powinny domyślnie mieć włączoną wspomnianą obsługę dowiązań symbolicznych, ale możesz jeszcze użyć opcji `--symbolic-links` w wierszu poleceń lub umieścić poniższe wiersze w pliku opcji:

```
[mysqld]
symbolic-links
```

11.3.6. Przeniesienie poszczególnych tabel

Przeniesienie poszczególnych tabel jest możliwe jedynie w pewnych sytuacjach:

- Trzeba używać systemu UNIX, a przenoszona tabela musi być typu MyISAM.
- System operacyjny musi mieć prawidłowo działające wywołanie `realpath()`. W takim przypadku wynikiem wykonania poniższego zapytania będzie wartość YES:

```
mysql> SHOW VARIABLES LIKE 'have_symlink';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_symlink  | YES   |
+-----+-----+
```

Po spełnieniu obu wymienionych warunków można przenieść pliki *.myd* (dane) i *.myi* (indeksy) do nowego położenia, a następnie utworzyć do nich dowiązania symboliczne w katalogu danych MySQL, zachowując oryginalne nazwy plików danych i indeksów. (Plik *.frm* powinien pozostać w katalogu danych). Przed operacją należy zatrzymać serwer lub nałożyć blokadę na przenoszoną tabelę, aby uniemożliwić serwerowi jej używanie. Informacje na ten temat znajdziesz w podrozdziale 14.2, zatytułowanym „Obsługa bazy danych w działającym serwerze”.

11.3.7. Przeniesienie systemowej przestrzeni tabel InnoDB

Początkowa konfiguracja systemowej przestrzeni tabel InnoDB odbywa się przez umieszczenie w pliku opcji położenia plików komponentów przestrzeni tabel za pomocą zmiennych systemowych `innodb_data_home_dir` i `innodb_data_file_path`. (Więcej informacji na ten temat znajdziesz w podpunkcie 12.5.3.1, zatytułowanym „Konfiguracja przestrzeni tabel InnoDB”). Jeżeli utworzyłeś już przestrzeń tabel, to istnieje możliwość przeniesienia tworzących ją zwykłych plików, na przykład w celu rozmieszczenia plików w różnych systemach plików. Ponieważ położenie plików wskazują zmienne systemowe, przeniesienie kilku lub wszystkich z plików tworzących przestrzeń tabeli można przeprowadzić w następujący sposób:

1. Zatrzymanie serwera, jeśli działa.
2. Przeniesienie pliku lub plików przestrzeni tabel do lokalizacji docelowej.

3. Uaktualnienie pliku opcji zawierającego konfigurację InnoDB w celu odzwierciedlenia nowego położenia przeniesionych plików.
4. Ponowne uruchomienie serwera.

11.3.8. Przeniesienie plików dzienników zdarzeń i stanu

W celu przeniesienia pliku PID lub dziennika zdarzeń należy zatrzymać serwer, a następnie ponownie go uruchomić wraz z odpowiednią opcją wskazującą nowe położenie pliku. Na przykład, jeśli plik PID zostanie utworzony jako */tmp/mysql.pid*, wtedy w wierszu poleceń użyj opcji `--pid_file=/tmp/mysql.pid` lub umieść poniższe wiersze w pliku opcji:

```
[mysqld]  
pid_file=/tmp/mysql.pid
```

Jeżeli nazwa pliku zostanie podana w postaci bezwzględnej ścieżki dostępu, serwer utworzy plik, używając podanej ścieżki dostępu. Jeśli użyto względnej ścieżki dostępu, serwer utworzy plik w katalogu danych MySQL. Na przykład, podanie opcji `--pid_file=mysqld.pid` powoduje utworzenie pliku PID o nazwie *mysqld.pid* w katalogu danych MySQL.

Pewne systemy przechowują pliki PID w określonym katalogu, na przykład */var/run*. W celu zachowania spójności w działającym serwerze plik PID MySQL możesz chcieć umieścić w tym samym katalogu. Podobnie, jeśli system używa katalogu */var/log* do przechowywania plików dzienników zdarzeń, tam możesz umieścić też pliki dzienników zdarzeń MySQL. Jednak wiele systemów pozwala na uzyskanie dostępu do wymienionych katalogów tylko użytkownikowi root. Oznacza to konieczność uruchomienia serwera jako użytkownik root, co ze względów bezpieczeństwa nie jest dobrym pomysłem. Rozwiązaniem może być utworzenie podkatalogów */var/run/mysql* i */var/log/mysql* i ustawienie ich jako własności konta używanego do uruchamiania serwera. Na przykład, jeśli nazwa konta i użytkownika to *mysql*, wtedy jako użytkownik root należy wykonać poniższe polecenia:

```
# mkdir /var/run/mysql  
# chown mysql /var/run/mysql  
# chgrp mysql /var/run/mysql  
# chmod u=rwx,go-rwx /var/run/mysql  
# mkdir /var/log/mysql  
# chown mysql /var/log/mysql  
# chgrp mysql /var/log/mysql  
# chmod u=rwx,go-rwx /var/log/mysql
```

W takim przypadku serwer nie będzie miał żadnych problemów z zapisywaniem plików w nowo utworzonych katalogach. Serwer możesz uruchomić wraz z opcjami wskazującymi pliki w tych nowych katalogach, na przykład:

```
[mysqld]  
pid_file = /var/run/mysql/mysql.pid  
log_error = /var/log/mysql/log.err
```

```
general_log = 1  
general_log_file = /var/log/mysql/querylog  
log-bin = /var/log/mysql/binlog
```

Więcej informacji na temat opcji dotyczących plików dzienników zdarzeń oraz sposobu ich używania znajdziesz w podrozdziale 12.8, zatytułowanym „Dzienniki zdarzeń serwera”.

Ogólna administracja bazą danych MySQL

W tym rozdziale zostaną przedstawione obowiązki, które osoba odpowiadająca za administrację bazą danych MySQL musi wykonywać, aby zapewnić płynne i sprawne działanie bazy danych MySQL używanej przez witrynę internetową:

- Zabezpieczenie bazy danych MySQL po jej instalacji.
- Zapewnienie maksymalnego czasu działania serwera bez przestojów.
- Monitorowanie i ustawianie parametrów serwera.
- Wybór wtyczek służących do włączania funkcji serwera.
- Obsługa plików dzienników zdarzeń serwera.
- Dostrajanie serwera w celu zapewnienia lepszej wydajności jego działania.
- Analiza działania sprzętu i rozwiązywanie jego fizycznych ograniczeń, aby poprawić wydajność działania.
- Uruchamianie wielu serwerów MySQL.
- Określenie, jak i kiedy uaktualnić MySQL do nowszej wersji.

Inne ważne kwestie administracyjne zostaną omówione w rozdziałach 13., „Bezpieczeństwo i kontrola dostępu”, i 14., „Obsługa bazy danych, kopie zapasowe i replikacja”.

W tym rozdziale poznasz kilka programów niezbędnych podczas wykonywania zadań administracyjnych w bazie danych MySQL:

- Serwer MySQL, czyli `mysqld`.
- Programy służące do uruchamiania serwera: `mysqld_safe`, `mysql.server` i `mysqld_multi`.
- Narzędzie administracyjne `mysqladmin`.

Większość informacji przedstawionych w tym rozdziale będzie łatwiejsza do przyswojenia, jeśli rozumiesz strukturę i przeznaczenie katalogu danych MySQL,

miejsce przechowywania przez serwer baz danych, plików dzienników zdarzeń oraz innych informacji. Szczegóły na ten temat znajdziesz w rozdziale 11., zatytułowanym „Katalog danych w MySQL”. Informacje dodatkowe z zakresu omawianych tutaj zapytań SQL i programów znajdziesz w dodatkach: E, „Przewodnik po składni SQL”, i F, „Przewodnik po programie MySQL”.

12.1. Zabezpieczenie nowej instalacji MySQL

Zacznijmy od zadania administracyjnego, które powinno zostać wykonane natychmiast po instalacji MySQL, czyli zagwarantowania, że serwer pozostanie dostępny jedynie dla autoryzowanych użytkowników. To jest kwestia zrozumienia, jakie konta użytkowników MySQL są tworzone przez procedurę instalacyjną, ustawienia haseł dostępu do potrzebnych kont i usunięcia niepotrzebnych kont użytkowników.

Procedura instalacyjna MySQL umieszcza w katalogu danych serwera dwie bazy danych:

- Bazę danych `mysql`, zawierającą tabelę uprawnień, kontrolującą dostęp klientów do serwera.
- Bazę danych `test`, przeznaczoną do użycia w celach testowych.

Jeżeli serwer MySQL zainstalowałeś po raz pierwszy, tabele uprawnień w bazie danych `mysql` zawierają konta w ich domyślnej konfiguracji, czyli umożliwiają każdemu nawiązanie połączenia z bazą danych bez konieczności uwierzytelnienia się. To jest niebezpieczne i dlatego powinieneś przypisać hasła kontom użytkowników. Jeżeli konfigurujesz inną instalację w komputerze zawierającym serwer MySQL zainstalowany w innym położeniu, musisz ustawić hasła w nowym serwerze. Jednak w tym przypadku może wystąpić komplikacja w postaci haseł pobranych z plików opcji utworzonych dla istniejącej instalacji, zgodnie z opisem przedstawionym w punkcie 12.1.2, zatytułowanym „Konfiguracja haseł dla serwerów dodatkowych”.

Poniżej przyjęto założenie, że nie ustawiono żadnych haseł. Pewne instalatory dają możliwość zdefiniowania haseł w trakcie procedury instalacyjnej. Jednak nawet jeśli używasz tego rodzaju instalatora, to przedstawione tutaj informacje pomogą w lepszym zrozumieniu początkowych kont użytkowników MySQL.

12.1.1. Definiowanie haseł dla początkowych kont MySQL

W tym punkcie dowiesz się, jak sprawdzić istnienie kont w tabeli uprawnień oraz jak definiować dla nich hasła. Przyjmujemy założenie, że serwer MySQL działa w komputerze o nazwie `cobra.example.com` i nawiązywane jest połączenie z poziomu tego samego komputera. Kiedy w poleceniach zobaczysz wymienioną nazwę, zastąp ją nazwą używanego serwera. W przykładach przyjęto założenie, że serwer MySQL działa, ponieważ będziemy nawiązywali z nim połączenie.

Procedura instalacyjna MySQL powoduje wstawienie w bazie danych mysql tabeli uprawnień z dwoma rodzajami kont:

- Konta wraz z nazwą użytkownika root. To jest konto superużytkownika, przeznaczone do wykonywania zadań administracyjnych. Użytkownik root ma wszystkie uprawnienia i może zrobić wszystko, łącznie z usunięciem wszystkich baz danych i wyłączeniem serwera. (Zbieżność nazwy konta superużytkownika w systemach UNIX i bazie danych MySQL nie jest przypadkowa. Wspomniane konta root mają wyjątkowe uprawnienia, ale poza tym nic wspólnego ze sobą).
- Konto o pustej nazwie użytkownika. To jest konto „anonimowe”, pozwala użytkownikom na nawiązanie połączenia z serwerem bez konieczności wcześniejszego ustawienia im kont. Użytkownicy anonimowi z reguły mają jedynie niewielkie uprawnienia oraz ograniczony zakres możliwości.

Każde konto znane serwerowi MySQL jest wymienione w tabeli user bazy danych mysql, czyli tabeli zawierającej definicje kont początkowych. Domyślnie żadne z wymienionych kont nie ma przypisanego hasła, ponieważ twórcy bazy danych przyjęli założenie, że samodzielnie zdefiniujesz te hasła. Dlatego też pierwszym zadaniem administracyjnym po instalacji MySQL powinno być zdefiniowanie hasel. W przeciwnym razie nieautoryzowani użytkownicy mogą uzyskać dostęp do bazy danych, na przykład nawiązując połączenie jako użytkownik root.

Po zabezpieczeniu początkowych kont można przystąpić do konfiguracji innych kont w celu umożliwienia członkom społeczności użytkowników nawiązania połączenia z serwerem za pomocą zdefiniowanych kont o uprawnieniach odpowiednich do wykonywanych przez nich zadań. W podrozdziale 13.2, zatytułowanym „Zarządzanie kontami użytkowników w MySQL”, znajdziesz informacje dotyczące tworzenia nowych kont i modyfikacji istniejących.

Każdy rekord w tabeli user zawiera wartości Host i User, wskazujące komputer, z którego użytkownik może nawiązać połączenie, oraz nazwę użytkownika konieczną do podania w trakcie nawiązywania tego połączenia. Każdy rekord zawiera także kolumny Password i plugin, wskazujące sposób uwierzytelniania konta. Jeśli kolumna Password zawiera wartość, to jest ona hasłem dla danego konta (zakodowanym w postaci wartości hash, a więc nie pojawia się jako zwykły tekst). Z kolei kolumna plugin, jeśli zawiera wartość, to wskazuje wtyczkę uwierzytelnienia wywoływaną przez serwer w celu obsługi prób uwierzytelnienia w celu użycia danego konta. Rekord tabeli user zawiera jeszcze wiele innych kolumn wskazujących uprawnienia superużytkownika przypisane do danego konta.

Aby przekonać się, jakie są dostępne konta i czy wymagają uwierzytelniania, nawiąż połączenie z serwerem jako użytkownik root, a następnie wykonaj zapytanie do tabeli user. To powinno być możliwe bez podawania hasła, ponieważ na obecnym etapie użytkownik root nie ma zdefiniowanego hasła:

```
% mysql -u root
mysql> SELECT Host, User, Password, plugin FROM mysql.user;
+-----+-----+-----+-----+
| Host          | User          | Password | plugin |
+-----+-----+-----+-----+
```

localhost	root		
cobra.example.com	root		
127.0.0.1	root		
::1	root		
localhost			
cobra.example.com			

Dane wyjściowe, które zobaczysz w swoim serwerze, nie muszą być dokładnie takie same jak przedstawione powyżej. Jednak dla wszystkich wyświetlonych kont kolumny Password i plugin pozostaną puste, co oznacza brak zabezpieczeń. Jak najszybciej powinieneś więc zdefiniować hasła do kont.

Gdzie w katalogu danych znajdują się tabele uprawnień? To zależy od używanej platformy:

- W systemach UNIX katalog danych jest inicjalizowany przez skrypt `mysql_install_db` w trakcie procedury instalacyjnej. Jednym z zadań wymienionego skryptu jest konfiguracja tabel uprawnień w bazie danych `mysql`. Jeżeli instalujesz MySQL z pakietów RPM w systemie Linux lub za pomocą pakietu DMG w systemie OS X, wówczas skrypt `mysql_install_db` zostanie wywołany automatycznie. W innych przypadkach musisz go uruchomić samodzielnie. Więcej informacji na temat skryptu znajdziesz w podrozdziale F.7, zatytułowanym „Skrypt `mysql_install_db`”.
- W systemie Windows katalog danych i baza danych `mysql` zostały wstępnie zainicjalizowane i są dostarczane wraz z dystrybucją MySQL.

Większość kont początkowych jest taka sama na wszystkich platformach, a tylko kilka jest charakterystycznych dla konkretnej platformy. W przedstawionej poniżej tabeli 12.1. wymieniono konta istniejące na wszystkich platformach. Jak już wcześniej wspomniano, żadne z tych kont początkowo nie ma zdefiniowanego hasła, choć program instalacyjny może zaproponować możliwość ustawienia haseł dla tych kont.

Tabela 12.1. Konta początkowe istniejące na wszystkich platformach

Komputer	Użytkownik	Uprawnienia superużytkownika
localhost	root	Wszystkie
127.0.0.1	root	Wszystkie
::1	root	Wszystkie
localhost		Żadne

Wymienione konta tabeli `user` pozwalają na nawiązanie przez programy klienckie połączeń na następujące sposoby:

- Konto użytkownika `root` pozwala na nawiązanie połączenia z lokalnym serwerem MySQL za pomocą nazwy komputera `localhost`, adresu IPv4

127.0.0.1 lub IPv6 ::1. Na przykład, używając dowolnego z poniższych poleceń, można nawiązać połączenie jako użytkownik root:

```
% mysql -h localhost -u root
% mysql -h 127.0.0.1 -u root
```

Konto wraz z wartością ::1 dla Host pozwala na nawiązywanie połączeń z komputera lokalnego za pomocą protokołu IPv6. Jeżeli Twój system obsługuje IPv6, połączenie możesz nawiązać w następujący sposób:

```
% mysql -h ::1 -u root
```

Jako użytkownik root masz wszystkie uprawnienia i możesz przeprowadzić dowolną operację.

- Konto z pustą wartością kolumny User jest kontem anonimowym. Pozwala na nawiązywanie połączenia z lokalnym serwerem MySQL przy użyciu nazwy komputera localhost i z pominięciem nazwy użytkownika:

```
% mysql -h localhost
```

Użytkownik anonimowy nie posiada uprawnień superużytkownika.

W systemach UNIX, w których skrypt `mysql_install_db` jest uruchamiany do zainicjalizowania tabel uprawnień, istnieją również charakterystyczne dla danego komputera konta root i użytkownika anonimowego. Jeżeli komputer, w którym działa serwer MySQL, nosi nazwę `cobra.example.com`, dostępne są jeszcze konta wymienione w tabeli 12.2.

Tabela 12.2. Konta tworzone dla danego komputera

Komputer	Użytkownik	Uprawnienia superużytkownika
cobra.example.com	root	Wszystkie
cobra.example.com		Żadne

Wymienionych w tabeli 12.2 kont nie zobaczysz w systemie Windows, ponieważ tam tabele uprawnień są wstępnie zainicjalizowane w dystrybucji MySQL, a nazwa Twojego komputera jest przecież nieznana twórcom pakietu.

Jeżeli zalogujesz się w komputerze `cobra.example.com`, konta charakterystyczne dla komputera pozwalają na nawiązanie połączenia z serwerem lokalnym przez podanie nazwy komputera i użytkownika root lub anonimowego:

```
% mysql -h cobra.example.com -u root
% mysql -h cobra.example.com
```

W systemie Windows program instalacyjny może oferować możliwość utworzenia konta root, pozwalającego na nawiązanie połączenia z serwerem z dowolnego miejsca. Dla takiego konta wartościami Host i User będzie znak % (dopasowanie dowolnego komputera) i root, a konto będzie miało wszystkie uprawnienia superużytkownika.

Na wszystkich platformach inna tabela uprawnień (niepokazana tutaj db) zawiera informacje o uprawnieniach pozwalających wszystkim użytkownikom, nawet anonimowym, na użycie bazy danych test oraz każdej posiadającej nazwę rozpoczynającą się od test_.

W pozostałej części punktu przedstawiono sposób konfiguracji haseł dla kont użytkownika root i anonimowego. Przykłady dotyczą reprezentatywnego zestawu kont, *ale konkretne zapytania SQL zależą od kont faktycznie istniejących w używanym systemie*.

W zależności od sposobu definiowania haseł może istnieć potrzeba nakazania serwerowi odświeżenia zawartości tabel uprawnień, aby zmiany zostały zauważone. Serwer przeprowadza kontrolę dostępu za pomocą umieszczonych w pamięci tabel uprawnień. W przypadku niektórych metod zmiany haseł w tabeli user serwer może nie rozpoznać wprowadzonych zmian i dlatego trzeba wyraźnie nakazać mu ponowny odczyt tabel uprawnień.

Jedną z metod definiowania haseł jest nawiązanie połączenia z serwerem jako użytkownik root, ustalenie niezabezpieczonych kont, dla których nie zdefiniowano żadnych informacji o uwierzytelnieniu, a następnie wykonanie zapytania SET PASSWORD dla każdego z wspomnianych kont. Przyjmujemy założenie, że nawiązałeś połączenie z serwerem i odkryłeś, że poniższe konta są niezabezpieczone:

```
% mysql -u root
mysql> SELECT Host, User FROM mysql.user
-> WHERE Password = '' AND plugin = '';

+-----+-----+
| Host          | User |
+-----+-----+
| localhost     | root |
| cobra.example.com | root |
| 127.0.0.1     | root |
| ::1          | root |
| localhost     |      |
| cobra.example.com |      |
+-----+-----+
```

Wymienionym kontom można zdefiniować hasła za pomocą zapytań SET PASSWORD. Każde zapytanie powinno zawierać nazwę konta w postaci '*nazwa_użytkownika*'@'*nazwa_komputera*' z użyciem wartości User i Host modyfikowanego rekordu tabeli user. (Jeżeli kolumna User jest pusta, wówczas wartością *nazwa_użytkownika* jest '', czyli pusty ciąg tekstowy). Aby ustawić hasła dla wymienionych powyżej kont, należy wykonać poniższe zapytania:

```
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('rootpass');
mysql> SET PASSWORD FOR 'root'@'cobra.example.com' = PASSWORD('rootpass');
mysql> SET PASSWORD FOR 'root'@'127.0.0.1' = PASSWORD('rootpass');
mysql> SET PASSWORD FOR 'root'@'::1' = PASSWORD('rootpass');
mysql> SET PASSWORD FOR ''@'localhost' = PASSWORD('anonpass');
mysql> SET PASSWORD FOR ''@'cobra.example.com' = PASSWORD('anonpass');
```

Alternatywą dla zapytania SET PASSWORD jest bezpośrednia modyfikacja tabeli user za pomocą zapytania UPDATE. Ta metoda może zostać użyta do wskazania haseł dla wszystkich kont o danej wartości User, niezależnie od ich wartości Host, a tym samym jednoczesnego zmodyfikowania wielu kont. Aby ustawić hasło dla wszystkich kont użytkownika root i anonimowych, użyj poniższych zapytań:

```
mysql> UPDATE mysql.user SET Password=PASSWORD('rootpass') WHERE User='root';
mysql> UPDATE mysql.user SET Password=PASSWORD('anonpass') WHERE User='';
mysql> FLUSH PRIVILEGES;
```

Kiedy wykonujesz zapytania SET PASSWORD w celu zmiany haseł, serwer zauważa modyfikację tabel uprawnień i automatycznie je ponownie odczytuje, aby odświeżyć kopie tabel umieszczone w pamięci. Jeżeli użyjesz zapytania UPDATE w celu bezpośredniej modyfikacji tabeli user, konieczne jest wyraźne nakazanie serwerowi ponownego wczytania tabel uprawnień. Do tego celu służy zapytanie FLUSH PRIVILEGES wykonane natychmiast po zapytaniach UPDATE.

Innym rozwiązaniem dotyczącym kont anonimowych jest ich całkowite usunięcie. Zalecam właśnie takie rozwiązanie, o ile nie istnieje konkretny powód do ich zachowania. W celu usunięcia kont anonimowych należy wykonać poniższe zapytania DROP USER:

```
mysql> DROP USER ''@'localhost';
mysql> DROP USER ''@'cobra.example.com';
```

W przypadku zapytania DROP USER serwer automatycznie ponownie odczytuje tabelę uprawnień i nie ma konieczności wykonywania zapytania FLUSH PRIVILEGES.

Jedną z zalet usunięcia kont użytkowników anonimowych jest zwiększenie poziomu bezpieczeństwa. Kolejna to znaczne uproszczenie zadania konfiguracji kont innych niż anonimowe. Jeżeli pozostawisz konta anonimowe, wówczas będziesz musiał zmierzyć się z ciekawym fenomenem, omówionym w punkcie 13.4.4, zatytułowanym „Puzzle uprawnień”.

Po zdefiniowaniu haseł do kont (i ponownym wczytaniu tabeli uprawnień, o ile występuje konieczność) podczas nawiązywania połączenia z serwerem konieczne będzie podanie poprawnego hasła. W szczególności, bez podania hasła nie będzie można nawiązać połączenia jako użytkownik root:

```
% mysql -u root
ERROR 1045 (28000): Access denied for user 'root'@'localhost'
(using password: NO)
% mysql -p -u root
Enter password: rootpass
mysql>
```

Konieczność podania hasła podczas nawiązywania połączenia z serwerem dotyczy nie tylko klienta mysql, ale także innych programów, takich jak mysqladmin i mysql dump. W celu zachowania zwięzłości i czytelności wiele przykładów przedstawionych w dalszej części rozdziału nie pokazuje opcji -u i -p. Przyjęto założenie, że podasz je, nawiązując połączenie z serwerem.

12.1.2. Konfiguracja haseł dla serwerów dodatkowych

Przedstawione wcześniej instrukcje dotyczyły definiowania haseł w systemie, w którym wcześniej nie była zainstalowana baza danych MySQL. Jednak jeśli serwer MySQL jest już zainstalowany w jednym położeniu, a Ty chcesz zdefiniować hasła dla nowego serwera

zainstalowanego w innym położeniu w tym samym komputerze, to może okazać się, że próba nawiązania połączenia z nowym serwerem bez podania hasła kończy się niepowodzeniem i wygenerowaniem następującego komunikatu błędu:

```
% mysql -u root
ERROR 1045 (28000): Access denied for user 'root'@'localhost'
(using password: YES)
```

Dlaczego serwer twierdzi, że otrzymał hasło, skoro żadnego nie podano? To z reguły oznacza istnienie pliku opcji zawierającego hasło umożliwiające uzyskanie dostępu do poprzednio zainstalowanego serwera. Klient mysql odszukuje plik opcji i automatycznie używa wymienionego tam hasła. Aby zawiesić to zachowanie i wyraźnie nakazać uruchomienie serwera bez hasła, trzeba użyć opcji `--skip-password`:

```
% mysql -u root --skip-password
```

Przedstawioną strategię można zastosować także w innych programach klienckich MySQL, na przykład w `mysqladmin`.

Informacje dodatkowe dotyczące używania wielu serwerów znajdziesz w podrozdziale 12.9, zatytułowanym „Uruchamianie wielu serwerów MySQL”.

12.2. Konfiguracja uruchamiania i zamykania serwera MySQL

Jednym z zadań administratora MySQL jest zagwarantowanie, by serwer (`mysqld`) działał jak najdłużej bez przestojów, aby użytkownicy mogli uzyskiwać do niego dostęp. Jednak czasami występuje konieczność zatrzymania serwera. Na przykład, podczas przeprowadzania operacji przeniesienia katalogu danych nie można pozwolić, aby serwer uaktualniał tabele, i dlatego musi być zatrzymany. Ta książka nie rozwiąże za Ciebie problemu pogodzenia dwóch sprzecznych potrzeb — zapewnienia jak najdłuższego działania serwera bez przestojów oraz chwilowego zatrzymania serwera w celu przeprowadzenia pewnych zadań administracyjnych. Dowiesz się jednak, jak uruchamiać i zatrzymywać serwer, aby mieć możliwość przeprowadzenia wspomnianych zadań. Wiele aspektów procedury różni się w systemach UNIX i Windows, a więc zostaną one omówione oddzielnie.

12.2.1. Uruchamianie serwera MySQL w systemach UNIX

W systemach UNIX serwer MySQL może być uruchomiony ręcznie z poziomu wiersza poleceń. Istnieje możliwość konfiguracji automatycznego uruchamiania serwera wraz z systemem jako części standardowej procedury uruchamiania systemu. (Prawdopodobnie w taki sposób będziesz najczęściej uruchamiał serwer po przeprowadzeniu całej jego konfiguracji). Zanim jednak przejdziemy do zagadnienia uruchamiania serwera, warto zastanowić się, które konto użytkownika powinno być użyte do jego uruchomienia.

W systemie wielodostępnym, takimi jak UNIX, masz możliwość użycia konta logowania do uruchomienia serwera. Jeżeli ręcznie uruchamiasz serwer, będzie on działał w ramach konta użytkownika systemu UNIX, którego użyłeś do zalogowania się do systemu. Na przykład, jeśli zaloguję się jako robert i uruchomię serwer, wtedy będzie on działał w ramach konta użytkownika robert. Natomiast jeśli użyję polecenia su w celu przełączenia się do konta użytkownika root, a następnie uruchomię serwer, będzie on działał w ramach konta root.

Podczas uruchamiania serwera MySQL w systemie UNIX pamiętaj o dwóch kwestiach:

- **Serwer powinien być uruchomiony przez użytkownika innego niż root.** Aby powiedzieć, że serwer działa „jako” dany użytkownik, proces serwera powinien być powiązany z identyfikatorem konta logowania w systemie UNIX tego użytkownika oraz mieć jego uprawnienia w zakresie odczytu i zapisu plików w systemie plików. To, oczywiście, wiąże się z pewnymi implikacjami z zakresu bezpieczeństwa, szczególnie dla procesów uruchamianych przez użytkownika root, ponieważ wymieniony użytkownik może zrobić w systemie wszystko, co oznacza pewne ryzyko. Jednym ze sposobów uniknięcia niebezpieczeństwa jest odebranie serwerowi jego specjalnych uprawnień. Procesy uruchamiane przez użytkownika root mają możliwość zmiany identyfikatora użytkownika na konto innego użytkownika, a tym samym zrzeczenia się uprawnień root i przyjęcia w zamian uprawnień zwykłego użytkownika. W takim przypadku proces jest mniej niebezpieczny. Ogólnie rzecz biorąc, należy ograniczać możliwości procesów, o ile naprawdę nie potrzebują one uprawnień użytkownika root — mysqld na pewno ich nie potrzebuje. Serwer musi mieć możliwość uzyskania dostępu do katalogu danych MySQL oraz zarządzania jego zawartością i niewiele poza tym. Oznacza to, że po uruchomieniu serwera przez użytkownika root należy w trakcie uruchamiania nakazać mu zmianę użytkownika na nieuprzywilejowanego.
- **Za każdym razem serwer powinien być uruchamiany w ramach tego samego użytkownika.** W przypadku uruchamiania serwera za każdym razem przez innego użytkownika wraz z jego uprawnieniami dojdzie do niespójności. Oznacza to, że pliki i katalogi tworzone w katalogu danych MySQL będą miały różnych właścicieli. Może się więc zdarzyć, że serwer nie będzie miał dostępu do pewnych baz danych lub tabel, w zależności od tego, w ramach jakiego użytkownika został uruchomiony. Uniknięcie tego problemu jest możliwe przez uruchamianie serwera za każdym razem przez tego samego użytkownika.

12.2.1.1. Działanie serwera w ramach pozbawionego uprawnień konta logowania

Istnieje wiele zalet użycia oddzielnego, pozbawionego szczególnych uprawnień konta użytkownika zamiast konta użytkownika root do wykonywania zadań powiązanych z MySQL:

- Nie będzie można wykorzystać serwera jako luki w zabezpieczeniach pozwalającej na uzyskanie uprawnień użytkownika root.

- Pliki tworzone przez serwer będą własnością nieuprzywilejowanego użytkownika, a nie użytkownika root. Na przykład, użytkownicy MySQL z uprawnieniem FILE nie mogą zmusić serwera do zapisu plików należących do użytkownika root. Im mniej tego rodzaju plików w serwerze, tym lepiej.
- Znacznie bezpieczniejsze jest przeprowadzanie zadań administracyjnych MySQL po zalogowaniu się jako użytkownik nieuprzywilejowany zamiast użytkownik root. Jeśli popełnisz błąd podczas przeprowadzania operacji na systemie plików, pracując jako użytkownik root, jego konsekwencje mogą być druzgocące.
- Pod względem koncepcyjnym i administracyjnym znacznie lepiej jest utworzyć oddzielne konto przeznaczone wyłącznie dla działalności związanej z MySQL. Dzięki temu znacznie łatwiej będzie sprawdzić, co w systemie jest powiązane z MySQL. Na przykład, w katalogu zawierającym pliki *crontab* znajdzie się oddzielny plik dla użytkownika MySQL. W przeciwnym razie zadania cron MySQL będą wymieniane w pliku *crontab* użytkownika root i wszystko będzie wykonywane jako użytkownik root.

Aby skonfigurować `mysqld` do działania w ramach konta pozbawionego uprawnień należy wykonać przedstawioną poniżej procedurę:

1. Jeżeli serwer działa, zatrzymaj go, wydając poniższe polecenie:

```
% mysqldadmin -p -u root shutdown
```

2. Wybierz konto użytkownika, w ramach którego będzie uruchomiony serwer `mysqld`. Istnieje również możliwość utworzenia grupy przeznaczonej specjalnie do użycia z MySQL. W omawianym przypadku użyjemy `mysql` jako nazw zarówno użytkownika, jak i grupy. Jeżeli zdecydujesz się na inne nazwy, zastosuj je w miejsce używanych tutaj `mysql`. Na przykład, możesz zainstalować MySQL w ramach swojego konta użytkownika, ponieważ nie ma ono żadnych specjalnych uprawnień administracyjnych w systemie. W takim przypadku serwer również możesz uruchamiać w ramach swojego konta użytkownika, a więc odpowiednio zastąp stosowaną tutaj nazwę użytkownika i grupy.
3. Jeśli jest to konieczne, utwórz konto logowania o wybranej nazwie użytkownika. W tym celu wykorzystaj standardową procedurę tworzenia konta. Będziesz potrzebował uprawnień użytkownika root.
Jeżeli zdecydujesz się na użycie konta o nazwie `mysql` do uruchamiania serwera, jego ręczne utworzenie nie musi być konieczne. W przypadku instalacji MySQL w systemie Linux za pomocą pakietu RPM procedura instalacyjna automatycznie tworzy odpowiednie konto. Podobnie się dzieje w systemie OS X. W innych systemach konto użytkownika `mysql` również może być automatycznie utworzone.
4. Zmodyfikuj użytkownika i grupę katalogu danych MySQL i wszystkich znajdujących się tam podkatalogów oraz plików. Właścicielem wymienionego katalogu powinien być użytkownik `mysql`. Na przykład, jeśli katalogiem danych MySQL jest `/usr/local/mysql/data`, zmiana jego właściciela następuje po wydaniu poniższych poleceń.

```
# chown -R mysql /usr/local/mysql/data
# chgrp -R mysql /usr/local/mysql/data
```

Wymienione polecenia należy wykonać jako użytkownik root.

5. Dobrym rozwiązaniem jest ustawienie takiego trybu dostępu do katalogu danych MySQL, aby uniemożliwić dostęp do niego innym użytkownikom. W tym celu należy zmodyfikować uprawnienia, nadając dostęp jedynie użytkownikowi mysql. Jeżeli katalogiem danych jest `/usr/local/mysql/data`, przedstawione poniżej polecenie powoduje, że cała jego zawartość będzie dostępna jedynie dla użytkownika mysql i niedostępna dla grupy oraz pozostałych użytkowników:

```
# chmod -R go-rwx /usr/local/mysql/data
```

Ostatnich kilka kroków w rzeczywistości jest częścią nieco bardziej skomplikowanej procedury, która szczegółowo będzie omówiona w punkcie 13.1.2, zatytułowanym „Zabezpieczenie instalacji MySQL”. W wymienionym punkcie znajdziesz informacje dodatkowe dotyczące konfiguracji własności i uprawnień, zwłaszcza w przypadku niestandardowej instalacji MySQL.

Po zakończeniu wspomnianej procedury należy się upewnić, aby zawsze uruchamiać serwer wraz z opcją `--user=mysql`, wskazującą na konieczność przełączenia użytkownika na mysql, jeśli serwer został uruchomiony przez użytkownika root. Dotyczy to zarówno ręcznego uruchomienia serwera przez użytkownika root, jak i konfiguracji serwera w celu jego uruchamiania wraz z systemem. W systemach UNIX procedura uruchomienia systemu jest przeprowadzana przez użytkownika root, a więc wszystkie zainicjalizowane w jej trakcie procesy również działają z uprawnieniami użytkownika root.

Najlepszym sposobem zagwarantowania uruchamiania serwera przez tego samego użytkownika za każdym razem jest jego podanie w pliku opcji odczytywanym przez serwer. Na przykład, w pliku `/etc/my.cnf` umieść poniższe wiersze:

```
[mysqld]
user=mysql
```

Więcej informacji na temat plików opcji znajdziesz w punkcie 12.2.3, zatytułowanym „Określanie opcji startowych serwera”.

Jeżeli uruchomienie serwera nastąpi w chwili, gdy jesteś zalogowany jako użytkownik mysql, obecność wiersza `user=mysql` w pliku opcji spowoduje wyświetlenie ostrzeżenia informującego, że wymieniona opcja może być używana jedynie przez użytkownika root. Oznacza to, że serwer nie ma możliwości zmiany identyfikatora użytkownika i zostanie uruchomiony w ramach użytkownika mysql. Ponieważ tego oczekujemy, możemy po prostu zignorować to ostrzeżenie.

12.2.1.2. Uruchomienie serwera w systemach UNIX

Po wybraniu konta logowania używanego do uruchomienia serwera MySQL mamy kilka możliwości uruchomienia serwera. Serwer można więc uruchomić ręcznie z poziomu wiersza poleceń lub automatycznie w trakcie procedury uruchamiania systemu. Poniżej wymieniono dostępne metody:

- **Bezpośrednie wywołanie** `mysqld`. To prawdopodobnie najrzadziej stosowana metoda. Nie będzie tutaj omawiana poza przedstawieniem polecenia, które jest użyteczne do sprawdzenia opcji startowych obsługiwanych przez serwer:

```
% mysqld --verbose --help
```

- **Wywołanie skryptu** `mysqld_safe`. Skrypt `mysqld_safe` powoduje uruchomienie serwera, a następnie monitoruje go i ponownie uruchamia, jeśli działanie serwera niespodziewanie się zakończy. Skrypt `mysqld_safe` jest powszechnie stosowany w systemach UNIX stylu BSD oraz wywoływany przez skrypt `mysql.server` w systemach innych niż BSD, na przykład OS X.

Komunikaty błędów oraz inne diagnostyczne dane wyjściowe serwera skrypt `mysqld_safe` przekierowuje do pliku w katalogu danych MySQL w celu wygenerowania dziennika błędów lub do dziennika zdarzeń `syslog`. Jeżeli dane wyjściowe błędów zostaną umieszczone w pliku, skrypt `mysqld_safe` definiuje właściciela pliku jako konto użytkownika wskazane w opcji `--user`. To prowadzi do problemów, jeśli stosujesz różne wartości wymienionej opcji. Symptomatic wskazującym na niepowodzenie operacji zapisu danych do dziennika błędów jest komunikat o braku uprawnień. To, oczywiście, problematyczne, ponieważ jeśli zajrzysz do dziennika błędów w celu sprawdzenia przyczyny problemów, to przekonasz się, że nie zawiera on żadnych informacji o przyczynie problemów! W przypadku wystąpienia tego rodzaju problemu po prostu usuń dziennik błędów i ponownie wywołaj skrypt `mysqld_safe`.

- **Wywołanie skryptu** `mysql.server`. Skrypt `mysql.server` powoduje uruchomienie serwera przez wywołanie skryptu `mysqld_safe`. Wywołanie skryptu z argumentem `start` lub `stop` powoduje odpowiednie uruchomienie lub zatrzymanie serwera. Ten skrypt działa w charakterze opakowania dla `mysqld_safe` i bardzo często jest używany w systemach wykorzystujących metody Systemu V umieszczenia skryptów uruchamiania i zatrzymywania systemu w wielu katalogach. Każdy katalog odpowiada określonemu poziomowi działania i zawiera skrypty wykonywane podczas przejścia lub opuszczenia danego poziomu działania.
- **W celu koordynacji wielu serwerów należy użyć skryptu** `mysqld_multi`. Wymieniony skrypt odczytuje plik opcji zawierający listę parametrów startowych dla wielu serwerów. Pozwala na uruchamianie lub zatrzymywanie dowolnego z nich lub sprawdzenie, czy serwer jest uruchomiony. Ten skrypt jest znacznie bardziej skomplikowany od pozostałych i dlatego jego omówienie odkładamy aż do podrozdziału 12.9, zatytułowanego „Uruchamianie wielu serwerów MySQL”.

Skrypty `mysqld_safe` i `mysqld_multi` są instalowane w podkatalogu `bin` katalogu instalacji MySQL, natomiast skrypt `mysql.server` znajdziesz w katalogu `support-files`. Aby użyć skryptu `mysql.server`, może być konieczne jego skopiowanie do katalogu odpowiedniego poziomu działania i nadanie uprawnień do uruchamiania. (Pewne procedury instalacyjne automatycznie umieszczają skrypt `mysql.server` w odpowiednich katalogach. Tak się dzieje na przykład podczas instalacji MySQL z pakietu RPM w systemie Linux i DMG w systemie OS X).

Uwaga

Zwykle skrypt `mysql.server` jest instalowany pod nazwą `mysqld` w katalogu odpowiedniego poziomu działania. Jednak tutaj będziemy kontynuowali jego omawianie z użyciem nazwy `mysql.server`, aby było jasne, do czego się odwołujemy. Jeżeli używasz pakietu RPM serwera MySQL pobranego od innego producenta, podobny skrypt startowy może być zainstalowany pod inną nazwą, na przykład `mysqld`.

Typ używanego systemu wpływa na operacje, które trzeba przeprowadzić, aby skrypt startowy był uruchamiany wraz z systemem. Zapoznaj się z kolejnymi przykładami, a następnie wykorzystaj lub zaadaptuj przedstawione wskazówki, które najbardziej odpowiadają procedurze startowej stosowanej przez Twój system.

W przypadku systemów w stylu BSD powszechna praktyka polega na istnieniu niewielu plików w katalogu `/etc`, które uruchamiają usługi wraz z systemem. Nazwy wspomnianych plików rozpoczynają się od `rc`. Wśród nich niemal na pewno znajdziesz `rc.local` (lub podobną), przeznaczoną specjalnie do uruchamiania lokalnie zainstalowanych usług. Dlatego też w systemie opartym na `rc` możesz dodać do `rc.local` poniższe wiersze powodujące uruchomienie serwera:

```
if [ -x /usr/local/bin/mysqld_safe ]; then
    /usr/local/bin/mysqld_safe &
fi
```

Jeżeli w Twoim systemie ścieżka dostępu do `mysqld_safe` jest inna, to odpowiednio zmodyfikuj powyższe wiersze.

W przypadku systemów w stylu System V możesz zainstalować skrypt `mysql.server`. Skopiuj go do znajdującego się w katalogu `/etc` odpowiedniego podkatalogu poziomu działania. To zostało już zrobione, jeśli używasz systemu Linux i zainstalowałeś MySQL z pakietu RPM. W innych przypadkach umieść skrypt w głównym katalogu skryptów startowych pod dowolną nazwą, upewnij się, że nadałeś mu uprawnienia do uruchamiania, a następnie umieść odniesienia do niego w katalogu odpowiedniego poziomu działania.

Układ katalogów poziomu działania zależy od konkretnego systemu, więc musisz to sprawdzić samodzielnie. Na przykład, wiele dystrybucji systemu Linux posiada zestaw katalogów zorganizowanych w `/etc/init.d` i `/etc/rc.d`. Tego rodzaju dystrybucje zwykle mają polecenie `chkconfig`, przeznaczone do zarządzania skryptami startowymi. Wymienione polecenie możesz wykorzystać do pomocy w instalacji skryptu `mysql.server`. Przedstawiona poniżej procedura pokazuje instalację skryptu `mysql.server` w katalogach startowych pod nazwą `mysqld`:

1. Skopiuj skrypt `mysql.server` z jego obecnego położenia do katalogu `init.d` oraz nadaj skryptowi uprawnienia do wykonywania:

```
# cp mysql.server /etc/init.d/mysqld
# chmod +x /etc/init.d/mysqld
```

2. Zarejestruj skrypt i włącz go:

```
# chkconfig --add mysqld
# chkconfig mysqld on
```

3. Aby upewnić się o prawidłowym włączeniu skryptu, uruchom polecenie `chkconfig` wraz z opcją `--list`:

```
# chkconfig --list mysql
mysql          0:off  1:off  2:on   3:on   4:on   5:on   6:off
```

Dane wyjściowe polecenia wskazują, że skrypt jest wykonywany automatycznie na poziomach działania 2, 3, 4 i 5.

Jeżeli polecenie `chkconfig` jest niedostępne, możesz użyć ręcznej procedury. Aby włączyć skrypt na trzecim poziomie działania, wykonaj przedstawione poniżej polecenia:

```
# cp mysql.server /etc/init.d/mysql
# cd /etc/init.d
# chmod +x mysql
# cd /etc/rc.d/rc3.d
# ln -s /etc/init.d/mysql S99mysql
```

W systemie OS X katalogi */Library/StartupItems* i */System/Library/StartupItems* zawierają podkatalogi przeznaczone dla usług inicjalizowanych w trakcie uruchamiania systemu. Znajdujący się na witrynie MySQL pakiet DMG przeznaczony dla systemu OS X zawiera instalator, który w jednym z wymienionych katalogów umieszcza element startowy serwera MySQL.

12.2.2. Uruchamianie serwera MySQL w systemach Windows

W systemie Windows serwer MySQL można uruchomić ręcznie z poziomu wiersza poleceń. Istnieje również możliwość utworzenia usługi Windows dla MySQL. Wspomniana usługa może być uruchamiana automatycznie wraz z Windows i kontrolowana z poziomu wiersza poleceń lub menedżera usług Windows.

Dystrybucje MySQL dla Windows zawierają wiele serwerów, a każdy zbudowany z różnymi opcjami. W przykładach wykorzystujemy serwer o nazwie `mysqld`. Używana przez Ciebie dystrybucja może zawierać także serwer o nazwie `mysqld-debug`, który jest jak `mysqld`, ale zapewnia obsługę mechanizmu usuwania błędów. Jeżeli potrzebujesz dostępu do mechanizmu usuwania błędów, wybieraj `mysqld`, ponieważ `mysqld-debug` zużywa większą ilość pamięci i działa nieco wolniej.

Windows oferuje dwa rodzaje połączeń niedostępnych w systemach UNIX:

- Połączenia za pomocą nazwanych potoków, jeśli serwer został uruchomiony wraz z opcją `--named-pipe`.
- Połączenia wykorzystujące pamięć współdzieloną, jeśli serwer został uruchomiony wraz z opcją `--shared-memory`.

Więcej informacji na temat włączania wymienionych typów połączeń znajdziesz w punkcie 12.2.4, zatytułowanym „W jaki sposób serwer nasłuchuje połączeń?”.

12.2.2.1. Ręczne uruchomienie serwera w Windows

W celu ręcznego uruchomienia serwera z poziomu wiersza poleceń należy przejść do katalogu zawierającego instalację serwera i wydać poniższe polecenie:

```
C:\> mysql
```

Istnieje możliwość podania opcji w wierszu poleceń lub w pliku opcji. Zapoznaj się z punktem 12.2.3, zatytułowanym „Określanie opcji startowych serwera”.

Aby komunikaty błędów były wyświetlane w oknie konsoli, a nie umieszczane w pliku dziennika błędów (domyślnie to plik *HOSTNAME.err* w katalogu danych MySQL), należy użyć opcji `--console`:

```
C:\> mysql --console
```

Po uruchomieniu serwera w trybie konsoli można podać inne opcje wiersza poleceń po opcji `--console` lub w pliku opcji.

Jeżeli serwer zostanie uruchomiony z poziomu wiersza poleceń, aż do jego zamknięcia możesz nie zobaczyć innego znaku zachęty. To jest jak najbardziej prawidłowe. Po prostu otwórz inne okno konsoli, a otrzymasz możliwość użycia programów klienckich.

W celu zatrzymania serwera należy użyć narzędzia `mysqladmin`:

```
C:\> mysqladmin -p -u root shutdown
```

12.2.2.2. Uruchomienie serwera jako usługi Windows

W systemie Windows za pomocą poniższego polecenia można utworzyć usługę dla Windows:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysql" --install
```

Powyższe polecenie wykorzystuje pełną ścieżkę dostępu do serwera. Jeżeli serwer został zainstalowany w innej lokalizacji, musisz odpowiednio zmodyfikować ścieżkę dostępu.

Polecenie tworzące usługę nie powoduje uruchomienia `mysqld`. Zamiast tego powoduje, że serwer `mysqld` jest uruchamiany automatycznie wraz z systemem Windows. Jeżeli preferujesz utworzenie usługi, która nie jest uruchamiana automatycznie, zainstaluj serwer jako usługę „ręczną”:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysql" --install-manual
```

Ogólna reguła jest następująca: po utworzeniu usługi Windows dla MySQL nie podajesz innych opcji w wierszu poleceń, ale wymieniasz je w pliku opcji (zapoznaj się z punktem 12.2.3, zatytułowanym „Określanie opcji startowych serwera”). Jednak istnieje możliwość podania nazwy usługi i pliku opcji jako argumentów, co zostanie wkrótce przedstawione. Takie rozwiązanie jest szczególnie użyteczne w trakcie tworzenia wielu usług MySQL. (Zapoznaj się również z podrozdziałem 12.9, zatytułowanym „Uruchamianie wielu serwerów MySQL”).

Kiedy tworzysz usługę Windows dla MySQL, domyślną nazwą usługi jest MySQL (wielkość liter nie ma znaczenia). Za pomocą opcji `--install` można podać własną nazwę dla usługi:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqld" --install nazwa_uslugi
```

Każda usługa Windows musi mieć unikalną nazwę, więc jednym z powodów wyboru nazwy innej niż MySQL jest umożliwienie uruchamiania wielu serwerów MySQL jako usług. Nazwa usługi wpływa również na grupy opcji odczytywane z pliku opcji przez serwer w chwili jego uruchamiania:

- W przypadku braku argumentu *nazwa_uslugi* lub jego wartości MySQL serwer użyje domyślnej nazwy usługi (MySQL) i odczyta grupę `[mysqld]` ze standardowych plików opcji.
- Jeśli argument *nazwa_uslugi* ma wartość inną niż MySQL, serwer użyje wskazanej nazwy jako nazwy usługi oraz odczyta grupy `[mysqld]` i `[nazwa_uslugi]` ze standardowych plików opcji.

Jeżeli podasz nazwę usługi, to możesz również użyć opcji `--defaults-file` jako ostatniej opcji wiersza poleceń podczas tworzenia usługi dla serwera MySQL (całe polecenie powinno być wpisane w jednym wierszu):

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqld"
--install nazwa_uslugi --defaults-file=nazwa_pliku
```

W ten sposób masz alternatywę dla podawania opcji charakterystycznych dla serwera. Taka składnia wymaga podania nazwy usługi; w celu użycia domyślnej nazwy usługi należy argumentowi *nazwa_uslugi* podać wartość MySQL. Nazwa pliku opcji zostaje zapamiętana i będzie używana przez serwer w trakcie jego uruchamiania, a opcje będą odczytywane jedynie z wskazanego pliku. Odczytywane grupy opcji to `[mysqld]` oraz `[nazwa_uslugi]`, jeśli argument *nazwa_uslugi* ma wartość inną niż MySQL.

Po nazwie usługi dozwolone jest podanie pojedynczej opcji innej niż `--defaults-file`. Jednak opcja `--defaults-file` jest znacznie elastyczniejsza, ponieważ pozwala na umieszczenie w pliku dowolnej liczby opcji. Więcej informacji na temat plików opcji znajdziesz w punkcie 12.2.3, zatytułowanym „Określanie opcji startowych serwera”.

Po zainstalowaniu serwera jako usługi można go kontrolować za pomocą nazwy usługi. Kontrolę można sprawować z poziomu wiersza poleceń lub menedżera usług Windows, jeśli preferujesz środowisko graficzne. Położenie wspomnianego menedżera usług zależy od używanej wersji systemu Windows.

Aby uruchomić lub zatrzymać usługę z poziomu wiersza poleceń, należy wydać poniższe polecenia:

```
C:\> net start MySQL
C:\> net stop MySQL
```

Jeżeli używasz graficznego menedżera usług, na ekranie zobaczysz wyświetlone okno wraz z listą usług oraz informacjami dodatkowymi, jak stan usługi (uruchomiona lub zatrzymana) i sposób jej uruchamiania (automatycznie lub ręcznie). W celu uruchomienia

lub zatrzymania serwera MySQL z wyświetlonej listy wybierz odpowiadającą mu usługę, a następnie kliknij odpowiedni przycisk lub element menu.

Serwer można zatrzymać z poziomu wiersza poleceń, wydając polecenie `mysqladmin shutdown`.

Uwaga

Wprowadzie usługi można kontrolować za pomocą graficznego menedżera usług lub poleceń wydawanych z poziomu wiersza poleceń, ale powinienś unikać interakcji między dwoma wymienionymi podejściami. Zamknij graficzny menedżer usług przed wykonaniem z poziomu wiersza poleceń jakichkolwiek poleceń związanych z usługami.

Aby usunąć serwer MySQL z listy usług, należy go zatrzymać (jeśli jest uruchomiony), a następnie wydać poniższe polecenie:

```
C:\> mysql --remove
```

Powyższe polecenie powoduje usunięcie usługi MySQL o nazwie domyślnej MySQL. Jeśli chcesz wyraźnie wskazać usługę do usunięcia, podaj jej nazwę po opcji `--remove`:

```
C:\> mysql --remove nazwa_uslugi
```

12.2.3. Określanie opcji startowych serwera

Na dowolnej platformie istnieją dwie podstawowe metody podawania opcji startowych podczas uruchamiania serwera:

- Wymienienie listy opcji w wierszu poleceń. W takim przypadku można używać długich lub krótkich nazw opcji, dla których obie formy są dostępne. Na przykład, można użyć opcji `--user=mysql` lub `-u mysql`.
- Wymienienie opcji w pliku opcji. W takim przypadku każda opcja powinna znajdować się w oddzielnym wierszu. Używane mogą być jedynie długie nazwy opcji i należy pominąć myślniki znajdujące się na początku:

```
[mysql]  
user=mysql
```

Ogólne omówienie formatu i składni plików opcji, a także miejsc, w których serwer ich poszukuje, znajdziesz w punkcie F.2.2, zatytułowanym „Pliki opcji”.

Dwie metody podawania opcji nie są wzajemnie wykluczające się. Serwer szuka opcji zarówno w plikach opcji, jak i wierszu poleceń, przy czym opcje podane w wierszu poleceń mają pierwszeństwo.

Ogólnie rzecz biorąc, użycie pliku opcji jest łatwiejsze, ponieważ tak zdefiniowane opcje są stosowane w trakcie każdego uruchomienia serwera, niezależnie od zastosowanej metody. Z kolei opcje podane w wierszu poleceń mają zastosowanie jedynie w przypadku ręcznego uruchomienia serwera lub użycia skryptu `mysql_safe`. Nie działają w przypadku skryptu `mysql.server`, który jest przystosowany jedynie do obsługi argumentów `start` i `stop` z poziomu wiersza poleceń. Ponadto, poza nielicznymi wyjątkami, nie można podać

opcji startowych w wierszu poleceń, jeśli używane są opcje `--install`, `--install-manual` lub `--remove` w celu instalacji lub usunięcia serwera jako usługi Windows. (Wyjątki zostały omówione w podpunkcie 12.2.2.2, zatytułowanym „Uruchomienie serwera jako usługi Windows”).

Lokalizacje, w których serwer szuka plików opcji, zależą od używanej wersji serwera MySQL (patrz punkt F.2.2, zatytułowany „Pliki opcji”). Jednak `/etc/my.cnf` w systemach UNIX i `C:\my.ini` w Windows to standardowe położenia plików opcji. Aby wyświetlić lokalizacje, w których serwer szuka plików opcji, wydaj poniższe polecenie:

```
% mysqlld --verbose --help
```

Jeżeli plik opcji, którego chcesz użyć, jeszcze nie istnieje, po prostu utwórz go w postaci pliku zwykłego tekstu.

Opcje startowe serwera są zwykle umieszczone w grupie opcji `[mysqlld]`. Na przykład, aby wskazać, że chcesz uruchomić serwer jako użytkownik `mysql` i użyć katalogu `/usr/local/mysql`, w pliku opcji umieść poniższe wiersze:

```
[mysqlld]
user=mysql
basedir=/usr/local/mysql
```

Powyższe wiersze są odpowiednikami uruchomienia serwera wraz z następującymi opcjami w wierszu poleceń:

```
% mysqlld --user=mysql --basedir=/usr/local/mysql
```

W tabeli 12.3 wymieniono standardowe grupy opcji używane przez serwery i programy startowe serwera. Wiersz dla `mysqlld` ma również zastosowanie do serwerów takich jak `mysqlld-debug` w Windows.

Tabela 12.3. Grupy opcji używane przez programy serwera

Program	Grupy opcji używane przez program
mysqlld	[mysqlld], [server]
mysqlld_safe	[mysqlld], [server], [mysqlld_safe]
mysql.server	[mysqlld], [server], [mysql_server], [mysql.server]

Po umieszczeniu opcji w grupie wybrana grupa będzie używana w wybranym kontekście lub kontekstach. Grupy `[mysqlld]` i `[server]` są używane w przypadku opcji dla `mysqlld`. Z kolei grupy `[mysqlld_safe]`, `[mysql_server]` i `[mysql.server]` pozwalają na zdefiniowanie opcji stosowanych jedynie we wskazanych skryptach uruchomieniowych.

W systemie Windows, jeśli dla MySQL utworzysz usługę Windows i nie użyjesz domyślnej nazwy usługi, będzie to miało wpływ na odczytywane przez serwer grupy opcji. Szczegółowe informacje na ten temat przedstawiono w podpunkcie 12.2.2.2, zatytułowanym „Uruchomienie serwera jako usługi Windows”.

Skrypt `mysql.server` odczytuje plik opcji i sprawdza jedynie wartości opcji `basedir`, `datadir`, `pid-file` i `service-startup-timeout`.

12.2.4. W jaki sposób serwer nasłuchuje połączeń?

Serwer MySQL nasłuchuje połączeń na kilku interfejsach sieciowych, które kontrolujesz w przedstawiony poniżej sposób:

- Na wszystkich platformach serwer nasłuchuje połączeń TCP/IP na adresach i portach, o ile nie zostanie użyta opcja `--skip-networking`. Począwszy od MySQL 5.6.6, wartością domyślną jest `*` (gwiazdka, czyli nasłuchiwanie na wszystkich interfejsach IPv4 i IPv6). Przed wydaniem 5.6.6 wartością domyślną było `0.0.0.0`, czyli nasłuchiwanie na wszystkich interfejsach IPv4. W celu wskazania adresu należy uruchomić serwer wraz z opcją `--bind-address=adres`. Podanym adresem może być dowolny IPv4, IPv6 lub nazwa komputera. W tym ostatnim przypadku serwer zmienia nazwę komputera na adres IP, którego następnie używa. Poniższa lista przedstawia efekt dołączania różnego rodzaju adresów i sposoby akceptowania połączeń TCP/IP przez serwer. Pewne *adresy* są wartościami specjalnymi i zezwalają na połączenia na wielu interfejsach sieciowych.
 - ◆ Wartość `0.0.0.0` oznacza, że serwer akceptuje połączenia na wszystkich interfejsach IPv4.
 - ◆ Wartość `::` oznacza, że serwer akceptuje połączenia na wszystkich interfejsach IPv4 i IPv6.
 - ◆ Wartość `*` jest jak `::` z pewnym wyjątkiem: w przypadku braku IPv6 serwer akceptuje jedynie połączenia na wszystkich interfejsach IPv4.
 - ◆ W przypadku mapowanych adresów IPv4 serwer akceptuje połączenia w obu formach adresu, to znaczy IPv4 i IPv6. Na przykład, dołączenie do `::ffff:192.168.0.3` pozwala klientom na nawiązanie połączenia z użyciem opcji `--host` o wartości `::ffff:192.168.0.3` lub `192.168.0.3`.
 - ◆ W przypadku pozostałych adresów IPv4 lub IPv6 serwer akceptuje połączenia dla wskazanego, pojedynczego adresu. Na przykład, dołączenie do `192.168.0.3` zezwala na połączenia jedynie dla wskazanego adresu.
 - ◆ Domyślny numer portu to 3306. Aby wskazać inny numer portu, należy użyć opcji `--port`.
- W systemach UNIX serwer nasłuchuje także pliku gniazda domeny UNIX w oczekiwaniu na klienty lokalne, które nawiązują połączenie ze specjalną nazwą komputera `localhost` lub używają opcji `--protocol=socket`. Nie ma możliwości wyłączenia użycia pliku gniazda przez serwer. Domyślnym plikiem gniazda zwykle jest `/tmp/mysql.sock`, choć dystrybucje MySQL pochodzące od dostawców systemów operacyjnych bardzo często używają innych plików. Aby wyraźnie wskazać ścieżkę dostępu do pliku gniazda, należy użyć opcji `--socket`.
- W serwerach zainstalowanych w systemie Windows połączenia za pomocą nazwanego potoku są domyślnie wyłączone. Ich włączenie następuje po uruchomieniu serwera wraz z opcją `--named-pipe`. Dzięki wymienionej opcji klienty lokalne mogą nawiązać połączenie z serwerem za pomocą nazwanego

potoku, podając opcję `--protocol=pipe` lub nawiązując połączenie ze specjalną nazwą komputera . (kropka). Domyślną nazwą potoku jest MySQL (wielkość liter nie ma znaczenia). Podanie innej nazwy jest możliwe za pomocą opcji `--socket`.

- W serwerach zainstalowanych w systemie Windows połączenia z użyciem pamięci współdzielonej są domyślnie wyłączone. Ich włączenie następuje po uruchomieniu serwera wraz z opcją `--shared-memory`. Po włączeniu wymienionej opcji staje się ona domyślnym protokołem połączenia dla klientów lokalnych. Ponadto, klienci lokalne mogą wyraźnie wskazać użycie pamięci współdzielonej, podając opcję `--protocol=memory`. Domyślną nazwą pamięci współdzielonej jest MySQL (wielkość liter ma znaczenie). Podanie innej nazwy jest możliwe za pomocą opcji `--shared-memory-base-name`.

Ponieważ w systemie Windows połączenia za pomocą nazwanego potoku i pamięci współdzielonej są domyślnie wyłączone, bardzo często spotykaną praktyką jest ich włączenie przez dodanie odpowiednich wierszy do grupy `[mysqld]` w pliku opcji:

```
[mysqld]
named-pipe
shared-memory
```

Klient, który chce nawiązać połączenie z serwerem lokalnym za pomocą TCP/IP, nawet gdy domyślnie użyte mogą być inne protokoły, powinien wskazać 127.0.0.1 jako nazwę komputera serwera. Wymieniony adres jest adresem interfejsu TCP/IP loopback. Innym sposobem wymuszenia połączenia TCP/IP jest użycie opcji `--protocol=tcp`.

Jeżeli uruchamiany jest pojedynczy serwer, najczęściej pozwala się, aby działał wraz z domyślnymi ustawieniami sieci. W przypadku uruchamiania więcej niż tylko jednego serwera konieczne jest upewnienie się, że każdy używa unikalnych parametrów sieci. Więcej informacji na ten temat znajdziesz w podrozdziale 12.9, zatytułowanym „Uruchamianie wielu serwerów MySQL”.

12.2.5. Zatrzymanie serwera

W celu ręcznego zatrzymania serwera należy użyć narzędzia `mysqladmin`:

```
% mysqladmin -p -u root shutdown
```

Powyższe polecenie działa w systemach UNIX i Windows. Jeżeli serwer został uruchomiony w Windows jako usługa, wtedy można również użyć graficznego menedżera usług w celu wybrania i zatrzymania serwera lub ręcznie zatrzymać go z poziomu wiersza poleceń, wydając poniższe polecenie:

```
C:\> net stop MySQL
```

Jeżeli serwer MySQL został skonfigurowany w celu automatycznego uruchamiania wraz z systemem, nie trzeba podejmować żadnych specjalnych działań, aby został zatrzymany automatycznie w trakcie zamykania systemu. Systemy z rodziny BSD UNIX standardowo zamykają procesy przez wysłanie sygnału TERM, a programy prawidłowo

odpowiadają na ten sygnał (lub są siłowo zamykane, jeśli nie udzielą odpowiedzi). Odpowiedzią udzielaną przez serwer `mysqld` po otrzymaniu sygnału `TERM` jest zatrzymanie serwera.

W przypadku systemów UNIX typu System V uruchamiających serwer MySQL za pomocą skryptu `mysql.server` proces zamknięcia polega na wywołaniu wymienionego skryptu wraz z argumentem `stop`. Samodzielnie również możesz wywołać wymieniony skrypt i tym samym ręcznie zakończyć działanie serwera. Na przykład, po instalacji skryptu `mysql.server` jako `/etc/init.d/mysql` wywołujesz go w przedstawiony poniżej sposób (do wywołania wymagane są uprawnienia użytkownika `root`):

```
# /etc/init.d/mysql stop
```

W przypadku uruchamiania serwera MySQL jako usługi w Windows menedżer usługi automatycznie powoduje zatrzymanie serwera w trakcie zamykania systemu Windows. Jeżeli serwer MySQL nie jest uruchamiany jako usługa w systemie Windows, przed zamknięciem systemu serwer należy zatrzymać ręcznie za pomocą polecenia `mysqladmin shutdown`.

12.2.6. Odzyskanie kontroli nad serwerem, gdy nie można nawiązać z nim połączenia

W pewnych okolicznościach po ponownym uruchomieniu serwera może się okazać, że nie ma możliwości nawiązania z nim połączenia. To oznacza pewien problem, ponieważ zwykle zatrzymanie serwera odbywa się przez nawiązanie z nim połączenia i wydanie odpowiedniego polecenia, na przykład `mysqladmin shutdown`. Jak można rozwiązać tego rodzaju problem?

Po pierwsze, hasło dla użytkownika `root` serwera MySQL ma wartość, której nie znasz. Taka sytuacja może się zdarzyć po zmianie hasła, gdy na przykład przez pomyłkę wpiszesz niewidoczny znak kontrolny podczas podawania nowego hasła. Ewentualnie, mogłeś po prostu zapomnieć hasło.

Po drugie, w systemach UNIX połączenia z `localhost` domyślnie są nawiązywane przez plik gniazda, taki jak `/tmp/mysql.sock`. Jeżeli plik gniazda zostanie usunięty, klienci lokalne nie będą mogły go wykorzystać podczas nawiązywania połączenia. Przyczyną usunięcia wymienionego pliku może być wykonanie zadania mechanizmu `cron`, które usuwa pliki tymczasowe w katalogu `/tmp`.

Jeżeli przyczyną braku możliwości nawiązania połączenia z serwerem jest usunięcie pliku gniazda w systemie UNIX, rozwiązanie polega po prostu na ponownym uruchomieniu serwera. (W trakcie uruchamiania serwera tworzy on plik gniazda). Trudność polega na tym, że z powodu braku pliku gniazda nie można nawiązać połączenia i nakazać serwerowi ponownego uruchomienia. Zamiast tego trzeba nawiązać połączenie TCP/IP. W tym celu nawiąż połączenie z serwerem lokalnym, używając opcji `--protocol=tcp` lub zamiast `localhost` podając adres `127.0.0.1`:

```
% mysqladmin -p -u root --protocol=tcp shutdown
% mysqladmin -p -u root -h 127.0.0.1 shutdown
```

Adres 127.0.0.1 to adres IP (odwołuje się do interfejsu loopback), a więc wyraźnie wymusza nawiązanie połączenia TCP/IP zamiast użycia pliku gniazda.

Jeżeli plik gniazda UNIX został usunięty na skutek działania zadania mechanizmu cron, problem związany z usuwaniem pliku gniazda będzie powracał, o ile nie zmodyfikujesz zadania mechanizmu cron lub nie umieścisz pliku gniazda w innym położeniu. Inny plik gniazda można wskazać w globalnym pliku opcji. Na przykład, jeśli katalogiem bazowym MySQL jest */usr/local/mysql*, możesz w nim umieścić plik gniazda, dodając do pliku */etc/my.cnf* następujące wiersze:

```
[mysqld]
socket=/usr/local/mysql/mysql.sock

[client]
socket=/usr/local/mysql/mysql.sock
```

Po wprowadzeniu modyfikacji uruchom ponownie serwer, aby plik gniazda został utworzony w nowym położeniu. Ścieżkę dostępu do pliku gniazda UNIX trzeba koniecznie podać zarówno dla serwera, jak i programów klienckich, ponieważ wtedy będą używały tego samego pliku. W przypadku podania ścieżki dostępu tylko dla serwera programy klientów nadal będą oczekiwały pliku gniazda w poprzednim położeniu. Wadą tej metody jest jej działanie jedynie z klientami odczytującymi plik opcji; pewne programy firm trzecich mogą nie odczytywać pliku opcji. Jeżeli kompilujesz serwer MySQL ze źródeł, możesz zmienić jego konfigurację w taki sposób, aby domyślnie używana była inna ścieżka dostępu do pliku gniazda, zarówno dla serwera, jak i klientów. Taka zmiana automatycznie dotyczy również klientów firm trzecich działających na bazie biblioteki klienta, o ile statycznie nie mają zdefiniowanego poprzedniego położenia pliku gniazda. W takim przypadku musisz ponownie skompilować także program klienta, aby móc używać nowej biblioteki.

Jeżeli nawiązanie połączenia wynika z tego, że zapomniałeś hasła użytkownika root lub go nie znasz, musisz odzyskać kontrolę nad serwerem i dopiero wtedy będziesz miał możliwość ponownego zdefiniowania hasła. W tym celu należy skorzystać z przedstawionej poniżej procedury:

1. Zatrzymaj serwer. W systemach UNIX, jeśli możesz zalogować się jako użytkownik root do komputera z uruchomionym serwerem MySQL, masz możliwość zamknięcia serwera za pomocą polecenia `kill`. W pliku PID (najczęściej znajduje się w katalogu danych MySQL) sprawdź identyfikator procesu serwera. Identyfikator można ustalić także za pomocą polecenia `ps`. Następnie nakaż procesowi serwera zamknięcie w standardowy sposób, wysyłając sygnał `TERM`:

```
# kill -TERM PID
```

W ten sposób tabele i dzienniki zdarzeń zostaną prawidłowo zamknięte.

Jeżeli serwer uległ zawieszeniu i nie reaguje na standardowy sygnał zakończenia działania, wtedy możesz wymusić jego zamknięcie za pomocą polecenia `kill -9`.

```
# kill -9 PID
```

Polecenie `kill -9` powinno być używane jedynie w ostateczności, ponieważ wszelkie zmiany niezapisane na dysku nie zostaną wprowadzone. Z tego powodu ryzykujesz pozostawieniem tabel w niespójnym stanie.

W systemie Linux narzędzie `ps` może wyświetlić wiele „procesów” `mysqld`. Tak naprawdę, to będą wątki tego samego procesu; zamknięcie dowolnego z nich powoduje zamknięcie wszystkich.

Jeżeli do uruchomienia serwera jest używany skrypt `mysqld_safe`, monitoruje on serwer pod kątem niespodziewanego zakończenia działania. Dlatego też zamknięcie serwera MySQL za pomocą polecenia `kill -9` spowoduje jego natychmiastowe ponowne uruchomienie. Aby tego uniknąć, trzeba w pierwszej kolejności ustalić PID procesu `mysqld_safe` i zakończyć jego działanie przed zamknięciem `mysqld`.

W przypadku uruchomienia serwera jako usługi w Windows można go zatrzymać w zwykły sposób bez konieczności znajomości hasła. Wystarczy wykorzystać menedżer usług i wydać polecenie:

```
C:\> net stop MySQL
```

W celu wymuszenia zamknięcia serwera w Windows trzeba użyć menedżera zadań (*Ctrl+Alt+Del*). Podobnie jak polecenie `kill -9` w systemie Linux, menedżer zadań to ostateczność w Windows.

2. Ponownie uruchom serwer z opcją `--skip-grant-tables` w celu zawieszenia użycia tabel uprawnień do weryfikacji połączeń. W ten sposób uzyskasz możliwość nawiązania połączenia bez hasła i ze wszystkimi uprawnieniami. Jednak w ten sposób serwer pozostanie otwarty także dla innych osób, które w dokładnie taki sam sposób będą mogły nawiązać z nim połączenie. Dlatego też natychmiast po nawiązaniu połączenia wykonaj zapytanie `FLUSH PRIVILEGES`:

```
% mysql
mysql> FLUSH PRIVILEGES;
```

Zapytanie `FLUSH` powoduje, że serwer ponownie wczyta tabele uprawnień i zacznie ich używać do kontroli dostępu. Pozostaniesz połączony z serwerem, ale wszystkie kolejne próby połączeń będą jak zwykle weryfikowane za pomocą tabel uprawnień. Wykonanie zapytania `FLUSH` umożliwi także ponowne wykonanie zapytania `SET PASSWORD`, które jest wyłączone, gdy serwer nie korzysta z tabel uprawnień. Po ponownym wczytaniu tabel uprawnień możesz przystąpić do zmiany hasła użytkownika `root`, zgodnie z informacjami przedstawionymi w podrozdziale 12.1, zatytułowanym „Zabezpieczenie nowej instalacji MySQL”, na przykład:

```
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('rootpass');
```

Upewnij się, że zostało zmienione hasło dla wszystkich kont użytkowników `root`, o ile istnieje więcej niż tylko jedno.

3. Po zmianie hasła konta użytkownika root zamknij serwer i uruchom go ponownie za pomocą standardowej procedury uruchamiania serwera. Teraz powinieneś mieć możliwość nawiązania połączenia jako użytkownik root, podając nowo zdefiniowane hasło dostępu.

Jeżeli zostaniesz zmuszony do użycia polecenia `kill -9` w systemie UNIX lub menedżera zadań w Windows, wymuszone zamknięcie serwera nie daje mu możliwości zapisania na dysku wszystkich niezapisanych jeszcze zmian. Aby pomóc serwerowi w rozwiązaniu problemów, które mogą się pojawić po tego rodzaju zamknięciu, dobrym rozwiązaniem jest włączenie automatycznego odzyskiwania. Więcej informacji na ten temat znajdziesz w punkcie 14.3.1, zatytułowanym „Używanie możliwości serwera w zakresie automatycznej naprawy”.

12.3. Używanie zmiennych systemowych i stanu

Serwer MySQL ma zmienne systemowe, pozwalające na jego konfigurację, oraz zmienne stanu, pozwalające na jego monitorowanie. W tym podrozdziale dowiesz się, jak korzystać z tych zmiennych. Parametry charakterystyczne dla poszczególnych silników bazy danych zostaną przedstawione w podrozdziale 12.5, zatytułowanym „Konfiguracja silnika bazy danych”. Informacje dotyczące dostrajania serwera znajdziesz w podrozdziale 12.7, zatytułowanym „Dostrajanie serwera”. Z kolei opis poszczególnych zmiennych przedstawiono w dodatku D, zatytułowanym „Przewodnik po zmiennych systemowych, stanu i użytkownika”.

Zmienne systemowe kontrolują parametry operacyjne serwera i można je wyświetlić za pomocą zapytania `SHOW VARIABLES`. Jeżeli wartość domyślna zmiennej jest nieodpowiednia, zawsze możesz ją zmienić na inną, zapewniającą lepsze środowisko, w którym działa serwer. Pewne zmienne systemowe są używane w celu dostrajania wydajności, na przykład kontrolują wielkość buforów pamięci. Jeżeli więc masz dużą ilość pamięci, możesz zezwolić serwerowi na użycie większych buforów dla operacji dyskowych i indeksu. W ten sposób większa ilość informacji będzie przechowywana w pamięci, co zmniejsza liczbę koniecznych operacji dostępu do dysku. Inne zmienne systemowe wpływają na sposób pracy serwera z klientami. Te zmienne kontrolują tryb SQL, domyślny silnik bazy danych oraz bieżącą strefę czasową.

Serwer ma również pewien zestaw zmiennych stanu dostarczających informacje o jego wydajności działania. Tego rodzaju zmienne są wyświetlane za pomocą zapytania `SHOW STATUS`. Zmienne stanu są użyteczne do monitorowania serwera i sprawdzania, czy zmiany konfiguracyjne wprowadzone przez modyfikację zmiennych systemowych przyniosły oczekiwany efekt.

12.3.1. Sprawdzanie i ustawienie wartości zmiennych systemowych

Większość zmiennych systemowych może być ustawiona w trakcie uruchamiania serwera, w wierszu poleceń lub w plikach opcji, przy użyciu tej samej składni, jak w opcjach programu opisanych w podrozdziale F.2, zatytułowanym „Określanie opcji programu”. W trakcie działania serwera zmienne systemowe można wyświetlić za pomocą zapytania `SHOW VARIABLES`, a wiele z nich może być zmodyfikowanych za pomocą zapytania `SET`. Możliwość ustawienia zmiennych w trakcie działania serwera daje lepszą kontrolę nad jego działaniem i pozwala na uniknięcie konieczności zatrzymania serwera, aby zmienić jego konfigurację. Na przykład, możesz eksperymentować z wielkością bufora i przekonać się, jak ona wpływa na wydajność serwera; nie trzeba będzie w przypadku każdej zmiany zatrzymywać, a później ponownie uruchamiać serwera. Zmiany wprowadzone w trakcie działania serwera są aktualne tylko do chwili jego zatrzymania. Jednak jeśli ustalisz lepszą wartość dla zmiennej systemowej, możesz ją zdefiniować w pliku opcji, a ta wartość będzie używana w trakcie kolejnych uruchomień serwera.

Zmienne systemowe istnieją na dwóch poziomach: globalnym i sesji. Zmienne globalne wpływają na działanie serwera jako całości. Z kolei zmienne sesji wpływają jedynie na sposób traktowania przez serwer danego połączenia z klientem, czyli sesji. Zmienne systemowe o wartościach na poziomie sesji klient może zmieniać w ramach własnej sesji i tym samym dostosować działania serwera do własnych wymagań i jednocześnie nie wpływać na inne klienty. W przypadku zmiennych mających obie wartości serwer używa wartości globalnych do inicjalizacji odpowiadających zmiennych sesji. To się zdarza jedynie podczas rozpoczęcia sesji przez nowego klienta. Zmiany wprowadzone w zmiennej globalnej w trakcie sesji nie wpływają na wartość bieżącą odpowiadającej jej zmiennej sesji klienta.

Zmienna systemowa może mieć wartość globalną i sesji, tylko wartość globalną lub tylko sesji. Wymienione poniżej przykłady ilustrują dostępne możliwości:

- Zmienna `sql_mode` określa tryb SQL i wpływa na wiele aspektów przetwarzania przez serwer zapytań SQL. Posiada obie wartości, globalną i sesji. Każdy klient nawiązujący połączenie otrzymuje wartość sesji dla zmiennej `sql_mode`, która początkowo ma taką samą wartość jak globalna. Każdy klient może zmodyfikować wartość sesji zmiennej `sql_mode` i tym samym zmienić zachowanie serwera w danej sesji, niezależnie od innych klientów. Z kolei klient posiadający uprawnienie `SUPER` ma możliwość zmiany wartości globalnej `sql_mode`. Nowa wartość globalna stanie się wartością domyślną sesji dla klientów nawiązujących połączenie po wprowadzeniu zmiany.
- Zmienna `innodb_buffer_pool_size` ma jedynie wartość globalną. Określa wielkość bufora przechowującego dane i indeksy tabeli InnoDB. Wspomniany bufor jest współdzielony przez wszystkie klienty, a więc nie ma powodu istnienia wartości sesji dla poszczególnych klientów.
- Zmienna `error_count` ma jedynie wartość sesji. Wskazuje liczbę błędów wygenerowanych przez ostatnie zapytanie w bieżącej sesji, które wygenerowało błędy.

W dodatku D, zatytułowanym „Przewodnik po zmiennych systemu, stanu i użytkownika”, wymieniono wszystkie zmienne systemowe, wskazano możliwe do ustawienia w trakcie uruchamiania serwera lub jego działania, a także ich wartości globalne i sesji. W poniższych podpunktach dowiesz się, jak pobierać i ustawiać wartości zmiennej systemowej.

12.3.1.1. Sprawdzenie wartości zmiennej systemowej

W celu sprawdzenia bieżących wartości zmiennej systemowej należy wykonać zapytanie `SHOW VARIABLES`:

```
mysql> SHOW VARIABLES;
```

Variable_name	Value
auto_increment_increment	1
auto_increment_offset	1
autocommit	ON
automatic_sp_privileges	ON
back_log	50
basedir	/usr/local/mysql
big_tables	OFF

...

Podobnie jak w przypadku klauzuli `LIKE`, dane wyjściowe można ograniczyć do rekordów zmiennych o nazwach dopasowanych do określonego wzorca SQL:

```
mysql> SHOW VARIABLES LIKE '%buffer%';
```

Variable_name	Value
bulk_insert_buffer_size	8388608
innodb_buffer_pool_instances	1
innodb_buffer_pool_size	134217728
innodb_change_buffering	all
innodb_log_buffer_size	8388608
join_buffer_size	131072
key_buffer_size	8388608

...

Aby wskazać ogólne warunki wyboru rekordów, użyj klauzuli `WHERE`. Poniższe zapytanie wyszukuje wszystkie zmienne, które definiują wartość czasu wygaśnięcia ważności krótszą niż 60 sekund:

```
mysql> SHOW VARIABLES
-> WHERE Variable_name LIKE '%timeout%' AND Value < 60;
```

Variable_name	Value
connect timeout	10
innodb_lock_wait_timeout	50
innodb_rollback_on_timeout	OFF
net_read_timeout	30

Domyślnie zapytanie `SHOW VARIABLES` wyświetla wartości sesji zmiennej. Aby wyraźnie wskazać, że mają być wyświetlone wartości globalne lub sesji, należy do zapytania dodać właściwy kwalifikator (odpowiednio `GLOBAL` lub `SESSION`):

```
SHOW GLOBAL VARIABLES;
SHOW SESSION VARIABLES;
```

Kwalifikator `LOCAL` jest synonimem `SESSION`.

W celu wyświetlenia pojedynczej wartości zmiennej należy użyć składni `@@GLOBAL.nazwa_zmiennej` dla zmiennej globalnej lub `@@SESSION.nazwa_zmiennej` dla zmiennej sesji. Jeżeli użyjesz `@@nazwa_zmiennej` bez kwalifikatora, wyświetlona będzie wartość sesji zmiennej, o ile istnieje; w przeciwnym razie wyświetlona zostanie wartość globalna zmiennej.

Słowa kluczowe kwalifikatorów i nazwy zmiennych nie rozróżniają wielkości liter.

Składnia oparta na `@@` jest ogólnego przeznaczenia i może być stosowana w zapytaniach `SET`, `SELECT`, a także innych zapytaniach `SQL`:

```
SELECT @@default_storage_engine AS 'Default storage engine';
```

Istnieje również możliwość wykonywania zapytań do tabel `GLOBAL_VARIABLES` i `SESSION_VARIABLES` bazy danych `INFORMATION_SCHEMA` w celu pobrania informacji o zmiennej systemowej, na przykład:

```
SELECT * FROM INFORMATION_SCHEMA.GLOBAL_VARIABLES
WHERE VARIABLE_NAME LIKE '%binlog%';
```

Wydanie polecenia `mysqladmin variables` z poziomu wiersza poleceń powoduje wyświetlenie wartości globalnych zmiennej systemowej.

12.3.1.2. Ustawienie zmiennych systemowych w trakcie uruchamiania serwera

Większość globalnych zmiennych systemowych można ustawić w trakcie uruchamiania serwera. Wystarczy potraktować nazwę zmiennej jako nazwę opcji i ustawić ją bezpośrednio. Na przykład, zmienna `max_connections` określa maksymalną liczbę jednoczesnych połączeń klientów. Aby wymienionej zmiennej przypisać wartość 200 w wierszu wywołania `mysqld`, użyj poniższej składni:

```
% mysql --max_connections=200
```

W celu ustawienia zmiennej w pliku opcji należy użyć poniższej składni:

```
[mysqld]
max_connections=200
```

Zwykle znacznie wygodniejsze jest ustawianie zmiennych systemowych w pliku opcji niż w wierszu poleceń, ponieważ wówczas nie trzeba pamiętać o ich ustawianiu w trakcie każdego uruchamiania serwera.

W trakcie uruchamiania serwera myślniki i znaki podkreślenia są wymienne w nazwach zmiennych systemowych, podobnie jak w przypadku nazw opcji. Dlatego też w wierszu poleceń stosujemy składnię:

```
% mysqld --max-connections=200
```

natomiast w pliku opcji:

```
[mysqld]
max-connections=200
```

W przypadku zmiennych przedstawiających wielkość bufora lub inne wielkości, gdy nie podano żadnego przyrostka, wartości są domyślnie wyrażane w bajtach. Ewentualnie można użyć przyrostka K, M lub G (wielkość liter nie ma znaczenia) i tym samym wskazać wielkość wyrażoną odpowiednio w kilobajtach, megabajtach lub gigabajtach. Wymienione poniżej wiersze powodują ustawienie limitu połączenia na 1024 i wielkości puli bufora InnoDB na 16 MB:

```
[mysqld]
max_connections=1K
innodb_buffer_pool_size=16M
```

Pewnych zmiennych systemowych nie można ustawić bezpośrednio jako opcji startowych. Jeśli podejmiesz taką próbę, serwer wygeneruje komunikat błędu. W takich przypadkach być może dostępna jest odpowiednia opcja. Na przykład, w trakcie uruchamiania serwera nie ma możliwości bezpośredniego ustawienia zmiennej `time_zone`, ale można użyć opcji `--default-time-zone`. W dodatku D, zatytułowanym „Przewodnik po zmiennych systemowych, stanu i użytkownika”, dowiesz się, które zmienne globalne mogą być ustawione bezpośrednio. Dla zmiennych, które nie mogą być ustawione bezpośrednio, w dodatku wymieniono odpowiednie opcje służące do ustawienia zmiennej, o ile takie opcje istnieją.

Do dodatku D powinienś również zajrzeć, jeśli w dodatku F, zatytułowanym „Przewodnik po programie MySQL”, szukałeś opisu opcji serwerowej, ale go tam nie znalazłeś. Opcja w rzeczywistości może być zmienną systemową.

12.3.1.3. Ustawienie zmiennej systemowej w trakcie działania serwera

Składnia pozwalająca na ustawienie zmiennej systemowej w trakcie działania serwera zależy od rodzaju ustawianej wartości: globalnej lub sesji. Aby ustawić zmienną globalną *nazwa_zmiennej*, użyj zapytania SET o jednym z poniższych formatów:

```
SET GLOBAL nazwa_zmiennej = wartość;
SET @@GLOBAL.nazwa_zmiennej = wartość;
```

Natomiast w celu ustawienia zmiennej sesji stosowane są podobne składnie:

```
SET SESSION nazwa_zmiennej = wartość;
SET @@SESSION.nazwa_zmiennej = wartość;
```

LOCAL jest synonimem dla SESSION.

Jeżeli nie zostanie podany kwalifikator, zapytanie SET modyfikuje wartość sesji zmiennej:

```
SET nazwa_zmiennej = wartość;  
SET @@ nazwa_zmiennej = wartość;
```

W celu ustawienia zmiennej globalnej konieczne jest posiadanie uprawnień SUPER. Ustawiona wartość pozostaje aktualna aż do chwili jej ponownej zmiany lub zamknięcia serwera. Ustawienie zmiennej sesji nie wymaga żadnych szczególnych uprawnień, wartość pozostaje aktualna aż do chwili jej ponownej zmiany lub zakończenia bieżącej sesji.

Istnieje możliwość ustawienia wielu zmiennych za pomocą pojedynczego zapytania SET. W tym celu przypisania zmiennych trzeba rozdzielić przecinkami:

```
SET SESSION sql_warnings = 0, GLOBAL default_storage_engine = InnoDB;
```

W zapytaniu ustawiającym wartość dla wielu zmiennych wyraźnie podany kwalifikator GLOBAL lub SESSION ma zastosowanie do zmiennych, dla których nie podano innego kwalifikatora. Poniższe zapytanie ustawia zmienne globalne v1 i v2 oraz zmienne sesji v3 i v4:

```
SET GLOBAL v1 = wartość1, v2 = wartość2, SESSION v3 = wartość3, v4 = wartość4;
```

W przeciwieństwie do zmiennych ustawianych w trakcie uruchamiania serwera, w nazwach zmiennych używanych w trakcie działania serwera myślników nie można zastąpić znakami podkreślenia oraz nie można podawać wartości z użyciem przyrostków K, M lub G. Jednak można stosować wyrażenia, które z kolei mogą odnosić się do wartości innych zmiennych. Poniższe polecenia powodują ustawienie zmiennej read_buffer_size wartości globalnej wynoszącej 2 MB i wartości sesji wynoszącej dwukrotność wartości globalnej:

```
SET GLOBAL read_buffer_size = 2*1024*1024;  
SET SESSION read_buffer_size = 2*@@GLOBAL.read_buffer_size;
```

Zmienne systemowe mogą mieć przypisaną wartość specjalną DEFAULT. Przypisanie DEFAULT zmiennej globalnej oznacza tak naprawdę przypisanie jej skompilowanej wartości domyślnej, nawet jeśli inna wartość została podana w trakcie uruchamiania serwera. Przypisanie DEFAULT zmiennej sesji powoduje przypisanie wartości bieżącej odpowiadającej jej zmiennej globalnej.

MySQL obsługuje koncepcję strukturalnej zmiennej systemowej składającej się ze zbioru powiązanych zmiennych systemowych, które są pogrupowane i dostępne jako komponenty zmiennej strukturalnej. Ten rodzaj zmiennej jest używany do konfiguracji buforów kluczy MySQL (patrz punkt 12.7.2, zatytułowany „Dostrajanie silnika bazy danych”).

12.3.2. Sprawdzenie wartości zmiennej stanu

Serwer posiada także zmienne stanu pozwalające na monitorowanie jego działania.

W celu wyświetlenia tych zmiennych należy wykonać zapytanie SHOW STATUS:

```
mysql> SHOW STATUS;
```

Variable_name	Value
Aborted_clients	0
Aborted_connects	1
Binlog_cache_disk_use	0
Binlog_cache_use	3
Binlog_stmt_cache_disk_use	0
Binlog_stmt_cache_use	0
Bytes_received	125
Bytes_sent	151
...	

Zmienne stanu są ustawiane jedynie przez serwer, a więc użytkownik może je jedynie odczytywać, ale nie ma możliwości ich modyfikacji za pomocą zapytania SET, jak ma to miejsce w przypadku zmiennych systemowych.

Podobnie jak zmienne systemowe, także zmienne stanu mają wartości globalne i sesji. Dlatego też w zapytaniu można stosować kwalifikatory GLOBAL i SESSION:

```
SHOW GLOBAL STATUS;
SHOW SESSION STATUS;
```

Kwalifikator GLOBAL powoduje wyświetlenie stanu serwera jako całości (wartość zagregowana dla wszystkich sesji, począwszy od uruchomienia serwera). Z kolei SESSION wyświetla stan dla bieżącej sesji. Domyślnie stosowany jest kwalifikator SESSION, dla którego synonimem jest LOCAL.

Jeżeli zmienna ma jedynie wartość globalną, tę samą wartość otrzymasz zarówno dla GLOBAL, jak i SESSION.

W przypadku zapytania SHOW STATUS, podobnie jak w przypadku SHOW VARIABLES, klauzule LIKE i WHERE mogą ograniczyć dane wyjściowe zapytania do zmiennych o nazwach dopasowanych do wzorca SQL lub spełniających ogólne zdefiniowane warunki. Na przykład, w celu sprawdzenia wartości zmiennych stanu powiązanych z aktywnością rejestracji zdarzeń przez InnoDB należy wykonać poniższe zapytanie:

```
mysql> SHOW GLOBAL STATUS LIKE 'innodb%log%';
```

Variable_name	Value
Innodb_log_waits	0
Innodb_log_write_requests	45504
Innodb_log_writes	1234
Innodb_os_log_fsyncs	1384
Innodb_os_log_pending_fsyncs	0
Innodb_os_log_pending_writes	0
Innodb_os_log_written	23465984

Istnieje również możliwość wykonywania zapytań do tabel GLOBAL_STATUS i SESSION_STATUS bazy danych INFORMATION_SCHEMA w celu pobierania informacji zmiennych stanu. Jedną z zalet takiego rozwiązania względem zapytania SHOW STATUS jest możliwość użycia wartości w obliczeniach, na przykład:

```
SET @queries =  
  (SELECT VARIABLE_VALUE FROM INFORMATION_SCHEMA.GLOBAL_STATUS  
   WHERE VARIABLE_NAME LIKE 'Queries');  
SET @uptime =  
  (SELECT VARIABLE_VALUE FROM INFORMATION_SCHEMA.GLOBAL_STATUS  
   WHERE VARIABLE_NAME LIKE 'Uptime');  
SELECT @queries/@uptime/60 AS 'Queries per minute';
```

12.4. Interfejs wtyczek

Na przestrzeni czasu architektura serwera MySQL stawała się coraz bardziej otwarta na wtyczki. Oznacza to, że pewne fragmenty kodu implementujące funkcje serwera mogą być skompilowane oddzielnie od serwera, przechowywane w plikach zewnętrznych i wczytywane na żądanie. W serwerze MySQL wtyczki implementują funkcje takie jak silniki bazy danych, tabele `INFORMATION_SCHEMA`, modyfikatory wyszukiwania pełnego tekstu, mechanizmy replikacji, możliwości w zakresie audytu oraz metody uwierzytelniania.

Otwarta na wtyczki architektura serwera daje administratorom MySQL możliwość dostosowania serwera do własnych potrzeb przez wybór wczytywanych wtyczek. Jednocześnie możliwe jest zmniejszenie obciążenia serwera przez niewczytywanie niepotrzebnych wtyczek. Na przykład, programiści firm trzecich mogą tworzyć silniki bazy danych jako wtyczki sprzedawane oddzielnie od serwera, bez konieczności powiązania prac nad wtyczką z cyklem prac nad serwerem.

W tym podrozdziale zostanie omówiona ogólna charakterystyka interfejsu wtyczek. Więcej informacji, zwłaszcza o wtyczkach uwierzytelniania, znajdziesz w punkcie 13.2.7, zatytułowanym „Wtyczki metod uwierzytelniania i użytkownicy proxy”.

Interfejs wtyczek składa się z wymienionych poniżej komponentów:

- **Pliki wtyczki.** Wtyczki serwera są przechowywane w plikach obiektów nazywanych bibliotekami wtyczki, ponieważ mogą zawierać wiele wtyczek.
- **Katalog wtyczki.** Wszystkie pliki wtyczki muszą znajdować się w tym katalogu. Bardzo często katalogiem wtyczki jest *lib/plugin* w katalogu instalacji MySQL, ale jego położenie można określić na podstawie wartości zmiennej systemowej `plugin_dir`. W celu użycia innego położenia trzeba w trakcie uruchamiania serwera ustawić wartość `plugin_dir`.
- **Język kontrolny.** Pozwala on administratorowi na wskazanie serwerowi wtyczek, które powinny zostać wczytane. Przybiera formę opcji serwera i zapytań SQL umożliwiających obsługę wtyczki. Na przykład, opcja `--plugin-load` powoduje wczytanie wtyczek w trakcie uruchamiania serwera, natomiast zapytanie `INSTALL PLUGIN` wczytuje je w trakcie działania serwera.
- **Rejestr wtyczki.** W trakcie uruchamiania serwer sprawdza tabelę `plugin` w bazie danych `mysql` i automatycznie wczytuje zarejestrowane tam wtyczki, o ile nie zostanie użyta opcja `--skip-grant-tables`. W celu rejestracji wtyczki w wymienionej tabeli należy wykonać zapytanie `INSTALL PLUGIN`.

Biblioteki wtyczek mają charakterystyczne dla platformy rozszerzenie nazwy pliku. Na przykład, wtyczka o nazwie *mylib.so* w systemie Linux ma nazwę *mylib.dll* w Windows. W tym rozdziale używamy *.so* jako rozszerzenia nazwy pliku biblioteki. W używanym systemie musisz stosować odpowiednie rozszerzenie.

Na potrzeby prezentowanej tutaj analizy przyjmujemy założenie, że w katalogu wtyczek serwera znajdują się zainstalowane dwa pliki bibliotek wtyczek:

- Biblioteka o nazwie *my_engine.so*, zawierająca pojedynczą wtyczkę implementującą silnik bazy danych o nazwie *MY_ENGINE*.
- Biblioteka o nazwie *info_tables.so*, zawierająca wiele wtyczek implementujących tabele *LOCKS* i *USERS* bazy danych *INFORMATION_SCHEMA* w celu wyświetlenia informacji o aktualnych blokadach i połączonych klientach.

Interfejs wtyczek pozwala na przeprowadzanie następujących operacji:

- W chwili uruchamiania serwera wczytanie wtyczek z pliku, pojedynczo lub grupowo.
- W trakcie działania serwera wczytanie lub usunięcie poszczególnych wtyczek z pamięci.
- W trakcie działania serwera sprawdzenie, jakie wtyczki są dostępne.

W celu wczytania wtyczek w trakcie uruchamiania serwera należy użyć opcji `--plugin-load` lub `--plugin-load-add`. Dla każdej z nich wartością jest lista składająca się z jednej lub wielu rozdzielonych średnikami nazw bibliotek wtyczek: *nazwa_wtyczki=nazwa_biblioteki* lub *nazwa_biblioteki*. Podanie nazwy wtyczki i biblioteki powoduje, że serwer wczyta tylko wskazaną wtyczkę z biblioteki. Z kolei podanie nazwy biblioteki bez nazwy wtyczki powoduje wczytanie przez serwer wszystkich wtyczek z biblioteki. Na przykład, aby wczytać wszystkie wtyczki w obu wymienionych wcześniej bibliotekach, należy w pliku opcji umieścić poniższe wiersze, a następnie ponownie uruchomić serwer:

```
[mysqld]
plugin-load=my_engine.so;info_tables.so
```

Powyższe wiersze można zapisać w bardziej rozwlekły sposób, wymieniając poszczególne wtyczki:

```
[mysqld]
plugin-load=my_engine=my_engine.so;locks=info_tables.so;users=info_tables.so
```

W celu wczytania jedynie wtyczki *LOCKS* z biblioteki *info_tables.so* należy w pliku opcji umieścić poniższe wiersze:

```
[mysqld]
plugin-load=locks=info_tables.so
```

Wielkość liter w nazwie wtyczki nie ma znaczenia. Z kolei wielkość liter w nazwie pliku ma znaczenie, jeśli system plików rozróżnia wielkość liter.

Opcja `--plugin-load-add` jest podobna do `--plugin-load`. Różnica pomiędzy nimi polega na tym, że w przeciwieństwie do `--plugin-load-add` każdy egzemplarz `--plugin-load` zeruje listę wtyczek do wczytania. Jeżeli wielokrotnie użyjesz opcji `--plugin-load`, tylko ostatnie jej wywołanie będzie miało jakikolwiek efekt. Egzemplarze `--plugin-load-add` powodują dodanie wtyczek do zestawu wtyczek przeznaczonych do wczytania. Na przykład, w celu wczytania wszystkich trzech wtyczek przez ich wymienienie w pliku opcji można umieścić następujące wiersze:

```
[mysql>]
plugin-load-add=my_engine=my_engine.so
plugin-load-add=locks=info_tables.so
plugin-load-add=users=info_tables.so
```

Opcja `--plugin-load-add` jest znacznie wygodniejsza w użyciu, gdy masz wiele wtyczek, ponieważ pozwala na ich wymienienie w wielu wierszach, które tym samym są łatwiejsze w odczycie. Ponadto, poszczególne wiersze można umieścić w komentarzu, jeśli zajdzie potrzeba. Z kolei opcja `--plugin-load` wymaga użycia pojedynczej, długiej wartości i jej edycji, gdy chcesz zmienić zestaw wczytywanych wtyczek.

Pomiędzy kolejnymi uruchomieniami serwera nie są zachowywane wtyczki wczytywane za pomocą opcji `--plugin-load` lub `--plugin-load-add`. Jeżeli serwer uruchomisz ponownie i pominiesz wymienione opcje, nie zostaną wczytane żadne wtyczki.

W celu wczytania wtyczki w trakcie działania serwera należy wykonać zapytanie `INSTALL PLUGIN`. Każdy egzemplarz zapytania zawiera nazwę wtyczki oraz zawierającego go pliku obiektu biblioteki, na przykład:

```
INSTALL PLUGIN my_engine SONAME 'my_engine.so';
INSTALL PLUGIN locks SONAME 'info_tables.so';
INSTALL PLUGIN users SONAME 'info_tables.so';
```

Zapytanie `INSTALL PLUGIN` przeprowadza „trwałą” instalację wtyczki. Oznacza to nie tylko jej wczytanie, ale również rejestrację w tabeli `mysql.plugin`, aby w trakcie kolejnych uruchomień serwera była automatycznie wczytywana.

W celu dezinstalacji wtyczki w trakcie działania serwera należy wykonać zapytanie `UNINSTALL PLUGIN`, na przykład:

```
UNINSTALL PLUGIN users;
```

Zapytanie `UNINSTALL PLUGIN` powoduje również wyrejestrowanie wtyczki z tabeli `mysql.plugin` (o ile została wcześniej zarejestrowana), aby nie była wczytywana w trakcie kolejnych uruchomień serwera. Jeżeli wtyczka nie została zaprojektowana w celu jej trwałego usunięcia w trakcie działania serwera i może być dezaktywowana tylko po zamknięciu serwera, wykonanie zapytania `UNINSTALL PLUGIN` zakończy się wygenerowaniem błędu.

Ponieważ zapytania `INSTALL PLUGIN` i `UNINSTALL PLUGIN` modyfikują tabelę `mysql.plugin`, ich wykonanie wymaga uprawnień odpowiednio `INSERT` i `DELETE` do wymienionej tabeli.

Niezależnie od chwili wczytania wtyczki, podczas uruchamiania serwera lub w trakcie jego działania obecność wtyczki można sprawdzić, wykonując zapytanie `SHOW PLUGINS` lub zaglądając do tabeli `PLUGINS` bazy danych `INFORMATION_SCHEMA`, na przykład:

```
mysql> SHOW PLUGINS;
```

Name	Status	Type	Library	License
...				
my_engine	ACTIVE	STORAGE ENGINE	my_engine.so	GPL
...				
LOCKS	ACTIVE	INFORMATION SCHEMA	info_tables.so	GPL
USERS	ACTIVE	INFORMATION SCHEMA	info_tables.so	GPL
...				

Jeżeli serwer wczyta wtyczkę w trakcie uruchamiania, ponieważ została ona wbudowana, zarejestrowana w tabeli `mysql.plugin` lub wskazana za pomocą opcji `--plugin-load` bądź `--plugin-load-add`, to istnieje możliwość kontrolowania stanu aktywacji wtyczki. W tym celu należy użyć opcji o takiej samej nazwie jak wtyczka. Na przykład, `--my_engine=ON` lub `--my_engine=OFF` pozwala na aktywację lub dezaktywację silnika bazy danych `MY_ENGINE` implementowanego za pomocą wtyczki o nazwie `my_engine`. Po nazwie wtyczki można podać następujące wartości opcjonalne:

- **OFF**: dezaktywacja wtyczki. Odpowiednikami tej wartości są `--disable-nazwa_wtyczki`, `--skip--nazwa_wtyczki` i `--nazwa_wtyczki=0`.
- **ON**: aktywacja wtyczki. To jest działanie domyślne podejmowane w przypadku, gdy po nazwie wtyczki nie zostanie podana żadna wartość. Odpowiednikami tej wartości są `--enable-nazwa_wtyczki` i `--nazwa_wtyczki=1`. Jeżeli wczytanie wtyczki zakończy się niepowodzeniem, będzie ona dezaktywowana.
- **FORCE**: wartość podobna do **ON**, ale serwer nie uruchomi się, jeśli wczytanie wtyczki zakończy się niepowodzeniem.
- **FORCE_PLUS_PERMANENT**: wartość podobna do **FORCE**, ale w trakcie działania serwera nie pozwoli on na jej dezaktywację za pomocą zapytania `UNINSTALL PLUGIN`. Ta wartość została wprowadzona w MySQL 5.5.7.

Wymienione wartości nie rozróżniają wielkości liter i nie dopasowują nazwy opcji do nazwy wtyczki.

Kolumna `LOAD_OPTION` w tabeli `PLUGINS` bazy danych `INFORMATION_SCHEMA` zawiera informacje o sposobie aktywacji wczytanych wtyczek.

12.5. Konfiguracja silnika bazy danych

Serwer MySQL obsługuje wiele silników baz danych i zapewnia ogromną elastyczność w zakresie dostępności wspomnianych silników. Ogólne omówienie roli silników bazy danych w MySQL przedstawiono w punkcie 2.6.1, zatytułowanym „Cechy charakterystyczne silników bazy danych”. W tym podrozdziale dowiesz się, jak skonfigurować silniki bazy danych używane przez serwer, a także poznasz ogólne informacje dotyczące silnika InnoDB. Informacje dotyczące wydajności silników InnoDB i MyISAM przedstawiono w punkcie 12.7.2, zatytułowanym „Dostrajanie silnika bazy danych”.

12.5.1. Wybór silnika bazy danych

Serwer MySQL zapewnia wysoką elastyczność pod względem dostępnych silników bazy danych. Domyślnie dostępny jest silnik InnoDB (to również jest domyślnie stosowany silnik bazy danych), ale można go wyłączyć. Zawsze dostępne są także silniki MyISAM, MEMORY, MERGE i CSV, ale ich nie można wyłączyć. Dostępne mogą być również inne silniki bazy danych; możesz je dowolnie włączać i wyłączać w trakcie uruchamiania serwera. Na przykład, wyłączając niepotrzebne silniki bazy danych, zmniejszasz zapotrzebowanie serwera na pamięć operacyjną. (Jednak po wyłączeniu silnika nie można uzyskać dostępu do wszelkich tabel utworzonych za jego pomocą).

W tabeli 12.4 wymieniono silniki standardowo znajdujące się w instalacji MySQL oraz ich stan domyślny, a także opcję startową pozwalającą na zmianę tego stanu. Więcej informacji na temat opcji kontrolujących stan wtyczek silników bazy danych zamieszczono w podrozdziale 12.4, zatytułowanym „Interfejs wtyczek”.

Tabela 12.4. Konfiguracja silników bazy danych

Silnik bazy danych	Stan domyślny, jeśli silnik jest dostępny	Opcja zmiany stanu
ARCHIVE	Włączony	--archive=OFF w celu wyłączenia.
BLACKHOLE	Włączony	--blackhole=OFF w celu wyłączenia.
CSV	Włączony	Nie może być wyłączony.
FEDERATED	Wyłączony	--federated w celu włączenia.
InnoDB	Włączony	--innodb=OFF w celu wyłączenia.
MyISAM	Włączony	Nie może być wyłączony.
MEMORY	Włączony	Nie może być wyłączony.
MERGE	Włączony	Nie może być wyłączony.

Aby w trakcie działania serwera wyświetlić dostępne silniki bazy danych oraz ich stan, należy wykonać zapytanie `SHOW ENGINES` lub wykonać zapytanie do tabeli `ENGINE` bazy danych `INFORMATION_SCHEMA`, na przykład:

```
mysql> SELECT ENGINE, SUPPORT FROM INFORMATION_SCHEMA.ENGINES;
```

```
+-----+-----+
| ENGINE | SUPPORT |
+-----+-----+
| CSV    | YES     |
| InnoDB | DEFAULT |
| MyISAM | YES     |
| MRG_MYISAM | YES  |
| MEMORY | YES     |
+-----+-----+
```

Informacje dodatkowe znajdziesz w podpunkcie 2.6.1.1, zatytułowanym „Ustalenie dostępnych silników bazy danych”.

12.5.2. Wybór domyślnego silnika bazy danych

InnoDB jest skompilowany jako domyślny silnik bazy danych, ale w trakcie uruchamiania serwera można wybrać inny silnik domyślny lub zmienić go już podczas działania serwera, używając do tego zmiennej systemowej `default_storage_engine`. Na przykład, aby domyślnym silnikiem bazy danych stał się MyISAM, w pliku opcji należy umieścić poniższe wiersze:

```
[mysqld]
default_storage_engine = myisam
```

W celu zmiany domyślnego silnika bazy danych w trakcie działania serwera należy użyć jednego z poniższych zapytań:

```
SET GLOBAL default_storage_engine = nazwa_silnika;
SET SESSION default_storage_engine = nazwa_silnika;
```

Pierwsze zapytanie wymaga uprawnienia `SUPER` i powoduje ustawienie domyślnego silnika bazy danych dla wszystkich klientów nawiązujących połączenie już po wprowadzeniu zmiany. Z kolei drugie zapytanie nie wymaga żadnych specjalnych uprawnień, wpływa jedynie na bieżącą sesję klienta i może być stosowane przez dowolnego klienta do zmiany używanego przez niego domyślnego silnika bazy danych.

Aby sprawdzić, jakie silniki bazy danych są ustawione globalnie i dla sesji, trzeba wykonać poniższe zapytanie:

```
SELECT @@GLOBAL.default_storage_engine, @@SESSION.default_storage_engine;
```

Przed wydaniem wersji MySQL 5.6.3 zmienna systemowa `default_storage_engine` miała zastosowanie w stosunku do tabel trwałych i tymczasowych. Począwszy od wydania MySQL 5.6.3, ma zastosowanie jedynie dla tabel trwałych. Dla tabel tymczasowych istnieje oddzielna zmienna `default_tmp_storage_engine`, określająca domyślny silnik bazy danych dla tabel tymczasowych. Typowym zastosowaniem tabel tymczasowych jest uniknięcie obciążenia związanego z transakcjami w InnoDB. W takim przypadku dla tabel wybierany jest silnik MyISAM lub MEMORY.

Jeżeli uruchomisz serwer wraz z opcją `--innodb=OFF` w celu wyłączenia silnika InnoDB zmiennej systemowej `default_storage_engine` (i `default_tmp_storage_engine` w MySQL 5.6) ustaw wartość wskazującą silnik inny niż InnoDB, ponieważ w przeciwnym razie serwer się nie uruchomi.

12.5.3. Konfiguracja silnika InnoDB

Począwszy od wersji MySQL 5.5, domyślnym silnikiem bazy danych jest InnoDB. (W poprzednich wersjach to był silnik MyISAM). Silnik InnoDB zarządza systemową przestrzenią tabel przeznaczoną do przechowywania zawartości tabel oraz katalogu danych. Istnieje również możliwość skonfigurowania InnoDB w taki sposób, aby poszczególne tabele były przechowywane w oddzielnych przestrzeniach tabel. InnoDB ma również własne pliki dzienników zdarzeń oraz bufora w pamięci.

12.5.3.1. Konfiguracja przestrzeni tabel InnoDB

Domyślnie silnik bazy danych InnoDB nie używa oddzielnych plików dla poszczególnych tabel. Zamiast tego wszystkie tabele InnoDB są umieszczane w pojedynczej systemowej przestrzeni tabel, która jest logicznie unifikowanym blokiem pamięci masowej traktowanej przez silnik jako ogromna struktura danych. (Przestrzeń tabel można porównać do wirtualnego systemu plików). Dla każdej tabeli InnoDB przechowywanej w systemowej przestrzeni tabel, w podkatalogu bazy danych, do której należy dana tabela, istnieje jeden unikalnie powiązany z nią plik *.frm*. Systemowa przestrzeń tabel zawiera także katalog danych InnoDB przechowujący informacje o strukturze tabeli.

Istnieje możliwość, aby każda tabela InnoDB była przedstawiana w postaci własnego pliku przestrzeni tabeli. Zastosowanie tego rodzaju konfiguracji wymaga uruchomienia serwera wraz z włączoną zmienną systemową o nazwie *innodb_file_per_table*. W takim przypadku systemowa przestrzeń tabel nadal jest potrzebna, ponieważ zawiera katalog danych InnoDB, ale nie będzie zajmowała dużo miejsca.

12.5.3.1.1. Parametry konfiguracyjne systemowej przestrzeni tabel InnoDB

Systemowa przestrzeń tabel InnoDB, choć pod względem logicznym jest pojedynczym obszarem pamięci masowej, to fizycznie składa się z jednego lub więcej plików na dysku. Każdy jej komponent może być zwykłym plikiem lub partycją, zgodnie z ustawieniami omówionych tutaj opcji konfiguracyjnych. Wspomniane opcje można podać w wierszu poleceń, ale w praktyce to bardzo rzadko stosowane rozwiązanie. Zamiast tego konfigurację przestrzeni tabel przeprowadza się w pliku opcji za pomocą odpowiedniej grupy serwera (na przykład *[mysqld]* lub *[server]*), aby za każdym razem serwer używał tej samej konfiguracji.

Za pomocą wymienionych poniżej zmiennych systemowych można kontrolować liczbę, wielkość oraz miejsce umieszczenia plików w systemowej przestrzeni tabel:

- Opcja *innodb_data_home_dir* definiuje katalog nadrzędny dla plików komponentów tworzących przestrzeń tabeli. Domyślnie to jest katalog danych serwera MySQL.
- Opcja *innodb_data_file_path* definiuje specyfikację plików komponentów przestrzeni tabel w katalogu głównym InnoDB. Wartości składające się ze specyfikacji jednego lub więcej plików muszą być rozdzielone średnikami. Każda specyfikacja składa się z nazwy pliku, wielkości oraz ewentualnie innych opcji rozdzielonych dwukropkami. Łączna wielkość komponentów przestrzeni tabel musi wynosić przynajmniej 10 MB. Domyślnie to pojedynczy plik o nazwie *ibdata1* o wielkości 10 MB, który może automatycznie zwiększyć tę wielkość.

Opierając się na wartościach domyślnych, jeśli nie zdefiniujesz żadnej z wymienionych powyżej zmiennych systemowych, InnoDB tworzy systemową przestrzeń tabel w postaci umieszczonego w katalogu danych serwera MySQL pojedynczego pliku *ibdata1* o wielkości 10 MB, która może być automatycznie zwiększona.

Przyjmujemy założenie, że chcesz utworzyć przestrzeń nazw składającą się z dwóch plików o wielkości 4 GB o nazwach *innodata1* i *innodata2*, umieszczonych w katalogu danych MySQL. W takim przypadku nie trzeba definiować zmiennej systemowej *innodb_data_home_dir*, ponieważ używany będzie domyślny katalog danych MySQL. Pliki trzeba skonfigurować w następujący sposób:

```
[mysqld]
innodb_data_file_path=innodata1:4G;innodata2:4G
```

Silnik InnoDB łączy wartości zmiennych systemowych *innodb_data_home_dir* i *innodb_data_file_path* w celu określenia ścieżek dostępu plików tworzących przestrzeń tabel:

- Jeżeli zmienna systemowa *innodb_data_home_dir* nie została podana, domyślną ścieżką dostępu będzie prowadząca do katalogu danych serwera MySQL, a InnoDB interpretuje nazwy plików w zmiennej *innodb_data_file_path* jako względne do katalogu danych MySQL.
- Jeżeli wartość zmiennej systemowej *innodb_data_home_dir* nie jest pusta, InnoDB interpretuje ją jako katalog, w którym umieszczone są wszystkie pliki wymienione w zmiennej *innodb_data_file_path*, oraz interpretuje te nazwy plików jako względne do wartości *innodb_data_home_dir*.
- Jeżeli wartość zmiennej systemowej *innodb_data_home_dir* została wyraźnie ustawiona jako pusta, InnoDB traktuje wszystkie specyfikacje plików w zmiennej *innodb_data_file_path* jako bezwzględne ścieżki dostępu.

Opierając się na powyższych regułach, wymienione poniżej trzy konfiguracje wskazują dokładnie ten sam zestaw plików przestrzeni tabel, przy założeniu, że katalogiem danych MySQL jest */var/mysql/data*:

```
[mysqld]
innodb_data_file_path=ibdata1:500M;ibdata2:500M
```

```
[mysqld]
innodb_data_home_dir=/var/mysql/data
innodb_data_file_path=ibdata1:500M;ibdata2:500M
```

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=/var/mysql/data/ibdata1:500M;/var/mysql/data/ibdata2:500M
```

Wartość zmiennej *innodb_data_file_path* składa się z rozdzielonych średnikami specyfikacji plików, w której poszczególne elementy są rozdzielone przecinkami. Najprostsza składnia specyfikacji pliku składa się z nazwy pliku i jego wielkości, choć dozwolone są także inne składnie:

```
ścieżka_dostępu:wielkość_pliku
ścieżka_dostępu:wielkość_pliku:autoextend
ścieżka_dostępu:wielkość_pliku:autoextend:max:maksymalna_wielkość_pliku
```

W pierwszym przykładzie podana została stała wielkość pliku (parametr *wielkość_pliku*). Wspomniana wielkość powinna być dodatnią liczbą całkowitą wraz z przyrostkiem M lub G wskazującym jednostki, odpowiednio megabajty lub gigabajty. W drugim przykładzie zdefiniowano, że wielkość pliku może być automatycznie zwiększana. Dlatego też po zapełnieniu pliku InnoDB automatycznie będzie zwiększał jego wielkość w krokach. Trzeci przykład jest podobny do drugiego, ale zawiera definicję maksymalnej wielkości pliku, do której może zostać on zwiększony przez mechanizm automatycznego zwiększania wielkości. Tylko ostatni komponent przestrzeni tabeli może zostać oznaczony jako automatycznie zwiększający się.

Domyślna wielkość kroku w mechanizmie automatycznego zwiększania wielkości pliku wynosi 8 MB. Aby zdefiniować inną wielkość kroku, trzeba użyć zmiennej systemowej o nazwie `innodb_autoexpand_increment`.

12.5.3.1.2. Konfiguracja systemowej przestrzeni tabel InnoDB

Standardowo, systemowa przestrzeń tabel składa się ze zwykłych plików i nie obejmuje żadnych partycji (plików urządzeń). Aby przeprowadzić początkową konfigurację systemowej przestrzeni tabel składającej się z jedynie zwykłych plików, należy użyć poniższej procedury:

1. Dodaj odpowiednie wiersze do pliku opcji.
2. Upewnij się, że istnieją katalogi, w których mają znaleźć się pliki komponentów tworzące przestrzeń tabel. InnoDB tworzy pliki, ale nie katalogi.
3. Upewnij się, że żaden z plików komponentów jeszcze nie istnieje.
4. Uruchom serwer. Silnik InnoDB sprawdzi, że pliki nie istnieją, więc utworzy je i zainicjalizuje.

Jeżeli uruchomiłeś serwer bez przeprowadzenia wyraźnej konfiguracji InnoDB, silnik InnoDB utworzy systemową przestrzeń tabel na podstawie konfiguracji domyślnej. W celu przeprowadzenia wyraźnej konfiguracji przestrzeni tabel w pierwszej kolejności zatrzymaj serwer i usuń pliki powiązane z InnoDB (pliki przestrzeni tabel i dzienników zdarzeń). Następnie zdefiniuj opcje konfiguracyjne i ponownie uruchom serwer. (Zrób to *przed* utworzeniem jakichkolwiek tabel InnoDB. W przeciwnym razie przed przeprowadzeniem konfiguracji powinieneś utworzyć kopię zapasową tabel za pomocą narzędzia `mysqldump` i przywrócić je po zakończeniu konfiguracji).

Procedura jest nieco bardziej skomplikowana w przypadku używania całych partycji jako komponentów systemowej przestrzeni tabel InnoDB. Poniżej wymieniono kwestie, które trzeba wziąć pod uwagę:

- Bardzo łatwo można tworzyć ogromne przestrzenie tabel. Komponent partycji może obejmować całą partycję. Natomiast komponenty w postaci zwykłych plików są ograniczone narzucaną przez używany system operacyjny maksymalną wielkością pliku.
- Użycie partycji gwarantuje całkowicie ciągłą przestrzeń na dysku, podczas gdy zwykłe pliki podlegają fragmentacji. W trakcie inicjalizacji przestrzeni tabel silnik InnoDB próbuje zminimalizować fragmentację zwykłych plików przez zapisanie

w nich zer, co wymusza od razu zaalokowanie ich w całości zamiast w krokach. Jednak takie rozwiązanie może co najwyżej zmniejszyć poziom fragmentacji, ale nie gwarantuje jej uniknięcia.

- Użycie całej partycji minimalizuje obciążenie na skutek eliminacji warstwy zarządzającej systemem plików. W pewnych systemach wspomniane obciążenie może być niewielkie, z kolei w innych wystarczająco duże, aby użycie partycji przynosiło wymierne korzyści.

Czynnikiem przeciwko użyciu całych partycji dla przestrzeni tabel InnoDB może być stosowane oprogramowanie do tworzenia kopii zapasowej. Wspomniane oprogramowanie może być przeznaczone do pracy z systemem plików, a nie całymi partycjami. W takim przypadku wykorzystanie całych partycji może nieco utrudnić wykonywanie kopii zapasowej systemu.

Wykorzystanie całej partycji w przestrzeni tabel wiąże się z konfiguracją zarówno początkową, jak i w późniejszym czasie. Przyjmujemy założenie, że używasz systemu UNIX, w którym partycja o wielkości 200 GB jest dostępna jako urządzenie `/dev/rdsk8`. Konieczne jest zdefiniowanie wartości zmiennej systemowej `innodb_data_home_dir`, ponieważ wymieniona partycja nie znajduje się w katalogu danych MySQL. Jeżeli zmiennej `innodb_data_home_dir` przypiszesz pustą wartość, wtedy w zmiennej `innodb_data_file_path` możesz podać pełną ścieżkę dostępu do pliku urządzenia i skonfigurować partycję w następujący sposób:

1. Przeprowadź konfigurację początkową partycji i podaj jej wielkość wraz z przyrostkiem `newraw`. Ten przyrostek informuje InnoDB, że plik jest całą partycją wymagającą inicjalizacji:


```
[mysql]
innodb_data_home_dir=
innodb_data_file_path=/dev/rdsk8:200Gnewraw
```
2. Uruchom serwer. Silnik InnoDB zauważy przyrostek `newraw` i zainicjalizuje partycję. Ponadto, przestrzeń tabel zostanie potraktowana jako tylko do odczytu, ponieważ silnik wie, że konieczne jest przeprowadzenie jej dalszej konfiguracji.
3. Po inicjalizacji partycji przez InnoDB zatrzymaj serwer.
4. Przeprowadź ponowną konfigurację partycji, zmieniając przyrostek z `newraw` na `raw`:


```
[mysql]
innodb_data_home_dir=
innodb_data_file_path=/dev/rdsk8:200Graw
```
5. Ponownie uruchom serwer. InnoDB dostrzeże zmianę przyrostka na `raw` i przyjmie założenie o zainicjalizowaniu partycji, co oznacza możliwość jej użycia w zwykłym trybie do odczytu i zapisu.

Jeżeli wskażesz całą partycję jako część przestrzeni tabel InnoDB, upewnij się, że serwer ma uprawnienia odczytu i zapisu do tej partycji. Ponadto, upewnij się, że partycja nie będzie używana do innych celów. W przeciwnym razie procesy będą między sobą

konkurować, sądząc, że są właścicielami partycji i że mogą jej używać w dowolny sposób, co może doprowadzić do wzajemnego nadpisywania danych. Na przykład, jeśli przez pomyłkę wskażesz partycję przestrzeni wymiany jako partycję dla przestrzeni tabel InnoDB, system operacyjny może zacząć zachowywać się zupełnie nieprzewidywalnie!

Aby przeprowadzić konfigurację systemowej przestrzeni tabel InnoDB w systemie Windows, ukośniki \ w ścieżkach dostępu należy zastąpić ukośnikami / lub podwójnymi \\. Ponadto, poszczególne części każdej specyfikacji pliku nadal powinnyś rozdzielać dwukropkami, nawet pomimo faktu, że dwukropki pojawiają się także w nazwach plików (pełna ścieżka dostępu w Windows rozpoczyna się od litery oznaczającej dysk i dwukropka). Po napotkaniu dwukropka InnoDB sprawdza kolejny znak. Jeżeli będzie nim cyfra, wtedy InnoDB uznaje kolejną część specyfikacji za wielkość, w przeciwnym razie za ścieżkę dostępu. Na przykład, poniższe polecenia powodują konfigurację przestrzeni tabel składającej się z umieszczonych w napędach C i D plików o wielkości 500 MB i 10 GB:

```
[mysql>]
innodb_data_home_dir=
innodb_data_file_path=C:/ibdata1:500M;D:/ibdata2:10G
```

Jeżeli podczas początkowej konfiguracji przestrzeni tabel uruchomienie serwera kończy się niepowodzeniem, ponieważ silnik InnoDB nie będzie mógł utworzyć niezbędnych plików, sprawdź dziennik błędów i przekonaj się, w czym tkwi problem. Następnie usuń wszystkie pliki utworzone przez InnoDB (wyłączając całe partycje, których chcesz użyć), popraw konfigurację i ponownie uruchom serwer. Jeżeli korzystasz z całych partycji, pamiętaj o zmianie specyfikacji polegającej na dodaniu przyrostka newraw w trakcie inicjalizacji, a następnie na raw po uruchomieniu i zatrzymaniu serwera.

12.5.3.1.3. Ponowna konfiguracja przestrzeni tabel InnoDB

Po inicjalizacji systemowej przestrzeni tabel InnoDB i rozpoczęciu korzystania z niej nie ma możliwości zmiany wielkości plików komponentów. Jednak zawsze można dodać kolejny plik na liście istniejących, co może okazać się konieczne po zapelnieniu aktualnie przydzielonego miejsca dla przestrzeni tabel. Jednym z symptomów zapelnienia przestrzeni tabel jest ciągle niepowodzenie w przeprowadzeniu transakcji InnoDB i ich wycofywanie mimo tego, że powinny się udać. Aby określić ilość wolnego miejsca, należy sprawdzić wartość `Data_free` w danych wyjściowych poniższego zapytania, w którym *nazwa_tabeli* oznacza nazwę dowolnej tabeli znajdującej się w systemowej przestrzeni tabel InnoDB:

```
mysql> SHOW TABLE STATUS LIKE 'nazwa_tabeli';
```

W celu zwiększenia systemowej przestrzeni tabel przez dodanie kolejnego komponentu należy użyć poniższej procedury:

1. Jeżeli serwer jest uruchomiony, zatrzymaj go.
2. Jeżeli ostatni komponent przestrzeni tabeli jest plikiem, który automatycznie zwiększa wielkość, musisz zmienić jego specyfikację na postać pliku o stałej wielkości, a dopiero później będziesz mógł na końcu dodać kolejny plik. W tym celu konieczne jest ustalenie rzeczywistej wielkości pliku, zaokrąglenie jej w dół

do najbliższej wielokrotności 1 MB (1 048 576, a nie 1 000 000 bajtów) i podanie tej wielkości w specyfikacji pliku. Przyjmujemy założenie, że plik aktualnie został zdefiniowany w następujący sposób:

```
[mysql]
innodb_data_file_path=ibdata1:100M:autoextend
```

Jeżeli rzeczywista wielkość pliku wynosi obecnie 121 634 816 bajtów, to podaj wielkość 121 634 816 / 1 048 576, czyli 116 MB, i zmień specyfikację na poniższą:

```
[mysql]
innodb_data_file_path=ibdata1:116M
```

3. Na końcu obecnej listy plików dodaj specyfikację dla nowego komponentu. Jeżeli nowy komponent jest zwykłym plikiem, upewnij się, że jeszcze nie istnieje. W przypadku, gdy komponent jest całą partycją, zastosuj omówioną wcześniej procedurę użycia całej partycji jako części przestrzeni tabel. (Pamiętaj o użyciu najpierw przyrostka newraw, a później raw po uruchomieniu i zatrzymaniu serwera).
4. Uruchom ponownie serwer.

12.5.3.1.4. Używanie oddzielnych przestrzeni tabel dla każdej tabeli InnoDB

Aby używać oddzielnych przestrzeni tabel dla każdej tabeli InnoDB, należy włączyć zmienną systemową `innodb_file_per_table`:

```
[mysql]
innodb_file_per_table=1
```

Po włączeniu wymienionej zmiennej każda nowa tabela InnoDB będzie utworzona wraz z plikami *.frm* (format) i *.ibd* (dane). Oba wymienione pliki są przechowywane w podkatalogu bazy danych dla bazy danych zawierającej tę tabelę. InnoDB automatycznie zwiększy wielkość plików *.ibd*, jeśli zajdzie potrzeba.

Włączenie lub wyłączenie użycia oddzielnych przestrzeni tabel wpływa jedynie na sposób tworzenia *nowych* tabel przez InnoDB. Silnik ma dostęp do tabel utworzonych w systemowej lub poszczególnych przestrzeniach tabel, niezależnie od zmian wartości zmiennej `innodb_file_per_table`.

W przypadku tabel przechowywanych w oddzielnych plikach *.ibd* InnoDB włącza także inne funkcje:

- Dla tabel nieużywanych w charakterze kluczy zewnętrznych w innych tabelach zapytanie `TRUNCATE TABLE` jest wykonywane znacznie szybciej i odzyskuje miejsce na dysku. Obie wymienione cechy nie są prawdziwe w przypadku tabel przechowywanych w systemowej przestrzeni tabel.
- InnoDB obsługuje wiele formatów plików, a format domyślny nosi nazwę Antelope. Po włączeniu zmiennej systemowej `innodb_file_per_table` istnieje możliwość używania formatu o nazwie Barracuda. W tym celu konieczne jest również włączenie zmiennej systemowej o nazwie `innodb_file_format`:

```
[mysql]
innodb_file_per_table=1
innodb_file_format=Barracuda
```

Format Barracuda pozwala na stosowanie rekordów w formacie COMPRESSED lub DYNAMIC dla tworzenia nowych tabel InnoDB. Więcej informacji na temat cech charakterystycznych wymienionych formatów rekordów przedstawiono w podrozdziale 5.4, „Wybór formatu tabeli dla efektywnych zapytań”.

Normalnie zmienne systemowe `innodb_file_per_table` i `innodb_file_format` są ustawiane podczas uruchamiania serwera, ale mogą być również zmienione w trakcie jego działania. To może być użyteczne dla ustawień, które mają zastosowanie jedynie do tworzenia pewnych tabel, na przykład:

```
SET GLOBAL innodb_file_per_table = 1;
SET GLOBAL innodb_file_format = 'Barracuda';
CREATE TABLE ... ENGINE=INNODB ROW_FORMAT=COMPRESSED;
SET GLOBAL innodb_file_format = 'Antelope';
SET GLOBAL innodb_file_per_table = 0;
```

12.5.3.2. Zmienne silnika bazy danych InnoDB

W poprzednich podpunktach przedstawiono sposoby konfiguracji przestrzeni tabel InnoDB. Silnik InnoDB ma własne pliki dzienników zdarzeń, bufor w pamięci oraz wiele innych parametrów konfiguracyjnych. Na przedstawionej poniżej liście wymieniono kilka najczęściej używanych parametrów, które mają wpływ na ogólne działanie silnika bazy danych InnoDB:

- `innodb_buffer_pool_size`

Jeżeli masz dostępną pamięć operacyjną, zwiększenie puli bufora InnoDB może zmniejszyć stopień użycia dysku podczas uzyskiwania dostępu do danych i indeksów. Więcej informacji na ten temat znajdziesz w punkcie 12.7.2, zatytułowanym „Dostrajanie silnika bazy danych”.

- `innodb_log_buffer_size`

Silnik InnoDB próbuje buforować w pamięci informacje o każdej transakcji i zapisuje je na dysku w pojedynczej operacji dyskowej po zakończeniu transakcji. Jeżeli transakcja jest ogromna i przekracza wielkość bufora, aktywność dyskowa jest większa i zapis zawartości bufora na dysku może wystąpić wielokrotnie jeszcze przed zakończeniem transakcji. Zwiększenie wielkości bufora pozwala na buforowanie w pamięci większej liczby transakcji bez konieczności wcześniejszego ich zapisu na dysku. Wartość domyślna bufora wynosi 1 MB, a maksymalna użyteczna wartość wynosi 8 MB.

- `innodb_log_group_home_dir`

Silnik InnoDB ma własne pliki dzienników zdarzeń, które jeśli jeszcze nie istnieją, to są tworzone w trakcie uruchamiania serwera. Wspomniane pliki mają nazwy rozpoczynające się od `ib_`. Domyślnie, silnik InnoDB tworzy je w katalogu danych. W celu wyraźnego wskazania ścieżki dostępu do katalogu, w którym powinny znajdować się pliki dzienników zdarzeń InnoDB, należy użyć zmiennej

`innodb_group_home_dir`. (Możesz zdecydować się na taki krok w celu bardziej równomiernego rozłożenia aktywności dyskowej przez umieszczenie plików dzienników zdarzeń w fizycznie innym napędzie, niezawierającym katalogu danych). InnoDB tworzy jedynie pliki, a nie katalogi, więc przed uruchomieniem serwera upewnij się, że istnieją katalogi przeznaczone do przechowywania plików dzienników zdarzeń.

■ `innodb_log_file_size`, `innodb_log_files_in_group`

Po wypełnieniu dziennika zdarzeń InnoDB zapisuje zawartość puli bufora na dysku. Użycie większych plików dzienników zdarzeń zmniejsza częstotliwość ich wypełniania, a tym samym zmniejsza liczbę operacji zapisu na dysku. (Wadą takiego rozwiązania jest wydłużenie czasu naprawy tabel po awarii). Wartość zmiennej `innodb_log_file_size` pozwala na wskazanie wielkości plików dzienników zdarzeń, natomiast zmiennej `innodb_log_files_in_group` na zmianę liczby plików. Ważną wartością jest tutaj całkowita wielkość dzienników zdarzeń, która powstaje na podstawie dwóch wartości. Nie może ona przekraczać 4 GB. Jeżeli pliki dzienników zdarzeń zostały już utworzone, ale chcesz zmienić ich wielkość, to konieczne jest całkowite zamknięcie serwera, aby silnik InnoDB miał szansę na zapisanie wszystkich transakcji na dysku. Następnie usuń pliki dzienników zdarzeń i przeprowadź ponowną konfigurację InnoDB. Po ponownym uruchomieniu serwera silnik InnoDB utworzy nowe pliki dzienników zdarzeń.

12.6. Kwestie związane z globalizacją

Globalizacja obejmuje internacjonalizację i lokalizację. Internacjonalizacja oznacza możliwość używania oprogramowania zgodnie z lokalnymi konwencjami. Z kolei lokalizacja oznacza możliwość wyboru określonego zestawu konwencji lokalnych spośród obsługiwanych. Wymienione poniżej aspekty konfiguracji MySQL są powiązane z internacjonalizacją i lokalizacją:

- Domyślna strefa czasowa serwera.
- Domyślne kodowanie znaków i kolejność sortowania.
- Język wyświetlania komunikatów diagnostycznych i błędów.
- Lokalne nazwy dni i miesięcy.

12.6.1. Konfiguracja obsługi stref czasowych

Serwer MySQL ustawia domyślną strefę czasową na podstawie przeprowadzonej analizy środowiska. Najczęściej będzie ona odpowiadała strefie czasowej ustawionej w komputerze, w którym działa serwer MySQL. Istnieje jednak możliwość wyraźnego zdefiniowania strefy czasowej w trakcie uruchamiania serwera. Ponadto, serwer pozwala każdemu klientowi na nadpisanie ustawienia domyślnego i samodzielny wybór strefy czasowej.

W ten sposób aplikacja będzie mogła używać ustawienia czasu zależnie od miejsca uruchomienia klienta, a nie od miejsca działania serwera. Poniżej przedstawiono analizę możliwości serwera MySQL w zakresie obsługi wielu stref czasowych.

Dwie zmienne systemowe przechowują informacje o strefie czasowej:

- `system_time_zone`. Ta zmienna przechowuje informacje o strefie czasowej ustalonej w trakcie uruchamiania serwera i stosowanej w komputerze, w którym działa serwer. Zmienna istnieje tylko w postaci globalnej zmiennej systemowej, nie ma możliwości jej zerowania w trakcie działania serwera. Możesz wpłynąć na sposób ustawienia wartości zmiennej `system_time_zone` podczas uruchamiania serwera. W tym celu wystarczy przed uruchomieniem serwera ustawić zmiennej środowiskowej TZ wartość żądanej strefy czasowej. Jednak ustawienie zmiennej TZ w pewnych sytuacjach nie będzie łatwe, na przykład gdy serwer jest uruchamiany wraz z systemem. W systemach UNIX innym sposobem ustawienia strefy czasowej jest podanie opcji `--timezone` skryptowi startowemu `mysqld_safe` (nie `mysqld`, który nie zna wymienionej opcji). Najlepszym rozwiązaniem jest umieszczenie wymienionej opcji w grupie `[mysqld_safe]` pliku opcji, zwłaszcza w przypadku wywoływania `mysqld_safe` pośrednio przez skrypt `mysql.server`, który nie obsługuje opcji wiersza poleceń. Na przykład, aby wskazać strefę czasową U.S. Central dla `mysqld_safe`, należy dodać poniższe wiersze do pliku opcji serwera:

```
[mysqld_safe]
timezone=US/Central
```

Przykład pokazuje powszechnie stosowaną składnię, która działa w systemach Linux i OS X. Kolejna często spotykana składnia ma poniższy format:

```
[mysqld_safe]
timezone=CST6CDT
```

Używaj dowolnej składni obsługiwanej przez Twój system.

- `time_zone`. Ta zmienna przechowuje domyślną strefę czasową serwera MySQL. Domyślnie, wartością tej zmiennej jest `SYSTEM`, co oznacza użycie ustawień zmiennej `system_time_zone`. Aby ustawić wartość zmiennej `time_zone` podczas uruchamiania serwera, należy użyć opcji `--default-time-zone` dla `mysqld`. W trakcie działania serwera używa on wartości globalnej `time_zone` do ustawienia wartości sesji `time_zone` dla każdego klienta nawiązującego połączenie i wspomniana wartość staje się domyślną strefą czasową klienta. Każdy klient może wyzerować strefę czasową w ramach sesji przez ustawienie wartości sesji zmiennej `time_zone`. Użytkownik administracyjny posiadający uprawnienie `SUPER` może ustawić wartość globalną zmiennej `time_zone` i tym samym zmienić wartość domyślną dla klientów, które nawiążą połączenie już po wprowadzeniu zmiany.

W celu ustalenia aktualnych wartości globalnej i sesji strefy czasowej należy wykonać poniższe zapytanie:

```
SELECT @@GLOBAL.time_zone, @@SESSION.time_zone;
```

Zmienna `time_zone` akceptuje trzy rodzaje wartości, choć jedna z nich wymaga dodatkowych działań administracyjnych. Przedstawione poniżej zapytania powodują ustawienie wartości sesji. Jeśli masz uprawnienie `SUPER`, wtedy kwalifikator `SESSION` możesz zastąpić kwalifikatorem `GLOBAL` i ustawić wartość globalną.

- W celu użycia wartości `system_time_zone` zmiennej `time_zone` przypisz wartość `SYSTEM`:

```
SET SESSION time_zone = 'SYSTEM';
```

- W celu wskazania przesunięcia względem UTC wskaż wartość przesunięcia w godzinach i minutach wraz ze znakiem:

```
SET SESSION time_zone = '+00:00';    # UTC.
SET SESSION time_zone = '+03:00';    # 3 godziny przed UTC.
SET SESSION time_zone = '-11:00';    # 11 godzin po UTC.
```

- W celu odwołania się do lokalizacji podaj nazwę strefy czasowej:

```
SET SESSION time_zone = 'US/Central';
SET SESSION time_zone = 'CST6CDT';
SET SESSION time_zone = 'Asia/Jakarta';
```

Aby użyć ostatniej wymienionej metody (ustawienie strefy czasowej przez podanie jej nazwy), trzeba włączyć rozpoznawanie nazw stref czasowych w serwerze przez wczytanie informacji z plików stref czasowych systemu operacyjnego do zestawu tabel w bazie danych `mysql`. To nie następuje automatycznie w trakcie procesu instalacyjnego `MySQL`, który tworzy wymienione tabele, ale ich nie wypełnia. Ręczne wypełnienie wspomnianych tabel informacjami pochodzącymi z plików zawierających strefy czasowe wymaga w pierwszej kolejności ustalenia położenia plików. Następnie trzeba wykorzystać program `mysql_tzinfo_to_sql`, który odczytuje pliki i na ich podstawie tworzy odpowiednie zapytania SQL przekazywane programowi `mysql` w celu ich wykonania. Jeżeli plik strefy czasowej to `/usr/share/zoneinfo`, wtedy polecenie wczytujące jego zawartość do bazy danych `mysql` przedstawia się następująco:

```
% mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -p -u root mysql
```

Po wczytaniu zawartości do tabel należy ponownie uruchomić serwer. To powinno wystarczyć w większości wersji systemu `UNIX`. Z kolei w `Windows` oraz innych systemach nieposiadających plików stref czasowych konieczne jest pobranie pakietu zawierającego przygotowane tabele `MyISAM` wraz z informacjami o strefach czasowych. Pakiet pobierzesz na stronie:

<http://dev.mysql.com/downloads/timezones.html>

Pobierz pakiet i rozpakuj go. Po zatrzymaniu serwera `MySQL` skopiuj pliki `.frm`, `.myd` i `.myi` do podkatalogu bazy danych `mysql` znajdującego się w katalogu danych `MySQL`. Następnie ponownie uruchom serwer.

12.6.2. Ustawienie domyślnego kodowania znaków i kolejności sortowania

Kodowanie znaków określa znaki, jakie mogą być używane w ciągach tekstowych. MySQL obsługuje wiele kodowań znaków, które można zdefiniować na poziomie serwera, bazy danych, tabeli, kolumny i ciągu tekstowego. Ponadto, MySQL obsługuje wiele sekwencji kolejności sortowania w poszczególnych kodowaniach znaków. Wspomniana kolejność sortowania wpływa na operacje sortowania i porównywania ciągów tekstowych.

Domyślne kodowanie znaków to `latin1`, natomiast domyślna kolejność sortowania to `latin1_swedish_ci`. Aby zmienić te wartości, w trakcie uruchamiania serwera należy ustawić zmienne systemowe `character_set_server` i `collation_server`. Wybrana kolejność sortowania musi być zgodna z kodowaniem znaków. (To znaczy, początkowa część nazwy kolejności sortowania musi być taka sama jak nazwa kodowania znaków). Na przykład:

```
% mysqlld --character_set_server=utf8 \
--collation_server=utf8_icelandic_ci
```

Jeżeli serwer został skompilowany ze źródeł, istnieje możliwość zmiany jego ustawień domyślnych za pomocą opcji `DEFAULT_CHARSET` i `DEFAULT_COLLATION` podczas uruchamiania CMake, na przykład:

```
% cmake -DDEFAULT_CHARSET=utf8 \
-DDEFAULT_COLLATION=utf8_icelandic_ci
```

Aby wskazać kodowanie znaków, którego program kliencki powinien używać po jego uruchomieniu, użyj opcji `--default-character-set`. Jeżeli pliki kodowania znaków nie znajdują się w położeniu domyślnie sprawdzanym przez program, trzeba je wskazać klientowi za pomocą opcji `--character-sets-dir`.

Informacje dodatkowe dotyczące możliwości serwera MySQL w zakresie kodowania znaków przedstawiono w podrozdziale 2.4, zatytułowanym „Obsługa kodowania znaków”. Natomiast dokładne informacje o tworzeniu kolumn znaków i ich używaniu znajdziesz w rozdziale 3., zatytułowanym „Typy danych”.

12.6.3. Wybór języka wyświetlania komunikatów błędów

Serwer MySQL ma możliwość generowania komunikatów diagnostycznych oraz błędów w kilku językach. Domyślnie to język angielski. Jeśli chcesz dowiedzieć się, jakie jeszcze języki są dostępne, zajrzyj do katalogu *share* instalacji MySQL. Nazwy jego podkatalogów odpowiadają dostępnym językom.

Aby wybrać język wyświetlania komunikatów błędów, należy zmiennej systemowej `lc_messages` przypisać odpowiednią wartość, na przykład `pl_PL` dla języka polskiego lub `en_US` dla języka angielskiego. Serwer wyszukuje katalog zawierający plik komunikatów błędów przez przełożenie nazwy lokalizacji na nazwę języka i wyszukanie w katalogu wskazanym przez zmienną `lc_messages_dir` podkatalogu o nazwie języka. Jeżeli katalog domyślny jest niepoprawny (domyślnie to *share* w katalogu instalacji MySQL), zmiennej

`lc_messages_dir` przypisz odpowiednią ścieżkę dostępu. Na przykład, aby wyświetlić komunikaty błędów w języku niemieckim i wskazać, że odpowiednie informacje znajdują się w katalogu `/var/mysql/share`, uruchom serwer w następujący sposób:

```
% mysql -d --lc_messages=de_DE --lc_messages_dir=/var/mysql/share
```

Obie zmienne systemowe mogą być ustawione podczas uruchamiania serwera. Poszczególne klienty mogą ustawiać wartość sesji zmiennej `lc_messages` w trakcie działania serwera i tym samym wybrać inny język. To użyteczne rozwiązanie, gdy różne klienty tego samego serwera chcą otrzymywać zlokalizowane komunikaty błędów w języku innym niż domyślny.

12.6.4. Wybór ustawień językowych

Serwer MySQL określa sposób wyświetlania nazw dni i miesięcy na podstawie wartości zmiennej systemowej `lc_time_names`. Wartość wymienionej zmiennej wpływa na wyniki generowane przez funkcje daty i czasu, takie jak `DAYNAME()`, `MONTHNAME()` i `DATE_FORMAT()`.

Domyślnie wybrany jest język angielski (`en_US`). W celu wybrania innego języka należy ustawić wartość zmiennej `lc_time_names` podczas uruchamiania serwera lub już w trakcie jego działania. Poszczególne klienty mogą ustawiać wartość sesji zmiennej `lc_time_names` w trakcie działania serwera i tym samym nadpisać wartość domyślną. Na przykład, uruchomienie serwera wraz z przypisaną wartością `es_ES` (język hiszpański) zmiennej `lc_time_names` powoduje otrzymanie następującego wyniku poniższego zapytania:

```
mysql> SELECT MONTHNAME('2000-07-01');
+-----+
| MONTHNAME('2000-07-01') |
+-----+
| julio                    |
+-----+
```

Klient preferujący język włoski może wykonać następujące zapytania:

```
mysql> SET lc_time_names='it_IT';
mysql> SELECT MONTHNAME('2000-07-01');
+-----+
| MONTHNAME('2000-07-01') |
+-----+
| luglio                   |
+-----+
```

12.7. Dostrojanie serwera

Administratorzy mający kontrolę nad serwerem MySQL lub komputerem, w którym został on uruchomiony, mogą przeprowadzić operację optymalizacji jego wydajności działania i dostrojenia serwera. Na przykład, można dostroić pewne parametry serwera dotyczące przetwarzania zapytania. Ponadto, pewne czynniki w konfiguracji sprzętowej

mają bezpośredni wpływ na szybkość przetwarzania zapytań. Wspomniane optymalizacje poprawiają wydajność serwera jako całości, a tym samym przynoszą korzyści wszystkim użytkownikom MySQL.

Podczas przeprowadzania optymalizacji administracyjnych musisz pamiętać o wymienionych poniżej zasadach.

- Uzyskanie dostępu do danych w pamięci zawsze jest szybsze niż do danych na dysku.
- Wydłużenie czasu przechowywania danych w pamięci zmniejsza liczbę operacji na dysku.
- Pobieranie informacji z indeksu jest znacznie ważniejsze niż pobieranie ich z rekordów danych.

Najważniejsze parametry, które można zmienić, to wielkości różnych buforów, na przykład bufora tabeli i bufora silnika bazy danych zapewniającego buforowanie informacji dla operacji indeksowania. Jeżeli masz dostępną pamięć, jej alokacja dla buforów serwera pozwala na dłuższe przechowywanie informacji w pamięci i zmniejszenie liczby operacji na dysku. To jest dobre rozwiązanie, ponieważ informacje są znacznie szybciej pobierane z pamięci niż odczytywane z dysku.

W kolejnych punktach zostaną przedstawione różne sposoby optymalizacji działania serwera:

- Informacje o wielu parametrach ogólnie wpływających na wydajność serwera przedstawiono w punkcie 12.7.1, zatytułowanym „Zmienne systemowe ogólnego przeznaczenia do dostrajania serwera”.
- Poszczególne silniki bazy danych dostarczają własne parametry pozwalające na konfigurację wydajności ich działania. Zapoznaj się z punktem 12.7.2, zatytułowanym „Dostrajanie silnika bazy danych”.
- Serwer MySQL może buforować wyniki zapytań, aby kolejne takie same zapytania przetwarzać bez konieczności ich ponownego wykonywania. Zapoznaj się z punktem 12.7.3, zatytułowanym „Używanie bufora zapytań”.

Ogólne informacje dotyczące ustawiania zmiennych systemowych przedstawiono w punkcie 12.3.1, zatytułowanym „Sprawdzanie i ustawienie wartości zmiennych systemowych”. Podczas zmiany wartości parametrów pamiętaj o następujących kwestiach:

- Jednorazowo zmieniaj tylko jeden parametr. Jednoczesna zmiana wielu zmiennych znacznie utrudnia określenie wyniku każdej z wprowadzonych zmian.
- Wartości zmiennych systemowych zwiększaj krokowo. Jeżeli zmienną zwiększysz o ogromną wartość, która w teorii im większa, tym lepsza, wówczas w systemie może zabraknąć dostępnych wartości. To może doprowadzić do gwałtownego załamania wydajności działania serwera z powodu zbyt wysokiej wartości zmiennej.

- Podczas dostrajania parametrów zamiast eksperymentować z produkcyjnym serwerem MySQL, lepszym rozwiązaniem będzie przygotowanie oddzielnego serwera testowego.

Pozostałe strategie, które mogą pomóc w znacznie wydajniejszym działaniu serwera, przedstawiają się następująco:

- Wyłączenie nieużywanych silników bazy danych. Dla wyłączonych silników serwer nie alokuje pamięci, a tym samym jej większa ilość pozostaje dla pozostałych zadań. Większość silników bazy danych można usunąć z pliku binarnego serwera w trakcie etapu konfiguracji, gdy serwer MySQL jest kompilowany ze źródeł. W przypadku silników wbudowanych w serwer można je wyłączyć w trakcie działania MySQL, używając odpowiednich opcji startowych. Dokładne informacje na ten temat znajdziesz w punkcie 12.5.1, zatytułowanym „Wybór silnika bazy danych”.
- Zapewnij proste uprawnienia dostępu. Wprawdzie serwer buforuje w pamięci zawartość tabel uprawnień, jeśli w tabelach `tables_priv`, `columns_priv` lub `procs_priv` bazy danych mysql będzie znajdowało się wiele rekordów, ale serwer musi sprawdzić ich zawartość podczas sprawdzania uprawnień dla zapytań SQL. Jeśli wymienione tabele są puste, serwer może zoptymalizować sprawdzanie uprawnień przez pominięcie wymienionych poziomów uprawnień. Zapoznaj się z punktem 13.4.2, zatytułowanym „Weryfikacja uprawnień zapytania”.

12.7.1. Zmienne systemowe ogólnego przeznaczenia do dostrajania serwera

Serwer MySQL ma wiele modyfikowanych zmiennych systemowych, których włączenie wpływa na jego działanie. Przykładami takich zmiennych są zmienne określające wielkość buforów. Jeżeli masz dużą ilość dostępnej pamięci, wtedy możesz użyć większych buforów dla operacji na dysku i indeksie. Przechowywanie większej ilości w pamięci znacznie zmniejsza liczbę wymaganych operacji dyskowych. W przypadku ograniczonych zasobów komputera można nakazać serwerowi użycie mniejszych buforów. To niewątpliwie spowolni działanie serwera, ale może poprawić ogólną wydajność systemu przez uniemożliwienie serwerowi MySQL wykorzystania zasobów systemowych przeznaczonych dla innych procesów.

Poniższa lista przedstawia wybrane zmienne systemowe, które są użyteczne podczas ogólnego dostrajania wydajności działania serwera MySQL:

- `back_log`

Maksymalna liczba oczekujących żądań połączenia, które mogą być kolejgowane w trakcie przetwarzania bieżących połączeń. W przypadku mocno obciążonej witryny internetowej zwiększenie wartości `back_log` może pomóc, jeśli klienci wolno nawiązują połączenia.

■ `max_connections`

Maksymalna liczba jednoczesnych połączeń z klientami obsługiwanych przez serwer. W przypadku mocno obciążonego serwera może wystąpić konieczność zwiększenia tej wartości. Na przykład, jeśli serwer MySQL jest używany przez aktywny serwer WWW do przetwarzania dużej liczby zapytań generowanych przez skrypty DBI i PHP, zbyt niska wartość wymienionej zmiennej spowoduje, że żądania odwiedzających witrynę internetową będą odrzucane. (Aby poznać największą liczbę jednocześnie otwartych połączeń, należy wykonać zapytanie `SHOW STATUS` i sprawdzić wartość zmiennej stanu `Max_used_connections`).

■ `table_open_cache`

Kiedy serwer otwiera pliki tabel, próbuje je pozostawić otwarte w celu minimalizacji koniecznych do wykonania operacji otwierania i zamykania plików. W tym celu w buforze tabeli przechowuje informacje o otwartych plikach. Jeżeli serwer uzyskuje dostęp do wielu tabel, bufor tabeli zostaje zapełniony i serwer zamyka tabele, które nie były używane przez pewien czas. W ten sposób robi miejsce na otwarcie nowych tabel.

Domyślna wielkość bufora tabeli wynosi 400 i może okazać się zbyt mała w bardzo obciążonym serwerze, w przypadku używania funkcji takich jak `innodb_file_per_table` lub tabel partycjonowanych, które zwiększają liczbę wymaganych deskryptorów plików. Jeżeli masz dostępne deskryptory plików, zwiększenie bufora tabeli przez zwiększenie wartości zmiennej systemowej `table_open_cache` pozwala `mysqld` na jednoczesne przechowywanie wielu otwartych tabel.

Aby wyświetlić efektywność bufora tabel, sprawdź wartość `Opened_tables`, wykonując poniższe zapytanie:

```
SHOW STATUS LIKE 'Opened_tables';
```

Wartość `Opened_tables` wskazuje, ile razy tabela musiała zostać otwarta, ponieważ była zamknięta w chwili, gdy była potrzebna. (Ta wartość jest wyświetlana również jako `Opens` w danych wyjściowych polecenia `mysqladmin status`). Jeżeli wartość pozostaje stała lub wzrasta powoli, to prawdopodobnie jest prawidłowa. Natomiast jeśli wzrasta szybko, oznacza to, że bufor tabeli jest zbyt mały i musi być ona często zamykana, aby zrobić miejsce dla innych tabel. Zamknięcie tabeli jest szczególnie kosztowne, gdy jej zawartość została zmodyfikowana i zmiany trzeba zapisać na dysku. Większy bufor tabel znacznie ogranicza wspomniany spadek wydajności.

■ `table_definition_cache`

Jeżeli zwiększysz wartość `table_open_cache`, rozważ także zwiększenie wartości `table_definition_cache`. Ta zmienna systemowa kontroluje wielkość bufora stosowanego do przechowywania definicji tabel (informacji pochodzących z plików *.frm*), które są używane podczas otwierania tabel.

■ `open_files_limit`

Po zwiększeniu wartości `max_connections` lub `table_open_cache` serwer będzie wymagał większej liczby deskryptorów plików. To może powodować problemy z nakładanymi przez system operacyjny ograniczeniami w postaci dozwolonej liczby deskryptorów plików używanych przez proces. W takim przypadku konieczne jest podniesienie wspomnianego limitu. Przede wszystkim, należy spróbować ustawić wartość zmiennej dla `open_files_limit`. Jeżeli w ten sposób nie da się ustawić wystarczająco dużego limitu, konieczna może okazać się zmiana w konfiguracji systemu operacyjnego, aby zezwalał na używanie większej liczby deskryptorów plików. Pewne systemy można skonfigurować po prostu przez edycję pliku systemowego i ponowne uruchomienie systemu. W przypadku innych może wystąpić konieczność edycji jądra i jego ponownej kompilacji. Sprawdź dokumentację systemu, aby dowiedzieć się, jak wygląda procedura w używanym przez Ciebie systemie.

Pewnym ominięciem problemu związanego z ograniczeniem liczby deskryptorów dostępnych dla procesu jest konfiguracja replikacji serwera głównego do jednego lub więcej serwerów podległych (zapoznaj się z podrozdziałem 14.8, zatytułowanym „Konfiguracja serwerów replikacji”). Wszystkie uaktualnienia powinny być kierowane do serwera głównego, natomiast żądania klientów dotyczące jedynie pobrania danych można rozproszyć wśród wszystkich serwerów. W ten sposób w serwerze głównym następuje zmniejszenie obciążenia i liczby wymaganych dostępnych deskryptorów plików.

■ `max_allowed_packet`

Maksymalna wielkość, jaką może osiągnąć bufor w trakcie komunikacji z klientem. Wielkością domyślną bufora jest 1 MB, a największa dozwolona wartość wynosi 1 GB. Ta zmienna nie jest parametrem zbyt mocno powiązanym z wydajnością, ale większa wartość chroni przed błędami występującymi dla dużych pakietów, a sama pamięć dla bufora pakietu jest alokowana tylko wtedy, gdy zachodzi potrzeba.

Klienci `mysql` i `mysqldump` mają własne zmienne `max_allowed_packet`. Jeżeli korzystasz z klientów wysyłających do serwera ogromne zapytania (na przykład zapytania zawierające ogromne wartości BLOB lub TEXT), może wystąpić potrzeba zwiększenia wartości tej zmiennej zarówno w serwerze, jak i kliencie. Na przykład, aby uruchomić serwer wraz z ograniczeniem wielkości pakietu do 16 MB, w pliku opcji serwera należy umieścić poniższe wiersze:

```
[mysqld]
max_allowed_packet=16M
```

Kiedy wystąpi potrzeba wywołania `mysql` lub `mysqldump` wraz z ograniczeniem wielkości pakietu do 16 MB, użyj następujących poleceń:

```
% mysql --max_allowed_packet=16M ...inne opcje...
% mysqldump --max_allowed_packet=16M ...inne opcje...
```

Aby wymienione ustawienia zawsze były używane przez klienta, w pliku opcji trzeba umieścić następujące wiersze:


```
[mysql]  
max_allowed_packet=16M
```

```
[mysqldump]  
max_allowed_packet=16M
```

Pewne zmienne kontrolują zasoby alokowane dla poszczególnych klientów. Zwiększenie wartości globalnej tego rodzaju zmiennej przekłada się na znaczne zwiększenie zapotrzebowania serwera na zasoby, w przypadku gdy jednocześnie będzie obsługiwał wiele klientów. Dwie zmienne, które mogłeś zwiększyć, licząc na poprawę wydajności, to `read_buffer_size` i `sort_buffer_size`. Wymienione zmienne określają wielkość bufora używanego w trakcie operacji odpowiednio odczytu i sortowania. Powinieneś jednak zachować ostrożność, ponieważ wspomniane bufony są alokowane dla każdej sesji. Dlatego też ustawienie wymienionym zmiennym ogromnych wartości może w rzeczywistości doprowadzić do spadku wydajności ze względu na nadmierne wykorzystanie dostępnych zasobów systemu.

Zamiast jednorazowo znacznie zwiększać wielkość buforów dla sesji, najlepszym rozwiązaniem jest krokowa zmiana wartości i sprawdzanie efektu wprowadzonych zmian. W ten sposób możesz przeanalizować efekt zmiany, a jednocześnie występuje o wiele mniejsze niebezpieczeństwo znacznego spadku wydajności działania serwera. Pamiętaj o zachowaniu realizmu podczas testowania. Serwer alokuje wspomniane bufony, gdy zachodzi potrzeba, a nie natychmiast po nawiązaniu połączenia przez klienta. Na przykład, bufor sortowania nie jest alokowany przez serwer, dopóki klient nie wykona zapytania wymagającego przeprowadzenia operacji sortowania. Zmienna `join_buffer_size` kontroluje wielkość bufora używanego podczas złączeń tabel, ale klient nieużywający złączeń tabel nie będzie miał tego bufora. (Z drugiej strony, klient wykonujący zapytania wymagające przeprowadzania skomplikowanych złączeń wielu tabel wymaga jednoczesnego użycia wielu buforów złączeń). Przeprowadzane testy powinny więc opierać się na klientach jednocześnie nawiązujących połączenia i wykonujących skomplikowane zapytania. Tylko wtedy będziesz mógł ocenić faktyczny wpływ wprowadzonej zmiany na zapotrzebowanie serwera na pamięć operacyjną.

Alternatywą dla zwiększenia wartości globalnej zmiennych określających wielkości buforów alokowanych dla poszczególnych klientów jest pozostawienie wartości bez zmian i przyjęcie założenia, że poszczególne klienty będą ustawiały wartości sesji dla zmiennych, których wartość trzeba będzie zwiększyć. Więcej informacji na temat różnic między wartościami globalnymi i sesji zmiennych systemowych przedstawiono w punkcie 12.3.1, zatytułowanym „Sprawdzanie i ustawienie wartości zmiennych systemowych”.

12.7.2. Dostrajanie silnika bazy danych

Pewne silniki bazy danych mają parametry dostrajania; zmiana ich wartości pozwala na osiągnięcie lepszych wyników w przypadku zadań wykonywanych przez konkretny serwer. W tym punkcie skoncentrujemy się na najważniejszych zasobach związanych z wydajnością silników bazy danych InnoDB i MyISAM: obsługiwanych przez poszczególne

silniki buforach przeznaczonych do przechowywania w pamięci informacji tabel. Ogólne informacje dotyczące konfiguracji silników bazy danych przedstawiono w podrozdziale 12.5, zatytułowanym „Konfiguracja silnika bazy danych”.

Bufory przeznaczone do przechowywania w pamięci informacji tabel mają różne nazwy (pula bufora w InnoDB i bufor kluczy w MyISAM), ale ogólnie działają na podstawie takich samych zasad:

- Początkowo bufor jest pusty.
- Kiedy w trakcie wykonywania zapytania serwer musi przeanalizować informacje pochodzące z tabeli, sprawdza, czy żądane informacje zostały już wcześniej odczytane i umieszczone w buforze. Jeśli tak, serwer korzysta z wartości znajdujących się w pamięci. W przeciwnym razie silnik bazy danych odczytuje z dysku zawartość tabeli i umieszcza ją w buforze.
- Jeżeli do zapełnionego bufora trzeba dodać nowe wartości, najpierw stare informacje trzeba usunąć, robiąc w ten sposób miejsce dla nowych. Domyślnie każdy silnik bazy danych wybiera bloki bufora do usunięcia na podstawie algorytmu określającego ich ostatnie użycie (ang. *Least Recently Used*, LRU). Oznacza to, że wybierane są bloki, które najdłużej nie były używane. W ten sposób aktywnie używane bloki pozostają w buforze, a usuwane są te, których prawdopodobieństwo ponownego wykorzystania jest najmniejsze.
- Jeżeli blok przeznaczony do usunięcia nie został zmodyfikowany, jego zawartość zostaje nadpisana nową wartością. W przeciwnym razie blok trzeba zapisać z powrotem w tabeli, z której pochodzi, i dopiero wtedy będzie mógł zostać nadpisany.

Wartości nieznajdowane w buforze, gdy są żądane, określamy mianem „brakujących” i muszą być odczytane z dysku. Z kolei wartości znalezione w buforze są określane mianem „trafionych”. Celem stosowania bufora jest zmniejszenie liczby operacji dyskowych, czyli minimalizacja współczynnika braków względem żądań. Dzięki temu znacznie wzrasta wydajność, ponieważ dostęp do elementów znajdujących się w pamięci jest znacznie szybszy niż do elementów na dysku. Buforowanie powoduje zmniejszenie liczby operacji wejścia-wyjścia zarówno dla odczytu, jak i zapisu:

- Mniejsza liczba operacji odczytu, ponieważ informacje są znajdowane w buforze, a więc nie trzeba ich ponownie odczytywać z dysku.
- Mniejsza liczba operacji zapisu, ponieważ zmodyfikowane informacje w buforze nie są natychmiast zapisywane na dysku. Serwer buforuje wiele wprowadzonych zmian, ponieważ ich jednoczesny zapis w postaci pojedynczej operacji jest znacznie efektywniejszy niż zapis po każdej wprowadzonej zmianie.

Nawet niewielki bufor zapewnia wymienione powyżej korzyści, większy bufor jeszcze bardziej je podkreśla. Bloki zawierające często używane wartości indeksu pozostają w buforze, a większy bufor znacznie zwiększa prawdopodobieństwo trafienia. To z kolei zmniejsza prawdopodobieństwo konieczności usuwania bloków dla nowych wartości i minimalizuje liczbę operacji dyskowych wymaganych do przetwarzania indeksu. Jeżeli

serwer zarządza wieloma tabelami dla danego silnika, a aktualny bufor silnika jest niewielki, jego zwiększenie to w rzeczywistości jedna z najszybciej zmieniających się konfiguracyjnych, jakie możesz wprowadzić.

Jednak określenie wielkości bufora wymaga rozważań. Nie chcesz przecież doprowadzić do sytuacji, w której pamięci zabraknie dla innych procesów działających w komputerze, a także na inne potrzeby samego serwera MySQL, na przykład dla buforów połączeń, buforów operacji sortowania i złączeń, tabel tymczasowych oraz bufora zapytań. Ponadto, podczas zwiększania bufora silnika bazy danych pamiętaj, aby nie zwiększyć go za bardzo i nie wykorzystać całej pamięci dostępnej w komputerze. To może doprowadzić do umieszczenia w pamięci wirtualnej na dysku samego bufora, co niweczy cel stosowania bufora w celu przechowywania informacji w pamięci operacyjnej.

W poniższych podpunktach przedstawione będą charakterystyczne dla poszczególnych silników bazy danych (InnoDB i MyISAM) informacje o konfiguracji buforów przechowujących informacje z tabel.

12.7.2.1. Konfiguracja puli bufora InnoDB

InnoDB traktuje pulę bufora jako listę bloków podzieloną na podlisty nową i starą, zawierające bloki używane odpowiednio ostatnio i bardzo dawno. Silnik InnoDB stosuje następującą strategię wstawiania: nowe bloki są umieszczane w puli na początku podlisty zawierającej stare bloki, czyli na końcu podlisty z nowymi blokami.

Bloki są umieszczane w puli, gdy zapytanie potrzebuje zawartych w nich informacji lub jeśli algorytm odczytu z wyprzedzeniem przewiduje, że wkrótce będą potrzebne. Wstawienie bloku nie oznacza uzyskania do niego dostępu. Faktyczne uzyskanie dostępu do bloku następuje wtedy, gdy zapytanie rzeczywiście analizuje zawartość bloku.

Domyślnie InnoDB wstawia blok na początku podlisty starych wartości, a następnie przenosi go na początek podlisty nowych, gdy blok zostanie użyty po raz pierwszy po jego wstawieniu. Pierwsze użycie następuje natychmiast, jeśli operacja wstawienia została przeprowadzona z powodu wymagań zapytania. Może również nastąpić znacznie później (lub w ogóle nie wystąpić), jeśli blok został wstawiony na skutek działania algorytmu odczytu z wyprzedzeniem.

Uzyskanie dostępu do bloku w podliście nowych wartości powoduje jego przeniesienie na początek listy tylko wtedy, jeśli nie znajduje się blisko jej początku. Wraz z upływem czasu i nieużywaniem danego bloku jest on przenoszony w kierunku końca listy. Po osiągnięciu pewnego wieku bloki znajdujące się na podliście nowych wartości przekraczają granicę rozdzielającą podlisty nowych i starych wartości. Z kolei bloki znajdujące się na końcu podlisty starych wartości są ostatecznie usuwane.

Dwie zmienne systemowe kontrolują wielkość puli bufora InnoDB i decydują o jej podziale na mniejsze pule:

- `innodb_buffer_pool_size`

Wyrażona w bajtach całkowita ilość pamięci alokowana dla puli bufora.

Wartością domyślną jest 128 MB. W systemach posiadających odpowiednią ilość wolnej pamięci i intensywnie przetwarzających tabele InnoDB warto

zwiększyć wartość tej zmiennej. Ogólna reguła zaleca przydzielenie 70 – 80% pamięci systemowej, ale pod uwagę trzeba wziąć także inne potrzeby serwera MySQL oraz pozostałych procesów działających w komputerze.

■ `innodb_buffer_pool_instances`

Domyślnie wartość tej zmiennej wynosi 1 i oznacza istnienie pojedynczej puli bufora. Jeżeli wartość zmiennej `innodb_buffer_pool_size` wynosi co najmniej 1 GB, wtedy zmiennej `innodb_buffer_pool_instances` można przypisać wartość większą niż 1. Silnik InnoDB traktuje pulę jako wskazaną liczbę mniejszych pul działających niezależnie od siebie. InnoDB w losowy sposób przypisuje danej puli bloki odczytane z dysku, co zmniejsza rywalizację, ponieważ mniejsza liczba sesji próbuje uzyskać dostęp do współdzielonych struktur danych. (Na przykład, nawet jeśli jedna sesja nałoży na pulę blokadę na wyłączność, zadania wykorzystujące pozostałe pule mogą być wykonywane). W celu uzyskania najlepszych wyników należy dla zmiennych `innodb_buffer_pool_size` i `innodb_buffer_pool_instances` wybrać takie wartości, aby każdy egzemplarz puli miał wielkość przynajmniej 1 GB. Użycie wielu pul jest zwykle stosowane w dużych systemach, na przykład 64-bitowych systemach wyposażonych w ogromne ilości pamięci.

Dwie dodatkowe zmienne systemowe wpływają na algorytm LRU w puli bufora InnoDB (dla każdego egzemplarza puli, o ile konfigurujesz wiele egzemplarzy):

■ `innodb_old_blocks_pct`

Procentowa wielkość bufora przeznaczona na podlistę starych wartości. Wartość procenta jest tylko przybliżona, ponieważ InnoDB zarządza pewnymi operacjami po prostu przez przenoszenie wskaźnika granicy do przodu i do tyłu (na przykład w trakcie przejścia bloku z podlisty nowych wartości do starych). Domyślnie to 37% (3/8 puli). Dozwolone jest użycie wartości z zakresu od 5 do 95%. Mniejsze lub większe wartości powodują, że bloki na podliście nowych przechodzą do podlisty starych odpowiednio znacznie wolniej lub szybciej.

■ `innodb_old_blocks_time`

Liczba milisekund, które muszą upłynąć, zanim blok wstawiony do podlisty starych wartości po jego pierwszym użyciu zostanie tam przed przeniesieniem do podlisty nowych w przypadku jego kolejnego użycia. Wartość domyślna wynosi zero. Bloki wstawione do podlisty starych wartości są natychmiast przenoszone do podlisty nowych po ich pierwszym użyciu.

Operacja skanowania tabeli zwykle wielokrotnie w krótkich odstępach czasu uzyskuje dostęp do bloków, a później w ogóle. W celu uniemożliwienia przeniesienia takich jednorazowo użytych bloków do podlisty nowych i usunięcia z niej znajdujących się tam wartości należy zmiennej `innodb_old_blocks_time` ustawić wartość większą niż zero. Wartość tej zmiennej warto ustawić podczas wykonywania kopii zapasowej, w trakcie której przeprowadzanych jest wiele jednorazowych operacji skanowania tabel. Z drugiej strony, w celu umieszczenia w puli bufora zawartości tabeli, która ma być często używana,

należy przypisać zmiennej `innodb_old_blocks_time` wartość zero. W takim przypadku operacja skanowania spowoduje umieszczenie zawartości tabeli w buforze i wtedy wymienionej zmiennej można przywrócić jej poprzednią wartość.

12.7.2.2. Konfiguracja bufora kluczy MyISAM

Silnik MyISAM przechowuje tabele w oddzielnych plikach danych i indeksów, które są obsługiwane odmiennie:

- Aby buforować rekordy danych odczytane lub zapisane do plików danych, MyISAM opiera się na systemie operacyjnym wykorzystującym własny mechanizm buforowania systemu plików.
- Aby przetworzyć pliki indeksów, MyISAM zawiera bufor kluczy używany podczas operacji pobierania danych na podstawie indeksów, operacji sortowania, a także operacji tworzenia i modyfikacji indeksu.

Najważniejszy parametr wpływający na działanie silnika bazy danych MyISAM to wielkość jego bufora kluczy. Wartość domyślna wynosi 8 MB i jest dość niska. Jeżeli intensywnie korzystasz z tabel MyISAM i masz dostępną pamięć operacyjną, zwiększenie bufora kluczy powinno znacznie usprawnić wydajność działania silnika MyISAM w trakcie operacji związanych z przetwarzaniem indeksu.

W celu ustawienia wielkości bufora należy przypisać wartość zmiennej systemowej o nazwie `key_buffer_size`. Na przykład, jeśli wielkość bufora ma wynosić 512 MB, w pliku opcji należy umieścić poniższe wiersze:

```
[mysqld]  
key_buffer_size=512M
```

Domyślnie bufor kluczy MyISAM jest współdzielony przez wszystkie tabele MyISAM. Jeżeli wartość klucza nie zostanie znaleziona w buforze, a sam bufor jest pełny, wtedy dochodzi do rywalizacji: pewne wartości w buforze muszą zostać usunięte i ponownie wczytane z dysku, gdy po raz kolejny będą potrzebne.

Jeżeli masz mocno używaną tabelę MyISAM, to przechowywanie jej kluczy w pamięci będzie niezwykle korzystne, choć stan rywalizacji w buforze jest wadą takiego rozwiązania. Wspomniany stan rywalizacji może wystąpić, gdy zachodzi konieczność wczytania kluczy z tej samej tabeli lub z innych tabel. Istnieje możliwość uniknięcia rywalizacji w tej samej tabeli przez utworzenie bufora na tyle dużego, aby w pełni pomieścił wszystkie indeksy danej tabeli. Jednak w takim przypadku klucze z innych tabel nadal będą rywalizowały o miejsce w buforze.

Silnik MyISAM oferuje rozwiązanie tego problemu, ponieważ obsługuje wiele buforów kluczy, co zapewnia lepszą kontrolę nad buforowaniem dzięki wykorzystaniu następujących cech:

- możliwość użycia domyślnego pojedynczego bufora kluczy lub utworzenia wielu buforów;

- zachowanie kontroli nad całkowitą wielkością bufora, wielkością bloku bufora oraz algorytmem opróżniania bufora;
- przypisanie tabel do konkretnych buforów i wczytywanie do nich indeksów tabel.

Wymienione możliwości mogą być użyteczne, jeśli masz intensywnie wykorzystywaną tabelę oraz ilość pamięci wystarczającą do wczytania wszystkich indeksów do bufora. Istnieje możliwość uniknięcia stanu rywalizacji zarówno w tej samej tabeli, jak i w różnych: należy utworzyć bufor na tyle duży, aby w pełni pomieścił wszystkie indeksy tabeli, a sam bufor przeznaczyć wyłącznie dla danej tabeli. Po wczytaniu zawartości tabeli do bufora użycie jej zawartości nie wymaga żadnych operacji wejścia-wyjścia. Ponadto, wartości kluczy nigdy nie zostaną usunięte z bufora, a operacje wyszukiwania kluczy będą mogły być przeprowadzane w pamięci.

Inna często stosowana strategia polega na przypisaniu intensywnie używanym tabelom oddzielnych buforów kluczy, aby ich buforów indeksów nie rywalizowały z innymi tabelami, które są przetwarzane z wykorzystaniem bufora domyślnego.

W celu umożliwienia konfiguracji bufora kluczy MyISAM serwer powiązuje każdy bufor z zestawem zmiennych systemowych. Ponieważ wspomniane zmienne są ze sobą powiązane, zostały zgrupowane jako komponenty tworzące strukturalnych zmiennych systemowych. Strukturalne zmienne systemowe stanowią rodzaj rozszerzenia prostych zmiennych systemowych. Dostęp do nich jest uzyskiwany za pomocą składni łączącej nazwę bufora i zmiennej:

nazwa_bufora.nazwa_zmiennej

Każda zmienna strukturalna bufora kluczy składa się z wymienionych poniżej komponentów:

- **key_buffer_size**
Wyrażona w bajtach całkowita wielkość bufora kluczy.
- **key_cache_block_size**
Wyrażona w bajtach wielkość bloku bufora kluczy. Domyślna wielkość bloku wynosi 1024 bajty.
- **key_cache_limit**
Ta zmienna wpływa na algorytm ponownego użycia bufora kluczy. Jeżeli będzie miała ustawioną wartość domyślną wynoszącą 100, bufor klucza używa strategii LRU w celu ustalenia buforów, które mają zostać ponownie wykorzystane. Wartość mniejsza niż 100 powoduje, że bufor kluczy używa strategii wstawiania pośrodku, dzieląc w ten sposób bufor na części nowych kluczy i starych. Wartość zmiennej `key_cache_limit` to wartość procentowa wskazująca, jaka część bufora kluczy będzie przeznaczona do przechowywania nowych kluczy. Wartość może być z zakresu od 1 do 100.
W przypadku strategii wstawiania kluczy pośrodku bufora najczęściej używane bloki bufora serwer próbuje utrzymywać w części zawierającej nowe wartości.

Bloki mogą być przenoszone między częściami bufora w zależności od tego, czy częstotliwość ich użycia wzrasta czy też maleje. Serwer zawsze wybiera rozwiązanie polegające na ponownym użyciu bloków i nadpisaniu ich z części bufora zawierającej stare wartości. (Podział bufora kluczy MyISAM przypomina istniejące w puli bufora InnoDB podlisty nowych i starych wartości).

■ `key_cache_age_threshold`

Ta wartość określa czas, przez jaki nieużywane bloki pozostaną w części bufora zawierającej nowe wartości, zanim zostaną przeniesione do części bufora zawierającej stare wartości. Wyższa wartość pozwala blokom na dłuższe pozostanie w części zawierającej nowe wartości. Dla tej zmiennej wartością domyślną jest 300, natomiast minimalną 100.

Domyślnie istnieje jeden bufor kluczy i nosi on nazwę `default`. Odniesienia do zmiennej komponentu bufora kluczy bez podania nazwy bufora dotyczą bufora domyślnego. Dlatego też odwołania `key_buffer_size` i `default.key_buffer_size` prowadzą do tej samej zmiennej. Nazwy bufora kluczy muszą być prawidłowymi identyfikatorami i nie rozróżniają wielkości liter. Podobnie jak inne identyfikatory, można je cytować (patrz podrozdział 2.2, zatytułowany „Identyfikatory składni MySQL i reguły nadawania nazw”).

W celu utworzenia nowego bufora kluczy należy przypisać wartość dowolnemu komponentowi powiązanemu ze zmienną bufora. Na przykład, aby podczas uruchamiania serwera utworzyć bufor o nazwie `my_cache` i wielkości 24 MB, do pliku opcji serwera dodaj poniższe wiersze:

```
[mysqld]  
my_cache.key_buffer_size=24M
```

W celu utworzenia bufora w trakcie działania serwera trzeba wykonać poniższe zapytanie:

```
SET GLOBAL my_cache.key_buffer_size = 24*1024*1024;
```

Kwalifikator `GLOBAL` jest konieczny, ponieważ bufor kluczy są globalne. Nie są wymagane żadne szczególne uprawnienia, aby uzyskać dostęp do wartości komponentu. Jednak do ich ustawienia konieczne jest posiadanie uprawnienia `SUPER`.

W celu przypisania tabel `MyISAM` do bufora kluczy po jego utworzeniu należy wykonać zapytanie `CACHE INDEX`. Tego rodzaju zapytanie musi zawierać nazwę bufora kluczy oraz jedną lub więcej przypisanych do niego tabel. Przedstawione poniżej zapytanie przypisuje tabele `member` i `president` do bufora o nazwie `my_cache`:

```
CACHE INDEX member, president IN my_cache;
```

Istnieje możliwość późniejszego przypisania innych tabel do tego samego bufora. Indeksy tabel można wczytać do przypisanych im buforów, wykorzystując do tego zapytanie `LOAD INDEX INTO CACHE`:

```
LOAD INDEX INTO CACHE member, president;
```

Nie ma potrzeby wczytywania indeksów do tabel, ale jeśli się zdecydujesz na takie rozwiązanie, to serwer będzie po kolei odczytywał bloki indeksu. To znacznie efektywniejsze rozwiązanie niż oczekiwanie na ich pobranie.

Zapytanie `LOAD INDEX INTO CACHE` wymaga uprawnień `INDEX` do wymienionych tabel.

W celu usunięcia bufora kluczy należy ustawić jego wielkość na zero. Wszystkie tabele powiązane z buforem zostaną przypisane do bufora domyślnego. Jeżeli buforowi domyślnemu ustawisz wielkość zero, przypisane do niego tabele będą przetwarzane z użyciem bufora systemu plików, dokładnie w ten sam sposób jak pliki danych MyISAM. Takie rozwiązanie minimalizuje zapotrzebowanie serwera na pamięć operacyjną, ale jednocześnie wiąże się ze spadkiem wydajności.

Przypisania bufora kluczy obowiązują tylko do zamknięcia serwera. Aby obowiązywały po jego każdym uruchomieniu, należy umieścić odpowiednie zapytania `CACHE INDEX` i `LOAD INDEX INTO CACHE` w pliku, a następnie uruchamiać serwer wraz z ustawioną zmienną systemową `init_file` wskazującą nazwę utworzonego pliku.

12.7.3. Używanie bufora zapytań

Serwer MySQL może używać bufora zapytań w celu przyspieszenia przetwarzania regularnie wykonywanych zapytań `SELECT`. Wzrost wydajności bardzo często jest wręcz ogromny. Bufor zapytań charakteryzuje się następującymi cechami:

- Po pierwszym wykonaniu zapytania `SELECT` serwer zapamiętuje jego treść oraz zwrócony wynik.
- Po kolejnym napotkaniu tego samego zapytania serwer nie wykonuje go. Zamiast tego pobiera wynik bezpośrednio z bufora zapytań i zwraca go klientowi.
- Bufor zapytań opiera się na dosłownych ciągach tekstowych zapytań, które zostały wysłane do serwera. Zapytania są uznawane za takie same, jeśli ich tekst jest dokładnie taki sam. Zapytania są uznawane za inne, jeśli różnią się wielkością liter, pochodzą od klientów używających innego kodowania znaków lub innego protokołu komunikacyjnego. Ponadto, są uznawane za inne, jeśli pozostają identyczne, ale w rzeczywistości nie odwołują się do tych samych tabel, na przykład odwołują się do tabel o identycznych nazwach, ale umieszczonych w różnych bazach danych.
- Zapytanie nie jest buforowane, jeśli zwraca niedeterministyczny wynik. Na przykład, zapytanie używające funkcji `NOW()` zwraca za każdym razem inny wynik i nie może być buforowane.
- Po modyfikacji tabeli wszelkie buforowane zapytania, które się do niej odwołują, zostają unieważnione i usunięte z bufora. To rodzaj zabezpieczenia przed zwróceniem nieaktualnych danych przez serwer.

W celu ustalenia, czy używany przez Ciebie serwer obsługuje bufor zapytań, sprawdź wartość zmiennej systemowej `have_query_cache`:


```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
```

```
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_query_cache | YES   |
+-----+-----+
```

W przypadku serwera zapewniającego obsługę bufora zapytań operacje bufora opierają się na wartościach trzech zmiennych systemowych:

- `query_cache_type`. Ta zmienna określa tryb działania bufora zapytań. Wartości, które można jej przypisać, wymieniono w tabeli 12.5.

Tabela 12.5. Wartości, które można przypisać zmiennej `query_cache_type`

Tryb	Opis
0	Nie buforuj wyników zapytania lub nie pobieraj buforowanych wyników zapytania.
1	Buforuj zapytania, które można buforować, za wyjątkiem rozpoczynających się od <code>SELECT SQL_NO_CACHE</code> .
2	Na żądanie buforuj zapytania, które można buforować, za wyjątkiem rozpoczynających się od <code>SELECT SQL_CACHE</code> .

- `query_cache_size`. Wartość tej zmiennej wskazuje wyrażoną w bajtach ilość pamięci alokowanej dla bufora.
- `query_cache_limit`. Wartość tej zmiennej wskazuje maksymalną wielkość buforowanego wyniku. Wyniki zapytań większe niż wartość tej zmiennej nigdy nie będą buforowane.

Na przykład, w celu włączenia bufora zapytań o wielkości 16 MB należy w pliku opcji użyć poniższych ustawień:

```
[mysqld]
query_cache_type=1
query_cache_size=16M
```

Ilość pamięci wskazywana przez zmienną `query_cache_size` jest alokowana nawet wtedy, gdy wartość `query_cache_type` wynosi zero. Aby uniknąć marnowania pamięci, ustawiaj wartość zero, o ile nie planujesz włączenia bufora. Zwróć uwagę, że wartość zero w praktyce oznacza wyłączenie bufora, nawet jeśli wartość zmiennej `query_cache_type` jest niezerowa.

Włączenie bufora zapytań wydaje się łatwym sposobem na poprawę wydajności i często tak jest. Istnieją jednak pewne sytuacje, w których takie rozwiązanie może nie tylko nie pomóc, ale jeszcze zaszkodzić:

- Alokacja ogromnej ilości pamięci dla bufora zapytań może spowodować, że serwer będzie poświęcał dużą ilość czasu na porównywanie zapytania bieżącego z buforowanymi w celu ustalenia, czy jego wynik jest już buforowany.

- W systemach wieloprocesorowych serwer MySQL może za pomocą różnych rdzeni procesora wykonywać zapytania pochodzące z różnych połączeń. Jednak sprawdzenie bufora zapytań odbywa się przez jeden wątek, co może stać się źródłem rywalizacji między wątkami działającymi w różnych rdzeniach. Jednym ze sposobów sprawdzenia, czy taka sytuacja występuje, jest wykonanie zapytania `SHOW PROCESSLIST`. Jeżeli w kolumnie `State` często znajdujesz komunikat `Waiting for query cache lock`, oznacza to, że występuje rywalizacja o dostęp do bufora zapytań i lepszym rozwiązaniem może być jego wyłączenie.

Przyjmując założenie, że bufor zapytań jest włączony, poszczególne klienty rozpoczynają pracę wraz z zachowaniem bufora zapytań wskazanym przez domyślny tryb buforowania w serwerze. Dzięki wykonaniu poniższego zapytania klient ma możliwość zmiany domyślnego trybu buforowania zapytań:

```
SET query_cache_type = wartość;
```

Parametrem *wartość* może być 0, 1 lub 2, co ma takie samo znaczenie jak ustawienie zmiennej `query_cache_type` podczas uruchamiania serwera. W zapytaniu `SET` wartości symboliczne `OFF`, `ON` i `DEMAND` są synonimami dla 0, 1 i 2.

Klient może również zachować kontrolę na buforowaniu poszczególnych zapytań, dodając modyfikator po słowie kluczowym `SELECT`. Użycie `SELECT SQL_CACHE` w przypadku zapytania, którego wyniki mogą być buforowane, powoduje, że jego wynik zostanie umieszczony w buforze, o ile tryb bufora to `ON` lub `DEMAND`. Z kolei `SELECT SQL_NO_CACHE` powoduje, że wynik zapytania nie będzie buforowany.

Zawieszenie buforowania może być użyteczne w przypadku zapytań pobierających informacje z nieustannie zmieniającej się tabeli. W takim przypadku jest mało prawdopodobne, aby bufor okazał się przydatny. Przyjmujemy założenie, że żądania serwera WWW są rejestrowane w tabeli MySQL, a co pewien czas wykonujesz zestaw zapytań podsumowujących tabelę. W przypadku bardzo obciążonego serwera WWW nowe rekordy są nieustannie wstawiane do tabeli, a tym samym buforowane wyniki zapytań bardzo szybko okazują się nieaktualne. W takim przypadku, pomimo regularnego wykonywania zapytania tworzącego podsumowanie, jest bardzo prawdopodobne, że wyniki zapytania nie będą pochodziły z bufora. W takich sytuacjach rozsądne jest wykonywanie zapytań wraz z modyfikatorem `SQL_NO_CACHE` w celu poinformowania serwera, że nie powinien sprawdzać bufora pod kątem istnienia w nim wyniku zapytania.

12.7.4. Optymalizacje sprzętowe

We wcześniejszej części rozdziału przedstawiono techniki, które mogły poprawić wydajność działania serwera niezależnie od jego konfiguracji sprzętowej. Oczywiście, można również kupić lepszy sprzęt i zapewnić w ten sposób szybsze działanie serwera. Jednak nie zawsze zmiany wprowadzane w sprzęcie mają taką samą wartość. Podczas przeprowadzania usprawnień w sprzęcie serwera najważniejsze zasady są takie same jak w przypadku dostrajania parametrów serwera: informacje należy umieszczać w jak najszybszej pamięci masowej i przechowywać je tam jak najdłużej.

Poniżej wymieniono kilka aspektów konfiguracji sprzętu, które można zmodyfikować, aby w ten sposób poprawić wydajność serwera:

Instalacja większej ilości pamięci w komputerze. Dzięki temu będzie można skonfigurować większe bufony serwera, co pozwoli na dłuższe przechowywanie danych w pamięci i rzadsze pobieranie informacji z dysku.

Użycie komputera wieloprocesorowego. W przypadku aplikacji wielowątkowych, takich jak serwer MySQL, komputer wieloprocesorowy może jednocześnie wykonywać wiele wątków.

Ponowna konfiguracja systemu w celu usunięcia przestrzeni wymiany. Takie rozwiązanie staje się możliwe, jeśli masz wystarczającą ilość pamięci RAM, aby wszystkie operacje związane z przestrzenią wymiany mogły być przeprowadzane w systemie plików w pamięci. W przeciwnym razie pewne systemy nadal będą w tym celu używały dysku, nawet jeśli masz wystarczającą ilość pamięci RAM.

Dodanie szybszych dysków, aby poprawić operacje wejścia-wyjścia.

W przypadku standardowych, mechanicznych dysków twardych największy wpływ na wydajność ma czas wyszukiwania. Przesuwanie głowicy jest wolne, natomiast po jej umieszczeniu we właściwym miejscu odczyt bloków ze ścieżki odbywa się bardzo szybko. Jednak jeśli musisz wybierać między dodaniem większej ilości pamięci a zamontowaniem szybszych dysków, zdecyduj się na pamięć. Operacje w pamięci zawsze będą szybsze niż na dyskach, a dzięki dodaniu pamięci zyskujesz możliwość użycia większych buforów, co zmniejszy liczbę potrzebnych operacji dyskowych.

Aby uniknąć wyszukiwania, możesz zainstalować napędy SSD (ang. *Solid State Drive*) i skonfigurować MySQL do ich używania. W przypadku napędów SSD praktycznie nie występuje opóźnienie związane z czasem wyszukiwania informacji, ponieważ nie mają ruchomych części.

Wykorzystanie zalet wynikających z przetwarzania równoległego przez rozłożenie aktywności dyskowej między oddzielnymi fizycznymi napędami.

Jeżeli operacje odczytu i zapisu można rozłożyć między wiele fizycznych urządzeń, odczyt i zapis czegokolwiek będzie odbywał się znacznie szybciej niż w przypadku ich przeprowadzania w jednym napędzie. Na przykład, jeżeli bazy danych przechowujesz w jednym napędzie, a dzienniki zdarzeń w innym, jednoczesny zapis danych w obu urządzeniach będzie przeprowadzony szybciej niż gdyby baza danych i dzienniki zdarzeń znajdowały się w jednym napędzie. Zwróć uwagę, że użycia różnych partycji w tym samym napędzie nie można uznać za przetwarzanie równoległe. Takie rozwiązanie nie pomoże, ponieważ partycje nadal będą rywalizowały o te same zasoby fizyczne (tutaj głowice dysku twardego). Procedura przenoszenia plików dzienników zdarzeń i baz danych została omówiona w podrozdziale 11.3, zatytułowanym „Przeniesienie zawartości katalogu danych”.

Zanim rozpoczniesz przenoszenie danych do innego urządzenia, upewnij się, że rozumiesz charakterystykę obciążenia w Twoim systemie. Jeśli w określonym napędzie będzie wykonywana inna ważna operacja, umieszczenie w nim bazy danych może w rzeczywistości doprowadzić do spadku wydajności. Na przykład, możesz nie odnieść żadnych korzyści, jeśli serwer WWW przetwarza ogromny ruch sieciowy, a Ty przeniesiesz bazę danych do urządzenia, w którym znajduje się drzewo dokumentów serwera WWW.

Użycie urządzeń macierzy RAID również może przynieść pewne korzyści z przetwarzania równoległego.

12.8. Dzienniki zdarzeń serwera

Serwer MySQL może generować wiele rodzajów dzienników zdarzeń. Są one użyteczne podczas diagnozowania problemów, poprawiania wydajności działania serwera, włączania replikacji oraz naprawy po wystąpieniu awarii. W trakcie uruchamiania serwera następuje analiza opcji startowych w celu sprawdzenia, czy zdarzenia mają być zapisywane w dziennikach, i ewentualne otworzenie odpowiednich plików dzienników zdarzeń. Poniższa lista wymienia poszczególne dzienniki zdarzeń, natomiast ich dokładniejsze omówienie znajdziesz w kolejnych kilku punktach:

- **Dziennik błędów.** W tym dzienniku są rejestrowane zdarzenia związane z uruchamianiem i zamykaniem serwera, a także komunikaty o problemach i sytuacjach wyjątkowych. Do tego dziennika powinieneś zajrzeć, jeśli próba uruchomienia serwera kończy się niepowodzeniem. Przed ostatecznym zamknięciem serwera zapisuje on komunikat w dzienniku błędów, wskazując tym samym problem, który wystąpił.
- **Ogólny dziennik zapytań.** W tym dzienniku są rejestrowane połączenia z klientami, zapytania SQL otrzymane od klientów oraz różne inne zdarzenia. Ten dziennik jest użyteczny do monitorowania aktywności serwera: kto nawiązuje połączenie, skąd i jakie podejmuje działania. To najwygodniejszy dziennik zdarzeń, gdy chcesz się dowiedzieć, jakie zapytania klienty wysyłają do serwera. Ta wiedza może być bardzo użyteczna w trakcie rozwiązywania problemów i usuwania błędów.
- **Dziennik wolno wykonywanych zapytań.** Pomaga w wykrywaniu zapytań wymagających zmodyfikowania, aby zapewniły lepszą wydajność w trakcie wykonywania. Domyślnie, jeśli wykonanie zapytania zajmuje więcej niż 10 sekund czasu rzeczywistego, jest ono uznawane za wolno wykonywane i serwer umieszcza je w tym dzienniku. Kilka zmiennych systemowych daje dodatkową kontrolę nad zapytaniami, o których informacje będą zapisywane w tym dzienniku.
- **Binarny dziennik zdarzeń i plik indeksu binarnego dziennika zdarzeń.** Ten dziennik składa się z jednego lub więcej plików rejestrujących modyfikacje wprowadzone przez zapytania UPDATE, DELETE, INSERT, CREATE TABLE, DROP TABLE, GRANT itd. Zawartość binarnego dziennika jest zapisywana w postaci

zakodowanych w formacie binarnym „zdarzeń” modyfikujących dane. Plikom dziennika binarnego towarzyszą pliki indeksu zawierające listę aktualnie istniejących plików binarnego dziennika zdarzeń.

Binarny dziennik zdarzeń istnieje w dwóch podstawowych celach:

- ◆ Może być używany w połączeniu z kopią zapasową w celu przywrócenia tabel po wystąpieniu awarii. Najpierw przywracasz bazy danych z plików kopii zapasowej. Następnie używasz narzędzia `mysqlbinlog` w celu konwersji zawartości binarnego dziennika zdarzeń na zapytania tekstowe. Pozostało już wykonanie wszystkich zapytań modyfikujących bazę danych od chwili utworzenia ostatniej kopii zapasowej. Wspomniane zapytania trzeba przekazać jako dane wejściowe klientowi `mysql`. W ten sposób baza danych zostanie przywrócona do stanu, w jakim znajdowała się w chwili awarii.
 - ◆ Może stanowić podstawę dla mechanizmu replikacji. Przechowywane w binarnym dzienniku zdarzenia modyfikujące dane są przekazywane do podległych serwerów replikacji.
- **Dziennik przekazywania i plik indeksu dziennika przekazywania.** Jeżeli serwer jest serwerem replikacji, to zawiera tak zwany dziennik przekazywania, zawierający otrzymane z serwera głównego i przeznaczone do wykonania rekordy zdarzeń modyfikujących dane. Pliki dziennika przekazywania mają taki sam format jak binarny dziennik zdarzeń. Ponadto, istnieje plik indeksu zawierający wymienione aktualnie istniejące w serwerze podległym pliki dziennika przekazywania.

Spośród wszystkich wymienionych dzienników zdarzeń, ogólny dziennik zdarzeń jest najbardziej użyteczny do monitorowania serwera. Dlatego też po rozpoczęciu używania MySQL zalecam jego włączenie poza innymi, które mają być tworzone przez serwer. Po zdobyciu pewnego doświadczenia w pracy z MySQL będziesz mógł wyłączyć ogólny dziennik zdarzeń i tym samym zmniejszyć zapotrzebowanie serwera na pamięć masową.

Domyślnie każdy włączony dziennik zapisuje zdarzenia w pliku (lub sekwencji plików) znajdującym się w katalogu danych. W przypadku pewnych dzienników można zdecydować o rejestracji zdarzeń w innych miejscach:

- Zdarzenia dziennika błędów mogą być wysyłane do `syslog`.
- Zdarzenia dziennika ogólnego i wolno wykonywanych zdarzeń mogą być zapisywane w tabelach bazy danych `mysql`.

Poza wymienionymi poniżej wyjątkami serwer nie tworzy dzienników zdarzeń, o ile wyraźnie mu tego nie zlecisz:

- W systemach UNIX uruchomienie serwera za pomocą skryptu `mysqld_safe` powoduje włączenie dziennika błędów i jego użycie przez serwer.
- W systemach Windows serwer tworzy dziennik błędów, o ile nie użyjesz opcji `--console` w celu wyświetlania informacji diagnostycznych w oknie konsoli zamiast zapisywania ich w pliku.

W celu włączenia rejestrowania zdarzeń przez serwer należy skorzystać z opcji wymienionych w tabeli 12.6. Jeżeli nazwa pliku dziennika jest opcjonalna (na co wskazuje nawias kwadratowy), nie musisz jej podawać, a serwer użyje nazwy domyślnej i zapisze plik dziennika zdarzeń w katalogu danych MySQL. Nazwy poszczególnych plików dzienników serwer ustala na podstawie nazwy komputera przedstawianej w poniższej analizie jako *HOSTNAME*. W przypadku podania nazwy pliku dziennika jako względnej ścieżki dostępu serwer będzie ją interpretował względem katalogu danych MySQL. Aby umieścić plik dziennika zdarzeń w innym katalogu, konieczne jest podanie bezwzględnej ścieżki dostępu. Jeżeli plik dziennika jeszcze nie istnieje, serwer go utworzy. Jednak serwer nie tworzy katalogu, w którym ma zostać umieszczony wskazany plik dziennika zdarzeń. Dlatego też odpowiednie katalogi musisz utworzyć samodzielnie przed uruchomieniem serwera, w przeciwnym razie próba uruchomienia serwera zakończy się niepowodzeniem.

Tabela 12.6. Opcje pozwalające serwerowi MySQL na używanie dzienników zdarzeń

Opcja	Opis
--general_log	Włącza ogólny dziennik zdarzeń.
--general_log_file= <i>nazwa_pliku</i>	Nazwa pliku ogólnego dziennika zdarzeń.
--log-bin[= <i>nazwa_pliku</i>]	Włączenie binarnego dziennika zdarzeń.
--log-bin-index= <i>nazwa_pliku</i>	Plik indeksu binarnego dziennika zdarzeń.
--log_error[= <i>nazwa_pliku</i>]	Włączenie dziennika błędów.
--log_output[= <i>destination</i>]	Miejsce umieszczenia dzienników zdarzeń ogólnego i wolno wykonywanych zapytań.
--relay-log[= <i>nazwa_pliku</i>]	Włączenie dziennika zdarzeń przekazywania.
--relay-log-index= <i>nazwa_pliku</i>	Plik indeksu dziennika zdarzeń przekazywania.
--slow_query_log	Włączenie dziennika wolno wykonywanych zapytań.
--slow_query_log_file= <i>nazwa_pliku</i>	Nazwa pliku dziennika wolno wykonywanych zapytań.

W rzeczywistości, pewne opcje wymienione w tabeli 12.6 są zmiennymi systemowymi (na co wskazują znaki podkreślenia w ich nazwach), ale można je ustawić podczas uruchamiania serwera. Wspomniane zmienne można również zmienić w trakcie działania serwera.

W wierszu poleceń, uruchamiając serwer za pomocą skryptu `mysqld` lub `mysqld_safe`, można użyć opcji dotyczących rejestrowania zdarzeń przez serwer. Jednak ponieważ zwykle te same opcje są podawane podczas każdego uruchamiania serwera, znacznie częściej stosowane rozwiązanie polega na ich umieszczeniu w odpowiedniej grupie pliku opcji. Opcje dotyczące dzienników zdarzeń są zwykle wymieniane w grupie `[mysqld]`, ale nie zawsze tak jest. W punkcie 12.2.3, zatytułowanym „Określanie opcji startowych serwera”, przedstawiono informacje szczegółowe dotyczące grup opcji dostępnych dla serwera i programów go

uruchamiających. Na przykład, przedstawione poniżej wiersze w pliku opcji powodują włączenie dziennika błędów o nazwie *log.err* i ogólnego dziennika zapytań o nazwie *qlog*. Oba wymienione dzienniki zostaną umieszczone w katalogu danych serwera:

```
[mysqld]  
log_error = log.err  
general_log  
general_log_file = qlog
```

Za wyjątkiem dzienników binarnego i przekazywania, wszystkie dzienniki zdarzeń są zapisywane w postaci zwykłych plików tekstowych i mogą być bezpośrednio wyświetlane. Natomiast wyświetlenie zawartości dziennika binarnego lub przekazywania wymaga użycia narzędzia `mysqlbinlog`.

Poza wymienionymi dziennikami zdarzeń serwera mogą istnieć także inne, tworzone przez poszczególne silniki bazy danych. Na przykład, silnik InnoDB tworzy dzienniki zdarzeń używane do automatycznej naprawy po wystąpieniu awarii. Nie masz kontroli nad tym, czy silnik InnoDB generuje dziennik zdarzeń, ale za pomocą zmiennej systemowej `innodb_log_group_home_dir` możesz wskazać katalog, w którym będzie on zapisywany. Domyślnie wspomniany dziennik jest zapisywany w katalogu danych MySQL. Więcej informacji na ten temat znajdziesz w punkcie 12.5.3, zatytułowanym „Konfiguracja silnika InnoDB”.

12.8.1. Dziennik błędów

Serwer używa dziennika błędów do rejestracji zdarzeń zachodzących w trakcie uruchamiania i zatrzymywania serwera, a także informacji diagnostycznych i komunikatów błędów. Na ilość zapisywanych informacji wpływ ma ustawienie zmiennej systemowej `log_warnings`, która może mieć wartość od 0 do 2, wskazującą rosnącą ilość rejestrowanych danych.

Pewne właściwości dziennika zdarzeń są odmiennie obsługiwane na platformach UNIX i Windows, jak przedstawiono w poniższych podpunktach.

12.8.1.1. Dziennik błędów w systemach UNIX

W systemach UNIX `mysqld` domyślnie nie tworzy dziennika błędów, a informacje diagnostyczne są przekazywane do konsoli. Jeżeli wywołasz `mysqld` bezpośrednio, informacje o błędach mogą być zapisywane w pliku zamiast przekazywane do konsoli. W tym celu trzeba użyć zmiennej systemowej `log_error` z poziomu wiersza poleceń lub zdefiniować ją w grupie `[mysqld]` pliku opcji.

W przypadku uruchomienia serwera za pomocą skryptu `mysqld_safe`, domyślnie tworzony jest dziennik błędów, ponieważ `mysqld_safe` wywołuje `mysqld` i przekierowuje dane wyjściowe serwera do pliku dziennika błędów. Domyślna nazwa pliku dziennika błędów to `HOSTNAME.err`. Aby wskazać inną nazwę dziennika błędów, należy ustawić zmienną systemową `log_error` z poziomu wiersza poleceń lub zdefiniować ją w grupie `[mysqld]` lub `[mysqld_safe]` pliku opcji. (Skrypt `mysqld_safe` szuka opcji w grupie `[mysqld]` i jeśli znajdzie tam wartość `log_error`, to jej użyje).

Jeżeli dziennik błędów istnieje, ale jest niedostępny do zapisu przez użytkownika, który uruchomił serwer, wówczas próba uruchomienia serwera kończy się niepowodzeniem i żadne dane wyjściowe nie są zapisywane w dzienniku błędów. Taka sytuacja może wystąpić podczas uruchamiania serwera z różnymi wartościami `--user`. Stosowanie za każdym razem tego samego konta użytkownika, jak to omówiono w podpunkcie 12.2.1.1, zatytułowanym „Działanie serwera w ramach pozbawionego uprawnień konta logowania”, pozwala na uniknięcie takiego problemu.

Jeżeli serwer zostanie uruchomiony przez skrypt `mysqld_safe`, dane wyjściowe błędów mogą być kierowane do `syslog` zamiast pliku dziennika błędów. W celu użycia `syslog` dla diagnostycznych danych wyjściowych uruchom `mysqld_safe` wraz z opcją `--syslog` zamiast zmiennej systemowej `log_error`. W trakcie rejestracji zdarzeń w `syslog` komunikaty pochodzące z `mysqld` i `mysqld_safe` są zapisywane wraz z prefiksem, odpowiednio `mysqld` i `mysqld_safe`. W przypadku użycia `--syslog-tag=ciqq_tekstowy` prefiksami będą odpowiednio `mysqld-ciqq_tekstowy` i `mysqld_safe-ciqq_tekstowy`.

Uruchomienie serwera za pomocą skryptu `mysql.server` zawsze powoduje utworzenie dziennika błędów, ponieważ wymieniony skrypt wywołuje `mysqld_safe`. Jednak `mysql.server` nie rozpoznaje żadnych opcji dotyczących dzienników zdarzeń podawanych w wierszu poleceń lub grupie `[mysql.server]` pliku opcji. Aby wskazać jakiejkolwiek opcje wymagane przez `mysqld_safe` lub `mysqld`, należy użyć grupy `[mysqld_safe]` lub `[mysqld]` pliku opcji, jak to wcześniej omówiono.

12.8.1.2. Dziennik błędów w systemach Windows

W systemach Windows serwer domyślnie zapisuje informacje diagnostyczne w pliku o nazwie `HOSTNAME.err` znajdującym się w katalogu danych MySQL. W celu wskazania innego pliku dziennika błędów należy ustawić zmienną systemową `log_error` z poziomu wiersza poleceń lub zdefiniować ją w grupie `[mysqld]` pliku opcji. Ręczne uruchomienie serwera wraz z opcją `--console` powoduje wyświetlanie danych diagnostycznych w oknie konsoli zamiast ich zapisu w dzienniku błędów. (Jeżeli serwer został uruchomiony jako usługa Windows, opcja `--console` nie ma efektu, ponieważ w tym przypadku nie istnieje konsola, w której można by wyświetlać komunikaty).

12.8.2. Ogólny dziennik zapytań

Ogólny dziennik zapytań rejestruje następujące informacje: kiedy klient nawiązał połączenie z serwerem, jakie zapytania SQL były wysyłane do serwera przez klienty, a także informacje o innych zdarzeniach, takich jak uruchamianie i zatrzymywanie serwera. W tym dzienniku serwer zapisuje zapytania w kolejności ich otrzymywania. To może być inna kolejność od tej, w której ich wykonywanie się zakończyło, zwłaszcza w przypadku łączenia zapytań prostych i skomplikowanych.

W celu włączenia ogólnego dziennika zapytań należy ustawić zmienną `general_log`. Aby wskazać nazwę pliku dziennika, trzeba użyć zmiennej systemowej `general_log_file`. Domyślnie to plik `HOSTNAME.log` utworzony w katalogu danych MySQL. Wymienione zmienne można ustawić podczas uruchamiania serwera i zmienić w trakcie jego działania.

Zdarzenia ogólnego dziennika zapytań mogą być zapisywane w pliku, w bazie danych lub w obu wymienionych miejscach. Więcej informacji na ten temat znajdziesz w punkcie 12.8.6, zatytułowanym „Rejestracja zdarzeń w tabelach”.

12.8.3. Dziennik wolno wykonywanych zapytań

W tym dzienniku są rejestrowane zapytania, których wykonanie wymagało dłuższego czasu (domyślnie to 10 sekund). Ponieważ czas wykonywania zapytania nie jest znany aż do chwili zakończenia jego wykonywania, w tym dzienniku serwer zapisuje zapytania po ich wykonaniu, a nie otrzymaniu. Wolno wykonywane zapytania powodują również zwiększenie przez serwer wartości zmiennej stanu `Slow_queries`.

Dziennik wolno wykonywanych zapytań jest przez serwer zapisywany w postaci zwykłego tekstu, a więc może być wyświetlony za pomocą niemalże dowolnego programu. Ewentualnie do podsumowania jego zawartości można użyć narzędzia `mysqldumpslow`.

Dziennik wolno wykonywanych zapytań może być użyteczny do identyfikacji zapytań, których wydajność można poprawić przez ich modyfikację. Jednak podczas interpretacji zawartości dziennika pod uwagę trzeba wziąć ogólne obciążenie systemu. Powolność jest wyrażana w czasie rzeczywistym (nie czasie procesora). Dlatego też w przypadku dużego obciążenia serwera istnieje znacznie większe prawdopodobieństwo, że zapytanie będzie wykonane wolno, nawet jeśli w innych sytuacjach jest wykonywane szybko.

Włączenie dziennika wolno wykonywanych zapytań wymaga ustawienia zmiennej systemowej `slow_query_log`. Aby wskazać nazwę pliku dziennika, należy użyć zmiennej systemowej `slow_query_log_file`. Domyślnie to plik `HOSTNAME-slow.log` utworzony w katalogu danych MySQL. Wymienione zmienne można ustawić podczas uruchamiania serwera i zmienić w trakcie jego działania.

Zdarzenia dziennika wolno wykonywanych zapytań mogą być zapisywane w pliku, w bazie danych lub w obu wymienionych miejscach. Więcej informacji na ten temat znajdziesz w punkcie 12.8.6, zatytułowanym „Rejestracja zdarzeń w tabelach”.

Wiele opcji i zmiennych systemowych wpływa na to, co serwer zapisuje w dzienniku wolno wykonywanych zapytań:

- Zapytanie jest określane jako „wolne”, jeśli jego wykonanie zajmuje więcej czasu niż wartość zmiennej systemowej `long_query_time` (domyślnie to 10 sekund). Wartość minimalna wymienionej zmiennej wynosi 0. Wartość może również zawierać część dziesiętną. Jeżeli wolno wykonywane zapytanie zostanie zapisane w tabeli dziennika zdarzeń, wtedy część dziesiętna czasu jego wykonywania będzie odrzucona.
- Zapytanie musi dotyczyć przynajmniej wskazanej przez zmienną `min_examined_row_limit` liczby rekordów, w przeciwnym razie nie będzie zarejestrowane. Wartość domyślna wymienionej zmiennej wynosi 0.
- Po uruchomieniu serwera wraz z opcją `--log-short-format` w pliku dziennika będzie zapisywana mniejsza ilość informacji.

- Po uruchomieniu serwera wraz z opcją `--log-slow-admin-statements` serwer będzie rejestrował także zapytania administracyjne, takie jak `ANALYZE TABLE` i `ALTER TABLE`.
- Po włączeniu zmiennej systemowej `log_queries_not_using_indexes` serwer będzie rejestrował także zapytania wykonywane bez użycia indeksu w celu wyszukiwania rekordów. Jednak to może spowodować rejestrację dużej ilości danych. Począwszy od wersji 5.6.5, wymienionej zmiennej systemowej można ustawić wartość niezerową wskazującą maksymalną liczbę możliwych do zapisania w ciągu minuty zapytań nieużywających indeksów.

12.8.4. Binarny dziennik zdarzeń

Serwer używa binarnego dziennika zdarzeń do rejestracji „zdarzeń” modyfikujących dane, na przykład na skutek wykonania zapytań typu `INSERT`, `DELETE` lub `UPDATE`. W dzienniku nie są zapisywane zapytania `SELECT`. Ponadto, zapytanie `UPDATE`, takie jak przedstawiono poniżej, również nie pojawia się w binarnym dzienniku zdarzeń, ponieważ tak naprawdę nie zmienia żadnej wartości:

```
UPDATE t SET i = i;
```

Binarny dziennik zdarzeń jest używany przede wszystkim w trakcie operacji przywracania i naprawy bazy danych, a także w celu obsługi replikacji. (Aby skonfigurować serwer jako serwer główny replikujący rekordy do serwerów podległych, absolutnie *konieczne* jest włączenie binarnego dziennika zdarzeń). Ponieważ dziennik jest używany w replikacji, zawiera informacje użyteczne podczas replikacji, na przykład znaczniki czasu wskazujące moment wykonania danego zapytania.

MySQL najpierw musi wykonać zapytanie, aby było możliwe ustalenie, czy zmodyfikowało ono dane. Dlatego też dane w binarnym dzienniku zdarzeń są zapisywane po zakończeniu wykonywania zapytania, a nie po jego otrzymaniu.

W przeciwieństwie do innych dzienników zdarzeń, ten nie jest w postaci zwykłego tekstu, ale w znacznie bardziej efektywnym formacie binarnym. Oznacza to brak możliwości bezpośredniego przeglądania zawartości binarnego dziennika zdarzeń. W celu wyświetlenia jego zawartości w postaci czytelnego tekstu konieczne jest użycie narzędzia `mysqlbinlog`.

Fakt zapisywania w binarnym dzienniku zdarzeń w kolejności ich zakończenia, a nie otrzymania jest ważną cechą, pozwalającą na działanie mechanizmu replikacji. W przypadku zapytań będących częścią transakcji serwer buforuje je aż do chwili zatwierdzenia transakcji.

Następnie serwer rejestruje wszystkie zdarzenia w transakcji. Jeżeli transakcja została wycofana, nie będzie zapisana w binarnym dzienniku zdarzeń, ponieważ jej wynik nie wprowadza żadnych zmian w bazie danych.

W rzeczywistości, bardziej odpowiada prawdzie stwierdzenie, że wycofane transakcje *zwykle* nie są zapisywane w binarnym dzienniku zdarzeń. Jeżeli transakcja wprowadza zmiany w tabelach nietransakcyjnych, takich jak `MyISAM`, tego rodzaju zmiany nie mogą być wycofane. W takim przypadku nawet wycofana transakcja zostaje zapisana w binarnym dzienniku zdarzeń, aby zagwarantować prawidłową replikację tabel nietransakcyjnych.

W celu włączenia binarnego dziennika zdarzeń należy użyć opcji `--log-bin`. Użycie wymienionej opcji bez nazwy pliku powoduje, że serwer generuje pliki binarnego dziennika zdarzeń w postaci ponumerowanej sekwencji, w której `HOSTNAME-bin` to nazwa bazowa plików: `HOSTNAME-bin.000001`, `HOSTNAME-bin.000002` itd. W przeciwnym razie serwer użyje nazwy wskazanej jako podstawowa dla sekwencji (jeśli nazwa zawiera rozszerzenie, zostanie ono zignorowane). Kolejny plik w sekwencji serwer generuje za każdym razem, gdy uruchamiasz serwer, lub po osiągnięciu maksymalnej wielkości przez bieżący dziennik zdarzeń. Wspomniana wielkość maksymalna to wartość zmiennej systemowej `max_binlog_size`.

Po włączeniu binarnego dziennika zdarzeń serwer tworzy towarzyszący mu plik indeksu zawierający nazwy istniejących plików binarnego dziennika zdarzeń. Domyślna nazwa pliku indeksu odpowiada nazwie bazowej plików dziennika zdarzeń, ale zawiera rozszerzenie `.index`. W celu wyraźnego wskazania nazwy należy użyć opcji `--log-bin-index`. Jeżeli nazwa nie zawiera rozszerzenia, serwer automatycznie dołączy rozszerzenie `.index`. Na przykład, po użyciu opcji `--log-bin-index=binlog` nazwą pliku indeksu będzie `binlog.index`.

Jeżeli użyjesz opcji `--log-short-format` w połączeniu z `--log-bin`, w binarnym dzienniku zdarzeń MySQL umieści mniejszą ilość informacji.

Serwer może zapisywać zdarzenia w dzienniku binarnym w formacie opartym na zapytaniach lub rekordach. Na przykład, w przypadku rejestracji zdarzeń w formacie opartym na zapytaniach zapytanie `UPDATE` zostanie zapisane w postaci zapytania `UPDATE`, podczas gdy w formacie opartym na rekordach zapytanie `UPDATE` będzie zapisane w postaci zmian wprowadzonych w poszczególnych rekordach, które zostały uaktualnione. (Zapoznaj się z punktem 14.8.3, zatytułowanym „Formaty rejestracji zdarzeń w dzienniku binarnym”). Zmienna systemowa `binlog_format` wskazuje stosowany format rejestracji zdarzeń. Dozwolonymi wartościami wymienionej zmiennej są `STATEMENT`, `ROW` i `MIXED`. (Ostatnia z wymienionych wartości stosuje format oparty na rekordach, za wyjątkiem sytuacji, w których konieczne jest użycie formatu opartego na zapytaniach). Wartością domyślną zmiennej `binlog_format` jest `STATEMENT`.

Jeżeli binarny dziennik zdarzeń jest używany przez replikację, nie usuwaj żadnego pliku dziennika aż do całkowitego upewnienia się, że jego cała zawartość została replikowana do wszystkich serwerów poległych, a sam plik nie będzie dłużej potrzebny. O tym, jak to sprawdzić, dowiesz się w podpunkcie 12.8.7.2, zatytułowanym „Utrata ważności plików dzienników zdarzeń binarnego i przekazywania”.

Pliki binarnego dziennika zdarzeń i kopie zapasowe systemu

Pliki dzienników zdarzeń binarnego i przekazywania nie będą przydatne w trakcie naprawy bazy danych lub replikacji, jeśli dojdzie do awarii dysku i ich utraty. Upewnij się, że regularnie są tworzone kopie zapasowe systemu plików. Dobrym rozwiązaniem jest zapis plików wymienionych dzienników na zupełnie innym dysku niż zawierający bazy danych. To, oczywiście, wymaga przeniesienia plików dzienników z katalogu danych serwera MySQL, w którym są domyślnie zapisywane. W tym celu użyj odpowiedniej opcji i wskaż nowe miejsce położenia dla dzienników zdarzeń.

12.8.5. Dziennik przekazywania

Dziennik przekazywania jest używany w serwerach podległych replikacji (patrz podrozdział 14.8, zatytułowany „Konfiguracja serwerów replikacji”). Serwer podległy w replikacji otrzymuje z serwera głównego „zdarzenia” modyfikujące dane i po otrzymaniu zapisuje je w dzienniku przekazywania. Wspomniany dziennik jest używany przez serwer podległy jako miejsce do przechowywania wspomnianych zdarzeń aż do chwili, gdy będą mogły zostać wykonane.

Dwa wątki serwera podległego zajmują się obsługą zdarzeń: ich odczytem i wykonywaniem. Wątek operacji wejścia-wyjścia odczytuje zdarzenia z serwera głównego i zapisuje je w dzienniku przekazywania. Z kolei wątek SQL odczytuje pliki dziennika przekazywania, wykonuje zdarzenia, a następnie usuwa poszczególne pliki po ich całkowitym przetworzeniu. Rozdzielenie obu wymienionych zadań pozwala wątkom na niezależne działanie.

Dziennik przekazywania ma pewne takie same cechy jak binarny dziennik zdarzeń:

- Serwer tworzy pliki dziennika przekazywania w postaci ponumerowanej sekwencji.
- Plik indeksu zawiera listę wszystkich aktualnych plików dziennika przekazywania.
- Pliki dziennika przekazywania mają taki sam format jak pliki binarnego dziennika zdarzeń. Dlatego też w celu wyświetlenia ich zawartości należy użyć narzędzia `mysqlbinlog`.

W celu włączenia dziennika przekazywania należy użyć opcji `--relay-bin`. Użycie wymienionej opcji bez nazwy pliku powoduje, że serwer generuje pliki binarnego dziennika zdarzeń w postaci ponumerowanej sekwencji, w której `HOSTNAME-relay-bin` to nazwa bazowa plików: `HOSTNAME-relay-bin.000001`, `HOSTNAME-relay-bin.000002` itd. W przeciwnym razie serwer użyje nazwy wskazanej jako podstawowa dla sekwencji (jeśli nazwa zawiera rozszerzenie, zostanie ono zignorowane). Kolejny plik w sekwencji serwer generuje za każdym razem, gdy uruchamiasz serwer, lub po osiągnięciu maksymalnej wielkości przez bieżący dziennik zdarzeń. Wspomniana wielkość maksymalna to wartość zmiennej systemowej `max_relay_log_size`.

Po włączeniu dziennika przekazywania serwer tworzy towarzyszący mu plik indeksu zawierający nazwy istniejących plików dziennika przekazywania. Domyślna nazwa pliku indeksu odpowiada nazwie bazowej plików dziennika zdarzeń, ale zawiera rozszerzenie `.index`. W celu wyraźnego wskazania nazwy należy użyć opcji `--relay-log-index`. Jeżeli nazwa nie zawiera rozszerzenia, serwer automatycznie dołączy rozszerzenie `.index`. Na przykład, po użyciu opcji `--relay-log-index=relay-log` nazwą pliku indeksu będzie `relay-log.index`.

12.8.6. Rejestracja zdarzeń w tabelach

Po włączeniu dzienników zdarzeń ogólnego lub wolno wykonywanych zapytań informacje można rejestrować w pliku dziennika, w tabeli bazy danych `mysql` lub w obu wymienionych miejscach.

W celu wybrania miejsca docelowego dla dziennika zdarzeń należy użyć zmiennej systemowej `log_output` podczas uruchamiania serwera. Wartość wymienionej zmiennej powinna składać się z jednej lub więcej rozdzielonych przecinkami nazw miejsc docelowych: `FILE` (rejestracja informacji w plikach), `TABLE` (rejestracja informacji w tabelach) lub `NONE` (dziennik zdarzeń nie będzie generowany). Wartość `NONE` powoduje nadpisanie innych ewentualnych miejsc docelowych. Jeśli zmienna `log_output` nie będzie ustawiona, domyślnie używana jest wartość `FILE`.

Zmienna systemowa `log_output` określa miejsce docelowe dla rejestrowanych informacji, ale nie powoduje włączenia rejestrowania zdarzeń. W celu włączenia dzienników ogólnego lub wolno wykonywanych zapytań konieczne jest ustawienie zmiennych systemowych `general_log` lub `slow_query_log`.

Aby podczas działania serwera zmienić miejsce docelowe dla zapisywanych zdarzeń, konieczne jest ustawienie globalnej wartości zmiennej systemowej `log_output`. Na przykład, w celu tymczasowego wyłączenia rejestracji zdarzeń trzeba wykonać poniższe zapytanie:

```
SET GLOBAL log_output='NONE';
```

Ponowne włączenie rejestracji zdarzeń w plikach oraz tabelach następuje po wykonaniu poniższego zapytania:

```
SET GLOBAL log_output='FILE, TABLE';
```

Jeżeli rejestrowanie zdarzeń jest włączone, komunikaty startowe są zapisywane przez serwer w pliku dziennika. O ile nie została ustawiona wartość `FILE` zmiennej `log_output`, to żadne inne informacje nie są zapisywane w pliku dziennika zdarzeń.

W przypadku ustawionej opcji `TABLE` zmiennej `log_output` serwer zapisuje informacje w tabelach `general_log` i `slow_log` w bazie danych `mysql`.

W przypadku ustawionej opcji `FILE` zmiennej `log_output` wartości globalne zmiennych systemowych `general_log_file` i `slow_query_log_file` określają nazwy plików dzienników zdarzeń. Domyślnie to odpowiednio `HOSTNAME.log` i `HOSTNAME-slow.log` umieszczone w katalogu danych `MySQL`. Po włączeniu rejestracji zdarzeń w plikach zmiana wartości dowolnej z wymienionych zmiennych powoduje zmianę nazwy pliku, w którym będą zapisywane zdarzenia odpowiadającego jej dziennika.

Zawartość tabel dzienników zdarzeń jest przeznaczona do wyświetlania i nie może być modyfikowana przez użytkownika, a jedynie przez serwer. Dlatego też możesz wykonywać zapytania `SELECT`, ale nie `INSERT`, `DELETE` lub `UPDATE`. (Jednak istnieje możliwość opróżnienia tabeli dziennika zdarzeń za pomocą zapytania `TRUNCATE TABLE`).

12.8.7. Zarządzanie dziennikami zdarzeń

Rejestracja zdarzeń jest ważna, ale może doprowadzić do wygenerowania ogromnej ilości informacji i ewentualnie zapełnić dyski. To niebezpieczeństwo istnieje zwłaszcza w bardzo obciążonych serwerach, przetwarzających ogromne ilości zapytań. Aby przechowywać informacje jedynie o ostatnich zapytaniach i nie pozwolić na nadmierne rozrośnięcie się

dzienników zdarzeń, należy stosować pewne techniki prowadzące do utraty ważności przez pliki dzienników zdarzeń. W celu ułatwienia zarządzania dziennikami zdarzeń można zastosować wiele metod:

- **Rotacja dzienników zdarzeń.** Ma zastosowanie w plikach dzienników zdarzeń o stałych nazwach, na przykład plików dzienników błędów, ogólnego dziennika zapytań i wolno wykonywanych zapytań.
- **Utrata ważności na podstawie wieku.** Ta metoda powoduje usunięcie plików dzienników zdarzeń starszych niż pewien ustalony wiek. Ma zastosowanie w przypadku plików dzienników tworzonych w ponumerowanych sekwencjach, czyli na przykład w binarnym dzienniku zdarzeń. Jeżeli binarny dziennik zdarzeń jest używany do replikacji, nie powinien być stosować tej metody.
- **Utrata ważności powiązana z replikacją.** Jeżeli binarny dziennik zdarzeń jest używany do replikacji, plików dzienników zdarzeń nie powinien być usuwać na podstawie wieku. Ich utrata ważności powinna następować dopiero po upewnieniu się, że zostały w całości przekazane do wszystkich serwerów podległych. Dlatego też taka forma utraty ważności opiera się na ustaleniu, które pliki binarnego dziennika zdarzeń nadal pozostają w użyciu. Serwer podległy replikacji tworzy pliki dziennika przekazywania w postaci ponumerowanej sekwencji i usuwa je automatycznie po zakończeniu ich przetwarzania. Aby zmniejszyć ilość przechowywanych na dysku informacji dziennika przekazywania, należy zmniejszyć jego maksymalną dozwoloną wielkość przez zmniejszenie wartości zmiennej systemowej `max_relay_log_size`.
- **Opróżnianie tabeli dzienników zdarzeń, czyli rotacja.** Jeżeli zdarzenia są zapisywane w tabelach bazy danych mysql, istnieje możliwość ich opróżniania lub zmiany nazwy i zastępowania pustymi tabelami.

W przedstawionych dalej podpunktach dowiesz się, jak używać wymienionych powyżej technik. Omówione tutaj przykładowe skrypty utraty ważności dzienników zdarzeń znajdziesz w katalogu *admin* dystrybucji *sampdb*.

W przypadku każdej stosowanej techniki należy zastanowić się również, jak pliki dzienników zdarzeń mają się względem używanych metod tworzenia kopii zapasowych bazy danych. Dobrym rozwiązaniem jest tworzenie kopii zapasowej wszelkich plików dzienników zdarzeń, które mogą być użyteczne w trakcie operacji przywracania bazy danych. Dlatego też nie należy usuwać tych plików przed wykonaniem ich kopii zapasowej.

Opróżnianie dzienników zdarzeń

Opróżnianie dzienników zdarzeń jest często stosowane jako część procedury utraty ich ważności lub rotacji. Upewnij się więc, że wszelkie informacje buforowane przez dziennik zdarzeń zostały zapisane na dysku. Opróżnienie dziennika zdarzeń powoduje, że serwer zamyka i ponownie otwiera plik dziennika zdarzeń.

Dlatego też wykonaj polecenie `mysqladmin flush-logs` lub zapytanie `FLUSH LOGS`. W systemach UNIX wysłanie sygnału `SIGHUP` do serwera również powoduje opróżnienie dzienników zdarzeń. Polecenie `mysqladmin refresh` także opróżnia dzienniki zdarzeń, ale przeprowadza też inne operacje, na przykład opróżnienie bufora tabeli. Dlatego też to niepotrzebnie wydawane polecenie, jeśli chcesz jedynie opróżnić dzienniki zdarzeń.

Serwer tworzy pliki dzienników binarnego i przekazywania w postaci ponumerowanej sekwencji. Opróżnienie dzienników zdarzeń powoduje więc zamknięcie przez serwer bieżącego pliku dziennika i utworzenie nowego wraz z kolejnym numerem w sekwencji.

Opróżnienie dzienników zdarzeń nie wpływa na dzienniki zdarzeń tworzone przez poszczególne silniki bazy danych.

12.8.7.1. Rotacja plików dzienników zdarzeń o na stałe zdefiniowanych nazwach

Pewne rodzaje informacji serwer MySQL zapisuje w plikach, które mają na stałe zdefiniowane nazwy. Dotyczy to na przykład dzienników błędów, ogólnego oraz wolno wykonywanych zapytań. Aby wskazać utratę ważności pliku dziennika o na stałe zdefiniowanej nazwie, trzeba przeprowadzić rotację. W ten sposób ogranicza się liczbę przechowywanych plików dzienników zdarzeń, ale jednocześnie ogranicza ich liczbę do wskazanej przez Ciebie i uniemożliwia zapełnienie dysku plikami dzienników.

Rotacja plików dzienników zdarzeń działa w następujący sposób. Przyjmujemy założenie, że mamy plik ogólnego dziennika zapytań o nazwie *qlog*. W trakcie pierwszej rotacji jego nazwa zostaje zmieniona na *qlog.1*, a serwer rozpoczyna zapisywanie zdarzeń do nowego pliku dziennika o nazwie *qlog*. (Nazwę bieżącego pliku dziennika zdarzeń zmień, gdy serwer ma otwarty ten plik, a następnie opróżnij dziennik. W takim przypadku serwer zamknie bieżący plik i otworzy nowy, czyli utworzy nowy plik dziennika zdarzeń o nazwie takiej samej jak początkowa). W trakcie drugiej rotacji należy zmienić nazwy następujących plików: *qlog.1* na *qlog.2* i *qlog* na *qlog.1* oraz nakazać serwerowi rozpoczęcie zapisu zdarzeń w nowym pliku *qlog*. W ten sposób każdy plik dziennika zdarzeń będzie miał kolejno nazwy *qlog*, *qlog.1*, *qlog.2* itd. Kiedy plik osiągnie określony punkt w rotacji, należy go po prostu usunąć, nadpisując innym. Na przykład, jeśli chcesz przechowywać pliki dzienników zdarzeń z całego tygodnia i przeprowadzać rotację każdego dnia, powinieneś zachować pliki od *qlog.1* do *qlog.7*. W trakcie każdej rotacji pozbywasz się pliku *qlog.7*, nadpisując go plikiem *qlog.6*, który w ten sposób staje się nowym plikiem *qlog.7*.

Częstotliwość rotacji plików dzienników zdarzeń i liczba starych dzienników, które mają być przechowywane, zależy od obciążenia serwera (intensywnie wykorzystywane serwery generują więcej informacji w dziennikach zdarzeń) oraz od ilości miejsca, jakie chcesz poświęcić dla starych dzienników zdarzeń.

Przedstawiony poniżej skrypt powłoki o nazwie *rotate_fixed_logs.sh* przeprowadza w systemach UNIX rotację plików dzienników zdarzeń o na stałe zdefiniowanych nazwach.

```
#!/bin/sh
# rotate_fixed_logs.sh - Rotacja pliku dziennika zdarzeń, który ma stałą nazwę.

# Argument 1: nazwa pliku dziennika zdarzeń.
if [ $# -ne 1 ]; then
    echo "Użycie: $0 logname" 1>&2
    exit 1
fi

logname=$1

mv $logname.6 $logname.7
mv $logname.5 $logname.6
mv $logname.4 $logname.5
mv $logname.3 $logname.4
mv $logname.2 $logname.3
mv $logname.1 $logname.2
mv $logname $logname.1
mysqladmin flush-logs
```

Powyższy skrypt jako argument pobiera nazwę pliku dziennika zdarzeń. Można również podać pełną ścieżkę dostępu do pliku lub przejść do katalogu zawierającego plik dziennika i podać nazwę pliku znajdującego się w tym katalogu. Na przykład, aby przeprowadzić rotację pliku dziennika zdarzeń o nazwie *qlog* znajdującego się w katalogu */usr/mysql/data*, należy skrypt uruchomić w następujący sposób:

```
% rotate_fixed_logs.sh /usr/mysql/data/qlog
```

Lub tak:

```
% cd /usr/mysql/data
% rotate_fixed_logs.sh qlog
```

Uwaga

Podczas kilku pierwszych uruchomień skryptu nie będziesz miał pełnego zestawu plików dzienników zdarzeń, więc skrypt wyświetli komunikaty informujące o nieznalezieniu plików przeznaczonych do rotacji. To normalne zachowanie.

W celu zagwarantowania uprawnień do zmiany nazw plików dzienników zdarzeń skrypt powinien zostać uruchomiony przez tego samego użytkownika, który jest używany do uruchamiania serwera (w tej książce przyjęto założenie, że to będzie użytkownik *mysql*). Zwróć uwagę, że polecenie *mysqladmin* w przedstawionym skrypcie nie zawiera argumentów parametrów połączenia, takich jak *-u* lub *-p*. Jeżeli odpowiednie parametry dla wywoływanego narzędzia *mysqladmin* są przechowywane w pliku opcji *.my.cnf* w katalogu domowym użytkownika *mysql*, nie ma konieczności ich podawania w poleceniu *mysqladmin* wywoływanym w skrypcie. W przypadku braku pliku opcji polecenie *mysqladmin* musi wiedzieć, jak nawiązać połączenie z serwerem MySQL, używając konta MySQL o uprawnieniach wystarczających do opróżnienia dzienników zdarzeń. Rozwiązaniem może być przygotowanie

konta MySQL o ograniczonych uprawnieniach, pozwalających jedynie na opróżnienie dzienników zdarzeń (takie konto powinno posiadać jedynie uprawnienia RELOAD). Na przykład, aby utworzyć użytkownika o nazwie `flush` i przypisać mu hasło `flushpass`, wykonaj następujące zapytania:

```
CREATE USER 'flush'@'localhost' IDENTIFIED BY 'flushpass';
GRANT RELOAD ON *.* TO 'flush'@'localhost';
```

Po utworzeniu wymienionego konta użytkownika zmień polecenie `mysqladmin` w skrypcie `rotate_fixed_logs.sh` na następujące:

```
mysqladmin -u flush -pflushpass flush-logs
```

Aby uniemożliwić odczytanie skryptu przez innych użytkowników, należy odebrać im uprawnienia. Po zalogowaniu się jako użytkownik `mysql` wydaj następujące polecenie:

```
% chmod go-rwx rotate_fixed_logs.sh
```

Jeśli chcesz dowiedzieć się, jak użyć skryptu `rotate_fixed_logs.sh` do automatycznej rotacji i opróżniania dzienników zdarzeń, zapoznaj się z podpunktem 12.8.7.3, zatytułowanym „Automatyzacja procedury utraty ważności plików dzienników zdarzeń”.

W systemach UNIX możesz preferować użycie narzędzia `logrotate` i zainstalować skrypt `mysql-log-rotate` dostarczany wraz z dystrybucją MySQL zamiast skryptu `rotate_fixed_logs.sh` bądź innego, utworzonego samodzielnie. Zajrzyj do podkatalogu `mysql-log-rotate` w `/usr/share/mysql` (dla dystrybucji zainstalowanej z pakietów RPM) lub katalogu `support-files` kodu źródłowego instalacji MySQL.

W systemach Windows można przeprowadzać rotację plików dzienników zdarzeń za pomocą przedstawionego poniżej skryptu wsadowego `rotate_fixed_logs.bat`:

```
@echo off
REM rotate_fixed_logs.bat - Rotacja pliku dziennika zdarzeń, który ma stałą nazwę.

REM Argument 1: nazwa pliku dziennika zdarzeń.
if not "%1" == "" goto ROTATE
@echo Użycie: rotate_fixed_logs logname
goto DONE

:ROTATE
set logname=%1
erase %logname%.7
rename %logname%.6 %logname%.7
rename %logname%.5 %logname%.6
rename %logname%.4 %logname%.5
rename %logname%.3 %logname%.4
rename %logname%.2 %logname%.3
rename %logname%.1 %logname%.2
rename %logname% %logname%.1
mysqladmin flush-logs
:DONE
```

Skrypt `rotate_fixed_logs.bat` jest wywoływany bardzo podobnie jak skrypt powłok `rotate_fixed_logs.sh`, ale wraz z pojedynczym argumentem wskazującym nazwy plików dziennika zdarzeń do rotacji. Na przykład, w celu rotacji pliku dziennika zdarzeń o nazwie `qlog` znajdującego się w katalogu `C:\mysql\data` należy wywołać skrypt w następujący sposób:

```
C:\> rotate_fixed_logs C:\mysql\data\qlog
```

Lub tak:

```
C:\> cd \mysql\data
```

```
C:\> rotate_fixed_logs qlog
```

12.8.7.2. Utrata ważności plików dzienników zdarzeń binarnego i przekazywania

Pliki dzienników zdarzeń o na stałe zdefiniowanych nazwach można poddawać rotacji, zgodnie z przedstawionym wcześniej opisem. W przypadku dzienników takich jak binarny i przekazywania, które są generowane w postaci ponumerowanej sekwencji, utrata ważności musi być przeprowadzona nieco inaczej.

Dla binarnego dziennika zdarzeń można zastosować jedno z dwóch dostępnych podejść:

- Utrata ważności plików dzienników zdarzeń na podstawie ich wieku (daty i godziny ostatniej modyfikacji). Takie podejście można zastosować, jeśli binarny dziennik zdarzeń nie jest używany w mechanizmie replikacji.
- Utrata ważności plików dzienników zdarzeń na podstawie ustalenia, czy nadal pozostają w użyciu. Takie rozwiązanie powinno być stosowane, jeśli binarny dziennik zdarzeń jest używany w mechanizmie replikacji.

Aby nastąpiła utrata ważności pliku dziennika zdarzeń na podstawie jego wieku, najłatwiejszym rozwiązaniem jest ustawienie zmiennej systemowej o nazwie `expire_logs_days`. Na przykład, w celu ustawienia utraty ważności plików binarnego dziennika zdarzeń, które nie zostały zmodyfikowane od tygodnia, w pliku opcji należy umieścić poniższe wiersze:

```
[mysqld]
expire_logs_days=7
```

Jeżeli zmienna `expire_logs_days` ma wartość n większą niż zero, serwer automatycznie unieważnia pliki binarnego dziennika zdarzeń, które są starsze niż n dni, i uaktualnia plik indeksu binarnego dziennika zdarzeń. Sprawdzenie plików następuje podczas uruchamiania serwera i otwierania nowego pliku dziennika zdarzeń (na przykład po opróżnieniu dziennika lub jeśli bieżący plik dziennika osiągnie wielkość maksymalną wskazywaną przez zmienną systemową `max_binlog_size`).

Jeżeli binarny dziennik zdarzeń jest używany przez mechanizm replikacji, nie należy stosować utraty ważności plików na podstawie ich wieku, ponieważ w tym przypadku wiek nie jest dobrym wskaźnikiem, czy replikowany plik dziennika zdarzeń można usunąć. Przyjmujemy założenie, że serwer podległy replikacji jest wyłączony i nie otrzymał zawartości danego pliku binarnego dziennika zdarzeń. Jeżeli wspomniany serwer podległy nie zostanie włączony przed upływem czasu powodującego unieważnienie pliku, ten plik dziennika zostanie usunięty, a replikacja zakończy się niepowodzeniem. W celu uniknięcia takiego problemu plik binarnego dziennika zdarzeń może być uznany za możliwy do usunięcia, gdy jego zawartość została replikowana do wszystkich serwerów podległych.

Trudność wiąże się z asynchroniczną naturą replikacji MySQL. Serwer główny nie wie, ile jest serwerów podległych oraz które pliki zostały już do nich przekazane. Dlatego też serwer główny replikacji nie usunie plików binarnego dziennika zdarzeń, które nie zostały jeszcze wysłane do serwerów podległych replikacji. Nie ma jednak gwarancji, że dany serwer podległy zostanie podłączony w określonym czasie. Trzeba wiedzieć, jakie serwery działają w charakterze serwerów podległych replikacji, i ustalić, który plik binarnego dziennika zdarzeń jest przetwarzany. W tym celu konieczne jest nawiązanie połączenia z każdym serwerem podległym i wykonanie zapytania `SHOW SLAVE STATUS`, a następnie sprawdzenie wartości kolumny `Master_Log_File`. Plik binarnego dziennika zdarzeń, który nie jest używany przez ani jeden serwer podległy, może zostać usunięty.

Aby zrozumieć, jak działa wspomniany mechanizm, poniżej przedstawiono przykładowy scenariusz:

- Serwer lokalny jest serwerem głównym replikacji i ma dwa serwery podległe, S1 i S2.
- W serwerze głównym pliki binarnego dziennika zdarzeń mają nazwy od `binlog.000038` do `binlog.000042`.
- Zapytanie `SHOW SLAVE STATUS` powoduje wygenerowanie następujących danych wyjściowych w serwerze S1:

```
mysql> SHOW SLAVE STATUS\G
...
Master_Log_File: binlog.000041
...
```

oraz poniższych w serwerze S2:

```
mysql> SHOW SLAVE STATUS\G
...
Master_Log_File: binlog.000040
...
```

W takim przypadku najstarszy plik binarnego dziennika zdarzeń wymagany przez zestaw serwerów podległych to `binlog.000040`. Oznacza to możliwość usunięcia wszystkich plików o mniejszym numerze w sekwencji. Wystarczy więc nawiązać połączenie z serwerem głównym i wykonać następujące zapytanie:

```
mysql> PURGE BINARY LOGS TO 'binlog.000040';
```

Powyższe zapytanie spowoduje usunięcie w serwerze głównym replikacji plików binarnego dziennika zdarzeń o nazwach aż do wymienionej, przy czym wymieniony plik pozostaje. W omawianej sytuacji usunięte zostaną więc pliki `binlog.000038` i `binlog.000039`.

Wykonanie zapytań `SHOW SLAVE STATUS` i `PURGE BINARY LOGS` wymaga uprawnień `SUPER`.

Aby nastąpiła utrata ważności plików dziennika przekazywania, nie trzeba podejmować żadnych specjalnych kroków. Serwer replikacji tworzy pliki dziennika przekazywania w postaci ponumerowanej sekwencji. Nowy plik dziennika jest generowany wtedy, gdy aktualny osiągnie maksymalną dozwoloną wielkość (lub w trakcie opróżniania dzienników zdarzeń). Po zakończeniu przetwarzania starych plików zostają one automatycznie usunięte.

Jednak jeśli maksymalna wielkość pliku dziennika przekazywania jest duża, bieżący plik może osiągnąć spore rozmiary. Aby używać mniejszej ilości miejsca na dysku, należy ustawić zmiennej systemowej `max_relay_log_size` mniejszą wartość i tym samym zmniejszyć maksymalną dozwoloną wielkość pliku dziennika zdarzeń.

12.8.7.3. Automatyzacja procedury utraty ważności plików dzienników zdarzeń

Skrypty powodujące utratę ważności dzienników zdarzeń można uruchamiać ręcznie, choć to wymaga pamiętania o konieczności ich wykonania. Dobrym rozwiązaniem jest więc skonfigurowanie systemu, aby wspomniane skrypty były uruchamiane automatycznie. W systemach UNIX jedną z możliwości jest użycie narzędzia `cron` i przygotowanie pliku `crontab` definiującego harmonogram wywoływania skryptów. (W systemie Windows należy skorzystać z harmonogramu zadań).

Informacje dotyczące `cron` znajdziesz na stronach podręcznika systemu UNIX po wydaniu poniższych poleceń:

```
% man cron
% man crontab
% man 5 crontab
```

Przyjmujemy założenie, że rotacji ma zostać poddany plik ogólnego dziennika zapytań o nazwie `qlog` za pomocą skryptu `rotate_fixed_logs.sh`, który znajduje się w katalogu `/home/mysql/bin`, natomiast pliki dzienników zdarzeń są umieszczone w katalogu `/var/mysql/data`. Po zalogowaniu się jako użytkownik `mysql` należy przeprowadzić edycję pliku `crontab` tego użytkownika, wydając poniższe polecenie:

```
% crontab -e
```

Wydanie powyższego polecenia pozwala na edycję kopii bieżącego pliku `crontab` (może być pusty, jeśli jeszcze nie zostały zdefiniowane żadne zadania `cron`). Do pliku dodaj wiersz podobny do poniższego:

```
30 4 * * * /home/mysql/bin/rotate_fixed_logs.sh /var/mysql/data/qlog
```

Zdefiniowane w ten sposób zadanie powoduje, że mechanizm `cron` codziennie o godzinie 04:30 będzie uruchamiał wskazany skrypt. Harmonogram ustaw wedle własnych potrzeb; dokładne omówienie stosowanego formatu zadań znajdziesz w podręczniku `crontab`. W przypadku bardzo obciążonego serwera, generującego wiele informacji, rotację plików dzienników zdarzeń trzeba prawdopodobnie przeprowadzać częściej. Z kolei w mniej obciążonych serwerach rotację można przeprowadzać rzadziej.

W celu zagwarantowania regularnego opróżniania dzienników zdarzeń (by na przykład wygenerować kolejny w sekwencji plik dziennika binarnego) można zdefiniować kolejne zadanie w pliku `crontab`, tym razem regularnie wykonujące polecenie `mysqladmin flush-logs`. Prawdopodobnie trzeba będzie podać pełną ścieżkę dostępu do `mysqladmin`, aby upewnić się, że wymienione narzędzie zostało znalezione przez `cron`.

12.8.7.4. Utrata ważności, czyli rotacja tabel dzienników zdarzeń

Jeżeli serwer zapisuje w tabelach bazy danych mysql zdarzenia ogólnego dziennika zapytań lub wolno wykonywanych zapytań, tabele można opróżniać lub stosować pewną formę ich rotacji.

Opróżnienie tabel następuje po wykonaniu poniższych zapytań:

```
USE mysql;
TRUNCATE TABLE general_log;
TRUNCATE TABLE slow_log;
```

W celu przeprowadzenia rotacji najpierw trzeba utworzyć pustą kopię tabel. Następnie w pojedynczym niepodzielnym zapytaniu trzeba dokonać ich „zamiany”:

```
USE mysql;
DROP TABLE IF EXISTS general_log_tmp, general_log_old;
CREATE TABLE general_log_tmp LIKE general_log;
RENAME TABLE general_log TO general_log_old, general_log_tmp TO general_log;
DROP TABLE IF EXISTS slow_log_tmp, slow_log_old;
CREATE TABLE slow_log_tmp LIKE slow_log;
RENAME TABLE slow_log TO slow_log_old, slow_log_tmp TO slow_log;
```

Jeżeli włączony jest pewien mechanizm harmonogramu zadań, rotację tabel można przeprowadzić automatycznie przez utworzenie zdarzenia takiego jak przedstawione poniżej. Zaprezentowane poniżej zdarzenie powoduje codzienną rotację tabel. Jeśli chcesz zdefiniować inną częstotliwość rotacji, wystarczy po prostu zmodyfikować klauzulę ON SCHEDULE:

```
CREATE EVENT mysql.rotate_log_tables
ON SCHEDULE EVERY 1 DAY
DO BEGIN
    DROP TABLE IF EXISTS general_log_tmp, general_log_old;
    CREATE TABLE general_log_tmp LIKE general_log;
    RENAME TABLE
        general_log TO general_log_old,
        general_log_tmp TO general_log;
    DROP TABLE IF EXISTS slow_log_tmp, slow_log_old;
    CREATE TABLE slow_log_tmp LIKE slow_log;
    RENAME TABLE
        slow_log TO slow_log_old,
        slow_log_tmp TO slow_log;
END;
```

12.9. Uruchamianie wielu serwerów MySQL

Większość osób uruchamia tylko jeden serwer MySQL w danym komputerze, ale czasami uruchomienie kilku staje się użyteczne lub wręcz niezbędne:

- Chcesz przetestować nową wersję serwera, ale pozostawiając nietknięty serwer produkcyjny. W takim przypadku rozwiązaniem jest uruchamianie różnych plików binarnych serwera.

- Chcesz wypróbować mechanizm replikacji, aby go nieco poznać. Ponieważ dysponujesz tylko jednym komputerem, to serwery główny i podległy replikacji muszą znajdować się w tym samym komputerze.
- Systemy operacyjne zwykle nakładają ograniczenie w postaci dozwolonej liczby otwartych deskryptorów plików dla procesu. Jeżeli zmiana wspomnianej liczby jest trudna w systemie operacyjnym (na przykład wymaga przeprowadzenia ponownej kompilacji jądra), wtedy uruchomienie wielu egzemplarzy serwera jest jednym ze sposobów ominięcia tego rodzaju ograniczenia.
- Dostawcy usług internetowych bardzo często dostarczają poszczególnym klientom ich własne instalacje MySQL, co wymaga uruchamiania wielu serwerów. Może to oznaczać uruchamianie wielu egzemplarzy tego samego pliku binarnego, jeśli wszystkie klienty mają do dyspozycji tę samą wersję MySQL. Druga możliwość to uruchamianie różnych plików binarnych, gdy pewne klienty mają do dyspozycji inną wersję serwera niż pozostałe.

12.9.1. Ogólne kwestie związane z uruchamianiem wielu serwerów

Uruchomienie wielu serwerów jest bardziej skomplikowane niż tylko jednego, ponieważ trzeba zagwarantować, że nie będą sobie przeszkadzały w pracy. Pewne problemy pojawiają się już w trakcie instalacji. Jednoczesna instalacja kilku różnych wersji wymaga ich umieszczenia w oddzielnych położeniach. W przypadku prekompilowanych dystrybucji binarnych można po prostu rozpakować je do innych katalogów. Z kolei jeśli serwer kompilujesz samodzielnie, musisz ustawić opcję `CMAKE_INSTALL_PREFIX` dla CMake, aby wskazać inne położenia poszczególnym instalacjom.

Kolejne problemy mogą się pojawiać w trakcie uruchamiania serwera. Każdy proces serwera musi mieć unikalne wartości dla wielu parametrów. Na przykład, każdy serwer musi nasłuchiwać połączeń na innym porcie TCP/IP, aby uniknąć konfliktów. Dotyczy to zarówno uruchamiania różnych plików binarnych serwera, jak i wielu egzemplarzy tego samego serwera. Dokładnie taka sama zasada dotyczy pozostałych interfejsów połączenia: plików gniazd systemu UNIX, nazwanych potoków w Windows i pamięci współdzielonej. Po włączeniu rejestracji zdarzeń każdy serwer musi zapisywać informacje do własnego zestawu plików dzienników zdarzeń. Jeżeli różne serwery będą zapisywały informacje w tych samych plikach, kłopoty będą nieuniknione.

Opcje serwera można wskazać w trakcie jego uruchamiania, zwykle w pliku opcji. Ewentualnie, jeśli uruchamiasz wiele serwerów skompilowanych samodzielnie ze źródeł, w trakcie kompilacji możesz podać inny zestaw wartości parametrów używanych przez serwer. W ten sposób staną się one wbudowanymi wartościami domyślnymi i nie trzeba będzie ich wskazywać w trakcie uruchamiania. Jednak podczas kompilacji serwera ze źródeł jednym z powodów użycia plików opcji jest to, że parametry wyraźnie podane w ten sposób stanowią rodzaj dokumentacji sposobu konfiguracji serwera.

W zaprezentowanej tutaj analizie znajdzie się omówienie wielu rodzajów opcji, które mogą potencjalnie doprowadzić do konfliktów, jeśli nie zostaną ustawione dla poszczególnych serwerów. Niektóre opcje wpływają na inne, a więc nie trzeba ich wyraźnie ustawiać dla każdego serwera. Na przykład, w trakcie uruchamiania każdy serwer musi mieć zdefiniowany unikalny zestaw plików dzienników zdarzeń. Ponieważ katalog danych serwera to domyślne położenie dla nich wszystkich, ustawienie poszczególnym serwerom innego katalogu danych pośrednio wpływa na zapisywanie plików dzienników zdarzeń w różnych katalogach.

Pewne opcje są w rzeczywistości zmiennymi systemowymi (na co wskazują znaki podkreślenia w nazwach), ale można je ustawić w trakcie uruchamiania serwera:

- Jeżeli uruchamiasz różne wersje serwera, każda dystrybucja jest zwykle instalowana w innym katalogu bazowym. Ponadto, każdy serwer ma własny katalog danych. (To jest obowiązkowe w Windows i gorąco zalecane w systemach UNIX). Aby wyraźnie zdefiniować wspomniane katalogi, należy użyć opcji wymienionych w tabeli 12.7.

Tabela 12.7. Opcje pozwalające na wskazanie położenia katalogów instalacji MySQL

Opcja	Opis
<code>--basedir=nazwa_katalogu</code>	Ścieżka dostępu do katalogu głównego instalacji MySQL.
<code>--datadir=nazwa_katalogu</code>	Ścieżka dostępu do katalogu danych.

W wielu przypadkach katalog danych to po prostu podkatalog katalogu bazowego, choć nie zawsze tak jest. Na przykład, dostawca usług internetowych może dostarczać klientom powszechny zestaw programów serwera MySQL i klienckich, ale dla każdego klienta uruchamiać egzemplarz serwera używający zdefiniowanego dla danego klienta katalogu danych. W takim przypadku katalog bazowy pozostanie taki sam dla wszystkich serwerów, natomiast poszczególne katalogi danych będą miały różne położenia, prawdopodobnie w katalogach domowych użytkowników.

- Wymienione w tabeli 12.8 opcje interfejsu sieciowego muszą mieć różne wartości dla poszczególnych serwerów, aby uniknąć sytuacji, w której wiele serwerów nasłuchuje na tym samym interfejsie.

Tabela 12.8. Opcje dotyczące interfejsów sieciowych, na których nasłuchuje MySQL

Opcja	Opis
<code>--port=numer_portu</code>	Numer portu dla połączeń TCP/IP.
<code>--socket=nazwa_pliku</code>	Ścieżka dostępu pliku gniazda domeny UNIX lub nazwa nazwanego potoku w Windows.
<code>--shared-memory-base-name=nazwa</code>	Nazwa pamięci współdzielonej używanej przez połączenia współdzielące pamięć (tylko Windows).

W systemie Windows opcje `--socket` i `--shared-memory-base-name` muszą być używane jedynie z serwerami uruchamianymi wraz z opcjami `--named-pipe` i `--shared-memory`, pozwalającymi na stosowanie połączeń za pomocą nazwanego potoku i pamięci współdzielonej. W takim przypadku jeden serwer może używać domyślnych nazw nazwanego potoku i pamięci współdzielonej (odpowiednio MySQL i MYSQL), natomiast pozostałe muszą korzystać z innych nazw.

- Każdy serwer musi używać innych nazw plików dla plików stanu (na przykład PID) oraz plików dzienników zdarzeń. W przeciwnym razie różne serwery będą rywalizowały o możliwość zapisu w tym samym pliku. W najlepszym przypadku zawartość takiego pliku będzie myląca, zaś w najgorszym uniemożliwi to prawidłowe działanie mechanizmu replikacji. Serwer tworzy pliki stanu i dzienników zdarzeń o nazwach wskazanych przez opcje wymienione w tabeli 12.9. W przypadku podania względnej ścieżki dostępu domyślne położenie wspomnianych plików to katalog danych MySQL. Jeżeli każdy serwer używa innego katalogu danych, nie trzeba podawać bezwzględnych ścieżek dostępu w poszczególnych serwerach, aby informacje były zapisywane w oddzielnych zestawach plików dzienników zdarzeń. (W podrozdziale 12.8, zatytułowanym „Dzienniki zdarzeń serwera”, znajdziesz więcej informacji na temat nadawania nazw plikom dzienników zdarzeń).

Tabela 12.9. Opcje dotyczące dzienników zdarzeń w MySQL

Opcja	Opis
<code>--general_log</code>	Włącza ogólny dziennik zdarzeń.
<code>--general_log_file=nazwa_pliku</code>	Nazwa pliku ogólnego dziennika zdarzeń.
<code>--log-bin[=nazwa_pliku]</code>	Włącza binarny dziennik zdarzeń.
<code>--log-bin-index=nazwa_pliku</code>	Plik indeksu binarnego dziennika zdarzeń.
<code>--log_error[=nazwa_pliku]</code>	Włącza dziennik błędów.
<code>--log_output[=położenie]</code>	Położenie dzienników: ogólnego i wolno wykonywanych zapytań.
<code>--pid_file=nazwa_pliku</code>	Plik identyfikatora procesu.
<code>--relay-log[=nazwa_pliku]</code>	Włączenie dziennika zdarzeń przekazywania.
<code>--relay-log-index=nazwa_pliku</code>	Plik indeksu dziennika zdarzeń przekazywania.
<code>--slow_query_log</code>	Włączenie dziennika wolno wykonywanych zapytań.
<code>--slow_query_log_file=nazwa_pliku</code>	Nazwa pliku dziennika wolno wykonywanych zapytań.

Pewne opcje są w rzeczywistości zmiennymi systemowymi (na co wskazują znaki podkreślenia w nazwach), ale można je ustawić podczas uruchamiania serwera. Wspomniane zmienne mogą być również zmienione w trakcie działania serwera.

- Serwery używane jako podległe w replikacji muszą mieć unikalne zestawy plików dzienników przekazywania i plików informacyjnych. Wspomniane pliki są domyślnie tworzone w katalogu danych i mogą być wyraźnie wskazane przez opcje `--master-info-file` i `--relay-log-info-file`.
- W systemach UNIX, jeśli do uruchomienia serwera jest używany skrypt `mysqld_safe`, powoduje on utworzenie dziennika błędów (domyślnie w katalogu danych). Aby wyraźnie wskazać nazwę dziennika błędów, należy użyć zmiennej systemowej o nazwie `log_error`. Alternatywne rozwiązanie polega na wysyłaniu błędów do `syslog`. Więcej informacji na ten temat znajdziesz w punkcie 12.8.1, zatytułowanym „Dziennik błędów”.
- Każdy serwer używający silnika InnoDB musi mieć skonfigurowaną własną systemową przestrzeń tabel. Ponadto, katalog, w którym InnoDB zapisuje swoje dzienniki zdarzeń, musi być unikalny dla każdego serwera. Domyślnie silnik InnoDB zapisuje dzienniki zdarzeń w katalogu danych MySQL. Aby zmienić to położenie, należy użyć zmiennej systemowej `innodb_log_group_home_dir`. Więcej informacji na ten temat znajdziesz w punkcie 12.5.3, zatytułowanym „Konfiguracja silnika InnoDB”.
- W systemach UNIX może być konieczne użycie opcji `--user` dla każdego serwera i tym samym wskazanie konta użytkownika, który będzie uruchamiał serwer. Takie rozwiązanie prawdopodobnie trzeba zastosować, dostarczając poszczególne egzemplarze serwera MySQL różnym użytkownikom, z których każdy jest właścicielem oddzielnego katalogu danych.
- W systemach Windows różne serwery instalowane jako usługi muszą mieć unikalne nazwy usług.

12.9.2. Konfiguracja i kompilacja różnych serwerów

Jeżeli różne wersje serwera kompilujesz ze źródeł, musisz je zainstalować w innych katalogach. Najłatwiejszym sposobem zachowania odrębności instalacji jest wskazanie różnych katalogów bazowych instalacji przez ustawienie `CMAKE_INSTALL_PREFIX` podczas uruchamiania CMake. Użycie nazwy katalogu bieżącego zawierającej wersję serwera bardzo ułatwia odróżnianie katalogów bazowych poszczególnych wersji MySQL. W tym punkcie poznasz jedno z możliwych rozwiązań. Omówione zostaną konwencje konfiguracyjne, których używam w celu posiadania oddzielnych instalacji MySQL.

Wszystkie instalacje MySQL umieszczam w jednym wspólnym katalogu `/var/mysql`. Aby zainstalować daną dystrybucję, tworzę podkatalog zawierający w nazwie wersję serwera. Na przykład, dla MySQL 5.5.25 utworzyłem podkatalog `/var/mysql/50525`, przeznaczony na bazowy katalog instalacyjny wymienionego serwera. W tym celu uruchomiłem CMake z opcją `-DCMAKE_INSTALL_PREFIX=/var/mysql/50525`. Stosuję także inne opcje określające kolejne wartości charakterystyczne dla danego serwera, na przykład numer portu TCP/IP i ścieżkę dostępu do pliku gniazda. W używanej przeze mnie konfiguracji numer portu

TCP/IP odpowiada numerowi wersji serwera, plik gniazda znajduje się w katalogu bazowym, a katalog danych nosi nazwę *data* i znajduje się w katalogu bazowym.

Aby skonfigurować wymienione opcje, korzystam ze skryptu powłoki o nazwie *config-ver*, którego zawartość jest następująca:

```
VERSION=50525
PREFIX="/var/mysql/$VERSION"
PORT="$VERSION"
ENGINES="
-DWITH_INNOBASE_STORAGE_ENGINE=ON
-DWITH_PARTITION_STORAGE_ENGINE=ON
-DWITH_PERFSCHEMA_STORAGE_ENGINE=ON
"
OTHER="
-DCPACK_MONOLITHIC_INSTALL=ON
-DENABLED_LOCAL_INFILE=ON
-DMYSQL_MAINTAINER_MODE=OFF
-DWITH_SSL=bundled
"
OPTIONS="
-DCMAKE_INSTALL_PREFIX=$PREFIX
-DMYSQL_DATADIR=$PREFIX/data
-DMYSQL_TCP_PORT=$PORT
-DMYSQL_UNIX_ADDR=$PREFIX/mysql.sock
$ENGINES $OTHER
"
cmake $OPTIONS .
```

Upewniam się, że w wierszu pierwszym został podany prawidłowy numer wersji, oraz modyfikuję pozostałe wartości zgodnie z wymaganiami opcjonalnych silników bazy danych, które mają zostać skompilowane itd. Po zakończeniu modyfikacji wymienione poniżej polecenia powodują przeprowadzenie konfiguracji, kompilacji i instalacji dystrybucji MySQL:

```
% sh config-ver
% make
% make install
```

Po instalacji danej wersji MySQL konieczna jest zmiana położenia jej katalogu bazowego i inicjalizacja katalogu danych oraz tabel uprawnień:

```
# cd /var/mysql/50525
# scripts/mysql_install_db --user=nazwa_uzytkownika
```

W powyższym poleceniu *nazwa_użytkownika* oznacza nazwę konta użytkownika, który będzie uruchamiał serwer (na przykład konto *mysql*). Wymienione polecenia trzeba wydać po zalogowaniu się jako użytkownik *root* lub o wskazanej nazwie.

Na tym etapie przeprowadzam procedurę zabezpieczenia katalogu instalacyjnego MySQL. Wspomniana procedura została pokrótce omówiona w podpunkcie 12.2.1.1, zatytułowanym „Działanie serwera w ramach pozbawionego uprawnień konta logowania”. Natomiast jej dokładne omówienie znajdziesz w punkcie 13.1.2, zatytułowanym „Zabezpieczenie instalacji MySQL”.

Po wykonaniu powyższych zadań pozostało już tylko skonfigurowanie pozostałych ustawień używanych w pliku opcji oraz sposobu uruchamiania serwera. Dokładne informacje na ten temat znajdziesz w punkcie 12.9.4, zatytułowanym „Użycie skryptu `mysqld_multi` do zarządzania serwerem”.

12.9.3. Strategie podawania opcji startowych

Po zainstalowaniu serwerów być może zastanawiasz się, w jaki sposób można je uruchomić wraz z prawidłowymi zestawami opcji. Do dyspozycji masz kilka rozwiązań:

- Jeżeli serwer kompilujesz samodzielnie, wtedy w trakcie kompilacji możesz zdefiniować zestaw wartości domyślnych dla serwera. W takim przypadku podczas jego uruchamiania nie trzeba podawać żadnych opcji. Wadą takiego rozwiązania jest brak wyraźnej informacji o parametrach używanych przez dany serwer.
- Podczas uruchamiania serwera opcje można podać w wierszu poleceń lub w pliku opcji. W przypadku stosowania dużej liczby opcji podawanie ich w wierszu poleceń jest niepraktyczne. Znacznie wygodniejsze będzie umieszczenie ich w pliku opcji, choć później nieco trudności sprawia nakazanie poszczególnym serwerom odczytania właściwych opcji. W tym zakresie są stosowane następujące strategie:
 - ◆ Użycie opcji `--defaults-file` w celu wskazania pliku, który serwer powinien odczytać, aby znaleźć wszystkie opcje, oraz wskazania oddzielnych plików dla poszczególnych serwerów. W ten sposób wszystkie opcje wymagane dla danego serwera można umieścić w pojedynczym pliku i przechowywać całą konfigurację serwera w jednym miejscu. (Zwróć uwagę, że podczas używania opcji `--defaults-file` nie są odczytywane żadne standardowe pliki opcji, na przykład `/etc/my.cnf`).
 - ◆ Umieszczenie opcji wspólnych dla wszystkich serwerów w globalnym pliku opcji, takim jak `/etc/my.cnf`, i użycie opcji `--defaults-extra-file` w wierszu poleceń do wskazania pliku zawierającego opcje dodatkowe dla danego serwera. Na przykład, można użyć grupy `[mysqld]` w pliku `/etc/my.cnf` do zdefiniowania opcji mających zastosowanie dla wszystkich serwerów. Wspomnianych opcji nie trzeba powtarzać w plikach opcji przeznaczonych dla poszczególnych serwerów.
 - ◆ Koniecznie należy się upewnić, że opcje umieszczone w grupie wspólnych opcji są obsługiwane przez wszystkie uruchamiane serwery. Na przykład, nie można użyć opcji `--plugin-load-add`, jeśli którykolwiek serwer jest w wersji starszej niż 5.6.3, ponieważ wymienioną opcję wprowadzono dopiero w wersji 5.6.3. Obecność tej opcji może spowodować, że próba uruchomienia starszej wersji serwera zakończy się niepowodzeniem. Ewentualnie, można użyć składni `loose-nazwa_opcji` w celu podania opcji, które mogą być nieobsługiwane przez pewne serwery. W takim przypadku serwer, który nie obsługuje danej opcji, po prostu ją zignoruje, umieści w dzienniku

komunikat ostrzeżenia i będzie kontynuował działanie. Więcej informacji na temat opcji `loose` znajdziesz w podrozdziale F.2, zatytułowanym „Określanie opcji programu”.

- ◆ Użycie skryptu `mysqld_multi` w celu zarządzania uruchamianiem wielu serwerów. Wymieniony skrypt pozwala na umieszczenie w pojedynczym pliku opcji dla wszystkich serwerów, ale powiązanie ich z poszczególnymi serwerami za pomocą oddzielnych grup opcji zdefiniowanych w tym pliku.
- W systemach Windows można uruchamiać wiele usług za pomocą specjalnej grupy w pliku opcji, stosując konwencje charakterystyczne dla stylu konfiguracji serwera. Zapoznaj się z podpunktem 12.2.2.2, zatytułowanym „Uruchomienie serwera jako usługi Windows”.

W kolejnych punktach zostaną pokazane pewne sposoby stosowania wymienionych strategii. Dowiesz się, jak używać skryptu `mysqld_multi` w systemach UNIX oraz jak uruchamiać wiele serwerów w Windows.

12.9.4. Użycie skryptu `mysqld_multi` do zarządzania serwerem

W systemach UNIX skrypty `mysqld_safe` i `mysql.server` są powszechnie używane w celu uruchomienia serwera; obie doskonale sprawdzają się w sytuacji, gdy w komputerze działa tylko jeden serwer. Aby nieco ułatwić obsługę wielu serwerów, można wykorzystać skrypt o nazwie `mysqld_multi`.

Skrypt `mysqld_multi` działa na bazie przypisania określonych liczb poszczególnym konfiguracjom serwerów, a następnie umieszcza te opcje serwera w grupie `[mysqldn]` pliku opcji, gdzie *n* jest wspomnianą liczbą. Plik opcji może zawierać także grupę `[mysqld_multi]`, zawierającą opcje dla samego skryptu `mysqld_multi`. Na przykład, jeżeli mam zainstalowane serwery MySQL w wersjach 5.1.64, 5.5.25 i 5.6.6, to mogę utworzyć grupy opcji o nazwach `[mysqld50164]`, `[mysqld50525]` i `[mysqld50606]` oraz w następujący sposób skonfigurować opcje w pliku `/etc/my.cnf`:

```
[mysqld50164]
basedir=/var/mysql/50164
datadir=/var/mysql/50164/data
mysqld=/var/mysql/50164/bin/mysqld_safe
socket=/var/mysql/50164/mysql.sock
port=50164
user=mysql
log_error=log.err
general_log
general_log_file=qlog
log-bin=binlog
skip-innodb

[mysqld50525]
basedir=/var/mysql/50525
datadir=/var/mysql/50525/data
```

```
mysqld=/var/mysql/50525/bin/mysqld_safe
socket=/var/mysql/50525/mysql.sock
port=50525
user=mysql
log_error=log.err
general_log
general_log_file=qlog
log-bin=binlog
innodb_data_file_path=ibdata1:50M:autoextend
event_scheduler=ON
```

```
[mysqld50606]
basedir=/var/mysql/50606
datadir=/var/mysql/50606/data
mysqld=/var/mysql/50606/bin/mysqld_safe
socket=/var/mysql/50606/mysql.sock
port=50606
user=mysql
log_error=log.err
general_log
general_log_file=qlog
log-bin=binlog
innodb_data_file_path=ibdata1:100M
lc_messages=fr_FR
character_set_server=utf8
```

Przedstawiony powyżej układ parametrów dla poszczególnych serwerów odpowiada konfiguracji katalogów przedstawionej w punkcie 12.9.2, zatytułowanej „Konfiguracja i kompilacja różnych serwerów”. Podane zostały także dodatkowe parametry charakterystyczne dla danego serwera, odpowiadające różnicom w typach dzienników zdarzeń, silników bazy danych itd.

W celu uruchomienia serwera należy wywołać skrypt `mysqld_multi` i podać w wierszu poleceń słowo kluczowe `start` i numer grupy opcji serwera:

```
% mysqld_multi --no-log start 50525
```

Opcja `--no-log` powoduje, że komunikaty stanu będą przekazywane do terminala zamiast do pliku dziennika. W ten sposób można znacznie łatwiej przekonać się, co tak naprawdę się dzieje. Istnieje możliwość podania więcej niż tylko jednego serwera, wtedy numery grupy trzeba podać w postaci listy rozdzielonej przecinkami. Zakres numerów grup można podać, rozdzielając liczby myślnikiem. Na liście serwerów nie mogą znajdować się znaki odstępu, na przykład:

```
% mysqld_multi --no-log start 50164,50525-50606
```

W celu zatrzymania serwerów lub otrzymania informacji stanu wskazujących, czy serwer działa, należy użyć słowa kluczowego `stop` lub `report` i listy serwerów. Począwszy od MySQL w wersji 5.6.3, można podać także słowo kluczowe `reload` (zatrzymanie i ponowne uruchomienie). Dla wymienionych poleceń skrypt `mysqld_multi` wywołuje narzędzie `mysqldadmin` w celu przeprowadzenia komunikacji z serwerem, a więc trzeba również podać nazwę użytkownika i hasło do konta administracyjnego:

```
% mysqld_multi --nolog --user=root --password=hasło_roota stop 50164
% mysqld_multi --nolog --user=root --password=hasło_roota report 50164,50606
```

Nazwa użytkownika i hasło muszą być akceptowane przez wszystkie serwery, które chcesz kontrolować za pomocą podanego polecenia. Skrypt `mysqld_multi` automatycznie próbuje ustalić położenie narzędzia `mysqladmin`. Możesz również wyraźnie podać jego ścieżkę dostępu w grupie `[mysqld_multi]` pliku opcji. Ponadto, możliwe jest podanie listy domyślnych danych konta administracyjnego w tej grupie opcji; będą wtedy używane wraz z poleceniami `stop` i `report`, na przykład:

```
[mysqld_multi]
mysqladmin=/usr/local/mysql/bin/mysqladmin
user=leeloo
password=multiPASS
```

Z punktu widzenia zapewnienia bezpieczeństwa zalecane jest podawanie danych konta administracyjnego (nazwa użytkownika i hasło) w pliku opcji zamiast ich ujawniania w wierszu poleceń. Po umieszczeniu hasła w pliku opcji upewnij się, że nie jest on publicznie dostępny! Zapoznaj się z podpunktem 13.1.2.2, zatytułowanym „Zabezpieczanie plików opcji”.

12.9.5. Uruchamianie wielu serwerów w Windows

W celu uruchomienia wielu serwerów w Windows można je uruchamiać ręcznie lub użyć wielu usług w Windows. Istnieje również możliwość połączenia obu podejść.

W celu ręcznego uruchomienia wielu serwerów należy utworzyć dla każdego z nich plik opcji zawierający odpowiednie parametry. Na przykład, aby uruchomić dwa serwery używające tego samego pliku binarnego, ale oddzielnych katalogów danych, trzeba utworzyć dwa pliki opcji. Przyjmujemy założenie, że MySQL zainstalowano w katalogu `C:\mysql` wraz z katalogiem danych `C:\mysql\data` i chcemy uruchamiać drugi egzemplarz serwera, używający katalogu danych `C:\mysql\data2`. Pliki opcji mogą przedstawiać się następująco:

```
Plik C:\my.ini1:
[mysqld]
basedir=C:/mysql
datadir=C:/mysql/data
port=3306
```

```
Plik C:\my.ini2:
[mysqld]
basedir=C:/mysql
datadir=C:/mysql/data2
port=3307
```

Katalog danych musi istnieć przed uruchomieniem drugiego serwera. Najłatwiejszym sposobem przygotowania katalogu `C:\mysql\data2` jest jego utworzenie jako kopii `C:\mysql\data`. Wydad poniższe polecenie, gdy serwer nie jest uruchomiony:

```
C:\> xcopy C:\mysql\data C:\mysql\data2 /E
```



```

C:\> C:\mysql\bin\mysqlld --install
C:\> net start MySQL
C:\> C:\mysql\bin\mysqlld --install mysqlsvc2
C:\> net start mysqlsvc2

```

Jeżeli podajesz nazwę usługi, możesz również podać `--defaults-file` jako ostatnią opcję wiersza poleceń, gdy tworzona jest usługa:

```

C:\> C:\mysql\bin\mysqlld --install nazwa_uslugi --defaults-file=nazwa_pliku

```

W ten sposób zyskujesz alternatywne rozwiązanie w zakresie podawania opcji charakterystycznych dla danego serwera. Nazwa pliku zostaje zapamiętana i jest używana przez serwer podczas jego uruchamiania. Serwer odczytuje jedynie opcje z grupy `[mysqlld]` tego pliku.

W celu zamknięcia serwerów należy użyć polecenia `mysqladmin shutdown`, `net stop` lub menedżera usług. Aby odinstalować usługi, należy zamknąć serwery, o ile działają, a następnie usunąć poszczególne usługi, podając opcję `--remove` i tę samą nazwę usługi, która została podana w trakcie instalacji serwera. Jeśli usługa ma nazwę domyślną (MySQL), wtedy można ją pominąć:

```

C:\> mysql --remove
C:\> mysql --remove mysqlsvc2

```

12.9.6. Uruchamianie klientów wielu serwerów

W celu użycia klienta, gdy uruchomionych jest wiele serwerów, należy nawiązać połączenie z żądanym serwerem, podając odpowiednie parametry połączenia. Dotyczy to również użycia narzędzia `mysqladmin` do zamknięcia serwerów. Na przykład, aby nawiązać połączenie z serwerem nasłuchującym na porcie 50525, należy wydać poniższe polecenie:

```
% mysql --protocol=tcp --port=50525
```

Aby nawiązać połączenie z serwerem działającym w Windows, używając pamięci współdzielonej o nazwie `mysqlsvc2`, należy wydać poniższe polecenie:

```
C:\> mysql --protocol=memory --shared-memory-base-name=mysqlsvc2
```

12.10. Uaktualnianie MySQL

Ponieważ serwer MySQL jest aktywnie rozwijany, uaktualnienia pojawiają się regularnie. W ten sposób rodzi się pytanie, kiedy administrator może uaktualnić instalację MySQL po pojawieniu się nowego wydania. Do podjęcia decyzji wykorzystaj przedstawione poniżej wskazówki.

Pierwszym zadaniem po wydaniu nowej wersji jest określenie, na ile różni się ona od poprzednich. Jeżeli chcesz na bieżąco być informowany o nowych wydaniach MySQL, zapisz się na listę dyskusyjną. (Informacje dotyczące list dyskusyjnych MySQL i sposobu zapisania się na nie znajdziesz na stronie <http://lists.mysql.com/>). Ogłoszenie o wydaniu nowej wersji zawiera listę zmian, a więc to dobry sposób, aby pozostać na bieżąco.

(Informacje o wprowadzonych zmianach znajdziesz też na stronie MySQL Release Notes). Ponadto, warto zapoznać się w podręczniku użytkownika z odpowiednim podrozdziałem dotyczącym uaktualniania instalacji. Znajdziesz tam ważne kwestie, które powinienś rozważyć, a także informacje o wszelkich krokach specjalnych koniecznych do podjęcia w trakcie uaktualniania serwera. Znajdujące się tam informacje są szczególnie ważne, gdy nowe wydanie wprowadza funkcje niezgodne z poprzednimi wydaniami.

Po sprawdzeniu informacji o nowym wydaniu i zapoznaniu się z podrozdziałem podręcznika użytkownika dotyczącym uaktualnień zadaj sobie następujące pytania:

- Czy spodziewasz się problemów lub wykorzystania luk w zabezpieczeniach istniejących w bieżącej wersji i poprawionych w nowej?
- Czy nowa wersja wprowadza funkcje, których chcesz lub musisz użyć?
- Czy w nowej wersji poprawiono wydajność operacji, które wykonujesz?

Jeżeli na wszystkie wymienione pytania odpowiedź jest negatywna, nie masz szczególnych powodów do przeprowadzenia uaktualnienia serwera. Jednak jeśli na niektóre pytania odpowiedziałeś twierdząco, wtedy możesz przystąpić do aktualizacji oprogramowania. Z reguły warto się z tym wstrzymać przez kilka dni i obserwować listy dyskusyjne poświęcone MySQL, aby przekonać się, co inni sądzą o danej wersji. Czy uaktualnienie przebiegło bez problemów? Czy zawiera jakieś błędy lub powoduje problemy?

Poniżej przedstawiono kilka innych czynników, których rozważenie może pomóc w podjęciu decyzji:

- Wydania wersji stabilnej najczęściej zawierają poprawki znalezionych błędów, a nie nowe funkcje. Ogólnie rzecz biorąc, mniejsze ryzyko wiąże się z uaktualnieniem wersji stabilnej niż wersji rozwojowych.
- Jeżeli uaktualniasz MySQL, może również wystąpić konieczność uaktualnienia innych programów zbudowanych na bazie utworzonej w języku C biblioteki klienta. Na przykład, po uaktualnieniu MySQL inne biblioteki lub aplikacje zależne od utworzonej w języku C biblioteki klienta mogą wymagać ponownego utworzenia i powiązania z nową biblioteką klienta. (Dotyczy to na przykład modułu Perl DBD::mysql i PHP. Oczwistym symptomem konieczności ponownego utworzenia modułów jest fakt, że wszystkie skrypty powiązane z DBD i PHP nie działają po uaktualnieniu MySQL). Jeżeli wolisz uniknąć wspomnianej konieczności ponownego utworzenia modułów, lepiej powstrzymaj się od uaktualnienia MySQL. W przypadku stosowania programów statycznie dołączających moduły zamiast ich dynamicznego linkowania prawdopodobieństwo wystąpienia wspomnianych problemów jest znacznie mniejsze, ale kosztem większego zużycia pamięci.

Jeśli nadal nie jesteś pewien, czy przeprowadzić uaktualnienie, możesz nowy serwer przetestować niezależnie od bieżącego. Wystarczy zainstalować go obok serwera produkcyjnego lub w zupełnie innym komputerze. Znacznie łatwiej zachować rozdzielnosc między serwerami, jeśli znajdują się w oddzielnych komputerach, ponieważ masz wtedy

swobodę w zakresie ich konfiguracji. Gdy zdecydujesz się na instalację nowego serwera obok istniejącego w tym samym komputerze, upewnij się, że skonfigurowałeś go z unikalnymi parametrami, takimi jak katalog instalacji, katalog danych i interfejs sieciowy, na którym serwer nasłuchuje połączeń. Szczegółowe informacje na ten temat znajdziesz w podrozdziale 12.9, zatytułowanym „Uruchamianie wielu serwerów MySQL”.

W każdym razie, nowy serwer testuj za pomocą kopii danych istniejących baz danych. Informacje dotyczące kopiowania baz danych znajdziesz w podrozdziale 14.4, zatytułowanym „Tworzenie kopii zapasowych bazy danych”.

W celu wygenerowania ciągłego źródła zapytań do wykonania przez serwer testowy rozważ użycie serwera produkcyjnego jako serwera głównego replikacji, natomiast testowy ustaw jako serwer podległy replikacji. W ten sposób uaktualnienia wykonywane przez serwer główny będą wysyłane do serwera podległego, zapewniając mu ciągły strumień danych wejściowych. Serwer główny nie wykonuje żadnych zapytań pobierających dane z serwera podległego, ale zawsze możesz wykorzystać program kliencki i wykonać zapytania SELECT, aby się przekonać, jak są przetwarzane przez nowy serwer.

Proszę, testuj wersje rozwojowe!

W środowisku produkcyjnym, na przykład do zarządzania zasobami biznesowymi, nie jest dobrym pomysłem testowanie wersji rozwojowych. Jednak zachęcam Cię do testowania nowych wydań rozwojowych za pomocą kopii danych produkcyjnych. Im więcej osób przetestuje nowe wydanie, tym lepsze ono będzie. Zwiększa się prawdopodobieństwo znalezienia błędów, co pozwala na ich usunięcie. Zgłoszenia błędów są ważnym czynnikiem pomagającym w posuwaniu naprzód prac nad rozwojem MySQL.

Bezpieczeństwo i kontrola dostępu

Jako administrator MySQL jesteś odpowiedzialny za zapewnienie bezpieczeństwa bazom danych, aby pozostały dostępne jedynie dla autoryzowanych użytkowników. Osiągnięcie tego celu wymaga zapewnienia bezpieczeństwa i nienaruszalności instalacji MySQL. W rozdziale 12., zatytułowanym „Ogólna administracja bazą danych MySQL”, poruszono już temat związany z bezpieczeństwem, a dokładnie wagę przypisywania haseł kontom użytkowników MySQL. Wymienione zagadnienie było omawiane przy okazji prezentacji procesu konfiguracji serwera. W tym rozdziale nieco bliżej przyjrzymy się kwestiom związanym z bezpieczeństwem:

- Dlaczego zapewnienie bezpieczeństwa jest tak ważne i przed jakimi rodzajami ataków trzeba się chronić?
- Bezpieczeństwo wewnętrzne oznacza stawianie czoła innym użytkownikom posiadającym konta w komputerze z działającym serwerem. Dowiesz się, co można z nimi zrobić za pomocą zarządzania dostępem do systemu plików.
- Bezpieczeństwo zewnętrzne oznacza stawianie czoła klientom nawiązującym połączenie z serwerem przez sieć. Dowiesz się, co można z nimi zrobić za pomocą zarządzania kontami użytkowników MySQL.

Kwestie związane z bezpieczeństwem wewnętrznym pojawiają się wraz z innymi użytkownikami, którzy mają bezpośredni dostęp do komputera zawierającego działający serwer, czyli po prostu użytkownikami z kontami logowania w komputerze. Ogólnie rzecz biorąc, bezpieczeństwo wewnętrzne ma zapewniać ochronę przed wykorzystaniem systemu plików w celu uzyskania nieautoryzowanego dostępu. Musisz więc chronić instalację MySQL przed atakami przez użytkowników posiadających konta w komputerze, w którym działa serwer. W szczególności, katalog danych serwera MySQL powinien być własnością użytkownika, który uruchamia serwer MySQL. Jeżeli tego nie zagwarantujesz, to inne zabezpieczenia będą mogły zostać złamane. Na przykład, wprowadzie musisz upewnić się o prawidłowej konfiguracji kont użytkowników MySQL wymienionych

w tabelach uprawnień kontrolujących dostęp klientów przez sieć, ale nienaruszalność wspomnianych tabel zależy od zastosowania odpowiedniej ochrony na poziomie systemu plików. Jeżeli uprawnienia dostępu do katalogu danych serwera będą zbyt duże, wtedy każdy będzie mógł zdefiniować zupełnie inną politykę dostępu klientów przez zastąpienie plików odpowiadających tabelom uprawnień.

Z kolei bezpieczeństwo zewnętrzne dotyczy połączeń przychodzących z sieci. Konieczna jest ochrona serwera MySQL przed atakami z zewnątrz sieci przez klientów żądających dostępu do zawartości bazy danych. Tabele uprawnień w MySQL powinny zostać skonfigurowane w taki sposób, aby uniemożliwić uzyskanie dostępu do baz danych zarządzanych przez serwer, o ile użytkownik nie poda prawidłowych danych uwierzytelniających (nazwa użytkownika i hasło). Inne niebezpieczeństwo wiąże się z ryzykiem monitorowania przez atakującego sieci i przechwytywania ruchu sieciowego między serwerem i klientem. Aby rozwiązać takie obawy, należy skonfigurować instalację MySQL w sposób zapewniający obsługę połączeń z użyciem protokołu SSL (ang. *Secure Sockets Layer*).

W tym rozdziale zostaną omówione kwestie związane z bezpieczeństwem. Powinieneś je poznać, co pozwoli Ci na ochronę serwera bazy danych przed nieautoryzowanym dostępem za pomocą sieci i systemu plików. W rozdziale często znajdziesz odniesienia do konta użytkownika, w ramach którego działa serwer MySQL, oraz innych zadań administracyjnych. Użyte w przykładach nazwa użytkownika i grupy wspomnianego konta to `mysql`. Jeżeli używasz innej nazwy użytkownika lub grupy (na przykład uruchamiasz serwer MySQL z poziomu konta logowania do komputera), wprowadź odpowiednie zmiany w zaprezentowanych przykładach.

13.1. Zabezpieczenie dostępu do MySQL przez system plików

W tym podrozdziale dowiesz się, jak zabezpieczyć instalację MySQL i tym samym uniemożliwić majstrowanie przy niej nieautoryzowanym użytkownikom komputera, w którym działa serwer. Przedstawione tutaj informacje dotyczą jedynie systemów UNIX, ponieważ przyjęto założenie, że jeśli serwer działa w systemie Windows, to masz pełną kontrolę nad komputerem i nie ma żadnych innych użytkowników lokalnych.

Procedura instalacyjna MySQL powoduje utworzenie wielu katalogów; część z nich wymaga ochrony przed innymi użytkownikami. Na przykład, nie ma żadnego powodu, aby program serwera był dostępny dla kogokolwiek innego niż konto użytkownika odpowiedzialnego za administrację MySQL. Z drugiej strony, programy klientów powinny być dostępne publicznie, aby inni użytkownicy mogli je uruchamiać, ale już nie modyfikować lub zastępować innymi.

Po początkowej instalacji tworzone są inne pliki wymagające ochrony. Wspomniane pliki będą tworzone przez Ciebie jako część procedury konfiguracyjnej przeprowadzanej po instalacji serwera lub przez sam serwer w trakcie jego działania. Tworzone przez Ciebie

pliki to między innymi pliki nagłówkowe i powiązane z SSL. Pliki i katalogi tworzone przez serwer obejmują katalogi bazy danych, odpowiadające tabelom pliki w katalogach baz danych, pliki dzienników zdarzeń oraz plik gniazda systemu UNIX.

Bez wątpienia chcesz zapewnić prywatność bazom danych przechowywanym przez serwer. Całkowicie zrozumiałe jest, że właściciel bazy danych najczęściej uznaje jej zawartość za prywatną. Nawet jeśli tak nie jest, ewentualne publiczne udostępnienie zawartości bazy danych powinno pozostać jego wyraźną decyzją, a nie skutkiem niewystarczającego zabezpieczenia katalogu danych serwera MySQL.

Bezpieczeństwo trzeba zapewnić również plikom dzienników zdarzeń, ponieważ zawierają treść zapytań wysyłanych przez klienty do serwera. Zawsze istnieje obawa, że osoba z dostępem do plików dzienników zdarzeń może monitorować zmiany wprowadzane w zawartości bazy danych. Jeszcze większe niebezpieczeństwo związane z plikami dzienników zdarzeń wynika z tego, że mogą one zawierać tekst wrażliwych zapytań wraz z hasłami. MySQL szyfruje hasła, ale dotyczy to połączenia nawiązanego po ustawieniu hasła. Proces ustawienia hasła obejmuje zapytania takie jak `CREATE USER`, `GRANT` lub `SET PASSWORD`; wymienione zapytania są rejestrowane w postaci zwykłego tekstu w pewnych dziennikach zdarzeń. Atakujący, który uzyska możliwość odczytu plików dzienników zdarzeń, może być w stanie odkryć informacje wrażliwe w dość prosty sposób, na przykład używając narzędzia `grep` w celu przeszukania pliku pod kątem słów takich jak `GRANT` lub `PASSWORD`.

Pewne pliki, na przykład gniazda systemu UNIX, muszą być dostępne dla programów klientów. Z reguły wystarczy udzielić dostępu do pliku, ale nie pełnej kontroli nad nim. Na przykład, użytkownik może nawiązać połączenie z serwerem za pomocą pliku gniazda, ale nie może usunąć wspomnianego pliku i uniemożliwić innym użytkownikom nawiązania połączenia z serwerem.

13.1.1. Jak ukraść dane?

Poniżej zaprezentowano krótki przykład pokazujący, dlaczego zapewnienie bezpieczeństwa jest tak ważne. Przekonasz się, że inni użytkownicy nie powinni mieć bezpośredniego dostępu do katalogu danych MySQL.

Serwer MySQL oferuje elastyczny system uprawnień zaimplementowany w postaci tabel uprawnień znajdujących się w bazie danych `mysql`. Zawartość wspomnianych tabel stworzysz, konfigurując klientom uprawnienia dostępu do bazy danych, co zapewnia ochronę przed uzyskaniem nieautoryzowanego dostępu do danych. Jednak przygotowanie dobrych zabezpieczeń przed dostępem do bazy danych z sieci będzie daremne, jeśli inni użytkownicy komputera będą mieli bezpośredni dostęp do katalogu danych MySQL. Jeżeli nie jesteś jedyną osobą posiadającą dostęp do komputera, w którym działa serwer MySQL, musisz brać pod uwagę możliwość, że inny użytkownik komputera uzyska dostęp do katalogu danych MySQL.

Oczywiście, nie chcesz, aby inni użytkownicy komputera mieli bezpośrednią możliwość zapisu plików w katalogu danych MySQL, ponieważ w ten sposób mogą napisać wszystkie pliki stanu i tabel baz danych. Jednak bezpośrednio uprawnienia

do odczytu plików są również niebezpieczne. Jeżeli użytkownik uzyska możliwość odczytu plików tabel, to bardzo łatwo będzie mógł je ukraść i tym samym pobrać zawartość baz danych MySQL. W jaki sposób może to zrobić? Oto jedna z możliwych sytuacji:

1. Instalacja własnego serwera MySQL w komputerze, ale ze skonfigurowanym innym niż w oficjalnym serwerze numerem portu, plikiem gniazda i katalogiem danych.
2. Uruchomienie `mysql_install_db` w celu inicjalizacji katalogu danych. W ten sposób zyskuje się pełne uprawnienia dostępu do serwera jako użytkownik MySQL root. Powstaje również baza danych test, która będzie wygodnym repozytorium dla ukradzionych tabel.
3. Uzyskanie dostępu do katalogu danych atakowanego serwera, kopiowanie plików odpowiadających tabelom, które chcesz ukraść i umieścić w katalogu test w katalogu danych nowo zainstalowanego serwera. Ta operacja wymaga jedynie uprawnień odczytu do katalogu danych oficjalnego serwera.
4. Uruchomienie nowo zainstalowanego serwera. Proszę bardzo! Baza danych test zawiera teraz kopię ukradzionych tabel, do których masz swobodny dostęp. Zapytanie `SHOW TABLES FROM test` wyświetla dostępne tabele, natomiast `SELECT *` całą zawartość dowolnej z nich.
5. Wyjątkowo złośliwy atakujący może teraz udzielić uprawnień kontu anonimowego użytkownika w nowo zainstalowanym serwerze. W ten sposób każdy, kto nawiąże z nim połączenie, będzie miał dostęp do bazy danych. W efekcie oznacza to upublicznienie zawartości ukradzionych tabel.

Zastanów się przez chwilę nad powyższym scenariuszem, a następnie odwróć role. Czy naprawdę chcesz, aby ktokolwiek w taki sposób potraktował Twoje tabele? Oczywiście, że nie. Dlatego też powinieneś chronić serwer bazy danych, korzystając z przedstawionych poniżej informacji.

13.1.2. Zabezpieczenie instalacji MySQL

Przedstawiona tutaj procedura pokazuje, w jaki sposób skonfigurować własność i tryb dostępu dla plików i katalogów tworzących instalację MySQL. Przyjęto tutaj założenie, że `mysql` to nazwa użytkownika (i grupy), który jest właścicielem tej instalacji. (Kimkolwiek ten użytkownik jest, to z powodów przedstawionych w podpunkcie 12.2.1.1, zatytułowanym „Działanie serwera w ramach pozbawionego uprawnień konta logowania”, powinien być innym użytkownikiem niż root). Przyjęto również założenie, że wszystkie komponenty instalacji MySQL znajdują się w pojedynczym katalogu bazowym, a nie są rozsiane po całym systemie plików. W omawianym przykładzie bazowy katalog instalacji to `/usr/local/mysql`, natomiast katalog danych to `/usr/local/mysql/data`.

Po omówieniu procedury dowiesz się, jak zapewnić obsługę pewnych niestandardowych rodzajów instalacji. Układ Twojego systemu może różnić się od przedstawionego tutaj, ale powinieneś móc odpowiednio zaadaptować reguły zaprezentowane w tym rozdziale.

Nazwy i ścieżki dostępu zmieniasz na stosowane w Twoim systemie. Jeżeli uruchamiasz wiele systemów, procedurę powinienś przeprowadzić dla każdego z nich.

Za pomocą polecenia `ls -la` możesz sprawdzić, czy katalog danych zawiera niezabezpieczone pliki lub katalogi. W danych wyjściowych wymienionego polecenia zwróć uwagę na katalogi posiadające ustawione uprawnienia dla grupy lub innych użytkowników. Poniżej pokazano przykład niezabezpieczonego katalogu danych z powodu niektórych katalogów bazy danych:

```
% ls -la /usr/local/mysql/data
total 10148
drwxrwxr-x 11 mysql wheel 1024 May 8 12:20 .
drwxr-xr-x 22 root wheel 512 May 8 13:31 ..
drwx----- 2 mysql mysql 512 Apr 16 15:57 menagerie
drwxrwxr-x 2 mysql wheel 512 Jun 25 1998 mysql
drwx----- 7 mysql mysql 1024 May 7 10:45 sampdb
drwxrwxr-x 2 mysql wheel 1536 Jun 25 1998 test
drwx----- 2 mysql mysql 1024 May 8 18:43 tmp
```

Kropka oznacza wyświetlany katalog, czyli w omawianym przykładzie `/usr/local/mysql/data`. Z kolei dwie kropki oznaczają katalog nadrzędny, którym tutaj jest `/usr/local/mysql`. Część katalogów baz danych ma właściwe uprawnienia: `drwx-----`, dające uprawnienia odczytu, zapisu i uruchamiania właścicielowi oraz żadnym uprawnieniom pozostałym użytkownikom. Jednak widzimy także katalogi z bardzo szerokimi uprawnieniami `drwxrwxr-x`, dającymi możliwość odczytu, zapisu i uruchamiania wszystkim użytkownikom, nawet spoza grupy `mysql`. Przedstawiona tutaj sytuacja jest skutkiem działań prowadzonych na przestrzeni czasu — bardzo stara instalacja MySQL była wielokrotnie uaktualniana do nowych wersji. Mniej restrykcyjne uprawnienia były nadawane przez starsze wersje serwerów MySQL, stosujące mniej rygorystyczną politykę w zakresie nadawania uprawnień niż nowsze wersje serwera. (Jak widać w powyższych danych wyjściowych, katalogi baz danych o bardziej rygorystycznych uprawnieniach — `menagerie`, `sampdb` i `tmp` — mają znacznie nowsze daty). Obecne wersje serwera MySQL nadają katalogom bazy danych restrykcyjne uprawnienia — są dostępne jedynie dla użytkownika, który uruchamia serwer.

Polecenie `ls -la` można wykorzystać także do sprawdzenia katalogu bazowego instalacji MySQL, czyli `/usr/local/mysql`. Na przykład, otrzymane dane wyjściowe mogą być następujące:

```
% ls -la /usr/local/mysql
total 44
drwxrwxr-x 13 mysql mysql 1024 May 7 10:45 .
drwxr-xr-x 24 root wheel 1024 May 1 12:54 ..
drwxr-xr-x 2 mysql mysql 1024 Jul 16 20:58 bin
drwxrwxr-x 12 mysql wheel 1024 May 8 12:20 data
drwxr-xr-x 3 mysql mysql 512 May 7 10:45 etc
drwxr-xr-x 3 mysql mysql 512 May 7 10:45 include
drwxr-xr-x 3 mysql mysql 512 May 7 10:45 lib
drwxr-xr-x 3 mysql mysql 512 May 7 10:45 man
drwxr-xr-x 6 mysql mysql 1024 May 7 10:45 mysql-test
drwxr-xr-x 3 mysql mysql 512 May 7 10:45 scripts
drwxr-xr-x 3 mysql mysql 512 May 7 10:45 share
drwxr-xr-x 7 mysql mysql 1024 May 7 10:45 support-files
```

Zastosuj przedstawioną poniżej procedurę, aby wszystkie pliki stały się dostępne jedynie dla użytkownika `mysql`, poza tymi komponentami instalacji, do których autoryzowania użytkownicy muszą mieć dostęp. Na przykład, jak wcześniej wspomniano, uprawnienia dostępu do katalogu `data` muszą być znacznie bardziej restrykcyjne.

Zwróć uwagę, że pewne fragmenty prezentowanej tutaj procedury *nie mają zastosowania*, jeśli serwer MySQL i programy klienckie są zainstalowane w ogólnych katalogach systemowych wraz z programami innymi niż MySQL. (Taka sytuacja zdarza się najczęściej po instalacji MySQL za pomocą pakietów RPM). Na przykład, serwer może być umieszczony w katalogu `/usr/sbin`, natomiast programy klientów w `/usr/bin`. W takim przypadku właściciel i tryb dostępu do programów MySQL powinien być taki sam jak innych programów we wspomnianych katalogach.

1. Jeżeli serwer MySQL jest uruchomiony, trzeba go zatrzymać.

```
% mysqladmin -p -u root shutdown
```

2. Ustaw nazwę właściciela i grupy dla całej instalacji MySQL; to powinien być użytkownik konta administracyjnego MySQL. Użyj do tego poniższych poleceń, które musisz wydać jako użytkownik `root`:

```
# chown -R mysql /usr/local/mysql
# chgrp -R mysql /usr/local/mysql
```

Inne popularne podejście polega na określeniu całości jako należącej do użytkownika `root`, poza katalogiem danych. W takim przypadku należy wydać poniższe polecenia:

```
# chown -R root /usr/local/mysql
# chgrp -R mysql /usr/local/mysql
# chown -R mysql /usr/local/mysql/data
# chgrp -R mysql /usr/local/mysql/data
```

Jeżeli oddasz wszystko użytkownikowi `root`, wtedy większość kolejnych kroków będziesz musiał przeprowadzić jako `root`. W przeciwnym razie wykonaj je jako użytkownik `mysql`.

3. W przypadku katalogu bazowego i jego podkatalogów, do których dostęp powinny mieć klienty, zmień ich uprawnienia w taki sposób, aby użytkownik `mysql` miał pełny dostęp, natomiast pozostali jedynie uprawnienia odczytu i uruchamiania. Domyślnie może być już zastosowana wspomniana konfiguracja; jeśli tak nie jest, zmień ją. Na przykład, konfigurację katalogu bazowego możesz przeprowadzić za pomocą jednego z poniższych poleceń:

```
% chmod 755 /usr/local/mysql
% chmod u=rwx,go=rx /usr/local/mysql
```

Podobnie, katalog `bin` zawierający programy klienckie powinien mieć ustawione uprawnienia za pomocą jednego z poniższych poleceń:

```
% chmod 755 /usr/local/mysql/bin
% chmod u=rwx,go=rx /usr/local/mysql/bin
```


4. Zmień uprawnienia dla katalogu danych oraz znajdujących się w nim plików i katalogów w taki sposób, aby pozostały prywatne dla użytkownika mysql:

```
% chmod -R go-rwx /usr/local/mysql/data
```

W ten sposób bezpośredni dostęp do zawartości katalogu danych będzie miał jedynie użytkownik używany do uruchamiania serwera MySQL.

Po przeprowadzeniu powyższej procedury katalog instalacyjny MySQL będzie miał właściciela i uprawnienia podobne do przedstawionych poniżej:

```
% ls -la /usr/local/mysql
total 44
drwxr-xr-x 13 mysql mysql 1024 May 7 10:45 .
drwxr-xr-x 24 root wheel 1024 May 1 12:54 ..
drwxr-xr-x 2 mysql mysql 1024 Jul 16 20:58 bin
drwx----- 12 mysql mysql 1024 May 8 12:20 data
drwxr-xr-x 3 mysql mysql 512 May 7 10:45 include
drwxr-xr-x 3 mysql mysql 512 May 7 10:45 lib
drwxr-xr-x 3 mysql mysql 512 May 7 10:45 man
drwxr-xr-x 6 mysql mysql 1024 May 7 10:45 mysql-test
drwxr-xr-x 3 mysql mysql 512 May 7 10:45 scripts
drwxr-xr-x 3 mysql mysql 512 May 7 10:45 share
drwxr-xr-x 7 mysql mysql 1024 May 7 10:45 support-files
```

Jak widać, właścicielem wszystkiego jest użytkownik mysql należący do grupy mysql. Wyjątkiem jest katalog .., odwołujący się do katalogu nadrzędnego dla /usr/local/mysql. Właścicielem katalogu nadrzędnego jest użytkownik root i tylko on może przeprowadzać jakiegokolwiek modyfikacje, co jest dobrym rozwiązaniem. Nie chcesz przecież, aby nieautoryzowani użytkownicy mieli możliwość wprowadzania zmian w katalogu zawierającym instalację MySQL.

Katalog danych znajdujący się w katalogu bazowym ma jeszcze bardziej restrykcyjne uprawnienia:

```
% ls -la /usr/local/mysql/data
total 10148
drwx----- 11 mysql mysql 1024 May 8 12:20 .
drwxr-xr-x 22 mysql mysql 512 May 8 13:31 ..
drwx----- 2 mysql mysql 512 Apr 16 15:57 menagerie
drwx----- 2 mysql mysql 512 Jun 25 1998 mysql
drwx----- 7 mysql mysql 1024 May 7 10:45 sampdb
drwx----- 2 mysql mysql 1536 Jun 25 1998 test
drwx----- 2 mysql mysql 1024 May 8 18:43 tmp
```

Tutaj dwie kropki odnoszą się do katalogu nadrzędnego dla katalogu danych, czyli do katalogu bazowego instalacji MySQL.

Wyjątki od polityki „tylko mysql” w kwestii dostępu do katalogu danych mogą być potrzebne w przypadku określonych plików. Na przykład, jeśli katalog danych zawiera plik gniazda systemu UNIX, konieczne może być ustawienie nieco mniej restrykcyjnych uprawnień do tego katalogu. W przeciwnym razie programy klientów nie będą mogły nawiązać połączenia z serwerem za pomocą pliku gniazda. Rozwiązanie alternatywne polega na umieszczeniu pliku gniazda systemu UNIX w innym położeniu, na przykład

katalogu bazowym. Ta sama zasada ma zastosowanie także względem innych plików, do których programy inne niż `mysqld` muszą mieć dostęp, na przykład plików opcji zawierających globalne parametry klienta. (Więcej informacji na temat implementacji wymienionych podejść znajdziesz w podpunkcie 13.1.2.1, zatytułowanym „Zabezpieczanie pliku gniazda w systemach UNIX”).

Jak już wcześniej wspomniano, w przedstawionej powyżej procedurze przyjęto założenie, że wszystkie pliki powiązane z MySQL znajdują się w pojedynczym katalogu bazowym. Jeśli jest inaczej, musisz odszukać każdy katalog związany z MySQL i przeprowadzić względem niego odpowiednie operacje. Na przykład, jeśli katalog danych to `/var/mysql/data` zamiast `/usr/local/mysql`, wtedy prawidłowa zmiana właściciela instalacji wymaga wydania poniższych poleceń:

```
# chown -R mysql /usr/local/mysql
# chgrp -R mysql /usr/local/mysql
# chown -R mysql /var/mysql/data
# chgrp -R mysql /var/mysql/data
```

Załóżmy na przykład, że w katalogu instalacyjnym MySQL tworzymy katalog *innodb* przeznaczony do przechowywania wszystkich plików powiązanych z InnoDB. Domyślnie, wymienione pliki są umieszczane w katalogu danych MySQL. Jeżeli umieścisz je w wymienionym katalogu *innodb*, jego uprawnienia dostępu powinny być takie same jak do katalogu danych MySQL. Ta zasada obowiązuje również po przeniesieniu innych plików, które standardowo znajdują się w katalogu danych MySQL, na przykład plików dzienników zdarzeń.

Inna komplikacja następuje, gdy dowolny z podkatalogów znajdujących się w katalogu głównym instalacji tak naprawdę będzie dowiązaniem symbolicznym prowadzącym do katalogu w zupełnie innym położeniu. Jeżeli używane wersje poleceń `chown` i `chgrp` nie podążają za dowiązaniami symbolicznymi, musisz samodzielnie sprawdzić, dokąd prowadzą, i zmienić uprawnienia tych katalogów docelowych. Jednym z możliwych rozwiązań jest zastosowanie polecenia `find`:

```
# find /usr/local/mysql -follow -print | xargs chown mysql
# find /usr/local/mysql -follow -print | xargs chgrp mysql
```

Podobne rozważania mają zastosowanie w zakresie zmiany uprawnień. Na przykład, jeśli w katalogu danych MySQL znajdują się dowiązania symboliczne, a polecenie `chmod` za nimi nie podąża, wtedy użyj poniższego polecenia:

```
# find /usr/local/mysql/data -follow -print | xargs chmod go-rwx
```

Ponieważ na tym etapie uprawnienia do katalogu danych są zdefiniowane w taki sposób, że jest on dostępny jedynie dla użytkownika `mysql`, to powinieneś się upewnić, że od tej chwili serwer jest uruchamiany tylko przez użytkownika `mysql`. Łatwe rozwiązanie polega na wskazaniu tego użytkownika w sekcji `[mysqld]` pliku konfiguracyjnego `/etc/my.cnf` lub innym pliku *my.cnf* odczytywanym przez serwer podczas jego uruchamiania:

```
[mysqld]
user=mysql
```

W ten sposób serwer zawsze będzie uruchamiany przez użytkownika `mysql`, niezależnie od tego, jakiego konta użyjesz w celu zalogowania się do systemu. Więcej informacji na temat uruchamiania serwera za pomocą określonego konta logowania znajdziesz w podpunkcie 12.2.1.1, zatytułowanym „Działanie serwera w ramach pozbawionego uprawnień konta logowania”.

Po zabezpieczeniu instalacji MySQL uruchom ponownie serwer.

13.1.2.1. Zabezpieczanie pliku gniazda w systemach UNIX

Serwer używa pliku gniazda systemu UNIX dla połączeń nawiązywanych przez klientów z komputerem lokalnym (`localhost`). Plik gniazda jest standardowo publicznie dostępny i każdy program klienta może go używać. Jednak nie powinien znajdować się w katalogu, do którego programy klienckie mają uprawnienia usuwania. Na przykład, często zdarza się tworzenie pliku gniazda w katalogu `/tmp`, choć niektóre dystrybucje systemów UNIX definiują dla wymienionego katalogu uprawnienia pozwalające użytkownikom na usuwanie ich plików. Oznacza to, że użytkownik może usunąć plik gniazda i tym samym uniemożliwić programom klienckim nawiązywanie połączeń `localhost` aż do chwili ponownego uruchomienia serwera, ponieważ dopiero wtedy nastąpi ponowne utworzenie pliku gniazda. Znacznie lepiej jest, jeśli katalog `/tmp` ma ustawiony tak zwany bit lepkości (ang. *sticky bit*) i wówczas, nawet mając możliwość tworzenia plików w tym katalogu, użytkownik będzie mógł usuwać jedynie własne pliki. Ustawienie bitu lepkości dla katalogu następuje przez wydanie poniższego polecenia jako użytkownik `root`:

```
# chmod +t /tmp
```

Pewne instalacje umieszczają plik gniazda w katalogu danych MySQL, co prowadzi do problemów, jeśli wspomniany katalog ma być prywatny i przeznaczony wyłącznie dla użytkownika `mysql`: żaden program klienta nie będzie mógł uzyskać dostępu do pliku gniazda, o ile nie zostanie uruchomiony przez użytkownika `root` lub `mysql`. W takim przypadku rozwiązaniem jest użycie nieco mniej rygorystycznych uprawnień do katalogu danych. Aby programy klienckie mogły uzyskać dostęp do pliku gniazda bez zapewniania pełnych uprawnień odczytu katalogu danych MySQL, należy wydać poniższe polecenie:

```
% chmod go+x /usr/local/mysql/data
```

Aby uniknąć utworzenia katalogu danych w ten sposób, inne podejście polega na zmianie katalogu, w którym serwer tworzy plik gniazda. Na przykład, możesz skonfigurować MySQL w celu tworzenia pliku gniazda w katalogu bazowym, czyli ścieżka dostępu do pliku będzie następująca: `/usr/local/mysql/mysql.sock`. Położenie należy wskazać w globalnym pliku opcji lub ponownie skompilować serwer ze źródeł i podać wybrane położenie jako domyślne. Jeżeli planujesz użycie pliku opcji, upewnij się, że podałeś położenie pliku gniazda w grupie zarówno serwera, jak i klientów:

```
[mysqld]
socket=/usr/local/mysql/mysql.sock
```

```
[client]
socket=/usr/local/mysql/mysql.sock
```

Ponowna kompilacja serwera ze źródeł oznacza nieco więcej pracy, ale powoduje przygotowanie kompletnego rozwiązania, ponieważ użycie pliku opcji sprawdza się jedynie w przypadku programów klienckich korzystających z tego pliku. (Wszystkie standardowe programy klienckie dostarczane z MySQL korzystają z pliku opcji, ale programy opracowane przez firmy trzecie wcale nie muszą). Po przeprowadzeniu ponownej kompilacji nowe położenie pliku gniazda będzie domyślnie stosowane przez bibliotekę klienta. Dlatego też każdy program oparty na bibliotece klienta będzie używał wskazanego położenia jako domyślnego, niezależnie od tego, czy odczytuje pliki opcji, czy ich nie odczytuje.

13.1.2.2. Zabezpieczanie plików opcji

Pliki opcji to potencjalne zagrożenie, jeśli zawierają opcje, które nie powinny być ujawniane:

- Jeżeli plik opcji zawiera informacje wrażliwe, takie jak nazwa użytkownika i hasło konta MySQL, to nie powinien być dostępny publicznie.
- Standardowo, plik `/etc/my.cnf` jest dostępny publicznie, ponieważ stanowi miejsce, w którym najczęściej są umieszczane globalne opcje klienta. Dlatego też nie należy używać go do zdefiniowania opcji serwera, takich jak hasła replikacji.
- Każdy plik `.my.cnf` dla danego użytkownika powinien być własnością jedynie użytkownika, w którego katalogu domowym się znajduje, i tylko dla niego być dostępny. Aby takie uprawnienia nadać posiadanemu plikowi, należy w katalogu domowym wydać poniższe polecenie:

```
% chmod u=rw,go-rwx .my.cnf
```

- Inne pliki opcji muszą mieć ustawione uprawnienia zgodnie z ich wymaganiami.

Jednym ze sposobów zagwarantowania, że plik opcji dla danego użytkownika nie będzie miał nieprawidłowych uprawnień, jest uruchomienie programu wyszukującego pliki `.my.cnf` w katalogach domowych użytkowników i poprawiającego wszelkie nieprawidłowości. Tak właśnie działa przedstawiony poniżej skrypt Perl o nazwie `chk_mysql_opt_files.pl`:

```
#!/usr/bin/perl
# chk_mysql_opt_files.pl - Skrypt sprawdza pliki .my.cnf poszczególnych użytkowników
# i gwarantuje, że mają prawidłowe uprawnienia. Właścicielem każdego pliku powinien być
# użytkownik, w którego katalogu domowym znajduje się dany plik. Grupa oraz pozostali
# użytkownicy nie powinni mieć żadnych uprawnień do pliku.

# Skrypt trzeba uruchomić jako użytkownik root. Dane wejściowe skryptu to plik haseł. Jeżeli hasła
# użytkownika znajdują się w pliku /etc/passwd file, skrypt uruchom w następujący sposób:
# chk_mysql_opt_file.pl /etc/passwd

use strict;
use warnings;

while (<>)
{
    next if /^#/ || /\s*$/;          # Pomińcie komentarzy, pustych wierszy.
    my ($uid, $home) = (split (/:/, $_))[2,5];
```

```

my $cnf_file = "$home/.my.cnf";
next unless -f $cnf_file;           # Czy katalog zawiera plik .my.cnf file?
if ((stat ($cnf_file))[4] != $uid) # Sprawdzenie, kto jest właścicielem pliku.
{
    warn "Zmiana właściciela pliku $cnf_file na $uid\n";
    chown ($uid, (stat ($cnf_file))[5], $cnf_file);
}
my $mode = (stat ($cnf_file))[2];
if ($mode & 077)                     # Sprawdzenie uprawnień grupy i pozostałych użytkowników.
{
    warn sprintf ("Zmiana uprawnień %s z %o na %o\n",
                  $cnf_file, $mode, $mode & ~077);
    chmod ($mode & ~077, $cnf_file);
}
}

```

Skrypt *chk_mysql_opt_files.pl* znajdziesz w katalogu *admin* dystrybucji *sampdb*. Skrypt musi być uruchomiony przez użytkownika z uprawnieniami *root*, aby mógł wprowadzić zmiany w uprawnieniach plików należących do innych użytkowników systemu. W celu automatycznego uruchamiania skryptu skonfiguruj codziennie wykonywane przez użytkownika *root* zadanie *cron*.

13.2. Zarządzanie kontami użytkowników w MySQL

Jako administrator MySQL musisz wiedzieć, w jaki sposób konfigurować konta, wskazując tym samym użytkowników, którzy mogą nawiązać połączenie z serwerem, oraz określając, skąd mogą je nawiązać i co mogą zrobić po jego nawiązaniu. Wspomniane informacje MySQL przechowuje w tabelach uprawnień bazy danych *mysql*. Uprawnienia te można zdefiniować za pomocą poniższych zapytań SQL:

- Zapytania *CREATE USER*, *DROP USER* i *RENAME USER* powodują utworzenie, usunięcie lub zmianę nazwy konta MySQL.
- Zapytanie *GRANT* pozwala na zdefiniowanie uprawnień konta (i utworzenie konta, o ile jeszcze nie istnieje).
- Zapytanie *REVOKE* pozwala na odebranie uprawnień konta MySQL.
- Zapytanie *SET PASSWORD* pozwala na zdefiniowanie hasła dla konta MySQL.
- Zapytanie *SHOW GRANTS* powoduje wyświetlenie uprawnień zdefiniowanych dla wskazanego konta.

Wymienione zapytania są wykonywane względem tabel uprawnień w bazie danych *mysql*, które zostały wymienione w tabeli 13.1.

Inna tabela uprawnień, o nazwie *host*, nie jest używana podczas wykonywania zapytań dotyczących zarządzania kontami użytkowników; została uznana za zbędną i nie będzie tutaj omawiana.

Tabela 13.1. Tabele uprawnień w MySQL

Tabela uprawnień	Zawartość tabeli
user	Dane użytkowników, którzy mogą nawiązać połączenie z serwerem, i ich globalne uprawnienia.
db	Uprawnienia baz danych.
tables_priv	Uprawnienia tabel.
columns_priv	Uprawnienia kolumn.
procs_priv	Uprawnienia procedur składowanych.
proxies_priv	Uprawnienia użytkowników proxy.

Kiedy wykonujesz zapytanie `CREATE USER`, wskazujesz nazwę konta i opcjonalnie informacje dotyczące uwierzytelnienia (hasło lub metodę autoryzacji), a serwer tworzy w tabeli `user` rekord dla danego konta. To samo dotyczy także zapytania `GRANT`, jeśli wskazane w nim konto użytkownika jeszcze nie istnieje. W przypadku zapytania `GRANT`, jeśli zapytanie wskazuje jakiekolwiek uprawnienia globalne (to znaczy uprawnienia administracyjne lub dotyczące wszystkich baz danych), zostaje ono również zapisane w tabeli `user`. Jeżeli zapytanie `GRANT` podaje uprawnienia charakterystyczne dla danej bazy danych, tabeli, kolumny lub procedury składowanej, wtedy zostaje ono zapisane w tabeli `db`, `tables_priv`, `columns_priv` lub `procs_priv`. Uprawnienia `PROXY` są zapisywane w tabeli `proxies_priv`. Zapytanie `REVOKE` powoduje usunięcie uprawnień z tabel uprawnień, natomiast `DROP USER` usuwa z tabel wszystkie rekordy powiązane z danym użytkownikiem.

W kolejnych punktach dowiesz się, jak tworzyć i usuwać konta użytkowników MySQL, nadawać i odbierać przywileje, a także jak zmieniać i zerować hasła. Istnieje również możliwość bezpośredniego manipulowania zawartością tabel uprawnień za pomocą zapytań `INSERT` i `UPDATE`. Jednak przeznaczone do zarządzania kontami zapytania, takie jak `CREATE USER` i `GRANT`, są pod względem koncepcyjnym znacznie wygodniejsze do pracy, ponieważ dokładnie opisują modyfikacje, które mają zostać wprowadzone, a serwer automatycznie mapuje żądania na odpowiednie zmiany wprowadzane w tabelach uprawnień. Wprawdzie znacznie łatwiej jest wykonać zapytanie `CREATE USER` i `GRANT`, niż bezpośrednio modyfikować tabele uprawnień, ale zalecam zapoznanie się z podrozdziałem 13.3, zatytułowanym „Struktura i zawartość tabel uprawnień”. W wymienionym podrozdziale znajdziesz znacznie dokładniejszą analizę tabel uprawnień, co powinno pomóc Ci w zrozumieniu ich działania „pod maską”, czyli poniżej poziomu poleceń przeznaczonych do zarządzania kontami. Przekonasz się, jak serwer używa tabel uprawnień do pracy z klientami podczas nawiązywania przez nich połączeń lub wykonywania zapytań SQL.

Uwaga

W pewnych wersjach MySQL na pewno pojawiają się nowe tabele lub kolumny uprawnień, które w ten sposób zmieniają ich strukturę. W trakcie pierwszej instalacji MySQL w komputerze procedura instalacyjna tworzy tabele uprawnień o strukturze odpowiadającej aktualnie używanej wersji serwera. Jeżeli uaktualnisz MySQL do nowszej wersji, uruchom `mysql_upgrade` w celu aktualizacji tabel uprawnień i wprowadzenia wszelkich modyfikacji, które mogły pojawić się od chwili wydania aktualnie używanej przez Ciebie wersji serwera.

```
% mysql_upgrade --password=hasło_roota
```

Domyślnie `mysql_upgrade` nawiązuje połączenie z serwerem lokalnym jako użytkownik MySQL `root`, więc należy go wywołać w następujący sposób wraz z hasłem użytkownika `root`, a następnie ponownie uruchomić serwer:

13.2.1. Zarządzanie kontem MySQL na wysokim poziomie

Istnieją trzy polecenia pozwalające na przeprowadzanie wysokiego poziomu operacji na kontach MySQL:

- Zapytanie `CREATE USER` tworzy nowe konto użytkownika i opcjonalnie przypisuje mu hasło lub definiuje metodę uwierzytelnienia:

```
CREATE USER użytkownik [typ_uwierzytelnienia];
```

Zapytanie `CREATE USER` nie nadaje żadnych uprawnień, do tego służy zapytanie `GRANT`.

- Zapytanie `DROP USER` powoduje usunięcie istniejącego konta i wszystkich związanych z nim uprawnień:

```
DROP USER użytkownik;
```

Zapytanie `DROP USER` nie powoduje usunięcia bazy danych lub znajdujących się w niej obiektów, które były dostępne dla usuwanego konta.

- Zapytanie `RENAME USER` powoduje zmianę nazwy istniejącego konta:

```
RENAME USER konto_źródłowe TO konto_docelowe;
```

Wszystkie trzy zapytania możesz wykonywać, jeśli masz globalne uprawnienie `CREATE USER`. W przeciwnym razie musisz mieć uprawnienia `INSERT`, `DELETE` lub `UPDATE` do bazy danych, aby móc wykonać zapytanie odpowiednio `CREATE USER`, `DROP USER` lub `RENAME USER`.

W celu przeprowadzenia konfiguracji nowego konta rodzaj koniecznego do wykonania zapytania `CREATE USER` można określić po udzieleniu odpowiedzi na trzy poniższe pytania:

- Jaka ma być nazwa użytkownika?
- Z jakiego komputera lub komputerów użytkownik będzie mógł nawiązywać połączenie?
- W jaki sposób użytkownik będzie uwierzytelniany?

Odpowiedzi na dwa pierwsze pytania określają wartość parametru *użytkownik* używaną w zapytaniu. Nazwa konta składa się z nazwy użytkownika i komputera. Reguły rządzące składnią znajdziesz w podpunkcie 13.2.1.1, zatytułowanym „Określanie nazw kont użytkowników”.

Odpowiedź na trzecie pytanie zależy od tego, czy dla konta ma zostać zdefiniowane hasło dostępu, czy też serwer powinien stosować inną metodę uwierzytelniania użytkownika. Więcej informacji na temat uwierzytelniania znajdziesz w podpunkcie 13.2.1.3, zatytułowanym „Określanie sposobu uwierzytelniania użytkownika”.

13.2.1.1. Określanie nazw kont użytkowników

Wartość parametru *użytkownik* w zapytaniach zarządzania kontem, takich jak `CREATE USER`, składa się z nazwy użytkownika i komputera w postaci `'nazwa_użytkownika'@'nazwa_komputera'`. W serwerze MySQL podaje się nie tylko, jaki użytkownik nawiązuje połączenie, ale także skąd może się połączyć. W ten sposób można skonfigurować oddzielne konta dla dwóch użytkowników o takich samych nazwach, ale łączących się z różnych miejsc. MySQL potrafi ich rozróżniać i niezależnie zdefiniować im uprawnienia. Serwer przechowuje *nazwę_użytkownika* i *nazwę_komputera* w kolumnach `User` i `Host` rekordu tabeli `user` dla danego konta oraz w innych rekordach tabel uprawnień powiązanych z tym kontem.

Nazwa konta może być również podana jako `CURRENT_USER` lub `CURRENT_USER()`; będzie nim wówczas wykorzystywane w bieżącej sesji konto użytkownika.

Twoja nazwa użytkownika w MySQL to po prostu nazwa używana do identyfikacji podczas nawiązywania połączenia z serwerem. Ta nazwa nie ma żadnego powiązania z nazwą logowania w systemie UNIX lub Windows. W systemach UNIX, jeśli wyraźnie nie podasz nazwy, programy klienckie domyślnie używają nazwy logowania jako nazwy użytkownika MySQL, ale to tylko konwencja. Nie ma również nic specjalnego w nazwie `root` stosowanej jako nazwa superużytkownika w MySQL, który może zrobić dosłownie wszystko. W tabelach uprawnień nazwę `root` możesz zastąpić na przykład nazwą `superduper`, a następnie nawiązać połączenie jako użytkownik `superduper` i przeprowadzić operacje wymagające uprawnień superużytkownika.

Dzięki wybraniu odpowiedniej wartości parametru *użytkownik* można umożliwić użytkownikowi nawiązanie połączenia z określonego komputera lub ich zestawu. Jedno ze skrajnych rozwiązań to ograniczenie dostępu tylko do pojedynczego komputera, jeśli wiesz, że użytkownik będzie nawiązywał połączenie jedynie z tego konkretnego komputera:

```
CREATE USER 'boris'@'localhost' IDENTIFIED BY 'frost';
CREATE USER 'fred'@'ares.mars.net' IDENTIFIED BY 'steam';
```

Pamiętaj, że człon *nazwa_komputera* oznacza komputer, z którego użytkownik będzie nawiązywał połączenie. To nie jest komputer serwera, z którym klient będzie nawiązywał połączenie (o ile to nie będzie ten sam komputer).

Umożliwienie użytkownikowi nawiązania połączenia jedynie z pojedynczego komputera to jedna z najbardziej restrykcyjnych form dostępu, jakie możesz zastosować. Kolejny przypadek skrajny może dotyczyć użytkownika, który często podróżuje i musi mieć

możliwość nawiązywania połączenia z komputerów rozsianych po całym świecie. Jeżeli nazwa użytkownika to `max`, możesz mu pozwolić na nawiązywanie połączenia z dowolnego komputera:

```
CREATE USER 'max'@'%' IDENTIFIED BY 'mist';
```

Znak `%` działa w charakterze znaku wieloznacznego i ma takie samo znaczenie jak w dopasowaniu wzorca za pomocą klauzuli `LIKE`. Dlatego też nazwa komputera `%` oznacza dowolny komputer. To jest najłatwiejszy sposób konfiguracji użytkownika, ale jednocześnie zapewniający najmniejsze bezpieczeństwo. (Użycie znaku `%` może również skutkować pewnymi problemami z powodów przedstawionych w punkcie 13.4.4, zatytułowanym „Puzzle uprawnień”).

Inny znak wieloznaczny stosowany w klauzulach `LIKE` to podkreślenie. Tego znaku również można używać w nazwach komputera; powoduje dopasowanie dowolnego pojedynczego znaku. Aby wskazać dosłowny znak `%` lub `_`, należy poprzedzić go ukośnikiem.

Pomijając przypadki skrajne, rozwiązaniem pośrednim jest umożliwienie użytkownikowi nawiązania połączenia z ograniczonego zestawu komputerów. Na przykład, aby pozwolić użytkownikowi `mary` na nawiązanie połączenia z dowolnego komputera w domenie `example.com`, należy człon *nazwa_komputera* zdefiniować jako `%example.com`:

```
CREATE USER 'mary'@'%.example.com' IDENTIFIED BY 'fog';
```

Jeżeli chcesz, członem *nazwa_komputera* parametru *użytkownik* może być adres IPv4 lub IPv6 zamiast typowej nazwy. W przypadku adresu IPv4 można podać dosłowny adres IP, adres zawierający znaki dopasowania wzorca lub adres IP wraz z maską sieciową wskazujący bity używane dla numerów sieciowych:

```
CREATE USER 'joe'@'192.168.128.3' IDENTIFIED BY 'water';  
CREATE USER 'ardis'@'192.168.128.%' IDENTIFIED BY 'snow';  
CREATE USER 'rex'@'192.168.128.0/255.255.255.0' IDENTIFIED BY 'ice';
```

Pierwsze z powyższych zapytań wskazuje pojedynczy adres (`192.168.128.3`), z którego użytkownik może nawiązać połączenie. Zapytanie drugie zawiera wzorzec IP wskazujący podsieć klasy C: `192.168.128`. Z kolei w trzecim zapytaniu adres IP (`192.168.128.3/255.255.255.0`) wskazuje maskę sieciową z włączonymi pierwszymi 24 bitami. Powoduje dopasowanie dowolnego komputera z `192.168.128` w pierwszych 24 bitach adresu IP. Wartości maski sieciowej muszą być następujące: `255.0.0.0`, `255.255.0.0`, `255.255.255.0` lub `255.255.255.255`.

Podobne reguły mają zastosowanie dla adresów IPv6 za wyjątkiem faktu, że zapis z użyciem maski sieciowej jest nieobsługiwany.

Użycie `localhost` jako wartości członu *nazwa_komputera* w parametrze *użytkownik* pozwala użytkownikowi na nawiązanie połączenia z serwerem z poziomu komputera lokalnego na kilka sposobów:

- W systemach UNIX użytkownik może nawiązać połączenie, podając wartość `localhost`, `127.0.0.1` lub `:::1`. Połączenie `localhost` odbywa się za pomocą pliku gniazda systemu UNIX. Adres `127.0.0.1` lub `:::1` powoduje, że połączenie TCP/IP jest nawiązywane za pomocą interfejsu `loopback` IPv4 lub IPv6.

- W systemach Windows użytkownik może nawiązać połączenie, podając wartość `localhost`, `127.0.0.1` lub `:::1`. Wymienione połączenia są nawiązywane za pomocą TCP/IP z następującym wyjątkiem: jeśli serwer obsługuje połączenia pamięci współdzielonej, wówczas użycie `localhost` domyślnie powoduje nawiązanie połączenia za pomocą pamięci współdzielonej. Jeżeli serwer obsługuje połączenia za pomocą nazwanego potoku, nawiązanie takiego połączenia odbywa się po podaniu kropki jako nazwy komputera.

Jeżeli nazwa użytkownika lub komputera w parametrze *użytkownik* może być używana jako niecytowany identyfikator, wtedy nie trzeba jej cytować. W przypadku, gdy zawiera jakiegokolwiek znaki specjalne, takie jak `-` lub `%`, konieczne jest jej cytowanie. Na przykład, w nazwie `boris@localhost` oba człony są prawidłowe bez znaków cytowania. Jednak używanie znaków cytowania w każdym przypadku jest bezpieczne i takie rozwiązanie jest stosowane w tej książce. Nazwy użytkownika i komputera mogą być cytowane za pomocą znaków cytowania ciągu tekstowego lub znaków cytowania identyfikatorów. Człony nazwy użytkownika i nazwy komputera muszą być cytowane oddzielnie: użyj `'boris'@'localhost'`, a nie `'boris@localhost'`.

Jeżeli w parametrze *użytkownik* nie zostanie podana nazwa komputera, skutek jest taki sam jak w przypadku użycia znaku wieloznacznego `%`. Dlatego też `'max'` i `'max'@'%'` są dokładnie takimi samymi wartościami parametru *użytkownik*. Oznacza to, że jeżeli zamierzasz utworzyć konto użytkownika `'boris'@'localhost'`, ale pomyłkowo podasz `'boris@localhost'`, MySQL uzna je za prawidłowe. Jednak MySQL zinterpretuje `'boris@localhost'` jedynie jako część określającą nazwę użytkownika i dołączy do niej człon nazwy komputera (`%`), co spowoduje powstanie nazwy konta `'boris@localhost'@'%'`. Aby tego uniknąć, pamiętaj o oddzielnym cytowaniu członów nazwy użytkownika i nazwy komputera.

13.2.1.2. Dopasowanie nazwy komputera w nazwie konta do DNS

Bardzo często występują problemy podczas nawiązywania połączenia z komputerem serwera, jeśli używana jest inna nazwa komputera serwera niż `localhost`. Problemy wynikają z rozbieżności między nazwą komputera podaną w tabelach uprawnień i sposobem, w jaki DNS zgłasza nazwę komputera programom. Przyjmujemy założenie, że w pełni kwalifikowana nazwa komputera serwera to `cobra.example.com`. Jeżeli DNS podaje nie w pełni kwalifikowaną nazwę, taką jak `cobra`, a tabele uprawnień zawierają w pełni kwalifikowaną nazwę (lub na odwrót), wtedy mamy do czynienia z rozbieżnością nazw.

Aby ustalić, czy taka sytuacja występuje w używanym systemie, należy spróbować nawiązać połączenie z serwerem lokalnym za pomocą opcji `-h` i podać nazwę komputera:

```
% mysql -h cobra.example.com
```

Następnie zajrzyj do ogólnego dziennika zdarzeń serwera. W jaki sposób serwer zapisuje nazwę komputera w komunikatach dotyczących próby nawiązania połączenia? Czy nazwa komputera jest podawana w postaci w pełni kwalifikowanej, czy nie? Dzięki

informacjom, które znajdziesz w dzienniku zdarzeń, będziesz wiedział, jaką trzeba podać nazwę komputera podczas jej używania w nazwach kont.

Podobne problemy związane z dopasowaniem nazw mogą występować w przypadku nazw kont zawierających zwrócony przez DNS adres IP komputera, ale w innym formacie. Na przykład, jeżeli DNS zwraca adres 192.168.10.14 dla danego komputera, to będzie on dopasowany do członu *nazwa_komputera* w parametrze *uzytkownik* zdefiniowanego w postaci 192.168.10.14 lub 192.168.10.%, ale nie jako 192.168.010.14 lub 192.168.010.%. Jeżeli masz jakiegokolwiek wątpliwości, przeprowadź kilka operacji wyszukiwania DNS i przekonaj się, jak wyglądają wartości zwracane przez DNS, a następnie użyj tego samego formatu w nazwach kont.

13.2.1.3. Określanie sposobu uwierzytelniania użytkownika

Składnia zapytania `CREATE USER` pozwala na zdefiniowanie sposobu uwierzytelniania konta dzięki podaniu opcjonalnej klauzuli *typ_uwierzytelnienia*:

```
CREATE USER uzytkownik [typ_uwierzytelnienia]
```

Klauzula *typ_uwierzytelnienia* jest również częścią składni zapytania `GRANT`; zapoznaj się z punktem 13.2.2, zatytułowanym „Nadawanie uprawnień”.

Istnieją dwie formy, które może przybrać klauzula *typ_uwierzytelnienia*:

- W celu zdefiniowania, że użytkownik jest uwierzytelniany za pomocą składni, należy użyć poniższej składni:

```
IDENTIFIED BY [PASSWORD] 'hasło'
```

Na przykład:

```
CREATE USER 'pradeep'@'localhost' IDENTIFIED BY 'aurum';
```

W takim przypadku serwer MySQL samodzielnie zarządza hasłami i przechowuje je w kolumnie `Password` rekordu tabeli `user` zawierającego dane użytkownika, używając formatu hash identycznego do wygenerowanego przez funkcję `PASSWORD()`. Normalnie pomijasz słowo kluczowe `PASSWORD` i podajesz hasło w postaci zwykłego tekstu. Przed jego umieszczeniem w bazie danych MySQL konwertuje je na postać wartości hash. Jeżeli chcesz podać hasło już znajdujące się w postaci wartości hash, poprzedź tę wartość słowem kluczowym `PASSWORD`. Taka sytuacja może się zdarzyć podczas używania danych wyjściowych zapytania `SHOW GRANTS` w celu ponownego utworzenia konta. (Zapytanie `SHOW GRANTS` wyświetla hasła jako wartości hash, a nie w postaci zwykłego tekstu).

- Jeżeli chcesz wskazać inną metodę uwierzytelnienia, użyj poniższej składni i podaj nazwę wtyczki uwierzytelniania implementującej żadaną metodę oraz opcjonalny ciąg tekstowy zawierający dodatkowe informacje dla wtyczki:

```
IDENTIFIED WITH wtyczka_uwierzytelnienia [AS 'ciag_tekstowy_uwierzytelnienia']
```

Na przykład:

```
CREATE USER 'felipe'@'localhost' IDENTIFIED BY 'myplugin';
```

W takim przypadku serwer MySQL nie zarządza bezpośrednio hasłami, ale podczas nawiązywania połączenia oczekuje otrzymania od wskazanej wtyczki informacji, czy uwierzytelnienie zakończyło się powodzeniem. Więcej informacji na ten temat znajdziesz w punkcie 13.2.7, zatytułowanym „Wtyczki metod uwierzytelniania i użytkownicy proxy”. Na chwilę obecną podawanie nazwy wtyczki jest rzadko stosowane, ponieważ wtyczki uwierzytelnienia wprowadzono stosunkowo niedawno (w MySQL 5.5.7). Wraz z upływem czasu wtyczki uwierzytelnienia powinny zyskiwać coraz większą popularność, ponieważ pozwalają na implementację metod autoryzacji używających uwierzytelnienia innego typu niż hasła, którymi wewnętrznie zarządza sam serwer MySQL. Na przykład, wtyczka uwierzytelnienia może uzyskiwać dostęp do haseł systemu operacyjnego lub stanowić interfejs dla zewnętrznego serwera uwierzytelnienia implementujący mechanizm pojedynczego logowania.

Po utworzeniu konta przez serwer klient może wykorzystać je w celu nawiązania połączenia z serwerem bez konieczności uwierzytelniania się, o ile nie podano klauzuli IDENTIFIED BY wraz z niepustym hasłem lub ogólnie klauzuli IDENTIFIED BY. Takie rozwiązanie jest jednak niebezpieczne i należy go unikać.

13.2.2. Nadawanie uprawnień

Aby nadać uprawnienia użytkownikowi konta, należy użyć zapytania GRANT, które ma następującą postać:

```
GRANT uprawnienia [(kolumny)]
ON poziom_uprawnień
TO użytkownik [typ_uwierzytelnienia]
[REQUIRE wymagane szyfrowanie]
[WITH nadawanie uprawnień lub opcje zarządzania zasobami];
```

Jeżeli wymienione w zapytaniu konto użytkownika istnieje, to GRANT modyfikuje jego uprawnienia. W przypadku, gdy konto użytkownika nie istnieje, zapytanie GRANT utworzy je wraz ze wskazanymi uprawnieniami. Aby uniknąć możliwości utworzenia przez zapytanie GRANT nowego konta bez zdefiniowanego uwierzytelnienia (co jest niebezpieczne), należy ustawić zmienną systemową `sql_mode` i włączyć tryb SQL o nazwie `NO_AUTO_CREATE_USER`. W ten sposób zapytanie GRANT nie utworzy konta, jeśli nie zostaną podane informacje o uwierzytelnieniu.

Kilka klauzul jest opcjonalnych i w ogóle nie muszą być podawane. Ogólnie rzecz biorąc, najczęściej podawane są następujące człony zapytania:

- *uprawnienia*: to po prostu uprawnienia nadawane użytkownikowi konta. Na przykład, uprawnienia SELECT pozwalają na wykonywanie zapytań SELECT, natomiast uprawnienia SHUTDOWN pozwalają użytkownikowi na zamknięcie serwera. Istnieje możliwość podania wielu uprawnień, wystarczy je rozdzielić przecinkami.

- *kolumny*: to są nazwy kolumn, dla których stosowane będą uprawnienia. Kolumny trzeba rozdzielić przecinkami i wymienić je w nawiasie. To jest opcjonalne i stosowane jedynie w przypadku uprawnień nadawanych wybranym kolumnom. Lista kolumn musi znajdować się po nazwie *każdego* uprawnienia.
- *poziom uprawnienie*: to jest poziom nadanych uprawnień. Największym poziomem jest globalny, wówczas uprawnienia dotyczą wszystkich tabel we wszystkich bazach danych. Uprawnienia globalne można traktować jako uprawnienia superużytkownika. Oczywiście, istnieje możliwość nadawania uprawnień dla konkretnej bazy danych, tabeli, kolumny (w przypadku użycia klauzuli *kolumn*) i procedury.
- *użytkownik*: wskazuje konto użytkownika otrzymujące uprawnienia. Format konta to `'nazwa_użytkownika'@'nazwa_komputera'` i jest zgodny z opisem przedstawionym w podpunkcie 13.2.1.1, zatytułowanym „Określanie nazw kont użytkowników”.

Klauzule *typ_uwierzytelnienia*, *REQUIRE* i *WITH* są opcjonalne:

- Klauzula *typ_uwierzytelnienia* powoduje zdefiniowanie metody uwierzytelniania lub hasła dla konta. To jest opcjonalne i niepotrzebne, jeśli informacje o typie uwierzytelnienia zostały już podane dla konta (na przykład w zapytaniu *CREATE USER*). W przypadku podania omawianej klauzuli ma ona taką samą składnię jak w zapytaniu *CREATE USER* (patrz podpunkt 13.2.1.3, zatytułowany „Określanie sposobu uwierzytelniania użytkownika”). Podanie hasła dla istniejącego konta powoduje zastąpienie dotychczasowego hasła dla danego konta.
- Klauzula *REQUIRE* pozwala na konfigurację konta, które musi nawiązywać połączenie za pomocą protokołu *SSL*.
- Klauzula *WITH* powoduje nadanie uprawnienia *GRANT OPTION*, pozwalającego kontu na nadawanie własnych uprawnień innym użytkownikom. Klauzula *WITH* jest używana także do wskazania opcji zarządzania zasobami, co pozwala na nakładanie ograniczeń w zakresie liczby połączeń lub zapytań, jakie mogą być wykonane przez dane konto użytkownika w ciągu godziny. Dzięki tym opcjom można uniemożliwić kontu zbyt duże obciążanie serwera.

W celu wskazania możliwości konta dokładną postać zapytania *GRANT* można ustalić, odpowiadając sobie na poniższe zapytania:

- Jaki rodzaj dostępu powinien zostać udzielony kontu? To znaczy, jaki rodzaj uprawnień powinien mieć użytkownik i do czego powinny mieć zastosowanie?
- Czy wymagane są bezpieczne połączenia (*SSL*)?
- Czy użytkownik powinien mieć nadane uprawnienia administracyjne?
- Czy należy ograniczyć zasoby dostępne dla użytkownika?

W kolejnych podpunktach dowiesz się, jak odpowiedzieć na powyższe pytania. Poznasz także przykłady ilustrujące użycie wymienionych wcześniej klauzul w zapytaniach *GRANT*.

13.2.2.1. Definiowanie uprawnień użytkownika

Istnieje wiele uprawnień, które można nadać użytkownikowi konta. Uprawnienia można zgrupować w dwóch pokrótce opisanych kategoriach: administracyjne i dotyczące obiektów. Ponadto, istnieją dwa specjalne specyfikatory uprawnień. Pierwszy, o nazwie ALL (lub ALL PRIVILEGES), oznacza „wszystkie uprawnienia” (za wyjątkiem GRANT OPTION). Drugi, o nazwie USAGE, oznacza „brak uprawnień”. Jest stosowany do zmiany danych konta innych niż jego uprawnienia, na przykład ograniczeń dotyczących zużycia zasobów lub określenia, czy wymagane jest użycie protokołu SSL. Uzupełnienie przedstawionych tutaj opisów znajdziesz w podrozdziale 13.3, zatytułowanym „Struktura i zawartość tabel uprawnień”, w którym omówiono uprawnienia w kategoriach ich powiązań z tabelami uprawnień.

Poniżej przedstawiono uprawnienia dotyczące operacji administracyjnych nadzorujących działanie serwera oraz możliwości użytkownika w zakresie nadawania uprawnień. Te uprawnienia z reguły są nadawane bardzo oszczędnie, ponieważ pozwalają użytkownikowi wpływać na działanie serwera. Na przykład, uprawnienie SHUTDOWN nie jest potrzebne do wykonywania codziennych zadań.

■ CREATE USER

Pozwala na wykonywanie zapytań CREATE USER, DROP USER, RENAME USER i REVOKE ALL PRIVILEGES.

■ FILE

Pozwala serwerowi na odczytywanie lub zapisywanie plików w komputerze serwera. Aby uniemożliwić stosowanie tego uprawnienia bez żadnych ograniczeń, serwer podejmuje pewne kroki:

- ◆ Dostęp można uzyskać jedynie dla plików dostępnych dla wszystkich, czyli nieuznawanych za chronione w jakikolwiek sposób.
- ◆ Plik, który ma zostać zapisany, nie może jeszcze istnieć. W ten sposób serwer uniemożliwia nadpisanie ważnych plików, na przykład */etc/passwd* lub plików bazy danych należących do innego użytkownika. (Gdyby to ograniczenie nie istniało, wtedy mógłbyś zupełnie zastąpić zawartość na przykład tabel uprawnień w bazie danych mysql).

Pomimo powyższych kroków podejmowanych przez serwer uprawnienia FILE nie należy nadawać bezkrytycznie, ponieważ — jak się przekonasz w punkcie 13.2.6, zatytułowanym „Unikanie ryzyka związanego z kontrolą dostępu” — to może być wyjątkowo niebezpieczne. Jeżeli nadajesz uprawnienie FILE, upewnij się, że serwer nie jest uruchamiany przez użytkownika root w systemie UNIX, ponieważ root ma możliwość tworzenia nowych plików w dowolnym miejscu systemu plików. Uruchomienie serwera z poziomu zwykłego konta gwarantuje, że pliki będą mogły być tworzone jedynie w katalogach dostępnych dla tego konta. Zapoznaj się z podpunktem 12.2.1.1, zatytułowanym „Działanie serwera w ramach pozbawionego uprawnień konta logowania”.

■ GRANT OPTION

Pozwala na nadawanie innym użytkownikom uprawnień posiadanych przez dane konto użytkownika, łącznie z uprawnieniem GRANT OPTION.

■ PROCESS

Serwer MySQL jest wielowątkowy i pozwala na jednoczesną obsługę połączeń wielu klientów. Wspomniane wątki można potraktować jako procesy działające w serwerze. Upewnienie PROCESS daje możliwość wykonania zapytania SHOW PROCESSLIST lub polecenia `mysqladmin processlist` w celu wyświetlenia informacji o aktualnie wykonywanych zadaniach. To upewnienie daje także możliwość wyświetlenia wszystkich zadań, nawet powiązanych z innymi użytkownikami. Bez upewnienia PROCESS zawsze można wyświetlić tylko własne zadania.

■ PROXY

Pozwala na zdobycie uprawnień innego użytkownika. Innymi słowy, można działać w charakterze proxy dla innego użytkownika i wykonać operacje, które są dozwolone dla niego.

■ RELOAD

Pozwala na przeprowadzanie operacji administracyjnych serwera. Dzięki upewnieniu RELOAD zyskujesz możliwość wykonywania zapytań takich jak FLUSH i RESET. Ponadto, będziesz mógł wykonać następujące polecenia narzędzia `mysqladmin`: `reload`, `refresh`, `flush-hosts`, `flush-logs`, `flush-privileges`, `flush-status`, `flush-tables` i `flush-threads`.

■ REPLICATION CLIENT

Pozwala na sprawdzenie położenia oraz stanu serwerów głównego i podległych replikacji za pomocą zapytań SHOW MASTER STATUS i SHOW SLAVE STATUS.

■ REPLICATION SLAVE

Pozwala klientowi na nawiązanie połączenia z serwerem głównym i żądanie uaktualnień serwera podległego, a także wykonywanie zapytań SHOW SLAVE HOSTS i SHOW BINLOG EVENTS. To upewnienie musi być nadane kontom użytkowników serwera podległego używanym do nawiązywania połączenia z serwerem głównym.

■ SHOW DATABASES

Pozwala na wyświetlenie nazw wszystkich baz danych za pomocą zapytania SHOW DATABASES. W przypadku braku tego upewnienia możesz zobaczyć nazwę bazy danych tylko wtedy, gdy masz do niej upewnienia. Jednak tę możliwość zyskujesz po otrzymaniu *jakiegokolwiek* globalnego upewnienia do baz danych, na przykład CREATE TEMPORARY TABLES lub LOCK TABLES, które najczęściej są udzielane globalnie. Aby zezwolić na wykonywanie zapytania SHOW DATABASES jedynie użytkownikom posiadającym upewnienie SHOW DATABASES, serwer należy uruchomić wraz z opcją `--skip-show-database`.

- SHUTDOWN

Pozwala na zamknięcie serwera, na przykład za pomocą polecenia `mysqladmin shutdown`.

- SUPER

Pozwala na zakończenie procesu serwera za pomocą zapytania `KILL` lub polecenia `mysqladmin kill`. To uprawnienie daje możliwość zakończenia dowolnego procesu, nawet powiązanego z innymi użytkownikami. Bez uprawnienia `SUPER` zawsze możesz kończyć własne procesy.

Inne zapytania udostępniane przez to uprawnienie to zapytania `SET`, pozwalające na modyfikację globalnych zmiennych systemowych i globalnych cech charakterystycznych transakcji: `CHANGE MASTER`, `PURGE BINARY LOGS`, `SHOW MASTER STATUS`, `SHOW SLAVE STATUS`, `START SLAVE` i `STOP SLAVE`. Uprawnienie `SUPER` pozwala również na wskazanie dowolnego konta w klauzuli `DEFINER` zapytań definiujących widoki lub programy składowane, a także na przeprowadzanie szyfrowania DES za pomocą funkcji `DES_DECRYPT()` na podstawie kluczy przechowywanych w pliku klucza DES.

Uprawnienie `SUPER` pozwala na użycie polecenia `mysqladmin debug` i nadpisuje wszelkie ustawienia `max_connections` podczas nawiązywania połączenia z serwerem. Dlatego też uzyskujesz dostęp do slotu połączenia zarezerwowanego przez serwer dla połączeń administracyjnych, nawet jeśli wszystkie zwykłe sloty są zajęte.

Poniższe uprawnienia mają zastosowanie względem operacji przeprowadzanych na obiektach, takich jak bazy danych, tabele i programy składowane. Kontrolują dostęp do danych zarządzanych przez serwer.

- ALTER

Pozwala na wykonanie zapytania `ALTER TABLE`, choć w zależności od operacji, które chcesz przeprowadzić na tabeli, mogą być wymagane także inne uprawnienia.

- ALTER ROUTINE

Pozwala na zmianę lub usunięcie składowanych funkcji i procedur.

- CREATE

Pozwala na tworzenie baz danych i tabel. To uprawnienie nie pozwala na tworzenie indeksów w tabeli, za wyjątkiem początkowo zdefiniowanych w zapytaniu `CREATE TABLE`.

- CREATE ROUTINE

Pozwala na tworzenie składowanych funkcji i procedur.

- CREATE TABLESPACE

Pozwala na tworzenie, usuwanie i zmianę przestrzeni tabel.

- CREATE TEMPORARY TABLES

Pozwala na tworzenie tabel tymczasowych za pomocą zapytania `CREATE TEMPORARY TABLE`.

- **CREATE VIEW**
Pozwala na tworzenie widoków.
- **DELETE**
Pozwala na usuwanie rekordów z tabel.
- **DROP**
Pozwala na usuwanie baz danych i tabel. To uprawnienie nie pozwala na usuwanie indeksów.
- **EVENT**
Pozwala na pracę z harmonogramem zdarzeń.
- **EXECUTE**
Pozwala na wykonywanie składowanych funkcji i procedur.
- **INDEX**
Pozwala na tworzenie i usuwanie indeksów w tabelach, przypisywanie indeksów buforom kluczy i wczytywanie indeksów do buforów kluczy.
- **INSERT**
Pozwala na wstawianie rekordów do tabel.
- **LOCK TABLES**
Pozwala na nakładanie blokad na tabele za pomocą zapytań `LOCK TABLES`. To uprawnienie ma zastosowanie jedynie względem tabel, do których masz uprawnienie `SELECT`, ale pozwala jedynie na nakładanie blokad odczytu i zapisu, a nie wyłącznie odczytu. Uprawnienie nie ma zastosowania względem blokad nakładanych przez serwer w Twoim imieniu w trakcie przetwarzania zapytania. Tego rodzaju blokady są automatycznie nakładane i zwalniane niezależnie od uprawnienia `LOCK TABLES`.
- **REFERENCES**
To uprawnienie jest nieużywane.
- **SELECT**
Pozwala na pobieranie danych z tabel za pomocą zapytań `SELECT`. To uprawnienie jest niepotrzebne dla zapytań takich jak `SELECT NOW()` lub `SELECT 4/2`, które jedynie obliczają wyrażenie i nie korzystają z tabel.
- **SHOW VIEW**
Pozwala na wykonywanie zapytań `SHOW CREATE VIEW` w celu wyświetlania definicji widoku.
- **TRIGGER**
Pozwala na dodawanie i usuwanie wyzwalaczy.
- **UPDATE**
Pozwala na modyfikację rekordów w tabelach.

Pewne operacje wymagają połączenia uprawnień. Na przykład, zapytanie REPLACE może powodować wykonanie zapytania DELETE i INSERT, więc wymaga także uprawnień DELETE i INSERT. Aby nadać uprawnienie, trzeba je samemu posiadać, a także mieć uprawnienie GRANT OPTION.

Uprawnienie można nadawać na różnych poziomach, od globalnego do bardzo szczegółowego. Jest to kontrolowane przez wymienione w tabeli 13.2 specyfikatory klauzuli ON. W przypadku specyfikatorów na poziomie tabeli można podać klauzulę (kolumny) i nazwę uprawnienia, co spowoduje jego nadanie na poziomie kolumny. Składnia zostanie przedstawiona nieco dalej w rozdziale.

Tabela 13.2. Specyfikatory poziomu uprawnień

Uprawnienia	Poziom, na którym zastosowanie mają dane uprawnienia
ON *.*	Uprawnienia globalne: wszystkie bazy danych i wszystkie obiekty w bazach danych.
ON *	Uprawnienia na poziomie bazy danych dla domyślnej bazy danych. W przypadku braku domyślnej bazy danych zgłaszany jest błąd.
ON nazwa_bazy_danych.*	Uprawnienia bazy danych. Wszystkie obiekty we wskazanej bazie danych.
ON nazwa_bazy_danych.nazwa_tabeli	Uprawnienia tabeli. Wszystkie kolumny we wskazanej tabeli.
ON nazwa_tabeli	Uprawnienia tabeli. Wszystkie kolumny we wskazanej tabeli w domyślnej bazie danych.
ON nazwa_bazy_danych.nazwa_procedury	Uprawnienia dla wskazanej procedury w wymienionej bazie danych.
ON użytkownik	Uprawnienia proxy: <i>użytkownik</i> to nazwa użytkownika proxy.

Aby w przypadku jakichkolwiek niejasności wyraźnie podać rodzaj obiektu, dla którego mają zastosowanie uprawnienia, można użyć słowa kluczowego TABLE, FUNCTION lub PROCEDURE (na przykład ON TABLE mydb.mytbl lub ON FUNCTION mydb.myfunc).

Uprawnienie USAGE jest nadawane jedynie na poziomie globalnym (to znaczy w klauzuli ON *.*).

Specyfikator ALL (lub ALL PRIVILEGES) nadaje wszystkie uprawnienia dostępne na danym poziomie. Na przykład, na poziomie globalnym ALL nadaje wszystkie uprawnienia. Z kolei na poziomie tabeli nadawane są jedynie uprawnienia stosowane względem tabel. Specyfikator ALL może być używany jedynie do nadawania uprawnień globalnych, dla bazy danych, tabeli lub procedury. W przypadku uprawnień kolumn konieczne jest podanie każdego uprawnienia, które ma zostać nadane. Pojęcie „wszystkie uprawnienia” tak naprawdę oznacza „wszystkie uprawnienia poza jednym” — uprawnienie GRANT OPTION jest wymagane dla operacji GRANT i REVOKE.

Uprawnienia globalne mają największe możliwości, ponieważ są stosowane względem wszystkich baz danych. Aby utworzyć konto superużytkownika, który będzie mógł zrobić wszystko, łącznie z nadawaniem uprawnień innym użytkownikom, należy wykonać poniższe zapytania:

```
CREATE USER 'ethel'@'localhost' IDENTIFIED BY 'coffee';  
GRANT ALL ON *.* TO 'ethel'@'localhost' WITH GRANT OPTION;
```

Klauzula `ON *.*` oznacza „wszystkie bazy danych wraz z wszystkimi zawartymi w nich obiektami”. Jako pewien rodzaj zabezpieczenia, konto użytkownika utworzone w powyższym przykładzie może nawiązać połączenie jedynie z komputera lokalnego. Ograniczenie komputerów, z których połączenie może nawiązać superużytkownik, jest dobrym rozwiązaniem, ponieważ ogranicza liczbę komputerów, gdzie można zainstalować programy służące do łamania haseł.

Uprawnienia administracyjne z natury są globalne i dlatego, za wyjątkiem `GRANT OPTION` i `PROXY`, mogą być nadane jedynie za pomocą specyfikatora `ON *.*`. Na przykład, uprawnienie `RELOAD` włącza użycie `FLUSH`, a więc poniższe zapytania powodują utworzenie użytkownika o nazwie `flush`, który może jedynie wykonywać zapytania `FLUSH`:

```
CREATE USER 'flush'@'localhost' IDENTIFIED BY 'flushpass';  
GRANT RELOAD ON *.* TO 'flush'@'localhost';
```

Ten rodzaj konta MySQL jest użyteczny podczas tworzenia skryptów administracyjnych wykonujących operacje takie jak czyszczenie dzienników zdarzeń (patrz punkt 12.8.7, zatytułowany „Zarządzanie dziennikami zdarzeń”).

Uprawnienia na poziomie bazy danych mają zastosowanie względem wskazanej bazy danych i wszystkich znajdujących się w niej obiektów. Aby nadać uprawnienia na tym poziomie, należy użyć klauzuli `ON nazwa_bazy_danych.*`:

```
CREATE USER 'bill'@'mamba.example.com' IDENTIFIED BY 'rock';  
GRANT ALL ON sampdb.* TO 'bill'@'mamba.example.com';
```

```
CREATE USER 'reader'@'%' IDENTIFIED BY 'dirt';  
GRANT SELECT ON menagerie.* TO 'reader'@'%';
```

Pierwszy zestaw zapytań nadaje użytkownikowi `bill` pełne uprawnienia do wszystkich tabel w bazie danych `sampdb`, gdy nawiąże połączenie z komputera `mamba.example.com`. Drugi zestaw zapytań tworzy użytkownika `reader` o znacznie mniejszych uprawnieniach. Może on nawiązać połączenie z dowolnego komputera, ma dostęp jedynie do bazy danych `menagerie`, ale może wykonywać tylko zapytania `SELECT`. Oznacza to, że użytkownik `reader` ma możliwość jedynie odczytu danych.

Istnieje możliwość podania rozdzielonej przecinkami listy uprawnień do nadania. Na przykład, aby użytkownikowi nadać uprawnienia odczytu i modyfikacji zawartości tabel w bazie danych `sampdb`, ale nie tworzenia nowych lub usuwania istniejących, nie należy nadawać uprawnień `ALL` dla bazy danych. Zamiast tego trzeba wymienić wszystkie nadawane uprawnienia:

```
CREATE USER 'jennie'@'%' IDENTIFIED BY 'boron';  
GRANT SELECT,INSERT,DELETE,UPDATE ON sampdb.* TO 'jennie'@'%';
```

W celu zachowania większej kontroli dostępu poniżej poziomu bazy danych, uprawnienia można nadawać poszczególnym tabelom lub nawet kolumnom. Uprawnienia charakterystyczne dla kolumn są użyteczne, gdy pewne fragmenty tabeli chcesz ukryć przed użytkownikiem lub jeśli użytkownik powinien mieć możliwość modyfikacji jedynie określonych kolumn. Przyjmujemy założenie, że masz wolontariusza pomagającego w biurze Ligi Historycznej. Podejmujesz decyzję o nadaniu nowemu pomocnikowi uprawnień pozwalających jedynie na odczyt zawartości tabeli `member` zawierającej informacje o członkostwie oraz uprawnienia `UPDATE` dla kolumn `expiration` i powiązanych z adresem członka Ligi. W ten sposób pomocnik ma bardzo ograniczone uprawnienia zapisu potrzebne do uaktualniania adresu lub informacji o dacie wygaśnięcia członkostwa. Zapytania wymagane do utworzenia w MySQL omówionego powyżej konta przedstawiają się następująco:

```
CREATE USER 'assistant'@'localhost' IDENTIFIED BY 'officehelp';
GRANT SELECT, UPDATE (expiration,street,city,state,zip)
ON sampdb.member TO 'assistant'@'localhost';
```

Zapytanie `GRANT` daje uprawnienia dostępu do całej zawartości kolumny `member` (ponieważ po uprawnieniu `SELECT` nie podano listy kolumn) oraz uprawnienie do uaktualniania kolumn wymienionych w nawiasie po słowie kluczowym `UPDATE`.

W celu nadania uprawnień na poziomie kolumn dla wielu różnych uprawnień w zapytaniu `GRANT`, lista kolumn musi być podana po każdym uprawnieniu.

Aby umożliwić użytkownikowi konta wykonywanie programów składowanych (funkcje i procedury), należy nadać uprawnienia `ALTER ROUTINE` i `EXECUTE` na poziomie globalnym, bazy danych lub (za wyjątkiem `CREATE ROUTINE`) dla poszczególnych procedur:

```
CREATE USER 'wilbur'@'localhost' IDENTIFIED BY 'sulfur';
GRANT CREATE ROUTINE ON sampdb.* TO 'wilbur'@'localhost';
GRANT EXECUTE ON PROCEDURE sampdb.count_students TO 'wilbur'@'localhost';
```

Uprawnienie `PROXY` nie ma zastosowania względem konkretnej operacji. Zamiast tego pozwala użytkownikowi na działanie w charakterze proxy dla innego użytkownika. W takim przypadku użytkownik proxy będzie miał wszystkie uprawnienia tego drugiego użytkownika. Poniższe zapytanie umożliwia użytkownikowi `clint` posiadanie uprawnień użytkownika `bart`:

```
GRANT PROXY ON 'bart'@'localhost' TO 'clint'@'localhost';
```

Składnia nadania uprawnienia `PROXY` jest ograniczona: uprawnienie musi być nadane przez samego użytkownika, niedozwolone jest użycie klauzuli `REQUIRE`, natomiast klauzuli `WITH` można użyć jedynie z `GRANT OPTION`. Więcej informacji znajdziesz w punkcie 13.2.7, zatytułowanym „Wtyczki metod uwierzytelniania i użytkownicy proxy”.

Aby w zapytaniu `GRANT` zacytować nazwę bazy danych, tabeli, kolumny lub procedury, należy cytować je jak identyfikatory, a nie ciągi tekstowe, na przykład:

```
GRANT SELECT, UPDATE (`expiration`,`street`,`city`,`state`,`zip`)
ON `sampdb`.`member` TO 'assistant'@'localhost';
```

Rekordy w tabelach uprawnień nie „podążają” za operacjami zmiany nazw obiektów bazy danych. Na przykład, uprawnienia nadane nazwie danej tabeli lub kolumny nie są uaktualniane po zmianie nazwy tej tabeli lub kolumny.

13.2.2.2. Używanie uprawnienia USAGE

Uprawnienie specjalne o nazwie USAGE oznacza „brak uprawnień”. To może nie wydawać się zbyt użyteczne, ale naprawdę się przydaje. Pozwala na zmianę cech charakterystycznych konta innych niż uprawnienia, a same uprawnienia pozostają w dotychczasowej postaci. Aby uprawnienie USAGE nadać na poziomie globalnym, należy podać nazwę konta oraz jego nowe właściwości. Na przykład, w celu zmiany hasła, zdefiniowania wymogu nawiązywania połączenia za pomocą protokołu SSL lub nałożenia ograniczeń na konto użytkownika bez wpływu na jego dotychczasowe uprawnienia należy wykonać zapytania takie jak przedstawione poniżej:

```
GRANT USAGE ON *.* TO uzytkownik IDENTIFIED BY 'nowe_haslo';  
GRANT USAGE ON *.* TO uzytkownik REQUIRE SSL;  
GRANT USAGE ON *.* TO uzytkownik WITH MAX_CONNECTIONS_PER_HOUR 10;
```

13.2.2.3. Wymaganie nawiązania połączenia za pomocą protokołu SSL

MySQL pozwala klientom na nawiązywanie bezpiecznych połączeń za pomocą protokołu SSL (ang. *Secure Socket Layer*), który szyfruje strumień danych między klientem i serwerem i tym samym dane nie są przesyłane w postaci zwykłego tekstu. Ponadto, certyfikat X509 może być użyty w celu zapewnienia klientom dostarczania informacji uwierzytelniających przez połączenia SSL. Bezpieczne połączenie stanowi dodatkową warstwę zabezpieczenia, ale kosztem większej liczby cykli procesora wymaganych do przeprowadzania szyfrowania i deszyfrowania.

Aby zdefiniować wymóg stosowania bezpiecznego połączenia, trzeba użyć klauzuli REQUIRE. W celu zdefiniowania jedynie wymogu nawiązywania połączenia za pomocą protokołu SSL bez podawania konkretnego typu bezpiecznego połączenia, użytkownik powinien użyć klauzuli REQUIRE SSL:

```
CREATE USER 'eladio'@'%.example.com' IDENTIFIED BY 'flint';  
GRANT ALL ON sampdb.* TO 'eladio'@'%.example.com' REQUIRE SSL;
```

Aby dokładnie zdefiniować wymagania, na przykład przedstawienie przez klienta ważnego certyfikatu X509, użyj poniższego zapytania:

```
GRANT ALL ON sampdb.* TO 'eladio'@'%.example.com' REQUIRE X509;
```

Klauzula REQUIRE X509 nie nakłada żadnych ograniczeń na zawartość certyfikatu, musi on jedynie być ważny. Oczywiście, istnieje możliwość zdefiniowania, że certyfikat X509 dostarczany przez klienta ma posiadać pewne cechy charakterystyczne. Są one podawane za pomocą opcji ISSUER i SUBJECT w klauzuli REQUIRE. Opcje ISSUER i SUBJECT odnoszą się do wydawcy i przedmiotu certyfikatu. Na przykład, katalog *ssl* dystrybucji sampdb zawiera plik certyfikatu klienta o nazwie *client-cert.pem*, który możesz wykorzystać

do testowania połączeń SSL. Informacje dotyczące wydawcy i przedmiotu certyfikatu można wyświetlić za pomocą polecenia `openssl`:

```
% openssl x509 -issuer -subject -noout -in client-cert.pem
issuer= /C=US/ST=WI/L=Madison/O=sampdb/OU=CA/CN=sampdb
subject= /C=US/ST=WI/L=Madison/O=sampdb/OU=client/CN=sampdb
```

Poniższe zapytanie GRANT wskazuje konto użytkownika, dla którego klient musi dostarczyć certyfikat, wraz z odpowiednimi wartościami ISSUER i SUBJECT:

```
GRANT ALL ON sampdb.* TO 'eladio'@'%example.com'
  REQUIRE ISSUER '/C=US/ST=WI/L=Madison/O=sampdb/OU=CA/CN=sampdb'
  AND SUBJECT '/C=US/ST=WI/L=Madison/O=sampdb/OU=client/CN=sampdb';
```

Klauzuli REQUIRE można również użyć do zdefiniowania, że połączenie ma być szyfrowane za pomocą wskazanego rodzaju szyfrowania:

```
GRANT ALL ON sampdb.* TO 'eladio'@'%example.com'
  REQUIRE CIPHER 'DHE-RSA-AES256-SHA';
```

Aby wyraźnie zdefiniować brak wymogu stosowania szyfrowanych połączeń, użyj klauzuli REQUIRE NONE. To jest wartość domyślna podczas tworzenia nowego konta, ale może być również stosowana w celu usunięcia wymogu używania protokołu SSL w koncie, które ma zdefiniowany tego rodzaju wymóg.

Podczas używania klauzuli REQUIRE należy pamiętać o kilku kwestiach:

- Wykonanie zapytania GRANT ustawiającego konto wymóg używania bezpiecznego połączenia jedynie definiuje wspomniane wymaganie. W rzeczywistości nie dostarcza programowi klienta możliwości nawiązania bezpiecznego połączenia za pomocą tego konta. Konieczne jest skonfigurowanie serwera MySQL do obsługi SSL, a serwer i programy klienckie trzeba uruchamiać w odpowiedni sposób. Więcej informacji na ten temat znajdziesz w podrozdziale 13.5, zatytułowanym „Konfiguracja bezpiecznych połączeń za pomocą SSL”.
- Jeżeli w nowo tworzonym koncie zdefiniujesz wymóg użycia protokołu SSL, który będzie nieobsługiwany przez serwer lub programy klienckie, takie konto użytkownika praktycznie pozostaje nieużyteczne.
- Klauzula REQUIRE jest używana jedynie do wskazania, czy konto użytkownika *musi* nawiązywać bezpieczne połączenie z serwerem. Jeżeli serwer i programy klienckie są skonfigurowane wraz z obsługą SSL, każdy użytkownik nadal *może* stosować bezpieczne połączenie, nawet jeśli nie ma takiego wymagania.
- Stosowanie klauzuli REQUIRE dla kont, które nie nawiązują połączenia z serwerem z poziomu zewnętrznej sieci, nie ma sensu. Tego rodzaju połączeń nie można podsłuchiwać, a więc nic nie zyskujesz z ich zaszyfrowania. Do tego rodzaju połączeń zaliczamy nawiązywane jedynie przez plik gniazda systemu UNIX, nazwany potok, pamięć współdzieloną i interfejs loopback o adresie IP 127.0.0.1 lub ::1. Wymienione połączenia używają interfejsów obsługiwanych wewnętrznie przez komputer, a generowany przez nie ruch nie przechodzi do sieci zewnętrznej.

13.2.2.4. Nadanie użytkownikowi uprawnień administracyjnych

Aby umożliwić użytkownikowi konta nadawanie własnych uprawnień innym użytkownikom, należy użyć klauzuli `WITH GRANT OPTION`. Użycie wymienionej klauzuli wymaga uprawnienia `GRANT OPTION`.

Jednym z powodów nadawania kontu uprawnienia `GRANT OPTION` jest umożliwienie właścicielowi bazy danych kontrolowania dostępu do niej: właściciel powinien mieć wszystkie uprawnienia do bazy danych, łącznie z uprawnieniem `GRANT OPTION`. Na przykład, aby umożliwić użytkownikowi `alicia` nawiązanie połączenia z dowolnego komputera w domenie `big-corp.com` i nadanie mu uprawnień administracyjnych do wszystkich tabel w bazie danych `sales`, konto użytkownika należy utworzyć w następujący sposób:

```
CREATE USER 'alicia'@'%.big-corp.com' IDENTIFIED BY 'shale';
GRANT ALL ON sales.* TO 'alicia'@'%.big-corp.com' WITH GRANT OPTION;
```

Klauzula `WITH GRANT OPTION` pozwala tak utworzonemu użytkownikowi na nadawanie innym użytkownikom uprawnień dostępu do jego bazy danych. Pamiętaj, że dwóch użytkowników z uprawnieniami `GRANT OPTION` może nadawać sobie nawzajem własne uprawnienia. Jeżeli jeden użytkownik ma tylko uprawnienie `SELECT`, natomiast drugi `GRANT OPTION` i inne uprawnienia poza `SELECT`, wtedy drugi z wymienionych może „wzmocnić” pierwszego.

Innym sposobem nadania uprawnienia `GRANT OPTION` jest po prostu jego umieszczenie na początku zapytania `GRANT`:

```
GRANT GRANT OPTION ON sales.* TO 'alicia'@'%.big-corp.com';
```

Jednak zapytanie takie jak przedstawione poniżej nie działa:

```
GRANT ALL,GRANT OPTION ON sales.* TO 'alicia'@'%.big-corp.com';
```

W powyższym zapytaniu `GRANT` specyfikator `ALL` może być użyty jedynie przez użytkownika, a nie na liście innych specyfikatorów.

Upewnienie `GRANT OPTION` ma zastosowanie względem wszystkich uprawnień na nadawanym poziomie lub poniżej, ale nie dla poszczególnych uprawnień. Jeżeli konto użytkownika otrzyma uprawnienie `GRANT OPTION` na danym poziomie, wtedy użytkownik może nadać dowolne uprawnienie z tego poziomu. Nie można zdefiniować, że użytkownik ma możliwość udzielania swoich uprawnień tylko niektórym innym użytkownikom.

13.2.2.5. Ograniczanie zasobów dostępnych dla użytkownika

System uprawnień w MySQL pozwala na nałożenie pewnych ograniczeń na użytkownika, na przykład określenie maksymalnej liczby połączeń z serwerem w trakcie godziny, maksymalnej liczby zapytań lub uaktualnień wykonywanych w trakcie godziny itd. Użytkownik nie może obejść nałożonych ograniczeń liczby zapytań przez użycie wielu połączeń z serwerem, ponieważ zapytania we wszystkich połączeniach inicjowanych przez użytkownika korzystającego z danego konta obciążają to konto użytkownika.

Aby zdefiniować wspomniane ograniczenia, należy użyć klauzuli `WITH`. Przedstawione poniżej zapytanie powoduje utworzenie konta z pełnym dostępem do bazy danych `sampdb`.

Jednak użytkownik tego konta może połączyć się z serwerem jedynie dziesięciokrotnie w ciągu godziny i wykonać w ciągu godziny maksymalnie 200 zapytań, z których co najwyżej 50 to operacje uaktualnienia:

```
CREATE USER 'spike'@'localhost' IDENTIFIED BY 'pyrite';
GRANT ALL ON sampdb.* TO 'spike'@'localhost'
  WITH MAX_CONNECTIONS_PER_HOUR 10 MAX_QUERIES_PER_HOUR 200
  MAX_UPDATES_PER_HOUR 50;
```

Kolejność definicji w klauzuli WITH opcji zarządzania zasobami nie ma znaczenia.

Wartość domyślna każdej opcji wynosi zero, co oznacza brak ograniczeń. Dlatego też po nałożeniu ograniczeń na konto użytkownika ich zniesienie odbywa się przez zmianę na zero wartości ograniczeń. Na przykład, przedstawione poniżej zapytanie znosi ograniczenie liczby połączeń, jakie w ciągu godziny użytkownik spike może nawiązać z serwerem bazy danych:

```
GRANT USAGE ON *.* TO 'spike'@'localhost'
  WITH MAX_CONNECTIONS_PER_HOUR 0;
```

Opcja MAX_CONNECTIONS wskazuje maksymalną liczbę jednoczesnych połączeń, jakie może mieć dany użytkownik. Jeżeli jej wartość wynosi zero (to jest wartość domyślna), ograniczenie jest kontrolowane przez zmienną systemową max_user_connections. Wartość niezerowa wymienionej zmiennej ogranicza użytkownika do wskazanej liczby jednoczesnych połączeń.

Użytkownik administracyjny posiadający uprawnienie RELOAD może wyzerować wartość bieżącą licznika, wykonując zapytanie FLUSH USER_RESOURCES. Taki sam efekt ma wykonanie zapytania FLUSH PRIVILEGES. Po wyzerowaniu licznika użytkownicy, którzy osiągnęli godzinny limit, ponownie mogą nawiązać połączenie i zacząć wykonywać zapytania. Wyzerowanie liczników następuje także dla poszczególnych kont podczas wykonywania zapytania GRANT ustawiającego wielkość ograniczenia dla konta.

13.2.3. Wyświetlanie uprawnień użytkownika

Aby wyświetlić uprawnienia danego konta, należy wykonać zapytanie SHOW GRANTS:

```
SHOW GRANTS FOR 'sampadm'@'localhost';
```

W celu wyświetlenia własnych uprawnień można użyć dowolnego z poniższych zapytań:

```
SHOW GRANTS;
SHOW GRANTS FOR CURRENT_USER();
```

13.2.4. Odbieranie uprawnień

Aby odebrać pewne lub wszystkie uprawnienia konta, należy wykonać zapytanie REVOKE. Składnia zapytania REVOKE jest podobna do składni zapytania GRANT, za wyjątkiem faktu zastąpienia klauzuli TO przez FROM i braku klauzul *typ_uwierzytelnienia*, REQUIRE i WITH:

```
REVOKE uprawnienia [(kolumny)] ON poziom_uprawnień FROM użytkownik;
```


Na przykład, poniższe zapytanie GRANT nadaje wszystkie uprawnienia do bazy danych sampdb, natomiast zapytanie REVOKE odbiera użytkownikowi uprawnienia do przeprowadzania zmian w istniejących rekordach:

```
GRANT ALL ON sampdb.* TO 'boris'@'localhost';  
REVOKE DELETE,UPDATE ON sampdb.* FROM 'boris'@'localhost';
```

Uprawnienie GRANT OPTION nie jest dołączone w ALL. Jego odebranie użytkownikowi wymaga wyraźnego podania na liście uprawnień w zapytaniu REVOKE:

```
REVOKE GRANT OPTION ON sales.* FROM 'alicia'@'%.big-corp.com';
```

Operacja odebrania uprawnień sama wymaga uprawnień, między innymi GRANT OPTION.

W celu odebrania wszystkich uprawnień na wszystkich poziomach należy wykonać poniższe zapytanie:

```
REVOKE ALL, GRANT OPTION FROM użytkownik;
```

Zwróć uwagę na brak klauzuli ON w użytej składni. Wymagane jest globalne uprawnienie CREATE USER lub INSERT dla bazy danych mysql.

Jeżeli odbierzesz użytkownikowi wszystkie uprawnienia na poziomie bazy danych, tabeli, kolumny, procedury lub PROXY, MySQL usunie odpowiadające im rekordy z tabel db, tables_priv, columns_priv, procs_priv i proxies_priv. Odebranie użytkownikowi uprawnień globalnych powoduje wstawienie wartości N do kolumn uprawnień w rekordzie tabeli, ale nie usunięcie rekordu. Zapytanie REVOKE nie usuwa więc całkowicie konta, co nadal daje użytkownikowi możliwość nawiązania połączenia z serwerem. Aby całkowicie usunąć konto użytkownika, wykonaj zapytanie DROP USER zamiast REVOKE (patrz punkt 13.2.1, zatytułowany „Zarządzanie kontem MySQL na wysokim poziomie”).

Paradoksem jest fakt, że istnieje kilka operacji „odbierania” uprawnień przeprowadzanych za pomocą zapytania GRANT. Na przykład, jeśli użytkownik musi nawiązywać połączenie za pomocą protokołu SSL, nie istnieje składnia REVOKE pozwalająca na zniesienie takiego wymogu. Zamiast tego należy wykonać zapytanie GRANT nadające uprawnienie USAGE na poziomie globalnym (pozostawiając istniejące uprawnienia) i zawierające klauzulę REQUIRE NONE, która wskazuje na brak wymogu stosowania połączenia SSL:

```
GRANT USAGE ON *.* TO użytkownik REQUIRE NONE;
```

Podobnie, jeśli nałożyłeś na użytkownika ograniczenia w zakresie dostępnych zasobów, tego rodzaju ograniczeń nie zniesiesz za pomocą REVOKE. Zamiast tego wykonaj zapytanie GRANT wraz z klauzulą USAGE i ustaw ograniczeniom wartości zero, oznaczające „brak ograniczeń”

```
GRANT USAGE ON *.* TO użytkownik  
WITH MAX_CONNECTIONS_PER_HOUR 0 MAX_QUERIES_PER_HOUR 0  
MAX_UPDATES_PER_HOUR 0;
```

13.2.5. Zmiana hasła lub zerowanie zapomnianego

Jednym ze sposobów zmiany lub wyzerowania hasła jest wykonanie zapytania UPDATE identyfikującego w rekordzie tabeli user wartości User i Host dla konta użytkownika, a następnie ponowne wczytanie uprawnień:

```
mysql> USE mysql;
mysql> UPDATE user SET Password=PASSWORD('silicon')
    -> WHERE User='boris' AND Host='localhost';
mysql> FLUSH PRIVILEGES;
```

Jednak znacznie łatwiej jest użyć zapytania SET PASSWORD, ponieważ nazwa konta jest podawana w takim samym formacie jak w przypadku zapytań służących do zarządzania kontem i nie ma konieczności wyraźnego odświeżenia uprawnień:

```
mysql> SET PASSWORD FOR 'boris'@'localhost' = PASSWORD('silicon');
```

Własne hasło zawsze możesz zmienić za pomocą zapytania SET PASSWORD, o ile nie nawiązałeś połączenia jako użytkownik anonimowy. Aby zmienić hasło dla innego użytkownika, konieczne jest posiadanie uprawnienia UPDATE do bazy danych mysql.

Inny, rzadziej stosowany sposób zmiany hasła polega na wykonaniu zapytania GRANT USAGE wraz z klauzulą IDENTIFIED BY. W takim przypadku hasło jest podawane dosłownie, a nie za pomocą funkcji PASSWORD():

```
mysql> GRANT USAGE ON *.* TO 'boris'@'localhost' IDENTIFIED BY 'silicon';
```

Jeżeli zachodzi konieczność wyzerowania hasła użytkownika root z powodu jego zapomnienia i tym samym braku możliwości nawiązania połączenia z serwerem, to mamy pewien problem, ponieważ standardowa procedura oznacza nawiązanie połączenia jako użytkownik root i zmianę jego hasła. Jeżeli nie znasz hasła użytkownika root, będziesz zmuszony zatrzymać serwer i uruchomić go ponownie bez korzystania z tabeli uprawnień. Ta procedura została omówiona w punkcie 12.2.6, zatytułowanym „Odzyskanie kontroli nad serwerem, gdy nie można nawiązać z nim połączenia”.

13.2.6. Unikanie ryzyka związanego z kontrolą dostępu

W tym punkcie zostaną omówione kwestie, na które trzeba zwrócić uwagę podczas nadawania uprawnień, a także ryzyko związane z niewłaściwym wyborem.

Unikaj tworzenia kont anonimowych, czyli kont nieposiadających zdefiniowanej nazwy użytkownika. Nawet jeśli tego rodzaju konto użytkownika nie posiada wystarczających uprawnień, aby bezpośrednio dokonać pewnych szkód, to umożliwienie użytkownikowi nawiązania połączenia pozwala mu na uzyskanie dostępu do serwera. W ten sposób będzie mógł się rozejrzeć i zebrać pewne informacje o istniejących bazach danych, tabelach, a także monitorować serwer za pomocą zapytań takich jak SHOW STATUS i SHOW VARIABLES.

Wyszukaj konta niewymagające uwierzytelnienia i usuń je lub zdefiniuj dla nich hasła. W celu wyszukania kont użytkowników, które nie mają przypisanego hasła lub metody uwierzytelnienia, należy wykonać poniższe zapytanie:

```
SELECT Host, User FROM mysql.user WHERE Password = '' AND plugin = '';
```

Aby usunąć tego rodzaju konto użytkownika, trzeba wykonać zapytanie DROP USER. Włączenie autoryzacji następuje po przypisaniu hasła za pomocą poniższego zapytania:

```
SET PASSWORD FOR użytkownik = PASSWORD('hasło');
```

Nie pozwalaj na używanie haseł przechowywanych w oryginalnym (stosowanym w wersjach wcześniejszych niż MySQL 4.1) formacie hash i zmień je na znacznie bezpieczniejszy, obecny format hash. Wartości w starszym formacie miały długość 16 i nie rozpoczynały się od gwiazdki. Dlatego też konta użytkowników, które ich używają, możesz zidentyfikować za pomocą poniższych zapytań:

```
SELECT Host, User FROM mysql.user WHERE LENGTH(Password) = 16;  
SELECT Host, User FROM mysql.user WHERE Password NOT LIKE '%%';
```

Jeżeli powyższe zapytania wyszukają jakiekolwiek konto użytkownika:

1. Jeśli zmienna systemowa `old_passwords` ma przypisaną wartość 1 (włączona), uruchom ponownie serwer bez jej włączania.
2. W trakcie kolejnych uruchomień serwera zawsze włączaj zmienną systemową `secure_auth`. W ten sposób klienci nie będą miały możliwości zerowania haseł do starego formatu za pomocą `OLD_PASSWORD()` i nawiązywania połączenia z użyciem tego hasła. Począwszy od MySQL 5.6.5, ten krok nie jest konieczny, ponieważ zmienna `secure_auth` jest domyślnie włączona.
3. Wykonaj zapytanie `SET PASSWORD` w celu wyzerowania hasła dla każdego konta stosującego stary format haseł.

Ustaw wartość globalną `sql_mode` tak, aby zawierała tryb `NO_AUTO_CREATE_USER`. W ten sposób zapytanie `GRANT` nie będzie tworzyło nowych, niezabezpieczonych kont użytkowników. Oznacza to, że jeśli konto użytkownika nie istnieje, to wykonanie zapytania `GRANT` zakończy się niepowodzeniem. Nie nastąpi również utworzenie konta użytkownika, o ile zapytanie nie zawiera klauzuli `IDENTIFIED BY` wskazującej niepuste hasło lub `IDENTIFIED WITH` wskazującej wtórczkę uwierzytelnienia. Włączenie trybu `NO_AUTO_CREATE_USER` nie uniemożliwia klientowi wyłączenia wymienionego trybu w sesji, ale pomaga chronić niedoświadczonych użytkowników przed popełnianiem błędów.

O ile naprawdę nie musisz stosować wzorców w specyfikatorach nazw komputerów, unikaj tego podczas konfiguracji kont użytkowników. Zdefiniowanie szerokiego zakresu komputerów, z poziomu których dany użytkownik może nawiązać połączenie, oznacza jednocześnie większy zakres, z którego oszust podający się za użytkownika może spróbować włamać się do serwera.

Oszczędnie nadawaj uprawnienia superużytkownika, to znaczy nie włączaj uprawnień w rekordach tabeli `user`. Te uprawnienia są globalne i pozwalają użytkownikowi wpływać na działanie serwera lub uzyskiwać dostęp do dowolnej bazy danych. Na przykład, po włączeniu uprawnienia `DELETE` w rekordzie tabeli `user` użytkownik konta powiązanego z danym rekordem zyskuje możliwość usuwania rekordów z dowolnej tabeli w dowolnej bazie danych. Z powodu natury uprawnień wskazywanych w tabeli `user` (uprawnienia superużytkownika) najlepiej unikać nadawania uprawnień globalnych i zamiast tego nadawać uprawnienia na bardziej szczegółowym poziomie. W ten sposób użytkownik ma dostęp ograniczony do określonych baz danych lub obiektów, takich jak tabele bądź procedury składowane. Od wymienionej reguły istnieją dwa wyjątki:

- Superużytkownik taki jak root lub inny administrator potrzebuje uprawnień globalnych w celu zarządzania serwerem. Tego rodzaju kont powinno być jak najmniej.
- Kilka określonych uprawnień globalnych zwykle można bezpiecznie nadawać. Obejmują one tworzenie tabel tymczasowych, nakładanie blokad i (prawdopodobnie) możliwość wykonania zapytania `SHOW DATABASES`. Wiele instalacji nadaje wymienione uprawnienia, z kolei inne, o ścisłej kontroli, już niekoniecznie.

Nie nadawaj uprawnień dla bazy danych `mysql`. Zawiera ona tabele uprawnień, a więc użytkownik z uprawnieniami do wymienionej bazy danych będzie mógł modyfikować jej tabele w celu uzyskania uprawnień do innych baz danych. W efekcie nadanie użytkownikowi uprawnień do modyfikacji tabel bazy danych `mysql` daje mu wszystkie globalne uprawnienia. Jeżeli będzie mógł bezpośrednio modyfikować tabele, odpowiada to możliwości wykonania dowolnego zapytania związanego z zarządzaniem kontem.

Zachowaj ostrożność z uprawnieniem `GRANT OPTION`. Dwóch użytkowników z różnymi uprawnieniami i uprawnieniem `GRANT OPTION` może nawzajem zwiększyć swoje uprawnienia.

Uprawnienie `FILE` jest szczególnie niebezpieczne i nie nadawaj go lekką ręką. Poniżej przedstawiono przykład tego, co może zrobić użytkownik z uprawnieniem `FILE`:

```
CREATE TABLE etc_passwd (pwd_entry TEXT);
LOAD DATA INFILE '/etc/passwd' INTO TABLE etc_passwd;
```

Po wykonaniu powyższych zapytań użytkownik zyskuje dostęp do zawartości pliku haseł w serwerze. Wystarczy, że wykona proste zapytanie `SELECT`:

```
SELECT * FROM etc_passwd;
```

W zapytaniu `LOAD DATA` zamiast `/etc/passwd` można podać nazwę każdego dostępnego publicznie pliku w komputerze serwera. Jeżeli użytkownik nawiązał połączenie ze zdalnego komputera, efektem nadania mu uprawnienia `FILE` jest zapewnienie mu dostępu sieciowego do potencjalnie ogromnej części systemu plików komputera serwera.

Uprawnienie `FILE` można wykorzystać w celu włamania się do bazy danych w systemach, które mają niewystarczająco restrykcyjnie zdefiniowane uprawnienia do katalogu danych MySQL. To jest jeden z powodów, dla którego zawartość katalogu danych powinna być możliwa do odczytu jedynie przez serwer. Jeżeli pliki odpowiadające tabelom bazy danych są dostępne dla wszystkich użytkowników, odczytywać będą je mogli nie tylko użytkownicy posiadający konta w komputerze serwera, ale również dowolne klienty z uprawnieniem `FILE`. W takim przypadku mogą nawiązać połączenie przez sieć i odczytywać zawartość plików! Poniższa procedura pokazuje, jak można to zrobić:

1. Utwórz tabelę zawierającą kolumnę `LONGBLOB`:

```
USE test;
CREATE TABLE tmp (b LONGBLOB);
```

2. Użyj tabeli do odczytu zawartości każdego pliku odpowiadającego tabeli, którą chcesz ukraść. Przyjmujemy założenie, że użytkownik ma tabelę `MyISAM` o nazwie `x` w bazie danych `other_db`. Wymieniona tabela jest przedstawiona

w postaci trzech plików: *x.frm*, *x.myd* i *x.myi*. Odczyt tych plików i skopiowanie ich zawartości do tabel można przeprowadzić za pomocą poniższych zapytań:

```
LOAD DATA INFILE './other_db/x.frm' INTO TABLE tmp
  FIELDS ESCAPED BY '' LINES TERMINATED BY '';
SELECT * FROM tmp INTO OUTFILE 'x.frm'
  FIELDS ESCAPED BY '' LINES TERMINATED BY '';
DELETE FROM tmp;
LOAD DATA INFILE './other_db/x.MYD' INTO TABLE tmp
  FIELDS ESCAPED BY '' LINES TERMINATED BY '';
SELECT * FROM tmp INTO OUTFILE 'x.MYD'
  FIELDS ESCAPED BY '' LINES TERMINATED BY '';
DELETE FROM tmp;
LOAD DATA INFILE './other_db/x.MYI' INTO TABLE tmp
  FIELDS ESCAPED BY '' LINES TERMINATED BY '';
SELECT * FROM tmp INTO OUTFILE 'x.MYI'
  FIELDS ESCAPED BY '' LINES TERMINATED BY '';
```

- Po wykonaniu powyższych zapytań katalog bazy danych *test* będzie zawierał pliki o nazwach *x.frm*, *x.myd* i *x.myi*. Innymi słowy, baza danych *test* zawiera tabelę *x*, która jest ukradzionym duplikatem tabeli znajdującej się w bazie danych *other_db*.

Aby uniknąć tego rodzaju ataków, należy odpowiednio zdefiniować uprawnienia do katalogu danych MySQL, posługując się informacjami przedstawionymi w punkcie 13.1.2, zatytułowanym „Zabezpieczenie instalacji MySQL”. Dodatkowym zabezpieczeniem jest unikanie nadawania uprawnień do wykonywania zapytania `SHOW DATABASE` i uruchamianie serwera wraz z opcją `--skip-show-database`. W ten sposób użytkownicy nie będą mogli wykonywać zapytań `SHOW DATABASES` i `SHOW TABLES` względem baz danych, do których nie mają dostępu. Nie będą więc dowiadywali się o istnieniu baz danych i tabel, do których nie powinni mieć dostępu.

Niebezpieczeństwo związane z uprawnieniem `FILE` gwałtownie rośnie, jeśli serwer MySQL jest uruchamiany przez użytkownika `root`. Takie rozwiązanie jest gorąco odradzane, zwłaszcza w połączeniu z uprawnieniem `FILE`. Ponieważ użytkownik `root` może tworzyć pliki w dowolnym miejscu systemu plików, użytkownik z uprawnieniem `FILE` będzie mógł nakazać serwerowi to samo, nawet użytkownik, który nawiązał połączenie ze zdalnego komputera. Serwer nie nadpisze istniejących plików, ale czasami możliwe jest utworzenie nowych plików wpływających na działanie komputera serwera lub naruszających jego bezpieczeństwo. Na przykład, jeśli nie istnieje jeden z wymienionych plików: */etc/resolv.conf*, */etc/hosts.equiv*, */etc/hosts.lpd* lub */etc/sudoers*, użytkownik z uprawnieniami pozwalającymi MySQL na utworzenie tych plików może drastycznie zmienić zachowanie serwera.

Aby uniknąć tego rodzaju problemów, nie uruchamiaj serwera `mysqld` jako użytkownik `root`. (Zapoznaj się z podpunktem 12.2.1.1, zatytułowanym „Działanie serwera w ramach pozbawionego uprawnień konta logowania”).

Uprawnienia `PROCESS` i `SUPER` powinny być nadawane jedynie zaufanym użytkownikom MySQL. Mając uprawnienie `PROCESS`, użytkownik może wykonać zapytanie `SHOW PROCESSLIST` i tym samym poznać treść zapytań wykonywanych przez serwer. Zyskuje więc możliwość

„podglądania” innych użytkowników i prawdopodobnie informacji, które powinny pozostać prywatne. Z kolei mając uprawnienie SUPER, użytkownik może usuwać zapytania innych. Ponadto, uprawnienie SUPER pozwala na opróżnianie dzienników zdarzeń i przeprowadzanie innych operacji, które mogą zakłócić działanie serwera.

Nie nadawaj uprawnienia RELOAD użytkownikom, którzy go nie potrzebują. Wymienione uprawnienie pozwala na wykonywanie zapytań FLUSH i RESET, których można nadużyć na wiele sposobów:

- Pliki dzienników zdarzeń binarnego i przekazywania mogą mieć nazwy w postaci ponumerowanej sekwencji. Po włączeniu wymienionych dzienników zdarzeń każde zapytanie FLUSH LOGS tworzy kolejny plik w sekwencji. Użytkownik z uprawnieniem RELOAD może regularnie opróżniać dzienniki zdarzeń i wymusić utworzenie w serwerze ogromnej liczby plików.
- Użytkownik z uprawnieniem RELOAD może udaremnić działanie mechanizmu zarządzania zasobami przez ponowne wczytanie tabel uprawnień za pomocą FLUSH PRIVILEGES lub FLUSH USER_RESOURCES. Oba wymienione zapytania powodują wyzerowanie liczników zarządzania zasobami.
- Zapytanie FLUSH TABLES powoduje opróżnienie przez serwer bufora otwartych plików. Nieustanne wykonywanie tego zapytania zmniejsza wydajność serwera, uniemożliwiając mu używanie bufora. Podobnie, ciągle wykonywanie zapytania RESET QUERY CACHE niweluje korzyści płynące z użycia bufora zapytań.
- Zapytanie RESET MASTER powoduje, że serwer główny replikacji usuwa wszystkie pliki binarnego dziennika zdarzeń, nawet jeśli one nadal pozostają w użyciu. Uniemożliwia to przeprowadzenie prawidłowej replikacji do serwerów podległych.

Uprawnienie ALTER może być na wiele sposobów używane niezgodnie z przeznaczeniem. Przyjmujemy założenie, że chcesz, aby użytkownik miał dostęp do tabel a1, ale nie do tabel a2. Inny użytkownik z uprawnieniem ALTER może to zniwelować przez wykonanie zapytania ALTER TABLE w celu zmiany nazwy tabel a2 na tabel a1.

13.2.7. Wtyczki metod uwierzytelniania i użytkownicy proxy

Jak wyjaśniono w podpunkcie 13.2.1.3, zatytułowanym „Określanie sposobu uwierzytelniania użytkownika”, informacje o sposobie uwierzytelnienia w zapytaniach CREATE USER i GRANT są podawane za pomocą klauzuli IDENTIFIED BY definiującej hasło lub IDENTIFIED WITH definiującej wtyczkę uwierzytelnienia. W tym punkcie zostaną przedstawione dalsze informacje na temat użycia wtyczek uwierzytelnienia. Poznasz także użytkowników proxy, czyli dodatkowe możliwości oferowane przez wspomniane wtyczki w zakresie zyskiwania przez użytkownika uprawnień innego użytkownika.

13.2.7.1. Używanie wtyczek uwierzytelnienia

Składnia klauzuli IDENTIFIED WITH, służącej w zapytaniu CREATE USER (a także GRANT) do wskazania wtyczki uwierzytelnienia, jest następująca:

```
CREATE USER uzytkownik IDENTIFIED WITH wtyczka_uwierzytelnienia [AS  
'ciagg_tekstowy_uwierzytelnienia']
```

Aby wtyczka uwierzytelnienia była znana serwerowi, w pierwszej kolejności musi być wczytana. (Jeżeli nie została jeszcze wczytana, wczytaj ją za pomocą poleceń przedstawionych w podrozdziale 12.4, zatytułowanym „Interfejs wtyczek”). Przyjmując założenie o wczytaniu *wtyczki_uwierzytelnienia*, serwer tworzy nowy rekord w tabeli uprawnień user i przechowuje wartości *wtyczka_uwierzytelnienia* i *ciagg_tekstowy_uwierzytelnienia* w kolumnach plugin i authentication_string rekordu.

Kiedy klient spróbuje nawiązać połączenie z serwerem za pomocą tego konta użytkownika, serwer na podstawie wartości kolumny plugin w rekordzie tabeli user wie, że klient powinien być uwierzytelniony przez wymienioną wtyczkę. Wywołuje więc ją i przekazuje jej wartość *ciagg_tekstowy_uwierzytelnienia*. Wtyczka komunikuje się z programem klienta, otrzymuje jego dane uwierzytelniające i wskazuje klientowi, czy użytkownik został prawidłowo uwierzytelniony. Jeżeli nie, serwer odrzuca połączenie.

Wtyczki uwierzytelnienia po stronie serwera posiadają odpowiadające im wtyczki po stronie klienta, a każda wtyczka po stronie serwera wskazuje programowi klienta wtyczkę, którą powinien wywołać w celu przeprowadzenia procesu uwierzytelnienia. Z tego powodu użytkownicy nie muszą wskazywać wtyczek po stronie klienta wymaganych do nawiązania połączenia z serwerem. Jednak jeśli wtyczka po stronie klienta nie została zainstalowana w katalogu wtyczek w komputerze klienta, próba nawiązania połączenia zakończy się niepowodzeniem. Z kolei, jeżeli wtyczka znajduje się w innym położeniu niż domyślnie sprawdzane przez program klienta, należy go uruchomić z opcją --plugin-dir i wyraźnie wskazać położenie wtyczek.

Jeżeli serwer zostanie uruchomiony z opcją --skip-grant-tables, wtyczki uwierzytelniania nie będą używane, ponieważ serwer zezwoli każdemu klientowi na nawiązanie połączenia bez konieczności uwierzytelnienia.

13.2.7.2. Tworzenie użytkowników proxy

Wtyczki uwierzytelnienia w MySQL pozwalają na tworzenie użytkowników proxy. W ten sposób użytkownik A (proxy) może nawiązać połączenie z serwerem i być traktowany jak użytkownik B (czyli korzystający z uprawnień użytkownika działającego w charakterze proxy). Jeżeli użytkownik A działa w charakterze proxy dla B, a połączenie z serwerem zostanie nawiązane za pomocą konta użytkownika A, otrzymuje on uprawnienia użytkownika B i może przeprowadzać operacje dozwolone do wykonywania przez użytkownika B.

Nie każda wtyczka uwierzytelnienia obsługuje proxy. Aby zapewnić taką obsługę, musi być utworzona w sposób zwracający serwerowi nie tylko informacje o prawidłowym uwierzytelnieniu użytkownika A, ale także nazwę użytkownika B, jako który powinien być traktowany użytkownik A podczas sprawdzania uprawnień. Sposób, w jaki wtyczka

określa, czy i jak mapować użytkownika proxy na innego, w całości zależy od implementacji wtyczki. Najczęściej wtyczki używają wartości *ciąg_tekstowy_uwierzytelnienia* podanej w trakcie tworzenia konta użytkownika proxy. Dlatego też po nawiązaniu połączenia przez użytkownika proxy wtyczka sprawdza ciąg tekstowy uwierzytelnienia i interpretuje jego zawartość w celu ustalenia, jak mapować użytkownika na innego. Przyjmujemy założenie, że konto użytkownika zostało utworzone w następujący sposób:

```
CREATE USER '@'localhost
  IDENTIFIED WITH wtyczka_uwierzytelnienia
  AS 'user1=proxied_user1;user2=proxied_user2';
```

W takim przypadku nazwa konta używa pustej nazwy użytkownika i będzie dopasowywała próby nawiązania połączenia z komputera lokalnego, w których nazwa użytkownika nie zostanie dopasowana do wskazanego konta localhost. Podczas tego rodzaju prób, jeśli klient podał wartość user1 lub user2 opcji --user, wtyczka będzie mapowała klienta na odpowiednio proxied_user1 lub proxied_user2 i odrzucała próby pochodzące od klientów o innych nazwach użytkownika.

Aktualnie jedynie komercyjne dystrybucje MySQL zawierają wtyczki zapewniające obsługę proxy. (Więcej informacji szczegółowych na ten temat znajdziesz w podręczniku użytkownika MySQL). Dystrybucje utrzymywane przez społeczność nie zawierają tego rodzaju wtyczek. Z tego powodu dalsza analiza opisuje utworzenie użytkowników proxy za pomocą hipotetycznej wtyczki.

Przyjmujemy założenie, że mamy wtyczkę uwierzytelnienia o nazwie *unix_auth*, przeznaczoną dla systemów UNIX i działającą w następujący sposób:

- Wtyczka pozwala użytkownikom na nawiązanie połączenia z serwerem MySQL przy użyciu hasła do ich kont w systemie UNIX. Wtyczka może być używana bez proxy, ale obsługuje tę możliwość.
- Aby użyć wtyczki bez proxy, należy utworzyć konto użytkownika bez podawania ciągu tekstowego uwierzytelnienia:

```
CREATE USER 'user1'@'localhost' IDENTIFIED WITH unix_auth;
```

W takim przypadku użytkownik user1 nawiązuje połączenie w poniższy sposób i podaje hasło do swojego konta w systemie UNIX. Będzie uwierzytelniony dla konta MySQL o takiej samej nazwie użytkownika.

```
% mysql -p
```

```
Enter password: ...podaj hasło konta użytkownika user1 w systemie UNIX...
```

Wiersz poleceń nie zawiera nazwy użytkownika. Możemy założyć, że wtyczka działająca po stronie klienta sprawdzi środowisko uruchomieniowe i ustali prawidłową nazwę użytkownika w systemie UNIX, która powinna być przekazana do serwera MySQL.

- W celu użycia wtyczki wraz z proxy należy podać ciąg tekstowy uwierzytelnienia zawierający nazwy użytkowników MySQL, którzy będą stanowili proxy dla danego użytkownika, na przykład:

```
CREATE USER 'user2'@'localhost' IDENTIFIED WITH unix_auth AS 'my_user';
```


W takim przypadku użytkownik `user2` nawiązuje połączenie z serwerem, podając hasło dla konta użytkownika `user2` w systemie UNIX, ale otrzyma uprawnienia użytkownika `my_user` w MySQL.

Powyższa analiza jest pod pewnymi względami niekompletna. Poza wtyczką uwierzytelnienia zapewniającą obsługę proxy i kontem dla użytkownika proxy, procedura wymaga także istnienia konta otrzymującego uprawnienia. Ponadto, użytkownik proxy powinien mieć uprawnienie `PROXY` dla konta. Poniżej przedstawiono ogólną sekwencję zapytań, których wykonanie prowadzi do utworzenia użytkownika proxy:

```
CREATE USER uzytkownik_proxy informacje_uwierzytelnienia;  
CREATE USER uzytkownik_otrzymujacy_uprawnienia informacje_uwierzytelnienia;  
GRANT PROXY ON uzytkownik_otrzymujacy_uprawnienia TO uzytkownik_proxy;
```

Dla użytkownika proxy klauzula *informacje_uwierzytelnienia* musi zawierać nazwę wtyczki obsługującej proxy. Z kolei w przypadku użytkownika otrzymującego uprawnienia to musi być hasło lub nazwa wtyczki uwierzytelnienia.

Pełny zestaw poleceń pozwalających na utworzenie użytkownika `user2` jako proxy dla `my_user` przedstawia się następująco:

```
CREATE USER 'user2'@'localhost' IDENTIFIED WITH unix_auth AS 'my_user';  
CREATE USER 'my_user'@'localhost' IDENTIFIED BY 'my_user_password';  
GRANT PROXY ON 'my_user'@'localhost' TO 'user2'@'localhost';
```

Teraz, kiedy użytkownik `user2` nawiąże połączenie z serwerem MySQL, wydawane w trakcie sesji zapytania będą wykonywane w kontekście uprawnień użytkownika `my_user`. (Zwróć uwagę na przypisanie hasła dla użytkownika `my_user`. To uniemożliwia klientom bezpośrednie nawiązanie połączenia za pomocą tego konta użytkownika bez wcześniejszego uwierzytelnienia).

Być może zastanawiasz się, dlaczego zadawać sobie trud związany z mapowaniem użytkownika `user2` na `my_user`. Dlaczego uprawnień nie można bezpośrednio nadać użytkownikowi `user2`? Faktycznie, w przypadku tylko jednego użytkownika dla proxy takie rozwiązanie nie ma sensu. Przyjmijmy jednak założenie, że chcesz, aby wielu użytkowników systemu UNIX było traktowanych jako `my_user`. W takim przypadku możesz utworzyć dla nich konta, a następnie mapować je na `my_user`.

```
CREATE USER 'user3'@'localhost' IDENTIFIED WITH unix_auth AS 'my_user';  
GRANT PROXY ON 'my_user'@'localhost' TO 'user3'@'localhost';  
CREATE USER 'user4'@'localhost' IDENTIFIED WITH unix_auth AS 'my_user';  
GRANT PROXY ON 'my_user'@'localhost' TO 'user4'@'localhost';  
...
```

Ewentualnie chcesz, aby każdy użytkownik systemu UNIX nieposiadający konta w MySQL był mapowany na użytkownika `my_user`. W takim przypadku można skonfigurować ogólnego domyślnego użytkownika proxy i uprościć proces mapowania:

```
CREATE USER ''@'localhost' IDENTIFIED WITH unixauth AS 'my_user';  
GRANT PROXY ON 'my_user'@'localhost' TO ''@'localhost';
```

Takie podejście, polegające na mapowaniu wielu użytkowników na jednego, znacznie ułatwia zarządzanie uprawnieniami: uprawnienia dla każdego z nich możesz zmienić jednocześnie, po prostu modyfikując uprawnienia użytkownika proxy. W przeciwnym razie musiałbyś uaktualniać uprawnienia oddzielnie dla każdego konta.

13.3. Struktura i zawartość tabel uprawnień

System kontroli dostępu w MySQL jest bardzo elastyczny i pozwala skonfigurować uprawnienia na wiele różnych sposobów. Normalnie do zarządzania kontem użytkownika wykonujesz zapytania takie jak CREATE USER, GRANT i REVOKE, które w Twoim imieniu modyfikują tabele uprawnień kontrolujących uprawnienia dostępu danego klienta. Jednak być może uznasz, że uprawnienia użytkownika nie działają w oczekiwany sposób. W takich przypadkach przydaje się znajomość struktury tabel uprawnień w MySQL oraz sposobu ich używania przez serwer podczas ustalania uprawnień dostępu. Taka wiedza pozwoli Ci na dodawanie, usuwanie i modyfikowanie uprawnień użytkowników przez bezpośrednią modyfikację tabel uprawnień. Ponadto, w trakcie analizy tabel uprawnień zyskujesz możliwość diagnozowania związanych z nimi problemów.

Przyjmuję tutaj założenie, że zapoznałeś się z punktem 13.2.1, zatytułowanym „Zarządzanie kontem MySQL na wysokim poziomie”, i dobrze rozumiesz sposób działania zapytań zarządzania kontem użytkownika. Wspomniane zapytania stanowią wygodny interfejs pozwalający na konfigurację kont użytkowników w MySQL i powiązanych z nim uprawnień, ale rzeczywiste operacje są przeprowadzane w tabelach uprawnień MySQL.

Tabele uprawnień kontrolują dostęp do baz danych MySQL dla klientów nawiązujących połączenie z serwerem przez sieć. Wspomniane tabele znajdują się w bazie danych mysql i są inicjalizowane w trakcie pierwszego procesu instalacji serwera MySQL w komputerze. Tabele uprawnień noszą nazwy user, db, tables_priv, columns_priv, procs_priv i proxies_priv. Serwer w następujący sposób używa wymienionych tabel:

- Tabela user zawiera listę wszystkich użytkowników, którzy mogą nawiązać połączenie z serwerem i mają uprawnienia globalne (superużytkownika). Posiadane uprawnienia również są wymienione. Trzeba koniecznie pamiętać, że wszystkie uprawnienia zdefiniowane w tabeli user są globalne i mają zastosowanie względem *wszystkich baz danych*. Informacje dotyczące bezpiecznego nadawania uprawnień globalnych przedstawiono w punkcie 13.2.6, zatytułowanym „Unikanie ryzyka związanego z kontrolą dostępu”.

Tabela user zawiera także kolumny dotyczące uwierzytelnienia, opcji SSL związanych z wymogiem nawiązywania bezpiecznych połączeń oraz opcji związanych z zarządzaniem zasobami. Te ostatnie opcje mogą być używane w celu uniemożliwienia użytkownikowi zmonopolizowania serwera.

- Tabela db zawiera listę kont oraz baz danych, do których mają uprawnienia. Jeżeli w tym miejscu nadasz uprawnienia, będą stosowane do wszystkich obiektów (tabele, procedury składowane itd.) w danej bazie danych.

- Tabela `tables_priv` zawiera listę uprawnień na poziomie tabeli. Wszelkie wymienione tutaj uprawnienia mają zastosowanie do wszystkich kolumn w tabeli.
- Tabela `columns_priv` zawiera listę uprawnień na poziomie kolumn. Wszelkie wymienione tutaj uprawnienia mają zastosowanie do wskazanej kolumny w tabeli.
- Tabela `procs_priv` zawiera listę uprawnień procedur składowanych (funkcje i procedury). Wszelkie wymienione tutaj uprawnienia mają zastosowanie do wskazanej procedury w bazie danych.
- Tabela `proxies_priv` wymienia konta użytkowników, które mogą być używane w charakterze proxy dla innych kont użytkowników i przejąć ich uprawnienia.

Baza danych `mysql` zawiera także tabelę uprawnień o nazwie `host`, która została uznana za przestarzałą i nie będzie tutaj omawiana.

W kolejnych kilku tabelach, od 13.3 do 13.6, przedstawiono struktury poszczególnych tabel uprawnień podzielonych według typu kolumny. Każda tabela uprawnień zawiera dwa podstawowe rodzaje kolumn: zasięgu dostępu, wskazujące rekordy, względem których mają zastosowanie, oraz kolumny uprawnień, określające uprawnienia nadawane rekordom. Kolumny uprawnień można dalej dzielić na kolumny operacji administracyjnych i kolumny uprawnień powiązanych z operacjami na poszczególnych rodzajach obiektów. W tabeli `user` znajdują się dodatkowe kolumny dotyczące uwierzytelnienia, połączeń SSL i zarządzania zasobami. Wymienione kolumny znajdują się jedynie w tabeli `user`, ponieważ są stosowane globalnie. Niektóre z tabel uprawnień zawierają także inne kolumny, ale nie będziemy się nimi tutaj zajmować, ponieważ nie mają one zastosowania w zarządzaniu kontem.

Tabela 13.3. Istniejące w tabelach uprawnień kolumny dotyczące zasięgu udzielanego dostępu

Tabela user	Tabela db	Tabela tables_priv	Tabela columns_priv	Tabela procs_priv	Tabela proxies_priv
Host	Host	Host	Host	Host	Host
User	User	User	User	User	User
	Db	Db	Db	Db	Proxied_host
		Table_name	Table_name	Routine_name	Proxied_user
			Column_name	Routine_type	

Tabela 13.4. Istniejące w tabelach uprawnień kolumny dotyczące uprawnień w zakresie zadań administracyjnych

Tabela user	Tabela db	Tabela proxies_priv
Create_user_priv		
File_priv		
Grant_priv	Grant_priv	With_grant
Process_priv		
Reload_priv		

Tabela 13.4. Istniejące w tabelach uprawnień kolumny dotyczące uprawnień w zakresie zadań administracyjnych (ciąg dalszy)

Tabela user	Tabela db	Tabela proxies_priv
Repl_client_priv		
Repl_slave_priv		
Show_db_priv		
Shutdown_priv		
Super_priv		

Tabela 13.5. Istniejące w tabelach uprawnień kolumny dotyczące uprawnień do obiektów

Tabela user	Tabela db	Tabela tables_priv	Tabela columns_priv	Tabela procs_priv
Alter_priv	Alter_priv	Table_priv		Table_priv
Alter_routine_priv	Alter_routine_priv	Column_priv	Column_priv	
Create_priv	Create_priv			
Create_routine_priv	Create_routine_priv			
Create_tablespace_priv				
Create_tmp_table_priv	Create_tmp_table_priv			
Create_view_priv	Create_view_priv			
Delete_priv	Delete_priv			
Drop_priv	Drop_priv			
Event_priv	Event_priv			
Execute_priv	Execute_priv			
Index_priv	Index_priv			
Insert_priv	Insert_priv			
Lock_tables_priv	Lock_tables_priv			
References_priv	References_priv			
Select_priv	Select_priv			
Show_view_priv	Show_view_priv			
Trigger_priv	Trigger_priv			
Update_priv	Update_priv			

Tabela 13.6. Istniejące w tabeli `user` kolumny dotyczące uwierzytelniania, SSL i zarządzania zasobami

Kolumny uwierzytelnienia	Kolumny SSL	Kolumny zarządzania zasobami
Password	ssl_type	max_connections
plugin	ssl_cipher	max_questions
authentication_string	x509_issuer	max_updates
	x509_subject	max_user_connections

System uprawnień obejmuje tabele `tables_priv`, `columns_priv` i `procs_priv` w celu definicji uprawnień dla poszczególnych tabel, kolumn oraz procedur i funkcji składowanych. Nie ma tabeli `rows_priv`, ponieważ MySQL nie obsługuje uprawnień na poziomie rekordu. Na przykład, nie możesz ograniczyć użytkownikowi dostępu jedynie do tych rekordów w tabeli, które zawierają określoną wartość we wskazanej kolumnie. Aplikacje wymagające takich możliwości muszą je implementować we własnym zakresie. Jednym ze sposobów osiągnięcia takiego rozwiązania jest utworzenie widoku zdefiniowanego za pomocą `WITH CHECK OPTION` i nadanie mu odpowiednich uprawnień. Inne rozwiązanie polega na implementacji wspólnego nakładania blokad na poziomie rekordu za pomocą funkcji takich jak `GET_LOCK()` i `RELEASE_LOCK()`. Procedura stosująca wymienione rozwiązanie została przedstawiona w punkcie C.2.8, zatytułowanym „Funkcje nakładania blokad doradczych”.

Nowe wydania MySQL czasami powodują dodanie nowych tabel uprawnień lub kolumn. Na przykład, tabela `proxies_priv` została wprowadzona w wersji MySQL 5.5.7 w celu implementacji uprawnienia `PROXY`. Podczas uaktualnienia istniejącej instalacji do wymienionej wersji konieczne będzie uaktualnienie tabel uprawnień, zanim będzie można używać nowo wprowadzonego uprawnienia. W podrozdziale 13.2, zatytułowanym „Zarządzanie kontami użytkowników w MySQL”, omówiono odpowiednią procedurę niezbędną do przeprowadzenia wspomnianej operacji.

13.3.1. Istniejące w tabelach uprawnień kolumny dotyczące zasięgu udzielanego dostępu

Kolumny zasięgu określają rekordy tabeli uprawnień, których serwer będzie używać w celu ustalenia uprawnień, gdy użytkownik spróbuje wykonać daną operację. Każdy rekord tabeli uprawnień zawiera kolumny `Host` i `User` wskazujący, że dany rekord dotyczy połączeń nawiązywanych z podanego komputera przez wymienionego użytkownika. Na przykład, rekord tabeli `user` wraz z wartościami `localhost` i `bill` w kolumnach `Host` i `User` będzie używany w przypadku połączeń lokalnych nawiązywanych przez użytkownika `bill`, ale nie dla połączeń nawiązywanych przez `betty`. Pozostałe tabele mogą zawierać dodatkowe kolumny zasięgu. Tabela `db` zawiera kolumnę `Db` wskazującą bazę danych, dla której ma zastosowanie. Podobnie, rekordy w tabelach `tables_priv` i `columns_priv`

zawierają kolumny zasięgu, które jeszcze bardziej zawężają zasięg do konkretnej tabeli w bazie danych lub kolumny w tabeli. Z kolei kolumny zasięgu w tabeli `procs_priv` wskazują funkcję lub procedurę, względem której ma zastosowanie dany rekord.

13.3.2. Istniejące w tabelach uprawnień kolumny uprawnień

Tabele uprawnień zawierają także kolumny uprawnień. Dla każdego rekordu tego rodzaju kolumna wskazuje uprawnienia, jakie użytkownik ma w zakresie definiowanym przez kolumny zasięgu. Opis uprawnień obsługiwanych przez MySQL przedstawiono w podpunkcie 13.2.2.1, zatytułowanym „Definiowanie uprawnień użytkownika”. W większości przypadków nazwy uprawnień są powiązane z nazwami kolumn uprawnień znajdujących się w tabelach uprawnień. Na przykład, uprawnienie `SELECT` odpowiada kolumnie `Select_priv`.

W tabelach `user` i `db` każde uprawnienie jest podawane jako oddzielna kolumna. Wspomniane kolumny są zdefiniowane jako typ `ENUM('N', 'Y')`, a wartością domyślną jest `N` (uprawnienie wyłączone). Na przykład, kolumna `Select_priv` jest zdefiniowana w poniższy sposób:

```
Select_priv ENUM('N','Y') CHARACTER SET utf8 NOT NULL DEFAULT 'N'
```

Uprawnienia w tabelach `tables_priv`, `columns_priv` i `procs_priv` są przedstawiane przez zbiór (SET) pozwalający na przechowywanie w pojedynczej kolumnie dowolnej kombinacji uprawnień. Kolumna `Table_priv` w tabeli `tables_priv` została zdefiniowana w następujący sposób:

```
SET('Select','Insert','Update','Delete','Create','Drop','Grant',
    'References','Index','Alter','Create_view','Show_view','Trigger')
CHARACTER SET utf8 NOT NULL DEFAULT ''
```

Kolumna `Column_priv` w tabeli `columns_priv` jest zdefiniowana następująco:

```
SET('Select','Insert','Update','References')
CHARACTER SET utf8 NOT NULL DEFAULT ''
```

Istnieje mniej uprawnień kolumn niż uprawnień tabel, ponieważ mniejsza liczba operacji ma sens na poziomie kolumn. Na przykład, możesz usunąć rekord z tabeli, pozbywając się go w ten sposób, ale nie możesz usunąć poszczególnych kolumn rekordu.

Zwróć uwagę na istnienie uprawnienia `INSERT` na poziomie kolumny. Jeżeli masz uprawnienie `INSERT` jedynie dla niektórych kolumn tabeli, wtedy podczas wstawiania nowego rekordu możesz podać wartości jedynie dla tych kolumn, a pozostałe kolumny otrzymają wartości domyślne.

Kolumna `Proc_priv` w tabeli `procs_priv` została zdefiniowana w następujący sposób:

```
SET('Execute','Alter Routine','Grant')
CHARACTER SET utf8 NOT NULL DEFAULT ''
```

Tabele `tables_priv`, `columns_priv` i `procs_priv` są nowsze niż tabele `user` i `db` i dlatego używają zbioru (SET), czyli znacznie efektywniejszego rozwiązania w zakresie przedstawienia wielu wartości w pojedynczej kolumnie.

Tabela `user` zawiera pewne kolumny uprawnień administracyjnych nieobecnych w innych tabelach uprawnień. Zaliczamy do nich `File_priv`, `Process_priv`, `Reload_priv` i `Shutdown_priv`. Wymienione uprawnienia znajdują się jedynie w tabeli `user`, ponieważ są globalne i niepowiązane z żadną konkretną bazą danych lub tabelą. Na przykład, nie ma przecież żadnego sensu zezwalać użytkownikowi na wyłączanie serwera na podstawie wybranej domyślnej bazy danych lub tego zabraniać.

Tabela `proxies_priv` przedstawia relacje uprawnienia PROXY przez wskazanie w kolumnach `User` i `Host` konta użytkownika proxy oraz w kolumnach `Proxied_user` i `Proxied_host` konta użytkownika nabywającego uprawnień.

13.3.3. Istniejące w tabelach uprawnień kolumny uwierzytelnienia

Tabela `user` zawiera trzy kolumny wskazujące sposób uwierzytelniania kont: `Password`, `plugin` i `authentication_string`. Jeżeli w rekordzie `user` dla danego konta kolumna `plugin` jest pusta, klient będzie uwierzytelniany za pomocą hasła zdefiniowanego w kolumnie `Password`. Hasła są puste lub niepuste, a znaki wieloznaczne są niedozwolone. Puste hasło nie oznacza dopasowania każdego hasła, ale brak konieczności podawania hasła w trakcie nawiązywania połączenia.

Jeżeli kolumna `plugin` nie jest pusta, klient będzie uwierzytelniany zgodnie z metodą implementowaną przez wymienioną wtyczkę, a serwer przekazuje wartość kolumny `authentication_string` jako informacje dla wtyczki. Więcej informacji na ten temat przedstawiono w podpunkcie 13.2.1.3, zatytułowanym „Określanie sposobu uwierzytelniania użytkownika”.

Hasła są przechowywane w postaci zaszyfrowanych wartości, a nie zwykłego tekstu. Jeżeli w kolumnie `Password` będziesz przechowywał hasło w postaci zwykłego tekstu, użytkownik nie będzie w stanie nawiązać połączenia. Zapytania `CREATE USER` i `GRANT` oraz polecenie `mysqladmin password` powodują automatyczne szyfrowanie hasła. Jeżeli do modyfikacji tabel uprawnień używasz zapytań takich jak `INSERT`, `REPLACE`, `UPDATE` lub `SET PASSWORD`, hasło podaj jako `PASSWORD('nowe_hasło')`, a nie jako `'nowe_hasło'`.

13.3.4. Istniejące w tabelach uprawnień kolumny dotyczące SSL

Kilka kolumn tabeli `user` dotyczy uwierzytelniania za pomocą protokołu SSL. (Zapoznaj się z podrozdziałem 13.5, zatytułowanym „Konfiguracja bezpiecznych połączeń za pomocą SSL”). Podstawowa kolumna tego rodzaju to `ssl_type`, określająca, czy dany użytkownik musi nawiązywać bezpieczne połączenie, i wskazująca jego typ. Kolumna `ssl_type` została zdefiniowana jako typ `ENUM` i pozwala na użycie jednej z czterech wartości:

```
ENUM('', 'ANY', 'X509', 'SPECIFIED') CHARACTER SET utf8 NOT NULL DEFAULT ''
```

Poniżej omówiono wartości typu wyliczeniowego `ssl_type`:

- `' '` (pusty ciąg tekstowy) wskazuje, że użytkownik konta nie musi nawiązywać bezpiecznego połączenia. To jest wartość domyślna używana w trakcie konfiguracji konta bez podania klauzuli `REQUIRE` lub po podaniu klauzuli `REQUIRE NONE`.
- `ANY` oznacza, że użytkownik musi nawiązywać bezpieczne połączenie, ale może być ono dowolnego rodzaju. To jest tylko „ogólny” wymóg. Kolumna ma ustawioną tę wartość po użyciu klauzuli `REQUIRE SSL` w zapytaniu `GRANT`.
- `X509` wskazuje na konieczność użycia bezpiecznego połączenia oraz dostarczenia przez klienta ważnego certyfikatu `X509`. Zawartość certyfikatu nie ma znaczenia. Kolumna ma ustawioną tę wartość po użyciu klauzuli `REQUIRE X509`.
- `SPECIFIED` wskazuje na konieczność użycia bezpiecznego połączenia spełniającego określone wymagania. Kolumna ma ustawioną tę wartość po użyciu w klauzuli `REQUIRE` dowolnego połączenia wartości `ISSUER`, `SUBJECT` lub `CIPHER`.

Dla wszystkich wartości `ssl_type` poza `SPECIFIED` podczas weryfikacji próby nawiązania połączenia przez klienta serwer ignoruje wartości w pozostałych kolumnach dotyczących SSL. Z kolei w przypadku typu `SPECIFIED` serwer sprawdza pozostałe kolumny i jeśli za wierają one niepuste wartości, to klient musi dostarczyć odpowiednie informacje:

- `ssl_cipher` — jeśli wartość jest niepusta, to wskazuje metodę szyfrowania, która musi być stosowana przez klienta w trakcie nawiązywania połączenia. Dzięki zdefiniowanemu tutaj wymaganiu można uniemożliwić klientowi stosowanie słabych metod szyfrowania.
- `x509_issuer` — jeśli wartość jest niepusta, to wskazuje wartość definiującą wydawcę, która musi znajdować się w certyfikacie `X509` przedstawianym przez klienta.
- `x509_subject` — jeśli wartość jest niepusta, to wskazuje wartość definiującą temat, która musi znajdować się w certyfikacie `X509` przedstawianym przez klienta.

Wartości `ssl_cipher`, `x509_issuer` i `x509_subject` są przedstawiane w tabeli `user` jako kolumny typu `BLOB`.

13.3.5. Istniejące w tabelach uprawnień kolumny zarządzania zasobami

Wymienione poniżej kolumny w tabeli `user` pozwalają na ograniczenie zasobów serwera, jakie mogą być wykorzystane przez danego użytkownika:

- `max_connections` — określa maksymalną liczbę połączeń w ciągu godziny, jakie użytkownik może nawiązać z serwerem. Wartość zero oznacza „brak ograniczeń”. Nazwa tej kolumny jest taka sama jak zmiennej systemowej `max_connections`, ale są one niepowiązane ze sobą.
- `max_questions` — określa maksymalną liczbę zapytań w ciągu godziny, jakie użytkownik może wykonać. Wartość zero oznacza „brak ograniczeń”.

- `max_updates` — ta wartość ma znaczenie podobne do `max_questions`, ale jest stosowana względem zapytań modyfikujących dane. Wartość zero oznacza „brak ograniczeń”.
- `max_user_connections` — określa maksymalną dozwoloną liczbę jednoczesnych połączeń klienta. W przypadku wartości zero serwer będzie stosował ograniczenie zdefiniowane za pomocą globalnej wartości zmiennej systemowej `max_user_connections`. Wartość większa niż zero ma pierwszeństwo przed wartością wymienionej zmiennej systemowej.

Jeżeli nastąpi ponowne uruchomienie serwera, wartość liczników jest zerowana. Zerowanie (poza wartością `max_user_connections`) następuje również po ponownym odczytaniu tabel uprawnień lub wykonaniu zapytania `FLUSH USER_RESOURCES`.

Więcej informacji na temat nakładania ograniczeń na konto użytkownika przedstawiono w podpunkcie 13.2.2.5, zatytułowanym „Ograniczanie zasobów dostępnych dla użytkownika”.

13.4. W jaki sposób serwer kontroluje dostęp uzyskiwany przez klientów?

Serwer MySQL wymusza dwa etapy kontroli dostępu uzyskiwanego przez klienta. Etap pierwszy następuje podczas próby nawiązania połączenia z serwerem. Wówczas serwer sprawdza tabelę `user` w poszukiwaniu rekordu dopasowanego do komputera, z którego zainicjowano próbę połączenia, nazwy użytkownika i danych uwierzytelniających (na przykład hasła). W przypadku braku takiego rekordu nie będzie można nawiązać połączenia. Jeśli rekord zostanie dopasowany, serwer sprawdzi także kolumny w tabeli `user` powiązane z protokołem SSL i zarządzaniem zasobami:

- Po osiągnięciu godzinnego limitu połączeń lub liczby maksymalnych jednoczesnych połączeń serwer odrzuci dane połączenie.
- Gdy tabela `user` wskazuje na konieczność nawiązania bezpiecznego połączenia, serwer ustala, czy dostarczone przez klienta dane uwierzytelniające odpowiadają zdefiniowanym w kolumnach dotyczących SSL. Jeśli nie, serwer odrzuca połączenie.

Jeżeli wszystko przebiegnie prawidłowo, serwer nawiąże połączenie i będzie można przejść do etapu drugiego. W przypadku nawiązywania bezpiecznego połączenia program klienta i serwer szyfrują prowadzoną między sobą komunikację.

Na drugim etapie serwer sprawdza dwie rzeczy w każdym wykonywanym zapytaniu. Po pierwsze, serwer sprawdza wartość licznika wskazującego liczbę zapytań wykonanych w trakcie godziny oraz zapytań uaktualniających dane. Po drugie, serwer sprawdza tabele uprawnień w celu potwierdzenia wystarczających uprawnień klienta do wykonania danego zapytania. Liczniki zapytań są sprawdzane jako pierwsze, ponieważ w przypadku osiągnięcia ich wartości maksymalnych nie ma sensu sprawdzanie uprawnień. Etap drugi trwa aż do chwili zakończenia połączenia z serwerem.

Poniżej znajdziesz dokładniejszy opis reguł stosowanych przez serwer MySQL podczas dopasowywania rekordów tabeli uprawnień z otrzymywanymi żądaniami nawiązania połączenia i wykonania zapytań. Dowiesz się między innymi, jakie typy wartości są uznawane za prawidłowe w kolumnach zasięgu tabeli uprawnień, jak łączone są uprawnienia z różnych tabel uprawnień oraz w jakiej kolejności serwer wyszukuje rekordy w danej tabeli uprawnień.

13.4.1. Zawartość kolumn zasięgu

Każda kolumna zasięgu rządzi się regułami definiującymi dozwolone wartości oraz sposobami ich interpretacji przez serwer. Niektóre z wspomnianych kolumn wymagają wartości dosłownych, natomiast większość z nich dopuszcza użycie znaków wieloznacznych lub innych wartości specjalnych.

■ Host

Wartością kolumny Host może być nazwa komputera lub adres IP (IPv4 bądź IPv6). Wartość `localhost` oznacza komputer lokalny. Będzie dopasowana, jeśli klient nawiązuje połączenie z komputera lokalnego do jednego z lokalnych interfejsów sieciowych serwera zdefiniowanych w następujący sposób:

- ◆ Plik gniazda UNIX w systemach UNIX.
- ◆ Nazwany potok lub pamięć współdzielona w systemach Windows.
- ◆ Interfejs TCP loopback, to znaczy interfejs o adresie IP `127.0.0.1` lub `::1`.

Ten interfejs działa w dowolnym systemie operacyjnym.

Wartość `localhost` nie zostanie dopasowana, jeśli klient nawiązuje połączenie, podając rzeczywistą nazwę komputera lub jego adres IP. Przyjmujemy założenie, że nazwa komputera lokalnego to `cobra.example.com`, a tabela `user` zawiera dwa rekordy dla użytkownika bob. W jednym wartością kolumny Host jest `localhost`, natomiast w drugim wartością jest `cobra.example.com`. Rekord zawierający wartość `localhost` będzie dopasowany, gdy użytkownik bob nawiąże połączenie z komputera lokalnego, używając jednego z poniższych poleceń w systemach UNIX lub Windows:

```
% mysql -p -u bob -h localhost
% mysql -p -u bob -h 127.0.0.1
% mysql -p -u bob -h ::1
```

W systemie Windows rekord `localhost` będzie dopasowany także, jeśli użytkownik bob spróbuje nawiązać połączenie w następujący sposób:

```
C:\> mysql -p -u bob -h .
C:\> mysql -p -u bob --protocol=pipe
C:\> mysql -p -u bob --protocol=memory
```

Rekord o wartości `cobra.example.com` kolumny Host zostanie dopasowany, jeśli użytkownik bob nawiąże połączenie z komputera lokalnego, podając nazwę komputera serwera (`cobra.example.com`) lub odpowiadający mu adres IP.

W obu przypadkach połączenie odbywa się za pomocą *protokołu* TCP/IP.

Wartości kolumny Host można także podawać, używając znaków wieloznacznych. Mogą być stosowane wzorce SQL w postaci znaków % i _, a ich znaczenie jest dokładnie takie samo, jak w operatorze LIKE w zapytaniu. (Niedozwolone jest użycie wyrażeń regularnych typu stosowanego w REGEXP). Znaki wzorca SQL działają zarówno dla nazw, jak i adresów IP. Na przykład, %.example.com powoduje dopasowanie dowolnego komputera w domenie example.com, natomiast %.edu dopasowuje każdy komputer z dowolnej instytucji edukacyjnej. Podobnie, 10.0.% dopasowuje dowolny komputer w podsieci 10.0 (klasa B), podczas gdy 192.168.3.% dopasowuje dowolny komputer w podsieci 192.168.3 (klasa C).

Wartość % w kolumnie Host dopasowuje dowolny komputer i pozwala użytkownikowi na nawiązanie połączenia z dowolnego miejsca. Pusta wartość kolumny Host w tabeli uprawnień ma takie samo znaczenie jak wartość %, ale z jednym wyjątkiem. W tabeli db pusta wartość Host oznacza „sprawdź tabelę host w celu uzyskania dalszych informacji”. Jednak tabela host została uznana za przestarzałą i dlatego nie powinien używać pustej wartości db.Host.

W przypadku wartości IPv4 można podać liczbę sieciową wraz z maską wskazującą bity adresu IP klienta, które muszą być dopasowane. Na przykład, 192.168.128.0/255.255.255.0 wskazuje 24-bitowy numer sieciowy i powoduje dopasowanie każdego komputera klienta, którego pierwsze 24 bity adresu IP mają wartość identyczną z 192.168.128. Możesz to uznać za jeszcze inny rodzaj znaku wieloznacznego. Wartością maski sieciowej musi być 255.0.0.0, 255.255.0.0, 255.255.255.0 lub 255.255.255.255. Oznacza to, że musi rozpoczynać się od wielokrotności ośmiu bitów, którym przypisano wartość 1, a pozostałe bity muszą mieć wartość 0.

Maski sieciowe nie są obsługiwane w adresach IPv6.

■ User

Nazwa użytkownika musi być podana w postaci wartości dosłownej lub pustej. Wartość pusta powoduje dopasowanie każdej nazwy, a więc oznacza użytkownika „anonimowego”. W pozostałych przypadkach wartość musi dokładnie odpowiadać podanej nazwie. Wartość % w kolumnie User nie jest uznawana za pustą. Powoduje dopasowanie użytkownika o dosłownej nazwie %, czyli prawdopodobnie nie jest rozwiązaniem, którego oczekiwałeś.

Kiedy nadchodzące połączenie jest weryfikowane w tabeli user, jeśli pierwszy dopasowany rekord zawiera pustą wartość kolumny User, klient będzie uznany za użytkownika anonimowego.

■ Db

W tabeli db wartość Db może być podana dosłownie lub za pomocą znaków % i _, uznawanych w SQL za znaki wieloznaczne. Wartość % lub pusta powoduje dopasowanie dowolnej bazy danych. W tabelach columns_priv, tables_priv i procs_priv wartości Db muszą być dosłownymi nazwami baz danych. Wówczas dopasowana zostaje dokładna nazwa bazy danych. Niedozwolone jest używanie wzorców i pustych wartości.

■ **Table_name, Column_name, Routine_name**

Wartość w wymienionych kolumnach musi być dosłowną nazwą odpowiednio tabeli, kolumny lub procedury składowej. Wartości powodują dopasowanie dokładnie podanej nazwy. Niedozwolone jest używanie wzorców i pustych wartości.

■ **Routine_type**

Wartością w tej kolumnie musi być FUNCTION lub PROCEDURE. Wskazuje ona, czy wartość w kolumnie Routine_name rekordu odnosi się do funkcji składowanej, czy do procedury składowanej. Wartości Routine_name i Routine_type unikalnie identyfikują procedurę składowaną w bazie danych wskazanej w kolumnie Db.

■ **Proxied_host, Proxied_user**

Wymienione kolumny pojawiają się w tabeli proxies_priv, która również ma kolumny Host i User wskazujące konto użytkownika proxy. Kolumny Proxied_host i Proxied_user oznaczają użytkownika otrzymującego uprawnienia i wartości takie jak podane wcześniej w kolumnach Host i User.

Kolumny dotyczące zasięgu są przez serwer traktowane jako rozróżniające wielkość liter lub jej nierozróżniające, zgodnie z tabelą 13.7. Zwróć szczególnie uwagę na fakt, że wartości Db i Table_name zawsze rozróżniają wielkość liter. Nie ma tutaj żadnego znaczenia, że w zapytaniach SQL traktowanie nazw baz danych i tabel zależy od rozróżniania wielkości liter przez system plików, w którym został uruchomiony serwer. (W systemach UNIX wielkość liter zwykle ma znaczenie, w systemach Windows wręcz przeciwnie).

Tabela 13.7 Rozróżnianie wielkości liter w kolumnach tabeli Grant

Kolumna	Rozróżnianie wielkości liter
Host, Proxied_host	Nie
User, Proxied_user	Tak
Password	Tak
Db	Tak
Table_name	Tak
Column_name	Nie
Routine_name	Nie

13.4.2. Weryfikacja uprawnień zapytania

Za każdym razem, gdy wykonujesz zapytanie SQL, serwer sprawdza, czy osiągnąłeś limity zasobów dotyczących zapytań. Wspomniane limity są określone przez wartości max_questions i max_updates przechowywane w tabeli user. Jeżeli limity nie zostały osiągnięte, serwer sprawdza, czy masz wystarczające uprawnienia do wykonania zapytania. Określenie uprawnień następuje przez sprawdzenie uprawnień w tabelach

user, db, tables_priv, columns_priv i procs_priv aż do chwili potwierdzenia przez serwer uprawnień lub przeszukania wszystkich tabel. Dokładna procedura przedstawia się następująco:

1. Serwer sprawdza rekord tabeli user dopasowany podczas nawiązywania połączenia, aby przekonać się, czy masz jakiekolwiek uprawnienia globalne. W przypadku znalezienia uprawnień globalnych wystarczających do wykonania zapytania serwer je wykonuje.
2. Jeżeli uprawnienia globalne są niewystarczające, serwer wyszukuje rekord dopasowany do użytkownika w tabeli db. Gdy taki rekord zostanie znaleziony, zdefiniowane tam uprawnienia są dodawane do uprawnień globalnych użytkownika. Jeżeli tak zebrane uprawnienia są wystarczające do wykonania zapytania, serwer je wykonuje.
3. Jeżeli połączenie uprawnień globalnych i na poziomie bazy danych jest niewystarczające, serwer sprawdza tabele tables_priv, columns_priv i procs_priv, aby przekonać się, czy masz uprawnienia wystarczające do wykonania zapytania.
4. Jeżeli po sprawdzeniu wszystkich wymienionych tabel nadal nie masz uprawnień wymaganych do wykonania zapytania, serwer odrzuca tę próbę wykonania zapytania.

W kategoriach boolowskich uprawnienia z tabel uprawnień serwer łączy następująco:

user OR db OR tables_priv OR columns_priv or procs_priv

Przedstawiony powyżej opis jasno pokazuje, że sprawdzenie uprawnień dostępu to zdecydowanie skomplikowany proces, zwłaszcza jeśli pod uwagę wziąć sprawdzanie przez serwer uprawnień dla każdego zapytania otrzymywanego od klienta. Jednak sam proces jest całkiem szybki, ponieważ w rzeczywistości serwer nie musi sprawdzać informacji w tabelach uprawnień dla każdego zapytania. Zamiast tego podczas uruchamiania wczytuje zawartość tabel uprawnień do pamięci, a następnie weryfikuje uprawnienia za pomocą umieszczonej w pamięci kopii tabel uprawnień. W ten sposób ich sprawdzanie przebiega naprawdę szybko. Co więcej, im prostszy system uprawnień, tym szybciej przebiega cała operacja. Kiedy serwer wczytuje tabele uprawnień do pamięci, zauważa, czy jakiekolwiek konto użytkownika ma ograniczenia w dostępnych zasobach bądź też zdefiniowane uprawnienia na poziomie tabel, kolumn lub procedur składowanych. W przypadku ich braku serwer wie, że nie musi sprawdzać żadnych tego rodzaju informacji podczas weryfikacji uprawnień dla zapytań wysyłanych przez klientów. Oznacza to możliwość pominięcia przez serwer pewnych kroków z pełnej procedury sprawdzania dostępu.

Użycie umieszczonych w pamięci kopii tabel uprawnień podczas weryfikacji uprawnień dostępu ma jeden ważny efekt uboczny: po bezpośredniej zmianie zawartości tabel uprawnień serwer nie zauważy zmiany. Na przykład, dodanie nowego użytkownika MySQL za pomocą zapytania INSERT wstawiającego nowy rekord do tabeli user nie oznacza automatycznego włączenia temu użytkownikowi możliwości nawiązania połączenia z serwerem. Bardzo

często to stanowi zaskoczenie dla początkujących administratorów (a czasem także dla doświadczonych). Rozwiązanie jest jednak bardzo proste: po przeprowadzeniu bezpośredniej modyfikacji tabel uprawnień należy nakazać serwerowi ponowne wczytanie ich zawartości do pamięci. Wystarczy wykonać zapytanie `FLUSH PRIVILEGES` bądź też wykonać polecenie `mysqladmin flush-privileges` lub `mysqladmin reload`.

Nie ma konieczności nakazania serwerowi ponownego wczytania tabel uprawnień po wykonaniu zapytania `CREATE USER`, `DROP USER`, `RENAME USER`, `GRANT`, `REVOKE` lub `SET PASSWORD` w celu konfiguracji lub modyfikacji konta użytkownika. Wymienione zapytania serwer mapuje jako modyfikujące tabele uprawnień i automatycznie odświeża ich kopie znajdujące się w pamięci.

13.4.3. Kolejność dopasowania kolumn zasięgu

Serwer MySQL sortuje rekordy tabel uprawnień w szczególny sposób, a następnie próbuje dopasować nadchodzące połączenia przez sprawdzanie kolejnych rekordów. Pierwsze znalezione dopasowanie określa używany rekord. Bardzo ważne jest zrozumienie stosowanej kolejności sortowania, zwłaszcza dla tabeli `user`. Na tym potyka się wiele osób, które próbują zrozumieć zapewnienie bezpieczeństwa przez MySQL.

Kiedy serwer odczytuje zawartość tabeli `user`, rekordy sortuje według wartości kolumn `Host` i `User`. Kolumna `Host` jest dominująca, więc rekordy o takiej samej wartości `Host` są sortowane razem, a następnie układane w kolejności wartości kolumny `User`. Jednak sortowanie nie jest przeprowadzane w kolejności leksykalnej, a jedynie częściowo w leksykalnej. Reguła, o której należy pamiętać, jest następująca: serwer preferuje wartości dosłowne zamiast wzorców i dokładne wzorce zamiast bardziej ogólnych. Oznacza to, że jeśli nawiązujesz połączenie z komputera `boa.example.com` i tabela `user` zawiera rekordy o wartości `boa.example.com` i `%.example.com` dla kolumny `Host`, serwer preferuje pierwszy z wymienionych rekordów. Podobnie, wartość `%.example.com` jest bardziej preferowana niż `%.com`, która z kolei jest bardziej preferowana od `%`. Dopasowanie adresów IP działa bardzo podobnie. W przypadku klienta nawiązującego połączenie z komputera o adresie IP `192.168.3.14` zostaną dopasowane rekordy zawierające następujące wartości kolumny `Host`, w kolejności od najbardziej do najmniej preferowanej:

```
192.168.3.14
192.168.3.%
192.168.%
192.%
%
```

Inna reguła, o której należy pamiętać: kiedy serwer próbuje dopasować rekordy tabeli `user`, najpierw szuka dopasowania wartości kolumny `Host`, a dopiero później `User`, a nie na odwrót.

13.4.4. Puzzle uprawnień

W tym punkcie poznasz scenariusz pokazujący użyteczność zrozumienia kolejności, w jakiej serwer wyszukuje rekordy tabeli `user` podczas weryfikacji próby nawiązania połączenia. Dowiesz się także, jak rozwiązywać problem dość często pojawiający się w przypadku nowych instalacji MySQL, o czym świadczy częstotliwość, z jaką pojawia się na listach dyskusyjnych poświęconych MySQL: administrator konfiguruje nową instalację łącznie z domyślnymi rekordami w tabeli `user` dla użytkowników `root` i anonimowego. Dobry administrator przypisze hasło użytkownikowi `root`, ale często spotykaną praktyką (mimo odradzania jej stosowania) jest pozostawienie konta użytkownika anonimowego bez zmian, czyli bez przypisanego hasła. Przyjmujemy teraz założenie, że administrator chce skonfigurować nowe konto dla użytkownika nawiązującego połączenie z wielu różnych komputerów. Najłatwiejszym rozwiązaniem jest utworzenie konta zawierającego znak `%` jako nazwę konta w zapytaniu `GRANT`, co pozwala użytkownikowi na nawiązywanie połączenia z dowolnego miejsca:

```
GRANT ALL ON sampdb.* TO 'fred'@'%' IDENTIFIED BY 'cocoa';
```

Celem jest nadanie użytkownikowi `fred` wszystkich uprawnień do bazy danych `sampdb` i umożliwienie mu nawiązania połączenia z dowolnego komputera. Niestety, prawdopodobnym wynikiem będzie możliwość nawiązania połączenia przez użytkownika `fred` z dowolnego komputera *poza* samym serwerem! Przyjmujemy założenie, że komputer serwera nosi nazwę `cobra.example.com`. Jeżeli `fred` będzie próbował nawiązać połączenie zdalnie z komputera `boa.example.com`, próba zakończy się powodzeniem:

```
% mysql -p -u fred -h cobra.example.com sampdb
Enter password: cocoa
mysql>
```

Natomiast jeżeli `fred` spróbuje nawiązać połączenie lokalne z komputera `cobra.example.com`, próba zakończy się niepowodzeniem pomimo podania prawidłowego hasła:

```
% mysql -p -u fred -h localhost sampdb
Enter password: cocoa
ERROR 1045 (28000): Access denied for user 'fred'@'localhost'
(using password: YES)
```

Taki problem występuje, jeżeli tabela `user` zawiera rekord dla jakiegokolwiek domyślnego użytkownika anonimowego bez podanej nazwy użytkownika. Takie rekordy są tworzone w systemach UNIX przez skrypt inicjalizacyjny `mysql_install_db` i istnieją także w preinicjalizowanej tabeli `user` w dystrybucjach dla Windows. (W podrozdziale 12.1, zatytułowanym „Zabezpieczenie nowej instalacji MySQL”, omówiono rekordy początkowo znajdujące się w tabeli `user`). Druga z wymienionych prób nawiązania połączenia kończy się niepowodzeniem, gdy serwer próbuje zweryfikować użytkownika `fred`, a jeden z rekordów użytkownika anonimowego zostaje dopasowany wcześniej niż rekord użytkownika `fred`. Konto użytkownika anonimowego wymaga nawiązania połączenia bez hasła (a nie z hasłem `cocoa`, jak w powyższym przykładzie), stąd niepowodzenie operacji.

Aby zrozumieć, dlaczego tak się dzieje, konieczne jest rozważenie początkowej konfiguracji tabel uprawnień w MySQL oraz sposobu, w jaki serwer używa rekordów tabeli `user` podczas weryfikacji połączeń klientów. Na przykład, w systemie UNIX uruchomienie skryptu `mysql_install_db` w komputerze `cobra.example.com` spowoduje inicjalizację tabel uprawnień, a tabela `user` będzie zawierała rekordy o następujących wartościach kolumn `Host` i `User`:

Host	User
localhost	root
cobra.example.com	root
127.0.0.1	root
::1	root
localhost	
cobra.example.com	

Pierwsze kilka rekordów pozwala użytkownikom na nawiązanie połączenia jako `root` z serwera lokalnego. Z kolei dwa ostatnie rekordy pozwalają na anonimowe nawiązanie połączenia z serwera lokalnego. Kiedy administrator utworzy konto dla użytkownika `fred` za pomocą pokazanego wcześniej zapytania `GRANT`, tabela `user` będzie zawierała następujące rekordy:

Host	User
localhost	root
cobra.example.com	root
127.0.0.1	root
::1	root
localhost	
cobra.example.com	
%	fred

Jednak pokazana kolejność rekordów nie jest kolejnością używaną przez serwer podczas sprawdzania żądań nawiązania połączenia. Serwer sortuje rekordy względem nazw komputerów, następnie użytkowników dla danej nazwy, a wartości dokładniejsze są umieszczane przed bardziej ogólnymi:

Host	User
localhost	root
localhost	
127.0.0.1	root
::1	root
cobra.example.com	root
cobra.example.com	
%	fred

Dwa rekordy z wartością `localhost` w kolumnie `Host` są sortowane razem; rekord zawierający nazwę użytkownika `root` jest pierwszy, ponieważ podana nazwa użytkownika jest uznawana za wartość dokładniejszą niż jej brak. Rekordy z `cobra.example.com` są sortowane w podobny sposób. Co więcej, wszystkie wymienione dotąd rekordy mają zdefiniowane konkretne wartości kolumny `Host` bez użycia żadnych znaków wieloznacznych. Będą więc umieszczone przed rekordem użytkownika `fred`, zawierającym znak wieloznaczny w kolumnie `Host`. W omawianym przykładzie oba rekordy użytkowników anonimowych znajdują się przed rekordem użytkownika `fred`.

Dlatego też, gdy użytkownik `fred` próbuje nawiązać połączenie z komputera lokalnego, jeden z rekordów użytkownika anonimowego zostaje dopasowany wcześniej niż rekord zawierający znak `%` w kolumnie `Host`. Puste hasło w rekordzie użytkownika anonimowego nie pasuje do hasła `cocoa` użytkownika `fred` i próba nawiązania połączenia kończy się niepowodzeniem. Jedną z implikacji tego fenomenu jest to, że użytkownik `fred` może nawiązać połączenie z komputera lokalnego, ale tylko wtedy, gdy *nie poda żadnego hasła*. Niestety, w takim przypadku zostanie uznany za użytkownika anonimowego i nie będzie miał uprawnień nadanych użytkownikowi `fred%`.

Co z tego wynika? Wprawdzie bardzo wygodne jest stosowanie znaków wieloznacznych podczas konfiguracji konta dla użytkownika nawiązującego połączenie z wielu komputerów, ale jednocześnie taki użytkownik może mieć problemy w trakcie nawiązywania połączenia z komputera lokalnego, co wiąże się z istniejącymi w tabeli `user` rekordami użytkowników anonimowych.

Istnieją dwa rozwiązania tego problemu. Pierwsze polega na przygotowaniu drugiego konta dla użytkownika `fred` i wyraźnym podaniu wartości `localhost` jako nazwy komputera:

```
GRANT ALL ON sampdb.* TO 'fred'@'localhost' IDENTIFIED BY 'cocoa';
```

Jeżeli zdecydujesz się na taki krok, serwer będzie sortował tabelę `user` w następujący sposób:

Host	User
localhost	fred
localhost	root
localhost	
127.0.0.1	root
::1	root
cobra.example.com	root
cobra.example.com	
%	fred

Teraz, kiedy użytkownik `fred` nawiąże połączenie z komputera lokalnego, rekord zawierający wartości `localhost` i `fred` zostanie dopasowany przed rekordami użytkowników anonimowych. Z kolei w trakcie nawiązywania połączeń z innych komputerów dopasowany będzie rekord z wartościami `%` i `fred`. Wadą stosowania dwóch kont dla użytkownika `fred` jest to, że jeśli zajdzie potrzeba zmiany jego uprawnień lub hasła, wtedy modyfikacje trzeba będzie wprowadzać dwukrotnie.

Drugie rozwiązanie jest znacznie łatwiejsze i polega na usunięciu z tabeli user kont użytkowników anonimowych:

```
DROP USER '@'localhost';
DROP USER '@'cobra.example.com';
```

Pozostałe rekordy tabeli user będą sortowane w następującej kolejności:

Host	User
localhost	root
127.0.0.1	root
::1	root
cobra.example.com	root
%	fred

Teraz, gdy użytkownik fred spróbuje nawiązać połączenie z komputera lokalnego, próba zakończy się powodzeniem, ponieważ w tabeli user nie ma innych rekordów, które zostałyby wcześniej dopasowane.

Aby ułatwić administrację serwerem MySQL, zalecam usunięcie z tabel uprawnień wszystkich kont użytkowników anonimowych. W mojej opinii te konta nie są zbyt użyteczne, a jedynie mogą spowodować problemy niewarte ich pozostawienia.

Puzzle przedstawione w tym punkcie dotyczą pewnej konkretnej sytuacji, ale stanowią bardziej ogólną lekcję. Jeżeli uprawnienia dla danego konta nie działają zgodnie z oczekiwaniami, sprawdź tabele uprawnień i przekonaj się, czy istnieje inny rekord zawierający wartość Host znacznie bardziej szczegółową niż rekord danego użytkownika. Być może w trakcie próby nawiązania połączenia jest on dopasowany wcześniej niż rekord użytkownika. Jeśli tak, to może być przyczyną problemu. Powinieneś zmodyfikować rekord użytkownika, aby stał się bardziej szczegółowy, lub dodać drugi rekord, dopasowywany w konkretnej sytuacji sprawiającej problem.

13.5. Konfiguracja bezpiecznych połączeń za pomocą SSL

MySQL obsługuje bezpieczne, szyfrowane połączenia za pomocą protokołu SSL. Domyślnie instalacja zawierająca obsługę SSL pozwala klientowi na opcjonalne żądanie nawiązania bezpiecznego połączenia. Administrator za pomocą zapytania GRANT może również wskazać, że dany użytkownik *musi* nawiązywać bezpieczne połączenie.

Korzyści z użycia protokołu SSL występują przede wszystkim w połączeniach nawiązywanych ze zdalnych serwerów, ponieważ wówczas cała komunikacja sieciowa jest szyfrowana, co utrudnia jej przechwytywanie. Nie ma zbyt wielkiego sensu stosowanie SSL dla połączeń lokalnych nawiązywanych za pomocą pliku gniazda systemu UNIX, nazwanego potoku, pamięci współdzielonej bądź sieciowego interfejsu loopback o adresie IP 127.0.0.1 lub ::1. Ruch sieciowy w tego rodzaju połączeniach nigdy nie opuszcza komputera lokalnego.

Aby nawiązać połączenie szyfrowane SSL między serwerem i programem klienta, należy zastosować przedstawioną poniżej ogólną procedurę:

1. Upewnij się, że serwer i program klienta zostały skompilowane wraz z obsługą SSL.
2. Uruchom serwer wraz z opcjami wskazującymi miejsce położenia plików jego certyfikatu i kluczy; wspomniane pliki są niezbędne do nawiązywania bezpiecznych połączeń.
3. Aby nawiązać bezpieczne połączenie z programem klienta, uruchom go wraz z opcjami wskazującymi miejsce położenia plików jego certyfikatu i kluczy.

Poniżej znacznie dokładniej omówiono proces nawiązywania bezpiecznego połączenia.

Dystrybucja MySQL musi być skompilowana wraz z wbudowaną obsługą SSL.

Do wyboru masz pobranie dystrybucji binarnej wraz z obsługą SSL lub też samodzielną kompilację MySQL ze źródeł. Dystrybucje binarne dla większości platform zapewniają obsługę SSL. Jeżeli samodzielnie kompilujesz MySQL, upewnij się, że podałeś niezbędne opcje w trakcie konfiguracji (na przykład, `-DWITH_SSL=bundled` w celu użycia yaSSL lub `-DWITH_SSL=system` w celu użycia OpenSSL). Obsługę SSL przez serwer możesz potwierdzić, wykonując poniższe zapytanie:

```
mysql> SHOW VARIABLES LIKE 'have_ssl';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_ssl      | DISABLED |
+-----+-----+
```

Jeżeli wyświetlona została wartość `DISABLED` lub `YES`, obsługa SSL jest dostępna. Wartość `DISABLED` oznacza dostępną, choć jeszcze niewłączoną obsługę SSL. To w porządku, pliki niezbędne do włączenia SSL zostaną wkrótce omówione.

Kiedy masz instalację MySQL z wbudowaną obsługą SSL, serwer i klienci mogą prowadzić bezpieczną komunikację. Obie komunikujące się ze sobą strony używają trzech plików do nawiązania bezpiecznej komunikacji:

- Certyfikat wydany przez centrum (ang. *Certificate Authority*, CA). Wspomniane centrum to zaufana organizacja zajmująca się wydawaniem certyfikatów. Wydawane przez nią certyfikaty służą do autoryzacji klienta i serwera. Najczęściej certyfikat jest kupowany w tego rodzaju centrum, choć można go również wygenerować samodzielnie.
- Plik certyfikatu uwierzytelniający jedną stronę połączenia w drugiej. To jest plik klucza publicznego.
- Plik używany do szyfrowania i deszyfrowania komunikacji. To jest plik klucza prywatnego.

Jako pierwsze muszą być zainstalowane pliki certyfikatu i kluczy serwera. Jeżeli nie posiadasz tego rodzaju plików, znajdziesz je w katalogu `ssl` dystrybucji `sampdb`:

- *ca-cert.pem* — plik certyfikatu wydanego przez centrum;
- *server-cert.pem* — certyfikat serwera;
- *server-key.pem* — klucz publiczny serwera.

Wszystkie trzy wymienione pliki skopiuj do katalogu danych serwera oraz dodaj kilka wierszy do grupy `[mysqld]` w pliku opcji odczytywanym przez serwer w trakcie jego uruchamiania, na przykład `/etc/my.cnf` w systemach UNIX lub `C:\my.ini` w Windows. Wspomniane opcje powinny zawierać ścieżki dostępu do plików certyfikatu i kluczy. Na przykład, jeśli katalogiem danych jest `/usr/local/mysql/data`, opcje należy wymienić w następujący sposób:

```
[mysqld]
ssl-ca=/usr/local/mysql/data/ca-cert.pem
ssl-cert=/usr/local/mysql/data/server-cert.pem
ssl-key=/usr/local/mysql/data/server-key.pem
```

Oczywiście, pliki certyfikatu i kluczy można umieścić w innym miejscu, ale powinno być to położenie, do którego dostęp ma wyłącznie serwer. Po instalacji plików SSL i modyfikacji pliku opcji należy ponownie uruchomić serwer.

Na tym etapie masz serwer pozwalający na nawiązywanie szyfrowanych połączeń, a wartością zmiennej `have_ssl` powinno być `YES`. Jednak programy klienckie nadal mogą nawiązywać z serwerem jedynie nieszyfrowane połączenia. Aby umożliwić klientom nawiązywanie szyfrowanych połączeń, konieczne jest wskazanie plików certyfikatu i kluczy także po stronie klienta. W katalogu `ssl` dystrybucji `mysql` znajdziesz odpowiednie pliki dla klienta. Możesz użyć tego samego certyfikatu (*ca-cert.pem*). Natomiast certyfikat klienta i pliki kluczy noszą nazwy *client-cert.pem* i *client-key.pem*. Skopiuj wymienione trzy pliki do wybranego katalogu w ramach swojego konta użytkownika. Następnie wskaż ich położenie programowi klienta przez dodanie odpowiednich wierszy do pliku opcji odczytywanego przez klienta w chwili jego uruchamiania, na przykład pliku `.my.cnf` w katalogu domowym użytkownika systemu UNIX.

Przyjmujemy założenie, że chcę nawiązywać szyfrowane połączenia za pomocą programu klienckiego `mysql`. Katalog z plikami SSL kopiuję więc do katalogu domowego `/home/paul`, a następnie umieszczam poniższe wiersze w pliku `.my.cnf`:

```
[mysql]
ssl-ca=/home/paul/ca-cert.pem
ssl-cert=/home/paul/client-cert.pem
ssl-key=/home/paul/client-key.pem
```

W podobny sposób możesz skonfigurować własne konto użytkownika. Dobrym rozwiązaniem jest upewnienie się, że pliki certyfikatu i kluczy są dostępne jedynie dla Ciebie. Po przeprowadzeniu modyfikacji pliku `.my.cnf` wskaż położenie plików SSL, uruchom klienta `mysql` i wydaj polecenie `\s` lub `status`. W danych wyjściowych wiersz SSL powinien zawierać informacje, że połączenie jest szyfrowane:

```
mysql> status;
-----
mysql Ver 14.14 Distrib 5.5.21, for Linux (i686)
Connection id:          5
```

```
Current database:
Current user:      sampadm@localhost
SSL:              Cipher in use is DHE-RSA-AES256-SHA
...
```

Możesz również wykonać poniższe zapytanie i sprawdzić wartości, jakie mają zmienne stanu serwera powiązane z SSL:

```
SHOW STATUS LIKE 'Ssl%';
```

Istnienie powiązanych z SSL opcji w grupie [mysql]d pliku konfiguracyjnego powoduje, że klient mysql domyślnie nawiązuje szyfrowane połączenia SSL. Jeżeli wspomniane wiersze umieścisz w komentarzu lub po prostu je usuniesz, wtedy mysql nawiązuje zwykłe, nieszyfrowane połączenia. Opcje dotyczące SSL można również zignorować przez wywołanie klienta mysql w poniższy sposób:

```
% mysql --skip-ssl
```

Opcje w grupie [mysql]d można skopiować do innych grup przeznaczonych dla konkretnych programów i tym samym umożliwić im nawiązywanie połączeń SSL. Powstaje pytanie, czy te opcje należy umieścić w ogólnej grupie [client]. Prawdopodobnie nie. Ich obecność spowoduje niepowodzenie nawiązania połączenia w przypadku programu klienta, który nie zawiera obsługi SSL. (Jeżeli mimo wszystko będziesz chciał opcje związane z SSL umieścić w grupie [client], użyj prefiksu loose-, aby programy bez obsługi SSL po prostu je ignorowały).

Alternatywnym rozwiązaniem dla umieszczenia opcji SSL w pliku opcji jest ich podanie w wierszu poleceń. Na przykład, w katalogu zawierającym pliki powiązane z SSL program klienta mysql możesz wywołać w następujący sposób (całe polecenie musi znajdować się w jednym wierszu):

```
% mysql --ssl-ca=ca-cert.pem --ssl-cert=client-cert.pem
--ssl-key=client-key.pem
```

Jednak wydawanie powyższego polecenia za każdym razem jest męczące.

Pliki certyfikatu i kluczy dostarczone wraz z dystrybucją sampdb są wystarczające do umożliwienia nawiązywania szyfrowanych połączeń. Ponieważ są one dostępne publicznie (dla każdego, kto pobierze dystrybucję), to połączenia nawiązywane za ich pomocą nie mogą być uznane za w pełni bezpieczne. Po wykorzystaniu tych plików do upewnienia się o prawidłowym działaniu SSL powinieneś je zastąpić wygenerowanymi samodzielnie. Instrukcje wskazujące sposób wygenerowania własnych plików certyfikatu i kluczy znajdziesz w pliku *ssl/README.txt* w dystrybucji sampdb. Możesz również rozważyć zakup certyfikatu komercyjnego.

Jak dotąd dowiedziałeś się, jak dowolny użytkownik może opcjonalnie nawiązywać szyfrowane połączenia za pomocą protokołu SSL. Konto użytkownika można też skonfigurować w sposób zabraniający użycia SSL lub je nakazujący. Aby zmodyfikować istniejące konto użytkownika i zdefiniować wymóg stosowania połączeń SSL, należy użyć zapytania GRANT USAGE wraz z klauzulą REQUIRE zawierającą właściwości, które muszą być spełnione przez połączenia:

```
GRANT USAGE ON *.* TO 'użytkownik' REQUIRE wymagane_opcje;
```

Zapytanie `GRANT USAGE ON *.*` pozostawia uprawnienia użytkownika bez zmian i modyfikuje jedynie atrybuty konta związane z SSL.

Przyjmujemy założenie, że istnieje konto dla użytkownika `laura`, który nawiązuje połączenie z komputera `viper.example.com`. Aby wymusić nawiązywanie jedynie połączeń szyfrowanych, należy wykonać poniższe zapytanie:

```
GRANT USAGE ON *.* TO 'laura'@'viper.example.com' REQUIRE SSL;
```

W celu zapewnienia większego bezpieczeństwa należy użyć `REQUIRE X509`. W takim przypadku użytkownik `laura` musi podczas nawiązywania połączenia przedstawić ważny certyfikat X509. (To będzie plik wymieniony przez opcję `--ssl-cert`). Dopóki certyfikat pozostanie ważny, zawartość pliku nie ma znaczenia. Jeśli ma być wymagana określona zawartość certyfikatu, użyj połączenia wartości `CIPHER`, `ISSUER` i `SUBJECT` w klauzuli `REQUIRE`. Wartość `CIPHER` wskazuje metodę szyfrowania, jaka ma być stosowana przez połączenie. Z kolei wartości `ISSUER` i `SUBJECT` wskazują, że certyfikat klienta musi być wydany przez określone źródło i dla wskazanego odbiory. Wymienione klauzule powodują zawężenie zasięgu certyfikatu, który musi zawierać wskazaną treść. Poniższe zapytanie `GRANT` wymaga stosowania certyfikatu wydanego przez określonego wydawcę i nakazuje użycie szyfrowania `EXP1024-RC4-SHA`:

```
GRANT USAGE ON *.* TO 'laura'@'viper.example.com'
  REQUIRE ISSUER '/C=US/ST=WI/L=Madison/O=sampdb/OU=CA/CN=sampdb'
  CIPHER 'EXP1024-RC4-SHA';
```

Jeżeli konto użytkownika aktualnie ma zdefiniowany wymóg stosowania połączeń SSL i chcesz znieść to wymaganie, wykonaj zapytanie `GRANT USAGE` w połączeniu z klauzulą `REQUIRE NONE`:

```
GRANT USAGE ON *.* TO 'laura'@'viper.example.com' REQUIRE NONE;
```

Więcej informacji na temat klauzuli `REQUIRE` znajdziesz w podpunkcie 13.2.2.3, zatytułowanym „Wymaganie nawiązania połączenia za pomocą protokołu SSL”.

Jeżeli używasz API MySQL dla języków takich jak Perl lub PHP, wtedy możliwość nawiązywania połączeń SSL nie zależy jedynie od API języka, ale także od połączonej z nim biblioteki klienta MySQL. Wspomniana biblioteka klienta musi być skompilowana wraz z obsługą SSL, aby mogła obsługiwać połączenia SSL z serwerem. Ponadto, API języka musi być w możliwie najnowszej wersji, potrafiącej wykorzystać oferowane przez bibliotekę klienta możliwości w zakresie obsługi SSL. Na przykład, rozszerzenie `mysqlnd` dla PHP zapewnia obsługę połączeń SSL, natomiast starsze rozszerzenie `mysql` nie oferuje już takiej możliwości.

Obsługa bazy danych, kopie zapasowe i replikacja

W idealnej sytuacji serwer MySQL będzie działał płynnie i bezproblemowo od chwili jego instalacji. Jednak wcześniej lub później problemy występują, na przykład na skutek awarii zasilania lub komponentu sprzętowego bądź też nieprawidłowego zamknięcia serwera (po wystąpieniu awarii serwera lub wymuszenia jego zamknięcia poleceniem `kill -9`). Tego rodzaju zdarzenia, z których większość jest poza Twoją kontrolą, mogą doprowadzić do uszkodzenia bazy danych, najczęściej na skutek niedokończonych operacji zapisu w trakcie uaktualniania tabel.

W tym rozdziale dowiesz się, jak zminimalizować ryzyko i być przygotowanym na ewentualne katastrofy. Zaprezentowane tutaj techniki obejmują między innymi tworzenie kopii zapasowej bazy danych, przeprowadzanie operacji sprawdzania i naprawy tabel, a także sposoby odzyskiwania danych. W rozdziale przedstawiono procedury kopiowania bazy danych w celu jej przeniesienia z jednego serwera do innego, ponieważ bardzo często to są techniki podobne do stosowanych podczas tworzenia kopii zapasowych. Kolejną techniką „kopiowania” omówioną w rozdziale będzie replikacja. Pozwala ona na inicjalizację serwera podległego powielającego dane znajdujące się w serwerze głównym. Konfiguracja replikacji pozwala, aby zmiany wprowadzone w serwerze głównym były natychmiast przekazywane do serwerów podległych. Dlatego też serwery podległe „nieustannie kopiuja” dane znajdujące się w serwerze głównym. Replikacja może być użyteczna również w innych celach. Na przykład, obciążenie można rozłożyć między kilka serwerów, odciążając tym samym serwer główny. Ponadto, serwer podległy może być znacznie łatwiej wstrzymany lub zatrzymany niż serwer główny w celu wykonania kopii zapasowej.

14.1. Zasady obsługi profilaktycznej

W tym podpunkcie zostaną przedstawione ogólne zasady obsługi profilaktycznej.

W kolejnych punktach podrozdziału poznasz szczegóły dotyczące implementacji.

Aby przygotować się na ewentualne problemy z bazą danych, należy podjąć wymienione poniżej kroki:

- Włączenie oferowanych przez serwer MySQL możliwości w zakresie automatycznej naprawy.
- Konfiguracja harmonogramu przeprowadzania obsługi profilaktycznej i regularne sprawdzanie tabel. Rutynowe procedury sprawdzania tabel mogą pomóc w wykryciu i usunięciu drobnych błędów, nim staną się naprawdę poważne.
- Konfiguracja harmonogramu wykonywania kopii zapasowej. Kiedy stanie się najgorsze i staniesz przed obliczem poważnej katastrofy, do przeprowadzenia operacji odzyskania danych będziesz potrzebował kopii zapasowej. Włącz także binarny dziennik zdarzeń, aby zarejestrowane zostały zmiany wprowadzone od chwili wykonania ostatniej kopii zapasowej. (Zapoznaj się z punktem 12.8.4, zatytułowanym „Binarny dziennik zdarzeń”). Binarna rejestracja danych zapewnia duże korzyści dla tworzenia kopii zapasowej i mechanizmu replikacji, a przy tym jedynie w minimalnym stopniu wpływa na wydajność działania serwera. Nie ma więc żadnych przeciwwskazań do włączenia binarnego dziennika zdarzeń.

Jeżeli pomimo Twoich wysiłków dojdzie do uszkodzenia lub utraty danych, do dyspozycji pozostają jeszcze następujące rozwiązania:

- Sprawdź tabele i usuń wszelkie znalezione błędy, o ile będzie to możliwe. Mniejsze uszkodzenia z reguły można naprawić za pomocą oferowanych przez MySQL możliwości w zakresie naprawy tabel.
- Jeżeli operacja sprawdzenia i naprawy tabel nie pomoże, przystąp do przywrócenia danych z posiadanej kopii zapasowej i binarnego dziennika zdarzeń. Rozpocznij od użycia kopii zapasowej i tym samym przywróć bazę danych do stanu, w jakim znajdowała się w chwili tworzenia tej kopii zapasowej. Następnie z binarnego dziennika zdarzeń przenieś uaktualnienia, które zostały wprowadzone już po utworzeniu kopii zapasowej, ale przed wystąpieniem awarii. W ten sposób w pełni przywrócisz tabele.

Powyższą operację możesz przeprowadzić za pomocą różnych narzędzi — wewnętrznych możliwości oferowanych przez sam serwer MySQL oraz innych narzędzi dostarczanych wraz z dystrybucją MySQL:

- Podczas uruchamiania serwera transakcyjne silniki bazy danych mogą przeprowadzić operacje automatycznej naprawy. Funkcję automatycznej naprawy tabel można włączyć także dla silnika MyISAM. Wspomniane możliwości są użyteczne, gdy po awarii nastąpi ponowne uruchomienie.
- Użyj programu `mysql dump` do utworzenia kopii zapasowych baz danych, które później będzie można wykorzystać w procesie przywracania danych.

- Skonfiguruj serwer do przeprowadzania na żądanie operacji obsługi oraz użycia zapytań SQL takich jak `CHECK TABLE` i `REPAIR TABLE`. Interfejsem wiersza poleceń dla wymienionych zapytań jest program `mysqlcheck`. Narzędzie `mysamchk` również może sprawdzać tabele pod kątem ewentualnych problemów i na tej podstawie podejmować odpowiednie akcje.

Pewne programy, takie jak `mysqlcheck` i `mysqldump`, działają we współpracy z serwerem. Nawiązują połączenie z serwerem jako klienci, a następnie wykonują zapytania SQL wskazujące serwerowi rodzaj operacji do przeprowadzenia. Z kolei `mysamchk` to niezależny, samodzielny program operujący bezpośrednio na plikach przedstawiających tabele. Ponieważ uruchomiony serwer również ma dostęp do wspomnianych plików, program `mysamchk` stanowi konkurencję dla serwera MySQL. Oznacza to konieczność podjęcia pewnych kroków w celu uniemożliwienia serwerowi i programowi `mysamchk` przeszkadzania sobie nawzajem. Na przykład, jeśli naprawiasz tabele za pomocą programu `mysamchk`, konieczne jest uniemożliwienie serwerowi przeprowadzania operacji zapisu w naprawianych tabelach. W przeciwnym razie można doprowadzić do jeszcze większych problemów niż te, które próbujesz usunąć!

Konieczność współpracy z serwerem występuje także w przypadku wykonywania wielu zadań administracyjnych przedstawionych w tym rozdziale, począwszy od tworzenia kopii zapasowych aż po naprawę tabel. Dlatego też w kolejnym podrozdziale dowiesz się, jak współpracować z serwerem, gdy zachodzi potrzeba. W następnych podrozdziałach dowiesz się, jak być przygotowanym na problemy, jak tworzyć kopie zapasowe oraz jak stosować techniki naprawy i odzyskiwania danych, gdy zajdzie potrzeba.

W systemach UNIX operacje wymagające bezpośredniej pracy z plikami tabel lub innymi plikami znajdującymi się w katalogu danych MySQL powinny być przeprowadzane po zalogowaniu się jako administrator MySQL, ponieważ tylko wtedy będziesz miał do nich uprawnienia dostępu. W tej książce przyjęto założenie, że nazwa konta logowania to `mysql`. Istnieje możliwość uzyskania dostępu do tych plików jako użytkownik `root`, ale w takim przypadku po zakończeniu pracy z nimi trzeba się upewnić, że będą miały takie same uprawnienia i właściciela jak w chwili rozpoczęcia pracy.

Pełną listę opcji obsługiwaną przez zapytania SQL i programy omawiane w rozdziale znajdziesz w dodatkach: E, „Przewodnik po składni SQL”, i F, „Przewodnik po programie MySQL”.

14.2. Obsługa bazy danych w działającym serwerze

Pewne operacje związane z obsługą są przeprowadzane przez nawiązanie połączenia z serwerem i następnie wskazanie konkretnego zadania do wykonania. Aby przeprowadzić operację sprawdzenia spójności lub naprawy tabeli MyISAM, jednym z rozwiązań jest wykonanie zapytania `CHECK TABLE` lub `REPAIR TABLE` (bądź też uruchomienie programu

mysql check) i pozwolenie serwerowi na przeprowadzenie operacji. W takim przypadku serwer uzyska dostęp do plików *.frm*, *.myd* i *.myi* przedstawiających tabelę. To jest najlepsze z możliwych rozwiązań, które warto wybierać, jeśli istnieje taka możliwość. Gdy serwer przeprowadza żadaną operację, zajmuje się obsługą wszelkich kwestii związanych z koordynacją dostępu do tabeli, więc Ty nie musisz się tym przejmować.

Z kolei inne operacje są przeprowadzane przez program zewnętrzny dla serwera. W takim przypadku *musisz* rozważyć kwestie związane z koordynacją dostępu do tabeli. Na przykład, innym sposobem sprawdzenia lub naprawy tabeli MyISAM jest wywołanie narzędzia *myisamchk*, które pliki tabeli otwiera bezpośrednio i bez udziału serwera. Kiedy wymienione narzędzie będzie pracowało z plikami tabeli, trzeba uniemożliwić serwerowi wprowadzanie jakichkolwiek zmian w tabeli. Jeżeli tego nie zapewnisz, to dojdzie do rywalizacji o dostęp do tabeli, która może ulec uszkodzeniu i stać się bezużyteczna. Oczywiście, niepożądane jest, aby serwer i program *myisamchk* jednocześnie przeprowadzały operacje zapisu w tabeli, ale nawet odczyt danych z tabeli, gdy drugi program przeprowadza jej naprawę, też nie jest pożądany. Program odczytujący dane może być zaskoczony nieustanną zmianą zawartości tabeli.

Konieczność uniemożliwienia serwerowi dostępu do tabel występuje także w innych okolicznościach:

- Kompresja tabeli MyISAM za pomocą narzędzia *myisampack*.
- Przeniesienie pliku danych lub indeksu tabeli MyISAM.
- Przeniesienie bazy danych.
- Tworzenie kopii zapasowej za pomocą techniki opierającej się na kopiowaniu plików tabeli. Aby zapewnić spójność plików kopii zapasowej, konieczne jest uniemożliwienie serwerowi zmiany tabel w trakcie trwania procedury tworzenia kopii zapasowej.
- Stosowanie metod odzyskiwania danych polegających na zastępowaniu uszkodzonych tabel dobrymi kopiami zapasowymi. W trakcie operacji zastępowania tabeli serwer nie powinien mieć do niej dostępu.

Najefektywniejszym sposobem uniemożliwienia serwerowi dostępu do tabeli jest jego zamknięcie. Oczywiście jest, że jeśli serwer będzie zamknięty, to nie ma dostępu do tabel, z którymi pracujesz. Administratorzy, co zrozumiałe, niechętnie odnoszą się do całkowitego zamknięcia serwera, ponieważ wtedy niedostępne są wszystkie bazy danych i tabele, a nie tylko te, które chcesz sprawdzić lub naprawić.

Aby uniknąć zatrzymania serwera i jednocześnie problemów związanych z wzajemnym zakłócaniem sobie działania serwera i zewnętrznego programu przeprowadzającego w nim operacje, konieczne jest zapewnienie koordynacji z serwerem przez nakładanie blokad. Serwer obsługuje dwa rodzaje blokad:

- Serwer używa wewnętrznego blokowania w celu rozdzielenia żądań pochodzących od różnych klientów. Na przykład, serwer nie dopuszcza, aby zapytanie *SELECT*

wykonywane przez jednego klienta zostało zakłócone przez zapytanie UPDATE wykonywane przez innego klienta. Wewnętrzny mechanizm nakładania blokad można wykorzystać także do uniemożliwienia klientom uzyskania dostępu do tabel podczas pracy z tymi tabelami za pomocą programu zewnętrznego dla serwera.

- Serwer może stosować zewnętrzne nakładanie blokad w celu uniemożliwienia innym programom modyfikacji plików, gdy serwer z nich korzysta. Rozwiązanie opiera się na możliwościach w zakresie nakładania blokad oferowanych przez system operacyjny na poziomie systemu plików. Powodem stosowania przez serwer zewnętrznego nakładania blokad jest współpraca z programami takimi jak `myisamchk` w trakcie przeprowadzania przez nie operacji sprawdzania tabel. Jednak zewnętrzne nakładanie blokad nie działa niezawodnie w pewnych systemach. Inne ograniczenie wiąże się z faktem, że zewnętrzne nakładanie blokad jest użyteczne jedynie w przypadku operacji wymagających odczytu plików tabel, czyli na przykład ich sprawdzania. Nie może być natomiast stosowane w przypadku operacji odczytu i zapisu, czyli na przykład w trakcie naprawy tabel. Zewnętrzne nakładanie blokad opiera się na nakładaniu blokad na poszczególne pliki, ale przeprowadzane przez narzędzie `myisamchk` operacje naprawy tworzą kopie plików, a następnie zastępują nimi oryginalne pliki tabeli. Serwer nic nie wie o nowych plikach, co czyni bezużytecznymi wszelkie próby zapewnienia koordynacji dostępu na podstawie blokad nakładanych na pliki.

Poniżej znajdziesz omówienie jedynie wewnętrznego mechanizmu nakładania blokad w celu zapewnienia koordynacji z serwerem w zakresie dostępu do tabel. Ponieważ zewnętrzne nakładanie jest problematyczne, jego stosowanie jest odradzane i nie będziemy go tutaj omawiali.

14.2.1. Blokowanie poszczególnych tabel w celu uzyskania dostępu tylko do odczytu lub odczytu i zapisu

Aby skorzystać z oferowanego przez serwer wewnętrznego mechanizmu nakładania blokad i uniemożliwić serwerowi dostęp do tabel w trakcie pracy z nimi, nawiąż połączenie z serwerem za pomocą klienta `mysql` i wykonaj zapytanie `LOCK TABLE` dla tabeli, która ma być przedmiotem operacji sprawdzenia lub naprawy. Następnie, gdy klient `mysql` pozostaje bezczynny (to znaczy nie wykonuje żadnych operacji poza nałożeniem blokady), przeprowadź wszystkie niezbędne operacje na plikach tabeli. Po zakończeniu pracy powróć do sesji `mysql` i zwolnij blokadę, tym samym ponownie dając serwerowi możliwość używania danej tabeli.

Uwaga

Omówione tutaj techniki wewnętrznego nakładania blokad na poszczególne tabele mają zastosowanie tylko dla silników bazy danych takich jak MyISAM, w których tabele są przedstawiane za pomocą unikalnych plików. Natomiast *nie mają zastosowania* dla silników takich jak InnoDB, ponieważ w danym pliku są przechowywane informacje o wielu tabelach. Na przykład, silnik bazy danych InnoDB domyślnie przechowuje wszystkie tabele razem w plikach tworzących systemową przestrzeń tabel. (Nawet jeśli silnik InnoDB zostanie skonfigurowany do przechowywania poszczególnych tabel w oddzielnych przestrzeniach tabel, to pewne informacje o każdej tabeli są przechowywane w katalogu danych InnoDB, który znajduje się w systemowej przestrzeni tabel).

Używany mechanizm nakładania blokad zależy od rodzaju oczekiwanej blokady: dostęp tylko do odczytu lub dostęp do odczytu i zapisu plików tabel. W celu przeprowadzenia operacji takich jak sprawdzenie tabeli lub skopiowanie jej plików wystarczający jest dostęp tylko do odczytu. Natomiast operacje modyfikujące pliki, na przykład naprawa tabeli lub zastąpienie uszkodzonych plików dobrymi, wymagają dostępu do odczytu i zapisu.

Mechanizmy nakładania blokad używają zapytań `LOCK TABLE` i `UNLOCK TABLE` w celu odpowiednio nałożenia i zwolnienia blokady. Ponadto, wykonując zapytanie `FLUSH TABLE`, nakazujące serwerowi zapisanie na dysku wszelkich buforowanych zmian i wskazujące konieczność ponownego otworzenia tabeli, gdy kolejnym razem będzie chciał uzyskać do niej dostęp. Przedstawione tutaj przykłady zawierają zapytania `FLUSH TABLE`, pobierające argument w postaci nazwy tabeli i dotyczące tylko wymienionej tabeli.

Wszystkie zapytania `LOCK`, `FLUSH` i `UNLOCK` *muszą* być wykonane w ramach pojedynczej sesji z serwerem. Na przykład, nie możesz nawiązać połączenia z serwerem za pomocą programu `mysql`, nałożyć blokady na tabelę, a następnie zakończyć działania programu z zamiarem późniejszego nawiązania ponownego połączenia i zwolnienia blokady. Takie rozwiązanie nie działa, ponieważ po zakończeniu działania klienta `mysql` serwer automatycznie zwalnia blokady. Serwer po prostu uznaje, że ponownie może korzystać z tabel. Dla Ciebie oznacza to, że praca z plikami tabel nie jest już bezpieczna.

Łatwym sposobem przeprowadzenia procedury nakładania blokady jest pozostawienie otwartych dwóch okien terminala. W jednym oknie masz działający program klienta `mysql`, natomiast w drugim przeprowadzasz niezbędne operacje na plikach zablokowanej tabeli. W środowisku tekstowym powłoka decyduje o możliwości wstrzymania i wznowienia procesów `mysql` podczas pracy z tabelą.

14.2.1.1. Nałożenie blokady pozwalającej tylko na odczyt tabeli

Blokada pozwalająca tylko na odczyt tabeli jest odpowiednia w przypadku operacji jedynie odczytujących pliki tabeli, jak kopiowanie plików lub sprawdzanie spójności tabeli. Dla wymienionych operacji blokada pozwalająca tylko na odczyt tabeli jest w zupełności wystarczająca. Serwer nie pozwala innym klientom na modyfikację tabeli, choć nadal mogą odczytywać jej zawartość. Tego rodzaju blokada *nie* nadaje się w przypadku operacji wymagających wprowadzenia zmian w tabeli.

1. W oknie A uruchom klienta `mysql` i za pomocą poniższych zapytań nałóż blokadę zezwalającą tylko na odczyt tabeli oraz opróżnij jej bufor:

```
% mysql nazwa_bazy_danych
mysql> LOCK TABLE nazwa_tabeli READ;
mysql> FLUSH TABLE nazwa_tabeli;
```

Nałożenie blokady pozwalającej jedynie na odczyt tabeli uniemożliwia innym klientom przeprowadzanie operacji zapisu oraz jakichkolwiek modyfikacji w tabeli. Zapytanie `FLUSH` nakazuje serwerowi zamknięcie plików tabel i zapisanie na dysku wszelkich zmian, które nadal mogą być buforowane w pamięci.

2. Kiedy klient `mysql` pozostaje bezczynny, przejdź do okna B, w którym będziesz mógł pracować z plikami. Na przykład, jeżeli `nazwa_tabeli` to tabela `MyISAM`, to jej kopię zapasową możesz wykonać przez skopiowanie plików przedstawiających tabelę. Jeśli katalogiem bieżącym jest katalog danych `MySQL`, skopiowanie plików do katalogu `/var/backup` następuje po wydaniu poniższego polecenia:

```
% cp nazwa_tabeli.* /var/backup
```

To jest tylko przykład. Wydawane polecenia będą zależały od konkretnej operacji, którą chcesz przeprowadzić.

3. Po zakończeniu pracy z tabelą powróć do okna A zawierającego sesję `mysql`, a następnie zwolnij blokadę tabeli, wykonując poniższe zapytanie:

```
mysql> UNLOCK TABLE;
```

W trakcie pracy może się okazać, że konieczne będzie przeprowadzenie dalszych operacji. Na przykład, procedura sprawdzania tabeli zasygnalizuje konieczność przywrócenia danych lub przeprowadzenia naprawy. Tego rodzaju operacje wymagają dostępu odczytu i zapisu, który można sobie bezpiecznie zapewnić za pomocą procedury omówionej w kolejnym podpunkcie.

14.2.1.2. Nałożenie blokady pozwalającej na odczyt i zapis tabeli

Omówiona tutaj blokada pozwalająca na odczyt i zapis tabeli jest odpowiednia podczas przeprowadzania operacji takich jak zamiana tabeli lub naprawa obejmująca modyfikację plików tabeli. Tego rodzaju operacje wymagają nałożenia blokady zapisu całkowicie uniemożliwiającej serwerowi dostęp do tabeli w czasie, gdy będziesz z nią pracował.

Procedura nakładania blokady pozwalającej na odczyt i zapis tabeli jest podobna do przedstawionej wcześniej procedury zezwalającej tylko na odczyt tabeli, ale z dwiema różnicami. Po pierwsze, konieczne jest nałożenie blokady zapisu, a nie odczytu. Ponieważ tabela będzie modyfikowana, serwer w ogóle nie może mieć do niej dostępu, nawet w celu odczytu. Po drugie, po zakończeniu pracy z tabelą konieczne jest wykonanie zapytania `FLUSH TABLE`. Jeżeli przeprowadzona operacja ponownie utworzy lub naprawi plik indeksu, serwer w ogóle nie zauważy nowego indeksu, o ile ponownie nie zostanie opróżniony bufor tabeli.

Aby nałożyć blokadę pozwalającą na odczyt i zapis tabeli, wykonać poniższą procedurę:

1. W oknie A uruchom klienta `mysql` i za pomocą poniższych zapytań nałóż blokadę zapisu na tabelę oraz opróżnij jej bufor:

```
% mysql nazwa_bazy_danych
mysql> LOCK TABLE nazwa_tabeli WRITE;
mysql> FLUSH TABLE nazwa_tabeli;
```

2. Kiedy klient `mysql` pozostaje beczynny, przejdź do okna B, w którym będziesz mógł pracować z plikami. Na przykład, jeżeli `nazwa_tabeli` to tabela MyISAM, to możesz ją przywrócić z kopii zapasowej utworzonej wcześniej przez skopiowanie plików przedstawiających tabelę. Jeśli katalogiem bieżącym jest katalog danych MySQL, skopiowanie plików z katalogu kopii zapasowej `/var/backup` następuje po wydaniu poniższego polecenia:

```
% cp /var/backup/nazwa_tabeli.* .
```

To jest tylko przykład. Wydawane polecenia będą zależały od konkretnej operacji, którą chcesz przeprowadzić.

3. Po zakończeniu pracy z tabelą powróć do okna A zawierającego sesję `mysql`, ponownie opróżnij bufor tabeli, a następnie zwolnij blokadę tabeli, wykonując poniższe zapytanie:

```
mysql> FLUSH TABLE nazwa_tabeli;
mysql> UNLOCK TABLE;
```

14.2.2. Nałożenie na wszystkie bazy danych blokady pozwalającej jedynie na ich odczyt

Wygodnym sposobem uniemożliwienia klientom wprowadzania jakichkolwiek zmian we wszystkich tabelach jest nałożenie blokady jednocześnie na wszystkie tabele we wszystkich bazach danych. W tym celu należy wykonać poniższe zapytania:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SET GLOBAL read_only = ON;
```

Zapytanie `FLUSH` nakłada globalną blokadę odczytu, natomiast `SET` wstrzymuje ją do chwili, aż wszyscy klienci zwolnią nałożone przez siebie blokady i zakończą przeprowadzane transakcje. Po wykonaniu powyższych zapytań można bezpiecznie przystąpić do pracy, wiedząc, że inne klienty nie będą zakłócać przebiegu operacji.

Aby ponownie zezwolić na wprowadzanie zmian i zwolnić blokadę, należy wykonać poniższe zapytania:

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

Kiedy tabele będą zablokowane w przedstawiony powyżej sposób, inne klienty mogą odczytywać ich zawartość, ale nie mają możliwości wprowadzania jakichkolwiek zmian. To jest dobre rozwiązanie, pozwalające na przeprowadzenie operacji takich jak wykonanie kopii zapasowej za pomocą programu `mysql dump`. Z drugiej strony, to rozwiązanie nieprzyjazne

dla pozostałych klientów, które muszą przeprowadzać uaktualnienia. Dlatego też tego rodzaju blokada powinna trwać jak najkrócej. Będzie również niewystarczająca dla operacji takich jak utworzenie binarnej kopii zapasowej wszystkich tabel zarządzanych przez transakcyjny silnik bazy danych, na przykład InnoDB. Powód tego jest prosty — silnik bazy danych może mieć niezapisane na dysku transakcje, które jedynie częściowo zostały zapisane w plikach dzienników zdarzeń. Tego rodzaju operacje wymagają zatrzymania serwera oraz upewnienia się o zapisaniu wszystkich danych i zamknięciu wszystkich plików.

14.3. Ogólne działania profilaktyczne

W tym podrozdziale zostaną przedstawione pewne ogólne strategie pozwalające na zachowanie spójności baz danych:

- Włączenie oferowanych przez serwer MySQL możliwości w zakresie automatycznej naprawy.
- Konfiguracja harmonogramu przeprowadzania działań profilaktycznych w celu regularnego sprawdzania tabel.
- Zdefiniowanie polityki regularnego tworzenia kopii zapasowej, z której będzie można odzyskać dane w przypadku utraty lub uszkodzenia bazy danych.

W tym miejscu zostaną omówione dwa pierwsze wymienione punkty. Z kolei informacje na temat technik tworzenia kopii zapasowych znajdziesz w podrozdziale 14.4, zatytułowanym „Tworzenie kopii zapasowej bazy danych”.

14.3.1. Używanie możliwości serwera w zakresie automatycznej naprawy

Jedną z pierwszych linii obrony w zachowaniu spójności bazy danych jest oferowana przez serwer MySQL możliwość naprawy po wystąpieniu awarii. Jedną z tych możliwości (naprawa transakcyjnego silnika bazy danych) jest przeprowadzana automatycznie podczas uruchamiania serwera. Kolejna (naprawa MyISAM) jest opcjonalna i musi być wyraźnie włączona.

Podczas uruchamiania serwera przeprowadza on pewne operacje sprawdzenia tabel, aby w ten sposób rozwiązać ewentualne problemy, które mogły powstać, jeśli komputer uległ awarii. Serwer MySQL został zaprojektowany do naprawy wielu różnych problemów. Dlatego też jeśli nic nie zrobisz i ograniczysz się jedynie do ponownego uruchomienia serwera, w wielu przypadkach przeprowadzi on niezbędne naprawy. Na przykład, silnik InnoDB automatycznie sprawdza tabele pod kątem błędów. Zatwierdzone transakcje zapisane w dzienniku zdarzeń, ale nie w tabelach, zostają ponownie wykonane. Niezatwierdzone transakcje, które jeszcze trwały w momencie awarii, zostają całkowicie wycofane.

W ten sposób tabele InnoDB pozostają w spójnym stanie, a ich zawartość odzwierciedla wszystkie transakcje zatwierdzone do chwili wystąpienia awarii.

Jeżeli przeprowadzana przez silnik InnoDB automatyczna naprawa zakończy się niepowodzeniem ze względu na błąd niemożliwy do usunięcia, serwer zapisze odpowiedni komunikat w dzienniku zdarzeń i zakończy działanie. Aby mimo tego wymusić uruchomienie serwera i przeprowadzić ręczną procedurę naprawy, zapoznaj się z punktem 14.7.4, zatytułowanym „Rozwiązywanie problemów związanych z automatyczną naprawą w InnoDB”.

W przypadku tabel MyISAM serwer obsługuje opcjonalną operację naprawy tabel, którą jednak trzeba wyraźnie włączyć. Kiedy to zrobisz, serwer będzie sprawdzał tabelę po każdym jej otwarciu. Jeżeli po ostatnim otwarciu tabela nie została prawidłowo zamknięta lub jest oznaczona jako tabela, która uległa awarii, serwer ją sprawdzi i naprawi. Włączenie opcji naprawy tabel MyISAM wymaga uruchomienia serwera wraz z ustawioną zmienną systemową `myisam_recover_options`. Wartością wymienionej zmiennej jest rozdzielona przecinkami lista jednej lub kilku następujących opcji: `BACKUP` (utworzenie kopii tabeli, jeśli naprawa ma spowodować jej modyfikację), `FORCE` (wymuszenie naprawy, nawet jeśli utracony będzie rekord danych), `QUICK` (szybka naprawa), `DEFAULT` (naprawa i nic poza tym) i `OFF` (bez naprawy). Na przykład, aby wymusić naprawę tabeli po wystąpieniu problemów, ale ze wcześniejszym wykonaniem jej kopii zapasowej, w pliku opcji serwera umieść następujące wiersze:

```
[mysql]
myisam_recover_options=BACKUP, FORCE
```

Włączenie automatycznej naprawy tabel MyISAM jest użyteczną strategią ogólnej obsługi, ponieważ w przeciwnym razie tabela MyISAM, w której serwer znajdzie problemy, staje się niedostępna aż do chwili, gdy się zorientujesz i naprawisz ją ręcznie. Naprawa tabel MyISAM jest szczególnie ważna w przypadku uruchamiania serwera wraz z ustawioną zmienną systemową `delay_key_write` lub skonfigurowanymi poszczególnymi tabelami do użycia zapisu kluczy z opóźnieniem. W pewnych sytuacjach zmiany w indeksie nie zostaną zapisane na dysku aż do zamknięcia tabeli, co zwiększa wydajność działającego serwera. Jednocześnie oznacza konieczność naprawy indeksów otwartych w momencie wystąpienia awarii tabel, które stosują zapis kluczy z opóźnieniem.

14.3.2. Harmonogram działań profilaktycznych

Poza włączeniem automatycznej naprawy rozważ także przygotowanie harmonogramu działań profilaktycznych. To pomaga w automatycznym wykrywaniu problemów, co pozwala na podjęcie kroków w celu ich usunięcia. Dzięki regularnemu przeprowadzaniu operacji sprawdzania tabel zmniejsza się ryzyko konieczności sięgania po kopię zapasową. W systemach UNIX najłatwiejszym rozwiązaniem jest zdefiniowanie zadania mechanizmu `cron`, najczęściej w pliku `crontab` dla użytkownika uruchamiającego serwer. (Zapoznaj się z podpunktem 12.8.7.3, zatytułowanym „Automatyzacja procedury utraty ważności

plików dzienników zdarzeń”, w którym znajdziesz więcej informacji na temat definiowania zadań cron).

Program `mysqlcheck` jest użyteczny do sprawdzania tabel InnoDB i MyISAM w trakcie działania serwera. Przyjmujemy założenie, że chcesz zdefiniować zadanie wywołujące `mysqlcheck`. Jeżeli serwer jest uruchamiany przez użytkownika `mysql`, regularne sprawdzanie tabeli można skonfigurować w pliku *crontab* wymienionego użytkownika. Dodaj wiersz podobny do przedstawionego poniżej. Wprowadź całość jako jeden wiersz i podaj prawidłową ścieżkę dostępu do programu `mysqlcheck` w Twoim systemie:

```
45 3 * * 0 /usr/local/mysql/bin/mysqlcheck
--all-databases --check-only-changed --silent
```

Powyższy wiersz powoduje uruchamianie programu `mysqlcheck` o godzinie 3:45 w każdą niedzielę. Harmonogram możesz zdefiniować zgodnie z własnymi potrzebami.

Opcja `--all-databases` powoduje, że program `mysqlcheck` sprawdza wszystkie table we wszystkich bazach danych. W ten sposób możesz go wykorzystać w celu osiągnięcia maksymalnego efektu. Jeżeli chcesz, aby program sprawdzał jedynie wybrane table lub bazy danych, zapoznaj się z opisem `mysqlcheck` przedstawionym w dodatku F, zatytułowanym „Przewodnik po programie MySQL”.

Opcja `--check-only-changed` nakazuje programowi `mysqlcheck` pominięcie wszystkich tabel, które nie zostały zmodyfikowane od chwili ostatniego sprawdzenia zakończonego powodzeniem. Z kolei opcja `--silent` wyłącza wyświetlanie danych wyjściowych programu, o ile nie zostaną znalezione błędy w tabelach. Mechanizm cron najczęściej wysła wiadomość e-mail, jeśli zadanie spowodowało wygenerowanie jakichkolwiek danych wyjściowych. Nie ma powodu otrzymywania wiadomości e-mail informującej, że nie zostały znalezione żadne błędy w tabelach. Zwróć uwagę, że nawet w przypadku użycia opcji `--silent` możesz otrzymać pewne diagnostyczne dane wyjściowe, jeśli bazy danych zawierają table dla silników baz danych nieobsługiwanych przez program `mysqlcheck`.

Wreszcie, pamiętaj o sprawdzaniu poczty elektronicznej konta `mysql` lub przekierowaniu jej na własny adres.

Uwaga

Kiedy tabela jest sprawdzana, wtedy nie może być uaktualniana. Strategie automatycznej obsługi mogą być nieodpowiednie w przypadku ogromnych tabel wymagających częstego uaktualniania, o ile nie możesz sobie pozwolić na nałożenie na nie blokady na czas przeprowadzenia operacji sprawdzenia.

14.4. Tworzenie kopii zapasowej bazy danych

Bardzo ważne jest tworzenie kopii zapasowej baz danych na wypadek utraty lub uszkodzenia danych. Jeżeli dojdzie do poważnej awarii systemu, kopia zapasowa pozwala na przywrócenie tabel i ograniczenie potencjalnych strat do minimum. Kopia zapasowa umożliwia też udzielenie pomocy użytkownikom, którzy przypadkowo wykonali zapytanie `DROP DATABASE`, `DROP TABLE` lub `DELETE` i proszą o przywrócenie danych.

Kopia bazy danych przydaje się także w operacjach niezwiązanych z odzyskiwaniem danych, na przykład do przeniesienia baz danych do innego serwera. Najczęściej zdarza się przeniesienie serwera do innego komputera, choć można również przenosić dane do innego serwera działającego w tym samym komputerze. Zdarza się to, jeśli na przykład testujesz nową wersję MySQL i chcesz w niej wykorzystać rzeczywiste dane pochodzące z serwera produkcyjnego.

Innym zastosowaniem dla kopii zapasowej jest konfiguracja serwera replikacji. Wówczas jednym z pierwszych kroków podczas konfiguracji serwera podległego jest utworzenie migawki serwera głównego w danym momencie. Kopia zapasowa służy w charakterze wspomnianej migawki, a jej wczytanie w serwerze podległym pozwala na jego uaktualnienie zgodnie z zawartością przechowywaną w serwerze głównym. Kolejne uaktualnienia w serwerze głównym są replikowane do podległego za pomocą standardowego protokołu replikacji. Zapoznaj się z podrozdziałem 14.8, zatytułowanym „Konfiguracja serwerów replikacji”.

Na początek przedstawione będą pewne ogólne zasady rządzące tworzeniem kopii zapasowej, co pomoże Ci w wyborze odpowiednich technik. Następnie dokładnie omówimy wybrane metody tworzenia kopii zapasowej.

Ogólnie rzecz biorąc, istnieją dwie ogólne kategorie kopii zapasowej bazy danych:

- Kopie zapasowe w formacie tekstowym wykonywane za pomocą narzędzia `mysqldump`, które zawartość tabel zapisuje w plikach. Wspomniane pliki zawierają zapytania SQL `CREATE TABLE` i `INSERT`, których późniejsze wykonanie powoduje przywrócenie tabel w serwerze.
- Binarne kopie zapasowe tworzone przez bezpośrednie kopiowanie plików zawierających tabele. Ten rodzaj kopii zapasowej może być wykonany na różne sposoby. Na przykład, można użyć programu takiego jak `cp`, `tar` lub `rsync`.

Każda z metod ma swoje wady i zalety. Wybrane czynniki, które trzeba wziąć pod uwagę, to: możliwość pozostawienia działającego serwera w trakcie tworzenia kopii zapasowej, czas potrzebny na wykonanie kopii zapasowej, możliwość jej przeniesienia, a także wskazanie danych przeznaczonych do umieszczenia w kopii zapasowej.

- Narzędzie `mysqldump` współpracuje z serwerem MySQL, a więc można używać wymienionego narzędzia, gdy serwer działa. Metody powodujące utworzenie binarnej kopii zapasowej obejmują operacje kopiowania plików, które są przeprowadzane poza serwerem. Niektóre tego rodzaju metody wymagają więc zatrzymania serwera. W przypadku metod niewymagających zatrzymania serwera nadal trzeba podjąć kroki gwarantujące, że serwer nie będzie modyfikował tabel w trakcie ich kopiowania.
- Narzędzie `mysqldump` działa wolniej niż techniki tworzenia binarnej kopii zapasowej, ponieważ wymaga odczytu tabel przez serwer, przeprowadzenia ich konwersji na postać gotową do transmisji i wysłania danych do `mysqldump` przez połączenie sieciowe. Z kolei metody tworzenia binarnej kopii zapasowej kopiuje pliki na poziomie systemu plików i nie wymagają przeprowadzania konwersji oraz wysyłania danych przez sieć.

- Narzędzie `mysql dump` generuje pliki tekstowe zawierające zapytania SQL. Tego rodzaju pliki można przenosić do innych komputerów, nawet o innej architekturze sprzętowej. Dlatego też są użyteczne do kopiowania baz danych między serwerami. Pliki wygenerowane przez metody tworzenia binarnej kopii zapasowej przez bezpośrednie kopiowanie plików tabel mogą, ale nie muszą być możliwe do przeniesienia do innych komputerów. To zależy między innymi od tego, czy tabele używają formatu silnika bazy danych charakterystycznego dla danego komputera. Tabele InnoDB i MyISAM normalnie są niezależne od komputera. W przypadku wymienionych tabel bezpośrednio skopiowane pliki mogą być przeniesione do serwera działającego w komputerze o innej architekturze sprzętowej. Więcej informacji na temat przenośności różnych silników bazy danych znajdziesz w punkcie 14.4.1, zatytułowanym „Cechy charakterystyczne przenośności silników bazy danych”.
- Dane wyjściowe narzędzia `mysql dump` składają się jedynie z zawartości bazy danych (tabele, widoki, procedury składowane itd.). Nie zawierają natomiast informacji przechowywanych poza bazą danych, na przykład plików konfiguracyjnych, plików dzienników zdarzeń lub plików stanu replikacji. Binarne kopie zapasowe mogą zawierać dowolne lub wszystkie z wymienionych plików, ponieważ w trakcie wykonywania tego rodzaju kopii zapasowej można kopiować dowolnie wybrane pliki.

Niezależnie od wybranej metody tworzenia kopii zapasowej, uwzględnienie wymienionych poniżej reguł gwarantuje otrzymanie najlepszych wyników, gdy kiedykolwiek wystąpi potrzeba przywrócenia zawartości bazy danych:

- Kopię zapasową trzeba wykonywać regularnie. Zdefiniuj harmonogram i trzymaj się go.
- Skonfiguruj serwer do tworzenia binarnego dziennika zdarzeń (patrz podrozdział 12.8, zatytułowany „Dzienniki zdarzeń serwera”). Binarny dziennik zdarzeń pomaga w przywróceniu bazy danych po wystąpieniu awarii. Za pomocą plików kopii zapasowej przywracasz bazy danych do stanu, w jakim znajdowały się w trakcie tworzenia danej kopii zapasowej. Następnie zmiany wprowadzone już po utworzeniu kopii zapasowej odzyskujesz przez ponowne wykonanie zapytań umieszczonych w plikach binarnych dzienników zdarzeń. W ten sposób tabele w bazach danych zostaną przywrócone do stanu w chwili awarii.
- Stosuj spójny i czytelny schemat nadawania nazw plikom kopii zapasowej. Nazwy takie jak *kopia1*, *kopia2* itd. nie są szczególnie użyteczne. Gdy wystąpi potrzeba przywrócenia danych, będziesz marnował czas na sprawdzanie zawartości tak nazwanych plików. Użyteczne może być tworzenie nazw plików kopii zapasowych zawierających nazwy baz danych i daty. Na przykład, jeżeli kopię zapasową bazy danych *sampdb* wykonasz 2 stycznia 2013 roku, to użyteczną nazwą będzie *sampdb-2013-01-02*. W przypadku posiadania lub obsługi wielu serwerów w nazwie należy umieścić także identyfikator serwera.

- Pliki kopii zapasowej przechowuj w innym systemie plików niż używany przez bazy danych. W ten sposób spada niebezpieczeństwo zapełnienia zawierającego katalog danych MySQL systemu plików na skutek generowania kopii zapasowych. Ponadto, jeżeli system plików zawierający kopie zapasowe znajduje się w zupełnie innym fizycznie napędzie, w ten sposób jeszcze bardziej zmniejszasz niebezpieczeństwo utraty danych na skutek awarii napędu. W takiej sytuacji awaria napędu nie spowoduje jednoczesnej utraty katalogu danych MySQL i kopii zapasowych.
- Pliki kopii zapasowej bazy danych umieszczaj w kopiach zapasowych zwykłego systemu plików. Jeżeli dojdzie do totalnej awarii, obejmującej nie tylko katalog danych MySQL, ale cały napęd zawierający kopie zapasowe bazy danych, wtedy masz prawdziwe kłopoty. Nie zapominaj o tworzeniu kopii zapasowych plików dzienników zdarzeń.
- Aby uniemożliwić kopiom zapasowym zapełnienie dysku, pamiętaj o okresowym usuwaniu starych kopii. Jedną z technik jest rotacja plików kopii zapasowej. W punkcie 12.8.7, zatytułowanym „Zarządzanie dziennikami zdarzeń”, dokładnie przedstawiono związane z tym kwestie w relacji do plików dzienników zdarzeń, ale te same reguły mają zastosowanie również względem plików kopii zapasowych.

W poniższych punktach przedstawiono analizę przenośności silników baz danych, a także kilku wybranych metod tworzenia kopii zapasowej. Jeżeli używasz replikacji, w punkcie 14.8.4, zatytułowanym „Użycie serwera podległego replikacji do tworzenia kopii zapasowych”, znajdziesz omówienie metod, dzięki którym działanie serwera głównego pozostanie niezakłócone.

14.4.1. Cechy charakterystyczne przenośności silników bazy danych

Każda tabela zarządzana przez dany serwer MySQL jest przenośna do innego serwera w tym sensie, że jej zawartość można za pomocą narzędzia `mysql dump` zapisać w pliku tekstowym. Następnie tak utworzony plik można przenieść do komputera, w którym działa inny serwer, i wczytać jego zawartość, tym samym ponownie tworząc tabele. Innym rodzajem przenośności jest „przenośność binarna”, oznaczająca możliwość bezpośredniego skopiowania plików przedstawiających tabele do innego komputera, umieszczenie ich w odpowiednich położeniach wewnątrz katalogu danych i oczekiwanie, że serwer MySQL będzie w stanie użyć przedstawianych przez te pliki tabel.

Ogólnym wymaganiem dla przenośności binarnej tabel jest to, aby serwery źródłowy i docelowy były zgodne pod względem obsługiwanych funkcji. Na przykład, serwer docelowy musi zapewniać obsługę silnika bazy danych zarządzającego przenoszonymi tabelami. Jeżeli serwer docelowy nie ma odpowiedniego silnika bazy danych, nie będzie mógł uzyskać dostępu do tabel utworzonych przez dany silnik w serwerze źródłowym.

Pewne silniki bazy danych tworzą tabele zapewniające przenośność binarną, natomiast inne nie. Poniżej przedstawiono podsumowanie przenośności binarnej kilku silników bazy danych:

- Tabele InnoDB i MyISAM są przechowywane w formacie niezależnym od komputera i są przenośne binarnie przy założeniu, że procesor komputera używa arytmetyki kodu uzupełnień do dwóch oraz formatu IEEE dla liczb zmiennoprzecinkowych. O ile nie posiadasz naprawdę starego komputera, wymieniony warunek nie stanowi problemu. („Zmiennoprzecinkowy” oznacza tutaj FLOAT i DOUBLE. Kolumny DECIMAL zawierają wartości o stałej wielkości, które stosują przenośny format).

W przypadku silnika InnoDB dodatkowym wymaganiem dla przenośności binarnej jest to, aby nazwy baz danych i tabel były zapisane małymi literami. W wewnętrznym katalogu InnoDB przechowuje je jako zapisane małymi literami, ale plik *.frm* ma nazwę zapisaną literami, które zostały użyte w zapytaniu CREATE TABLE. To może doprowadzić do niezgodności wielkości liter, jeśli nazwa bazy danych lub tabeli będzie podana wielkimi literami, a następnie przeniesiona na platformę rozróżniającą wielkość liter w nazwach plików.

W przypadku InnoDB przenośność musi być zapewniona dla wszystkich tabel InnoDB pobranych jako całość, a nie na poziomie poszczególnych tabel. Domyślnie silnik bazy danych InnoDB przechowuje zawartość wszystkich tabel we współdzielonej systemowej przestrzeni tabel, a nie poszczególnych plikach dla tabel. Dlatego też przenośne mogą być lub nie być pliki przestrzeni tabel InnoDB, a nie poszczególne tabele. Oznacza to, że związane z typem zmiennoprzecinkowym ograniczenie w przenośności ma zastosowanie, jeśli *dowolna* z tabel używa tego typu kolumn. Nawet jeśli skonfigurujesz InnoDB do przechowywania tabel w oddzielnych przestrzeniach tabel, wpisy w katalogu danych InnoDB znajdują się w systemowej przestrzeni tabel.

- Tabele CSV są przenośne binarnie, ponieważ ich pliki danych *.csv* są zwykłymi plikami tekstowymi.
- Tabele MEMORY nie są przenośne binarnie, ponieważ ich zawartość jest przechowywana w pamięci, a nie na dysku.

Niezależnie od ogólnych cech charakterystycznych w zakresie przenośności silnika bazy danych, nie powinieneś podejmować prób kopiowania plików tabel lub przestrzeni tabel do innego serwera, zanim nie nastąpi pełne, prawidłowe zamknięcie serwera źródłowego. Jeżeli przeprowadzisz kopiowanie po nieprawidłowym zamknięciu serwera, nie masz gwarancji zachowania spójności tabel. Tabele mogą wymagać naprawy, pewne informacje transakcji mogą nadal znajdować się w plikach dzienników zdarzeń silnika bazy danych i wymagać ich wprowadzenia lub wycofania transakcji.

Czasami istnieje możliwość zakazania serwerowi dostępu do określonych tabel w trakcie kopiowania ich plików. Jednak jeśli serwer jest uruchomiony i następuje aktywne uaktualnianie tabel lub zmiany są buforowane w pamięci, zawartość tabel zapisanych na dysku będzie ulegała ciągłym zmianom, a powiązane z nimi pliki nie dadzą użytecznej kopii tabel.

Omówienie sytuacji, w których można uniknąć całkowitego zatrzymania serwera podczas kopiowania tabel, znajdziesz w podrozdziale 14.2, zatytułowanym „Obsługa bazy danych w działającym serwerze”.

14.4.2. Tworzenie kopii zapasowej za pomocą narzędzia mysqldump

Narzędzie `mysqldump` generuje pliki tekstowe, które domyślnie są zapisywane w formacie SQL zawierającym zapytania `CREATE TABLE` tworzące tabele oraz zapytania `INSERT` zawierające dane dla rekordów wspomnianych tabel. W celu późniejszego odtworzenia tak zapisanych tabel plik utworzony przez `mysqldump` wczytaj do serwera MySQL za pomocą narzędzia `mysql`. Na przykład, aby utworzyć kopię zapasową pojedynczej tabeli (`sampdb.member`), a następnie ją wczytać do bazy danych, należy wykonać poniższe polecenia:

```
% mysqldump sampdb member > member.sql
% mysql sampdb < member.sql
```

Nie używaj narzędzia `mysql import` do wczytania danych wyjściowych programu `mysqldump` zapisanych w formacie SQL. Narzędzie `mysql import` oczekuje rekordów danych, a nie zapytań SQL.

W celu utworzenia w pojedynczym pliku kopii zapasowej wszystkich tabel znajdujących się we wszystkich bazach danych należy użyć polecenia takiego jak poniższe:

```
% mysqldump --all-databases > /archive/mysql/dump-all.2013-01-02
```

Jednak w przypadku dużej ilości danych powstanie ogromny plik kopii zapasowej. Rozwiązaniem może być umieszczenie każdej bazy danych w oddzielnym pliku:

```
% mysqldump mysql > /archive/mysql/mysql.2013-01-02
% mysqldump sampdb > /archive/mysql/sampdb.2013-01-02
% ...
```

Dane wyjściowe narzędzia `mysqldump` przedstawiają się następująco:

```
-- MySQL dump 10.13 Distrib 5.5.30, for Linux (i686)
--
-- Host: localhost Database: sampdb
--
-- Server version 5.5.30-log
... wiele zapytań SET ...

--
-- Struktura tabeli `absence`
--

DROP TABLE IF EXISTS `absence`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `absence` (
  `student_id` int(10) unsigned NOT NULL,
  `date` date NOT NULL,
  PRIMARY KEY (`student_id`,`date`),
```

```

CONSTRAINT `absence_ibfk_1` FOREIGN KEY (`student_id`)
REFERENCES `student` (`student_id`)
) ENGINE=InnoDB DEFAULT CHARSET=Tatin1;

/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dane dla tabeli `absence`
--

LOCK TABLES `absence` WRITE;
/*!40000 ALTER TABLE `absence` DISABLE KEYS */;
INSERT INTO `absence` VALUES (3,'2012-09-03'),(5,'2012-09-03'),
(10,'2012-09-06'),(10,'2012-09-09'),(17,'2012-09-07'),(20,'2012-09-07');
/*!40000 ALTER TABLE `absence` ENABLE KEYS */;
UNLOCK TABLES;
... pozostała część danych wyjściowych ...

```

W pozostałej części pliku znajdują się zapytania SQL, takie jak CREATE TABLE i INSERT.

Pliki kopii zapasowych bardzo często są ogromne i dlatego warto je zmniejszyć. Jednym ze sposobów jest kompresja pliku. W systemach Windows można do tego celu użyć narzędzia WinZip lub podobnego, co spowoduje kompresję pliku i utworzenie archiwum w formacie ZIP. Z kolei w systemach UNIX można użyć narzędzi gzip i bzip2. Istnieje nawet możliwość kompresji plików w trakcie tworzenia kopii zapasowej. Takie rozwiązanie wymaga użycia potoku:

```

% mysqldump sampdb | gzip > /archive/mysql/sampdb.2013-01-02.gz
% mysqldump sampdb | bzip2 > /archive/mysql/sampdb.2013-01-02.bz2

```

Jeżeli ogromne pliki kopii zapasowej są trudne w zarządzaniu, można tworzyć pliki poszczególnych tabel, podając w wywołaniu narzędzia mysqldump nazwę tabeli po nazwie bazy danych. Narzędzie mysqldump utworzy kopię zapasową wskazanej tabeli zamiast wszystkich tabel w bazie danych, co oznacza mniejsze, łatwiejsze w zarządzaniu pliki. Poniższy przykład pokazuje, jak utworzyć w oddzielnych plikach kopie zapasowe wybranych tabel bazy danych sampdb:

```

% mysqldump sampdb member president > hist-league.sql
% mysqldump sampdb student score grade_event absence > gradebook.sql

```

Narzędzie mysqldump ma wiele opcji, a poniżej wymieniono jedynie te, które możesz uznać za najszyteczniejsze. Pełną listę opcji znajdziesz w dodatku F, zatytułowanym „Przewodnik po programie MySQL”.

- W wierszu poleceń wywołującym narzędzie mysqldump zwykle podajesz nazwę bazy danych i opcjonalnie nazwy tabel. W celu jednoczesnego utworzenia kopii zapasowych wielu baz danych należy użyć opcji --databases. Narzędzie mysqldump zinterpretuje podane nazwy jako nazwy baz danych i utworzy kopie zapasowe wszystkich znajdujących się w nich tabel. Aby utworzyć kopie zapasowe wszystkich baz danych w serwerze, trzeba użyć opcji --all-databases. W takim przypadku nie podajesz żadnego argumentu w postaci nazwy bazy danych lub tabeli. Obie opcje, --databases i --all-databases, powodują umieszczenie na początku danych wyjściowych zapytań CREATE DATABASE IF NOT EXISTS i USE.

Zachowaj ostrożność podczas używania opcji `--all-databases`, jeśli wygenerowane w ten sposób dane wyjściowe chcesz wczytać w innym serwerze. Kopia zapasowa będzie zawierała tabele uprawnień w bazie danych `mysql`, a przecież możesz nie chcieć zastąpienia tabel uprawnień wersjami pochodzącymi z innego serwera.

- Domyślnie narzędzie `mysql dump` umieszcza w pliku kopii zapasowej zarówno strukturę tabeli (zapytania `CREATE TABLE`), jak i jej zawartość (zapytania `INSERT`). W celu umieszczenia tylko jednego typu informacji użyj opcji `--no-create-info` lub `--no-data`.
- Opcja `--opt` powoduje optymalizację procesu tworzenia kopii zapasowej. Powoduje włączenie wielu innych opcji, których efektem jest optymalizacja procesu tworzenia kopii zapasowej i wygenerowanie mniejszych plików. Wspomniane opcje optymalizują także proces przywracania danych, ponieważ dane wyjściowe mogą być przetwarzane znacznie szybciej w trakcie późniejszego wczytywania tak utworzonego pliku. Na przykład, jednym z efektów użycia opcji `--opt` jest generowanie przez narzędzie `mysql dump` zapytań `INSERT` wstawiających wiele rekordów. Tego rodzaju zapytania zabierają mniej miejsca i mogą być ponownie wykonane znacznie szybciej niż odpowiadające im zapytania `INSERT` wstawiające pojedyncze rekordy.

Opcja `--opt` jest domyślnie włączona, więc nie musisz jej wyraźnie podawać. Jeżeli naprawdę chcesz otrzymać nieoptymalizowaną kopię zapasową, wtedy użyj opcji `--skip-opt`.

Tworzenie kopii zapasowych z użyciem opcji `--opt` jest powszechnie stosowane, ponieważ oznacza przyspieszenie całego procesu. Musisz jednak wiedzieć, że opcja `--opt` ma również swoją cenę. Powoduje ona optymalizację procedury tworzenia kopii zapasowej, a nie dostępu innych klientów do bazy danych. Dlatego też opcja `--opt` uniemożliwia innym użytkownikom uaktualnianie jakichkolwiek tabel w trakcie procesu tworzenia kopii zapasowej, ponieważ nakłada jednoczesną blokadę na wszystkie tabele.

Opcja `--opt` jest użyteczna podczas tworzenia plików kopii zapasowych przeznaczonych do okresowego odświeżania zawartości innej bazy danych (na przykład bazy danych w innym serwerze). Wynika to z automatycznego włączenia opcji `--add-drop-table`, nakazującej narzędziu `mysql dump` poprzedzenie każdego zapytania `CREATE TABLE` w pliku zapytaniem `DROP TABLE IF EXISTS` dotyczącym tej samej tabeli. Kiedy tak utworzony plik kopii zapasowej wczytasz później w drugiej bazie danych, nie otrzymasz komunikatu błędu, nawet jeśli tabela już istnieje. Jeżeli masz uruchomiony drugi serwer i nie jest on serwerem podległym replikacji, tę technikę możesz wykorzystać do okresowego kopiowania danych z baz danych znajdujących się w serwerze produkcyjnym.

Innym efektem opcji `--opt` jest włączenie opcji `--extended-insert`, powodującej, że narzędzie `mysql dump` generuje zapytania `INSERT` wstawiające wiele rekordów. W ten sposób można zaoszczędzić miejsce, ale kosztem mniej czytelnego pliku

kopii zapasowej. W celu wygenerowania zapytań INSERT wstawiających pojedyncze rekordy należy użyć opcji `--skip-extended-insert`.

- Połączenie opcji `--flush-logs` i `--lock-all-tables` jest użyteczne do utworzenia punktu kontrolnego bazy danych. Opcja `--lock-all-tables` powoduje nałożenie globalnej blokady zezwalającej tylko na odczyt tabel, natomiast `--flush-logs` zamyka i ponownie otwiera pliki dzienników zdarzeń. Jeżeli włączony jest binarny plik dziennika zdarzeń, opcja `--flush-logs` powoduje utworzenie nowego pliku binarnego dziennika zdarzeń, zawierającego jedynie modyfikacje danych wprowadzone kolejno do następnego punktu kontrolnego. W ten sposób można zsynchronizować dziennik zdarzeń do chwili utworzenia kopii zapasowej. (Wadą tego rozwiązania jest nałożenie blokady na wszystkie tabele, co w trakcie tworzenia kopii zapasowej uniemożliwia innym klientom przeprowadzanie operacji uaktualniania tabel).

Jeżeli używasz opcji `--flush-logs` do utworzenia punktu kontrolnego dzienników zdarzeń do chwili wykonania kopii zapasowej, najlepszym rozwiązaniem będzie utworzenie kopii zapasowej całej bazy danych. W trakcie operacji przywracania danych często zdarza się wyodrębnianie zawartości dziennika zdarzeń dotyczących wskazanej bazy danych. Jeżeli utworzysz kopie zapasowe poszczególnych tabel, synchronizacja punktów kontrolnych dzienników zdarzeń będzie trudniejsza. (Nie ma opcji pozwalającej na wyodrębnianie z plików dzienników zdarzeń uaktualnień dla poszczególnych tabel, a więc będziesz musiał to robić ręcznie).

- W celu utworzenia kopii zapasowej tabel InnoDB użyj opcji `--single-transaction`. W ten sposób operacja będzie przeprowadzona w ramach transakcji i otrzymasz spójną kopię zapasową.
- Jeżeli bazy danych zawierają procedury składowane, wyzwalacze i zdarzenia, możesz wyraźnie je dołączyć do danych wyjściowych za pomocą opcji `--routines`, `--triggers` i `--events`. Wszystkie wymienione opcje mają również formę `--skip` (na przykład `--skip-triggers`), powodującą wykluczenie odpowiadających jej obiektów. Domyślnie, wyzwalacze są dołączone (ponieważ są powiązane z tabelami), natomiast procedury składowane i zdarzenia nie.
- Opcja `--master-data` jest użyteczna podczas generowania w serwerze głównym pliku kopii zapasowej przeznaczonego do wczytania w serwerze podległym replikacji. Dzięki wymienionej opcji plik kopii zapasowej zawiera informacje o binarnym dzienniku zdarzeń serwera głównego pozwalające serwerowi podległemu na rozpoczęcie replikacji po wczytaniu zawartości pliku kopii zapasowej.

14.4.3. Tworzenie binarnej kopii zapasowej

Metodą tworzenia kopii zapasowej baz danych lub tabel bez użycia narzędzia `mysqldump` jest bezpośrednie skopiowanie plików tabel. Zwykle odbywa się to za pomocą zwykłych narzędzi systemu plików (na przykład `cp`, `tar` lub `rsync`) lub specjalnego programu

opracowanego do tego celu (na przykład programu komercyjnego o nazwie MySQL Enterprise Backup). W trakcie stosowania metody bezpośredniego kopiowania plików trzeba zwrócić uwagę na dwie kwestie:

- Upewnij się, że tabele nie są używane. Jeżeli serwer zmodyfikuje tabele w trakcie kopiowania ich plików, tak powstała kopia jest bezużyteczna. Najlepszym sposobem zapewnienia spójności kopii zapasowej jest zatrzymanie serwera, skopiowanie plików i ponowne uruchomienie serwera. W rzeczywistości pewne metody tworzenia binarnej kopii zapasowej wymagają zatrzymania serwera. Jeżeli nie chcesz zatrzymywać serwera (i stosowana metoda wykonania kopii zapasowej tego nie wymaga), nałóż blokadę zezwalającą tylko na odczyt tabel, aby w ten sposób uniemożliwić serwerowi modyfikację tabel w trakcie kopiowania ich plików. Zapoznaj się z podrozdziałem 14.2, zatytułowanym „Obsługa bazy danych w działającym serwerze”.
- Konieczne jest skopiowanie wszystkich plików wymaganych do przywracania tabel, których kopię zapasową tworzysz. Metoda bezpośredniego kopiowania plików jest najłatwiejsza do stosowania w przypadku silników baz danych takich jak MyISAM, przedstawiających tabelę za pomocą unikalnego zestawu plików umieszczanych w katalogu danych MySQL. W celu utworzenia kopii zapasowej tabeli MyISAM musisz skopiować jedynie pliki *.frm*, *.myd* i *.myi*. Z kolei w przypadku silnika bazy danych takiego jak InnoDB procedura jest nieco bardziej skomplikowana: konieczne jest skopiowanie plików *.frm* oraz wszystkich plików przestrzeni tabel i plików dzienników zdarzeń silnika InnoDB.

Jeżeli tworzysz binarną kopię zapasową, uważaj na dowiązania symboliczne, na przykład znajdujące się w katalogu danych MySQL bądź dowiązania symboliczne do plików danych lub indeksów MyISAM. Stanowią one problem, ponieważ stosowana technika kopiowania plików może spowodować skopiowanie jedynie dowiązań symbolicznych, a nie danych, do których one prowadzą.

14.4.3.1. Utworzenie pełnej binarnej kopii zapasowej

Pełna binarna kopia zapasowa zawiera wszystkie pliki przechowujące zawartość tabel oraz wszystkie pliki dzienników zdarzeń używane przez konkretny silnik bazy danych. Poza wymienionymi powinienś skopiować także pliki binarnego dziennika zdarzeń. Jeżeli serwer jest serwerem podległym replikacji, skopiuj także pliki dziennika przekazywania oraz pliki *master.info* i *relay-log.info*. Ponadto, serwer podległy może tworzyć pliki o nazwach w postaci *SQL_LOAD-xxx* w katalogu plików tymczasowych. Te pliki również należy skopiować, ponieważ będą potrzebne dla zapytań *LOAD DATA*. Wspomniane pliki znajdują się w katalogu wskazanym przez zmienną systemową *slave_load_tmpdir*; w przypadku jej braku to będzie wartość domyślna zmiennej systemowej *tmpdir*. Aby ułatwić dodawanie tych plików do kopii zapasowej, utwórz katalog przeznaczony do wykorzystywania przez serwer podległy, a następnie uruchom serwer podległy wraz ze zmienną *slave_load_tmpdir* prowadzącą do nowo utworzonego katalogu.

W celu prawidłowego skopiowania wszystkich wymienionych plików trzeba zatrzymać serwer (to musi być pełne, prawidłowe zatrzymanie), aby umożliwić silnikowi bazy danych zamknięcie plików dzienników zdarzeń, a serwerowi zamknięcie innych używanych plików dzienników zdarzeń.

Wydaje się, że w tworzonej kopii zapasowej trzeba umieścić sporo plików, ale tak naprawdę to nie jest tak skomplikowane, jak możesz sądzić na początku. Na przykład, wszystkie katalogi baz danych znajdują się w katalogu danych MySQL, w którym domyślnie są tworzone również pliki dzienników zdarzeń i informacyjne. W takim przypadku kopię zapasową stworzysz, zatrzymując serwer i kopiując cały katalog danych MySQL. Na przykład, w celu utworzenia kopii zapasowej w postaci skompresowanego pliku *.tar* w katalogu */archive/mysql* przejdź do katalogu danych MySQL i utwórz jego kopię:

```
% cd /usr/local/mysql/data
% tar czf /archive/mysql/backup-all-2013-04-11.tar.gz .
```

14.4.3.2. Utworzenie częściowej binarnej kopii zapasowej

Utworzenie częściowej binarnej kopii zapasowej przez kopiowanie plików jest podobne do tworzenia pełnej kopii zapasowej, za wyjątkiem faktu, że kopiowane są jedynie wybrane pliki. Przyjmujemy założenie, że chcesz utworzyć kopię zapasową bazy danych *mydb* znajdującej się w katalogu */usr/local/mysql/data* i umieścić archiwum w katalogu */archive/mysql*. Zatrzymaj więc serwer, a następnie wydaj poniższe polecenia:

```
% cd /usr/local/mysql/data
% cp -r mydb /archive/mysql
```

Po wykonaniu powyższych poleceń katalog */archive/mysql/mydb* będzie zawierał kopię bazy danych *mydb*.

Po zakończeniu tworzenia kopii zapasowej uruchom serwer.

W pewnych sytuacjach częściowa kopia zapasowa może być utworzona bez konieczności zatrzymania serwera, o ile nałożysz blokadę pozwalającą jedynie odczytać tabele, które mają być skopiowane. Dotyczy to baz danych zawierających jedynie tabele, na przykład MyISAM. Jeżeli baza danych *mydb* użyta we wcześniejszych przykładach jest właśnie tego rodzaju bazą danych, możesz nałożyć blokadę, opróżnić dzienniki zdarzeń, wydać przedstawione polecenia, a następnie zwolnić blokadę po zakończeniu tworzenia kopii zapasowej.

14.4.4. Tworzenie kopii zapasowej tabel InnoDB

Podobnie jak w przypadku innych rodzajów tabel, kopie zapasowe tabel transakcyjnego silnika bazy danych InnoDB można tworzyć za pomocą narzędzia *mysql dump*. Użyteczną opcją jest tutaj *--single-transaction*, która powoduje utworzenie kopii zapasowej tabel w ramach pojedynczej transakcji. W przypadku InnoDB takie rozwiązanie gwarantuje, że tabele nie będą modyfikowane w trakcie całej operacji i otrzymasz spójną kopię zapasową.

W celu wykonania binarnej kopii zapasowej tabel InnoDB pod uwagę powinien być wziąć wymienione poniżej kwestie:

- Silnik InnoDB ma własne pliki dziennika zdarzeń do zarządzania transakcjami, które są aktywne w trakcie działania serwera. Dlatego też w celu wykonania binarnej kopii danych *trzeba* zatrzymać serwer. Co więcej, serwer musi być prawidłowo w pełni zamknięty, aby umożliwić silnikowi InnoDB zakończenie trwających transakcji i prawidłowe zamknięcie plików dzienników zdarzeń.
- W celu utworzenia binarnej kopii zapasowej tabel InnoDB należy skopiować wymienione poniżej pliki:
 - ◆ Pliki systemowej przestrzeni tabel.
 - ◆ Plik *.frm* każdej tabeli.
 - ◆ Plik *.ibd* każdej tabeli, o ile skonfigurowano InnoDB do używania oddzielnych przestrzeni tabel dla poszczególnych tabel.
 - ◆ Pliki dziennika zdarzeń InnoDB.
 - ◆ Plik opcji, w którym znajduje się konfiguracja systemowej przestrzeni tabel. (Upewnij się, że został skopiowany plik opcji, ponieważ będziesz go potrzebował do ponownej inicjalizacji systemowej przestrzeni tabel).

Binarną kopię zapasową tabel InnoDB można utworzyć także za pomocą programu komercyjnego o nazwie MySQL Enterprise Backup. Wymieniony program jest oferowany przez firmę Oracle i pozwala na tworzenie kopii zapasowych InnoDB, gdy serwer działa. Informacje szczegółowe na jego temat znajdziesz na witrynie <http://www.mysql.com/>.

14.5. Kopiowanie baz danych do innego serwera

Kopie zapasowe bazy danych mogą być używane w celu kopiowania bazy danych z jednego serwera MySQL do innego. W tym podrozdziale zostaną przedstawione wybrane metody przenoszenia bazy danych. Na potrzeby prezentowanej tutaj analizy przyjęto założenie, że baza danych jest przenoszona z serwera w komputerze lokalnym do serwera w komputerze zdalnym o nazwie `boa.example.com`. Jednak oba serwery mogą również znajdować się w tym samym komputerze. Z przedstawionej tutaj analizy dowiesz się, jak kopiować całe bazy danych, ale poznane techniki możesz zaadaptować także do pojedynczych tabel.

W podrozdziale omówione będą dwie metody kopiowania bazy danych do innego serwera. Pierwsza powoduje utworzenie kopii zapasowej bazy danych w postaci pliku lub zestawu plików. Wspomniane pliki można następnie skopiować do drugiego komputera i wczytać je w serwerze MySQL. Natomiast druga metoda pokazuje utworzenie kopii zapasowej przez sieć — dane z jednego serwera są bezpośrednio wczytywane w drugim. W ten sposób unika się tworzenia jakichkolwiek plików pośrednich.

14.5.1. Kopiowanie bazy danych za pomocą pliku kopii zapasowej

W celu skopiowania bazy danych za pomocą tekstowego pliku kopii zapasowej wspomniany plik trzeba utworzyć przy użyciu narzędzia `mysqldump`, skopiować go do komputera drugiego serwera, a następnie wczytać do uruchomionego w nim serwera MySQL. Przedstawiony poniżej przykład pokazuje, jak skopiować bazę danych `sampdb` za pomocą takiej procedury:

1. Utworzenie pliku kopii zapasowej:

```
% mysqldump --databases sampdb > sampdb.sql
```

Opcja `--databases` powoduje, że narzędzie `mysqldump` umieści zapytania `CREATE DATABASE IF NOT EXISTS` i `USE` dla bazy danych `sampdb`. W ten sposób po wczytaniu pliku w zdalnym serwerze nastąpi automatyczne utworzenie i wybranie bazy danych, a tabele z kopii zapasowej będą wczytane do nowo utworzonej bazy danych.

2. Skopiowanie pliku kopii zapasowej do zdalnego serwera. Poniższe polecenie używa narzędzia `scp` w celu skopiowania pliku do katalogu `/tmp` w komputerze `boa.example.com`:

```
% scp sampdb.sql boa.example.com:/tmp
```

3. Zalogowanie się w zdalnym komputerze i wczytanie pliku kopii zapasowej w uruchomionym tam serwerze MySQL:

```
% mysql < /tmp/sampdb.sql
```

Inne podejście polega na użyciu binarnej kopii zapasowej: skopiuj pliki bazy danych (a nie plik kopii zapasowej) z jednego komputera do drugiego. Przyjmujemy założenie, że baza danych `mydb` zawiera jedynie tabele `MyISAM`. W takim przypadku zawartość tabel znajduje się w całości w plikach umieszczonych w katalogu bazy danych `mydb`. Jeżeli lokalny katalog danych MySQL to `/usr/local/mysql/data`, natomiast katalog danych MySQL w zdalnym komputerze `boa.example.com` to `/var/mysql/data`, wówczas poniższe polecenia spowodują skopiowanie katalogu bazy danych `mydb` do zdalnego komputera:

```
% cd /usr/local/mysql/data
```

```
% scp -r mydb boa.example.com:/var/mysql/data
```

Aby w taki sposób kopiować pliki bazy danych do innego komputera, konieczne jest spełnienie określonych warunków:

- Oba komputery muszą mieć taką samą architekturę sprzętową lub wszystkie kopiowane tabele muszą być przeznaczone dla przenośnego binarnego silnika bazy danych. W przeciwnym razie tabele w serwerze docelowym mogą zawierać bardzo dziwną treść.
- W obu serwerach trzeba uniemożliwić podejmowanie prób modyfikacji kopiowanych tabel. Najbezpieczniejszym rozwiązaniem jest zatrzymanie serwerów na czas pracy z tabelami.

14.5.2. Kopiowanie baz danych z jednego serwera do innego

Przedstawione w poprzednim punkcie techniki polegające na użyciu narzędzia `mysql dump` powodują utworzenie plików kopii zapasowych, które można skopiować do drugiego serwera. Innym rozwiązaniem jest przekazanie danych wyjściowych narzędzia `mysql dump` przez sieć bezpośrednio do drugiego serwera, bez konieczności tworzenia jakichkolwiek plików pośrednich. Na przykład, aby skopiować bazę danych `sampdb` z komputera lokalnego do zdalnego komputera o nazwie `boa.example.com`, należy wydać poniższe polecenie:

```
% mysqldump --databases sampdb | mysql -h boa.example.com
```

Program `mysql` odczytuje dane wyjściowe narzędzia `mysqldump`, nawiązuje połączenie z serwerem o nazwie `boa.example.com` i wczytuje w nim otrzymane dane.

Jeżeli z poziomu komputera lokalnego nie możesz uzyskać dostępu do zdalnego serwera MySQL, ale masz do niego dostęp po zalogowaniu się w zdalnym systemie, wtedy użyj `ssh` do zdalnego uruchomienia klienta `mysql`:

```
% mysqldump --databases sampdb | ssh boa.example.com mysql
```

W przypadku wolnej sieci opcja `--compress` może poprawić wydajność operacji kopiowania bazy danych do innego komputera, ponieważ zmniejsza ilość danych przekazywanych przez sieć:

```
% mysqldump --databases sampdb | mysql --compress -h boa.example.com sampdb
```

Opcja `--compress` jest używana w programie komunikującym się z serwerem w zdalnym komputerze, a nie podczas komunikacji z lokalnie uruchomionym serwerem MySQL. Kompresja ma zastosowanie tylko względem ruchu sieciowego i nie powoduje kompresji tabel tworzonych w docelowej bazie danych.

14.6. Sprawdzanie i naprawianie tabel bazy danych

Uszkodzenia bazy danych zdarzają się z wielu różnych powodów i różnią się skalą. Jeżeli masz szczęście, to może być niewielkie uszkodzenie tabeli lub dwóch (na przykład, gdy na skutek braku zasilania doszło do nagłego wyłączenia komputera). W takim przypadku istnieje duże prawdopodobieństwo, że serwer sam naprawi uszkodzenia po uruchomieniu. Natomiast jeśli nie miałeś szczęścia, może wystąpić konieczność przywrócenia całego katalogu danych (na przykład uszkodzeniu uległ dysk i straciłeś znajdujący się na nim katalog danych MySQL). Proces odzyskiwania danych może być niezbędny także w innych okolicznościach, na przykład po usunięciu niewłaściwej bazy danych, tabeli bądź też po usunięciu zawartości tabeli.

Jeżeli podejrzewasz uszkodzenie tabeli, sprawdź ją pod kątem błędów. Gdy tabela okaże się dobra, na tym koniec Twojej pracy. W przeciwnym razie musisz ją naprawić, kierując się przedstawionymi poniżej wskazówkami:

- Rozpocznij od szybszej, ale najmniej inwazyjnej metody naprawy.
- Jeżeli okaże się, że zastosowana metoda nie jest wystarczająca, wtedy zacznij stosować bardziej inwazyjne (choć wolniejsze) metody naprawy aż do chwili udanej naprawy lub wyczerpania dostępnych metod.

W praktyce większość problemów można usunąć bez stosowania bardziej inwazyjnych i wolniejszych metod naprawy.

W tym podrozdziale zostaną przedstawione procedury sprawdzania i naprawy pozwalające na usunięcie mniejszych uszkodzeń. W przypadku wystąpienia poważniejszych problemów, takich jak usunięcie lub odwracalne uszkodzenie tabeli bądź bazy danych, konieczne będzie ich przywrócenie z kopii zapasowej i binarnego dziennika zdarzeń. Informacje na ten temat znajdziesz w podrozdziale 14.7, zatytułowanym „Użycie kopii zapasowej do przywrócenia danych”.

Zaprezentowane będą także metody sprawdzania i naprawy tabel InnoDB oraz MyISAM, a następnie pewne dokładne informacje dotyczące tego zagadnienia.

W celu sprawdzenia tabel InnoDB należy wykonać zapytanie `CHECK TABLE` lub użyć narzędzia `mysqlcheck`, które nawiązuje połączenie z serwerem i wykonuje wymienione zapytanie.

Aby naprawić tabelę InnoDB, w której odkryto problemy, trzeba w pierwszej kolejności wykonać jej kopię zapasową. Następnie należy usunąć tabelę i ponownie ją utworzyć za pomocą przygotowanego wcześniej pliku kopii zapasowej. Poniższa sekwencja poleceń pokazuje, jak sprawdzić tabelę, utworzyć jej kopię zapasową, a następnie ponownie utworzyć tabelę `absence` w bazie danych `sampdb`:

```
% mysqlcheck sampdb absence
% mysqldump sampdb absence > absence.sql
% mysql sampdb < absence.sql
```

W celu sprawdzenia i naprawy tabel MyISAM możesz zastosować wymienione poniżej podejścia:

- Wykonać zapytania `CHECK TABLE` i `REPAIR TABLE` lub użyć narzędzia `mysqlcheck`, które nawiązuje połączenie z serwerem i wykonuje wymienione zapytania.
- Użyć narzędzia `myisamchk`, które działa bezpośrednio na plikach tabel.

Jak wspomniano wcześniej w rozdziale, jeśli w tracie obsługi tabel masz do wyboru zlecenie tego zadania serwerowi lub wykorzystanie narzędzia zewnętrznego, lepszym rozwiązaniem będzie pozwolenie serwerowi na wykonanie zadania. Nie trzeba się wówczas przejmować nakładaniem jakichkolwiek blokad lub też koordynacją dostępu do tabel. Dotyczy to stosowania zapytań `CHECK TABLE`, `REPAIR TABLE` i narzędzia `mysqlcheck`. W kolejnych punktach dowiesz się, jak skorzystać z wymienionych możliwości.

Jeżeli używasz narzędzia `myisamchk` w trakcie działania serwera, musisz zająć się nakładaniem blokad (zapoznaj się z podrozdziałem 14.2, zatytułowanym „Obsługa bazy danych w działającym serwerze”). Jednak narzędzie `myisamchk` ma również pewne zalety:

- Narzędzie `myisamchk` można używać, kiedy serwer jest zatrzymany. Zapytania `CREATE TABLE` i `REPAIR TABLE` wymagają uruchomionego serwera.
- Narzędziu `myisamchk` można nakazać użycie większych buforów, co powoduje przyspieszenie operacji sprawdzania i naprawy. To użyteczna możliwość w przypadku ogromnych tabel.

Informacje dotyczące użycia narzędzia `myisamchk` w celu obsługi bazy danych znajdziesz w podrozdziale F.3, zatytułowanym „Narzędzie `myisamchk`”.

14.6.1. Sprawdzanie tabel za pomocą zapytania `CHECK TABLE`

Zapytanie `CHECK TABLE` zapewnia interfejs prowadzący do możliwości serwera w zakresie sprawdzania tabel. Wymienione zapytanie działa wraz z tabelami InnoDB, MyISAM, ARCHIVE i CSV, a także z widokami.

W celu wykonania zapytania `CHECK TABLE` należy podać nazwę jednej lub więcej tabel oraz opcjonalnie modyfikatory wskazujące rodzaj operacji sprawdzania, jaka ma zostać przeprowadzona. Na przykład, poniższe zapytanie przeprowadza na średnim poziomie sprawdzenie trzech tabel, ale jedynie wtedy, gdy nie zostały prawidłowo zamknięte:

```
CHECK TABLE tb11, tb12, tb13 FAST MEDIUM;
```

Poniższa lista wymienia dostępne opcje sprawdzania tabel. Mają one zastosowanie względem tabel MyISAM, inne silniki bazy danych mogą je zignorować. Więcej informacji na ten temat znajdziesz w dodatku E, zatytułowanym „Przewodnik po składni SQL”.

- Opcja `CHANGED` pomija sprawdzenie tabeli, jeśli została zamknięta prawidłowo i nie została zmieniona od chwili jej ostatniego sprawdzenia.
- Opcja `EXTENDED` przeprowadza rozszerzone sprawdzenie i próbuje zapewnić maksymalną spójność tabeli.
- Opcja `FAST` sprawdza tabelę tylko wtedy, gdy nie została poprawnie zamknięta.
- Opcja `MEDIUM` sprawdza indeks, przeprowadza skanowanie rekordów danych pod kątem problemów oraz sprawdza sumę kontrolną. To jest opcja domyślna, jeśli żadna nie zostanie wyraźnie podana.
- Opcja `QUICK` powoduje skanowanie jedynie indeksów, rekordy danych są pomijane.
- Opcja `FOR UPGRADE` ustala, czy sprawdzana tabela jest zgodna z bieżącą wersją MySQL. Ta opcja jest użyteczna po przeprowadzeniu uaktualnienia oprogramowania serwera MySQL.

14.6.2. Naprawa tabel za pomocą zapytania REPAIR TABLE

Zapytanie REPAIR TABLE zapewnia interfejs prowadzący do możliwości serwera w zakresie naprawy tabel. Wymienione zapytanie działa wraz z tabelami MyISAM, ARCHIVE i CSV.

W celu wykonania zapytania REPAIR TABLE należy podać nazwę jednej lub więcej tabel oraz opcjonalnie modyfikatory wskazujące rodzaj operacji naprawy, jaka ma zostać przeprowadzona. Na przykład, poniższe zapytanie próbuje naprawić trzy tabele, stosując tryb naprawy szybkiej:

```
REPAIR TABLE tb11, tb12, tb13 QUICK;
```

Poniższa lista wymienia dostępne opcje naprawy tabel. Mają one zastosowanie względem tabel MyISAM, inne silniki bazy danych mogą je zignorować. Więcej informacji na ten temat znajdziesz w dodatku E, zatytułowanym „Przewodnik po składni SQL”.

- Opcja EXTENDED przeprowadza rozszerzoną naprawę, obejmującą ponowne utworzenie indeksów.
- Opcja QUICK powoduje szybką naprawę jedynie indeksów, rekordy danych są pomijane.
- Opcja USE_FRM używa pliku *.frm* tabeli w celu ponownej inicjalizacji pliku indeksu i określenia sposobu interpretacji zawartości pliku danych, aby było możliwe ponowne utworzenie indeksów. Ta opcja jest użyteczna w przypadku utraty lub nieodwracalnego uszkodzenia indeksów. Jednak powinna być stosowana tylko w ostateczności oraz *jedynie* w przypadku, gdy bieżąca wersja MySQL jest taka sama, jaka była używana do utworzenia tabeli. W przeciwnym razie może dojść do dalszego uszkodzenia tabeli.

Jeżeli nie zostaną podane żadne opcje, zapytanie REPAIR TABLE przeprowadza zwykłą operację naprawy tabeli, która może spowodować usunięcie większości problemów. Wyjątkiem są powtarzające się wartości indeksu, który powinien zawierać jedynie unikalne wartości.

14.6.3. Użycie narzędzia mysqlcheck do sprawdzania i naprawy tabel

Narzędzie mysqlcheck to dostępny z poziomu wiersza poleceń interfejs dla zapytań CHECK TABLE i REPAIR TABLE. Nawiązuje połączenie z serwerem oraz wykonuje odpowiednie zapytania na podstawie użytych opcji. Dlatego też narzędzie mysqlcheck może sprawdzić lub naprawić tabele tych samych silników bazy danych, jakie wymieniono podczas omawiania zapytań CHECK TABLE i REPAIR TABLE.

Zwykle narzędzie jest wywoływane wraz z nazwą bazy danych i opcjonalnie jedną lub więcej nazw tabel. W przypadku podania jedynie nazwy bazy danych narzędzie mysqlcheck sprawdza wszystkie znajdujące się w niej tabele:

```
% mysqlcheck sampdb
```

Jeżeli po nazwie bazy danych zostaną podane nazwy tabel, narzędzie `mysqlcheck` sprawdzi wymienione tabele:

```
% mysqlcheck sampdb president member
```

W przypadku użycia opcji `--databases` narzędzie `mysqlcheck` zinterpretuje wszystkie argumenty niebędące opcjami jako nazwy baz danych i sprawdzi wszystkie tabele w każdej wymienionej bazie danych:

```
% mysqlcheck --databases sampdb test
```

Z kolei po użyciu opcji `--all-databases` narzędzie `mysqlcheck` sprawdzi wszystkie tabele we wszystkich bazach danych. Nie jest konieczne podawanie żadnej nazwy bazy danych lub tabeli:

```
% mysqlcheck --all-databases
```

Narzędzie `mysqlcheck` jest znacznie wygodniejsze niż bezpośrednie wykonywanie zapytań `CHECK TABLE` i `REPAIR TABLE`, ponieważ wymagają one wyraźnego podania nazwy każdej tabeli do sprawdzenia lub naprawy. Za pomocą narzędzia `mysqlcheck` można znacznie łatwiej sprawdzić wszystkie tabele w bazie danych: narzędzie samodzielnie wyszukuje wszystkie tabele w bazie danych, a następnie wykonuje odpowiednie zapytania w celu ich sprawdzenia.

Domyślnie narzędzie `mysqlcheck` sprawdza tabele na średnim poziomie, ale obsługuje także opcje pozwalające na dokładne wskazanie rodzaju przeprowadzanej operacji. W przedstawionej poniżej tabeli 14.1 wymieniono pewne opcje narzędzia `mysqlcheck` i odpowiadające im opcje zapytania `CHECK TABLE`. Podobnie jak w przypadku zapytania `CHECK TABLE`, te opcje mają zastosowanie jedynie do tabel `MyISAM`, pozostałe silniki bazy danych mogą je zignorować.

Tabela 14.1. Opcje narzędzia `mysqlcheck` i odpowiadające im opcje zapytania `CHECK TABLE`

Opcja narzędzia <code>mysqlcheck</code>	Opcja zapytania <code>CHECK TABLE</code>
<code>--check-only-changed</code>	<code>CHANGED</code>
<code>--extended</code>	<code>EXTENDED</code>
<code>--fast</code>	<code>FAST</code>
<code>--medium-check</code>	<code>MEDIUM</code>
<code>--quick</code>	<code>QUICK</code>

W przypadku tabel `MyISAM`, `ARCHIVE` i `CSV` narzędzie `mysqlcheck` może również przeprowadzać operacje naprawy tabel. W przedstawionej poniżej tabeli 14.2 wymieniono pewne opcje narzędzia `mysqlcheck` i odpowiadające im opcje zapytania `REPAIR TABLE`. Podobnie jak w przypadku zapytania `CHECK TABLE`, te opcje mają zastosowanie jedynie do tabel `MyISAM`, pozostałe silniki bazy danych mogą je zignorować.

Tabela 14.2. Opcje narzędzia mysqlcheck i odpowiadające im opcje zapytania REPAIR TABLE

Opcja narzędzia mysqlcheck	Opcja zapytania REPAIR TABLE
--repair	Brak opcji (wykonuje standardową operację naprawy).
--repair --extended	EXTENDED
--repair --quick	QUICK
--repair --use-frm	USE_FRM

14.7. Użycie kopii zapasowej do przywrócenia danych

Procedury odtworzenia danych wykorzystują dwa źródła: pliki kopii zapasowej oraz binarnych dzienników zdarzeń. Pliki kopii zapasowej mogą być wygenerowane przez narzędzia takie jak `mysqldump` lub skopiowane za pomocą jednej z dostępnych metod tworzenia binarnej kopii zapasowej.

Dzięki tym plikom można przywrócić tabele do stanu, w jakim się znajdowały w chwili utworzenia kopii zapasowej. Z kolei pliki binarnego dziennika zdarzeń zawierają zmiany wprowadzone w tabelach od chwili ostatniego utworzenia kopii zapasowej. Narzędzie `mysqlbinlog` tego rodzaju pliki konwertuje z powrotem na tekst zapytań SQL, które można wykonać za pomocą klienta `mysql`. W ten sposób ponownie wprowadzane są zmiany dokonane między chwilą utworzenia kopii zapasowej i momentem wystąpienia awarii lub problemów.

Konkretny przebieg procedury odtwarzania danych zależy od ilości danych, które trzeba przywrócić. W rzeczywistości, przywrócenie całej bazy danych może okazać się łatwiejsze niż pojedynczej tabeli, ponieważ informacje zapisywane w plikach binarnego dziennika zdarzeń łatwiej się stosuje względem bazy danych niż tabeli.

W przedstawionej tutaj analizie przyjęto założenie, że regularnie tworzysz kopie zapasowe oraz włączyłeś rejestrację informacji w binarnym dzienniku zdarzeń. Jeśli jest inaczej, można powiedzieć, że żyjesz na krawędzi. Przed kontynuacją lektury powinieneś natychmiast włączyć binarny dziennik zdarzeń i utworzyć nową kopię zapasową. Na pewno nie chcesz znaleźć się w sytuacji, w której następuje utrata tabeli z powodu niewystarczającego przyłożenia się do wygenerowania danych niezbędnych do jej przywrócenia. Aby dowiedzieć się, jak włączyć binarny dziennik zdarzeń, zajrzyj do punktu 12.8.4, zatytułowanego „Binarny dziennik zdarzeń”. Z kolei informacje dotyczące tworzenia kopii zapasowej przedstawiono w podrozdziale 14.4, zatytułowanym „Tworzenie kopii zapasowej bazy danych”.

14.7.1. Przywracanie całych baz danych

Ogólna procedura pozwalająca na przywrócenie jednej lub więcej baz danych przedstawia się następująco:

1. Utworzenie kopii zawartości katalogu lub katalogów bazy danych. Ta kopia katalogów może się przydać w przypadku popełnienia błędu w trakcie procesu odzyskiwania danych lub jego zakończenia niepowodzeniem.
2. Ponowne utworzenie baz danych za pomocą ostatnio wykonanej kopii zapasowej.
Jeżeli kopia zapasowa składa się z plików wygenerowanych przez narzędzie `mysql dump`, wczytaj ich zawartość za pomocą klienta `mysql`.
Jeżeli baza danych lub bazy danych do przywrócenia obejmują także bazę danych `mysql` zawierającą tabele uprawnień i używasz plików utworzonych przez `mysql dump`, pliki wczytaj do serwera, używając przy tym opcji `--skip-grant-tables`.
W przeciwnym razie serwer może wyświetlić komunikat informujący o braku możliwości znalezienia tabel uprawnień. Dobrym rozwiązaniem jest użycie także opcji `--skip-networking`, aby serwer odrzucał wszelkie próby nawiązania połączenia podczas przeprowadzania operacji odtwarzania danych. Po przywróceniu tabel zatrzymaj serwer i uruchom go w zwykły sposób, aby używał tabel uprawnień i nasłuchiwał na zdefiniowanych interfejsach sieciowych.
Jeżeli używasz plików z binarnej kopii zapasowej (na przykład utworzonej za pomocą narzędzi `tar` lub `cp`), wtedy zatrzymaj serwer, aby nie próbował uzyskać dostępu do baz danych w trakcie ich przywracania. Następnie pliki skopiuj do ich pierwotnych katalogów (prawdopodobnie do katalogu danych MySQL) i ponownie uruchom serwer.
3. Użyj binarnego dziennika zdarzeń do ponownego wprowadzenia modyfikacji danych, które wystąpiły od chwili utworzenia ostatniej kopii zapasowej. Ta procedura została przedstawiona w punkcie 14.7.3, zatytułowanym „Ponowne wykonanie zapytań zapisanych w plikach binarnego dziennika zdarzeń”.

14.7.2. Przywracanie poszczególnych tabel

Przywracanie poszczególnych tabel może być znacznie trudniejsze niż całej bazy danych. Jeżeli masz wygenerowany przez narzędzie `mysql dump` plik kopii zapasowej zawierającej tylko jedną tabelę, po prostu wczytaj jego zawartość. W przypadku pliku zawierającego dane z wielu tabel dane tylko jednej możesz odzyskać, przeprowadzając edycję pliku i usuwając dane z pozostałych nieinteresujących Cię tabel, a następnie wczytując zawartość tak spreparowanego pliku. To jest łatwe zadanie.

Najtrudniejsza jest sytuacja, gdy trzeba wyodrębnić z binarnego dziennika zdarzeń informacje dotyczące tylko jednej tabeli. Narzędzie `mysqlbinlog` zawiera opcję `--database`, powodującą ograniczenie danych wyjściowych do zapytań dotyczących tylko jednej bazy danych. Brakuje natomiast tego rodzaju opcji, ale dotyczącej pojedynczej tabeli. Użyteczną strategią w takiej sytuacji może być przywrócenie większej ilości danych, niż trzeba (na przykład całej bazy danych zawierających żadaną tabelę), a następnie usunięcie niepotrzebnych danych. W rzeczywistości taka procedura może okazać się znacznie łatwiejsza niż próba przywrócenia pojedynczej tabeli przez wyodrębnianie z binarnego dziennika zdarzeń informacji dotyczących danej tabeli:

1. Przywróć całą bazę danych zawierającą żadaną tabelę, ale zrób to w drugiej, pustej bazie danych. Możesz w tym celu wykorzystać posiadaną kopię zapasową i ponownie wprowadzić zapytania zapisane w binarnym dzienniku zdarzeń. To jednak wiąże się z dwiema komplikacjami:
 - ◆ Utworzony przez narzędzie `mysldump` plik kopii zapasowej może zawierać zapytanie `USE` dotyczące oryginalnej bazy danych. Nie chcesz jej zmieniać lub usuwać przed użyciem tego pliku jako danych wejściowych dla klienta `mysql`.
 - ◆ Dane wyjściowe narzędzia `mysqlbinlog` zawierają jedno lub więcej zapytań `USE` wskazujących oryginalną bazę danych. Dane wyjściowe zapisz więc w pliku, co pozwoli na edycję wspomnianych zapytań i podanie w nich nazwy drugiej bazy danych. Dopiero tak spreparowany plik przekaż klientowi `mysql` jako dane wejściowe.
2. W drugiej bazie danych użyj narzędzia `mysqldump` do utworzenia pliku kopii zapasowej zawierającej interesującą Cię tabelę.
3. Usuń oryginalną tabelę, a następnie w oryginalnej bazie danych wczytaj przygotowany w poprzednim punkcie plik, powodując tym samym odtworzenie tabeli. Jeśli uruchomisz narzędzie `mysqldump` wraz z opcją `--opt` lub `--add-drop-table`, plik kopii zapasowej będzie zawierał zapytanie `DROP TABLE`, powodujące usunięcie tabeli przed jej odtworzeniem.

W przypadku tabel `MyISAM` alternatywne rozwiązanie polega na użyciu narzędzia `mysqldump` w celu bezpośredniego skopiowania plików tabeli z katalogu drugiej bazy danych do katalogu oryginalnej. Aby zachować bezpieczeństwo, zatrzymaj oba serwery podczas przeprowadzania operacji kopiowania plików.

14.7.3. Ponowne wykonanie zapytań zapisanych w plikach binarnego dziennika zdarzeń

Po przywróceniu baz danych lub tabel z plików kopii zapasowych kolejnym krokiem jest ponowne wykonanie tych zapytań z plików binarnego dziennika zdarzeń, które zostały wykonane już po utworzeniu ostatniej kopii zapasowej. W ten sposób tabele będą przywrócone do stanu, w jakim znajdowały się w chwili awarii.

Narzędzie `mysqlbinlog` konwertuje pliki binarnego dziennika zdarzeń na zapytania w postaci tekstowej, które w ten sposób są łatwe do wykonania: danych wyjściowych narzędzia `mysqlbinlog` użyj jako danych wejściowych klienta `mysql`.

W zależności od tego, jakie dane zostały przywrócone z kopii zapasowych, konieczne może być ponowne wprowadzenie wszystkich zapytań z binarnego dziennika zdarzeń lub tylko tych, które dotyczą konkretnej bazy danych. Możesz również zdecydować się na ponowne wykonanie zapytań z jedynie wskazanego przedziału czasu. Narzędzie `mysqlbinlog` oferuje tego rodzaju możliwości. Potrafi przetwarzać wiele plików binarnego dziennika zdarzeń i ograniczać dane wyjściowe do zapytań dotyczących jedynie wskazanej bazy danych lub zdefiniowanego przedziału czasu.

W omówionym poniżej przykładzie ponownego wykonywania zapytań z binarnego dziennika zdarzeń przyjęto założenie, że nazwy wszystkich plików są w postaci *binlog.nnnnnnn*, gdzie *nnnnnn* to sześciocyfrowe rozszerzenie wskazujące numer pliku w sekwencji. Jeżeli używasz plików binarnego dziennika zdarzeń o innych nazwach, to odpowiednio zmień prezentowane tutaj polecenia. Ponadto, koncentrujemy się na użyciu lokalnych plików binarnego dziennika zdarzeń istniejących w tym samym komputerze, w którym uruchomiono narzędzie `mysqlbinlog`. Samo narzędzie ma możliwość odczytywania także zdalnych plików binarnego dziennika zdarzeń, ale to nie będzie tutaj omówione. Informacje szczegółowe dotyczące opcji przetwarzania przez narzędzie `mysqlbinlog` zdalnego dziennika zdarzeń znajdziesz w dodatku F, zatytułowanym „Przewodnik po programie MySQL”.

Jeżeli kopia zapasowa użyta do przywrócenia baz danych została utworzona przed powstaniem wszystkich plików binarnego dziennika zdarzeń, wtedy konieczne jest wykonanie zapytań znajdujących się w każdym pliku. W tym celu przejdź do katalogu zawierającego pliki binarnego dziennika zdarzeń i wydaj poniższe polecenie:

```
% mysqlbinlog binlog.[0-9]* | mysql
```

Wzorzec `binlog.[0-9]*` w poleceniu `mysqlbinlog` powoduje dopasowanie listy plików binarnego dziennika zdarzeń, standardowo w tej samej kolejności, w której zostały wygenerowane przez serwer.

Jeżeli przed wykonaniem zapytań znajdujących się w pliku konieczne jest przeprowadzenie jego edycji, zapytania trzeba skonwertować na postać tekstu i zapisać w pliku. Teraz będzie można przeprowadzić edycję pliku, a następnie przekazać go jako dane wejściowe klienta `mysql`, na przykład:

```
% mysqlbinlog binlog.[0-9]* > plik_tekstowy
% vi plik_tekstowy
% mysql < plik_tekstowy
```

Tego rodzaju strategia jest potrzebna, jeśli powodem przywracania danych i użycia plików binarnego dziennika zdarzeń do przywrócenia danych było wykonanie nieprawidłowego zapytania `DROP DATABASE`, `DROP TABLE` lub `DELETE`. Takie błędne zapytanie trzeba usunąć z dziennika zdarzeń przed ponownym wykonaniem zapisanych w nim zapytań.

Nie używaj narzędzia `mysqlbinlog` lub `mysql` do przetwarzania pojedynczo plików binarnego dziennika zdarzeń. Między plikami mogą występować pewne zależności, które nie będą dotrzymane w przypadku przetwarzania pojedynczych plików zamiast ich grupy. Na przykład, tabela tymczasowa utworzona w jednym pliku może być później używana w innym. Jeżeli pliki będziesz przetwarzać oddzielnie, wszystkie tabele tymczasowe utworzone przez poszczególne dzienniki będą usunięte po zakończeniu pracy przez dane wywołanie `mysql` i tym samym staną się niedostępne w kolejnych plikach dziennika zdarzeń.

W celu wyodrębnienia zapytań dotyczących jedynie wskazanej bazy danych w narzędziu `mysqlbinlog` użyj opcji `--database`:

```
% mysqlbinlog --database=nazwa_bazy_danych binlog.[0-9]* | mysql
```

Narzędzie `mysqlbinlog` obsługuje także wiele innych opcji pozwalających na wyodrębnianie zapytań wykonanych we wskazanym przedziale czasu (na przykład zapytań wykonanych po utworzeniu danej kopii zapasowej). Być może trzeba będzie przeanalizować zawartość plików dziennika zdarzeń i przekonać się, jakie należy dostarczyć wartości dla opcji. Poniżej przedstawiono przykładowe dane wyjściowe narzędzia `mysqlbinlog` (pewne wiersze komentarzy zostały skrócone, aby zmieściły się na stronie):

```
...
# at 1077
#121030 16:50:36 server id 1  end_log_pos 106  Query....
SET TIMESTAMP=1351633836;
INSERT INTO absence VALUES (3,'2012-09-03');
# at 1183
#121030 16:50:36 server id 1  end_log_pos 1210  Xid = 386
COMMIT;
# at 1210
#121030 16:50:36 server id 1  end_log_pos 106  Query....
SET TIMESTAMP=1351633836;
INSERT INTO absence VALUES (5,'2012-09-03');
# at 1316
#121030 16:50:36 server id 1  end_log_pos 1343  Xid = 387
COMMIT;
# at 1343
#121030 16:50:36 server id 1  end_log_pos 107  Query....
SET TIMESTAMP=1351633836;
INSERT INTO absence VALUES (10,'2012-09-06');
# at 1450
#121030 16:50:36 server id 1  end_log_pos 1477  Xid = 388
COMMIT;
...
```

Przyjmujemy założenie, że chcesz ponownie wprowadzić zapytania, które w binarnym dzienniku zdarzeń zostały zapisane po dacie 2012-10-20 16:50:36. Wartość konieczną do użycia w opcji `--start-datetime` można podać w jednym z dwóch formatów:

```
% mysqlbinlog --start-datetime="2012-10-30 16:50:36" binlog.[0-9]* | mysql
% mysqlbinlog --start-datetime=20121030165036 binlog.[0-9]* | mysql
```

Istnieje również opcja `--stop-datetime`, pozwalająca na podanie daty końcowej; zapytania zapisane po tej dacie nie będą ponownie wykonane. Narzędzie `mysqlbinlog` oferuje także inne opcje pozwalające na wskazanie położenia w dzienniku zdarzeń. Więcej informacji na ten temat znajdziesz w opisie narzędzia `mysqlbinlog` przedstawionym w dodatku F, zatytułowanym „Przewodnik po programie MySQL”.

14.7.4. Rozwiązywanie problemów związanych z automatyczną naprawą w InnoDB

Jeżeli awarii ulegnie serwer MySQL lub komputer, w którym on działa, wtedy po ponownym uruchomieniu MySQL silnik bazy danych InnoDB przeprowadzi procedurę automatycznej naprawy. W tym punkcie dowiesz się, co należy zrobić w rzadkich przypadkach, gdy automatyczna naprawa zakończy się niepowodzeniem.

Jeżeli podczas uruchamiania serwera silnik InnoDB wykryje niemożliwy do usunięcia problem, informacje o tym zapisze w dzienniku błędów i zakończy działanie serwera. W takim przypadku zmiennej systemowej `innodb_force_recovery` trzeba przypisać wartość niezerową z zakresu od 1 do 6, wymuszając w ten sposób uruchomienie serwera. Aby ustawić wartość zmiennej, umieść w grupie `[mysqld]` pliku opcji odpowiedni wiersz i następnie ponownie uruchom serwer:

```
[mysqld]
innodb_force_recovery=poziom
```

Dla niższych wartości *poziom* silnik InnoDB używa bardziej konserwatywnych strategii. Typową zalecaną wartością początkową jest 4. Po ustawieniu wymienionej zmiennej silnik InnoDB pozwala na dodawanie i usuwanie tabel, ale ich zawartość jest w trybie tylko do odczytu. Po uruchomieniu serwera utwórz kopię zapasową tabel InnoDB za pomocą narzędzia `mysqldump`, aby zebrać maksymalną ilość informacji. Następnie w pliku opcji usuń wiersz ustawiający zmienną `innodb_force_recovery` i ponownie uruchom serwer, aby móc przywrócić zawartość tabel: usuwaj je, a następnie przywracaj z plików utworzonych przez `mysqldump`. Ta procedura spowoduje ponowne utworzenie tabel w postaci zapewniającej ich wewnętrzną spójność i może okazać się wystarczającą do przywrócenia danych.

Jeżeli używasz serwera MySQL 5.6.3 lub nowszego, innym użytecznym ustawieniem poza `innodb_force_recovery` jest `innodb_force_load_corrupted`. Ta druga zmienna nakazuje silnikowi InnoDB wczytanie tabel, nawet jeśli są oznaczone jako uszkodzone. W pewnych sytuacjach to pozwoli na utworzenie kopii zapasowej tabel, które normalnie są ignorowane.

Jeżeli wystąpi konieczność przywrócenia wszystkich tabel InnoDB, wtedy trzeba skorzystać z utworzonej kopii zapasowej. Odpowiednie podejście zależy od rodzaju posiadanej kopii zapasowej:

- Jeżeli utworzyłeś binarną kopię zapasową, powinieneś mieć kopie plików przestrzeni tabel (systemowej oraz ewentualnie poszczególnych tabel), plików dziennika zdarzeń InnoDB, plików *.frm* dla każdej tabeli i pliku opcji definiującego konfigurację InnoDB. Po upewnieniu się, że serwer został zatrzymany, usuń wszystkie istniejące pliki InnoDB i zastąp je plikami z kopii zapasowej. Następnie upewnij się, że bieżąca konfiguracja serwera jest taka sama jak w pliku opcji znajdującym się w kopii zapasowej, i uruchom ponownie serwer.
- Jeżeli kopię zapasową tabel InnoDB utworzyłeś za pomocą narzędzia `mysqldump`, powinieneś ponownie zainicjalizować systemową przestrzeń tabel i dzienniki zdarzeń InnoDB. Następnie zawartość pliku kopii trzeba wstawić do InnoDB:
 - ◆ Zatrzymaj serwer i usuń wszystkie istniejące pliki, które mają związek z InnoDB — pliki przestrzeni tabel (systemowej oraz ewentualnie poszczególnych tabel), pliki dziennika zdarzeń InnoDB, pliki *.frm* dla wszystkich tabel InnoDB.

- ◆ Skonfiguruj systemową przestrzeń tabel w taki sam sposób jak wcześniej i ponownie uruchom serwer. Silnik InnoDB ponownie utworzy systemową przestrzeń tabel i pliki dziennika zdarzeń. Informacje na ten temat znajdziesz w punkcie 12.5.3.1, zatytułowanym „Konfiguracja przestrzeni tabel InnoDB”. Pamiętaj, że inicjalizacja przestrzeni tabel to proces składający się z dwóch kroków, jeśli używane są całe partycje.
- ◆ Ponownie wczytaj zawartość pliku lub plików kopii zapasowej, używając ich jako danych wejściowych dla klienta mysql. W ten sposób nastąpi odtworzenie tabel InnoDB.

Po przywróceniu tabel InnoDB z kopii zapasowej ponownie wykonaj zapytania z plików binarnego dziennika zdarzeń, które zostały w nim zapisane już po utworzeniu ostatniej kopii zapasowej. (Zapoznaj się z punktem 14.7.3, zatytułowanym „Ponowne wykonanie zapytań zapisanych w plikach binarnego dziennika zdarzeń”). To będzie łatwiejsze zadanie, jeśli przywracasz tabele InnoDB jako część procesu przywracania całego zestawu baz danych. W takim przypadku można ponownie wykonać wszystkie zapytania zapisane w dzienniku zdarzeń po utworzeniu ostatniej kopii zapasowej. Jeśli przywracasz jedynie wybrane tabele InnoDB, wykorzystanie dziennika zdarzeń jest znacznie trudniejsze, ponieważ trzeba wyszukać i wyodrębnić zapytania jedynie dla wybranych tabel.

14.8. Konfiguracja serwerów replikacji

Jedną z form „replikacji” bazy danych to po prostu skopiowanie bazy danych z jednego serwera do innego. Jednak jeśli źródłowa baza danych ulega zmianom, a chcesz mieć aktualną kopię, operację kopiowania trzeba później powtarzać. W celu zapewnienia nieustannego uaktualniania drugiej bazy danych zmianami wprowadzonymi w głównej bazie danych należy wykorzystać oferowane przez MySQL możliwości w zakresie replikacji „na żywo”. W ten sposób zachowujesz kopię bazy danych, która jest automatycznie uaktualniana zmianami wprowadzanymi w oryginalnej bazie danych.

14.8.1. Jak działa replikacja?

Replikacja bazy danych działa w następujący sposób:

- W relacji replikacji jeden z serwerów jest główny, natomiast drugi podległy. Każdemu z nich musi być przypisany unikalny identyfikator replikacji.
- Serwer główny może mieć więcej niż tylko jeden serwer podległy. Z kolei serwer podległy może działać w charakterze serwera głównego dla innego serwera podległego. W ten sposób powstaje łańcuch serwerów replikacji.
- Każdy serwer podległy musi na początku mieć bazy danych zsynchronizowane z serwerem głównym. Oznacza to, że w chwili rozpoczęcia replikacji wszystkie bazy danych replikowane w serwerze podległym muszą być identycznymi

kopiami baz danych serwera głównego. Następnie uaktualnienia wprowadzane w serwerze głównym są przekazywane do serwera podległego. Uaktualnienia nie powinny być wprowadzane bezpośrednio w bazach danych znajdujących się w serwerze podległym.

- Przekazywanie uaktualnień opiera się na binarnym dzienniku zdarzeń serwera głównego, w którym są zapisywane uaktualnienia wysyłane do klienta. Dlatego też w serwerze głównym musi być włączony binarny dziennik zdarzeń. Uaktualnienia przechowywane w binarnym dzienniku są nazywane „zdarzeniami”.
- Każdy serwer podległy musi mieć zezwolenie na połączenie z serwerem głównym i żądanie uaktualnień. Kiedy serwer podległy nawiąże połączenie z głównym, informuje go, na którym zapytaniu w binarnym dzienniku zdarzeń zakończył przetwarzanie w trakcie poprzedniej sesji. Postęp przetwarzania uaktualnień jest wyrażany w postaci koordynatu replikacji: nazwy pliku binarnego dziennika zdarzeń oraz położenia w tym pliku. Następnie serwer główny wysyła podległemu zdarzenia z binarnego dziennika, które wystąpiły po podanym koordynacie. Gdy serwer podległy odczyta wszystkie dostępne zdarzenia, wstrzymuje replikację i czeka na kolejne zdarzenia.
- Nowe zdarzenie występuje w serwerze głównym, który z kolei zapisuje je w binarnym dzienniku zdarzeń, aby mogło być przekazane serwerom podległym.
- Serwer główny obsługuje połączone serwery podległe podobnie jak zwykłe klienty. Tego rodzaju połączenie zwiększa wartość licznika definiowanego przez zmienną systemową `max_connections`.
- Po stronie serwera podległego serwer używa dwóch wątków do obsługi replikacji. Wątek operacji wejścia-wyjścia otrzymuje z serwera głównego zdarzenia do przetworzenia i zapisuje je w dzienniku przekazywania serwera podległego. Wątek SQL odczytuje zdarzenia z dziennika przekazywania i wykonuje je. Wspomniany dziennik przekazywania jest mechanizmem, za pomocą którego wątek operacji wejścia-wyjścia przekazuje wątkowi SQL zmiany do wprowadzenia. Po przetworzeniu każdego pliku dziennika przekazywania serwer podległy automatycznie go usuwa. Wątki operacji wejścia-wyjścia i SQL działają niezależnie, a więc mogą być uruchamiane i zatrzymywane zupełnie niezależnie od siebie. Podział wspomnianych funkcji między dwa różne wątki przynosi pewne ważne korzyści. Na przykład, wątek wejścia-wyjścia może kontynuować odczyt zdarzeń z serwera podległego, gdy działanie wątku SQL jest zatrzymane. W takim przypadku w trakcie tworzenia kopii zapasowej bazy danych serwera podległego nie są wprowadzane w niej żadne zmiany.

Obsługa replikacji jest aktywnie rozwijana i czasami trudno jest śledzić zmiany i nowo wprowadzane funkcje powiązane z replikacją. Powinieneś pamiętać o ewentualnych problemach związanych ze zgodnością różnych wersji serwera. Ogólnie rzecz biorąc, zaleca się stosowanie do wymienionych poniżej wskazówek:

- W ramach danej serii MySQL (5.5, 5.6 itd.) warto używać najnowszej dostępnej wersji serwera. W ten sposób masz dostęp do najnowszych funkcji i jednocześnie zwiększa się prawdopodobieństwo, że usunięto w nich większość problemów i ograniczeń.
- Spróbuj dopasować formaty binarnych dzienników zdarzeń w serwerach głównym i podległym. Na przykład, spróbuj dopasować serwer główny w wersji 5.5 do serwera podległego w wersji 5.5, a nie serwer główny 5.5 z serwerem podległym 5.1 i na odwrót. Jeżeli serwery główny i podległy muszą być w innych wersjach, replikację przeprowadzaj ze starszej wersji serwera głównego do nowszej wersji serwera podległego, ale nie na odwrót.
- Serwery muszą być zgodne w zakresie oferowanych funkcji. Na przykład, jeśli serwer główny używa tabeli silnika bazy danych niedostępnego w serwerze podległym, wtedy na pewno napotkasz problemy.

14.8.2. Utworzenie relacji typu główny – podległy

Poniższa procedura pokazuje, w jaki sposób skonfigurować replikację typu główny – podległy między dwoma serwerami:

1. Ustal wartości identyfikatorów przypisywanych poszczególnym serwerom i zapisz je w plikach opcji odczytywanych przez serwery w trakcie ich uruchamiania. Wspomniane identyfikatory muszą być różne i w postaci liczby całkowitej z zakresu od 1 do $2^{32}-1$. Wartości identyfikatorów trzeba umieścić w opcji startowej `server-id` używanej przez dany serwer. Poza tym, jeśli w serwerze głównym nie włączono binarnego dziennika zdarzeń, to należy go włączyć. Aby wymienione zadania wykonać w serwerach odpowiednio głównym i podległym, użyj grup opcji przedstawionych w poniższych wierszach:

```
[mysqld]
server-id=identyfikator_serwera_głównego
log-bin=nazwa_binarnego_dziennika_zdarzeń
```

```
[mysqld]
server-id=identyfikator_serwera_podległego
```

Uruchom ponownie oba serwery, aby wprowadzone zmiany zostały zastosowane.

2. W serwerze głównym skonfiguruj konto użytkownika dla serwera podległego, za pomocą którego będzie on nawiązywał połączenie z serwerem głównym i pobierał uaktualnienia:

```
CREATE USER 'uzytkownik_dla_serwera_podległego'@'komputer_serwera_podległego'
  IDENTIFIED BY 'hasło_dla_serwera_podległego';
GRANT REPLICATION SLAVE ON *.* TO
  'uzytkownik_dla_serwera_podległego'@'komputer_serwera_podległego';
```

Zachowaj wartości `uzytkownik_dla_serwera_podległego`

i `hasło_dla_serwera_podległego`, ponieważ będą później potrzebne, aby serwer podległy wiedział, jak nawiązać połączenie z głównym. Nie są wymagane żadne

inne uprawnienia, jeśli konto użytkownika będzie używane jedynie w celu przeprowadzania replikacji. Nadanie kolejnych uprawnień może okazać się konieczne, jeśli konto użytkownika ma być używane do „ręcznego” nawiązywania połączenia z serwera podległego z głównym za pomocą klienta mysql w celach testowych. W takim przypadku przecież nie chcesz mieć ograniczonych możliwości. (Na przykład, jeśli REPLICATION SLAVE to jedyne uprawnienie nadane użytkownikowi; nie będzie on nawet w stanie wykonać zapytania SHOW DATABASES w celu wyświetlenia nazw baz danych znajdujących się w serwerze głównym).

3. Nawiąż połączenie z serwerem głównym i ustal stan jego koordynatu replikacji, wykonując zapytanie SHOW MASTER STATUS:

```
mysql> FLUSH TABLES; SHOW MASTER STATUS;
+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| binlog.000093 | 1707    |              |                  |
+-----+-----+-----+-----+
```

Zapamiętaj wartości File i Position. Będą później potrzebne w celu wskazania serwerowi podległemu miejsca, od którego powinien zacząć odczytywać zdarzenia binarnego dziennika w serwerze głównym.

Bardzo ważne: upewnij się, że żadne zmiany w serwerze głównym nie zostaną wprowadzone od chwili ustalenia koordynatu replikacji aż do chwili wykonania migawki jego danych przekazywanej do serwera podległego.

4. Serwer podległy musi rozpocząć pracę, mając dokładną kopię replikowanych baz danych. Przeprowadź początkową synchronizację serwerów głównego i podległego, po prostu kopiując bazy danych z serwera głównego do podległego. Jednym z rozwiązań jest utworzenie kopii zapasowej w komputerze serwera głównego, przeniesienie jej do komputera serwera podległego i następnie wczytanie w serwerze podległym. Inna metoda polega na skopiowaniu przez sieć wszystkich baz danych z serwera głównego do podległego. Tworzenie kopii zapasowej i techniki kopiowania baz danych omówiono we wcześniejszej części rozdziału. Jeżeli w serwerze głównym nie utworzono jeszcze żadnych baz danych i tabel, ten krok można pominąć, ponieważ nie ma żadnych danych, które trzeba skopiować.
5. Nawiąż połączenie z serwerem podległym i wykonaj zapytanie CHANGE MASTER w celu konfiguracji parametrów nawiązywania połączenia z serwerem głównym i podania początkowego koordynatu replikacji.

```
CHANGE MASTER TO
  MASTER_HOST = 'komputer_serwera_głównego',
  MASTER_USER = 'uzytkownik_dla_serwera_podległego',
  MASTER_PASSWORD = 'hasło_dla_serwera_podległego',
  MASTER_LOG_FILE = 'nazwa_pliku_dziennika',
  MASTER_LOG_POS = położenie_w_pliku_dziennika
```

W powyższym zapytaniu podano nazwę komputera, w którym działa serwer główny (komputer_serwera_głównego). Z kolei wartości uzytkownik_dla_serwera_podległego

hasło_dla_serwera_podległego to nazwa użytkownika i hasło konta utworzonego wcześniej w serwerze głównym dla serwera podległego w celu nawiązania połączenia z serwerem głównym i przeprowadzania replikacji. Wartości *nazwa_pliku_dziennika* i *położenie_w_pliku_dziennika* są pobierane z danych wyjściowych zapytania `SHOW MASTER STATUS`.

W systemach UNIX serwer MySQL używa pliku gniazda dla połączeń localhost, ale replikacja za pomocą pliku gniazda nie jest obsługiwana. Dlatego też, jeśli komputer serwerów głównego i podległego jest fizycznie tym samym komputerem, zamiast localhost trzeba użyć adresu 127.0.0.1 i upewnić się, że serwer podległy używa połączenia TCP/IP.

Jeżeli serwer główny nie nasłuchuje na porcie domyślnym, w zapytaniu `CHANGE MASTER` umieść opcję `MASTER PORT` i wskaż numer portu.

6. Włącz replikację w serwerze podległym:

```
START SLAVE;
```

Serwer podległy powinien nawiązać połączenie z głównym i rozpocząć replikację. Możesz to sprawdzić za pomocą zapytania `SHOW SLAVE STATUS` wykonanego w serwerze podległym.

Parametry zapytania `CHANGE MASTER` serwer podległy przechowuje w pliku o nazwie *master.info* umieszczonym w katalogu danych. W wymienionym pliku jest zapisywany początkowy stan replikacji, a dane w pliku są uaktualniane w trakcie przeprowadzania replikacji. Jeżeli zajdzie potrzeba późniejszej zmiany parametrów replikacji, nawiąż połączenie z serwerem podległym i ponownie wykonaj zapytanie `CHANGE MASTER`. Serwer podległy automatycznie uaktualni plik *master.info* wprowadzonymi zmianami.

Informacje przechowywane w pliku *master.info* to między innymi nazwa użytkownika i hasło pozwalające na nawiązanie połączenia z serwerem głównym. Te informacje powinny pozostać tajne, a więc upewnij się, że plik jest dostępny jedynie dla administratora MySQL w serwerze podległym. Na przykład, zabezpiecz katalog danych MySQL i jego zawartość w sposób omówiony w punkcie 13.1.2, zatytułowanym „Zabezpieczenie instalacji MySQL”.

Przedstawioną tutaj procedurę oparto na założeniu, że z serwera głównego do podległego mają być replikowane wszystkie bazy danych. Jednak istnieje prawdopodobieństwo, że nie będziesz chciał używać tych samych kont użytkowników w obu serwerach. (Na przykład, możesz skonfigurować prywatny serwer podległy replikacji, z którym użytkownicy nie mogą nawiązać połączenia, nawet jeśli mają konto w serwerze głównym). Aby zachować rozdzielność kont użytkowników w serwerach głównym i podległym, musisz wykonać dwie operacje:

1. Podczas kopiowania do serwera podległego migawki danych serwera głównego nie kopiuj bazy danych `mysql` lub utwórz kopię zapasową bazy danych `mysql` w serwerze podległym, a następnie przywróć ją po operacji skopiowania danych z serwera głównego.

2. Nakaz serwerowi podległemu ignorowanie wszelkich uaktualnień w serwerze głównym dotyczących bazy danych mysql. W tym celu należy umieścić poniższą opcję w pliku opcji serwera podległego:

```
[mysql]
replicate-wild-ignore-table=mysql.%
```

Jeżeli serwer podległy ma ignorować tabele w wielu bazach danych, wymienioną opcję można stosować wielokrotnie, jeden raz dla każdej bazy danych.

Istnieje możliwość wykluczenia baz danych po stronie serwera głównego, a nie podległego. W tym celu należy umieścić opcję `binlog-ignore-db` w pliku opcji serwera głównego. W ten sposób zmniejsza się ruch sieciowy między serwerami głównym i podległym, ale jednocześnie binarny dziennik zdarzeń nie zawiera informacji dotyczących ignorowanych baz danych. Wspomniane informacje będą potrzebne w celu przeprowadzenia operacji odzyskiwania danych w serwerze głównym po wystąpieniu awarii. Dlatego też preferowanym rozwiązaniem jest wykluczanie baz danych po stronie serwera podległego.

Po skonfigurowaniu i uruchomieniu aplikacji wiele zapytań będzie użytecznych do monitorowania i kontrolowania serwerów głównego i podległego. Szczegółowe omówienie wspomnianych zapytań znajdziesz w dodatku E, zatytułowanym „Przewodnik po składni SQL”. Poniżej przedstawiono jedynie ich krótkie podsumowanie:

- Zapytanie `SHOW SLAVE STATUS` wykonane w serwerze podległym wyświetla informacje o tym, czy replikacja jest włączona, oraz podaje jej aktualne koordynaty. Wspomniane koordynaty można wykorzystać do ustalenia, które pliki binarnego dziennika zdarzeń w serwerze głównym nie są dłużej potrzebne.
- Zapytanie `PURGE BINARY LOGS` wykonane w serwerze głównym powoduje usunięcie plików binarnego dziennika zdarzeń. To zapytanie możesz wykonać po wykonaniu `SHOW SLAVE STATUS` w każdym serwerze podległym i ustaleniu nazw niepotrzebnych plików binarnego dziennika zdarzeń.
- Zapytania `STOP SLAVE` i `START SLAVE` powodują zawieszenie i wznowienie replikacji w serwerze podległym. Wymienione zapytania są użyteczne do wstrzymania replikacji, na przykład w trakcie wykonywania kopii zapasowej. (Patrz punkt 14.8.4, zatytułowany „Użycie serwera podległego replikacji do tworzenia kopii zapasowych”).

Jak wcześniej wspomniano, serwery podległe wewnętrznie używają dwóch wątków do obsługi replikacji. Wątek operacji wejścia-wyjścia komunikuje się z serwerem głównym, otrzymuje od niego uaktualnienia i zapisuje je w dzienniku przekazywania. Z kolei wątek SQL odczytuje wspomniany dziennik przekazywania i wykonuje znalezione tam zapytania. Możesz wykorzystać zapytania `STOP SLAVE` i `START SLAVE` do zawieszenia i wznowienia poszczególnych wątków przez dodanie na końcu zapytania `IO_THREAD` lub `SQL_THREAD`. Na przykład, zapytanie `STOP SLAVE SQL_THREAD` powoduje wstrzymanie wykonywania przez serwer podległy uaktualnień zapisanych w dzienniku przekazywania, choć serwer podległy nadal może je pobierać z serwera głównego i zapisywać w dzienniku przekazywania.

Pliki dziennika przekazywania są generowane w postaci ponumerowanej sekwencji, podobnie jak pliki binarnego dziennika zdarzeń. Istnieje również plik indeksu dziennika przekazywania, analogiczny do pliku indeksu dziennika binarnego. Domyślnie nazwy plików dziennika przekazywania i indeksu to *HOSTNAME-relay-bin.nnnnnnn* i *HOSTNAME-relay-bin.index*. Oba wymienione pliki znajdują się w katalogu danych MySQL. Wymienione nazwy domyślne można zmienić za pomocą opcji startowych serwera `--relay-log` i `--relay-log-index`. Więcej informacji na ten temat znajdziesz w punkcie 12.8.5, zatytułowanym „Dziennik przekazywania”.

14.8.3. Formaty rejestracji zdarzeń w dzienniku binarnym

Domyślnie zdarzenia modyfikujące dane są przez serwer zapisywane w binarnym dzienniku w postaci zapytań SQL. To nosi nazwę rejestrowania binarnego na podstawie zapytań (a oparta na takim dzienniku replikacja jest nazywana replikacją opartą na zapytaniach). Serwer może także rejestrować zmiany wprowadzane w poszczególnych rekordach danych. To nosi nazwę rejestracji opartej na rekordach. Trzeci rodzaj rejestracji to format mieszany, który pozwala serwerowi na przełączanie się między rejestracją na podstawie zapytań i rekordów. Serwer po prostu wybiera rozwiązanie najlepsze w danej chwili.

Ogólnie rzecz biorąc, rejestracja oparta na zapytaniach powoduje generowanie mniejszych plików dzienników o treści łatwiejszej do zrozumienia. Z kolei rejestracja oparta na rekordach pozwala na znacznie większą kontrolę podczas wprowadzania uaktualnień, co się przydaje w replikacji, gdy oryginalne zapytanie może być niedeterministyczne i generować różne efekty w serwerach głównym i podległym.

Wskazanie formatu rejestracji danych umożliwia zmienną systemową `binlog_format`, odczytywana podczas uruchamiania serwera oraz w trakcie jego działania. Dozwolone wartości wymienionej zmiennej to `STATEMENT`, `ROW` i `MIXED`.

14.8.4. Użycie serwera podległego replikacji do tworzenia kopii zapasowych

Jeżeli masz skonfigurowany serwer podległy replikacji, może on okazać się pomocny podczas rozwiązywania konfliktów interesów pojawiających się, gdy wykonujesz obowiązki administratora MySQL:

- Z jednej strony, bardzo ważne jest zapewnienie maksymalnego czasu działania i dostępności serwera dla członków społeczności, co obejmuje zapewnienie im możliwości przeprowadzania uaktualnień bazy danych.
- Z drugiej strony, bardzo ważne jest tworzenie kopii zapasowej, a to zadanie najlepiej przeprowadzić, gdy nikt nie może wprowadzać jakichkolwiek zmian w bazie danych. Ponadto, w trakcie operacji odzyskiwania danych kopie zapasowe są najużyteczniejsze, jeśli zapewniona jest synchronizacja plików kopii zapasowej i punktów kontrolnych w plikach dziennika zdarzeń. To wymaga zatrzymania serwera bądź nałożenia blokady na wszystkie tabele.

Powodem wspomnianych konfliktów na tle zapewnienia dostępności jest częściowa lub całkowita utrata przez klienty dostępu do bazy danych w trakcie tworzenia jej kopii zapasowej. Serwer podległy replikacji zapewnia pewne rozwiązanie takiego problemu. Kopię zapasową twórz w serwerze podległym replikacji zamiast w głównym. Przed rozpoczęciem operacji zatrzymaj lub wstrzymaj działanie serwera podległego. Po zakończeniu tworzenia kopii ponownie uruchom serwer lub wznów jego działanie, a replikacja będzie kontynuowana i zastosuje wszystkie uaktualnienia wprowadzone w serwerze głównym podczas tworzenia kopii zapasowej w serwerze podległym. W ten sposób nie musisz zatrzymywać serwera głównego lub w inny sposób ograniczać klientom dostępu do niego podczas tworzenia kopii zapasowej.

Poniżej przedstawiono pewne potencjalne strategie dotyczące tworzenia kopii zapasowej w serwerze podległym replikacji:

- W przypadku tworzenia binarnej kopii zapasowej wszystkich danych serwera podległego zatrzymaj go, a następnie wykonaj polecenia przedstawione w podpunkcie 14.4.3.1, zatytułowanym „Utworzenie pełnej binarnej kopii zapasowej”, i ponownie uruchom serwer.
- W przypadku tworzenia kopii zapasowej za pomocą narzędzia takiego jak `mysql dump`, które nie wymaga zatrzymania serwera podległego, operację można przeprowadzić po zatrzymaniu wątku SQL replikacji (wykonaj zapytanie `STOP SLAVE SQL_THREAD` i opróżnij dzienniki zdarzeń) i wznów go po wykonaniu kopii zapasowej. Teraz można przystąpić do tworzenia kopii zapasowej, a później wznówić replikację przez wykonanie zapytania `START SLAVE`. W ten sposób serwer podległy nie będzie wprowadzał żadnych zmian w bazie danych podczas tworzenia jej kopii zapasowej. Wątek operacji wejścia-wyjścia może działać nadal, w dzienniku przekazywania będzie zapisywał zdarzenia otrzymane z serwera głównego. Po ponownym uruchomieniu wątku SQL, już po utworzeniu kopii zapasowej, wykona on wszelkie zapytania zebrane w dzienniku przekazywania.

W powyższym podejściu przyjęto założenie, że klient nie wprowadza żadnych uaktualnień w serwerze podległym. Tej metody nie powinien również stosować, jeśli zamierzasz używać techniki binarnego tworzenia kopii zapasowej przez bezpośrednie kopiowanie plików bazy danych. Nawet pomimo zatrzymania wątku SQL w buforze mogą znajdować się pewne dane, które jeszcze nie zostały zapisane na dysku.

- Pewne metody tworzenia kopii zapasowej nie wymagają nawet zawieszenia replikacji. Na przykład, jeśli stworzysz kopię zapasową pojedynczej bazy danych zawierającej tylko tabelę MyISAM, możesz użyć narzędzia `mysql dump` wraz z odpowiednimi opcjami w celu nałożenia blokad na wszystkie tabele jednocześnie. W takim przypadku serwer podległy może nadal funkcjonować, ale nie będzie mógł wprowadzić jakichkolwiek zmian w tabelach podczas tworzenia kopii zapasowej. Po zakończeniu tworzenia kopii zapasowej nastąpi zwolnienie blokad i serwer automatycznie przetworzy wszystkie oczekujące uaktualnienia.
- Kopię zapasową należy tworzyć, gdy serwer działa w trybie tylko do odczytu.

Poniższa procedura pokazuje, jak zaimplementować strategię przedstawioną powyżej:

1. Uniemożliwienie uaktualniania bazy danych w serwerze podległym przez ustawienie jej trybu tylko do odczytu:

```
mysql> FLUSH TABLES WITH READ LOCK;  
mysql> SET GLOBAL read_only = ON;
```
2. Utworzenie kopii zapasowej, na przykład za pomocą narzędzia `mysql dump`.
(Ta technika nie działa w przypadku tworzenia binarnej kopii zapasowej przez bezpośrednie kopiowanie plików i katalogów, ponieważ transakcyjny silnik bazy danych może zawierać w buforze pewne transakcje, które nie zostały jeszcze zapisane na dysku).
3. Ponowne włączenie możliwości wprowadzania uaktualnień w serwerze podległym:

```
mysql> SET GLOBAL read_only = OFF;  
mysql> UNLOCK TABLES;
```


Oprogramowanie wymagane do użycia tej książki

W tym dodatku dowiesz się, skąd pobrać dystrybucję `sampdb` potrzebną do przygotowania przykładowej bazy danych używanej w przykładach zaprezentowanych w książce. Do użycia wymienionej dystrybucji trzeba mieć działający serwer MySQL. Z lektury dodatku dowiesz się również, skąd pobrać MySQL oraz powiązane z nim oprogramowanie, takie jak moduły Perl DBI i CGI.pm, PHP oraz Apache. Przedstawiono też krótkie wprowadzenie do instalacji MySQL oraz wymaganego oprogramowania Perl i PHP, a także wskazówki dotyczące konfiguracji serwera Apache do użycia wraz z PHP.

W przypadku instalacji każdego pakietu oprogramowania dokładnie zapoznaj się z informacjami dostarczonymi wraz z pakietem oraz sprawdź inne źródła, o ile zachodzi potrzeba. Na przykład, podręcznik MySQL zawiera rozdział dokładnie omawiający procedurę instalacji serwera.

A.1. Pobranie dystrybucji `sampdb` zawierającej przykładową bazę danych

Dystrybucja `sampdb` jest dostępna na stronie <http://www.kitebird.com/mysql-book/> i zawiera wszystkie pliki niezbędne do przygotowania przykładowej bazy danych `sampdb` oraz uzyskania do niej dostępu. Dystrybucja jest dostępna w postaci skompresowanego pliku `.tar` oraz jako archiwum w formacie `.zip`. W celu rozpakowania dystrybucji w formacie `.tar` należy wydać jedno z poniższych poleceń (drugie z wymienionych zastosuj, gdy używana przez Ciebie wersja narzędzia `tar` nie rozpoznaje opcji `z`):

```
% tar xzf sampdb.tar.gz
% gunzip < sampdb.tar.gz | tar xf -
```

Z kolei rozpakowanie dystrybucji w formacie `.zip` wymaga użycia narzędzia takiego jak WinZip, `pkunzip` lub `unzip`.

Po rozpakowaniu dystrybucji na dysku znajdziesz katalog o nazwie *sampdb*, zawierający wiele plików i podkatalogów:

- Plik *README.txt* zawiera informacje dotyczące użycia dystrybucji. To jest pierwszy plik, do którego powinieneś zajrzeć. Poszczególne podkatalogi dystrybucji również mogą zawierać plik *README.txt* wraz z informacjami dodatkowymi.
- Pliki używane do utworzenia i wczytania bazy danych *sampdb* w rozdziale 1., zatytułowanym „Rozpoczęcie pracy z MySQL”.
- Katalog *capi*, zawierający programy w języku C używane w rozdziale 7., zatytułowanym „Tworzenie programów MySQL przy użyciu języka C”.
- Katalog *perlapi*, zawierający skrypty Perl DBI użyte w rozdziale 8., zatytułowanym „Tworzenie programów MySQL przy użyciu Perl DBI”.
- Katalog *phpapi*, zawierający skrypty PHP użyte w rozdziale 9., zatytułowanym „Tworzenie programów MySQL przy użyciu języka PHP”.
- Katalog *ssl*, zawierający pliki certyfikatu i kluczy pozwalające na konfigurację bezpiecznych połączeń SSL między programami klienckimi MySQL i serwerem. Zapoznaj się z podrozdziałem 13.5, zatytułowanym „Konfiguracja bezpiecznych połączeń za pomocą SSL”.

Katalog *sampdb* zawiera także inne podkatalogi, w których znajdziesz pliki wymienione w książce. Więcej informacji na ich temat zamieszczono w pliku *README.txt*.

A.2. Pobieranie serwera MySQL i powiązanego z nim oprogramowania

Aby użyć niniejszej książki, musisz zainstalować serwer MySQL, o ile jeszcze tego nie zrobiłeś. W zależności od tego, w jaki sposób planujesz uzyskiwanie dostępu do MySQL, może wystąpić potrzeba instalacji oprogramowania dodatkowego:

- W celu tworzenia skryptów Perl uzyskujących dostęp do baz danych MySQL potrzebne są moduły DBI i DBD::mysql. Jeżeli planujesz tworzenie internetowych skryptów DBI, wymagany jest także moduł CGI.pm. Ponadto, będzie potrzebny również serwer WWW.
- W celu utworzenia skryptów PHP omówionych w książce trzeba mieć zainstalowane PHP i rozszerzenie PDO (ang. *PHP Data Objects*). Ponadto, będzie potrzebny również serwer WWW.

W tej książce skrypty internetowe są uruchamiane w serwerze WWW Apache, ale możesz wykorzystać także inny serwer WWW.

Jeżeli korzystasz z usług oferowanych przez ISP (ang. *Internet Service Provider*, dostawca usług internetowych), istnieje duże prawdopodobieństwo, że wszystkie wymienione pakiety są już zainstalowane. W takim przypadku możesz z nich korzystać. W przeciwnym

razie zajrzyj do tabeli A.1, w której wymieniono pakiety i witryny WWW, w których możesz je znaleźć.

Tabela A.1. Pakiety używane w książce i ich strony domowe

Pakiet	Lokalizacja
MySQL	http://dev.mysql.com/
Moduły Perl	http://cpan.perl.org/
PHP	http://www.php.net/
Apache	http://httpd.apache.org/

Wersja instalowanego pakietu będzie zależała od potrzeb:

- W celu zachowania maksymalnej stabilności powinien być konserwatywny i używać ostatniej dostępnej stabilnej wersji pakietu. Takie rozwiązanie przynosi korzyść w postaci dostępności najnowszych funkcji, a jednocześnie pakiet zawiera poprawki ewentualnych błędów. Wersja stabilna nie zawiera kodu eksperymentalnego.
- Jeżeli chcesz mieć absolutnie najnowsze funkcje lub musisz użyć funkcji dostępnej jedynie w najnowszej wersji, skorzystaj z najnowszej rozwojowej wersji pakietu.

Strony domowe poszczególnych pakietów wskazują wersje stabilne i rozwojowe. Zawierają także listę zmian wprowadzonych w poszczególnych wersjach, co pomaga w wyborze najlepszego wydania.

Stabilne wersje MySQL to obecnie seria 5.5, natomiast wersje rozwojowe to seria 5.6. Materiał przedstawiemy w książce opiera się na wersji MySQL 5.5, ale wspomniano także o pewnych funkcjach wprowadzonych we wczesnych wersjach 5.6, dostępnych w trakcie pisania książki. Do obsługi witryny produkcyjnej zalecam użycie wersji stabilnej, a nie rozwojowej. Dlatego też moim zaleceniem dla większości czytelników jest MySQL w wersji 5.5. Jeżeli chcesz poeksperymentować z nowszymi funkcjami, wtedy użyj wydania MySQL 5.6.

Prekompilowane pliki binarne są dostępne dla wielu pakietów instalacyjnych. Dla niektórych platform dystrybucje binarne są zwykle dostępne w rodzimych formatach, na przykład pakietach RPM dla systemu Linux lub pakietach DMG dla systemu OS X. Ponadto, dostępne są bardziej ogólne formaty, na przykład skompresowane pliki *.tar* dla systemu UNIX i archiwa *.zip* dla Windows. Jeżeli preferujesz kompilację oprogramowania ze źródeł lub jeśli dystrybucja binarna nie jest dostępna dla używanej przez Ciebie platformy, wtedy będziesz potrzebował kompilatora C (C++ dla MySQL).

Pewne platformy mają własne systemy pakietów. Bardzo często zdarza się to w systemach UNIX i pochodnych, na przykład Linux. W takich przypadkach istnieje duże prawdopodobieństwo, że możesz użyć wspomnianego systemu pakietów do instalacji całego niezbędnego oprogramowania. Na przykład, możesz wykorzystać system portów FreeBSD, emerge w Gentoo Linux, yum w Red Hat Linux lub apt-get w Debianie.

Pozostała część niniejszego dodatku zawiera informacje dotyczące instalacji MySQL oraz wymaganego oprogramowania Perl i PHP.

A.3. Instalacja MySQL

Instalacja MySQL obejmuje wymienione poniżej kroki:

1. Jeżeli serwer będzie instalowany w systemie UNIX, trzeba wybrać konto logowania używane do uruchamiania MySQL, a następnie utworzyć je, jeśli to konieczne.
2. Pobranie i rozpakowanie dystrybucji do zainstalowania. W przypadku użycia dystrybucji źródłowej trzeba ją skompilować i zainstalować.
3. Konfiguracja zmiennej środowiskowej PATH, aby zawierała katalog bin znajdujący się w katalogu instalacyjnym MySQL. To znacznie ułatwia uruchamianie programów MySQL, ponieważ nie trzeba podawać ich pełnych ścieżek dostępu.
4. Jeżeli nie zamierzasz obsługiwać serwera, a jedynie używać programów klienckich MySQL w celu uzyskania dostępu do serwera obsługiwanego przez kogoś innego, wtedy możesz pominąć pozostałe kroki.
5. Inicjalizacja katalogu danych i tabel uprawnień w bazie danych mysql. Pewne pakiety instalacyjne wykonują to zadanie za Ciebie. Dotyczy to na przykład pakietów RPM w systemie Linux i DMG w systemie OS X. Z kolei w Windows nie ma potrzeby inicjalizacji katalogu danych lub tabel uprawnień, ponieważ są one dostarczane przez dystrybucję w postaci już wstępnie zainicjalizowanej.
6. Uruchomienie serwera.
7. Inicjalizacja innych tabel systemowych w bazie danych mysql.

W kolejnych punktach zostaną przedstawione dodatkowe informacje szczegółowe dotyczące poszczególnych kroków.

Po instalacji MySQL są jeszcze inne zadania, które prawdopodobnie trzeba będzie wykonać:

- Zdefiniowanie haseł dla kont MySQL w celu ich lepszego zabezpieczenia. Instalacja domyślna zezwala na nawiązywanie połączenia z serwerem MySQL bez użycia hasła.
- Konfiguracja automatycznego uruchamiania i zatrzymywania serwera jako części standardowej procedury uruchamiania i zatrzymywania serwera. Ponadto, umieszczenie opcji `--user` w pliku opcji, aby uniknąć konieczności jej podawania w trakcie każdego uruchamiania serwera.
- Włączenie rejestracji zdarzeń, co jest użyteczne podczas monitorowania serwera, konfiguracji replikacji oraz w trakcie procedur odzyskiwania danych.
- Dalsza konfiguracja serwera. Na przykład, możesz włączać lub wyłączać silniki bazy danych bądź też dostosowywać dla nich parametry.

Informacje dotyczące przeprowadzania powyższych oraz innych zadań administracyjnych przedstawiono w rozdziale 12., zatytułowanym „Ogólna administracja bazą danych MySQL”. W szczególności powinieneś zapoznać się z punktami dotyczącymi definiowania haseł, uruchamiania i zatrzymywania serwera oraz jego uruchamiania w ramach konta pozbawionego uprawnień.

A.3.1. Tworzenie konta logowania dla użytkownika MySQL

Ten krok jest niezbędny jedynie wtedy, gdy zamierzasz zajmować się obsługą serwera MySQL. Jeśli planujesz tylko użycie oprogramowania klienckiego MySQL, możesz pominąć ten krok.

W systemach UNIX serwer MySQL może być uruchamiany za pomocą dowolnego konta w systemie. Jednak z powodów bezpieczeństwa i administracyjnych serwera nie należy uruchamiać jako użytkownik root. Zalecam utworzenie oddzielnego konta użytkownika przeznaczonego do administracji MySQL i uruchamianie serwera za pomocą tego konta. W ten sposób możesz się zalogować do systemu jako dany użytkownik i będziesz miał pełne uprawnienia do katalogu danych MySQL, co pozwoli na jego obsługę i usuwanie ewentualnych problemów. Więcej informacji na temat korzyści płynących z użycia dla serwera MySQL oddzielnego konta użytkownika innego niż root znajdziesz w podpunkcie 12.2.1.1, zatytułowanym „Działanie serwera w ramach pozbawionego uprawnień konta logowania”.

W książce przyjęto założenie, że w systemie UNIX nazwa użytkownika i grupy konta przeznaczonego do administracji serwerem MySQL to `mysql`. Jeżeli planujesz instalację MySQL jedynie na własny użytek, wtedy serwer możesz uruchamiać z poziomu własnego konta. W takim przypadku podawaj własną nazwę użytkownika i grupy zamiast używanej w książce nazwy `mysql`.

Procedura tworzenia konta użytkownika zależy od konkretnego systemu. Szczegółowe informacje znajdziesz w dokumentacji używanego systemu. (Jeżeli w systemie Linux do instalacji MySQL użyto pakietów RPM, procedura instalacyjna automatycznie utworzyła konto użytkownika o nazwie `mysql`).

Zanim przystąpisz do tworzenia konta użytkownika przeznaczonego do uruchamiania serwera MySQL, w pierwszej kolejności sprawdź, czy wspomniane konto użytkownika istnieje w systemie. Wiele systemów UNIX zawiera użytkownika i grupę `mysql` wśród standardowych kont użytkowników. Na przykład, system OS X zawiera konto użytkownika `mysql` (co spełnia założenie instalatora MySQL w pakiecie DMG, że konto użytkownika `mysql` już istnieje w systemie).

A.3.2. Instalacja MySQL

W celu instalacji MySQL w Windows zalecam użycie instalatora MySQL. Wspomniany instalator przeprowadzi Cię przez wszystkie kroki wymagane do instalacji MySQL.

Z kolei w systemach UNIX dystrybucje MySQL zawierają jeden lub więcej poniższych komponentów:

- serwer `mysqld`;
- programy klienckie (`mysql`, `mysqladmin` itd.);
- obsługę programowania po stronie klienta (biblioteki C oraz pliki nagłówkowe).

Pewne formaty pakietów mogą dzielić pełną dystrybucję na wiele komponentów, więc upewnij się, że zainstalowałeś wszystkie pakiety dostarczające komponenty wymagane do Twojej pracy. Na przykład, pakiety RPM mogą dzielić dystrybucję na pakiety serwera, klienta i programistyczne. Jeżeli planujesz nawiązywanie połączeń z serwerem działającym w innym komputerze, wtedy nie trzeba instalować serwera. Jednak zawsze należy zainstalować oprogramowanie klienta, aby można było nawiązywać połączenie z używanym serwerem. Jeżeli planujesz tworzenie programów, konieczne jest zainstalowanie pakietu dla programistów.

Istnieje również możliwość utworzenia MySQL na podstawie dystrybucji źródłowej. To znacznie trudniejsze niż instalacja z dystrybucji binarnej, ponieważ wymaga kompilacji oprogramowania. Z drugiej strony, zyskujesz większą kontrolę nad parametrami konfiguracyjnymi. Na przykład, możesz skompilować dystrybucję i umieścić w niej tylko używane silniki bazy danych.

A.3.3. Konfiguracja zmiennej środowiskowej PATH

Powłoka (interpreter wiersza poleceń) używa zmiennej środowiskowej PATH w celu ustalenia miejsca wyszukiwania programów po wprowadzeniu poleceń w wierszu poleceń. Wywoływanie programów MySQL stanie się łatwiejsze, jeśli zmienna PATH będzie zawierała katalog *bin* znajdujący się w katalogu instalacji MySQL. W takim przypadku nazwy poleceń będzie można wpisywać bez ich pełnych ścieżek dostępu.

W systemach UNIX zmienna PATH jest najczęściej definiowana w jednym z plików startowych powłoki, na przykład *.tcshrc* dla *tcsh* bądź też *.bashrc* lub *.bash_profile* dla *bash*. Na przykład, jeśli używasz powłoki *tcsh*, w pliku *.tcshrc* może znajdować się poniższy wiersz:

```
setenv PATH /bin:/usr/bin:/usr/local/bin
```

Jeżeli programy MySQL zostały zainstalowane w */usr/local/mysql/bin*, wtedy wartość zmiennej PATH zmień na następującą:

```
setenv PATH /usr/local/mysql/bin:/bin:/usr/bin:/usr/local/bin
```

Jeśli używasz powłoki *bash*, jeden lub więcej plików startowych powłoki może zawierać poniższy wiersz:

```
PATH=/bin:/usr/bin:/usr/local/bin
```

Zmień go na następujący:

```
PATH=/usr/local/mysql/bin:/bin:/usr/bin:/usr/local/bin
```


Po wprowadzeniu zmian w plikach startowych powłoki nowe ustawienia będą zastosowane po kolejnym zalogowaniu się.

W systemie Windows ustawienie wartości zmiennej PATH odbywa się następująco:

1. Kliknij prawym przyciskiem myszy ikonę *Mój komputer* na pulpicie i wybierz opcję *Właściwości*.
2. Wybierz kartę *Zaawansowane*, a następnie kliknij przycisk *Zmienne środowiskowe....*
3. Z listy *Zmienne systemowe* wybierz Path, a następnie kliknij przycisk *Edytuj....*
4. Dodaj odpowiedni katalog bin, poprzedzając go średnikiem w celu oddzielenia od poprzedniego katalogu na liście. Na przykład, jeśli serwer MySQL został zainstalowany w katalogu *C:\mysql*, dodaj *;C:\mysql\data*.

Konieczne może być otworenie nowego okna konsoli (lub nawet ponowne uruchomienie Windows), aby wprowadzona zmiana została zastosowana.

A.3.4. Inicjalizacja katalogu danych i tabel uprawnień

Zanim będzie można używać instalacji MySQL, konieczne jest przeprowadzenie inicjalizacji katalogu danych oraz tabel uprawnień w bazie danych *mysql*. Wspomniane tabele są odpowiedzialne za kontrolę dostępu do serwera. Ten krok jest niezbędny tylko wtedy, gdy zajmujesz się obsługą MySQL. Możesz go pominąć, jeśli planujesz jedynie użycie oprogramowania klienckiego MySQL. Także w Windows można pominąć ten krok, ponieważ tabele uprawnień są wstępnie zainicjalizowane w dystrybucji dla Windows.

W przedstawionych poniżej poleceniach *DATADIR* oznacza ścieżkę dostępu do katalogu danych. Ten katalog najczęściej znajduje się w katalogu bazowym instalacji MySQL i nosi nazwę *data* lub *var*. Normalnie przedstawione poniżej polecenia wydajesz jako użytkownik *root*. Jeżeli zalogowałeś się jako użytkownik MySQL (na przykład *mysql*) lub zainstalowałeś MySQL w ramach własnego konta użytkownika, ponieważ tylko Ty będziesz używał serwera, wymienione polecenia możesz wykonać bez uprawnień użytkownika *root* i powinieneś pominąć opcję *--user*. Możesz także pominąć polecenia *chown* i *chgrp*.

Aby zainstalować katalog danych, bazę danych *mysql* i domyślne tabele uprawnień, przejdź do katalogu instalacji MySQL i uruchom skrypt *mysql_install_db*. (Nie musisz tego robić po instalacji serwera z pakietów RPM lub pakietu DMG w systemie OS X, ponieważ ich procedury instalacyjne automatycznie uruchamiają wymieniony skrypt za Ciebie). Na przykład, po instalacji MySQL w katalogu */usr/local/mysql* odpowiednie polecenia przedstawiają się następująco:

```
# cd /usr/local/mysql
# scripts/mysql_install_db --user=mysql
```

Po uruchomieniu skryptu *mysql_install_db* zmień użytkownika i grupę, a także uprawnienia dostępu do wszystkich plików znajdujących się w katalogu danych MySQL.

Przyjmując założenie, że nazwą użytkownika i grupy powinno być `mysql`, polecenia przedstawiają się następująco:

```
# chown -R mysql DATADIR
# chgrp -R mysql DATADIR
# chmod -R go-rwx DATADIR
```

Polecenia `chown` i `chgrp` powodują zmianę właściciela i grupy katalogu `DATADIR` — właścicielem staje się użytkownik `mysql`. Z kolei polecenie `chmod` zmienia uprawnienia dostępu do zawartości `DATADIR` — dostęp ma tylko użytkownik `mysql` i nikt poza nim.

Ostatnie kilka kroków to tak naprawdę jest część znacznie obszerniejszej procedury, omówionej szczegółowo w punkcie 13.1.2, zatytułowanym „Zabezpieczenie instalacji MySQL”. W wymienionym punkcie znajdziesz informacje dodatkowe pomagające w zabezpieczeniu instalacji.

Jeżeli działanie skryptu `mysql_install_db` zakończy się niepowodzeniem, wówczas zajrzyj do znajdującego się w podręczniku użytkownika MySQL rozdziału poświęconego instalacji i sprawdź, jak rozwiązać napotkany problem. Jeżeli działanie skryptu `mysql_install_db` nie zakończy się powodzeniem, tabele uprawnień prawdopodobnie będą niekompletne. Powinieneś je usunąć, ponieważ wymieniony skrypt nie próbuje ponownie utworzyć jakichkolwiek znalezionych tabel. Usunięcie całej bazy danych `mysql` następuje po wydaniu polecenia:

```
# rm -rf DATADIR/mysql
```

A.3.5. Uruchamianie serwera

Po zainstalowaniu serwera MySQL można go uruchomić. Procedura jest inna w systemach UNIX i Windows.

A.3.5.1. Uruchamianie serwera w systemach UNIX

Wymienione tutaj polecenia należy wydać z poziomu katalogu instalacji MySQL. Normalnie polecenia wydajesz jako użytkownik `root` i używasz opcji `--user` w celu nakazania uruchomienia serwera jako użytkownik `mysql`. Jednak jeśli zalogowałeś się jako użytkownik MySQL (na przykład `mysql`) lub uruchamiasz serwer w ramach własnego konta użytkownika, wtedy możesz pominąć opcję `--user`.

Przejdź do katalogu bazowego instalacji MySQL (na przykład `/usr/local/mysql`), a następnie użyj skryptu `mysql_safe` do uruchomienia serwera:

```
# cd /usr/local/mysql
# bin/mysqld_safe --user=mysql &
```

W wierszu poleceń lub w pliku opcji możesz podać także inne opcje. Zapoznaj się z punktem 12.2.3, zatytułowanym „Określanie opcji startowych serwera”.

A.3.5.2. Uruchomienie serwera w systemach Windows

Jeżeli serwer MySQL został zainstalowany w innym katalogu niż domyślnie wybrany przez instalator, konieczne jest umieszczenie grupy opcji `[mysqld]` w pliku opcji odczytywanym przez serwer w chwili jego uruchamiania. Dzięki temu serwer będzie mógł ustalić położenie katalogu bazowego oraz katalogu danych MySQL. Plik opcji to najczęściej *my.ini* w katalogu instalacji MySQL lub *C:\my.ini*. Na przykład, po zainstalowaniu MySQL w katalogu *C:\mysql* wspomniana wcześniej grupa opcji powinna przedstawiać się następująco (zwróć uwagę na użycie w ścieżkach dostępu ukośników odwrotnych niż standardowo stosowane w Windows):

```
[mysqld]
basedir=C:/mysql
datadir=C:/mysql/data
```

Jeżeli zainstalowałeś serwer w innym katalogu, w pliku opcji podaj odpowiednie ścieżki dostępu.

W celu ręcznego uruchomienia serwera z poziomu wiersza poleceń przejdź do katalogu zawierającego instalację MySQL i wydaj poniższe polecenie:

```
C:\> mysql
```

W wierszu poleceń lub w pliku opcji możesz podać także inne opcje. Zapoznaj się z punktem 12.2.3, zatytułowanym „Określanie opcji startowych serwera”.

Aby uruchomić serwer w trybie konsoli, w którym komunikaty błędów są wyświetlane w oknie konsoli, wywołaj go wraz z opcją `--console`:

```
C:\> mysql --console
```

Podczas uruchamiania serwera w trybie konsoli inne opcje w wierszu poleceń można podać po opcji `--console` lub w pliku opcji.

Jeżeli uruchomisz serwer z poziomu wiersza poleceń, to znak zachęty może nie być wyświetlony ponownie aż do zakończenia działania serwera. W takim przypadku po prostu otwórz nowe okno konsoli, w którym będziesz mógł uruchamiać programy klienckie MySQL.

Innym sposobem uruchamiania serwera w Windows jest utworzenie usługi dla MySQL uruchamianej automatycznie wraz z Windows. Informacje szczegółowe na ten temat znajdziesz w podpunkcie 12.2.2.2, zatytułowanym „Uruchomienie serwera jako usługi Windows”.

A.3.6. Inicjalizacja innych tabel systemowych

Po uruchomieniu serwera w bazie danych *mysql* znajdują się jeszcze inne tabele systemowe, które opcjonalnie można skonfigurować, na przykład tabele stref czasowych i tabele pomocy po stronie serwera.

Tabele stref czasowych pozwalają na użycie nazw stref czasowych. Aby je skonfigurować, należy zapoznać się z informacjami przedstawionymi w punkcie 12.6.1, zatytułowanym „Konfiguracja obsługi stref czasowych”.

Klient wiersza poleceń `mysql` może uzyskać dostęp do tabel pomocy po stronie serwera za pomocą polecenia `help`. Możliwość użycia wymienionego polecenia wymaga przeprowadzenia konfiguracji tabel pomocy w bazie danych `mysql`. Większość metod instalacyjnych wykonuje to zadanie w trakcie pierwszej instalacji, a bieżące dystrybucje MySQL dla Windows zawierają wstępnie zainstalowane tabele pomocy. Nie powinna występować konieczność ręcznej konfiguracji wspomnianych tabel.

Jeżeli użyta metoda instalacji MySQL nie przeprowadziła konfiguracji tabel pomocy, możesz je wczytać ręcznie. W tym celu upewnij się, że serwer działa. Następnie odszukaj plik `fill_help_tables.sql`, zawierający zapytania SQL odpowiedzialne za utworzenie i wypełnienie tabel pomocy. Wspomniany plik znajduje się prawdopodobnie w katalogu `/usr/share/mysql`, podkatalogu `share` w katalogu bazowym instalacji MySQL lub katalogu `scripts` dystrybucji źródłowej. Po odszukaniu pliku wydaj poniższe polecenie z poziomu katalogu, w którym znalazłeś plik:

```
% mysql -p -u root mysql < fill_help_tables.sql
```

Powyższe polecenie wyświetli pytanie o hasło użytkownika `root` MySQL. Jeśli nie zdefiniowałeś hasła, wtedy pomiń opcję `-p`.

A.4. Informacje dotyczące instalacji Perl DBI

W celu utworzenia skryptów Perl posiadających dostęp do baz danych MySQL konieczne jest zapewnienie dostępu do oprogramowania DBI. Jeżeli jeszcze ich nie ma w systemie, to należy zainstalować następujące komponenty:

- Moduł DBI, zapewniający ogólny sterownik DBI, oraz moduł `DBD::mysql`, zapewniający sterownik przeznaczony dla MySQL. DBI wymaga Perla w wersji przynajmniej 5.6.0 (zalecana wersja to 5.6.1). Począwszy od DBI 1.6.11, minimalną wersją Perl jest 5.8.1. (Jeżeli nie masz zainstalowanego Perla, przejdź na witrynę <http://www.perl.org/>, pobierz dystrybucję Perl i zainstaluj ją przed instalacją oprogramowania DBI).
- Utworzona w języku C biblioteka klienta oraz pliki nagłówkowe są wymagane, ponieważ korzysta z nich sterownik `DBD::mysql`. (Wspomniane pliki powinny zostać zainstalowane w trakcie procedury instalacyjnej MySQL).
- Skrypty Perl uruchamiane w internecie wymagają modułu `CGI.pm`.

Aby sprawdzić, czy dany moduł Perl jest zainstalowany, użyj polecenia `perl doc`:

```
% perl doc DBI
% perl doc DBD::mysql
% perl doc CGI
```

Jeżeli moduł jest zainstalowany, polecenie `perl doc` wyświetli jego dokumentację. Brakujące moduły możesz zainstalować, odwiedzając witrynę <http://cpan.perl.org/>.

A.5. Informacje dotyczące instalacji PHP i PDO

W tej książce wykorzystano PHP wraz z rozszerzeniem PDO (ang. *PHP Data Objects*) do utworzenia skryptów PHP działających w serwerze WWW. Przyjęto założenie, że używany jest serwer WWW Apache, choć prawdopodobnie zamiast niego można wykorzystać także inny serwer WWW. Aby użyć PDO, trzeba zainstalować PHP. Jeśli PHP jest niedostępny w systemie, przejdź na witrynę <http://www.php.net/> i pobierz odpowiednią dystrybucję.

Sterownik PDO dla bazy danych MySQL można połączyć z utworzoną w języku C biblioteką klienta (`libmysqlclient`) lub biblioteką `mysqlnd`. Biblioteka `mysqlnd` jest rodzimym sterownikiem implementującym ten sam protokół komunikacyjny, który jest stosowany przez `libmysqlclient`, i może być używana w charakterze zamiennika dla `libmysqlclient`. Jeżeli nie używasz `mysqlnd`, wtedy oprócz PDO trzeba zainstalować także bibliotekę klienta MySQL. (Wspomniana biblioteka powinna zostać zainstalowana w trakcie procedury instalacyjnej MySQL).

Rozszerzenie PDO działa wraz z PHP w wersji 5.0 lub nowszej, ale w książce przyjęto założenie, że używana jest wersja PHP 5.1, ponieważ od tego wydania PDO zostało dołączone do PHP. W celu użycia PDO wraz z `mysqlnd` trzeba użyć PHP w wersji minimum 5.3. Więcej informacji znajdziesz na stronie <http://www.php.net/pdo>.

Jeżeli kompilujesz PHP ze źródeł, upewnij się, że użyłeś opcji konfiguracyjnych zapewniających obsługę MySQL i PDO. W przypadku systemu Windows dystrybucja binarna PHP jest dostępna w postaci archiwum `.zip` i instalatora `.msi`. Jeżeli zdecydujesz się na archiwum `.zip`, rozpakuj je w katalogu, w którym ma być zainstalowane PHP. Pakiet instalatora jest znacznie wygodniejszy, ponieważ przeprowadza Cię przez konfigurację Apache zapewniającą obsługę PHP i tak ustawia zmienną środowiskową `PATH`, aby zawierała katalog instalacji PHP. Jednak jeśli użyjesz instalatora, to upewnij się, że została wybrana instalacja rozszerzenia. W przeciwnym razie obsługa PDO i MySQL nie zostanie zainstalowana.

Jeżeli napotkasz jakiegokolwiek problemy podczas konfiguracji PHP, sprawdź sekcję `VERBOSE INSTALL` w pliku `INSTALL` dołączonym do dystrybucji PHP.

Po zainstalowaniu PHP i PDO sprawdź plik konfiguracyjny Apache o nazwie `httpd.conf`. Konieczne jest nakazanie serwerowi Apache wczytania modułu PHP podczas jego uruchamiania oraz wskazanie sposobu rozpoznawania skryptów PHP. (Plik `httpd.conf` może dołączać inne pliki za pomocą dyrektyw `Include`. Jeżeli w wymienionym pliku nie znajdujesz danych wspomnianych w kolejnych akapitach, sprawdź również dołączane pliki).

Aby nakazać serwerowi Apache wczytanie modułu PHP, plik `httpd.conf` musi zawierać dyrektywy `LoadModule` i `AddModule` w odpowiednich sekcjach (wyszukaj inne podobne dyrektywy). Wspomniane dyrektywy mogły zostać dodane podczas instalacji PHP. Jeżeli tak się nie stało, będziesz musiał dodać je samodzielnie. Dyrektywy mogą przedstawiać się, jak pokazano poniżej, choć ścieżka dostępu w dyrektywie `LoadModule` może być dostosowana do Twojego systemu:

```
LoadModule php5_module libexec/libphp5.so
AddModule mod_php5.c
```

Następnie trzeba przeprowadzić konfigurację pliku *httpd.conf* i wskazać sposób rozpoznawania skryptów PHP. Rozpoznawanie skryptów PHP odbywa się na podstawie rozszerzenia nazwy pliku przypisywanego skryptom. Najczęściej spotykane to rozszerzenie *.php* i jest ono używane w tej książce. Aby zdefiniować *.php* jako rozszerzenie skryptu PHP, w pliku *httpd.conf* umieść poniższy wiersz:

```
AddType application/x-httpd-php .php
```

Istnieje również możliwość wskazania serwerowi Apache, że *index.php* to dozwolony plik domyślny w katalogu, jeśli na końcu adresu URL nie została podana żadna nazwa pliku. W pliku konfiguracyjnym *httpd.conf* prawdopodobnie znajdziesz wiersz podobny do poniższego:

```
DirectoryIndex index.html
```

Zmień go na następujący:

```
DirectoryIndex index.php index.html
```

Po przeprowadzeniu edycji pliku konfiguracyjnego serwera Apache zatrzymaj serwer, jeśli działał, a następnie ponownie go uruchom. W wielu systemach służące do tego celu polecenia są następujące (należy je wykonać jako użytkownik root):

```
# /usr/local/apache/bin/apachectl stop  
# /usr/local/apache/bin/apachectl start
```

Serwer Apache można również skonfigurować do uruchamiania i zatrzymywania wraz z systemem. Odpowiednie informacje na ten temat znajdziesz w dokumentacji Apache. Zwykle oznacza to wydanie polecenia *apachectl start* w trakcie uruchamiania systemu i *apachectl stop* w trakcie jego zamykania.

Przewodnik po typach danych

Ten dodatek zawiera opis typów danych obsługiwanych przez MySQL. Więcej informacji na temat sposobu użycia poszczególnych typów danych znajdziesz w rozdziale 3., zatytułowanym „Typy danych”.

W specyfikacji nazw typów danych zastosowano następujące konwencje:

- Nawiasy kwadratowe ([]) w składni wskazują na informacje opcjonalne.
- *M* oznacza maksymalną szerokość typów liczb całkowitych, precyzję (ilość znaczących cyfr) w liczbach zmiennoprzecinkowych oraz o stałej ilości cyfr, liczbę bitów w typie BIT i maksymalną długość ciągu tekstowego. W definicjach kolumn tekstowych dla typów binarnych ciągów tekstowych długość jest wyrażona w bajtach, natomiast dla niebinarnych ciągów tekstowych długość jest wyrażona w znakach.
- *D* oznacza skalę (liczbę cyfr znajdujących się po przecinku dziesiętnym) w przypadku typów liczbowych posiadających część dziesiętną. Wartość *D* musi być równa *M* lub mniejsza, w przeciwnym razie wystąpi błąd.
- *fsp* oznacza precyzję dla typów daty i godziny pozwalających na stosowanie ułamkowych części sekund.

Poszczególne opisy typów danych zawierają jeden lub więcej wymienionych poniżej rodzajów informacji:

Opis. Krótki opis typu.

Dozwolone atrybuty. Opcjonalne słowa kluczowe atrybutów, które w zapytaniach CREATE TABLE lub ALTER TABLE mogą być powiązane z typami danych. Atrybuty są wymienione w kolejności alfabetycznej, ale to niekoniecznie odpowiada kolejności narzucanej przez składnię zapytań CREATE TABLE lub ALTER TABLE (przedstawioną w dodatku E, zatytułowanym „Przewodnik po składni SQL”). Atrybuty wymienione w opisach dla poszczególnych typów danych są używane

oprócz atrybutów globalnych dostępnych dla wszystkich lub większości typów danych. Poniżej wymieniono atrybuty globalne, dzięki czemu unikamy konieczności ich podawania w opisie poszczególnych typów danych:

- ◆ **NULL lub NOT NULL** — ten atrybut może być użyty w każdym typie danych.
- ◆ **DEFAULT *wartość_domyślna*** — wartość domyślną można podać we wszystkich definicjach kolumn poza kolumnami liczb całkowitych, dla których użyto atrybutu `AUTO_INCREMENT`, kolumnami `BLOB`, `TEXT` i kolumnami wartości przestrzennych. Z wyjątkiem typów `TIMESTAMP` i (począwszy od MySQL 5.6.5) `DATETIME`, wartości domyślne muszą być stałymi. Na przykład, jako wartości domyślnej dla kolumny `DATE` nie można wskazać `DEFAULT CURDATE()`.

Dozwolona długość. W przypadku kolumn o tekstowych typach danych to będzie maksymalna dozwolona długość wartości, jaką można przechowywać w danej kolumnie.

Zakres. W przypadku typów wartości liczbowych oraz daty i godziny to będzie zakres wartości, jakie może przedstawiać dany typ. W przypadku typów liczbowych podawane są dwa zakresy, ponieważ kolumny liczb całkowitych mogą być ze znakiem lub bez znaku, co oznacza istnienie różnych zakresów.

Wartość zero. W przypadku wartości daty i godziny to będzie wartość „zero” umieszczana w kolumnie po wstawieniu nieprawidłowej wartości do kolumny. (Tryb SQL musi zezwalać na tę zmianę, w przeciwnym razie wstawienie nieprawidłowej wartości powoduje wygenerowanie błędu).

Wartość domyślna. To jest wartość domyślna używana w sytuacji, gdy nie został wyraźnie zdefiniowany atrybut `DEFAULT` w danej specyfikacji typu. Zastosowanie ma jedynie w przypadku, kiedy nie jest włączony tryb ścisły SQL. Jeżeli nie została podana klauzula `DEFAULT`, w trybie ścisłym kolumna zostanie zdefiniowana wraz z domyślną wartością `NULL`, o ile może akceptować wartości `NULL`, w przeciwnym razie nie będzie miała wartości. Więcej informacji na ten temat znajdziesz w punkcie 3.2.3, zatytułowanym „Definiowanie wartości domyślnych kolumn”.

Wymagana pamięć masowa. Liczba bajtów lub znaków wymaganych do przechowywania wartości danego typu. W przypadku pewnych typów to będzie stała wartość. Z kolei w innych typach ilość wymaganej pamięci masowej zależy od długości wartości, która ma być przechowywana w kolumnie.

Porównania. Dla typów tekstowych ta wartość wskazuje sposób przeprowadzania porównań (tego rodzaju operacje wpływają na grupowanie, sortowanie i indeksowanie). Binarne ciągi tekstowe są porównywane bajt po bajcie za pomocą wartości liczbowych kolejnych bajtów. Natomiast niebinarne ciągi tekstowe są porównywane znak po znaku na podstawie kodowania znaków i kolejności sortowania.

Synonimy. W tym miejscu są podawane synonimy dla nazwy typu.

Uwagi. Tutaj znajdziesz różne informacje dotyczące typu.

Jeżeli nie masz pewności, jak używana wersja MySQL potraktuje daną definicję kolumny, utwórz tabelę zawierającą kolumnę zdefiniowaną w interesujący Cię sposób. Następnie wykonaj zapytanie `SHOW CREATE TABLE` lub `DESCRIBE`, aby przekonać się, jak MySQL zinterpretuje definicję kolumny. Na przykład, jeśli nie pamiętasz efektu użycia atrybutu `UNICODE` lub `SERIAL`, utwórz tabelę, w której zostaną one użyte, a następnie nakaż MySQL wyświetlenie definicji tabeli:

```
mysql> CREATE TABLE t (c CHAR(10) UNICODE, s SERIAL);
mysql> SHOW CREATE TABLE t\G
***** 1. row *****
      Table: t
Create Table: CREATE TABLE `t` (
  `c` char(10) CHARACTER SET ucs2 DEFAULT NULL,
  `s` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
  UNIQUE KEY `s` (`s`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

B.1. Typy liczbowe

MySQL oferuje typy danych dla wartości dokładnych oraz przybliżonych. Typy liczbowe mają różne zakresy wartości, więc wybieraj je według zakresu wartości, które mają być przedstawiane. Ponadto, istnieje także typ `BIT`, przeznaczony do przedstawiania wartości pól bitowych.

Typy liczb całkowitych i liczb o stałej ilości cyfr (`DECIMAL`) są typami przedstawiającymi dokładne wartości. Z kolei `FLOAT` i `DOUBLE` to typy danych przedstawiające wartości przybliżone. W przypadku typów danych wartości dokładnych wartości są przechowywane dokładnie w postaci podanej przez użytkownika, a wszelkie obliczenia są przeprowadzane bez błędu związanego z zaokrągleniem, o ile wartości i obliczenia mieszczą się w zakresie obsługiwanym przez dany typ. Natomiast w przypadku typów danych wartości przybliżonych obliczenia podlegają błędowi związanemu z zaokrągleniem wartości.

W przypadku typów liczb całkowitych, jeśli zostanie podany atrybut `AUTO_INCREMENT`, to kolumna musi być indeksowana. Wstawienie wartości `NULL` do kolumny `AUTO_INCREMENT` powoduje, że do kolumny faktycznie będzie wstawiona kolejna wartość sekwencji. Zwykle to wartość o jeden większa od obecnej wartości maksymalnej kolumny. W rozdziale 3., zatytułowanym „Typy danych”, szczegółowo omówiono zachowanie kolumn `AUTO_INCREMENT`. (Atrybut `AUTO_INCREMENT` można stosować również w przypadku kolumn przechowujących liczby zmiennoprzecinkowe, ale to rzadko spotykane rozwiązanie).

Atrybuty `ZEROFILL` i `UNSIGNED` mogą być stosowane wraz z dowolnymi typami liczbowymi poza `BIT`:

- Podanie atrybutu `ZEROFILL` powoduje, że wartości w kolumnie będą dopełnione zerami umieszczanymi na początku w celu dopasowania szerokości wartości do szerokości kolumny.

- Jeżeli podany będzie atrybut UNSIGNED, niedozwolone jest używanie wartości ujemnych. (Dozwołonym atrybutem jest także SIGNED, choć jego użycie nie ma żadnego efektu, ponieważ liczbowe typy danych domyślnie używają znaku).

Atrybut SERIAL DEFAULT VALUE jako atrybut dla typu danych liczby całkowitej lub zmiennoprzecinkowej jest skrótem dla NOT NULL AUTO_INCREMENT UNIQUE.

W pewnych przypadkach podanie jednego atrybutu powoduje włączenie także innego. Podanie atrybutu ZEROFILL dla typu wartości liczbowych automatycznie powoduje dodanie atrybutu UNSIGNED dla danej kolumny. Z kolei użycie atrybutu AUTO_INCREMENT automatycznie powoduje, że kolumna otrzymuje atrybut NOT NULL.

Zwróć uwagę, że zapytania DESCRIBE i SHOW COLUMNS podają NULL jako wartość domyślną dla kolumny AUTO_INCREMENT, choć w tego rodzaju kolumnie nie można przechowywać dosłownych wartości NULL. To wskazuje, że podczas wstawiania nowego rekordu generujesz wartość domyślną kolumny (to znaczny kolejną liczbę w sekwencji) przez podanie NULL dla tej kolumny.

B.1.1. Typy liczb całkowitych

■ TINYINT[(M)]

Opis. Bardzo mała liczba całkowita. M oznacza maksymalną szerokość wyświetlanej liczby i przyjmuje wartość od 1 do 255. W przypadku jej pominięcia wartością domyślną M jest 4 (lub 3 dla kolumn UNSIGNED).

Dozwołone atrybuty. AUTO_INCREMENT, SERIAL DEFAULT VALUE, UNSIGNED, ZEROFILL.

Zakres. Od -128 do 127 (od -2^7 do 2^7-1) lub od 0 do 255 (od 0 do 2^8-1) po użyciu atrybutu UNSIGNED.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest 0 .

Wymagana pamięć masowa. 1 bajt.

Synonimy. INT1[(M)]. BOOL i BOOLEAN są synonimami typu TINYINT(1).

■ SMALLINT[(M)]

Opis. Mała liczba całkowita. M oznacza maksymalną szerokość wyświetlanej liczby i przyjmuje wartość od 1 do 255. W przypadku jej pominięcia wartością domyślną M jest 6 (lub 5 dla kolumn UNSIGNED).

Dozwołone atrybuty. AUTO_INCREMENT, SERIAL DEFAULT VALUE, UNSIGNED, ZEROFILL.

Zakres. Od -32768 do 32767 (od -2^{15} do $2^{15}-1$) lub od 0 do 65535 (od 0 do $2^{16}-1$) po użyciu atrybutu UNSIGNED.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest 0 .

Wymagana pamięć masowa. 2 bajty.

Synonimy. `INT2[(M)]`.

■ `MEDIUMINT[(M)]`

Opis. Średnia liczba całkowita. M oznacza maksymalną szerokość wyświetlanej liczby i przyjmuje wartość od 1 do 255. W przypadku jej pominięcia wartością domyślną M jest 9 (lub 8 dla kolumn `UNSIGNED`).

Dozwolone atrybuty. `AUTO_INCREMENT`, `SERIAL` `DEFAULT` `VALUE`, `UNSIGNED`, `ZEROFILL`.

Zakres. Od -8388608 do 8388607 (od -2^{23} do $2^{23}-1$) lub od 0 do 16777215 (od 0 do $2^{24}-1$) po użyciu atrybutu `UNSIGNED`.

Wartość domyślna. Wartość `NULL`, o ile kolumna akceptuje wartości `NULL`. Jeśli kolumna jest zdefiniowana jako `NOT NULL`, wtedy wartością domyślną jest 0.

Wymagana pamięć masowa. 3 bajty.

Synonimy. `INT3[(M)]` i `MIDDLEINT[(M)]`.

■ `INT[(M)]`

Opis. Normalna liczba całkowita. M oznacza maksymalną szerokość wyświetlanej liczby i przyjmuje wartość od 1 do 255. W przypadku jej pominięcia wartością domyślną M jest 11 (lub 10 dla kolumn `UNSIGNED`).

Dozwolone atrybuty. `AUTO_INCREMENT`, `SERIAL` `DEFAULT` `VALUE`, `UNSIGNED`, `ZEROFILL`.

Zakres. Od -2147483648 do 2147483647 (od -2^{31} do $2^{31}-1$) lub od 0 do 4294967295 (od 0 do $2^{32}-1$) po użyciu atrybutu `UNSIGNED`.

Wartość domyślna. Wartość `NULL`, o ile kolumna akceptuje wartości `NULL`. Jeśli kolumna jest zdefiniowana jako `NOT NULL`, wtedy wartością domyślną jest 0.

Wymagana pamięć masowa. 4 bajty.

Synonimy. `INTEGER[(M)]` i `INT4[(M)]`.

■ `BIGINT[(M)]`

Opis. Ogromna liczba całkowita. M oznacza maksymalną szerokość wyświetlanej liczby i przyjmuje wartość od 1 do 255. W przypadku jej pominięcia wartością domyślną M jest 20.

Dozwolone atrybuty. `AUTO_INCREMENT`, `SERIAL` `DEFAULT` `VALUE`, `UNSIGNED`, `ZEROFILL`.

Zakres. Od -9223372036854775808 do 9223372036854775807 (od -2^{63} do $2^{63}-1$) lub od 0 do 18446744073709551615 (od 0 do $2^{64}-1$) po użyciu atrybutu `UNSIGNED`.

Wartość domyślna. Wartość `NULL`, o ile kolumna akceptuje wartości `NULL`. Jeśli kolumna jest zdefiniowana jako `NOT NULL`, wtedy wartością domyślną jest 0.

Wymagana pamięć masowa. 8 bajtów.

Synonimy. `INT8[(M)]`.

Uwagi. SERIAL jako nazwa typu danych jest skrótem dla BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE.

B.1.2. Typy liczb o stałej ilości cyfr

■ DECIMAL(M , D))

Opis. Liczba o stałej ilości cyfr. M oznacza liczbę znaczących cyfr, którą może mieć wartość, od 1 do 65. D oznacza liczbę miejsc po przecinku, od 0 do 30.

Jeżeli D wynosi 0, wartości kolumny nie będą miały części ułamkowej.

Po pominięciu M i D wartościami domyślnymi są odpowiednio 10 i 0.

Dozwolone atrybuty. UNSIGNED, ZEROFILL.

Zakres. Zakres danej kolumny DECIMAL jest określany na podstawie wartości M i D oraz tego, czy został użyty atrybut UNSIGNED.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL.

Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest 0.

Wymagana pamięć masowa. Ilość wymaganej pamięci masowej zależy od liczby cyfr po obu stronach przecinka dziesiętnego. Dla każdej strony wymagane są po cztery bajty dla każdych dziewięciu cyfr, plus od jednego do czterech bajtów dla pozostałych cyfr, o ile istnieją. Ilość wymaganej pamięci masowej jest sumą wymienionych wartości.

Synonimy. NUMERIC(M , D)), DEC(M , D)) i FIXED(M , D)).

B.1.3. Typy liczb zmiennoprzecinkowych

■ FLOAT(p)

Opis. Liczba zmiennoprzecinkowa. W standardzie SQL precyzja (p) oznacza minimalną wymaganą liczbę bitów precyzji. W MySQL specyfikator p jest używany jedynie do ustalenia, czy typ danych jest pojedynczej, czy podwójnej precyzji:

- ◆ Dla wartości specyfikatora p z zakresu od 0 do 24 typ jest pojedynczej precyzji i odpowiada typowi FLOAT bez specyfikatorów M i D .
- ◆ Dla wartości specyfikatora p z zakresu od 25 do 53 typ jest podwójnej precyzji i odpowiada typowi DOUBLE bez specyfikatorów M i D .

Wartości specyfikatora p spoza zakresu od 0 do 53 są nieprawidłowe.

Dozwolone atrybuty. UNSIGNED, ZEROFILL.

Zakres. Zapoznaj się z opisem typów FLOAT i DOUBLE w dalszej części dodatku.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL.

Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest 0.

Wymagana pamięć masowa. 4 bajty dla pojedynczej precyzji, 8 bajtów dla podwójnej precyzji.

■ FLOAT $[(M,D)]$

Opis. Mała liczba zmiennoprzecinkowa, pojedynczej precyzji (mniej precyzyjna niż DOUBLE). M oznacza liczbę znaczących cyfr, którą może mieć wartość, od 1 do 255. D oznacza liczbę miejsc po przecinku, od 0 do 30. Jeżeli D wynosi 0, wartości kolumny nie będą miały części ułamkowej. Po pominięciu M i D szerokość wyświetlanej liczby pozostaje niezdefiniowana. Wartości są przechowywane do pełnej precyzji obsługiwanej przez sprzęt.

Dozwolone atrybuty. UNSIGNED, ZEROFILL.

Zakres. Minimalna wartość niezerowa wynosi $\pm 1.175494351\text{E}-38$; natomiast maksymalna wartość niezerowa to $\pm 3.402823466\text{E}+38$. Jeżeli kolumna została zdefiniowana jako UNSIGNED, wtedy wartości ujemne są niedozwolone.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest 0.

Wymagana pamięć masowa. 4 bajty.

Synonimy. FLOAT4 jest synonimem dla FLOAT bez specyfikatorów M lub D . Po włączeniu trybu SQL o nazwie REAL_AS_FLOAT, REAL $[(M,D)]$ jest synonimem FLOAT $[(M,D)]$.

■ DOUBLE $[(M,D)]$

Opis. Ogromna liczba zmiennoprzecinkowa, podwójnej precyzji (bardziej precyzyjna niż FLOAT). Specyfikatory M i D mają takie samo znaczenie jak w przypadku typu FLOAT.

Dozwolone atrybuty. UNSIGNED, ZEROFILL.

Zakres. Minimalna wartość niezerowa wynosi $\pm 2.2250738585072014\text{E}-308$, natomiast maksymalna wartość niezerowa to $\pm 1.7976931348623157\text{E}+308$. Jeżeli kolumna została zdefiniowana jako UNSIGNED, wtedy wartości ujemne są niedozwolone.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest 0.

Wymagana pamięć masowa. 8 bajtów.

Synonimy. DOUBLE PRECISION $[(M,D)]$ jest synonimem dla DOUBLE $[(M,D)]$, podobnie jak REAL $[(M,D)]$ po włączeniu trybu SQL o nazwie REAL_AS_FLOAT. FLOAT8 jest synonimem DOUBLE bez specyfikatorów M i D .

B.1.4. Typ BIT

■ BIT $[(M)]$

Opis. Wartość pola bitowego. Specyfikator M powinien być liczbą całkowitą z zakresu od 1 do 64, wskazującą liczbę bitów na wartość. W przypadku pominięcia M wartością domyślną jest 1.

Dozwolone atrybuty. Żadne poza innymi atrybutami globalnymi.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest 0.

Wymagana pamięć masowa. Około $(M + 7)/8$ bajtów.

B.2. Typy tekstowe

Oferowane przez MySQL typy tekstowe są typami ogólnego przeznaczenia, często wykorzystywanymi do przechowywania danych binarnych lub tekstowych (znaków).

Typy tekstowe mogą przechowywać wartości o różnej wielkości maksymalnej.

Do wyboru masz typy traktujące dane jako binarne lub niebinarne ciągi tekstowe.

BINARY, VARBINARY i BLOB to typy binarnych ciągów tekstowych. Wspomniany binarny ciąg tekstowy jest sekwencją bajtów, a jego długość jest wyrażona w bajtach. W przypadku binarnych ciągów tekstowych nie jest definiowane kodowanie znaków, a porównywanie wartości odbywa się na podstawie ich liczbowych wartości bajtów.

CHAR, VARCHAR i TEXT to typy niebinarnych ciągów tekstowych. Niebinarny ciąg tekstowy jest sekwencją znaków, a także posiada zdefiniowane kodowanie znaków i kolejność sortowania. Kodowanie znaków definiuje znaki, których stosowanie jest dozwolone w danym typie danych, natomiast kolejność sortowania określa kolejność, w jakiej będą sortowane znaki. Długość podana w definicji kolumny niebinarnego ciągu tekstowego wskazuje maksymalną liczbę znaków, które mogą być przechowywane w danej kolumnie.

Standardowo długość niebinarnego ciągu tekstowego jest podawana w znakach, ale może być również wyrażona w bajtach. W celu sprawdzenia wyrażonej w znakach lub bajtach długości niebinarnego ciągu tekstowego należy użyć funkcji odpowiednio CHAR_LENGTH() lub LENGTH(). Niebinarny ciąg tekstowy o długości n znaków ma jednocześnie długość n bajtów, o ile zawiera znaki jednobajtowe; jeśli zawiera znaki wielobajtowe, wtedy jego długość wyrażona w bajtach jest większa niż n . To wpływa na ilość pamięci masowej wymaganej do przechowywania kolumn niebinarnych ciągów tekstowych:

- Kolumny o stałej długości, takie jak CHAR(M), wymagają ilości pamięci masowej wystarczającej do przechowywania M egzemplarzy najszerszego znaku w zestawie znaków. Na przykład, znaki w kodowaniu utf8 mają od jednego do trzech bajtów, więc CHAR(M) wymaga $M \cdot 3$ bajtów.
- Kolumny o zmiennej długości, takie jak VARCHAR(M), wymagają jedynie wystarczającej ilości pamięci masowej do przechowywania rzeczywistych znaków w danej wartości plus prefiksu wyrażającego w bajtach długość wartości. Kolumna VARCHAR(10) z wartościami w kodowaniu ucs2 wymaga jednego bajta dla prefiksu plus od zera do 20 bajtów dla dziesięcioznakowego ciągu tekstowego.

Kodowanie znaków i kolejność sortowania podawane są zarówno dla typów niebinarnych ciągów tekstowych (CHAR, VARCHAR, TEXT), jak również dla typów ENUM i SET:

- Składnia pozwalająca na podanie kodowania znaków to CHARACTER SET *kodowanie*, gdzie *kodowanie* to nazwa kodowania, na przykład latin1, greek lub utf8. CHARSET jest synonimem dla CHARACTER SET.
- Składnia pozwalająca na podanie kolejności sortowania to COLLATE *kolejność*, gdzie *kolejność* to lista dozwolonych przez dane kodowanie znaków kolejności sortowania.
- Jeżeli nie zostanie podane kodowanie znaków lub kolejność sortowania, zostaną one określone na podstawie wartości domyślnych tabeli. W przypadku podania kodowania znaków i pominięcia kolejności sortowania użyta zostanie domyślna kolejność sortowania w danym kodowaniu znaków. Z kolei podanie kolejności sortowania, ale bez kodowania znaków spowoduje ustalenie kodowania znaków na podstawie nazwy kolejności sortowania. Jeżeli podane zostanie kodowanie znaków i kolejność sortowania, wskazana kolejność sortowania musi być zgodna z danym kodowaniem znaków. Na przykład, kolejność sortowania latin1_bin jest zgodna z kodowaniem znaków latin1, ale już nie z kodowaniem utf8.
- Kodowanie znaków binary i atrybut kolumny BINARY są traktowane w sposób specjalny:
 - ◆ Jeżeli dla typu niebinarnego ciągu tekstowego kodowanie zostanie zdefiniowane jako CHARACTER SET binary, spowoduje konwersję ciągu tekstowego na odpowiadający mu typ binarnego ciągu tekstowego. Oznacza to, że typ CHAR zostaje zamieniony na BINARY, VARCHAR na VARBINARY, a TEXT na BLOB. Typy ENUM i SET nie mają odpowiadających im typów binarnych, więc CHARACTER SET binary po prostu pozostawia atrybut kolumny bez zmian.
 - ◆ Atrybut BINARY jest odpowiednikiem zdefiniowania binarnej kolejności sortowania dla kodowania znaków (nazwa kolejności sortowania kończąca się przyrostkiem _bin). Na przykład, kolumna zdefiniowana jako CHAR(10) CHARACTER SET utf8 BINARY staje się kolumną typu CHAR(10) CHARACTER SET utf8 COLLATE utf8_bin.
- W przypadku typów niebinarnych ciągów tekstowych atrybuty ASCII i UNICODE są skrótami dla odpowiednio CHARACTER SET latin1 i CHARACTER SET ucs2.

Dozwolone kodowania znaków i kolejności sortowania obsługiwane przez serwer można ustalić za pomocą zapytań SHOW CHARACTER SET i SHOW COLLATION. Wymienione zapytania powodują wyświetlenie domyślnej kolejności sortowania dla każdego obsługiwanego kodowania znaków. Możesz również przeanalizować tabele CHARACTER_SETS i COLLATIONS w bazie danych INFORMATION_SCHEMA, które zawierają takie same informacje jak dane wyjściowe wymienionych wcześniej zapytań.

Sposób obsługi wartości zbyt długich do umieszczenia w kolumnie tekstowej zależy od wybranego trybu SQL. Jeżeli nie został włączony tryb ścisły, wartości są odpowiednio skracane do długości pozwalającej na ich wstawienie do kolumny. W trybie ścisłym następuje wygenerowanie błędu, a żadna wartość nie zostaje wstawiona do kolumny, jeśli będą musiały być usunięte znaki inne niż spacje.

Sposób obsługi znaków dopełniających zależy od typu ciągu tekstowego:

- W przypadku typu CHAR, jeśli zachodzi potrzeba, to wartości są dopełniane spacjami aż do pełnej długości kolumny. Podczas pobierania wartości dodane spacje są usuwane.
- W przypadku typu BINARY, jeśli zachodzi potrzeba, to wartości są dopełniane bajtami 0x00 aż do pełnej długości kolumny. Podczas pobierania wartości nic nie jest usuwane.
- W przypadku typów VARBINARY, VARCHAR, BLOB i TEXT podczas wstawiania oraz pobierania wartości nie są dodawane ani usuwane żadne znaki.
- W przypadku typów ENUM i SET wszelkie spacje znajdujące się na końcu wartości składowych wymienionych w definicji kolumny są ignorowane. Dlatego też wszelkie spacje na końcu wartości przechowywanych w kolumnie są usuwane, ponieważ MySQL konwertuje każdą wartość na odpowiadającą jej wewnętrzną wartość liczbową elementu składowego kolumny. To wpływa również na operacje porównania — spacje znajdujące się na końcu wartości nie mają znaczenia w przypadku wartości porównywanych do kolumn ENUM lub SET.

B.2.1. Typy binarnych ciągów tekstowych

■ BINARY(*M*)

Opis. Binarny ciąg tekstowy o stałej długości wynoszącej od 0 do *M* bajtów.

Specyfikator *M* powinien być liczbą całkowitą z zakresu od 0 do 255.

W przypadku pominięcia wartość domyślna *M* wynosi 1.

Dozwolone atrybuty. Żadne poza innymi atrybutami globalnymi.

Dozwolona długość. Od 0 do *M* bajtów.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL.

Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest '' (pusty ciąg tekstowy).

Wymagana pamięć masowa. *M* bajtów.

Porównania. Bajt po bajcie, na podstawie liczbowych wartości bajtów.

■ VARBINARY(*M*)

Opis. Binarny ciąg tekstowy o zmiennej długości wynoszącej od 0 do *M* bajtów.

Specyfikator *M* powinien być liczbą całkowitą z zakresu od 0 do 65535.

Dozwolone atrybuty. Żadne poza innymi atrybutami globalnymi.

Dozwolona długość. Od 0 do M bajtów (prawdopodobnie mniej niż M , jak wspomniano w uwagach).

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest '' (pusty ciąg tekstowy).

Wymagana pamięć masowa. Długość wartości (wyrażona w bajtach) plus jedno- lub dwubajtowy prefiks określający długość wartości. Prefiks wymaga jednego bajta dla maksymalnej długości wartości kolumny wynoszącej mniej niż 256 bajtów. Dla większych wartości długość prefiksu wynosi 2 bajty.

Porównania. Bajt po bajcie, na podstawie liczbowych wartości bajtów.

Uwagi. W praktyce maksymalna długość kolumny VARBINARY jest ograniczona do 65535 bajtów, a prawdopodobnie nawet mniej. To zależy od nakładanego przez silnik bazy danych ograniczenia wielkości rekordu oraz ilości pamięci masowej zajmowanej przez pozostałe kolumny w tabeli.

■ TINYBLOB

Opis. Mała wartość w postaci binarnego ciągu tekstowego.

Dozwolone atrybuty. Żadne poza innymi atrybutami globalnymi.

Dozwolona długość. Od 0 do 255 (od 0 do 2^8-1) bajtów.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest '' (pusty ciąg tekstowy).

Wymagana pamięć masowa. Długość wartości (wyrażona w bajtach) plus jeden bajt na zapis długości wartości.

Porównania. Bajt po bajcie, na podstawie liczbowych wartości bajtów.

■ BLOB[(M)]

Opis. Normalnej wielkości wartość w postaci binarnego ciągu tekstowego.

Dozwolone atrybuty. Żadne poza innymi atrybutami globalnymi.

Dozwolona długość. Od 0 do 65535 (od 0 do $2^{16}-1$) bajtów. Jeżeli podana zostanie długość (M), będzie użyta do wyboru odpowiedniego typu danych, a następnie odrzucona. Dla długości z zakresu od 1 do 65535 typem danych jest BLOB. W przypadku długości 65536 i większych typem danych staje się MEDIUMBLOB lub LONGBLOB, w zależności od tego, który typ będzie wymagany do pomieszczenia wartości o podanej długości.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest '' (pusty ciąg tekstowy).

Wymagana pamięć masowa. Długość wartości (wyrażona w bajtach) plus dwa bajty na zapis długości wartości.

Porównania. Bajt po bajcie, na podstawie liczbowych wartości bajtów.

■ MEDIUMBLOB

Opis. Średniej wielkości wartość w postaci binarnego ciągu tekstowego.

Dozwolone atrybuty. Żadne poza innymi atrybutami globalnymi.

Dozwolona długość. Od 0 do 16777215 (od 0 do $2^{24}-1$) bajtów.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest '' (pusty ciąg tekstowy).

Wymagana pamięć masowa. Długość wartości (wyrażona w bajtach) plus trzy bajty na zapis długości wartości.

Porównania. Bajt po bajcie, na podstawie liczbowych wartości bajtów.

Synonimy. LONG VARBINARY.

■ LONGBLOB

Opis. Ogromnej wielkości wartość w postaci binarnego ciągu tekstowego.

Dozwolone atrybuty. Żadne poza innymi atrybutami globalnymi.

Dozwolona długość. Od 0 do 4294967295 (od 0 do $2^{32}-1$) bajtów.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest '' (pusty ciąg tekstowy).

Wymagana pamięć masowa. Długość wartości (wyrażona w bajtach) plus cztery bajty na zapis długości wartości.

Porównania. Bajt po bajcie, na podstawie liczbowych wartości bajtów.

B.2.2. Typy niebinarnych ciągów tekstowych

■ CHAR(*M*)

Opis. Niebinarny ciąg tekstowy o stałej długości wynoszącej od 0 do *M* bajtów. Specyfikator *M* powinien być liczbą całkowitą z zakresu od 0 do 255. W przypadku pominięcia wartość domyślna *M* wynosi 1.

Dozwolone atrybuty. BINARY, CHARACTER SET, COLLATE.

Dozwolona długość. Od 0 do *M* znaków.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest '' (pusty ciąg tekstowy).

Wymagana pamięć masowa. *M* znaków, co oznacza *M-w* bajtów, gdzie *w* oznacza liczbę bajtów wymaganych przez najszerszy znak w zdefiniowanym dla kolumny kodowaniu znaków.

Porównania. Znak po znaku, na podstawie kolejności sortowania zdefiniowanej dla danej kolumny.

Synonimy. NCHAR(*M*) i NATIONAL CHAR(*M*) są synonimami dla CHAR(*M*) CHARACTER SET utf8.

■ VARCHAR(*M*)

Opis. Niebinarny ciąg tekstowy o zmiennej długości wynoszącej od 0 do *M* bajtów. Specyfikator *M* powinien być liczbą całkowitą z zakresu od 0 do 65535.

Dozwolone atrybuty. BINARY, CHARACTER SET, COLLATE.

Dozwolona długość. Od 0 do *M* bajtów (prawdopodobnie mniej niż *M*, jak wspomniano w uwagach).

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest '' (pusty ciąg tekstowy).

Wymagana pamięć masowa. Długość wartości (wyrażona w bajtach) plus jedno- lub dwubajtowy prefiks określający długość wartości. Prefiks wymaga jednego bajta dla maksymalnej długości wartości kolumny wynoszącej mniej niż 256 bajtów. Dla większych wartości długość prefiksu wynosi 2 bajty.

Porównania. Znak po znaku, na podstawie kolejności sortowania zdefiniowanej dla danej kolumny.

Synonimy. CHAR VARYING(*M*), NCHAR VARYING(*M*) i NATIONAL CHAR VARYING(*M*) są synonimami dla VARCHAR(*M*) CHARACTER SET utf8.

Uwagi. W praktyce maksymalna długość kolumny VARCHAR jest ograniczona do 65535 bajtów, a prawdopodobnie nawet mniej. To zależy od nakładanego przez silnik bazy danych ograniczenia wielkości rekordu oraz ilości pamięci masowej zajmowanej przez pozostałe kolumny w tabeli.

■ TINYTEXT

Opis. Mała wartość w postaci niebinarnego ciągu tekstowego.

Dozwolone atrybuty. BINARY, CHARACTER SET, COLLATE.

Dozwolona długość. Od 0 do 255 (od 0 do 2^8-1) bajtów; liczba znaków będzie mniejsza, jeśli wartość zawiera znaki wielobajtowe.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest '' (pusty ciąg tekstowy).

Wymagana pamięć masowa. Długość wartości (wyrażona w bajtach) plus jeden bajt na zapis długości wartości.

Porównania. Znak po znaku, na podstawie kolejności sortowania zdefiniowanej dla danej kolumny.

■ TEXT[(*M*)]

Opis. Normalnej wielkości wartość w postaci niebinarnego ciągu tekstowego.

Dozwolone atrybuty. BINARY, CHARACTER SET, COLLATE.

Dozwolona długość. Od 0 do 65535 (od 0 do $2^{16}-1$) bajtów; liczba znaków będzie mniejsza, jeśli wartość zawiera znaki wielobajtowe. Jeżeli podana zostanie

długość (M), będzie użyta do wyboru odpowiedniego typu danych, a następnie odrzucona. Dla długości z zakresu od 1 do 65535 typem danych jest TEXT.

W przypadku długości 65536 i większych typem danych staje się MEDIUMTEXT lub LONGTEXT, w zależności od tego, który typ będzie wymagany do pomieszczenia wartości o podanej długości.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest '' (pusty ciąg tekstowy).

Wymagana pamięć masowa. Długość wartości (wyrażona w bajtach) plus dwa bajty na zapis długości wartości.

Porównania. Znak po znaku, na podstawie kolejności sortowania zdefiniowanej dla danej kolumny.

■ MEDIUMTEXT

Opis. Średniej wielkości wartość w postaci niebinarnego ciągu tekstowego.

Dozwolone atrybuty. BINARY, CHARACTER SET, COLLATE.

Dozwolona długość. Od 0 do 16777215 (od 0 do $2^{24}-1$) bajtów; liczba znaków będzie mniejsza, jeśli wartość zawiera znaki wielobajtowe.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest '' (pusty ciąg tekstowy).

Wymagana pamięć masowa. Długość wartości (wyrażona w bajtach) plus trzy bajty na zapis długości wartości.

Porównania. Znak po znaku, na podstawie kolejności sortowania zdefiniowanej dla danej kolumny.

Synonimy. LONG VARCHAR.

■ LONGTEXT

Opis. Ogromnej wielkości wartość w postaci niebinarnego ciągu tekstowego.

Dozwolone atrybuty. BINARY, CHARACTER SET, COLLATE.

Dozwolona długość. Od 0 do 4294967295 (od 0 do $2^{32}-1$) bajtów; liczba znaków będzie mniejsza, jeśli wartość zawiera znaki wielobajtowe.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest '' (pusty ciąg tekstowy).

Wymagana pamięć masowa. Długość wartości (wyrażona w bajtach) plus cztery bajty na zapis długości wartości.

Porównania. Znak po znaku, na podstawie kolejności sortowania zdefiniowanej dla danej kolumny.

B.2.3. Typy ENUM i SET

- `ENUM('wartość1','wartość2',...)`

Opis. Typ wyliczeniowy, wartością kolumny może być dokładnie jeden element składowy listy dozwolonych wartości.

Dozwolone atrybuty. CHARACTER SET, COLLATE.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest pierwsza wartość typu wyliczeniowego.

Wymagana pamięć masowa. Jeden bajt dla typów wyliczeniowych zawierających od 1 do 255 elementów, dwa bajty dla typów wyliczeniowych zawierających od 256 do 65535 elementów.

Porównania. Na podstawie liczbowych wartości kolumny.

Uwagi. W definicji typu danych wszelkie spacje znajdujące się na końcu wartości są ignorowane.

- `SET('wartość1','wartość2',...)`

Opis. Zbiór; wartością kolumny może być zero lub więcej elementów składowych listy dozwolonych wartości.

Dozwolone atrybuty. CHARACTER SET, COLLATE.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest '' (pusty ciąg tekstowy).

Wymagana pamięć masowa. Jeden bajt (dla zbiorów składających się z od 1 do 8 elementów), dwa bajty (dla zbiorów składających się z od 9 do 16 elementów), trzy bajty (dla zbiorów składających się z od 17 do 24 elementów), cztery bajty (dla zbiorów składających się z od 25 do 32 elementów) lub osiem bajtów (dla zbiorów składających się z od 33 do 64 elementów).

Porównania. Na podstawie liczbowych wartości kolumny.

Uwagi. W definicji typu danych wszelkie spacje znajdujące się na końcu wartości są ignorowane.

B.3. Typy daty i godziny

MySQL oferuje kilka typów danych przeznaczonych do przedstawiania daty i godziny. Dostępne są typy danych pozwalające na przedstawianie daty i godziny oddzielnie lub razem, a także typ do przechowywania jedynie lat, gdy nie ma potrzeby przechowywania całej daty. Pewne typy mogą być inicjalizowane automatycznie z bieżącą datą i godziną dla nowych rekordów i automatycznie uaktualniane bieżącą datą i godziną w trakcie zmiany innych kolumn rekordu.

Komponenty *CC*, *YY*, *MM* i *DD* w formacie daty przedstawiają odpowiednio wiek, rok, miesiąc i dzień. Z kolei komponenty *hh*, *mm* i *ss* w formacie godziny przedstawiają odpowiednio godzinę, minutę i sekundę. Ponadto, począwszy od wersji 5.6.4, serwer MySQL obsługuje także ułamkowe części sekundy w typach *TIME*, *DATETIME* i *TIMESTAMP*. Dozwolona precyzja części ułamkowej wynosi do 6 cyfr (mikrosekundy). Komponent *uuuuuu* w formatach godziny oznacza część ułamkową sekundy. (W przypadku wersji wcześniejszych niż 5.6.4 możesz zignorować ten komponent).

W opisach składni typu *fps* oznacza precyzję ułamkowej części sekundy w typach zezwalających na jej stosowanie. Precyzja musi być wartością z zakresu od 0 do 6, gdzie 0 oznacza brak części ułamkowej, natomiast 6 oznacza precyzję wyrażoną w mikrosekundach. W przypadku braku wartości dla *fps* wartością domyślną jest 0.

Począwszy od MySQL w wersji 5.6.4, zmianie ulegała ilość pamięci masowej wymaganej do przechowywania typów zezwalających na użycie części ułamkowej sekundy. Ilość pamięci masowej dla części nieułamkowej jest taka, jak podano w opisach poszczególnych typów. Z kolei wymagania w zakresie pamięci masowej dla części ułamkowej są takie same we wszystkich typach i zostały wymienione w tabeli B.1.

Tabela B.1. Liczba bajtów wymaganych do przechowywania informacji o precyzji części ułamkowej sekundy

Liczba miejsc po przecinku dziesiętnym	Wymagana ilość pamięci masowej
0	0 bajtów
1,2	1 bajt
3,4	2 bajty
5,6	3 bajty

W MySQL 5.6.5 wprowadzono rozszerzoną obsługę automatycznego użycia bieżącego znacznika czasu jako wartości początkowej oraz podczas uaktualnień. We wcześniejszych wersjach wspomniane właściwości były dostępne co najwyżej dla kolumny typu *TIMESTAMP* w tabeli, natomiast obecnie mogą być stosowane w dowolnej kolumnie typu *TIMESTAMP* i *DATETIME*.

■ DATE

Opis. Data w formacie '*CCYY-MM-DD*'.

Dozwolone atrybuty. Żadne poza innymi atrybutami globalnymi.

Zakres. Od '*1000-01-01*' do '*9999-12-31*'.

Wartość zero. '*0000-00-00*'.

Wartość domyślna. Wartość *NULL*, o ile kolumna akceptuje wartości *NULL*. Jeśli kolumna jest zdefiniowana jako *NOT NULL*, wtedy wartością domyślną jest '*0000-00-00*'.

Wymagana pamięć masowa. Trzy bajty.

■ DATETIME[(fsp)]

Opis. Wartość daty i godziny w formacie 'CCYY-MM-DD hh:mm:ss[.uuuuuu]'. Począwszy od MySQL 5.6.4, komponent *fps* pozwala na podanie precyzji jako wartości z zakresu od 0 do 6. W przypadku braku wartości dla *fps* wartością domyślną jest 0.

Dozwolone atrybuty. Przed wydaniem MySQL 5.6.5 nie były dozwolone żadne atrybuty poza globalnymi. Począwszy od wersji 5.6.5, kolumna DATETIME może zawierać atrybut DEFAULT CURRENT_TIMESTAMP lub ON UPDATE CURRENT_TIMESTAMP bądź też oba wymienione. Ich znaczenie zostało omówione w typie TIMESTAMP. Zapoznaj się z podpunktem 3.2.6.6, zatytułowanym „Automatyczne właściwości dla typów wartości daty i godziny”.

Zakres. Od '1000-01-01 00:00:00[.000000]' do '9999-12-31 23:59:59[.999999] '.

Wartość zero. '0000-00-00 00:00:00[.000000] '.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest '0000-00-00 00:00:00[.000000] '.

Wymagana pamięć masowa. Przed wydaniem MySQL 5.6.4 to osiem bajtów, w późniejszych wersjach dodatkowo pięć bajtów dla części ułamkowej sekundy.

■ TIME[(fsp)]

Opis. Wartość godziny w formacie 'hh:mm:ss[.uuuuuu]' (lub w formacie '-hh:mm:ss[.uuuuuu]' dla wartości ujemnych). Począwszy od MySQL 5.6.4, komponent *fps* pozwala na podanie precyzji jako wartości z zakresu od 0 do 6. W przypadku braku wartości dla *fps* wartością domyślną jest 0.

Dozwolone atrybuty. Żadne poza innymi atrybutami globalnymi.

Zakres. Od '-838:59:59[.000000]' do '838:59:59[.000000] '.

Wartość zero. '00:00:00[.000000] '.

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest '00:00:00[.000000] '.

Wymagana pamięć masowa. Przed wydaniem MySQL 5.6.4 to trzy bajty, w późniejszych wersjach dodatkowo trzy bajty dla części ułamkowej sekundy.

Uwagi. Wprawdzie po próbie wstawienia niedozwolonej wartości do kolumny TIME następuje wstawienie wartości zero ('00:00:00[.000000] '), ale wspomniane zero to także prawidłowa wartość mieszcząca się w zakresie dozwolonych wartości kolumny.

■ TIMESTAMP[(fsp)]

Opis. Znacznik czasu (data i godzina) w formacie 'CCYY-MM-DD hh:mm:ss[.uuuuuu]'. Począwszy od MySQL 5.6.4, komponent *fps* pozwala na podanie precyzji jako wartości z zakresu od 0 do 6. W przypadku braku

wartości dla *fps* wartością domyślną jest 0. Typ `TIMESTAMP` charakteryzuje się kilkoma cechami specjalnymi:

- ◆ Próba wstawienia wartości `NULL` do kolumny typu `TIMESTAMP` powoduje wstawienie bieżącej daty i godziny, o ile definicja kolumny nie zezwala na akceptację wartości `NULL`.
- ◆ Kolumna `TIMESTAMP` może mieć dwie właściwości automatycznej modyfikacji:
 - ◆ Automatyczna inicjalizacja: podczas tworzenia rekordu wartością domyślną kolumny jest bieżący znacznik czasu.
 - ◆ Automatyczne uaktualnienie: podczas uaktualnienia rekordu zmiana dowolnej innej kolumny rekordu może spowodować uaktualnienie kolumny `TIMESTAMP` datą i godziną, o której została wprowadzona modyfikacja.

Przed wydaniem MySQL 5.6.5 co najwyżej jedna kolumna typu `TIMESTAMP` w tabeli mogła mieć wymienione właściwości; miałeś możliwość wskazania tej, która ma być traktowana w ten sposób. Począwszy od wersji MySQL 5.6.5, dowolna kolumna typu `TIMESTAMP` może mieć wymienione właściwości, w ich dowolnym połączeniu. Więcej informacji na ten temat znajdziesz w podpunkcie 3.2.6.6, zatytułowanym „Automatyczne właściwości dla typów wartości daty i godziny”.

Dozwolone atrybuty. Przed wydaniem MySQL 5.6.5 co najwyżej jedna kolumna typu `TIMESTAMP` w tabeli mogła mieć atrybut `DEFAULT CURRENT_TIMESTAMP` lub `ON UPDATE CURRENT_TIMESTAMP` bądź też oba wymienione. Nie było możliwości zdefiniowania jednego atrybutu dla jednej kolumny typu `TIMESTAMP` i drugiego atrybutu dla innej kolumny typu `TIMESTAMP`. Wymienionych atrybutów nie można było również użyć w więcej niż tylko jednej kolumnie typu `TIMESTAMP`. Począwszy od wersji MySQL 5.6.5, dowolna kolumna typu `TIMESTAMP` może mieć wymienione atrybuty, w ich dowolnym połączeniu.

Atrybut `DEFAULT CURRENT_TIMESTAMP` powoduje, że w trakcie tworzenia rekordu kolumna otrzyma wartość w postaci bieżącej daty i godziny, o ile dla kolumny nie została podana żadna wartość. Atrybut `ON UPDATE CURRENT_TIMESTAMP` powoduje uaktualnienie kolumny bieżącą datą i godziną, gdy jakkolwiek inna kolumna w rekordzie zostanie zmodyfikowana. Funkcje `CURRENT_TIMESTAMP()` i `NOW()` to znane synonimy dla `CURRENT_TIMESTAMP`.

Jeżeli początkowe słowo kluczowe `TIMESTAMP` w definicji kolumny zawiera wartość *fps*, ta sama wartość musi być używana w dowolnej klauzuli `DEFAULT` lub `ON UPDATE` wskazującej automatyczną właściwość.

Stała `DEFAULT` może być podana w celu przypisania kolumnie typu `TIMESTAMP` określonej wartości daty i godziny lub wartości zero.

Atrybutu `NULL` można użyć w celu umożliwienia kolumnie typu `TIMESTAMP` przechowywania wartości `NULL`. Bez wspomnianego atrybutu próba wstawienia wartości `NULL` do kolumny `TIMESTAMP` spowoduje wstawienie bieżącej daty i godziny.

Zakres. Od '1970-01-01 00:00:01[.000000]' do pewnej daty w roku 2038.

Wartość zero. '0000-00-00 00:00:00[.000000]'.

Wartość domyślna. Zapytania DESCRIBE i SHOW COLUMNS wyświetlają wartość domyślną jako CURRENT_TIMESTAMP, o ile kolumna została zdefiniowana do automatycznego wstawiania bieżącej daty i godziny podczas tworzenia rekordu. W przeciwnym razie wyświetlona będzie domyślna wartość daty i godziny. Zapoznaj się z analizą dotyczącą dozwolonych atrybutów.

Wymagana pamięć masowa. Przed wydaniem MySQL 5.6.4 to cztery bajty, w późniejszych wersjach dodatkowo cztery bajty plus odpowiednia liczba bajtów dla części ułamkowej sekundy.

■ YEAR[(M)]

Opis. Wartość określająca rok. Jeżeli zostanie podany, specyfikator *M* musi mieć wartość 2 lub 4, określający format *YY* lub *CCYY*. W przypadku pominięcia specyfikatora *M* wartością domyślną *M* jest 4.

Dozwolone atrybuty. Żadne poza innymi atrybutami globalnymi.

Zakres. Od 1901 do 2155 oraz 0000 dla YEAR(4). W przypadku YEAR(2) zakres jest taki sam, ale wyświetlane są tylko jego dwie ostatnie cyfry. Aby uniknąć niejasności w przypadku wartości przechowywanych jako YEAR(2), należy je ograniczyć do przedstawiania dat z lat od 1970 do 2069 lub też użyć typu YEAR(4).

Wartość zero. 0000 dla YEAR(4) i 00 dla YEAR(2).

Wartość domyślna. Wartość NULL, o ile kolumna akceptuje wartości NULL. Jeśli kolumna jest zdefiniowana jako NOT NULL, wtedy wartością domyślną jest 0000 lub 00.

Wymagana pamięć masowa. Jeden bajt.

Uwagi. Począwszy od wersji 5.6.6 serwera MySQL, typ YEAR(2) jest uznawany za przestarzały. Tego typu kolumny będą tworzone jako YEAR(4).

C

Przewodnik po operatorach i funkcjach

Ten dodatek przedstawia operatory i funkcje używane do tworzenia wyrażeń w zapytaniach SQL. O ile nie wskazano inaczej, każdy z nich jest dostępny w wydaniu MySQL 5.5.0. Zmiany wprowadzone względem wersji MySQL 5.5.0 są podane w opisach poszczególnych operatorów i funkcji.

W większości przykładów operatorów i funkcji zastosowano poniższy format:

wyrażenie

→ *wynik*

Wyrażenie pokazuje, jak używać operatora lub funkcji, natomiast *wynik* pokazuje wartość wygenerowaną przez dane wyrażenie, na przykład:

RIGHT('my cat',3)

→ 'cat'

Powyższy przykład pokazuje, że wywołanie funkcji RIGHT('my cat',3) wygenerowało ciąg tekstowy 'cat'. Aby przykłady przedstawione w tym dodatku wypróbować samodzielnie, uruchom klienta mysql, a następnie wpisz wyrażenie w zapytaniu SELECT, dodaj średnik na końcu i naciśnij klawisz *Enter*:

```
mysql> SELECT RIGHT('my cat',3);
```

```
+-----+
| RIGHT('my cat',3) |
+-----+
| cat                |
+-----+
```

MySQL nie wymaga klauzuli FROM w zapytaniu SELECT, co znacznie ułatwia prowadzenie eksperymentów z operatorami i funkcjami — wystarczy wprowadzić je w przedstawiony powyżej sposób.

Przykłady przedstawione w tym dodatku są pełnymi zapytaniami SELECT w przypadku funkcji, których działania nie można zademonstrować w inny sposób. W punkcie C.2.6, zatytułowanym „Funkcje podsumowań”, znajdziesz tego rodzaju funkcje, ponieważ nie mają one sensu bez odwołania do konkretnej tabeli.

Nazwy funkcji, podobnie jak operatorów, są słowami, takimi jak BETWEEN, i mogą być podane wielkimi literami.

W opisie składni operatorów i funkcji zastosowano następujące konwencje:

- Element *expr* przedstawia wyrażenie. W zależności od kontekstu to może być wyrażenie liczbowe, tekstowe, daty lub godziny. Istnieje możliwość, że wyrażenie będzie zawierało stałe, odniesienia do kolumn tabeli lub innych wyrażań.
- Element *str* przedstawia ciąg tekstowy. To może być dosłowny ciąg tekstowy, odniesienie do kolumny tabeli zdefiniowanej jako typu ciągu tekstowego lub po prostu wyrażenie generujące ciąg tekstowy.
- Litera *n* oznacza liczbę całkowitą (podobnie jak inne litery, które w alfabecie znajdują się w pobliżu *n*).
- Litera *x* oznacza liczbę zmiennoprzecinkową (podobnie jak inne litery, które w alfabecie znajdują się w pobliżu *x*).
- Inne nazwy argumentów są używane znacznie rzadziej i doskonale opisane w miejscu ich stosowania.
- Nawiasy kwadratowe ([]) oznaczają informacje opcjonalne.
- Pionowe kreski (|) służą w charakterze separatorów elementów listy. Jeżeli lista zostanie ujęta w nawias kwadratowy, można z niej wybrać jeden element. W przypadku ujęcia listy w nawias klamrowy konieczne jest wybranie jednego elementu z listy.
- Wielokropek (...) wskazuje na możliwość powtórzenia terminu, który go poprzedza.

Obliczenie wyrażenia często oznacza konieczność przeprowadzenia konwersji typów wartości wyrażenia. Informacje szczegółowe dotyczące warunków wystąpienia konwersji i reguł stosowanych przez MySQL podczas konwersji wartości z jednego typu na inny przedstawiono w punkcie 3.5.2, zatytułowanym „Konwersja typu”.

C.1. Operatory

Operatory są używane w celu łączenia elementów wyrażenia i przeprowadzania operacji arytmetycznych, porównywania wartości, wykonywania operacji bitowych i logicznych, a także dopasowywania wzorca.

C.1.1. Kolejność operatorów

Operatory mają różne pierwszeństwo. Poniższa lista pokazuje pierwszeństwo operatorów, od najwyższego do najniższego. Operatory wymienione w tym samym wierszu mają takie samo pierwszeństwo. Obliczenie operatorów o wyższym pierwszeństwie w wyrażeniu następuje przed obliczeniem operatorów o niższym pierwszeństwie. Z kolei obliczanie

operatorów o takim samym pierwszeństwie następuje w wyrażeniu od lewej do prawej strony, za wyjątkiem przypisania, obliczanego od prawej do lewej strony.

```

INTERVAL
BINARY COLLATE
!
- (minus jednoargumentowy) ~ (jednoargumentowa negacja bitowa)
^
* / DIV % MOD
+ -
<< >>
&
|
< <= = <=> <> != >= > IN IS LIKE REGEXP RLIKE
BETWEEN CASE WHEN THEN ELSE
NOT
AND &&
XOR
OR ||
:=

```

Operatory jednoargumentowe (jednoargumentowy minus, jednoargumentowa negacja bitowa, NOT, BINARY i COLLATE) wiążą się ściślej niż operatory binarne. Oznacza to, że wiążą się z elementem znajdującym się bezpośrednio po nich, a nie z pozostałą częścią wyrażenia rozumianą jako całość:

```

-2+3          1
-(2+3)        → -5

```

Pierwszeństwo pewnych operatorów różni się w zależności od trybu serwera SQL i wersji MySQL:

- Jeżeli włączony jest tryb SQL o nazwie PIPES_AS_CONCAT, wtedy || jest operatorem konkatencji ciągu tekstowego, a nie logicznym operatorem OR. Dlatego też w takim przypadku jego miejsce na powyższej liście znajduje się między ^ i operatorami jednoargumentowymi.
- NOT ma mniejsze pierwszeństwo niż operator !. Aby operator NOT miał takie samo pierwszeństwo jak !, należy włączyć tryb SQL o nazwie HIGH_NOT_PRECEDENCE.

C.1.2. Operatory grupowania

Te operatory pozwalają na grupowanie elementów wyrażenia w celu zapewnienia kontroli kolejności obliczania wyrażenia lub pogrupowania wartości w krotkach.

■ (...)

Nawiasy grupują części wyrażenia. Pozwalają na zmianę domyślnej kolejności pierwszeństwa operatorów, która określa kolejność obliczania wyrażenia. (Zapoznaj się z punktem C.1.1, zatytułowanym „Kolejność operatorów”).

Nawiasy mogą być również stosowane jedynie w celu wizualnego zwiększenia

czytelności wyrażenia. Zagnieżdżone nawiasy są obliczane w kolejności od najbardziej wewnętrznego do najbardziej zewnętrznego.

```
1 + 2 * 3 / 4           → 2.5000
((1 + 2) * 3) / 4       → 2.2500
```

■ **(*expr* [, *expr*] ...)**

ROW(*expr* [, *expr*] ...)

Powyższe konstruktory rekordów mogą być używane w celu przedstawienia porównania między dwoma krotkami (zbiorami) wartości. Porównywane krotki muszą zawierać tę samą liczbę wartości. Obie zaprezentowane składnie (z użyciem oraz bez użycia słowa kluczowego ROW) działają identycznie. Na przykład, jeśli podzapytanie zwraca rekord zawierający trzy wartości, wynik można porównać do krotki zawierającej trzy wartości, używając do tego jednej z poniższych konstrukcji:

```
SELECT ... FROM t2 WHERE (0,1,2) = (SELECT col1, col2, col3 FROM ...);
SELECT ... FROM t2 WHERE ROW(0,1,2) = (SELECT col1, col2, col3 FROM ...);
```

Konstruktory rekordów mogą być również używane w kontekstach innych niż zapytanie. Przedstawione poniżej zapytanie jest prawidłowe:

```
SELECT * FROM president WHERE (first_name,last_name) = ('John','Adams');
```

C.1.3. Operatory arytmetyczne

Te operatory są przeznaczone do przeprowadzania standardowych działań arytmetycznych. Operatory arytmetyczne działają na liczbach, a nie na ciągach tekstowych (MySQL automatycznie konwertuje ciągi tekstowe wyglądające jak liczby na odpowiadające im wartości liczbowe).

Operatory arytmetyczne stosują wymienione poniżej reguły:

- Ciągi tekstowe używane w kontekście liczbowym są konwertowane na liczby o podwójnej precyzji.
- W operacjach dodawania, odejmowania i mnożenia obliczenia są przeprowadzane na liczbach 64-bitowych, jeśli oba operandy są liczbami całkowitymi. Oznacza to, że wyrażenia zawierające ogromne wartości mogą przekroczyć zakres obliczeń dla 64-bitowych liczb całkowitych. Skutkiem może być błąd przepełnienia.
- Jeżeli oba operandy są liczbami całkowitymi i przynajmniej jeden z nich jest bez znaku, wynik również będzie bez znaku.
- W operacjach dodawania, odejmowania, dzielenia, mnożenia i obliczania procentu (o ile którykolwiek operand operacji % jest liczbą rzeczywistą) operand o największej precyzji określa poziom precyzji wyniku.
- Operacja dzielenia przeprowadzana za pomocą operatora / używa obliczeń na 64-bitowych liczbach całkowitych, o ile wynik będzie używany jako liczba całkowita.

- Dzielenie dwóch dokładnych wartości liczbowych za pomocą operatora / ma skalę identyczną ze skalą dzielnej plus wartość zmiennej systemowej `div_precision_increment`, która domyślnie wynosi 4.
- Wynikiem operacji arytmetycznej z użyciem wartości NULL jest NULL.

Dostępne są przedstawione poniżej operatory arytmetyczne:

■ +

Dodawanie, oblicza sumę operandów.

<code>2 + 2</code>	→ 4
<code>3.2 + 4.7</code>	→ 7.9
<code>'43bc' + '21d'</code>	→ 64
<code>'abc' + 'def'</code>	→ 0

Ostatni przykład pokazuje, że operator + nie działa jako operator konkatencji ciągów tekstowych w sposób znany z innych języków programowania. Zamiast tego przed przeprowadzeniem działań arytmetycznych MySQL konwertuje ciągi tekstowe na liczby. Ciągi tekstowe nieprzypominające liczb są konwertowane na wartość 0. W celu łączenia ciągów tekstowych należy użyć funkcji `CONCAT()`.

■ -

Odejmowanie lub jednoargumentowy minus, oblicza różnicę operandów po umieszczeniu operatora między dwoma elementami wyrażenia lub negację operandu (to znaczy zmienia znak operandu) po umieszczeniu operatora na początku pojedynczego elementu.

<code>10 - 7</code>	→ 3
<code>-(10 - 7)</code>	→ -3

■ *

Mnożenie, oblicza iloczyn operandów.

<code>2 * 3</code>	→ 6
<code>2.3 * -4.5</code>	→ -10.35

■ /

Dzielenie, oblicza współczynnik operandów. Dzielenie przez zero powoduje wygenerowanie wyniku NULL.

<code>3 / 1</code>	→ 3.0000
<code>1 / 3</code>	→ 0.3333
<code>1 / 0</code>	→ NULL

■ DIV

Dzielenie liczb całkowitych, oblicza współczynnik operandów bez części ułamekowej. Dzielenie przez zero powoduje wygenerowanie wyniku NULL. Jeżeli którykolwiek operand nie jest liczbą całkowitą, wtedy oba operandy będą traktowane jako wartości `DECIMAL`, a wynik zostanie skonwertowany na liczbę całkowitą. Jeśli wynik będzie większy niż wartość `BIGINT`, nastąpi wygenerowanie błędu.

<code>3 DIV 1</code>	→ 3
<code>1 DIV 3</code>	→ 0
<code>1 DIV 0</code>	→ NULL

■ %, MOD

Operator modulo, oblicza resztę z dzielenia m przez n . Składnia $m \% n$ lub $m \text{ MOD } n$ odpowiada składni funkcji $\text{MOD}(m, n)$. Podobnie jak w przypadku dzielenia, jeśli dzielnik będzie miał wartość zero, wynikiem będzie NULL.

12 % 4	→ 0
12 % 5	→ 2
12 % 0	→ NULL

W przypadku wartości z częścią ułamkową operator modulo zwraca dokładną resztę z dzielenia.

14.4 % 3.2	→ 1.6
------------	-------

C.1.4. Operatory porównania

Operatory porównania zwracają wartość 1, jeśli wynikiem porównania jest prawda, oraz 0, gdy wynikiem porównania jest fałsz. Istnieje możliwość porównywania liczb z ciągami tekstowymi. Jeśli zachodzi potrzeba, to operandy są konwertowane według następujących reguł:

- W przypadku użycia operatora innego niż $<=>$ wynikiem porównania obejmującego wartość NULL jest NULL. (Operator $<=>$ jest jak $=$, za wyjątkiem faktu, że $\text{NULL} <=> \text{NULL}$ zwraca true, podczas gdy $\text{NULL} = \text{NULL}$ zwraca NULL).
- Jeżeli oba operandy są ciągami tekstowymi, są porównywane leksykalnie jak ciągi tekstowe. Binarne ciągi tekstowe są porównywane bajt po bajcie za pomocą wartości liczbowych poszczególnych bajtów. Porównania niebinarnych ciągów tekstowych są przeprowadzane znak po znaku na podstawie kolejności sortowania dla kodowania znaków stosowanego przez ciągi tekstowe. Jeżeli ciągi tekstowe mają różne kodowania znaków, wynikiem porównania może być błąd lub brak sensownego wyniku. Porównanie między binarnym i niebinarnym ciągiem tekstowym jest traktowane jako porównanie binarnych ciągów tekstowych.
- Jeżeli oba operandy są liczbami całkowitymi, wówczas są porównywane liczbowo jako liczby całkowite.
- Stałe szesnastkowe, które nie są porównywane do liczby, są porównywane jako binarne ciągi tekstowe.
- Inaczej niż w przypadku funkcji $\text{IN}()$, jeżeli którykolwiek operand jest wartością TIMESTAMP lub DATETIME , a drugi operand jest stałą, wtedy operandy są porównywane jako wartości TIMESTAMP . Przyjęto takie rozwiązanie, aby operacje porównania lepiej sprawdzały się w aplikacjach ODBC.
- Jeżeli jeden operand jest wartością dziesiętną, wtedy operandy są porównywane jako wartości dziesiętne, nawet jeśli drugi z nich jest wartością dziesiętną lub liczbą całkowitą. Jeśli drugi operand jest innego typu, wartości będą porównane jako liczby zmiennoprzecinkowe o podwójnej precyzji.

- W pozostałych przypadkach operandy będą porównane liczbowo jako wartości zmiennoprzecinkowe o podwójnej precyzji. Zwróć uwagę, że to dotyczy również porównywania ciągu tekstowego i liczby. Ciąg tekstowy zostaje skonwertowany na liczbę o podwójnej precyzji, co skutkuje wartością 0, jeśli ciąg tekstowy nie przypomina liczby. Na przykład, ciąg tekstowy '14.3' będzie skonwertowany na 14.3, natomiast '14.3' na 0.

Poniższe przykłady porównań ilustrują przedstawione reguły:

<code>2 < 12</code>	$\rightarrow 1$
<code>'2' < '12'</code>	$\rightarrow 0$
<code>'2' < 12</code>	$\rightarrow 1$

Pierwsze porównanie dotyczy dwóch liczb całkowitych i jest przeprowadzane liczbowo. Drugie porównanie obejmuje dwa ciągi tekstowe, które są porównywane leksykalnie. Z kolei w trzecim przykładzie porównywany jest ciąg tekstowy z liczbą, a więc ciąg tekstowy zostaje skonwertowany na liczbę o podwójnej precyzji i oba operandy są porównywane jako wartości o podwójnej precyzji.

MySQL przeprowadza porównanie ciągów tekstowych w następujący sposób: binarne ciągi tekstowe są porównywane bajt po bajcie za pomocą wartości liczbowych poszczególnych bajtów. Porównania niebinarnych ciągów tekstowych są przeprowadzane znak po znaku na podstawie kolejności sortowania dla kodowania znaków stosowanego przez ciągi tekstowe. Jeżeli ciągi tekstowe mają różne kodowania znaków, wynikiem porównania może być błąd lub brak sensownego wyniku. Porównanie między binarnym i niebinarnym ciągiem tekstowym jest traktowane jako porównanie binarnych ciągów tekstowych.

■ =

Jeśli operandy są równe, wynikiem jest 1. W przeciwnym razie wynikiem jest 0.

<code>1 = 1</code>	$\rightarrow 1$
<code>1 = 2</code>	$\rightarrow 0$
<code>'abc' = 'abc'</code>	$\rightarrow 1$
<code>'abc' = 'ABC'</code>	$\rightarrow 1$
<code>'abc' = 'def'</code>	$\rightarrow 0$
<code>'abc' = 0</code>	$\rightarrow 1$

W przypadku porównania ciągu tekstowego nierozróżniającego wielkości liter ciąg tekstowy 'abc' jest uznawany za taki sam jak 'abc' i 'ABC'. Wartość 'abc' wynosi 0, ponieważ jest konwertowana na liczbę, zgodnie z przedstawionymi wcześniej regułami. Ponieważ ciąg tekstowy 'abc' nie przypomina liczby, na potrzeby porównania zostaje skonwertowany na 0.

W przypadku niebinarnych ciągów tekstowych kolejność sortowania stosowana w operandach wpływa na porównywanie wartości znaków, które są podobne, ale różnią się wielkością liter lub znakami akcentu.

Operacje porównania ciągów tekstowych nie rozróżniają wielkości liter, o ile porównanie nie obejmuje binarnego ciągu tekstowego lub niebinarnego ciągu tekstowego wraz z kolejnością sortowania binarną bądź rozróżniającą wielkość liter. Na przykład, porównanie uwzględniające wielkość liter jest przeprowadzane,

jeśli używane jest słowo kluczowe BINARY lub porównywane są wartości kolumn BINARY, VARBINARY lub BLOB.

```
'abc' = 'ABC' → 1
BINARY 'abc' = 'ABC' → 0
BINARY 'abc' = 'abc' → 1
_latin1 'abc' COLLATE latin1_bin = 'ABC' → 0
_latin1 'abc' COLLATE latin1_general_cs = 'ABC' → 0
```

Spacje znajdujące się na końcu mają znaczenie w operacjach porównania binarnych ciągów tekstowych, ale nie w przypadku porównywania niebinarnych ciągów tekstowych.

```
BINARY 'a' = 'a ' → 0
'a' = 'a ' → 1
```

■ <=>

Równość bezpieczna w przypadku użycia wartości NULL. Ten operator jest podobny do =, za wyjątkiem faktu, że wynikiem jest wartość 1, gdy operandy są równe, nawet jeśli mają wartość NULL.

```
1 <=> 1 → 1
1 <=> 2 → 0
NULL <=> NULL → 1
NULL = NULL → NULL
```

Ostatnie dwa przykłady pokazują, jak operatory <=> i = w odmienny sposób obsługują porównania wartości NULL.

■ <>, !=

Jeśli operandy są nierówne, wynikiem jest 1. W przeciwnym razie wynikiem jest 0.

```
3,4 != 3.4 → 0
'abc' <> 'ABC' → 0
BINARY 'abc' <> 'ABC' → 1
'abc' != 'def' → 1
```

■ <

Jeśli lewy operand jest mniejszy niż prawy, wynikiem jest 1. W przeciwnym razie wynikiem jest 0.

```
3 < 10 → 1
105.4 < 10e+1 → 0
'abc' < 'ABC' → 0
'abc' < 'def' → 1
```

■ <=

Jeśli lewy operand jest mniejszy niż prawy lub mu równy, wynikiem jest 1. W przeciwnym razie wynikiem jest 0.

```
'abc' <= 'a' → 0
'a' <= 'abc' → 1
13.5 <= 14 → 1
(3 * 4) - (6 * 2) <= 0 → 1
```

■ >

Jeśli lewy operand jest większy niż prawy, wynikiem jest 1. W przeciwnym razie wynikiem jest 0.

```
PI() > 3 → 1
'abc' > 'a' → 1
SIN(0) > COS(0) → 0
```

■ >=

Jeśli lewy operand jest większy niż prawy lub mu równy, wynikiem jest 1. W przeciwnym razie wynikiem jest 0.

```
'abc' >= 'a' → 1
'a' >= 'abc' → 0
13.5 >= 14 → 0
(3 * 4) - (6 * 2) >= 0 → 1
```

■ *expr* BETWEEN *min* AND *max*
expr NOT BETWEEN *min* AND *max*

Wynikiem działania operatora BETWEEN jest 1, jeśli *expr* znajduje się w zakresie wartości od *min* do *max* (włącznie). W przeciwnym razie wynikiem jest 0. Wynik działania operatora NOT BETWEEN jest odwrotny do BETWEEN. Jeżeli operandy wyrażenia (*expr*), a także *min* i *max* są takiego samego typu, wtedy wymienione poniżej wyrażenia odpowiadają sobie:

```
expr BETWEEN min AND max
(min <= expr AND expr <= max)
```

Jeżeli operandy nie są takiego samego typu, następuje konwersja typu i powyższe wyrażenia mogą już nie być takie same. Wówczas porównanie jest przeprowadzane z użyciem reguł przedstawionych na początku tego punktu.

```
'def' BETWEEN 'abc' AND 'ghi' → 1
'def' BETWEEN 'abc' AND 'def' → 1
13.3 BETWEEN 10 AND 20 → 1
13.3 BETWEEN 10 AND 13 → 0
2 BETWEEN 2 AND 2 → 1
'B' BETWEEN 'A' AND 'a' → 0
BINARY 'B' BETWEEN 'A' AND 'a' → 1
```

W przypadku wyrażeń BETWEEN używających połączenia typów daty i godziny oraz połączenia typów daty i godziny oraz ciągów tekstowych najlepszym rozwiązaniem jest użycie funkcji CAST() w celu zagwarantowania, że wszystkie operandy są tego samego typu.

■ CASE [*expr*] WHEN *expr1* THEN *result1* ... [ELSE *default*] END

Kiedy wyrażenie początkowe (*expr*) jest obecne, operator CASE porównuje je do wyrażenia znajdującego się po WHEN. Dla pierwszej znalezionej równości odpowiadająca jej wartość THEN stanie się wynikiem. To jest użyteczne podczas porównywania danej wartości do zbioru wartości.

```
CASE 0 WHEN 1 THEN 'T' WHEN 0 THEN 'F' END → 'F'
CASE 'F' WHEN 'T' THEN 1 WHEN 'F' THEN 0 END → 0
```

Kiedy wyrażenie początkowe (*expr*) jest nieobecne, operator CASE oblicza wyrażenia WHEN. Dla pierwszego zwracającego wartość true (wartość niezerowa i inna niż NULL) odpowiadającą mu wartość THEN stanie się wynikiem. To jest użyteczne do przeprowadzania testów nierówności lub sprawdzania różnych warunków.

```
CASE WHEN 1=0 THEN 'absurd' WHEN 1=1 THEN 'oczywistość' END
                                → 'oczywistość'
```

Jeżeli nie zostanie dopasowane żadne wyrażenie WHEN, wynikiem będzie wartość ELSE. W przypadku braku ELSE wynikiem obliczenia CASE jest NULL.

```
CASE 0 WHEN 1 THEN 'true' ELSE 'false' END      → 'false'
CASE 0 WHEN 1 THEN 'true' END                    → NULL
CASE WHEN 1=0 THEN 'true' ELSE 'false' END      → 'false'
CASE WHEN 1/0 THEN 'true' END                    → NULL
```

Typ zwrotny dla wyrażenia CASE jest domyślnie określany na podstawie zagregowanych typów wartości zwrotnej.

```
CASE 1 WHEN 0 THEN 0 ELSE 1 END                  → 1
CASE 1 WHEN 0 THEN '0' ELSE '1' END              → '1'
```

Jednak na domyślny typ wartości zwrotnej wpływ ma również kontekst, który może spowodować przeprowadzenie konwersji na ciąg tekstowy, liczbę itd.

Zwróć uwagę, że wyrażenie CASE różni się od polecenia CASE omówionego w punkcie E.2.1, zatytułowanym „Polecenia struktury kontrolnej”.

- *expr* IN (*wartość1*,*wartość2*,...)
- expr* NOT IN (*wartość1*,*wartość2*,...)

Wynikiem działania IN() jest 1, jeśli wyrażenie (*expr*) będzie jedną z wartości wymienionych na liście. W przeciwnym razie wynikiem jest 0. Wynik działania NOT IN() jest odwrotny niż w IN(). Poniższe wyrażenia odpowiadają sobie:

```
expr NOT IN (wartość1, wartość2,...)
NOT (expr IN (wartość1, wartość2,...))
```

Jeżeli wszystkie wartości na liście są stałymi, MySQL sortuje je i oblicza IN(), przeprowadzając wyszukiwanie binarne, które jest bardzo szybkie.

```
3 IN (1,2,3,4,5)                → 1
'd' IN ('a','b','c','d','e')    → 1
'f' IN ('a','b','c','d','e')    → 0
3 NOT IN (1,2,3,4,5)            → 0
'd' NOT IN ('a','b','c','d','e') → 0
'f' NOT IN ('a','b','c','d','e') → 1
```

- *expr* IS {FALSE | TRUE | UNKNOWN}

Ta konstrukcja sprawdza wyrażenie (*expr*) względem logicznego fałszu, prawdy i nieznanego wartości, a jej wartością zwrotną jest 0 (fałsz) lub 1 (prawda).

Wartość 0 jest uznawana za fałsz, wartość niezerowa lub inna niż NULL są uznawane za prawdę, natomiast NULL jest wartością nieznaną.

```
2 IS FALSE                → 0
2 IS TRUE                 → 1
2 IS UNKNOWN              → 0
```

NULL IS FALSE	→ 0
NULL IS TRUE	→ 0
NULL IS UNKNOWN	→ 1

■ *expr* IS NULL

expr IS NOT NULL

Wynikiem działania IS NULL jest 1, jeśli wyrażenie (*expr*) będzie miało wartość NULL. W przeciwnym razie wynikiem jest 0. Wynik działania NOT NULL jest odwrotny niż w IS NULL. Poniższe wyrażenia odpowiadają sobie:

expr IS NOT NULL
NOT (*expr* IS NULL)

Operatory IS NULL i IS NOT NULL powinny być używane do ustalenia, czy wartością wyrażenia (*expr*) jest NULL. Do tego celu nie można wykorzystać zwykłych operatorów porównania równości i nierówności (=, <, !=). Jednak do sprawdzenia wartości NULL można użyć operatora <=>.

NULL IS NULL	→ 1
0 IS NULL	→ 0
NULL IS NOT NULL	→ 0
0 IS NOT NULL	→ 1
NOT (0 IS NULL)	→ 1
NOT (NULL IS NULL)	→ 0

C.1.5. Operatory bitowe

W tym punkcie zostaną przedstawione operatory przeprowadzające operacje bitowe. Wspomniane operacje bitowe są przeprowadzane na wartościach typu BIGINT (64-bitowe liczby całkowite), co ogranicza maksymalny zakres operacji. Wynikiem operacji bitowych są 64-bitowe wartości bez znaku lub NULL, jeśli którykolwiek z operandów ma wartość NULL.

■ &

Oblicza bitowe AND (intersekcja) operandów.

1 & 1	→ 1
1 & 2	→ 0
7 & 5	→ 5

■ |

Oblicza bitowe OR (unia) operandów.

1 1	→ 1
1 2	→ 3
1 2 4 8	→ 15
1 2 4 8 15	→ 15

■ ^

Oblicza bitowe XOR (wykluczające OR) operandów.

1 ^ 1	→ 0
1 ^ 0	→ 1
255 ^ 127	→ 128

■ <<

Operand znajdujący się po lewej stronie zostaje przesunięty w lewą stronę o liczbę pozycji bitowych wskazanych przez prawy operand. Przesunięcie o wartość ujemną powoduje, że wynikiem jest zero.

```
1 << 2           → 4
2 << 2           → 8
1 << 63          → 9223372036854775808
1 << 64          → 0
```

Ostatnie dwa przykłady pokazują ograniczenia obliczeń przeprowadzanych na liczbach 64-bitowych.

■ >>

Operand znajdujący się po lewej stronie zostaje przesunięty w prawą stronę o liczbę pozycji bitowych wskazanych przez prawy operand. Przesunięcie o wartość ujemną powoduje, że wynikiem jest zero.

```
16 >> 3          → 2
16 >> 4          → 1
16 >> 5          → 0
```

■ ~

Przeprowadza bitową negację (inwersję) operandu znajdującego się po operatorze. Oznacza to, że bity 0 stają się bitami 1 i na odwrót.

```
~0               → 18446744073709551615
~(-1)            → 0
~~(-1)           → 18446744073709551615
```

C.1.6. Operatory logiczne

Operatory logiczne (nazywane także „operatorami boolowskimi”) sprawdzają prawdziwość lub fałsz wyrażeń. Wartością zwrotną operacji logicznej jest 1 dla prawdy, 0 dla fałszu i NULL w przypadku nieznanego wyniku. Operatory logiczne interpretują operandy niezerowe i inne niż NULL jako prawdę, 0 jako fałsz i NULL jako wartość nieznaną.

Operatory logiczne oczekują, że operandy będą liczbami. Dlatego też operandy w postaci ciągów tekstowych są konwertowane na postać liczb w trakcie obliczania wartości operatora.

W MySQL znaki `!`, `||` i `&&` oznaczają operacje logiczne, podobnie jak w języku C. Warto w szczególności zapamiętać, że `||` nie oznacza konkatencji ciągu tekstowego, jak ma to miejsce w standardowym SQL. Do konkatencji ciągów tekstowych należy używać funkcji `CONCAT()`. Aby znaki `||` były traktowane jako operator konkatencji, należy włączyć tryb SQL o nazwie `PIPES_AS_CONCAT`.

■ NOT, !

Logiczna negacja. Wynikiem będzie 1, jeśli operand znajdujący się po operatorze przyjmuje wartość `false`, i zero, jeśli operand przyjmuje wartość `true`. Wyjątkiem jest to, że `NOT NULL` przyjmuje wartość `NULL`.

NOT 0	→ 1
NOT 1	→ 0
NOT NULL	→ NULL
NOT 3	→ 0
NOT NOT 1	→ 1
NOT '1'	→ 0
NOT '0'	→ 1
NOT 'abc'	→ 1

Ostatnie kilka przykładów pokazuje konwersję operandu w postaci ciągu tekstowego na liczbę przed przeprowadzeniem obliczeń.

Pierwszeństwo operatora NOT można zmienić zgodnie z opisem przedstawionym w punkcie C.1.1, zatytułowanym „Kolejność operatorów”.

■ AND, &&

Logiczne AND. Wynikiem będzie 1, jeśli oba operandy przyjmują wartość true (wartość niezerowa i inna niż NULL), 0, jeśli którykolwiek operand przyjmie wartość false, i NULL w pozostałych przypadkach (brak możliwości określenia wyniku).

4 AND 2	→ 1
0 AND 0	→ 0
0 AND 3	→ 0
1 AND NULL	→ NULL
0 AND NULL	→ 0
NULL AND NULL	→ NULL

■ OR, ||

Logiczne OR. Wynikiem będzie 1, jeśli którykolwiek operand przyjmie wartość true (wartość niezerowa i inna niż NULL), 0, jeśli oba operandy przyjmą wartość false, i NULL w pozostałych przypadkach (brak możliwości określenia wyniku).

4 OR 2	→ 1
0 OR 3	→ 1
0 OR 0	→ 0
1 OR NULL	→ 1
0 OR NULL	→ NULL
NULL OR NULL	→ NULL

■ XOR

Logiczne wykluczające OR. Wynikiem będzie 1, jeśli dokładnie jeden operand przyjmie wartość true (wartość niezerowa i inna niż NULL), 0 w pozostałych przypadkach. Wartością będzie NULL (wartość nieznana), jeśli którykolwiek operand przyjmie wartość NULL.

0 XOR 0	→ 0
0 XOR 9	→ 1
7 XOR 0	→ 1
5 XOR 2	→ 0

C.1.7. Operatory rzutowania

Operatory rzutowania zmieniają sposób interpretacji wartości lub przeprowadzają jej konwersję z jednego typu na inny.

■ `_charset str`

Operator `_charset` jest nazywany „introducer”. Powoduje, że umieszczony po nim ciąg tekstowy lub wartość kolumny będą interpretowane za pomocą wskazanego kodowania znaków. *charset* musi być nazwą kodowania znaków obsługiwanego przez serwer. Na przykład, poniższe wyrażenia powodują interpretację ciągu tekstowego 'abcd' za pomocą kodowania znaków `latin2` lub `utf8`:

```
_latin2 'abcd'
_utf8 'abcd'
```

W przypadku wielobajtowych kodowań znaków może wystąpić dopełnienie wyniku, jeśli ostatni operand nie zawiera prawidłowej liczby bajtów do utworzenia pełnego znaku.

■ `BINARY str`

Operator `BINARY` powoduje, że znajdujący się po nim operand będzie traktowany jako binarny ciąg tekstowy. Porównania wyniku będą przeprowadzane bajt po bajcie za pomocą wartości liczbowych poszczególnych bajtów. Jeżeli operand znajdujący się po operatorze jest liczbą, zostanie skonwertowany na postać ciągu tekstowego:

```
'abc' = 'ABC'           → 1
'abc' = BINARY 'ABC'    → 0
BINARY 'abc' = 'ABC'    → 0
'2' < 12                → 1
'2' < BINARY 12         → 0
```

W ostatnim przykładzie operator `BINARY` powoduje przeprowadzenie konwersji liczby na postać ciągu tekstowego. Następnie operandy są porównywane jak binarne ciągi tekstowe.

■ `str COLLATE kolejność_sortowania`

Operator `COLLATE` powoduje, że dany ciąg tekstowy (*str*) będzie stosował wskazaną kolejność sortowania (która musi być jedną z dozwolonych kolejności sortowania dla kodowania znaków użytego w ciągu tekstowym *str*). Operator `COLLATE` wpływa na przebieg porównań, sortowania, grupowania i wykonania zapytania `SELECT DISTINCT`.

```
SELECT ... WHERE utf8_str COLLATE utf8_icelandic_ci > 'M';
SELECT MAX(greek_str COLLATE greek_general_ci) FROM ... ;
SELECT ... GROUP BY latin1_str COLLATE latin1_german2_ci;
SELECT ... ORDER BY sjis_str COLLATE sjis_bin;
SELECT DISTINCT latin2_str COLLATE latin2_croatian_ci FROM ...;
```


C.1.8. Operatory dopasowania wzorca

MySQL oferuje obsługę dopasowania wzorca SQL za pomocą klauzuli LIKE oraz dopasowanie wzorca wyrażeń regularnych za pomocą REGEXP. Dopasowanie wzorca SQL kończy się powodzeniem tylko wtedy, gdy wzorec dopasuje cały ciąg tekstowy przeznaczony do dopasowania. Z kolei dopasowanie wzorca wyrażenia regularnego kończy się powodzeniem, jeśli wzorec zostanie znaleziony w dowolnym miejscu ciągu tekstowego.

W podpunkcie 3.5.1.1, zatytułowanym „Typy operatorów”, znajdziesz więcej informacji oraz przykładów dopasowania wzorca.

- `str LIKE wzorec [ESCAPE 'c']`
`str NOT LIKE wzorec [ESCAPE 'c']`

Klauzula LIKE pozwala na dopasowanie wzorca SQL. Wynikiem działania będzie 1, jeśli ciąg tekstowy *wzorec* zostanie dopasowany do całego ciągu tekstowego wyrażenia (*str*). Jeśli wzorec nie zostanie dopasowany, wynikiem będzie 0. Wynik działania NOT LIKE jest odwrotny niż w przypadku LIKE.

Poniższe wyrażenia odpowiadają sobie:

```
str NOT LIKE wzorec [ESCAPE 'c']
NOT (str LIKE wzorec [ESCAPE 'c'])
```

Wynikiem będzie NULL, gdy którykolwiek z operandów ma wartość NULL.

Dwa znaki mają znaczenie specjalne we wzorcach SQL i działają w charakterze znaków wieloznacznych:

- ◆ Znak procentu (%) powoduje dopasowanie dowolnej sekwencji znaków (w tym także pustego ciągu tekstowego) innych niż NULL;
- ◆ Znak podkreślenia (_) powoduje dopasowanie pojedynczego znaku.

Wzorce mogą zawierać dowolny lub oba wymienione znaki wieloznaczne.

```
'catnip' LIKE 'cat%'           → 1
'dogwood' LIKE '%wood'        → 1
'bird' LIKE '____'            → 1
'bird' LIKE '____'            → 0
'dogwood' LIKE '%wo__'        → 1
```

Operator LIKE porównuje ciągi tekstowe jako binarne, gdy którykolwiek z operandów jest binarnym ciągiem tekstowym. W przypadku, gdy operandy są niebinarnymi ciągami tekstowymi, używana jest kolejność sortowania operandu.

```
'abc' LIKE 'ABC'               → 1
BINARY 'abc' LIKE 'ABC'        → 0
'abc' LIKE BINARY 'ABC'        → 0
'abc' LIKE 'ABC' COLLATE latin1_general_ci → 1
'abc' LIKE 'ABC' COLLATE latin1_general_cs → 0
```

Ponieważ znak % powoduje dopasowanie dowolnej sekwencji znaków, może także nie dopasować żadnego znaku.

```
' ' LIKE '%'                   → 1
'cat' LIKE 'cat%'              → 1
```

W MySQL operatora LIKE można używać w wyrażeniach liczbowych.

```
50 + 50 LIKE '1%'           → 1
200 LIKE '2__'              → 1
```

W celu dopasowania dosłownego znaku wieloznacznego trzeba wyłączyć jego znaczenie specjalne w ciągu tekstowym wzorca przez poprzedzenie znaku ukośnikiem \.

```
'100% pure' LIKE '100%'      → 1
'100% pure' LIKE '100\%'     → 0
'100% pure' LIKE '100\% pure' → 1
```

Aby dosłownie interpretować ukośnik \, konieczne jest włączenie trybu SQL o nazwie NO_BACKSLASH_ESCAPES. Alternatywnym rozwiązaniem jest użycie klauzuli ESCAPE.

```
'100% pure' LIKE '100^%' ESCAPE '^' → 0
'100% pure' LIKE '100^% pure' ESCAPE '^' → 1
```

Po włączeniu trybu SQL o nazwie NO_BACKSLASH_ESCAPES klauzula ESCAPE nie może wskazywać pustego ciągu tekstowego.

- *str* REGEXP *wzorzec*
str NOT REGEXP *wzorzec*

Operator REGEXP przeprowadza dopasowanie wzorca wyrażenia regularnego. Wynikiem działania będzie 1, jeśli ciąg tekstowy *wzorzec* zostanie dopasowany do ciągu tekstowego wyrażenia (*str*). Jeśli wzorzec nie zostanie dopasowany, wynikiem będzie 0. Wynik działania NOT REGEXP jest odwrotny niż w przypadku REGEXP. Poniższe wyrażenia odpowiadają sobie:

```
str NOT REGEXP wzorzec
NOT (str REGEXP wzorzec)
```

Wynikiem będzie NULL, gdy którykolwiek z operandów ma wartość NULL.

Operator REGEXP porównuje ciągi tekstowe jako binarne, gdy którykolwiek z operandów jest binarnym ciągiem tekstowym. W przypadku, gdy operandy są niebinarnymi ciągami tekstowymi, używana jest kolejność sortowania operandu.

```
'abc' REGEXP 'ABC'           → 1
BINARY 'abc' REGEXP 'ABC'    → 0
'abc' REGEXP BINARY 'ABC'    → 0
'abc' REGEXP 'ABC' COLLATE latin1_bin → 0
'abc' COLLATE latin1_bin REGEXP 'ABC' → 0
```

Operator REGEXP nie zapewnia bezpieczeństwa dla wielobajtowych kodowań znaków i działa jedynie z jednobajtowymi kodowaniami znaków.

Wyrażenia regularne są podobne do wzorców stosowanych w narzędziach systemu UNIX o nazwach grep i sed. Dozwolone sekwencje wzorców wymieniono w tabeli C.1.

Tabela C.1. Sekwencje wzorców, które można stosować w operatorze REGEXP

Element	Opis
<code>^</code>	Dopasowanie początku ciągu tekstowego.
<code>\$</code>	Dopasowanie końca ciągu tekstowego.
<code>.</code>	Dopasowanie pojedynczego znaku, włączając znak nowego wiersza.
<code>[...]</code>	Dopasowanie dowolnego znaku pojawiającego się pomiędzy nawiasami.
<code>[^...]</code>	Dopasowanie dowolnego znaku niepojawiającego się pomiędzy nawiasami.
<code>e*</code>	Dopasowanie zero lub więcej wystąpień wzorca elementu <i>e</i> .
<code>e+</code>	Dopasowanie jednego lub więcej wystąpień wzorca elementu <i>e</i> .
<code>e?</code>	Dopasowanie zero lub jednego wystąpienia wzorca elementu <i>e</i> .
<code>e1 e2</code>	Dopasowanie wzorca elementu <i>e1</i> lub <i>e2</i> .
<code>e{m}</code>	Dopasowanie <i>m</i> wystąpień wzorca elementu <i>e</i> .
<code>e{m,}</code>	Dopasowanie <i>m</i> lub więcej wystąpień wzorca elementu <i>e</i> .
<code>e{,n}</code>	Dopasowanie zero do <i>n</i> wystąpień wzorca elementu <i>e</i> .
<code>e{m,n}</code>	Dopasowanie <i>m</i> do <i>n</i> wystąpień wzorca elementu <i>e</i> .
<code>(...)</code>	Zgrupowanie elementów wzorca na postać pojedynczego elementu.
<i>inne</i>	Zwykłe znaki dopasowujące same siebie.

Wzorec wyrażenia regularnego nie musi dopasować całego ciągu tekstowego; wystarczy, że zostanie znaleziony w dowolnym miejscu ciągu tekstowego.

```
'cats and dogs' REGEXP 'dogs'      → 1
'cats and dogs' REGEXP 'cats'      → 1
'cats and dogs' REGEXP 'c.*a.*d'   → 1
'cats and dogs' REGEXP 'o'         → 1
'cats and dogs' REGEXP 'x'         → 0
```

Użyj znaku `^` lub `$` w celu wymuszenia dopasowania jedynie na początku lub końcu ciągu tekstowego.

```
'abcde' REGEXP 'b'      → 1
'abcde' REGEXP '^b'     → 0
'abcde' REGEXP 'b$'     → 0
'abcde' REGEXP '^a'     → 1
'abcde' REGEXP 'e$'     → 1
'abcde' REGEXP '^a.*e$' → 1
```

Konstrukcje `[...]` i `[^...]` pozwalają na określenie klas znaków. W ramach klasy zakres znaków można wskazać za pomocą myślnika umieszczonego między pierwszym i ostatnim znakiem zakresu. Na przykład, konstrukcja `[a-z]` powoduje dopasowanie dowolnej małej litery z zakresu od *a* do *z*, natomiast `[0-9]` dopasowuje dowolną cyfrę dziesiętną.

```
'bin' REGEXP '^b[aeiou]n$' → 1
'bxn' REGEXP '^b[aeiou]n$' → 0
```

```
'oboeist' REGEXP '^ob[aeiou]+st$'           → 1
'wolf359' REGEXP '[a-z]+[0-9]+'               → 1
'wolf359' REGEXP '[0-9a-z]+'                 → 1
'wolf359' REGEXP '[0-9]+[a-z]+'              → 0
```

Aby dołączyć do klasy dosłowny znak], musi on być pierwszym znakiem klasy. Aby dołączyć minus (-) do klasy, ten znak musi być podany jako pierwszy lub ostatni. Z kolei by dołączyć do klasy dosłowny znak ^, nie może on być pierwszym znakiem po znaku [.

Dostępnych jest również wiele specjalnych konstrukcji klas znaków wyrażeń regularnych typu POSIX. Zostały one wymienione w tabeli C.2.

Tabela C.2. Klasy znaków wyrażeń regularnych POSIX, które można stosować w operatorze REGEXP

Klasa	Opis
[a1num:]	Znaki alfanumeryczne.
[alpha:]	Litery.
[blank:]	Znak odstępu (spacje lub tabulatory).
[cntrl:]	Znaki sterujące.
[digit:]	Cyfry dziesiętne (0–9).
[graph:]	Niewidoczne znaki graficzne.
[lower:]	Małe litery.
[print:]	Znaki graficzne lub odstępu.
[punct:]	Znaki przestankowe.
[space:]	Spacja, tabulator, znak nowego wiersza lub wysuw wiersza.
[upper:]	Wielkie litery.
[xdigit:]	Cyfry szesnastkowe (0–9, a–f, A–F).

Konstruktory klas POSIX są używane w klasach znaków, a nazwy wspomnianych klas zawierają znaki [i]. Dlatego też podczas tworzenia wyrażeń klas znaków odwołujących się do klas wymienionych w tabeli C.2 upewnij się, że została podana odpowiednia liczba nawiasów.

```
'abc' REGEXP '[:space:]'           → 0
'a c' REGEXP '[:space:]'           → 1
'abc' REGEXP '[:digit:][:punct:]' → 0
'a0c' REGEXP '[:digit:][:punct:]' → 1
'a,c' REGEXP '[:digit:][:punct:]' → 1
```

W klasach znaków znaczniki specjalne [:<:] i [:>:] powodują dopasowanie odpowiednio początku i końca granicy słowa. Za znak słowa jest uznawany dowolny znak w klasie a1num lub znak podkreślenia. Słowo składa się z jednego lub więcej znaków słowa, przed i po którym nie znajduje się żaden inny znak słowa.

```
'a few words' REGEXP '[:<:]few[:>:]' → 1
'a few words' REGEXP '[:<:]fe[:>:]'  → 0
```

W celu kodowania sekwencji w ciągach tekstowych wyrażenia regularnego MySQL używa składni podobnej do stosowanej w języku C. Na przykład, `\n`, `\t` i `\\` są interpretowane jako znak nowego wiersza, tabulator i ukośnik `\`. Aby wymienionych znaków użyć we wzorcu, konieczne jest poprzedzenie ich ukośnikiem (`\\n`, `\\t`, `\\\\`). Jeden ukośnik zostanie usunięty w trakcie przetwarzania zapytania, interpretacja pozostałej części zakodowanej sekwencji nastąpi w trakcie operacji dopasowania wzorca.

- `str` RLIKE *pattern*
`str` NOT RLIKE *pattern*

Operatory RLIKE i NOT RLIKE są synonimami dla REGEXP i NOT REGEXP.

C.2. Funkcje

Funkcje są wywoływane w celu przeprowadzenia pewnych obliczeń i zwrócenia wartości. Domyślnie funkcja musi być wywołana bez spacji między nazwą funkcji i nawiasem otwierającym. W przeciwnym razie zostanie wygenerowany błąd.

```
mysql> SELECT NOW();
+-----+
| NOW() |
+-----+
| 2013-01-08 15:34:46 |
+-----+
mysql> SELECT NOW ();
ERROR 1630 (42000): FUNCTION NOW does not exist
```

Po włączeniu trybu SQL o nazwie IGNORE_SPACE serwer zezwala na stosowanie spacji po nazwach funkcji wbudowanych, ale efektem ubocznym jest to, że nazwy wszystkich funkcji stają się słowami zarezerwowanymi. Pewne programy mogą pozwalać na stosowanie spacji po nazwie funkcji, nawet jeśli nie został włączony wymieniony tryb SQL. Na przykład, program `mysql` możesz uruchomić wraz z opcją `--ignore-space`, a w programach utworzonych w języku C można wywołać funkcję `mysql_real_connect()` wraz z opcją `CLIENT_IGNORE_SPACE`.

W większości przypadków wiele argumentów przekazywanych funkcji jest rozdzielonych przecinkami. Spacje są dozwolone do stosowania wokół argumentów funkcji. Oba przedstawione poniżej wiersze pokazują prawidłowe wywołanie funkcji:

```
CONCAT('abc','def')
CONCAT( 'abc' , 'def' )
```

Istnieje kilka wyjątków używających składni alternatywnej, na przykład `TRIM()` i `EXTRACT()`:

```
TRIM(' ' FROM ' x ' )      → 'x'
EXTRACT(YEAR FROM '2018-01-01') → 2018
```

W opisie każdej funkcji przedstawiono składnię jej wywołania.

C.2.1. Funkcje porównań

Wymienione w tym punkcie funkcje przeprowadzają operacje porównywania wartości.

■ `ELT(n,str1,str2,...)`

Zwraca *n*-ty element listy ciągów tekstowych *str1*, *str2*,... Wartością zwrótną będzie NULL, jeśli *n* ma wartość NULL, *n*-ty ciąg tekstowy to NULL lub gdy nie ma *n*-tego ciągu tekstowego. Indeks pierwszego ciągu tekstowego to 1. Funkcja `ELT()` jest uzupełnieniem funkcji `FIELD()`.

<code>ELT(3,'a','b','c','d','e')</code>	→ 'c'
<code>ELT(0,'a','b','c','d','e')</code>	→ NULL
<code>ELT(6,'a','b','c','d','e')</code>	→ NULL
<code>ELT(FIELD('b','a','b','c'),'a','b','c')</code>	→ 'b'

■ `FIELD(arg0,arg1,arg2,...)`

Wyszukuje *arg0* na liście argumentów *arg1*, *arg2*,... i zwraca indeks dopasowanego argumentu (indeksy rozpoczynają się od 1). Wartością zwrótną będzie 0 w przypadku braku dopasowania lub jeśli *arg0* wynosi NULL. Jeżeli wszystkie argumenty są ciągami tekstowymi, zastosowane będzie porównanie ciągów tekstowych. Gdy wszystkie argumenty są liczbami, zastosowane będzie porównanie liczbowe. W pozostałych przypadkach porównywane są wartości o podwójnej precyzji. Funkcja `FIELD()` jest uzupełnieniem funkcji `ELT()`.

<code>FIELD('b','a','b','c')</code>	→ 2
<code>FIELD('d','a','b','c')</code>	→ 0
<code>FIELD(NULL,'a','b','c')</code>	→ 0
<code>FIELD(ELT(2,'a','b','c'),'a','b','c')</code>	→ 2

■ `GREATEST(expr1,expr2,...)`

Zwraca największy argument, przy czym słowo „największy” jest definiowane zgodnie z wymienionymi poniżej regułami:

- ◆ Wynikiem będzie NULL, jeśli którykolwiek argument ma wartość NULL.
- ◆ Jeżeli funkcja została wywołana w kontekście liczb całkowitych lub jej wszystkie argumenty są liczbami całkowitymi, wtedy argumenty będą porównywane jako liczby całkowite.
- ◆ Jeżeli funkcja została wywołana w kontekście liczb zmiennoprzecinkowych lub jej wszystkie argumenty są wartościami zmiennoprzecinkowymi, wtedy argumenty będą porównywane jako wartości zmiennoprzecinkowe.
- ◆ Jeżeli nie zostanie użyta żadna z wcześniej wymienionych reguł, argumenty będą porównywane jako ciągi tekstowe. Stosowane będą reguły porównywania ciągów tekstowych wymienione w punkcie C.1.4, zatytułowanym „Operatory porównania”.

<code>GREATEST(2,3,1)</code>	→ 3
<code>GREATEST(38.5,94.2,-1)</code>	→ 94.2
<code>GREATEST('a','ab','abc')</code>	→ 'abc'
<code>GREATEST(1,3,5)</code>	→ 5
<code>GREATEST('A','b','C')</code>	→ 'C'
<code>GREATEST(BINARY 'A','b','C')</code>	→ 'b'

■ IF(*expr1*,*expr2*,*expr3*)

Jeżeli *expr1* przyjmuje wartość true (wartość niezerowa i inna niż NULL), wtedy funkcja zwraca *expr2*; w przeciwnym razie funkcja zwraca *expr3*. Typ wartości zwrótej funkcji IF() jest określany na podstawie następujących testów (kolejno): ciąg tekstowy, jeśli *expr2* lub *expr3* to ciąg tekstowy; wartość zmiennoprzecinkowa, jeśli którykolwiek argument jest liczbą zmiennoprzecinkową; liczba całkowita, jeśli dowolny argument jest liczbą całkowitą.

IF(1,'true','false')	→ 'true'
IF(0,'true','false')	→ 'false'
IF(NULL,'true','false')	→ 'false'
IF(1.3,'nonzero','zero')	→ 'nonzero'
IF(0.3 <> 0,'nonzero','zero')	→ 'nonzero'

Zwróć uwagę, że funkcja IF() różni się od polecenia IF omówionego w punkcie E.2.1, zatytułowanym „Polecenia struktury kontrolnej”.

■ IFNULL(*expr1*,*expr2*)

Zwraca *expr2*, jeśli wartością wyrażenia *expr1* jest NULL, w przeciwnym razie zwracane jest wyrażenie *expr1*. Typem wartości zwrótej funkcji IFNULL() jest liczba lub ciąg tekstowy, zależnie od kontekstu wywołania funkcji.

IFNULL(NULL,'null')	→ 'null'
IFNULL('not null','null')	→ 'not null'

■ INTERVAL(*n*,*n1*,*n2*,...)

Zwraca 0, jeśli $n < n1$, 1, jeśli $n < n2$ itd. lub -1, jeśli wartością *n* jest NULL. Funkcja INTERVAL() wyszukuje położenie pierwszego argumentu w odstępach zdefiniowanych przez pozostałe argumenty. Wszystkie argumenty muszą być liczbami całkowitymi. Wartości *n1*, *n2*,... bezwzględnie muszą być podane w kolejności rosnącej ($n1 < n2 < \dots$), ponieważ używane jest szybkie wyszukiwanie binarne. W przeciwnym razie zachowanie funkcji INTERVAL() będzie nieprzewidywalne.

INTERVAL(2,0,1,3)	→ 2
INTERVAL(7,1,3,5,7,9)	→ 4

■ ISNULL(*expr*)

Zwraca 1, jeśli wartością wyrażenia *expr* jest NULL, w przeciwnym razie funkcja zwraca 0.

ISNULL(NULL)	→ 1
ISNULL(0)	→ 0
ISNULL(1)	→ 0

■ LEAST(*expr1*,*expr2*,...)

Zwraca najmniejszy argument, przy czym słowo „najmniejszy” jest definiowane zgodnie z regułami wymienionymi w opisie funkcji GREATEST().

LEAST(2,3,1)	→ 1
LEAST(38.5,94.2,-1)	→ -1
LEAST('a','ab','abc')	→ 'a'

■ **NULLIF(*expr1*,*expr2*)**

Zwraca *expr1*, jeśli wartości obu wyrażeń są różne. W przypadku gdy są takie same, wartością zwrótną jest NULL.

```
NULLIF(3,4)           → 3
NULLIF(3,3)           → NULL
```

■ **STRCMP(*str1*,*str2*)**

Ta funkcja zwraca 1, 0 lub -1, w zależności od tego, czy pierwszy argument jest leksykalnie odpowiednio większy niż drugi, równy mu lub od niego mniejszy. Jeżeli którykolwiek argument ma wartość NULL, wtedy funkcja również zwraca NULL. Funkcja STRCMP() porównuje ciągi tekstowe jako binarne, gdy którykolwiek z argumentów jest binarnym ciągiem tekstowym. W przypadku, gdy argumenty są niebinarnymi ciągami tekstowymi, używana jest kolejność sortowania argumentu.

```
STRCMP('a','a')       → 0
STRCMP('a','A')       → 0
STRCMP(BINARY 'a','A') → 1
STRCMP('A' COLLATE latin1_general_ci,'a') → 0
STRCMP('A' COLLATE latin1_general_cs,'a') → -1
```

C.2.2. Funkcje rzutowania

Wymienione w tym punkcie funkcje przeprowadzają konwersje wartości z jednego typu na inny.

■ **CAST(*expr AS typ*)**

Rzutuje wartość wyrażenia *expr* na wskazany typ. Wartością *typ* może być BINARY[(*n*)] (binarny ciąg tekstowy), CHAR[(*n*)] (niebinarny ciąg tekstowy), DATE, DATETIME, TIME, SIGNED [INTEGER], UNSIGNED [INTEGER] lub DECIMAL[(*M*),(*D*)].

```
CAST(304 AS BINARY)       → '304'
CAST(-1 AS UNSIGNED)     → 18446744073709551615
CAST(13 AS DECIMAL(5,2)) → 13.00
```

Dla typów BINARY i CHAR można podać opcjonalną długość (*n*), która powoduje, że wynik nie będzie dłuższy niż *n* odpowiednio bajtów lub znaków. W przypadku typu BINARY wartości mniejsze niż *n* bajtów będą dopełnione do długości *n* bajtami 0x00.

Funkcja CAST() może być użyteczna w celu wymuszenia, aby kolumny miały określony typ w trakcie tworzenia nowej tabeli za pomocą zapytania CREATE TABLE ... SELECT.

```
mysql> CREATE TABLE t SELECT CAST(20130101 AS DATE) AS date_val;
mysql> SHOW COLUMNS FROM t;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| date_val | date | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
```



```
mysql> SELECT * FROM t;
+-----+
| date_val |
+-----+
| 2013-01-01 |
+-----+
```

Funkcja `CONVERT()` jest podobna do `CAST()`, ale `CONVERT()` ma składnię ODBC, podczas gdy `CAST()` ma standardową składnię SQL.

■ `CONVERT(expr, typ)`

`CONVERT(expr USING charset)`

Pierwsza forma funkcji `CONVERT()` służy do takiego samego celu jak `CAST()`, ale ma nieco inną składnię. Znaczenie argumentów *expr* i *typ* pozostaje bez zmian. Natomiast druga forma (z `USING`) powoduje konwersję wartości na ciąg tekstowy o wskazanym kodowaniu znaków.

<code>CONVERT(304, BINARY)</code>	→ '304'
<code>CONVERT(-1, UNSIGNED)</code>	→ 18446744073709551615
<code>CONVERT('abc' USING utf8);</code>	→ 'abc'

C.2.3. Funkcje liczbowe

Wartością zwrótną funkcji liczbowej jest `NULL`, gdy przekazane zostaną argumenty spoza zakresu lub nieprawidłowe pod innym względem.

■ `ABS(x)`

Funkcja zwraca bezwzględną wartość *x*.

<code>ABS(13.5)</code>	→ 13.5
<code>ABS(-13.5)</code>	→ 13.5

■ `ACOS(x)`

Funkcja zwraca wartość arcus cosinus *x*. Jeżeli *x* nie jest z zakresu od -1 do 1 , wtedy wartością zwrótną jest `NULL`.

<code>ACOS(1)</code>	→ 0
<code>ACOS(0)</code>	→ 1.5707963267949
<code>ACOS(-1)</code>	→ 3.1415926535898

■ `ASIN(x)`

Funkcja zwraca wartość arcus sinus *x*. Jeżeli *x* nie jest z zakresu od -1 do 1 , wtedy wartością zwrótną jest `NULL`.

<code>ASIN(1)</code>	→ 1.5707963267949
<code>ASIN(0)</code>	→ 0
<code>ASIN(-1)</code>	→ -1.5707963267949

■ `ATAN(x)`

`ATAN(y, x)`

Ta jednoargumentowa forma funkcji `ATAN()` zwraca arcus tangens *x*. Dwuargumentowa forma jest synonimem `ATAN2()`.

<code>ATAN(1)</code>	→ 0.78539816339745
<code>ATAN(0)</code>	→ 0
<code>ATAN(-1)</code>	→ -0.78539816339745

■ **ATAN2(y, x)**

Ta funkcja jest podobna do ATAN(y, x), ale używa znaków w obu argumentach w celu ustalenia kwadrantu wartości zwrotnej.

ATAN2(1,1)	→ 0.78539816339745
ATAN2(1,-1)	→ 2.3561944901923
ATAN2(-1,1)	→ -0.78539816339745
ATAN2(-1,-1)	→ -2.3561944901923

■ **CEILING(x)**

CEIL(x)

Funkcja zwraca najmniejszą liczbę całkowitą nie mniejszą niż x . Jeżeli argument ma typ w postaci dokładnej wartości liczbowej, wartość zwrotna również będzie tego samego typu. W przeciwnym razie wartość zwrotna jest typu liczby zmiennoprzecinkowej (wartość przybliżona). Dotyczy to nawet sytuacji, w której wynik nie ma części ułamkowej.

CEILING(3.8)	→ 4
CEILING(-3.8)	→ -3

■ **COS(x)**

Funkcja zwraca wartość cosinus x , gdzie x jest podawane w radianach.

COS(0)	→ 1
COS(PI())	→ -1

■ **COT(x)**

Funkcja zwraca wartość cotangens x , gdzie x jest podawane w radianach.

COT(PI()/4)	→ 1
-------------	-----

■ **CRC32(str)**

Funkcja oblicza wartość CRC (ang. *Cyclic Redundancy Check*, cykliczny kod nadmiarowy) na podstawie argumentu traktowanego jako ciąg tekstowy. Wartością zwrótną jest 32-bitowa wartość bez znaku z zakresu od 0 do $2^{32}-1$. Jeżeli argumentem jest NULL, wartością zwrótną również będzie NULL.

CRC32('xyz')	→ 3951999591
CRC32('0')	→ 4108050209
CRC32(0)	→ 4108050209
CRC32(NULL)	→ NULL

■ **DEGREES(x)**

Funkcja zwraca wartość x skonwertowaną z radianów na stopnie.

DEGREES(PI())	→ 180
DEGREES(PI()*2)	→ 360
DEGREES(PI()/2)	→ 90
DEGREES(-PI())	→ -180

■ **EXP(x)**

Funkcja zwraca wartość e^x , gdzie e jest podstawą logarytmu naturalnego.

EXP(1)	→ 2.718281828459
EXP(2)	→ 7.3890560989307
EXP(-1)	→ 0.36787944117144
1/EXP(1)	→ 0.36787944117144

■ FLOOR(x)

Funkcja zwraca największą liczbę całkowitą nie większą niż x . Jeżeli argument ma typ w postaci dokładnej wartości liczbowej, wartość zwrotna również będzie tego samego typu. W przeciwnym razie wartość zwrotna jest typu liczby zmiennoprzecinkowej (wartość przybliżona). Dotyczy to nawet sytuacji, w której wynik nie ma części ułamkowej.

FLOOR(3.8)	→ 3
FLOOR(-3.8)	→ -4

■ LN(x)

To jest synonim funkcji LOG().

■ LOG(x)

LOG(b, x)

Jednoargumentowa forma funkcji LOG() zwraca logarytm naturalny (o podstawie e) dla wartości x .

LOG(0)	→ NULL
LOG(1)	→ 0
LOG(2)	→ 0.69314718055995
LOG(EXP(1))	→ 1

Dwuargumentowa forma funkcji LOG() zwraca logarytm o podstawie b dla wartości x .

LOG(10, 100)	→ 2
LOG(2, 256)	→ 8

Logarytm o podstawie b dla wartości x można obliczyć również za pomocą funkcji LOG(x)/LOG(b).

LOG(100)/LOG(10)	→ 2
LOG10(100)	→ 2

■ LOG10(x)

Funkcja zwraca logarytm o podstawie 10 dla wartości x .

LOG10(0)	→ NULL
LOG10(10)	→ 1
LOG10(100)	→ 2

■ LOG2(x)

Funkcja zwraca logarytm o podstawie 2 dla wartości x .

LOG2(0)	→ NULL
LOG2(255)	→ 7.9943534368589
LOG2(32767)	→ 14.99995597177

Funkcja LOG2() podaje wyrażoną w bitach „szerokość” wartości. Jednym z jej zastosowań jest sprawdzenie, jaka ilość pamięci masowej będzie wymagana do przechowywania danej wartości.

■ MOD(m, n)

Funkcja MOD() oblicza resztę z dzielenia. Składnia funkcji MOD(m, n) odpowiada składni operatora $m \% n$ lub $m \text{ MOD } n$ (patrz punkt C.1.3, zatytułowany „Operatory arytmetyczne”).

■ **PI()**

Funkcja zwraca wartość π .

PI() → 3.141593

■ **POW(x,y)**

POWER(x,y)

Funkcja zwraca wartość x , czyli x podniesione do potęgi y .

POW(2,3) → 8

POW(2,-3) → 0.125

POW(4,.5) → 2

POW(16,.25) → 2

■ **RADIANS(x)**

Funkcja zwraca wartość x skonwertowaną ze stopni na radiany.

RADIANS(0) → 0

RADIANS(360) → 6.2831853071796

RADIANS(-360) → -6.2831853071796

■ **RAND()**

RAND(n)

Funkcja RAND() zwraca losowo wybraną wartość zmiennoprzecinkową z zakresu od 0.0 do 1.0. Jeżeli podany zostanie argument n , to powinna być liczba całkowita. Jeżeli argument będzie stałą, funkcja RAND() użyje jej jako wartości dla generatora liczb losowych. Argumentu można używać, gdy trzeba wygenerować powtarzalną sekwencję liczb dla wartości z kolumny zbioru wynikowego.

RAND() → 0.1036697114852

RAND() → 0.5725383884949

RAND(10) → 0.65705152196535

RAND(10) → 0.65705152196535

Jeżeli argumentem nie będzie stała, wtedy ta wartość zostanie użyta w trakcie każdego wywołania funkcji RAND(). (Na przykład, jeśli podasz kolumnę, wówczas dla każdego rekordu wartość kolumny będzie przekazywana generatorowi liczb losowych dla danego rekordu).

Operacja generowania liczby losowej jest charakterystyczna dla klienta. Jeżeli jeden klient wywołuje funkcję RAND(n) w celu przekazania wartości generatorowi liczb losowych, to nie wpływa na liczby zwracane innym klientom.

Jeżeli funkcja RAND() zostanie użyta w klauzuli WHERE, będzie wywoływana w trakcie każdego wykonania klauzuli.

■ **ROUND(x)**

ROUND(x,d)

Funkcja ROUND() zwraca wartość x zaokrągloną do liczby z d miejscami po przecinku. Jeżeli argument d ma wartość 0 lub w ogóle został pominięty, wynik nie będzie miał części ułamkowej. Wartość zwrotna ma ten sam typ liczbowy jak pierwszy argument, więc jeśli pierwszy argument będzie liczbą całkowitą,

wynik nie ma części ułamkowej. Liczby podane w postaci ciągów tekstowych podlegają zwykłej konwersji na wartości o podwójnej precyzji i są obsługiwane w taki sposób.

ROUND(15.3)	→ 15
ROUND(15.5)	→ 16
ROUND(-33.27834,2)	→ -33.28
ROUND(1,4)	→ 1
ROUND('1',4)	→ 1.0000

Jeżeli argument d ma wartość ujemną, funkcja ROUND() powoduje usunięcie części ułamkowej, natomiast w wywołaniu ABS(d) cyfry do przecinka dziesiętnego stają się zerem.

ROUND(123456, -2)	→ 123500
-------------------	----------

Funkcja ROUND() obsługuje zaokrąglanie x w następujący sposób:

- ◆ W przypadku wartości przybliżonych sposób przeprowadzenia zaokrąglania zależy od używanej biblioteki matematycznej.
- ◆ Wartości dokładne o części ułamkowej .5 lub większej są zaokrąglane w górę. Wartości dokładne o części ułamkowej mniejszej niż .5 są zaokrąglane w dół. Na przykład, wartości 1.5 i -1.5 są zaokrąglane do odpowiednio 2 i -2, natomiast 1.49 i -1.49 do odpowiednio 1 i -1.

Więcej informacji na temat wartości dokładnych i przybliżonych znajdziesz w podpunkcie 3.1.1.1, zatytułowanym „Dokładne i przybliżone wartości liczbowe”.

■ SIGN(x)

Funkcja zwraca wartość -1, 0 lub 1 w zależności od tego, czy wartość x jest ujemna, równa zero czy dodatnia.

SIGN(15.803)	→ 1
SIGN(0)	→ 0
SIGN(-99)	→ -1

■ SIN(x)

Funkcja zwraca wartość sinus x , gdzie x jest podawane w radianach.

SIN(0)	→ 0
SIN(PI()/2)	→ 1

■ SQRT(x)

Funkcja zwraca nieujemny kwadrat wartości x .

SQRT(625)	→ 25
SQRT(2.25)	→ 1.5
SQRT(-1)	→ NULL

■ TAN(x)

Funkcja zwraca wartość tangens x , gdzie x jest podawane w radianach.

TAN(0)	→ 0
TAN(PI()/4)	→ 1

■ TRUNCATE(*x*,*d*)

Funkcja zwraca wartość *x*, w której część ułamkowa zostaje skrócona do *d* miejsc po przecinku. Jeżeli argument *d* ma wartość 0, wynik nie będzie miał części ułamkowej. Natomiast jeżeli argument *d* jest większy niż liczba miejsc dziesiętnych w wartości *x*, wtedy część ułamkowa zostanie po prawej stronie dopełniona zerami do żądanej długości.

TRUNCATE(1.23,1)	→ 1.2
TRUNCATE(1.23,0)	→ 1
TRUNCATE(1.23,4)	→ 1.2300

Jeżeli argument *d* ma wartość ujemną, funkcja TRUNCATE() powoduje usunięcie części ułamkowej i zer, natomiast w wywołaniu ABS(*d*) cyfry do przecinka dziesiętnego stają się zerem.

TRUNCATE(123456.789,-3)	→ 123000
-------------------------	----------

C.2.4. Funkcje ciągu tekstowego

Większość funkcji przedstawionych w tym punkcie zwraca wynik w postaci ciągu tekstowego. Niektóre z nich, na przykład LENGTH(), pobierają argumenty w postaci ciągów tekstowych, ale zwracają liczby. W przypadku funkcji operujących na ciągach tekstowych opartych na położeniu w ciągu tekstowym pozycja pierwszego (wysuniętego najbardziej na lewo) znaku to 1, a nie 0.

Części funkcji ciągu tekstowego, na przykład CHAR_LENGTH(), INSERT(), INSTR(), LCASE(), LEFT(), LOCATE(), LOWER(), LTRIM(), MID(), POSITION(), REPLACE(), REVERSE(), RIGHT(), RPAD(), RTRIM(), SUBSTRING(), SUBSTRING_INDEX(), TRIM(), UCASE() i UPPER(), można bezpiecznie używać z ciągami tekstowymi stosującymi wielobajtowe kodowanie znaków.

■ ASCII(*str*)

Funkcja zwraca wartość w postaci liczby całkowitej dla pierwszego z lewej bajta ciągu tekstowego *str*. Wspomniana wartość będzie z zakresu od 0 do 255. Jeśli ciąg tekstowy jest pusty, wartością zwrótną będzie zero, w przypadku NULL wartością zwrótną też jest NULL. Ciąg tekstowy *str* powinien zawierać jedynie 8-bitowe znaki.

ASCII('abc')	→ 97
ASCII('')	→ 0
ASCII(NULL)	→ NULL

■ BIN(*n*)

Funkcja zwraca ciąg tekstowy zawierający wyrażoną w systemie dwójkowym postać argumentu *n*. Poniższe wywołania są odpowiednikami, więcej informacji znajdziesz w opisie funkcji CONV().

BIN(65)	→ '1000001'
CONV(65,10,2)	→ '1000001'

■ CHAR(*n1*,*n2*,... [USING *charset*])

Funkcja interpretuje argumenty *n1*, *n2*, ... jako liczbowe kody znaków i zwraca ciąg tekstowy w kodowaniu znaków ustawionym przez połączenie. Wspomniany ciąg tekstowy zawiera połączenie wartości znaków odpowiadających zinterpretowanym kodom. Kody znaków o wartości większej niż 255 powodują wygenerowanie wyniku wielobajowego. Bez użycia słowa kluczowego USING wartością zwrótną jest binarny ciąg tekstowy. W przypadku użycia USING wartość zwrótna stosuje wskazane kodowanie znaków. Jeżeli wynik nie jest prawidłowy dla kodowania znaków, nastąpi wyświetlenie ostrzeżenia (w trybie ścisłego SQL wynikiem będzie NULL). Argumenty NULL są ignorowane.

CHAR(65)	→ 'A'
CHAR(97)	→ 'a'
CHAR(89,105,107,101,115,33)	→ 'Yikes!'

■ CHAR_LENGTH(*str*)

CHARACTER_LENGTH(*str*)

Te funkcje są podobne do LENGTH(), za wyjątkiem faktu, że argument długości jest wyrażany w znakach, nie w bajtach. (Każdy znak wielobajowy jest uznawany za znak o długości 1).

■ CHARSET(*str*)

Funkcja zwraca nazwę kodowania znaków w danym ciągu tekstowym.

Jeżeli argumentem jest NULL, wtedy wartością zwrótną również jest NULL.

CHARSET('abc')	→ 'latin1'
CHARSET(CONVERT('abc' USING utf8))	→ 'utf8'
CHARSET(123)	→ 'binary'

■ COALESCE(*expr1*,*expr2*,...)

Funkcja zwraca pierwszy element listy inny niż NULL. Jeżeli wszystkie argumenty mają wartość NULL, wtedy wartością zwrótną również jest NULL.

COALESCE(NULL,1/0,2,'a',45+97)	→ '2'
COALESCE(NULL,1/0)	→ NULL

■ COERCIBILITY(*str*)

Funkcja zwraca prawdopodobieństwo wymuszenia zmiany kolejności sortowania w ciągu tekstowym. Jeśli argument jest nieprawidłowy, wtedy wartością zwrótną będzie NULL. Funkcja określa prawdopodobieństwo zmiany kolejności sortowania w ciągu tekstowym w wyrażeniach wykorzystujących inne ciągi tekstowe. W tabeli C.3 wymieniono wartości zwrótnie funkcji, od najmniejszego do największego prawdopodobieństwa.

COERCIBILITY(_utf8 'abc' COLLATE utf8_bin)	→ 0
COERCIBILITY('abc')	→ 4

Tabela C.3. Prawdopodobieństwo zmiany kolejności sortowania w ciągu tekstowym

Prawdopodobieństwo zmiany	Opis
0	Kolejność sortowania została wyraźnie podana i nie może być zmieniona.
1	Brak podanej kolejności sortowania.
2	Kolejność sortowania można określić na podstawie innych informacji.
3	Kolejność sortowania wartości systemowych, na przykład <code>USER()</code> .
4	Kolejność sortowania może ulec zmianie.
5	Kolejność sortowania jest ignorowana (na przykład dla <code>NULL</code>).

■ `COLLATION(str)`

Funkcja zwraca nazwę kolejności sortowania w podanym ciągu tekstowym.

Jeśli argument jest nieprawidłowy, wtedy wartością zwrótną będzie `NULL`.

```
COLLATION(_latin2 'abc')           → 'latin2_general_ci'
COLLATION(CONVERT('abc' USING utf8) COLLATE utf8_bin)
                                     → 'utf8_bin'
```

■ `CONCAT(str1,str2,...)`

Funkcja zwraca ciąg tekstowy składający się z połączonych argumentów funkcji.

Jeśli którykolwiek argument ma wartość `NULL`, wtedy wartością zwrótną będzie `NULL`. Jeżeli choć jeden argument funkcji jest binarnym ciągiem tekstowym, wynik również będzie binarnym ciągiem tekstowym. Natomiast gdy wszystkie argumenty są niebinarnymi ciągami tekstowymi, wynikiem także jest niebinarny ciąg tekstowy. Argumenty liczbowe są konwertowane na postać ciągów tekstowych. Począwszy od MySQL 5.5.3, wspomniana konwersja generuje niebinarne ciągi tekstowe, natomiast we wcześniejszych wydaniach generowane są binarne ciągi tekstowe.

```
CONCAT('abc','def')                → 'abcdef'
CONCAT('abc')                      → 'abc'
CONCAT('abc',NULL)                 → NULL
CONCAT('Hello',' ','','goodbye')   → 'Hello, goodbye'
```

Innym sposobem łączenia ciągów tekstowych jest po prostu ich umieszczenie obok siebie:

```
'three' 'blind' 'mice'             → 'threeblindmice'
'abc' 'def' = 'abcdef'              → 1
```

■ `CONCAT_WS(delim,str1,str2,...)`

Funkcja działa podobnie do `CONCAT()`, ale zwraca ciąg tekstowy składający się z drugiego i kolejnych argumentów. Pierwszy argument (*delim*) jest separatorem stosowanym między ciągami tekstowymi. Jeżeli wartością *delim* jest `NULL`, wtedy funkcja zwraca `NULL`. Wartości `NULL` na liście ciągów tekstowych do połączenia są ignorowane.

```
CONCAT_WS(' ','a','b',' ','d')     → 'a,b,d'
CONCAT_WS('*-*','lemon','lime',NULL,'grape') → 'lemon*-*lime*-*grape'
```


■ CONV(*n*,*from_base*,*to_base*)

Mając podaną liczbę *n* o podstawie *from_base*, funkcja zwraca ciąg tekstowy przedstawiający liczbę *n*, ale o podstawie *to_base*. Jeśli którykolwiek argument ma wartość NULL, wtedy wartością zwrótną będzie NULL. Argumenty *from_base* i *to_base* powinny być liczbami całkowitymi z zakresu od 2 do 36. Z kolei *n* jest traktowane jako wartość BIGINT (64-bitowa liczba całkowita), ale może być podana w postaci ciągu tekstowego, ponieważ liczby o podstawach większych niż 10 mogą zawierać cyfry inne niż dziesiętne. (Z tego powodu wartością zwrótną funkcji CONV() jest ciąg tekstowy, wynik może zawierać znaki od A do Z dla podstaw od 11 do 36). Wartością zwrótną będzie 0, jeśli *n* nie jest prawidłową liczbą o podstawie *from_base*. Na przykład, jeśli *from_base* to 16, a *n* ma wartość 'abcdefg', wtedy wynikiem będzie 0, ponieważ g nie jest prawidłową cyfrą szesnastkową.

Znaki inne niż dziesiętne w liczbie *n* mogą być podane w postaci małych lub wielkich liter. Natomiast w wyniku znaki inne niż dziesiętne będą podane w postaci wielkich liter.

Konwersja liczby szesnastkowej 14 na postać liczby dwójkowej:

```
CONV('e',16,2) → '1110'
```

Konwersja liczby dwójkowej 255 na postać liczby ósemkowej:

```
CONV(11111111,2,8) → '377'
CONV('11111111',2,8) → '377'
```

Domyślnie liczba *n* jest traktowana jako liczba bez znaku. W przypadku podania *to_base* jako liczby ujemnej *n* będzie traktowane jako liczba ze znakiem.

```
CONV(-10,10,16) → 'FFFFFFFFFFFFFFF6'
CONV(-10,10,-16) → '-A'
```

■ EXPORT_SET(*n*,*on*,*off*[,*delim*[,*bit_count*]])

Funkcja zwraca ciąg tekstowy składający się z ciągów tekstowych *on* i *off* rozdzielonych separatorem *delim*. Domyślnym separatorem jest przecinek. Argument *on* jest używany do przedstawienia każdego ustawionego bitu wartości *n*, natomiast *off* oznacza każdy nieustawiony bit wartości *n*. Pierwszy z lewej ciąg tekstowy wyniku odpowiada dolnemu bitowi w wartości *n*. Argument *bit_count* określa maksymalną liczbę bitów wartości *n* do przeanalizowania. Wartością domyślną *bit_count* jest 64, która jednocześnie jest wartością maksymalną. Jeśli którykolwiek argument ma wartość NULL, wtedy wartością zwrótną będzie NULL.

```
EXPORT_SET(7,'+', '-', '',5) → '++++-'
EXPORT_SET(0xa,'1','0','',6) → '010100'
EXPORT_SET(97,'Y','N','',8) → 'Y,N,N,N,N,Y,Y,N'
```

■ FIND_IN_SET(*str*,*str_list*)

Argument *str_list* to ciąg tekstowy zawierający podciągi tekstowe rozdzielone przecinkami (to znaczy przypomina wartość SET). Wartością zwrótną funkcji FIND_IN_SET() jest indeks *str* na liście *str_list*. Jeżeli *str* nie istnieje na liście

str_list, funkcja zwraca 0. Natomiast jeśli którykolwiek argument ma wartość NULL, wtedy wartością zwrótną będzie NULL. Indeks pierwszego podciągu tekstowego ma wartość 1.

```
FIND_IN_SET('cow', 'moose,cow,pig')      → 2
FIND_IN_SET('dog', 'moose,cow,pig')      → 0
```

■ **FORMAT(*x*,*d*[,*locale*])**

Funkcja formatuje liczbę *x* na zawierającą *d* miejsc po przecinku dziesiętnym, używając formatu '*nn,nnn.nnn*', i zwraca wynik w postaci ciągu tekstowego. Jeżeli *d* wynosi 0, wtedy wynik nie zawiera przecinka dziesiętnego i części ułamkowej.

```
FORMAT(1234.56789,3)      → '1,234.568'
FORMAT(999999.99,2)      → '999,999.99'
FORMAT(999999.99,0)      → '1,000,000'
```

W ostatnim przykładzie zwróć uwagę na zaokrąglenie liczby.

Argument *locale* określa przecinek dziesiętny, separator tysięcy i grupowanie między separatorami. Wartością domyślną *locale* jest '*en_US*', co można zmienić, podając odpowiednią wartość argumentu. Dozwolone wartości argumentu *locale* są takie same jak dla zmiennej systemowej *lc_time_names*.

■ **FROM_BASE64(*str*)**

Funkcja konwertuje ciąg tekstowy zakodowany jako Base64 i zwraca jego wartość oryginalną lub NULL, jeśli argument nie jest ciągiem tekstowym zakodowanym jako Base64. Ta funkcja jest przeciwieństwem funkcji *TO_BASE64()*.

```
TO_BASE64('hello')      → 'aGVsbG8='
FROM_BASE64(TO_BASE64('hello'))      → 'hello'
```

Funkcja *FROM_BASE64()* została wprowadzona w MySQL 5.6.1.

■ **HEX(*n*)**

HEX(*str*)

Wywołana z argumentem liczbowym *n*, funkcja *HEX()* zwraca ciąg tekstowy przedstawiający szesnastkową postać argumentu. Poniższe wywołania są odpowiednikami; więcej informacji na ten temat znajdziesz w opisie funkcji *CONV()*:

```
HEX(65)      → '41'
CONV(65,10,16)      → '41'
```

Z kolei wywołana z argumentem tekstowym, funkcja *HEX()* zwraca ciąg tekstowy składający się z każdego znaku argumentu przedstawionego w postaci dwóch cyfr szesnastkowych. Ta forma funkcji *HEX()* jest odwrotnością funkcji *UNHEX()*.

```
HEX('255')      → '323535'
HEX('abc')      → '616263'
UNHEX(HEX('abc'))      → 'abc'
```

■ **INSERT(*str*,*pos*,*len*,*ins_str*)**

Funkcja zwraca ciąg tekstowy (*str*), w którym podciąg tekstowy rozpoczynający się w położeniu *pos* o długości *len* znaków został zastąpiony przez ciąg tekstowy *ins_str*. Jeżeli *pos* wykracza poza zakres, funkcja zwraca początkowy ciąg

tekstowy. Jeśli którykolwiek argument ma wartość NULL, wtedy wartością zwrótną będzie NULL.

```
INSERT('nighttime',6,4,'fall')      → 'nightfall'
INSERT('sunshine',1,3,'rain or ')   → 'rain or shine'
INSERT('sunshine',0,3,'rain or ')   → 'sunshine'
```

■ INSTR(*str*,*substr*)

Funkcja INSTR() przypomina dwuargumentową formę funkcji LOCATE(), ale z argumentami o zamienionej kolejności. Oba poniższe wywołania są odpowiednikami:

```
INSTR(str,substr)
LOCATE(substr,str)
```

■ LCASE(*str*)

Ta funkcja jest synonimem dla LOWER().

■ LEFT(*str*,*len*)

Funkcja zwraca pierwsze *len* znaków z lewej strony ciągu tekstowego *str* lub cały ciąg tekstowy, jeśli nie ma wystarczającej liczby znaków. Jeżeli *len* ma wartość NULL lub mniejszą niż 1, wtedy wartością zwrótną jest pusty ciąg tekstowy. Ponadto, jeśli którykolwiek argument ma wartość NULL, wtedy wartością zwrótną będzie NULL.

```
LEFT('my left foot',2)      → 'my'
LEFT(NULL,10)               → NULL
LEFT('abc',NULL)            → NULL
LEFT('abc',0)                → ''
```

■ LENGTH(*str*)

Funkcja zwraca wyrażoną w bajtach długość ciągu tekstowego *str*. Znaki wielobajtowe są uznawane za znaki o długości większej niż 1. Aby sprawdzić długość wyrażoną w znakach, należy użyć funkcji CHAR_LENGTH().

```
LENGTH('abc')               → 3
LENGTH(CONVERT('abc' USING ucs2)) → 6
LENGTH('')                   → 0
LENGTH(NULL)                 → NULL
```

■ LOCATE(*substr*,*str*)

LOCATE(*substr*,*str*,*pos*)

Dwuargumentowa postać funkcji LOCATE() zwraca położenie pierwszego wystąpienia ciągu tekstowego *substr* w ciągu tekstowym *str* lub 0, jeśli *substr* nie występuje w *str*. Jeśli którykolwiek argument ma wartość NULL, wtedy wartością zwrótną będzie NULL. W przypadku podania argumentu *pos* funkcja LOCATE() spróbuje wyszukać *substr* we wskazanym położeniu. Funkcja LOCATE() porównuje ciągi tekstowe jako binarne ciągi tekstowe, jeśli choć jeden z operandów jest binarnym ciągiem tekstowym. Natomiast jeśli operandy są niebinarnymi ciągami tekstowymi, to używana jest kolejność sortowania operandów.

```
LOCATE('b','abc')           → 2
LOCATE('b','ABC')           → 2
```

```
LOCATE(BINARY 'b','ABC')           → 0
LOCATE('b' COLLATE latin1_general_ci,'ABC') → 2
LOCATE('b' COLLATE latin1_general_cs,'ABC') → 0
```

■ LOWER(*str*)

Funkcja zwraca ciąg tekstowy *str*, w którym wszystkie znaki zostały skonwertowane na małe litery. Jeśli *str* ma wartość NULL, wtedy wartością zwrótną funkcji również jest NULL.

```
LOWER('New York, NY')           → 'new york, ny'
LOWER(NULL)                     → NULL
```

Konwersja liter jest przeprowadzana na podstawie kolejności sortowania w kodowaniu znaków argumentu. Jeśli argument jest binarnym ciągiem tekstowym, wtedy nie ma zdefiniowanego kodowania znaków i kolejności sortowania, a funkcja LOWER() zwraca niezmodyfikowany argument.

```
LOWER(BINARY 'New York, NY')     → 'New York, NY'
LOWER(0x414243)                 → 'ABC'
```

Aby rozwiązać ten problem, należy argument skonwertować lub rzutować na niebinarny ciąg tekstowy posiadający zdefiniowaną odpowiednią kolejność sortowania.

```
LOWER(CONVERT(BINARY 'New York, NY' USING latin1)) → 'new york, ny'
LOWER(_latin1 0x414243)           → 'abc'
```

■ LPAD(*str*,*len*,*pad_str*)

Funkcja zwraca ciąg tekstowy składający się z wartości ciągu tekstowego *str* dopełnionego po lewej stronie ciągiem tekstowym *pad_str* aż do długości *len* znaków. Jeśli którykolwiek argument ma wartość NULL, wtedy wartością zwrótną również będzie NULL.

```
LPAD('abc',12,'def')           → 'defdefdefabc'
LPAD('abc',10,'.')             → '.....abc'
```

Funkcja LPAD() skraca wynik o *len* znaków, jeśli *str* ma długość większą niż *len*.

```
LPAD('abc',2,'.')             → 'ab'
```

■ LTRIM(*str*)

Funkcja zwraca ciąg tekstowy *str*, w którym usunięte zostały spacje znajdujące się po lewej stronie (na początku). Jeśli *str* ma wartość NULL, wtedy wartością zwrótną również będzie NULL.

```
LTRIM(' abc ')                → 'abc '
```

■ MAKE_SET(*n*,*bit0_str*,*bit1_str*,...)

Funkcja tworzy wartość SET (ciąg tekstowy składający się z podciągów tekstowych rozdzielonych przecinkami) na podstawie wartości liczby całkowitej *n* i ciągów tekstowych *bit0_str*, *bit1_str*, Dla każdego bitu ustawionego w wartości *n* wynik będzie zawierał odpowiadający bitowi ciąg tekstowy. (Jeżeli ustawiony jest bit 0, wynik będzie zawierał *bit0_str* itd.). Jeżeli *n* wynosi 0, wtedy wynikiem jest pusty ciąg tekstowy. Jeżeli *n* wynosi NULL,

wartością zwrótną również jest NULL. Jeśli którykolwiek ciąg tekstowy na liście ma wartość NULL, zostanie zignorowany podczas tworzenia wynikowego ciągu tekstowego.

MAKE_SET(8, 'a', 'b', 'c', 'd', 'e')	→ 'd'
MAKE_SET(1 2 4, 'a', 'b', 'c', 'd', 'e')	→ 'a,b,c'
MAKE_SET(2+16, 'a', 'b', 'c', 'd', 'e')	→ 'b,e'
MAKE_SET(-1, 'a', 'b', 'c', 'd', 'e')	→ 'a,b,c,d,e'

Ostatni przykład powoduje pobranie wszystkich ciągów tekstowych, ponieważ wartość -1 ma ustawione wszystkie bity.

■ **MATCH(*col_list*) AGAINST(*str* [*search_mode*])**

Funkcja MATCH() przeprowadza operację wyszukiwania z użyciem indeksu typu FULLTEXT. Lista w wywołaniu funkcji MATCH() składa się z jednej lub więcej nazw kolumn rozdzielonych przecinkami. To muszą być kolumny tworzące indeks typu FULLTEXT w przeszukiwanej tabeli. Argument *str* funkcji AGAINST() wskazuje słowo lub słowa do wyszukania w wymienionych kolumnach. Słowa to sekwencje znaków składających się z liter, cyfr, apostrofów i znaku podkreślenia. Nawias jest opcjonalny w przypadku MATCH(), ale obowiązkowy dla AGAINST().

Domyślnie wyszukiwanie jest przeprowadzane w trybie języka naturalnego.

Argument *search_mode* może przyjąć jedną z wymienionych poniżej wartości:

- ◆ IN NATURAL LANGUAGE MODE
- ◆ IN BOOLEAN MODE
- ◆ [IN NATURAL LANGUAGE MODE] WITH QUERY EXPANSION

W przypadku wyszukiwania w języku naturalnym funkcja MATCH() powoduje utworzenie rankingu trafności dla każdego rekordu. Ranking to nieujemne liczby zmiennoprzecinkowe, gdzie zero oznacza brak szukanego słowa.

Wartości dodatnie oznaczają znalezienie przynajmniej jednego wystąpienia szukanego słowa. Słowa występujące w więcej niż połowie rekordów tabeli są ignorowane (uznawane za słowa o trafności zerowej, ponieważ występują zbyt często). Ponadto, MySQL posiada wewnętrzną listę słów, które nigdy nie będą uznawane za trafne (na przykład „the” i „but”).

Jeżeli trybem wyszukiwania będzie IN BOOLEAN MODE, wyniki wyszukiwania będą w całości oparte na istnieniu lub braku szukanego słowa bądź słów, niezależnie od częstotliwości ich występowania w tabeli. W przypadku tego rodzaju wyszukiwania słowa znajdujące się w szukanym ciągu tekstowym mogą być modyfikowane za pomocą poniższych operatorów i tym samym zmieniać sposób przeprowadzania operacji wyszukiwania:

- ◆ Znak + lub – umieszczony na początku wskazuje odpowiednio, że słowo musi lub nie może być obecne.
- ◆ Znak < lub > umieszczony na początku odpowiednio zmniejsza i zwiększa udział słowa w obliczaniu wartości trafności.

- ◆ Znak `~` umieszczony na początku wyklucza słowo z udziału w obliczeniu wartości trafności, ale nie wyklucza całkowicie zawierającego go rekordu, jak ma to miejsce w przypadku znaku `-`.
- ◆ Znak `*` na końcu działu w charakterze znaku wieloznacznego. Na przykład, `akt*` powoduje dopasowanie słów `akt`, `akty`, `aktówka` itd.
- ◆ Wyszukiwanie frazy może się odbywać przez jej ujęcie w cudzysłów ("`fraza`"). Aby taka fraza została dopasowana, wszystkie słowa frazy muszą istnieć w kolejności podanej we frazie.
- ◆ Nawiasy pozwalają na grupowanie słów w wyrażenia. Wyrażenia umieszczone w nawiasach można zagnieżdżać.

Słowa bez modyfikatorów są traktowane jako opcjonalne w wyszukiwaniu boolowskim, podobnie jak w wyszukiwaniu naturalnym.

W przypadku braku indeksu typu `FULLTEXT` istnieje możliwość przeprowadzenia wyszukiwania w trybie boolowskim, ale taka operacja może być bardzo wolna.

Jeżeli trybem wyszukiwania będzie `WITH QUERY EXPANSION` lub `IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION`, wyszukiwanie w języku naturalnym zostanie przeprowadzone jednokrotnie z użyciem szukanego ciągu tekstowego, a następnie ponownie z użyciem szukanego ciągu tekstowego i informacji z pierwszych kilku najtrafniejszych dopasowań początkowej operacji wyszukiwania. To pozwala na znalezienie rekordów zawierających treść powiązaną z początkowym szukanym ciągiem tekstowym.

Więcej informacji na temat wyszukiwania pełnego tekstu znajdziesz w podrozdziale 2.14, zatytułowanym „Wyszukiwanie pełnego tekstu”.

■ `MID(str,pos,len)`

`MID(str,pos)`

To trójargumentowa forma funkcji `MID()`, zwraca znajdujący się w ciągu tekstowym `str` podciąg, który rozpoczyna się w położeniu `pos` i ma `len` znaków długości. Dwuargumentowa postać funkcji zwraca podciąg tekstowy rozpoczynający się w położeniu `pos` i rozciągający się aż do końca ciągu tekstowego. Jeśli którykolwiek argument ma wartość `NULL`, wtedy wartością zwrótną będzie `NULL`.

```
MID('what a dull example',8,4)      → 'dull'
MID('what a dull example',8)        → 'dull example'
```

Funkcja `MID()` jest w rzeczywistości synonimem dla funkcji `SUBSTRING()` i może być używana w dowolnych formach składni dozwolonej przez `SUBSTRING()`.

■ `OCT(n)`

Funkcja zwraca ciąg tekstowy zawierający wyrażoną w systemie ósemkowym postać argumentu `n`. Poniższe wywołania są odpowiednikami; więcej informacji znajdziesz w opisie funkcji `CONV()`.

```
OCT(65)          → '101'
CONV(65,10,8)    → '101'
```

■ `OCTET_LENGTH(str)`

To jest synonim funkcji `LENGTH()`.

■ `ORD(str)`

Funkcja zwraca zwykłą wartość pierwszego znaku ciągu tekstowego `str` lub `NULL`, jeśli ciąg tekstowy `str` ma wartość `NULL`. Jeżeli pierwszy znak jest znakiem jednobajtowym, działanie funkcji `ORD()` jest takie samo jak funkcji `ASCII()`.

```
ORD('abc')           → 97
ASCII('abc')          → 97
```

W przypadku znaku wielobajtowego funkcja `ORD()` zwraca wartość określoną na podstawie liczbowych wartości poszczególnych bajtów znaku, począwszy od `b1` do `bn` (od prawej do lewej strony):

$$b1 + (b2 \times 256) + (b3 \times 256 \times 256) + \dots$$

■ `POSITION(substr IN str)`

To jest dwuargumentowa forma funkcji `LOCATE()`. Poniższe wywołania są odpowiednikami:

```
POSITION(substr IN str)
LOCATE(substr, str)
```

■ `QUOTE(str)`

Funkcja przetwarza argument i zwraca ciąg tekstowy, który jest prawidłowo cytowany do użycia w zapytaniu SQL. To jest użyteczna funkcja podczas tworzenia zapytań, których wynikiem są inne zapytania. W przypadku wartości innych niż `NULL` w wartości zwróconej każdy znak apostrofu, ukośnika, `Control+Z` i `NUL` (bajt o wartości zero) jest poprzedzony ukośnikiem, a całość ujęta w cudzysłów. Jeżeli wartością `str` jest `NULL`, wartością zwróconą będzie słowo `NULL` nieujęte w cudzysłów.

```
QUOTE("Let's go!")    → 'Let\'s go!'
QUOTE(NULL)           → NULL
```

■ `REPEAT(str, n)`

Funkcja zwraca ciąg tekstowy składający się z `n` powtórzeń ciągu tekstowego `str`. Jeśli wartość `n` nie jest dodatnia, zwrócony będzie pusty ciąg tekstowy. Natomiast jeśli którykolwiek argument ma wartość `NULL`, wtedy wartością zwróconą będzie `NULL`.

```
REPEAT('x', 10)        → 'xxxxxxxxxx'
REPEAT('abc', 3)       → 'abcbcbcb'
```

■ `REPLACE(str, from_str, to_str)`

Funkcja zwraca ciąg tekstowy składający się z ciągu tekstowego `str`, w którym wszystkie wystąpienia `from_str` zostały zastąpione przez `to_str`. Jeżeli `to_str` jest pustym ciągiem tekstowym, efektem będzie usunięcie wystąpień `from_str`. Natomiast jeżeli `from_str` jest pustym ciągiem tekstowym, funkcja `REPLACE()` zwraca niezmodyfikowany ciąg tekstowy `str`. Jeśli którykolwiek argument ma wartość `NULL`, wtedy wartością zwróconą będzie `NULL`.

```

REPLACE('abracadabra', 'a', 'oh')      → 'ohbrohcohdohbroh'
REPLACE('abracadabra', 'a', '')        → 'brcdbr'
REPLACE('abracadabra', '', 'x')        → 'abracadabra'

```

■ REVERSE(*str*)

Funkcja zwraca ciąg tekstowy składający się z ciągu tekstowego *str*, w którym odwrócono kolejność wszystkich znaków. Jeśli *str* ma wartość NULL, wtedy wartością zwrótną również będzie NULL.

```

REVERSE('abracadabra')                → 'arbadacarba'
REVERSE('tararA ta tar a rat')        → 'Tar a rat at Ararat'

```

■ RIGHT(*str*, *len*)

Funkcja zwraca pierwsze *len* znaków z prawej strony ciągu tekstowego *str* lub cały ciąg tekstowy, jeśli nie ma wystarczającej liczby znaków. Jeżeli *len* ma wartość NULL lub mniejszą niż 1, wtedy wartością zwrótną jest pusty ciąg tekstowy. Ponadto, jeśli którykolwiek argument ma wartość NULL, wtedy wartością zwrótną będzie NULL.

```

RIGHT('rightmost', 4)                  → 'most'

```

■ RPAD(*str*, *len*, *pad_str*)

Funkcja zwraca ciąg tekstowy składający się z wartości ciągu tekstowego *str* dopełnionego po prawej stronie ciągiem tekstowym *pad_str* aż do długości *len* znaków. Jeśli którykolwiek argument ma wartość NULL, wtedy wartością zwrótną również będzie NULL.

```

RPAD('abc', 12, 'def')                  → 'abcdefdefdef'
RPAD('abc', 10, '.')                    → 'abc.....'

```

Funkcja RPAD() skracza wynik o *len* znaków, jeśli *str* ma długość większą niż *len*.

```

RPAD('abc', 2, '.')                     → 'ab'

```

■ RTRIM(*str*)

Funkcja zwraca ciąg tekstowy *str*, w którym usunięte zostały spacje znajdujące się po prawej stronie (na końcu). Jeśli *str* ma wartość NULL, wtedy wartością zwrótną również będzie NULL.

```

RTRIM(' abc ')                          → ' abc'

```

■ SOUNDEX(*str*)

expr1 SOUNDS LIKE *expr2*

Funkcja SOUNDEX() zwraca ciąg tekstowy wygenerowany na podstawie ciągu tekstowego *str* lub wartość NULL, jeśli ciąg tekstowy *str* ma wartość NULL. Znaki inne niż alfanumeryczne w *str* są ignorowane. Z kolei międzynarodowe znaki inne niż alfanumeryczne spoza zakresu od A do Z są traktowane jak samogłoski. Działanie funkcji SOUNDEX() może nie przynieść dobrych wyników dla znaków wielobajtowych oraz dla języków innych niż angielski.

```

SOUNDEX('Cow')                         → 'C000'
SOUNDEX('Cow1')                        → 'C400'
SOUNDEX('How1')                        → 'H400'
SOUNDEX('Hello')                       → 'H400'

```

Operator SOUNDS LIKE jest odpowiednikiem funkcji SOUNDEX().

■ SPACE(*n*)

Funkcja zwraca ciąg tekstowy składający się z *n* spacji lub pustego ciągu tekstowego, jeśli wartość *n* jest niedodatnia. Jeśli *n* ma wartość NULL, wartością zwrótną również będzie NULL.

SPACE(6)	→ ' '
SPACE(0)	→ ''
SPACE(NULL)	→ NULL

■ SUBSTR(*arguments*)

Funkcja SUBSTR() jest synonimem funkcji SUBSTRING(). Zastosowanie mają te same formaty argumentów.

■ SUBSTRING(*str*,*pos*)

SUBSTRING(*str*,*pos*,*len*)
 SUBSTRING(*str* FROM *pos*)
 SUBSTRING(*str* FROM *pos* FOR *len*)

Funkcja zwraca z ciągu tekstowego *str* podciąg tekstowy rozpoczynający się w położeniu *pos*. Jeśli którykolwiek argument ma wartość NULL, wtedy wartością zwrótną będzie NULL. W przypadku podania argumentu *len* zwrócony podciąg tekstowy będzie miał wskazaną liczbę znaków. W przeciwnym razie zostanie zwrócona cała prawa część ciągu tekstowego *str*, począwszy od położenia *pos*.

SUBSTRING('abcdef',3)	→ 'cdef'
SUBSTRING('abcdef',3,2)	→ 'cd'

Wymienione poniżej wywołania są odpowiednikami:

SUBSTRING(*str*,*pos*,*len*)
 SUBSTRING(*str* FROM *pos* FOR *len*)
 MID(*str*,*pos*,*len*)

■ SUBSTRING_INDEX(*str*,*delim*,*n*)

Funkcja zwraca podciąg tekstowy z ciągu tekstowego *str*. Jeżeli wartość *n* jest dodatnia, funkcja SUBSTRING_INDEX() wyszukuje *n*-te wystąpienie ciągu tekstowego ogranicznika (*delim*), a następnie zwraca wszystko, co znajduje się po lewej stronie wspomnianego ogranicznika. Jeżeli wartość *n* jest ujemna, funkcja SUBSTRING_INDEX() wyszukuje *n*-te wystąpienie ciągu tekstowego ogranicznika (*delim*), licząc od końca ciągu tekstowego *str*, a następnie zwraca wszystko, co znajduje się po prawej stronie wspomnianego ogranicznika. Jeżeli funkcja SUBSTRING_INDEX() nie znajdzie *delim* w *str*, zwrócony zostanie cały ciąg tekstowy. Jeśli którykolwiek argument ma wartość NULL, wtedy wartością zwrótną również będzie NULL.

SUBSTRING_INDEX('jar-jar','j',-2)	→ 'ar-jar'
SUBSTRING_INDEX('sampadm@localhost','@',1)	→ 'sampadm'
SUBSTRING_INDEX('sampadm@localhost','@',-1)	→ 'localhost'

■ TO_BASE64(*str*)

Funkcja konwertuje argument na postać ciągu tekstowego zakodowanego jako Base64. Wartością zwrótną będzie NULL, jeśli argumentem jest NULL. Ta funkcja jest przeciwieństwem funkcji FROM_BASE64().

```

TO_BASE64('abc')           → 'YWJj'
TO_BASE64('123')           → 'MTIz'
TO_BASE64(123)             → 'MTIz'

```

Funkcja `TO_BASE64()` została wprowadzona w MySQL 5.6.1.

■ `TRIM([trim_str FROM] str)`

```
TRIM([{LEADING | TRAILING | BOTH} [trim_str] FROM] str)
```

Pierwsza forma powoduje zwrot ciągu tekstowego *str*, w którym na początku i na końcu usunięto ciąg tekstowy *trim_str*. W drugiej formie użycie `LEADING` powoduje, że funkcja `TRIM()` usunie wystąpienia *trim_str* na początku ciągu tekstowego. Z kolei użycie `TRAILING` powoduje, że funkcja `TRIM()` usuwa wystąpienia *trim_str* na końcu ciągu tekstowego. Natomiast `BOTH` powoduje usunięcie wystąpień *trim_str* na początku i na końcu ciągu tekstowego.

W przypadku braku `LEADING`, `TRAILING` lub `BOTH` domyślnie używana jest opcja `BOTH`. Jeśli nie będzie podany ciąg tekstowy *trim_str*, wtedy usuwane są spacje.

```

TRIM('^' FROM '^xyz^')      → 'xyz'
TRIM(LEADING '^' FROM '^xyz^') → 'xyz^'
TRIM(TRAILING '^' FROM '^xyz^') → '^xyz'
TRIM(BOTH '^' FROM '^xyz^')   → 'xyz'
TRIM(BOTH FROM '  abc  ')     → 'abc'
TRIM('  abc  ')               → 'abc'

```

■ `UCASE(str)`

Ta funkcja jest synonimem funkcji `UPPER()`.

■ `UNHEX(expr)`

Argument funkcji jest interpretowany jako ciąg tekstowy zawierający parę cyfr szesnastkowych. Każda para cyfr będzie skonwertowana na znak, a wartością zwrótną funkcji jest binarny ciąg tekstowy składający się z wspomnianych znaków. Ta funkcja jest odwrotnością funkcji `HEX()`.

```

UNHEX('414243')           → 'ABC'
HEX(UNHEX('414243'))       → '414243'
UNHEX(HEX('ABC'))          → 'ABC'
UNHEX(414243)              → 'ABC'
CHARSET(UNHEX('414243'))    → 'binary'

```

■ `UPPER(str)`

Funkcja zwraca ciąg tekstowy *str*, w którym wszystkie znaki zostały skonwertowane na wielkie litery. Jeśli *str* ma wartość `NULL`, wtedy wartością zwrótną funkcji również jest `NULL`.

```

UPPER('New York, NY')      → 'NEW YORK, NY'
UPPER(NULL)                 → NULL

```

Zapoznaj się z opisem funkcji `LOWER()`, aby dowiedzieć się więcej na temat konwersji binarnych ciągów tekstowych.

■ `WEIGHT_STRING(str [AS type(n)] [LEVEL levels] [flags])`

Funkcja zwraca wagę ciągu tekstowego *str* jako binarnego ciągu tekstowego.

Wspomniana waga wskazuje, jak ciąg tekstowy *str* będzie obsługiwany w trakcie

operacji porównywania i sortowania. Dwa ciągi tekstowe o takiej samej wadze są uznawane za identyczne, w przeciwnym razie mają taką samą względną kolejność jak ich wagi. Opcja AS powoduje, że ciąg tekstowy *str* otrzyma dany typ i długość, natomiast opcja LEVEL określa zwrócony poziom kolejności sortowania. Obecnie nie zostały jeszcze zaimplementowane żadne wartości *flags*. Jeżeli *str* jest binarnym ciągiem tekstowym, jego waga będzie taka sama jak *str*. Jeśli *str* jest niebinarnym ciągiem tekstowym i ma zdefiniowaną kolejność sortowania, wynikowy ciąg tekstowy zawiera wagę kolejności sortowania. Gdy *str* ma wartość NULL, wynikiem również jest NULL. Poniższe przykłady używają funkcji HEX() w celu przedstawienia ciągów tekstowych wagi w postaci możliwej do wyświetlenia:

```
HEX(WEIGHT_STRING(BINARY 'Hello'))      → '48656C6C6F'
HEX(WEIGHT_STRING('Hello'))              → '48454C4C4F'
HEX(WEIGHT_STRING(_utf8'Hello'))          → '00480045004C004C004F'
```

Za pomocą klauzuli AS ciąg tekstowy *str* można rzutować na dany typ i długość. Aby *str* traktować jako ciąg tekstowy typu CHAR o długości *n* znaków, należy użyć AS CHAR(*n*). Na końcu ciąg tekstowy będzie dopełniony spacjami, jeśli zajdzie potrzeba. Klauzula AS BINARY(*n*) traktuje ciąg tekstowy jako binarny ciąg tekstowy o *n* bajtach długości, a dopełnieniem są bajty 0x00. Wartość *n* musi być równa 1 lub większa. Jeżeli długość ciągu tekstowego *str* będzie większa niż *n*, zostanie on skrócony.

Kolejność sortowania może mieć poziomy. Domyślnie wynik zawiera wagę dla wszystkich poziomów. Aby zwrócić wagę jedynie dla określonego poziomu, należy użyć klauzuli LEVEL. Wartość *levels* może być listą jednej lub więcej liczb całkowitych rozdzielonych przecinkami lub zakresem (dwie liczby całkowite rozdzielone myślnikiem).

Poziomy na liście muszą być podane w kolejności rosnącej. Drugi poziom w zakresie jest traktowany jako pierwszy, jeśli będzie mniejszy niż pierwszy. Jeśli wykroczą poza zakres, poszczególne wartości poziomów będą przycinane, aby mieściły się w zakresie od 1 do maksymalnego poziomu dla kolejności sortowania.

Poziomy wartości na liście mogą mieć dodany modyfikator: ASC powoduje zwrot niezmodyfikowanych wag (wartość domyślna), DESC zwraca wagi, w których wartości bitów zostały odwrócone, natomiast REVERSE zwraca wagi dla odwróconej wartości *str*.

```
HEX(WEIGHT_STRING('abc' LEVEL 1 ASC))      → '414243'
HEX(WEIGHT_STRING('abc' LEVEL 1 DESC))      → 'BEBDBC'
HEX(WEIGHT_STRING('abc' LEVEL 1 REVERSE))    → '434241'
```

Funkcja WEIGHT_STRING() została wprowadzona w MySQL 5.6.0.

C.2.5. Funkcje daty i godziny

Funkcje daty i godziny pobierają różne typy argumentów. Ogólnie rzecz biorąc, funkcja oczekująca argumentu typu DATE będzie akceptowała również argument typu DATETIME lub TIMESTAMP i po prostu zignoruje część poświęconą godzinie w danej wartości. Pewne funkcje oczekujące wartości TIME akceptują argumenty typu DATETIME lub TIMESTAMP i ignorują część dotyczącą daty.

Wiele funkcji przedstawionych w tym punkcie ma możliwość interpretacji argumentów liczbowych jako wartości daty i godziny.

```
MONTH('1906-04-18')      → 4
MONTH(19060418)          → 4
```

Podobnie, w przypadku wielu funkcji zwracających wartość daty i godziny, MySQL konwertuje wartość zwrótną na postać ciągu tekstowego lub liczby, w zależności od kontekstu.

```
CURDATE()                → '2012-08-26'
CONCAT('Today is ', CURDATE()) → 'Today is 2012-08-26'
CURDATE() + 0             → 20120826
```

Kiedy przeprowadzana jest konwersja wartości daty i godziny lub godziny na postać liczbową, wartości liczbowe mają część wyrażającą mikrosekundy (.000000). Aby się pozbyć wymienionej części, wynik należy rzutować na postać liczby całkowitej.

```
NOW()+0                  → 20120806163317.000000
CAST(NOW() AS UNSIGNED)  → 20120806163317
CURTIME()+0              → 163317.000000
CAST(CURTIME() AS UNSIGNED) → 163317
```

Kilka funkcji wyodrębniających komponenty daty zwraca „niekompletne” daty. Na przykład, funkcje MONTH() i DAYOFMONTH() zwracają wartość 0 dla argumentu '2013-00-00'. To samo dotyczy specyfikatorów formatu daty używanych w funkcji DATE_FORMAT().

Jeżeli funkcji daty i godziny nie zostanie podana prawidłowa wartość daty lub godziny, wtedy wynik może być bezsensowny. W pierwszej kolejności należy sprawdzić argumenty.

■ ADDDATE(*date*, INTERVAL *expr interval*)

ADDDATE(*date*, *expr*)

W przypadku pierwszej składni funkcja ADDDATE() pobiera wartość daty lub daty i godziny (*date*), dodaje wskazany odstęp czasu (*interval*) i zwraca wynik. To jest synonim funkcji DATE_ADD().

```
ADDDATE('2004-12-01', INTERVAL 1 YEAR)      → '2005-12-01'
```

W przypadku drugiej składni funkcja ADDDATE() pobiera wartość daty lub daty i godziny (*date*), dodaje wartość wskazującą liczbę dni i zwraca wynik.

```
ADDDATE('2004-12-01', 365)                  → '2005-12-01'
```

Druga składnia może być przepisana na postać stosującą pierwszą składnię:

```
ADDDATE( date, expr ) = ADDDATE( date, INTERVAL expr DAY)
```

■ ADDTIME(*expr1*,*expr2*)

Funkcja dodaje dwa wyrażenia i zwraca wynik. Wyrażenie *expr1* powinno być wartością godziny lub daty i godziny, natomiast wyrażenie *expr2* powinno być wartością godziny.

```
ADDTIME('06:30:00.5','12:30:00.4')      → '19:00:00.900000'
ADDTIME('2004-01-01 00:00:00','12:30:00') → '2004-01-01 12:30:00'
```

■ CONVERT_TZ(*date*,*from_zone*,*to_zone*)

Mając podaną wartość daty lub daty i godziny (*date*), funkcja CONVERT_TZ() traktuje ją jako wartość w strefie czasowej *from_zone* i konwertuje na wartość w strefie czasowej *to_zone*, a następnie zwraca wynik. Jeżeli którykolwiek argument jest nieprawidłowy, wartością zwrótną funkcji będzie NULL. Strefy czasowe są takie, jak przedstawiono w punkcie 12.6.1, zatytułowanym „Konfiguracja obsługi stref czasowych”. Aby funkcja CONVERT_TZ() działała prawidłowo, wartość zwrótna musi znajdować się w zakresie typu danych TIMESTAMP.

```
CONVERT_TZ('2019-02-11 00:00:00','US/Central','US/Eastern')
                                                    → '2019-02-11 01:00:00'
CONVERT_TZ('2019-02-11','+00:00','-03:00')      → '2019-02-10 21:00:00'
```

■ CURDATE()

Funkcja zwraca bieżącą datę w strefie czasowej sesji. Wartość zwrótna jest typu DATE w formacie 'CCYY-MM-DD'.

```
CURDATE()      → '2012-08-26'
```

■ CURRENT_DATE()

To jest synonim funkcji CURDATE(). Nawiasy są opcjonalne.

■ CURRENT_TIME([*fsp*])

To jest synonim funkcji CURTIME(). Nawiasy są opcjonalne, o ile nie zostanie podany argument *fsp*.

■ CURRENT_TIMESTAMP([*fsp*])

To jest synonim funkcji NOW(). Nawiasy są opcjonalne, o ile nie zostanie podany argument *fsp*.

■ CURTIME([*fsp*])

Funkcja zwraca bieżącą godzinę dnia w strefie czasowej sesji. Wartość zwrótna jest typu TIME w formacie 'hh:mm:ss'. Począwszy od MySQL 5.6.4, można podać opcjonalny argument *fsp*. W takim przypadku wartość zwrótna będzie zawierała część ułamkową o precyzji od 0 do 6 cyfr. Przed wersją MySQL 5.6.4 wszelkie argumenty funkcji są ignorowane.

```
CURTIME()      → '15:01:02'
CURTIME(2)     → '15:01:02.20'
```

■ DATE(*expr*)

Funkcja zwraca komponent daty w wyrażeniu *expr*, które powinno być wyrażeniem przedstawiającym datę lub datę i godzinę.

```
DATE('2008-03-12')      → '2008-03-12'
DATE('2008-03-12 16:15:00') → '2008-03-12'
```

■ `DATE_ADD(date, INTERVAL expr interval)`

Funkcja pobiera wartość daty lub daty i godziny (*date*), dodaje wskazany odstęp czasu (*interval*), a następnie zwraca wynik. Wyrażenie *expr* wskazuje wartość czasu, jaką należy dodać do *date* (lub odjąć, jeśli wyrażenie *expr* jest ujemne). Z kolei *interval* określa sposób interpretacji odstępu czasu. Wynikiem jest wartość DATE, jeśli argument *date* jest wartością typu DATE i wygenerowanie wyniku nie wymaga żadnych obliczeń związanych z godziną. W przeciwnym razie wynikiem jest wartość typu DATETIME. Jeżeli argument *date* nie jest prawidłową datą, wtedy wartością zwrótną funkcji będzie NULL.

```
DATE_ADD('2009-12-01', INTERVAL 1 YEAR)      → '2010-12-01'
DATE_ADD('2009-12-01', INTERVAL 60 DAY)     → '2010-01-30'
DATE_ADD('2009-12-01', INTERVAL -3 MONTH)   → '2009-09-01'
DATE_ADD('2009-12-01 08:30:00', INTERVAL 12 HOUR) → '2009-12-01 20:30:00'
```

W tabeli C.4 wymieniono dozwolone wartości argumentu *interval*, ich znaczenie oraz format, w którym należy podawać wartości dla poszczególnych typów odstępu czasu. W przypadku słowa kluczowego INTERVAL i specyfikatora *interval* wielkość liter nie ma znaczenia.

Tabela C.4. Dozwolone wartości argumentu interval

Jednostka wyrażenia odstępu czasu	Opis	Format wartości
MICROSECOND	Mikrosekundy	<i>uuuuuu</i>
SECOND	Sekundy	<i>ss</i>
SECOND_MICROSECOND	Sekundy i mikrosekundy	' <i>ss .uuuuuu</i> '
MINUTE	Minuty	<i>mm</i>
MINUTE_SECOND	Minuty i sekundy	' <i>mm:ss</i> '
MINUTE_MICROSECOND	Minuty, sekundy i mikrosekundy	' <i>mm:ss .uuuuuu</i> '
hour	Godziny	<i>hh</i>
hour_minute	Godziny i minuty	' <i>hh:mm</i> '
hour_second	Godziny, minuty i sekundy	' <i>hh:mm:ss</i> '
hour_microsecond	Godziny, minuty, sekundy i mikrosekundy	' <i>hh:mm:ss .uuuuuu</i> '
DAY	Dni	<i>DD</i>
day_hour	Dni i godziny	' <i>DD hh</i> '
day_minute	Dni, godziny i minuty	' <i>DD hh:mm</i> '
day_second	Dni, godziny, minuty i sekundy	' <i>DD hh:mm:ss</i> '
day_microsecond	Dni, godziny, minuty, sekundy i mikrosekundy	' <i>DD hh:mm:ss .uuuuuu</i> '
WEEK	Tygodnie	<i>WW</i>

Tabela C.4. Dozwolone wartości argumentu `interval` (ciąg dalszy)

Jednostka wyrażenia odstępu czasu	Opis	Format wartości
MONTH	Miesiące	<i>MM</i>
QUARTER	Kwartały	<i>QQ</i>
YEAR	Lata	<i>YY</i>
YEAR_MONTH	Lata i miesiące	<i>'YY-MM'</i>

Wyrażenie *expr* dodawane do daty może być podane jako liczba lub ciąg tekstowy, o ile nie zawiera znaków innych niż cyfry, ponieważ wówczas musi być ciągiem tekstowym. Ogranicznikiem znaków może być dowolny znak przestankowy.

```
DATE_ADD('2005-12-01', INTERVAL '2:3' YEAR_MONTH) → '2008-03-01'
DATE_ADD('2005-12-01', INTERVAL '2-3' YEAR_MONTH) → '2008-03-01'
```

Elementy wartości wyrażenia *expr* są dopasowywane od prawej do lewej strony względem elementów oczekiwanych na podstawie specyfikatora *interval*.

Na przykład, oczekiwanym formatem dla `HOUR_SECOND` jest *'hh:mm:ss'*.

Wartość *'15:21'* wyrażenia *expr* jest interpretowana jako *'00:15:21'*, a nie jako *'15:21:00'*.

```
DATE_ADD('2003-12-01 12:00:00', INTERVAL '15:21' HOUR_SECOND)
→ '2003-12-01 12:15:21'
```

Jeżeli argument *interval* będzie miał wartość `YEAR`, `MONTH` lub `YEAR_MONTH` i element daty wyniku będzie większy niż liczba dni w wynikowym miesiącu, dniem będzie ostatni dzień danego miesiąca.

```
DATE_ADD('2003-12-31', INTERVAL 2 MONTH) → '2004-02-29'
```

Podczas dodawania daty można użyć alternatywnej składni:

```
'2003-12-31' + INTERVAL 2 MONTH → '2004-02-29'
INTERVAL 2 MONTH + '2003-12-31' → '2004-02-29'
```

■ `DATE_FORMAT(date, format)`

Funkcja formatuje wartość daty lub daty i godziny (*date*) zgodnie z ciągiem tekstowym formatowania (*format*) i zwraca wygenerowany ciąg tekstowy.

Funkcję `DATE_FORMAT()` można stosować w celu ponownego sformatowania wartości daty lub daty i godziny z formatu używanego przez MySQL na inny wybrany format.

```
DATE_FORMAT('2014-12-01', '%M %e, %Y') → 'December 1, 2014'
DATE_FORMAT('2014-12-01', 'The %D of %M') → 'The 1st of December'
```

W tabeli C.5 wymieniono dozwolone specyfikatory, które można stosować w ciągu tekstowym formatowania. Zakresy wymienione dla liczbowych specyfikatorów dnia i miesiąca rozpoczynają się od zera, ponieważ zero może spowodować, że data stanie się niekompletna, na przykład *'2004-00-13'* lub *'1998-12-00'*. Zmienna systemowa `lc_time_names` określa nazwy dni i miesięcy w danym języku dla specyfikatorów generujących wymienione nazwy.

Tabela C.5. Specyfikatory, które można stosować w ciągu tekstowym formatowania

Specyfikator	Opis
%f	Mikrosekundy w formie sześciocyfrowej (000000, 000001, ...).
%S, %s	Sekundy w formie dwucyfrowej (00, 01, ..., 59).
%i	Minuty w formie dwucyfrowej (00, 01, ..., 59).
%H	Godzina w formie dwucyfrowej, czas 24-godzinny (00, 01, ..., 23).
%h, %I	Godzina w formie dwucyfrowej, czas 12-godzinny (01, 02, ..., 12).
%k	Godzina w postaci liczbowej, czas 24-godzinny (0, 1, ..., 23).
%l	Godzina w postaci liczbowej, czas 12-godzinny (1, 2, ..., 12).
%T	Czas w formacie 24-godzinnym (<i>hh:mm:ss</i>).
%r	Czas w formacie 12-godzinnym (<i>hh:mm:ss AM</i> lub <i>hh:mm:ss PM</i>).
%p	Godzina przed południem lub po południu.
%W	Nazwy dni tygodnia (niedziela, poniedziałek, ..., sobota).
%a	Nazwy dni tygodnia w formie skrótu (ndz, pon, ..., sob).
%d	Dzień miesiąca w formie dwucyfrowej (00, 01, ..., 31).
%e	Dzień miesiąca w formie liczbowej (0, 1, ..., 31).
%D	Dzień miesiąca z angielskim przyrostkiem (0th, 1st, 2nd, 3rd, ...).
%w	Dzień tygodnia w formie liczbowej (0=niedziela, 1=poniedziałek, ..., 6=sobota).
%j	Dzień roku w formie trzycyfrowej (001, 002, ..., 366).
%U	Tydzień (00, ..., 53), w którym niedziela jest pierwszym dniem tygodnia.
%u	Tydzień (00, ..., 53), w którym poniedziałek jest pierwszym dniem tygodnia.
%V	Tydzień (01, ..., 53), w którym niedziela jest pierwszym dniem tygodnia.
%v	Tydzień (01, ..., 53), w którym poniedziałek jest pierwszym dniem tygodnia.
%M	Nazwa miesiąca (styczeń, luty, ..., grudzień).
%b	Nazwy miesiąca w postaci skrótu (sty, lut, ..., gru).
%m	Miesiące w formie dwucyfrowej (00, 01, ..., 12).
%c	Miesiące w formie liczbowej (0, 1, ..., 12).
%Y	Rok wyrażony w formie dwucyfrowej.
%y	Rok wyrażony w formie dwucyfrowej.
%X	Rok dla tygodnia, w którym niedziela jest pierwszym dniem, format czterocyfrowy.
%x	Rok dla tygodnia, w którym poniedziałek jest pierwszym dniem, format czterocyfrowy.
%%	Dosłowny znak %.

Znak % poprzedzający każdy kod formatu jest wymagany. Niewymienione w tabeli C.5 znaki po ich umieszczeniu w ciągu tekstowym formatowania są bez żadnych zmian kopiowane do wynikowego ciągu tekstowego.

Jeżeli odwołujesz się do specyfikatorów godziny w wartości typu DATE, wartością elementu godziny jest '00:00:00'.

```
DATE_FORMAT('2014-12-01', '%i') → '00'
```

■ DATE_SUB(*date*, INTERVAL *expr interval*)

Funkcja DATE_SUB() przeprowadza operacje arytmetyczne na dacie w ten sam sposób jak w przypadku funkcji DATE_ADD(), za wyjątkiem faktu, że wartość wyrażenia *expr* jest odejmowana od wartości daty lub daty i godziny (*date*). Więcej informacji znajdziesz w opisie funkcji DATE_ADD().

```
DATE_SUB('2009-12-01', INTERVAL 1 MONTH) → '2009-11-01'
DATE_SUB('2009-12-01', INTERVAL '13-2' YEAR_MONTH) → '1996-10-01'
DATE_SUB('2009-12-01 04:53:12', INTERVAL '13-2' MINUTE_SECOND)
→ '2009-12-01 04:40:10'
```

Podczas odejmowania daty można użyć alternatywnej składni:

```
'2009-12-01' - INTERVAL 1 MONTH → '2009-11-01'
```

W przypadku stosowania powyższej składni klauzula INTERVAL musi znajdować się po prawej stronie operatora odejmowania, ponieważ nie można odjąć daty od wartości odstępu czasu.

■ DATEDIFF(*expr1*, *expr2*)

Wartością zwrótną funkcji jest wyrażona w liczbie dni różnica między dwoma wyrażeniami, które powinny być wartościami daty lub daty i godziny. Wynik będzie dodatni, jeśli pierwszy argument wskazuje datę późniejszą niż drugi. Wszelkie elementy komponentu godziny będą ignorowane.

```
DATEDIFF('1987-01-01', '1987-01-08') → -7
DATEDIFF('1987-01-08', '1987-01-01') → 7
DATEDIFF('1987-01-01 12:00:00', '1987-01-08') → -7
DATEDIFF('1987-01-08', '1987-01-01 12:00:00') → 7
```

■ DAY(*date*)

To jest synonim funkcji DAYOFMONTH().

■ DAYNAME(*date*)

Funkcja zwraca ciąg tekstowy zawierający nazwę dnia dla wartości daty (*date*) lub NULL, jeśli nazwy nie można ustalić. Zmienna systemowa `lc_time_names` określa nazwy dni we wskazanym języku.

```
DAYNAME('2004-12-01') → 'Wednesday'
DAYNAME('1900-12-01') → 'Saturday'
DAYNAME('1900-12-00') → NULL
```

■ DAYOFMONTH(*date*)

Funkcja zwraca wartość liczbową wskazującą dzień miesiąca dla danej wartości daty (*date*). Wartość zwrótna jest z zakresu od 1 do 31 (0 dla dat częściowych pozbawionych elementu wskazującego dzień).

DAYOFMONTH('2002-12-01')	→ 1
DAYOFMONTH('2002-12-25')	→ 25
DAYOFMONTH('2002-12-00')	→ 0

■ **DAYOFWEEK(*date*)**

Funkcja zwraca wartość liczbową wskazującą dzień tygodnia dla danej wartości daty (*date*). Wartość zwrotna jest z zakresu od 1 (niedziela) do 7 (sobota), zgodnie ze standardem ODBC. Zapoznaj się również z opisem funkcji WEEKDAY().

DAYOFWEEK('2004-12-05')	→ 1
DAYNAME('2004-12-05')	→ 'Sunday'
DAYOFWEEK('2004-12-18')	→ 7
DAYNAME('2004-12-18')	→ 'Saturday'

■ **DAYOFYEAR(*date*)**

Funkcja zwraca wartość liczbową wskazującą dzień roku dla danej wartości daty (*date*). Wartość zwrotna jest z zakresu od 1 do 366.

DAYOFYEAR('2002-12-01')	→ 335
DAYOFYEAR('2004-12-31')	→ 366

■ **EXTRACT(*interval* FROM *datetime*)**

Funkcja zwraca element daty i czasu wartości *datetime* wskazany przez argument *interval*, który może być dowolnym specyfikatorem odstępu czasu dozwolonym do użycia w funkcji DATE_ADD().

EXTRACT(YEAR FROM '2002-12-01 13:42:19')	→ 2002
EXTRACT(MONTH FROM '2002-12-01 13:42:19')	→ 12
EXTRACT(DAY FROM '2002-12-01 13:42:19')	→ 1
EXTRACT(HOUR MINUTE FROM '2002-12-01 13:42:19')	→ 1342
EXTRACT(SECOND FROM '2002-12-01 13:42:19')	→ 19

Funkcja EXTRACT() może być używana wraz z datami, w których „brakuje” pewnych elementów.

EXTRACT(YEAR FROM '2004-00-12')	→ 2004
EXTRACT(MONTH FROM '2004-00-12')	→ 0
EXTRACT(DAY FROM '2004-00-12')	→ 12

■ **FROM_DAYS(*n*)**

Mając podaną wartość *n* przedstawiającą liczbę dni od roku 0 (najczęściej uzyskaną za pomocą wywołania TO_DAYS()), funkcja zwraca datę odpowiadającą podanej wartości.

TO_DAYS('2009-12-01')	→ 734107
FROM_DAYS(734107 + 3)	→ '2009-12-04'

Funkcja FROM_DAYS() jest przeznaczona do użycia jedynie z datami kalendarza gregoriańskiego (począwszy od roku 1582).

■ **FROM_UNIXTIME(*unix_timestamp*)**

FROM_UNIXTIME(*unix_timestamp*, *format*)

Mając podaną wartość znacznika czasu systemu UNIX (*unix_timestamp*), taką jak zwracana przez funkcję UNIX_TIMESTAMP(), funkcja FROM_UNIXTIME() zwraca datę i godzinę w strefie czasowej sesji. Wartość zwrotna jest typu DATETIME w formacie 'CCYY-MM-DD hh:mm:ss'. W przypadku użycia argumentu *format*

wartość zwrótna zostanie sformatowana według podanego ciągu tekstowego formatowania, jak przez funkcję `DATE_FORMAT()`.

```
UNIX_TIMESTAMP()           → 1328381427
FROM_UNIXTIME(1328381427)   → '2012-02-04 12:50:27'
FROM_UNIXTIME(1328381427, '%Y') → '2012'
```

■ `GET_FORMAT(val_type, format_type)`

Wartością zwrótną funkcji jest ciąg tekstowy formatowania, którego można użyć w wywołaniach funkcji `DATE_FORMAT()`, `TIME_FORMAT()` i `STR_TO_DATE()`. Argument *val_type* wskazuje typ danych: `DATE`, `TIME`, `DATETIME` lub `TIMESTAMP`. Z kolei argument *format_type* określa typ ciągu tekstowego formatowania: `'EUR'` (europejski), `'INTERNAL'` (międzynarodowy), `'ISO'` (ISO 9075, nie ISO 8601), `'JIS'` (japoński) lub `'USA'` (amerykański).

Funkcja `GET_FORMAT()` zwraca ciągi tekstowe formatowania dla każdego połączenia argumentów *val_type* i *format_type*, jak przedstawiono w tabeli C.6.

Tabela C.6. Ciągi tekstowe formatowania stosowane na skutek połączenia różnych typów argumentów

val_type	format_type	Ciąg tekstowy formatowania
DATE	'EUR'	'%d.%m.%Y'
DATE	'INTERNAL'	'%Y%m%d'
DATE	'ISO'	'%Y-%m-%d'
DATE	'JIS'	'%Y-%m-%d'
DATE	'USA'	'%m.%d.%Y'
TIME	'EUR'	'%H.%i.%s'
TIME	'INTERNAL'	'%H%i%s'
TIME	'ISO'	'%H:%i:%s'
TIME	'JIS'	'%H:%i:%s'
TIME	'USA'	'%h:%i:%s %p'
DATETIME	'EUR'	'%Y-%m-%d %H.%i.%s'
DATETIME	'INTERNAL'	'%Y%m%d%H%i%s'
DATETIME	'ISO'	'%Y-%m-%d %H:%i:%s'
DATETIME	'JIS'	'%Y-%m-%d %H:%i:%s'
DATETIME	'USA'	'%Y-%m-%d %H.%i.%s'

Zwróć uwagę, że element daty w ciągach tekstowych formatowania `'EUR'` i `'USA'` dla wartości `DATETIME` różni się od ciągów tekstowych formatowania `'EUR'` i `'USA'` dla wartości `DATE`.

■ **hour(*time*)**

Funkcja zwraca wartość liczbową wskazującą godzinę dla danej wartości godziny (*time*). Wartość zwrotna jest z zakresu od 0 do 23.

```
hour('12:31:58')           → 12
hour(123158)               → 12
```

■ **last_day(*date*)**

Funkcja zwraca datę dla ostatniego dnia miesiąca wskazanego przez argument. Argument *date* powinien być wartością daty lub daty i godziny.

```
last_day('2003-07-01')     → '2003-07-31'
last_day('2003-07-01 12:30:00') → '2003-07-31'
```

■ **localtime([*fsp*])**

localtimestamp([*fsp*])

Wymienione funkcje są synonimami dla funkcji now(). Nawiasy są opcjonalne, o ile nie zostanie podany argument *fsp*.

■ **makedate(*year*, *day_of_year*)**

Mając podany rok i dzień roku, funkcja zwraca wartość daty lub NULL, jeśli argument *day_of_year* ma wartość mniejszą niż 1.

```
makedate(2010,365)         → '2010-12-31'
makedate(2010,367)         → '2011-01-02'
makedate(2010,0)           → NULL
```

■ **maketime(*hour*, *minute*, *second*)**

Funkcja zwraca wartość godziny utworzoną na podstawie podanej godziny, minuty i sekundy. W przypadku, gdy wartości argumentów funkcji są spoza zakresu, wartością zwrotną będzie NULL. Wartości minuty i sekundy powinny być z zakresu od 0 do 59. Godzina może być spoza zakresu. Jeżeli wartość określająca godzinę będzie ujemna, wynik również będzie ujemny.

```
maketime(0,0,0)            → '00:00:00'
maketime(12,59,59)         → '12:59:59'
maketime(12,59,60)         → NULL
maketime(-12,59,59)        → '-12:59:59'
```

■ **microsecond(*expr*)**

Funkcja zwraca element wskazujący mikrosekundy w podanej wartości godziny lub daty i godziny. Wartość zwrotna może być z zakresu od 0 do 999999.

```
microsecond('00:00:00.000001'); → 1
microsecond('2004-06-30 23:59:59.5'); → 500000
```

■ **minute(*time*)**

Funkcja zwraca liczbową wartość wskazującą minutę w danej wartości godziny (*time*). Wartość zwrotna może być z zakresu od 0 do 59.

```
minute('12:31:58')         → 31
minute(123158)             → 31
```

■ MONTH(*date*)

Funkcja zwraca wartość liczbową wskazującą miesiąc roku dla danej wartości daty (*date*). Wartość zwrotna jest z zakresu od 1 do 12 (0 dla dat częściowych pozbawionych elementu wskazującego miesiąc).

```
MONTH('2002-12-01')      → 12
MONTH(20021201)          → 12
MONTH('2002-00-01')      → 0
```

■ MONTHNAME(*date*)

Funkcja zwraca ciąg tekstowy zawierający nazwę miesiąca dla wartości daty (*date*) lub NULL, jeśli nazwy nie można ustalić. Zmienna systemowa `lc_time_names` określa nazwy miesięcy we wskazanym języku.

```
MONTHNAME('2002-12-01')  → 'December'
MONTHNAME(20021201)      → 'December'
MONTHNAME('2002-00-01')  → NULL
```

■ NOW([*fsp*])

Funkcja zwraca bieżącą datę i godzinę dnia w strefie czasowej sesji. Wartość zwrotna jest typu DATETIME w formacie `'CCYY-MM-DD hh:mm:ss'`. Począwszy od MySQL 5.6.4, można podać opcjonalny argument *fsp*. W takim przypadku wartość zwrotna będzie zawierała część ułamkową o precyzji od 0 do 6 cyfr. Przed wersją MySQL 5.6.4 wszelkie argumenty funkcji są ignorowane.

```
NOW()                    → '2012-02-04 12:51:22'
```

Funkcja NOW() zwraca datę i godzinę, kiedy rozpoczyna się wykonanie zawierającego ją polecenia, niezależnie od czasu, przez jaki będzie trwało wykonywanie tego polecenia. Jeżeli wywołanie funkcji NOW() następuje w funkcji składowanej lub wyzwalaczu, wartość zwrotna zawiera godzinę rozpoczęcia wykonywania tej funkcji składowanej lub wyzwalacza. Porównaj zachowanie funkcji NOW() z zachowaniem funkcji SYSDATE().

■ PERIOD_ADD(*period*, *n*)

Funkcja dodaje *n* miesięcy do wartości *period* i zwraca wynik. Wartość zwrotna jest w formacie `CCYYMM`. Argument *period* może być w formacie `CCYYMM` lub `YYMM` (żaden z nich nie jest wartością daty).

```
PERIOD_ADD(201002,12)    → 201102
PERIOD_ADD(0802,-3)      → 200711
```

■ PERIOD_DIFF(*period1*, *period2*)

Funkcja pobiera różnicę argumentów i zwraca liczbę miesięcy dzielących wspomniane argumenty. Argumenty mogą być w formacie `CCYYMM` lub `YYMM` (żaden z nich nie jest wartością daty).

```
PERIOD_DIFF(200302,200202) → 12
PERIOD_DIFF(200711,0802)   → -3
```

■ QUARTER(*date*)

Funkcja zwraca wartość liczbową wskazującą kwartał roku dla danej wartości daty (*date*). Wartość zwrotna jest z zakresu od 1 do 4.

```

QUARTER('2008-12-01')      → 4
QUARTER('2009-01-01')      → 1

```

■ **SECOND(*time*)**

Funkcja zwraca liczbową wartość wskazującą sekundę w danej wartości godziny (*time*). Wartość zwrotna może być z zakresu od 0 do 59.

```

SECOND('12:31:58')         → 58
SECOND(123158)              → 58

```

■ **SEC_TO_TIME(*seconds*)**

Mając podaną liczbę sekund (*seconds*), funkcja zwraca odpowiadającą jej godzinę jako wartość typu TIME w formacie '*hh:mm:ss*'.

```

SEC_TO_TIME(29834)         → '08:17:14'

```

■ **STR_TO_DATE(*str*, *format_str*)**

Funkcja interpretuje argument w postaci ciągu tekstowego *str*, używając argumentu formatowania *format_str*, i zwraca wartość TIME, DATE lub DATETIME, w zależności od specyfikatorów formatowania wymienionych w *format_str*. Tę funkcję można wykorzystać do interpretacji wartości daty i godziny w formatach innych niż ISO. Funkcja STR_TO_DATE() przeprowadza operację odwrotną do wykonywanej przez funkcję DATE_FORMAT(). Specyfikatory formatu wymienione w opisie funkcji DATE_FORMAT() są jak najbardziej poprawne i obsługiwane przez funkcję STR_TO_DATE(). Jeżeli argument *str* jest nieprawidłowy i nie może być zinterpretowany za pomocą danego ciągu tekstowego formatowania, wartością zwrótną funkcji będzie NULL.

```

STR_TO_DATE('3/16/1960', '%m/%d/%Y')      → '1960-03-16'
STR_TO_DATE('12.20.32', '%H.%i.%s')       → '12:20:32'
STR_TO_DATE('3/16/1960 12:20:32', '%m/%d/%Y %H:%i:%s')
                                             → '1960-03-16 12:20:32'
STR_TO_DATE('3/16/1960', '%m-%d-%Y')       → NULL

```

■ **SUBDATE(*date*, INTERVAL *expr interval*)**

SUBDATE(*date*, *expr*)

W przypadku pierwszej składni funkcja SUBDATE() pobiera wartość daty lub daty i godziny (*date*), odejmuje wskazany odstęp czasu (*interval*) i zwraca wynik. To jest synonim funkcji DATE_SUB().

```

SUBDATE('2009-12-01', INTERVAL 1 MONTH)   → '2009-11-01'

```

W przypadku drugiej składni funkcja SUBDATE() pobiera wartość daty lub daty i godziny (*date*), odejmuje wartość wskazującą liczbę dni i zwraca wynik. To jest składnia podobna do stosowanej w funkcji ADDDATE().

```

SUBDATE('2009-12-01', 30)                  → '2009-11-01'

```

■ **SUBTIME(*expr1*, *expr2*)**

Funkcja odejmuje wartość drugiego wyrażenia od pierwszego i zwraca wynik. Wartością wyrażenia *expr1* powinna być godzina lub data i godzina, natomiast wartością wyrażenia *expr2* powinna być godzina.

```

SUBTIME('06:30:00.5', '12:30:00.4')       → '-05:59:59.900000'
SUBTIME('2009-01-01 00:00:00', '12:30:00') → '2008-12-31 11:30:00'

```

■ SYSDATE()

Funkcja zwraca bieżącą datę i godzinę dnia w strefie czasowej sesji. Wartość zwrótna jest typu DATETIME w formacie 'CCYY-MM-DD hh:mm:ss'. Ta funkcja jest podobna do funkcji NOW(), za wyjątkiem faktu, że SYSDATE() zwraca datę i godzinę w trakcie wykonywania funkcji, podczas gdy NOW() zwraca początek wykonywania polecenia zawierającego tę funkcję. (Więcej informacji znajdziesz w opisie funkcji NOW()). Aby funkcja SYSDATE() zachowywała się jak NOW(), serwer należy uruchomić z opcją --sysdate-is-now.

■ TIME(*expr*)

Funkcja zwraca element godziny wyrażenia *expr*, które powinno być wyrażeniem godziny lub daty i godziny.

```
TIME('16:15:00')           → '16:15:00'
TIME('2005-03-12 16:15:00') → '16:15:00'
```

■ TIME_FORMAT(*time*, *format*)

Funkcja formatuje wartość godziny (*time*) zgodnie z podanym ciągiem tekstowym formatowania i zwraca wygenerowany ciąg tekstowy. Omawiana funkcja akceptuje również argumenty w postaci DATETIME lub TIMESTAMP. Ciąg tekstowy formatowania jest taki sam, jak używany przez funkcję DATE_FORMAT(), ale dozwolone jest stosowanie jedynie specyfikatorów związanych z godziną. Użycie któregośkolwiek z pozostałych specyfikatorów skutkuje wygenerowaniem wartości zwrótniej NULL lub 0.

```
TIME_FORMAT('12:31:58', '%H %i') → '12 31'
TIME_FORMAT(123158, '%H %i')     → '12 31'
```

■ TIME_TO_SEC(*time*)

Mając podaną wartość godziny (*time*) przedstawiającą czas, który upłynął, funkcja zwraca liczbę wskazującą liczbę sekund odpowiadającą danej wartości godziny. Wartość zwrótna może być przekazana funkcji SEC_TO_TIME() w celu jej ponownej konwersji na wartość godziny.

```
TIME_TO_SEC('08:17:14') → 29834
SEC_TO_TIME(29834)      → '08:17:14'
```

Jeżeli podana zostanie wartość typu DATETIME lub TIMESTAMP, funkcja TIME_TO_SEC() ignoruje element daty.

```
TIME_TO_SEC('2012-03-26 08:17:14') → 29834
```

■ TIMEDIFF(*expr1*, *expr2*)

Funkcja zwraca różnicę czasu między wyrażeniami *expr1* i *expr2*, które przedstawiają odpowiednio godzinę początkową i końcową. Oba wyrażenia powinny być wartościami godziny lub daty i godziny, nie wolno mieszać wartości godziny z wartością daty i godziny.

```
TIMEDIFF('00:00:00', '09:30:45') → '-09:30:45'
TIMEDIFF('09:30:45', '00:00:00') → '09:30:45'
```

■ **TIMESTAMP(*expr1* [, *expr2*])**

Forma jednoargumentowa pobiera datę lub datę i godzinę wyrażenia *expr1* i zwraca wartość typu DATETIME. Z kolei forma dwuargumentowa dodaje wartość godziny wyrażenia *expr2* do *expr1* i zwraca wartość typu DATETIME.

```
TIMESTAMP('1985-12-14');           → '1985-12-14 00:00:00'
TIMESTAMP('1985-12-14 09:00:00');  → '1985-12-14 09:00:00'
TIMESTAMP('1985-12-14', '18:00:00'); → '1985-12-14 18:00:00'
TIMESTAMP('1985-12-14 09:00:00', '18:00:00'); → '1985-12-15 03:00:00'
TIMESTAMP('1985-12-14 09:00:00', '-18:00:00'); → '1985-12-13 15:00:00'
```

■ **TIMESTAMPADD(*interval*, *expr1*, *expr2*)**

Funkcja interpretuje *expr1* jako liczbę całkowitą w danych jednostkach wskazanych przez argument *interval*, dodaje ją do wartości daty lub daty i godziny wyrażenia *expr2* i zwraca wynik. Dozwolone wartości argumentu *interval* to MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER i YEAR. Każda z wymienionych wartości może być podana wraz z prefiksem SQL_TSI_.

```
TIMESTAMPADD(DAY, 12, '1995-07-01')   → '1995-07-13'
TIMESTAMPADD(MONTH, 12, '1995-07-01')  → '1996-07-01'
TIMESTAMPADD(SQL_TSI_MONTH, 12, '1995-07-01') → '1996-07-01'
```

■ **TIMESTAMPDIFF(*interval*, *expr1*, *expr2*)**

Funkcja oblicza różnicę między wyrażeniami daty lub daty i godziny (*expr1* i *expr2*), a następnie zwraca wynik w jednostkach wskazanych przez argument *interval*. Dozwolone są te same wartości *interval*, które wymieniono w opisie funkcji TIMESTAMPADD().

```
TIMESTAMPDIFF(DAY, '1995-07-01', '1995-08-01') → 31
TIMESTAMPDIFF(MONTH, '1995-07-01', '1995-08-01') → 1
```

■ **TO_DAYS(*date*)**

Funkcja zwraca wartość liczbową przedstawiającą wartość daty (*date*) skonwertowaną na liczbę dni, które upłynęły od roku 0. Wartość zwrótną można przekazać funkcji FROM_DAYS() w celu jej konwersji z powrotem na postać daty.

```
TO_DAYS('2010-12-01')           → 734472
FROM_DAYS(734472)                → '2010-12-01'
```

Jeżeli podana zostanie wartość typu DATETIME lub TIMESTAMP, wtedy funkcja TO_DAYS() ignoruje część dotyczącą godziny.

```
TO_DAYS('2010-12-01 12:14:37')   → 734472
```

Funkcja TO_DAYS() jest przeznaczona do użycia jedynie z datami kalendarza gregoriańskiego (począwszy od roku 1582).

■ **TO_SECONDS(*expr*)**

Dla podanego wyrażenia zawierającego wartość daty lub daty i godziny funkcja zwraca liczbę sekund, które upłynęły od roku 0. Jeżeli argument jest nieprawidłowy, wartością zwrótną będzie NULL.

```
TO_SECONDS('1790-07-17')         → 56504044800
```


■ UNIX_TIMESTAMP()

UNIX_TIMESTAMP(*date*)

Po wywołaniu bez argumentów funkcja zwraca liczbę sekund, które upłynęły od daty początku epoki '1970-01-01 00:00:00' czasu UTC. W przypadku wywołania z argumentem daty (*date*) wartością jest liczba sekund, które upłynęły między datą początku epoki i datą wskazaną w argumencie. Argument *date* może być podany jako wartość DATE, DATETIME lub TIMESTAMP bądź jako liczba w formacie CCYYMMDD lub YYMMDD. Serwer interpretuje argument *date* jako wartość w strefie czasowej sesji i konwertuje ją na czas UTC, o ile wartość nie pochodzi z kolumny typu TIMESTAMP (w takim przypadku wartość jest już przechowywana jako czas UTC).

```
UNIX_TIMESTAMP()                → 1328381580
UNIX_TIMESTAMP('2007-12-01')    → 1196488800
UNIX_TIMESTAMP(20071201)        → 1196488800
```

■ UTC_DATE()

Funkcja zwraca bieżącą datę UTC jako wartość DATE w formacie 'CCYY-MM-DD'. Nawiasy są opcjonalne.

```
UTC_DATE()                      → '2012-08-26'
```

■ UTC_TIME([fsp])

Funkcja zwraca bieżącą godzinę w strefie czasowej UTC. Wartość zwrotna jest typu TIME w formacie 'hh:mm:ss'. Począwszy od MySQL 5.6.4, można podać opcjonalny argument *fsp*. W takim przypadku wartość zwrotna będzie zawierała część ułamkową o precyzji od 0 do 6 cyfr. Przed wersją MySQL 5.6.4 wszelkie argumenty funkcji są ignorowane. Nawiasy są opcjonalne, o ile nie zostanie podany argument *fsp*.

```
UTC_TIME()                      → '20:00:18'
UTC_TIME(3)                     → '20:00:18.465'
```

■ UTC_TIMESTAMP([fsp])

Funkcja zwraca bieżącą datę i godzinę w strefie czasowej UTC. Wartość zwrotna jest typu DATETIME w formacie 'CCYY-MM-DD hh:mm:ss'. Począwszy od MySQL 5.6.4, można podać opcjonalny argument *fsp*. W takim przypadku wartość zwrotna będzie zawierała część ułamkową o precyzji od 0 do 6 cyfr. Przed wersją MySQL 5.6.4 wszelkie argumenty funkcji są ignorowane. Nawiasy są opcjonalne, o ile nie zostanie podany argument *fsp*.

```
UTC_TIMESTAMP()                 → '2012-08-26 20:48:23'
```

■ WEEK(*date* [,*mode*])

Po wywołaniu z pojedynczym argumentem funkcja zwraca wartość liczbową wskazującą tydzień w roku dla danej wartości daty (*date*). Wartość zwrotna jest z zakresu od 1 do 53. Przyjęto założenie, że tydzień rozpoczyna się w niedzielę. Po wywołaniu z dwoma argumentami funkcja WEEK() zwraca ten sam rodzaj wartości, ale argument *mode* wskazuje pierwszy dzień tygodnia, a wartość zwrotna jest z zakresu od 0 do 53 lub od 1 do 53. W tabeli C.7 wymieniono znaczenie wartości argumentu *mode*.

Tabela C.7. Znaczenie wartości argumentu *mode* funkcji WEEK()

Tryb	Dzień początkowy	Zakres wartości zwrotnej	Opis
0	niedziela	0–53	Za pierwszy tydzień będzie uznany tydzień, który obejmuje niedzielę.
1	poniedziałek	0–53	Za pierwszy tydzień będzie uznany tydzień obejmujący więcej niż trzy dni.
2	niedziela	1–53	Za pierwszy tydzień będzie uznany tydzień obejmujący niedzielę.
3	poniedziałek	1–53	Za pierwszy tydzień będzie uznany tydzień obejmujący więcej niż trzy dni.
4	niedziela	0–53	Za pierwszy tydzień będzie uznany tydzień obejmujący więcej niż trzy dni.
5	poniedziałek	0–53	Za pierwszy tydzień będzie uznany tydzień, który obejmuje poniedziałek.
6	niedziela	1–53	Za pierwszy tydzień będzie uznany tydzień obejmujący więcej niż trzy dni.
7	poniedziałek	1–53	Za pierwszy tydzień będzie uznany tydzień, który obejmuje poniedziałek.

Jeżeli argument *mode* zostanie pominięty, zastosowanie ma wartość zmiennej systemowej *default_week_format*.

```
WEEK('2003-12-08')           → 49
WEEK('2003-12-08',0)         → 49
WEEK('2003-12-08',1)         → 50
```

Wartość 0 dla WEEK() oznacza, że data dotyczy dnia przed pierwszym wystąpieniem dnia rozpoczynającego tydzień (niedziela lub poniedziałek, w zależności od wartości argumentu *mode*).

```
WEEK('2005-01-01')           → 0
DAYNAME('2005-01-01')        → 'Saturday'
WEEK('2006-01-01',1)         → 0
DAYNAME('2006-01-01')        → 'Sunday'
```

■ WEEKDAY(*date*)

Funkcja zwraca wartość liczbową wskazującą dzień tygodnia dla danej wartości daty (*date*) lub NULL, jeśli nie można ustalić dnia. Wartość zwrotna jest z zakresu od 0 (niedziela) do 6 (sobota). Zapoznaj się również z opisem funkcji DAYOFWEEK().

```
WEEKDAY('2002-12-08')        → 6
DAYNAME('2002-12-08')        → 'Sunday'
WEEKDAY('2002-12-16')        → 0
DAYNAME('2002-12-16')        → 'Monday'
WEEKDAY('2002-12-00')        → NULL
```

■ **WEEKOFYEAR(*date*)**

Funkcja działa dokładnie tak samo jak wywołanie `WEEK(date, 3)`.

■ **YEAR(*date*)**

Funkcja zwraca wartość liczbową wskazującą rok dla danej wartości daty (*date*).

`YEAR('1974-12-01')` → 1974

`YEAR(19741201)` → 1974

■ **YEARWEEK(*date* [,*mode*])**

Funkcja zwraca liczbę w formacie *CCYYWW* wskazującą rok i tydzień roku dla podanej wartości daty (*date*). Jeśli zostanie podany argument *mode*, ma takie samo działanie jak w przypadku funkcji `WEEK()`.

`YEARWEEK('2006-01-01')` → 200601

`YEARWEEK('2006-01-01', 0)` → 200601

`YEARWEEK('2006-01-01', 1)` → 200552

W przypadku pierwszego i ostatniego tygodnia roku rok podany w wyniku może się różnić od roku użytego w argumencie.

`WEEK('2008-01-01')` → 0

`YEARWEEK('2008-01-01')` → 200752

C.2.6. Funkcje podsumowań

Funkcje podsumowań (inaczej „agregujące”) obliczają pojedynczą wartość na podstawie grupy wartości. Wartość wynikowa jest obliczana jedynie na podstawie wartości innych niż NULL w grupie, za wyjątkiem funkcji `COUNT(*)`, zliczającej wszystkie rekordy. Funkcje agregujące są użyteczne podczas tworzenia podsumowań całego zestawu wartości lub jeśli zapytanie zawiera klauzulę `GROUP BY` do generowania podsumowań dla wszystkich podgrup zbioru wartości. Patrz podpunkt 1.4.9.9, zatytułowany „Generowanie podsumowania”.

Na potrzeby przykładów przedstawionych w tym punkcie przyjmujemy założenie istnienia tabeli `mytbl` z kolumną o nazwie `mycol` przeznaczoną do przechowywania liczb całkowitych i zawierającą osiem rekordów z wartościami 1, 3, 5, 5, 7, 9, 9 i NULL.

```
mysql> SELECT mycol FROM mytbl;
```

```
+-----+
| mycol |
+-----+
| 1     |
| 3     |
| 5     |
| 5     |
| 7     |
| 9     |
| 9     |
| NULL  |
+-----+
```

■ **AVG([DISTINCT] *expr*)**

Funkcja zwraca wartość średnią obliczoną dla wszystkich wartości innych niż NULL wyrażenia *expr*. Jeśli nie ma wartości innych niż NULL, wtedy wartością zwrótną funkcji jest NULL.

```
SELECT AVG(mycol) FROM mytbl           → 5.5714
SELECT AVG(mycol)*2 FROM mytbl         → 11.1429
SELECT AVG(mycol*2) FROM mytbl         → 11.1429
```

Słowo kluczowe **DISTINCT** powoduje, że funkcja **AVG()** zwraca średnią unikalnych wartości wyrażenia *expr*.

• **BIT_AND(*expr*)**

Funkcja zwraca bitową wartość AND dla wszystkich wartości innych niż NULL wyrażenia *expr*. Jeśli nie ma wartości innych niż NULL, wtedy wartością zwrótną funkcji jest ~0.

```
SELECT BIT_AND(mycol) FROM mytbl       → 1
```

■ **BIT_OR(*expr*)**

Funkcja zwraca bitową wartość OR dla wszystkich wartości innych niż NULL wyrażenia *expr*. Jeśli nie ma wartości innych niż NULL, wtedy wartością zwrótną funkcji jest 0.

```
SELECT BIT_OR(mycol) FROM mytbl        → 15
```

■ **BIT_XOR(*expr*)**

Funkcja zwraca bitową wartość XOR dla wszystkich wartości innych niż NULL wyrażenia *expr*. Jeśli nie ma wartości innych niż NULL, wtedy wartością zwrótną funkcji jest 0.

```
SELECT BIT_XOR(mycol) FROM mytbl       → 5
```

■ **COUNT(*expr*)**

COUNT(*)

COUNT(DISTINCT *expr1*, *expr2*,...)

W przypadku argumentu w postaci wyrażenia funkcja zwraca liczbę wartości innych niż NULL lub wyrażenie *expr*. Jeśli nie ma wartości innych niż NULL, wtedy wartością zwrótną funkcji jest 0. Jeżeli argumentem jest gwiazdka (*), funkcja zwraca liczbę rekordów znajdujących się w zbiorze wynikowym, niezależnie od ich zawartości.

```
SELECT COUNT(mycol) FROM mytbl         → 7
SELECT COUNT(*) FROM mytbl             → 8
```

Dla tabel **MyISAM** funkcja **COUNT(*)** bez klauzuli **WHERE** jest zoptymalizowana w celu szybkiego zwrócenia liczby rekordów znajdujących się w tabeli wymienionej w klauzuli **FROM**. Jeśli podana będzie więcej niż jedna tabela, funkcja **COUNT(*)** zwraca iloczyn liczby rekordów w poszczególnych tabelach.

```
SELECT COUNT(*) FROM mytbl AS m1 INNER JOIN mytbl AS m2
→ 64
```

Słowo kluczowe **DISTINCT** powoduje, że funkcja **AVG()** zwraca średnią unikalnych wartości wyrażenia *expr*.

```
SELECT COUNT(DISTINCT mycol) FROM mytbl → 5
SELECT COUNT(DISTINCT MOD(mycol,3)) FROM mytbl → 3
```

W przypadku podania wielu wyrażeń funkcja COUNT(DISTINCT) zlicza liczbę unikalnych połączeń wartości innych niż NULL.

■ **GROUP_CONCAT([DISTINCT] *var_list* [ORDER BY ...] [SEPARATOR *str*])**

Ta funkcja łączy wartości inne niż NULL w grupę ciągów tekstowych i zwraca wynik. Jeśli nie ma wartości innych niż NULL, wtedy wartością zwrótną funkcji jest NULL. Słowo kluczowe DISTINCT można wykorzystać do pozbycia się duplikatów, ORDER BY do posortowania wyniku, a SEPARATOR do określenia separatora między ciągami tekstowymi. Domyślnie funkcja GROUP_CONCAT() nie usuwa duplikatów i nie sortuje wyniku, a wartości są rozdzielone przecinkami.

```
mysql> CREATE TABLE t (name CHAR(10));
mysql> INSERT INTO t VALUES('dog'),('cat'),('rat'),('dog'),('rat');
mysql> SELECT GROUP_CONCAT(name) FROM t;
+-----+
| GROUP_CONCAT(name) |
+-----+
| dog,cat,rat,dog,rat |
+-----+
mysql> SELECT GROUP_CONCAT(name SEPARATOR ':') FROM t;
+-----+
| GROUP_CONCAT(name SEPARATOR ':') |
+-----+
| dog:cat:rat:dog:rat |
+-----+
mysql> SELECT GROUP_CONCAT(name ORDER BY name DESC) FROM t;
+-----+
| GROUP_CONCAT(name ORDER BY name DESC) |
+-----+
| rat,rat,dog,dog,cat |
+-----+
mysql> SELECT GROUP_CONCAT(DISTINCT name ORDER BY name) FROM t;
+-----+
| GROUP_CONCAT(DISTINCT name ORDER BY name) |
+-----+
| cat,dog,rat |
+-----+
```

Wartości zwracane przez funkcję GROUP_CONCAT() mają długość ograniczoną do wartości zmiennej systemowej group_concat_max_len. Aby zezwolić na stosowanie dłuższych wartości, należy zwiększyć wartość wymienionej zmiennej.

■ **MAX([DISTINCT] *expr*)**

Funkcja zwraca maksymalną wartość inną niż NULL wyrażenia *expr*. Jeśli nie ma wartości innych niż NULL, wtedy wartością zwrótną funkcji jest NULL. W przypadku ciągów tekstowych lub wartości daty i godziny funkcja MAX() zwraca największą wartość pod względem odpowiednio leksykalnym lub daty i godziny.

```
SELECT MAX(mycol) FROM mytbl
```

Słowo kluczowe DISTINCT powoduje, że funkcja MAX() zwraca maksymalną unikalną wartość wyrażenia *expr* (co nie powoduje zmiany wyniku).

■ **MIN([DISTINCT] *expr*)**

Funkcja zwraca minimalną wartość inną niż NULL wyrażenia *expr*. Jeśli nie ma wartości innych niż NULL, wtedy wartością zwrótną funkcji jest NULL. W przypadku ciągów tekstowych lub wartości daty i godziny funkcja MIN() zwraca najmniejszą wartość pod względem odpowiednio leksykalnym lub daty i godziny.

```
SELECT MIN(mycol) FROM mytbl
```

→ 1

Słowo kluczowe DISTINCT powoduje, że funkcja MIN() zwraca maksymalną unikalną wartość wyrażenia *expr* (co nie powoduje zmiany wyniku).

■ **STD(*expr*)**

STDDEV(*expr*)

STDDEV_POP(*expr*)

Funkcja zwraca standardowe odchylenie populacji dla wartości innych niż NULL wyrażenia *expr*. Jeśli nie ma wartości innych niż NULL, wtedy wartością zwrótną funkcji jest NULL.

```
SELECT STDDEV_POP(mycol) FROM mytbl
```

→ 2.7701

■ **STDDEV_SAMP(*expr*)**

Funkcja zwraca standardowe odchylenie z próby dla wartości innych niż NULL wyrażenia *expr*. Jeśli nie ma wartości innych niż NULL, wtedy wartością zwrótną funkcji jest NULL.

```
SELECT STDDEV_SAMP(mycol) FROM mytbl
```

→ 2.9921

■ **SUM([DISTINCT] *expr*)**

Funkcja zwraca sumę wartości innych niż NULL wyrażenia *expr*. Jeśli nie ma wartości innych niż NULL, wtedy wartością zwrótną funkcji jest NULL.

```
SELECT SUM(mycol) FROM mytbl
```

→ 39

Słowo kluczowe DISTINCT powoduje, że funkcja SUM() zwraca sumę unikalnych wartości wyrażenia *expr*.

■ **VARIANCE(*expr*)**

VAR_POP(*expr*)

Funkcja zwraca wariancję populacji dla wartości innych niż NULL wyrażenia *expr*. Jeśli nie ma wartości innych niż NULL, wtedy wartością zwrótną funkcji jest NULL.

```
SELECT VAR_POP(mycol) FROM mytbl
```

→ 7.6735

■ **VAR_SAMP(*expr*)**

Funkcja zwraca wariancję z próby dla wartości innych niż NULL wyrażenia *expr*. Jeśli nie ma wartości innych niż NULL, wtedy wartością zwrótną funkcji jest NULL.

```
SELECT VAR_SAMP(mycol) FROM mytbl
```

→ 8.9524

C.2.7. Funkcje zapewnienia bezpieczeństwa oraz związane z kompresją

Omówione w tym punkcie funkcje przeprowadzają operacje związane z bezpieczeństwem, na przykład szyfrowanie ciągów tekstowych, oraz z kompresją. Wiele z tych funkcji jest dostarczanych w parach, w których jedna generuje zaszyfrowaną wartość, natomiast druga deszyfruje tę wartość. Tego rodzaju pary funkcji zwykle używają ciągu tekstowego jako klucza lub hasła. Aby przywrócić pierwotną (niezaszyfrowaną) wartość, konieczne jest przeprowadzenie operacji deszyfrowania wartości za pomocą tego samego klucza, który był użyty do jej zaszyfrowania. W przeciwnym razie wynik operacji deszyfrowania jest bezsensowny.

W celu wstawienia do bazy danych wyniku działania funkcji szyfrującej zwracającej binarny ciąg tekstowy konieczne jest użycie kolumny przeznaczonej do przechowywania danych typu binarny ciąg tekstowy (VARBINARY lub jeden z typów BLOB). W ten sposób unikniesz ewentualnych problemów, które mogą się pojawić w przypadku użycia typu niebinarnego ciągu tekstowego. Przykłady wspomnianych problemów to konwersja kodowania znaków lub usunięcie spacji znajdujących się na końcu ciągu tekstowego.

Pewne funkcje szyfrujące zwracają ciągi tekstowe ASCII, a wartość zwrótna jest niebinarnym ciągiem tekstowym w kodowaniu znaków stosowanym przez połączenie. Przykłady tego rodzaju funkcji to MD5(), OLD_PASSWORD(), PASSWORD(), SHA(), SHA1() i SHA2(). Wymienione funkcje zwracają binarne ciągi tekstowe w wersjach MySQL wcześniejszych niż 5.5.3 (5.5.6 w przypadku SHA2()).

■ AES_DECRYPT(*str*, *key_str*)

Mając zaszyfrowany ciąg tekstowy *str* pobrany jako wynik wywołania funkcji AES_ENCRYPT(), funkcja przeprowadza deszyfrowanie wspomnianego ciągu tekstowego za pomocą klucza *key_str* i zwraca wynikowy ciąg tekstowy. Jeżeli którykolwiek argument ma wartość NULL, wtedy wartością zwrótną funkcji również będzie NULL.

```
AES_DECRYPT(AES_ENCRYPT('secret','scramble'),'scramble')
→ 'secret'
```

■ AES_ENCRYPT(*str*, *key_str*)

Funkcja szyfruje ciąg tekstowy *str* za pomocą klucza *key_str*, stosując algorytm AES (ang. *Advanced Encryption Standard*) i klucz o długości 128 bitów. Wynik jest zwracany w postaci binarnego ciągu tekstowego. Jeżeli którykolwiek argument ma wartość NULL, wtedy wartością zwrótną funkcji również będzie NULL. Zaszyfrowany ciąg tekstowy można odszyfrować za pomocą funkcji AES_DECRYPT(), używając tego samego klucza.

■ COMPRESS(*str*)

Funkcja zwraca skompresowaną wersję ciągu tekstowego podanego jako jej argument lub wartość NULL, jeśli serwer nie został skompilowany z obsługą biblioteki kompresji. Wartość zwrótna jest w postaci binarnego ciągu tekstowego.

■ `DECODE(str, key_str)`

Mając zaszyfrowany ciąg tekstowy *str* pobrany jako wynik wywołania funkcji `ENCODE()`, funkcja przeprowadza deszyfrowanie wspomnianego ciągu tekstowego za pomocą klucza *key_str* i zwraca wynikowy ciąg tekstowy. Jeżeli którykolwiek argument ma wartość `NULL`, wtedy wartością zwrótną funkcji również będzie `NULL`.

`DECODE(ENCODE('secret', 'scramble'), 'scramble') → 'secret'`

■ `DES_DECRYPT(str [,key_str])`

Funkcja deszyfruje ciąg tekstowy *str*, który powinien być wartością zaszyfrowaną przez funkcję `DES_ENCRYPT()`. Jeżeli nie została włączona obsługa SSL lub operacja deszyfrowania zakończy się niepowodzeniem, wartością zwrótną funkcji będzie `NULL`.

Jeżeli podany będzie argument *key_str*, zostanie użyty jako klucz deszyfrowania. W przeciwnym razie do odszyfrowania ciągu tekstowego funkcja `DES_DECRYPT()` użyje klucza pochodzącego z pliku klucza DES serwera. Numer wpisu klucza jest określany na podstawie bitów 0–6 pierwszego bajta zaszyfrowanego ciągu tekstowego. Położenie pliku klucza jest wskazywane w trakcie uruchamiania serwera przez podanie opcji `--des-key-file`.

Jeżeli ciąg tekstowy *str* nie przypomina zaszyfrowanego ciągu tekstowego, funkcja `DES_DECRYPT()` zwraca niezmieniony ciąg tekstowy. (Taka sytuacja może wystąpić, jeśli na przykład pierwszy bajt ciągu tekstowego nie ma ustawionego bitu 7).

Użycie jednoargumentowej formy funkcji `DES_DECRYPT()` wymaga uprawnienia `SUPER`.

■ `DES_ENCRYPT(str [, {key_num|key_str}])`

Funkcja szyfruje ciąg tekstowy *str*, wynik jest zwracany w postaci binarnego ciągu tekstowego, który można później deszyfrować za pomocą funkcji `DES_DECRYPT()`. Jeżeli nie została włączona obsługa SSL lub operacja szyfrowania zakończy się niepowodzeniem, wartością zwrótną funkcji będzie `NULL`.

Jeżeli podany będzie argument *key_str*, zostanie użyty jako klucz szyfrowania. Argument *key_str* powinien mieć wartość od 0 do 9, wskazującą numer wpisu klucza w pliku klucza DES serwera. W takim przypadku wykorzystany będzie klucz szyfrowania z podanego wpisu. Jeżeli nie zostanie podany argument *key_str* lub *key_num*, szyfrowanie zostanie przeprowadzone za pomocą pierwszego klucza pochodzącego z pliku klucza DES serwera. (To niekoniecznie będzie taki sam klucz jak po podaniu wartości 0 dla argumentu *key_num*).

Pierwszy bajt wygenerowanego ciągu tekstowego wskazuje sposób jego zaszyfrowania. Ten bajt będzie miał ustawiony bit 7, a bity od 0 do 6 wskazują numer klucza. Wspomniany numer jest z zakresu od 0 do 9 i określa klucz z pliku klucza DES, który będzie używany do szyfrowania ciągu tekstowego. Jeżeli został użyty argument *key_str*, wartość wynosi 127. Na przykład, w przypadku szyfrowania ciągu tekstowego za pomocą klucza 3 pierwszy bajt wyniku ma

wartość 131 (128+3). W przypadku szyfrowania za pomocą wartości *key_str* pierwszy bajt ma wartość 255 (128+127).

Jeżeli szyfrowanie jest przeprowadzane na podstawie numeru klucza, serwer odczytuje plik klucza DES w celu odszukania odpowiedniego ciągu tekstowego klucza. Położenie pliku klucza jest wskazywane w trakcie uruchamiania serwera przez podanie opcji `--des-key-file`. Wspomniany plik klucza zawiera wiersze w następującym formacie:

```
key_num key_str
```

Każda wartość *key_num* powinna być liczbą z zakresu od 0 do 9, natomiast *key_str* to odpowiadający jej klucz szyfrowania. Wartości *key_num* i *key_str* powinny być rozdzielone przynajmniej jednym znakiem odstępu. Wiersze w pliku klucza można ułożyć w dowolnej kolejności.

W przeciwieństwie do funkcji `DES_DECRYPT()`, funkcja `DES_ENCRYPT()` nie wymaga uprawnień SUPER do użycia kluczy z pliku klucza DES. (Każdy może szyfrować informacje na podstawie pliku klucza, natomiast tylko uprawnieni użytkownicy mogą używać klucza do deszyfrowania informacji).

- **ENCODE(*str*, *key_str*)**

Funkcja szyfruje ciąg tekstowy *str* za pomocą klucza *key_str* i zwraca wynik w postaci binarnego ciągu tekstowego. Zasyfrowany ciąg tekstowy można odszyfrować za pomocą funkcji `DECODE()`, używając tego samego klucza.

- **ENCRYPT(*str* [,*salt*])**

Funkcja szyfruje ciąg tekstowy *str*, wynik jest zwracany w postaci binarnego ciągu tekstowego. Jeżeli którykolwiek argument ma wartość NULL, wtedy wartością zwrótną funkcji również będzie NULL. To jest szyfrowanie jednokierunkowe. Jeśli zostanie podany argument *salt*, powinien być ciągiem tekstowym składającym się z przynajmniej dwóch znaków. Podanie wartości *salt* oznacza, że zasyfrowany wynik *str* będzie za każdym razem taki sam. W przypadku braku argumentu *salt* MySQL użyje losowo wybranej wartości i wówczas identyczne wywołania funkcji `ENCRYPT()` będą za każdym razem generowały inne wyniki.

```
ENCRYPT('secret','AB')           → 'ABS5SGh1EL6bk'
ENCRYPT('secret','AB')           → 'ABS5SGh1EL6bk'
ENCRYPT('secret')                → 'ByUthKNv3.LsE'
ENCRYPT('secret')                → 'Hzx4rhb7Qdvpk'
```

Funkcja `ENCRYPT()` używa oferowanego przez system UNIX wywołania systemowego `crypt()`, a jej zachowanie zależy od systemu. W pewnych systemach funkcja `crypt()` sprawdza jedynie pierwsze osiem znaków ciągu tekstowego przeznaczonego do zasyfrowania. Jeśli wywołanie `crypt()` jest niedostępne w Twoim systemie, funkcja `ENCRYPT()` zawsze będzie zwracać wartość NULL.

Ciąg tekstowy *str* nie powinien stosować kodowania znaków `ucs2`, `utf16`, `utf16le` lub `utf32`. Wywołanie `crypt()` interpretuje bajt NUL jako koniec argumentu, choć wymienione kodowania znaków zezwalają na stosowanie NUL w ciągu tekstowym.

■ MD5(*str*)

Funkcja oblicza 128-bitową sumę kontrolną ciągu tekstowego *str*, opierając się na algorytmie RSA Data Security, Inc. MD5 Message-Digest. Wartością zwrótną jest niebinarny ciąg tekstowy składający się z 32 cyfr szesnastkowych lub NULL, jeśli argument miał wartość NULL. Zapoznaj się z przedstawionymi na początku punktu informacjami dotyczącymi typu wartości zwrótnego ciągu tekstowego.

```
MD5('secret')           → '5ebe2294ecd0e0f08eab7690d2a6ee69'
```

■ OLD_PASSWORD(*str*)

Funkcja zwraca zaszyfrowane hasło; w wersjach wcześniejszych niż MySQL 4.1 ta funkcja nosiła nazwę PASSWORD(). Zapoznaj się z przedstawionymi na początku punktu informacjami dotyczącymi typu wartości zwrótnego ciągu tekstowego.

■ PASSWORD(*str*)

Mając podany ciąg tekstowy *str*, funkcja oblicza i zwraca ciąg tekstowy zawierający zaszyfrowane hasło w postaci używanej w tabelach uprawnień MySQL. To jest szyfrowanie jednokierunkowe. Zapoznaj się z przedstawionymi na początku punktu informacjami dotyczącymi typu wartości zwrótnego ciągu tekstowego.

```
PASSWORD('secret')      → '*14E65567ABDB5135D0CFD9A70B3032C179A49EE7'
```

Funkcja PASSWORD() *nie* używa takiego samego algorytmu jak stosowany przez system UNIX do szyfrowania haseł kont użytkowników. Do tego rodzaju szyfrowania należy użyć funkcji ENCRYPT().

Jeżeli zmienna systemowa old_passwords ma wartość 1, funkcja PASSWORD() szyfruje hasła za pomocą algorytmu używanego w wersjach wcześniejszych niż MySQL 4.1. W takim przypadku funkcje PASSWORD() i OLD_PASSWORD() zwracają taką samą wartość.

■ SHA1(*str*)

SHA(*str*)

Funkcja oblicza 160-bitową sumę kontrolną ciągu tekstowego *str*, opierając się na algorytmie *Secure Hash Algorithm*. Wartością zwrótną jest niebinarny ciąg tekstowy składający się z 40 cyfr szesnastkowych lub NULL, jeśli argument miał wartość NULL. Zapoznaj się z przedstawionymi na początku punktu informacjami dotyczącymi typu wartości zwrótnego ciągu tekstowego.

```
SHA1('secret')          → 'e5e9fa1ba31ecd1ae84f75caaa474f3a663f05f4'
```

Funkcja SHA1() jest znacznie bezpieczniejsza niż MD5(), ale mniej bezpieczna niż SHA2().

■ SHA2(*str*,*hash_length*)

Funkcja podobna do SHA1(), ale znacznie bezpieczniejsza. Tworzy wartość hash na podstawie pierwszego argumentu i generuje wynik o wyrażonej w bitach długości wskazanej przez drugi argument. Wspomnianą długością musi być 224, 256, 384 lub 512. Wartością zwrótną jest niebinarny ciąg tekstowy

składający się ze wskazanej liczby bitów przedstawionych jako cyfry szesnastkowe. Jeśli argument miał wartość NULL lub podano nieprawidłową długość, wtedy wartością zwrótną będzie NULL. Zapoznaj się z przedstawionymi na początku punktu informacjami dotyczącymi typu wartości zwrótej ciągu tekstowego.

```
SHA2('secret',224)
→ '95c7fbca92ac5083afda62a564a3d014fc3b72c9140e3cb99ea6bf12'
```

Funkcja SHA2() została wprowadzona w wersji MySQL 5.5.5.

■ UNCOMPRESS(str)

Mając ciąg tekstowy skompresowany za pomocą funkcji COMPRESS(), funkcja UNCOMPRESS() przywraca początkowy ciąg tekstowy. Wartością zwrótną jest NULL, jeśli argument nie jest skompresowanym ciągiem tekstowym lub jeżeli serwer nie został skompilowany wraz z biblioteką kompresji.

■ UNCOMPRESSED_LENGTH(str)

Mając ciąg tekstowy skompresowany za pomocą funkcji COMPRESS(), funkcja UNCOMPRESSED_LENGTH() zwraca długość początkowego, nieskompresowanego ciągu tekstowego. Wartością zwrótną jest NULL, jeśli serwer nie został skompilowany wraz z biblioteką kompresji.

C.2.8. Funkcje nakładania blokad doradczych

Funkcje omówione w tym punkcie implementują nakładanie blokad doradczych. Możesz je wykorzystywać podczas tworzenia aplikacji współpracujących na podstawie stanu blokady o ustalonej wcześniej nazwie. Podstawowe funkcje, o nazwach GET_LOCK() i RELEASE_LOCK(), są odpowiedzialne za odpowiednio nakładanie i zwalnianie blokad. Dwie pozostałe funkcje, IS_FREE_LOCK() i IS_USED_LOCK(), sprawdzają stan blokady w celu ustalenia, czy klient nałożył blokadę.

W przypadku omawianych blokad doradczych ich cechą charakterystyczną jest to, że nadajesz nazwę blokadzie; wspomniana nazwa to po prostu tylko ciąg tekstowy. Blokada doradcza jest prywatna w tym sensie, że tylko klient, który ją nałożył, może zwolnić tę blokadę. Z drugiej strony, jest globalna, ponieważ każdy klient może sprawdzić stan blokady o danej nazwie.

Nałożenie blokady wymaga wywołania funkcji GET_LOCK(str, timeout), gdzie str określa nazwę blokady, natomiast timeout wyrażony w sekundach czas utraty ważności wywołania funkcji. Funkcja GET_LOCK() zwraca wartość 1, jeśli nałożenie blokady zakończyło się powodzeniem we wskazanym czasie, lub wartość 0, jeśli nałożenie blokady zakończyło się niepowodzeniem ze względu na minięcie wskazanego czasu. W przypadku wystąpienia błędu wartością zwrótną jest NULL.

Wartość timeout wskazuje, jak długo należy czekać podczas próby nałożenia blokady; nie wskazuje czasu trwania blokady. Po nałożeniu blokady pozostaje ona ważna aż do chwili jej zwolnienia.

Poniższe wywołanie powoduje nałożenie blokady o nazwie `Nellie` i oczekiwanie maksymalnie 10 sekund na jej nałożenie:

```
GET_LOCK('Nellie', 10)
```

Blokada ma zastosowanie jedynie względem samego ciągu tekstowego. Nie powoduje nałożenia blokady na bazę danych, tabelę, rekordy lub kolumny w tabeli. Innymi słowy, blokada nie uniemożliwia innym klientom przeprowadzania dowolnych operacji na tabelach bazy danych. Z tego powodu funkcja `GET_LOCK()` jest jedynie blokadą zalecaną — po prostu pozwala innym klientom na sprawdzenie, czy blokada jest stosowana.

Klient nakładający blokadę na nazwę uniemożliwia innym klientom nałożenie blokady na tę nazwę (lub podejmowanie takich prób przez inne wątki, jeśli dana sesja z serwerem jest obsługiwana przez klienta wielowątkowego). Przyjmujemy założenie, że klient 1 nakłada blokadę na ciąg tekstowy `Nellie`. Jeżeli klient 2 spróbuje nałożyć blokadę na ten sam ciąg tekstowy, nie będzie miał takiej możliwości aż do chwili zwolnienia blokady przez klienta 1 lub upłynięcia czasu utraty ważności. Jeżeli klient 1 zwolni blokadę we wspomnianym czasie, podjęta przez klienta 2 próba nałożenia blokady zakończy się powodzeniem. W przeciwnym przypadku zakończy się niepowodzeniem.

Ponieważ dwie sesje nie mogą w tym samym czasie nałożyć blokad na ten sam ciąg tekstowy, aplikacja zgadzająca się na nazwę używa stanu blokady dla danej nazwy jako wskaźnika, czy można bezpiecznie przeprowadzać operacje związane z tą nazwą. Na przykład, możesz przygotować nazwę blokady na podstawie klucza unikalnej wartości rekordu w tabeli, aby zezwolić na stosowanie blokad dla danego rekordu.

W celu zwolnienia blokady należy wywołać funkcję `RELEASE_LOCK()` wraz z nazwą blokady:

```
RELEASE_LOCK('Nellie')
```

Wartością zwrótną funkcji `RELEASE_LOCK()` jest 1, jeśli zwolnienie blokady zakończyło się powodzeniem, 0, jeśli blokada należy do innej sesji, lub `NULL`, jeśli wskazana blokada nie istnieje.

Dowolna blokada nałożona przez klienta zostaje automatycznie zwolniona po wydaniu przez niego innego wywołania `GET_LOCK()`, ponieważ w danej chwili w sesji klienta może być zablokowany tylko jeden ciąg tekstowy. W takim przypadku przed nałożeniem nowej blokady następuje zwolnienie poprzedniej, nawet jeśli ich nazwy są takie same. Zwolnienie blokady następuje również po przerwaniu sesji klienta. Dlatego też, jeśli dla długo działającego klienta minie czas ważności ze względu na brak aktywności, serwer zwolni nałożoną przez niego blokadę.

Aby sprawdzić stan blokady dla nazwy, masz dwie możliwości:

- Wywołać funkcję `IS_FREE_LOCK(str)`, która zwraca wartość 1, jeśli nazwa nie jest aktualnie zablokowana, wartość 0, jeśli nazwa jest zablokowana, lub wartość `NULL`, jeśli wystąpił błąd.
- Wywołać funkcję `IS_USED_LOCK(str)` zwracającą identyfikator połączenia klienta, który nałożył blokadę na daną nazwę. Jeżeli nie ma wskazanej blokady, wartością zwrótną funkcji będzie `NULL`.

Wywołanie `GET_LOCK(str, 0)` można wykorzystać jako proste sprawdzenie bez oczekiwania na nałożenie blokady na ciąg tekstowy `str`. Jednak takie rozwiązanie ma efekt uboczny w postaci nałożenia blokady na podany ciąg tekstowy, o ile nie jest aktualnie zablokowany. Jeśli zachodzi potrzeba, to nie zapomnij o wywołaniu `RELEASE_LOCK()`.

Wszystkie funkcje nakładania blokad doradczych zwracają wartość `NULL`, jeśli argumentem jest `NULL`.

- **GET_LOCK(*str*, *timeout*)**

Funkcja podejmuje próbę nałożenia blokady na nazwę wskazywaną przez ciąg tekstowy `str` w czasie wskazanym przez `timeout` (wyrażonym w sekundach). Funkcja `GET_LOCK()` zwraca wartość 1, jeśli nałożenie blokady zakończyło się powodzeniem we wskazanym czasie, wartość 0, jeśli nałożenie blokady zakończyło się niepowodzeniem ze względu na minięcie wskazanego czasu, lub wartość `NULL` w przypadku wystąpienia błędu.

- **IS_FREE_LOCK(*str*)**

Funkcja sprawdza stan blokady na ciągu tekstowym `str`. Omawiana funkcja zwraca wartość 1, jeśli nazwa nie jest aktualnie zablokowana, a wartość 0, jeśli nazwa jest zablokowana. W przypadku wystąpienia błędu wartością zwrótną jest `NULL`.

- **IS_USED_LOCK(*str*)**

Funkcja zwraca identyfikator połączenia klienta, który nałożył blokadę na daną nazwę (`str`). Jeżeli nie ma wskazanej blokady, wartością zwrótną funkcji będzie `NULL`.

- **RELEASE_LOCK(*str*)**

Funkcja powoduje zwolnienie blokady nałożonej na `str`. Wartością zwrótną funkcji `RELEASE_LOCK()` jest 1, jeśli zwolnienie blokady zakończyło się powodzeniem, 0, jeśli blokada należy do innej sesji, lub `NULL`, jeśli wskazana blokada nie istnieje.

C.2.9. Funkcje związane z adresem IP

Funkcje omawiane w tym punkcie są związane z operacjami przeprowadzanymi na adresach IP, zarówno w wersji IPv4, jak i IPv6 (począwszy od MySQL 5.6.3).

- **INET_ATON(*str*)**

Mając dany adres IPv4 w postaci ciągu tekstowego w formacie z użyciem kropek, funkcja zwraca odpowiadającą adresowi liczbę całkowitą z zachowaniem kolejności bajtów. Jeżeli argumentem nie jest prawidłowy adres IPv4, wtedy wartością zwrótną jest `NULL`.

<code>INET_ATON('64.28.67.70')</code>	→ 1075594054
<code>INET_ATON('255.255.255.255')</code>	→ 4294967295
<code>INET_ATON('256.255.255.255')</code>	→ NULL
<code>INET_ATON('www.mysql.com')</code>	→ NULL

■ INET_NTOA(*n*)

Mając podany adres IPv4 w postaci liczby całkowitej, funkcja zwraca odpowiadający jej ciąg tekstowy adresu IPv4 zapisanego w formacie z użyciem kropek. Jeżeli argument jest nieprawidłowy, wtedy wartością zwrótną będzie NULL.

```
INET_NTOA(1075594054)      → '64.28.67.70'
INET_NTOA(2130706433)     → '127.0.0.1'
```

■ INET6_ATON(*expr*)

Mając dany adres IPv6 lub IPv4 w postaci ciągu tekstowego, funkcja zwraca odpowiadającą adresowi wartość liczbową z zachowaniem kolejności bajów podaną jako binarny ciąg tekstowy. Jeżeli argument jest nieprawidłowy, wtedy wartością zwrótną jest NULL. W przypadku wartości zwrótniej innej niż NULL jej typem jest VARBINARY(16) lub VARBINARY(4) dla adresów odpowiednio IPv6 i IPv4. Aby wynik skonwertować na postać czytelną do wyświetlenia, należy użyć funkcji HEX().

```
HEX(INET6_ATON('::1'))      → '00000000000000000000000000000001'
HEX(INET6_ATON('192.168.10.14')) → 'COA80A0E'
```

Funkcja INET6_ATON() została wprowadzona w MySQL 5.6.3.

■ INET6_NTOA(*expr*)

Mając podany adres IPv6 lub IPv4 w postaci liczbowej jako binarny ciąg tekstowy, funkcja zwraca odpowiadający jej ciąg tekstowy adresu. Jeżeli argument jest nieprawidłowy, wtedy wartością zwrótną będzie NULL.

```
INET6_NTOA(INET6_ATON('::1'))      → '::1'
INET6_NTOA(INET6_ATON('192.168.10.14')) → '192.168.10.14'
```

Funkcja INET6_NTOA() została wprowadzona w MySQL 5.6.3.

■ IS_IPV4(*expr*)

Jeżeli argumentem jest ciąg tekstowy przedstawiający prawidłowy adres IPv4, funkcja zwraca wartość 1. W przeciwnym razie wartością zwrótną jest 0.

```
IS_IPV4('::1')      → 0
IS_IPV4('192.168.10.14') → 1
```

Funkcja IS_IPV4() została wprowadzona w MySQL 5.6.3.

■ IS_IPV4_COMPAT(*expr*)

IS_IPV4_MAPPED(*expr*)

Wymienione funkcje interpretują adresy IPv6 w postaci zwróconej przez funkcję INET6_ATON(), to znaczy przedstawione liczbowo jako binarny ciąg tekstowy. Funkcja określa, czy argument przedstawia odpowiednio prawidłowy ciąg tekstowy zgodny z IPv4 lub mapowany adres IPv4 na IPv6. W takich przypadkach wartością zwrótną funkcji jest 1, w przeciwnym razie funkcje zwracają 0. Adres zgodny z IPv4 lub mapowany IPv4 ma format odpowiednio ::ipv4_addr lub ::ffff:ipv4_addr.

```
IS_IPV4_COMPAT(INET6_ATON('::1'))      → 0
IS_IPV4_COMPAT(INET6_ATON('::192.168.10.14')) → 1
IS_IPV4_COMPAT(INET6_ATON('::ffff:192.168.10.14')) → 0
```

```
IS_IPV4_MAPPED(INET6_ATON('::1'))          → 0
IS_IPV4_MAPPED(INET6_ATON('::192.168.10.14')) → 0
IS_IPV4_MAPPED(INET6_ATON('::ffff:192.168.10.14')) → 1
```

Funkcje `IS_IPV4_COMPAT()` i `IS_IPV4_MAPPED()` zostały wprowadzone w MySQL 5.6.3.

■ `IS_IPV6(expr)`

Jeżeli argumentem jest ciąg tekstowy przedstawiający prawidłowy adres IPv6, funkcja zwraca wartość 1. W przeciwnym razie wartością zwrótną jest 0.

W przypadku adresu IPv4 funkcja również zwraca wartość 0.

```
IS_IPV6('::1')          → 1
IS_IPV6('192.168.10.14') → 0
```

Funkcja `IS_IPV6()` została wprowadzona w MySQL 5.6.3.

C.2.10. Funkcje XML

Funkcje przedstawione w tym punkcie pozwalają na przetwarzanie ciągu tekstowego przedstawiającego fragment XML za pomocą wyrażenia XPath w celu wyodrębnienia tekstu z wspomnianego fragmentu lub zwrócenia fragmentu wraz z dopasowanym elementem zastąpionym przez inny ciąg tekstowy. Argumenty ciągu tekstowego muszą zawierać prawidłowo zrównoważone i zagnieżdżone znaczniki.

Omawiane tutaj funkcje używają XPath 1.0. W celu uzyskania większej ilości informacji na temat XPath zapoznaj się z jego specyfikacją dostępną na stronie <http://www.w3.org/TR/xpath/>. Warto wspomnieć o pewnych ograniczeniach w obsłudze XPath; znajdziesz je wymienione w podręczniku użytkownika MySQL.

■ `EXTRACTVALUE(xml_str, xpath_expr)`

Stosuje wyrażenie XPath w celu przetworzenia ciągu tekstowego XML i zwraca zawartość pierwszego węzła tekstowego z elementu dopasowanego przez wyrażenie. Jeżeli wyrażenie spowoduje dopasowanie wielu elementów, wynikiem będzie pierwszy węzeł tekstowy z każdego dopasowanego elementu. Poszczególne węzły tekstowe będą rozdzielone spacjami.

```
EXTRACTVALUE('<a><b>B</b><c></c></a>', '/b') → 'B'
EXTRACTVALUE('<a><b>B1</b><b>B2</b><b>B3</b></a>', '/b') → 'B1 B2 B3'
```

Jeżeli nie zostanie dopasowany żaden element, wynikiem będzie pusty ciąg tekstowy (tak samo jak w przypadku dopasowania elementu niezawierającego tekstu).

■ `UPDTEXML(xml_str, xpath_expr, xml_new)`

Stosuje wyrażenie XPath w celu przetworzenia ciągu tekstowego XML, zastępuje dopasowany element nowym `xml_new` i zwraca wynik. Jeżeli wyrażenie nie dopasuje żadnego elementu lub dopasuje więcej niż jeden, ciąg tekstowy XML zostanie zwrócony bez żadnych modyfikacji.

C.2.11. Funkcje przestrzenne

MySQL obsługuje funkcje operujące na wartościach przestrzennych. Tego rodzaju funkcji jest sporo, ale zaliczają się do różnych grup zawierających bardzo podobne do siebie funkcje. Na przykład, funkcje konwersji punktów, linii, wielokątów i innych typów przestrzennych są w dużym stopniu takie same. Z tego powodu oraz w celu oszczędności miejsca nie zostały tutaj omówione. Znajdziesz je w podręczniku użytkownika MySQL.

C.2.12. Różne funkcje

Funkcje omówione w tym punkcie ciężko zaliczyć do wymienionych wcześniej kategorii.

■ BENCHMARK(*n*, *expr*)

Wyrażenie *expr* zostanie obliczone *n* razy. Niezwykłość funkcji BENCHMARK() polega na tym, że została przeznaczona do wywołania w programie klienckim mysql. Jej wartością zwrótną zawsze jest 0, a tym samym nie ma praktycznego zastosowania. Interesującą wartością jest czas wykonania zapytania, wyświetlony przez mysql tuż za wynikiem wykonania zapytania:

```
mysql> SELECT BENCHMARK(1000000,PASSWORD('secret'));
+-----+
| BENCHMARK(1000000,PASSWORD('secret')) |
+-----+
| 0 |
+-----+
1 row in set (2.35 sec)
```

Wspomniany czas to tylko przybliżony wskaźnik, pokazujący szybkość, z jaką serwer może obliczyć wyrażenie. Wartość jest przybliżona, ponieważ pokazuje czas rzeczywisty, a nie procesora w serwerze. Na wspomniany czas rzeczywisty obliczenia wyrażenia wpływ ma wiele czynników, takich jak aktualne obciążenie serwera, stan serwera itd. Wyrażenie należy obliczyć wielokrotnie, aby uzyskać w miarę wiarygodny wynik.

■ BIT_COUNT(*n*)

Funkcja zwraca liczbę bitów ustawionych w argumentcie, który jest traktowany jako wartość BIGINT (64-bitowa liczba całkowita).

BIT_COUNT(0)	→ 0
BIT_COUNT(1)	→ 1
BIT_COUNT(2)	→ 1
BIT_COUNT(7)	→ 3
BIT_COUNT(-1)	→ 64
BIT_COUNT(NULL)	→ NULL

■ BIT_LENGTH(*str*)

Funkcja zwraca wyrażoną w bitach długość ciągu tekstowego *str*. Jeżeli argument ma wartość NULL, wtedy wartością zwrótną funkcji również będzie NULL.

BIT_LENGTH('abc')	→ 24
BIT_LENGTH('a long string')	→ 104
BIT_LENGTH(CONVERT('abc' USING ucs2))	→ 48

■ CONNECTION_ID()

Funkcja zwraca identyfikator połączenia, który serwer powiązał z bieżącą sesją klienta. Każdy klient ma unikalny identyfikator odróżniający go od pozostałych połączonych klientów.

CONNECTION_ID() → 10146

■ CURRENT_USER()

Serwer MySQL uwierzytelnia każdego klienta względem pewnego rekordu tabeli mysql.user. Funkcja CURRENT_USER() zwraca wartości pochodzące z kolumn User i Host wspomnianego rekordu dla bieżącej sesji klienta. Wartość zwrótna jest ciągiem tekstowym utf8 w formacie 'nazwa_użytkownika@nazwa_komputera'. Nawiasy są opcjonalne.

CURRENT_USER() → 'sampadm@localhost'
SUBSTRING_INDEX(CURRENT_USER(), '@', 1) → 'sampadm'

Funkcję CURRENT_USER() można wykorzystać do sprawdzenia, za jakiego użytkownika jesteś uznawany przez serwer. Jeśli serwer uwierzytelnia Cię względem innego konta użytkownika, to może być zupełnie inny użytkownik niż wskazany podczas nawiązywania połączenia. W szczególności, jeśli serwer uwierzytelnia Cię jako użytkownika anonimowego, część wskazująca nazwę użytkownika w wartości zwrótniej będzie pusta, natomiast ta sama część zwracana przez funkcję USER() zawiera nazwę użytkownika podaną podczas nawiązywania połączenia.

W ramach widoku lub programu składowanego funkcja CURRENT_USER() domyślnie zwraca nazwę konta użytkownika odpowiadającą atrybutowi DEFINER obiektu. Dla widoków i procedur składowanych zdefiniowanych za pomocą SQL SECURITY INVOKER funkcja CURRENT_USER() zwraca konto użytkownika wywołującego dany widok lub procedurę składowaną.

■ DATABASE()

Funkcja zwraca ciąg tekstowy utf8 zawierający nazwę domyślnej bazy danych lub NULL, jeśli nie ma domyślnej bazy danych. Jeżeli funkcja zostanie wywołana z poziomu procedury składowanej, wartością zwrótną będzie baza danych powiązana z daną procedurą.

DATABASE() → 'sampdb'

■ FOUND_ROWS()

Funkcja zwraca liczbę rekordów, które poprzednio wykonane zapytanie SELECT zwróciłoby bez klauzuli LIMIT. Na przykład, poniższe zapytanie zwraca maksymalnie 10 rekordów:

```
mysql> SELECT * FROM mytbl LIMIT 10;
```

W celu ustalenia liczby rekordów zwracanych przez zapytanie bez klauzuli LIMIT należy zastosować poniższe rozwiązanie:

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM mytbl LIMIT 10;
mysql> SELECT FOUND_ROWS();
```

■ **DEFAULT(*col_name*)**

Zapytanie INSERT pozwala na użycie słowa kluczowego DEFAULT w celu wstawienia wartości domyślnej kolumny w nowym rekordzie. Jednak to słowo kluczowe nie jest dozwolone w każdym wyrażeniu lub w innych kontekstach. Na przykład, nie można go wykorzystać w celu wyzerowania kolumny do jej wartości domyślnej w zapytaniu UPDATE. Do takich zadań służy funkcja DEFAULT(). Mając podaną nazwę kolumny, funkcja zwraca wartość domyślną kolumny lub generuje komunikat błędu, jeśli kolumna nie ma wartości domyślnej.

```
UPDATE counts SET counter = DEFAULT(counter)
WHERE max_time > expire_time;
```

■ **LAST_INSERT_ID()**

LAST_INSERT_ID(*expr*)

W przypadku wywołania bez argumentu funkcja zwraca wartość AUTO_INCREMENT wygenerowaną z powodzeniem przez ostatnie zapytanie INSERT lub 0, jeśli wspomniana wartość nie została wygenerowana. Jeżeli zapytanie wstawiło wiele rekordów, funkcja LAST_INSERT_ID() zwraca wartość pierwszego z nich. W przypadku wystąpienia błędu wartość LAST_INSERT_ID() pozostaje niezdefiniowana.

Wywołanie funkcji z argumentem powoduje zwrot wartości argumentu, choć ta wartość jest traktowana, jakby została wygenerowana automatycznie, co jest użyteczne podczas generowania sekwencji.

Więcej informacji na ten temat znajdziesz w podrozdziale 3.4, zatytułowanym „Praca z sekwencjami”. Dla obu form funkcji LAST_INSERT_ID() serwer przechowuje wartość w poszczególnych sesjach. Wartość nie może być zmieniona przez inne klienty, nawet te, które powodują automatyczne wygenerowanie nowych wartości.

■ **LOAD_FILE(*file_name*)**

Funkcja odczytuje plik o nazwie *file_name* i zwraca jego zawartość w postaci ciągu tekstowego. Jeżeli nazwa pliku jest dosłownym ciągiem tekstowym, będzie zinterpretowana w kodowaniu znaków wskazanym przez zmienną `character_set_filesystem`. Plik musi znajdować się w komputerze serwera, musi być podany jako bezwzględna (pełna) ścieżka dostępu i musi być dostępny do odczytu przez wszystkich — to rodzaj zabezpieczenia gwarantującego, że nie następuje próba odczytu pliku chronionego. Jeżeli zmienna systemowa `secure_file_priv` ma ustawioną wartość, wspomniana wartość powinna być katalogiem, a plik musi znajdować się w tym katalogu. Ponieważ plik musi znajdować się w komputerze serwera, konieczne jest posiadanie uprawnienia FILE. Jeżeli nie uda się spełnić któregośkolwiek z wymienionych warunków, wartością zwrótną funkcji LOAD_FILE() będzie NULL.

■ **MASTER_POS_WAIT(*log_file*, *pos* [,*timeout*])**

Ta funkcja jest używana do testowania replikacji. Jej wywołanie w serwerze podległym powoduje zablokowanie serwera aż do chwili odczytania i przetworzenia przez serwer podległy wszystkich zdarzeń z serwera głównego replikacji aż do współrzędnych wskazanych przez argumenty *log_file* i *pos*. Wartość *timeout*, o ile zostanie podana, powoduje określenie liczby sekund, przez które funkcja MASTER_POS_WAIT() powinna czekać. Wartość 0 lub mniejsza jest odpowiednikiem braku wspomnianego czasu oczekiwania.

Wartością zwrótną funkcji MASTER_POS_WAIT() jest liczba zdarzeń, na które musi czekać, zanim osiągnie wskazane współrzędne replikacji. Jeżeli serwer podrzędny osiągnął wymienione współrzędne, działanie funkcji zostaje natychmiast zakończone wraz z wartością zwrótną 0. Wartość zwrótna -1 oznacza upływanie wskazanego czasu, wystąpienie błędu lub niezainicjalizowane informacje z serwera głównego replikacji. Z kolei wartość zwrótna NULL oznacza, że wątek SQL serwera podległego nie działa lub był zatrzymany w chwili, gdy funkcja oczekiwała na zdarzenia.

■ **NAME_CONST(*name*, *value*)**

Ta funkcja jest używana wewnętrznie (na przykład w celu zapisu zapytań w binarnym dzienniku zdarzeń). Zwraca wartość (*value*) wraz z nazwą kolumny (*name*). Oba argumenty muszą być stałymi.

■ **ROW_COUNT()**

Ta funkcja przypomina mysql_affected_rows() z API C. Wartość zwrótna zależy od typu poprzedniego zapytania:

- ◆ W przypadku innych niż SELECT zapytań operujących na danych wartością zwrótną będzie liczba rekordów, których dotyczyło zapytanie (wstawione, usunięte lub uaktualnione).
- ◆ W przypadku zapytań SELECT wartość zwrótna -1 wskazuje, że zapytanie zwróciło zbiór wynikowy. W przeciwnym razie zwracana jest liczba rekordów pobranych do miejsca przeznaczenia. Na przykład, SELECT ... INTO var_name zwraca 1, natomiast SELECT ... INTO file_name zwraca liczbę rekordów zapisanych w pliku.
- ◆ W przypadku zapytań definicji danych (na przykład CREATE TABLE) wartością zwrótną jest 0.

Jeżeli zapytanie wygeneruje błąd, funkcja ROW_COUNT() zwróci wartość -1.

■ **SCHEMA()**

To jest synonim funkcji DATABASE().

■ **SESSION_USER()**

To jest synonim funkcji USER().

■ **SLEEP(*seconds*)**

Funkcja wstrzymuje działanie na podaną liczbę sekund i zwraca wartość 0 lub 1 w przypadku zakłóceń. Argument *seconds* może posiadać część ułamkową.

■ **SYSTEM_USER()**

To jest synonim funkcji `USER()`.

■ **USER()**

Funkcja zwraca ciąg tekstowy utf8 zawierający nazwę użytkownika podaną przez klienta podczas nawiązywania połączenia z serwerem MySQL oraz komputer, z którego klient nawiązał połączenie. Wartością zwrótną jest ciąg tekstowy w formacie `'nazwa_uzytkownika@nazwa_komputera'`.

<code>USER()</code>	→ <code>'paul@localhost'</code>
<code>SUBSTRING_INDEX(USER(), '@', 1)</code>	→ <code>'paul'</code>
<code>SUBSTRING_INDEX(USER(), '@', -1)</code>	→ <code>'localhost'</code>

■ **UUID()**

Funkcja zwraca uniwersalny, unikatowy identyfikator. Wartość zwrótna z jednego wywołania funkcji `UUID()` powinna być inna od wartości zwrótniej kolejnego wywołania. Nie ma gwarancji zachowania unikalności wartości zwrótniej funkcji, ale wystąpienie duplikatów jest mało prawdopodobne.

`UUID()` → `'4550868e-3c1f-1027-9cc8-78fa7f8d46b6'`

Wartością zwrótną jest składający się z pięciu części ciąg tekstowy utf8 cyfr szesnastkowych wygenerowany na podstawie 128-bitowej liczby. Pierwsze cztery części powinny być unikalne pod względem czasu, natomiast ostatnia powinna być unikalna przestrzennie. Pierwsze trzy części wartości wywodzą się ze znacznika czasu. Czwarty element gwarantuje unikalność w sytuacjach, w których znacznik czasu sekwencji może nie być monotoniczny, co następuje na przykład w trakcie zmiany czasu. Piąta część to numer węzła IEEE 802. Może być wygenerowany na podstawie wartości uznawanej za unikalną dla komputera serwera, na przykład adresu interfejsu sieciowego. Jeżeli nie ma możliwości pobrania wspomnianej unikalnej wartości, wówczas stosowana jest 48-bitowa losowo wybrana liczba.

■ **UUID_SHORT()**

Funkcja działa podobnie do `UUID()`, ale uniwersalny identyfikator jest 64-bitową liczbą całkowitą bez znaku, a nie ciągiem tekstowym.

`UUID_SHORT()` → `94344395712626688`

W celu zapewnienia unikalności wartości muszą być spełnione trzy warunki.

Po pierwsze, wartość zmiennej systemowej `server_id` musi być unikalna wśród serwerów replikacji i mieć wartość z zakresu od 0 do 255. Po drugie, nie należy cofać czasu systemowego serwera między jego kolejnymi uruchomieniami.

Po trzecie, średnia częstotliwość wywołania `UUID_SHORT()` nie może przekroczyć 16 milionów razy na sekundę między ponownymi uruchomieniami serwera.

■ **VALUES(*col_name*)**

Ta funkcja jest przeznaczona do używania w zapytaniach `INSERT ... ON DUPLICATE KEY UPDATE`. W zapytaniu `UPDATE` klauzula `VALUES(col_name)` zwraca wartość przeznaczoną do wstawienia w wymienionej kolumnie. Poza

wymienionym kontekstem zwraca wartość NULL. Funkcja może być używana do tworzenia alternatywnych wartości przeznaczonych do wstawienia powiązanych z początkową wartością do wstawienia.

■ **VERSION()**

Funkcja zwraca ciąg tekstowy utf8 zawierający informacje o wersji serwera.

VERSION() → '5.5.28-log'

Wartość składa się z numeru wersji oraz prawdopodobnie jednego lub więcej przyrostków, takich jak -log wskazujących włączoną rejestrację lub -debug wskazujących na uruchomienie serwera w trybie debug.

D

Przewodnik po zmiennych systemowych, stanu i użytkownika

W tym dodatku zostanie opisanych kilka typów zmiennych MySQL:

- Zmienne systemowe pozwalające na konfigurację serwera lub sprawdzenie jego bieżących ustawień.
- Zmienne stanu dostarczające informacje o bieżącym stanie serwera.
- Zmienne użytkownika, które można zdefiniować, przypisać im wartość oraz odwoływać się do nich w wyrażeniach.

Wartości zmiennych przedstawiających wielkość buforów lub długość są najczęściej podawane w bajtach. Jeśli jest inaczej, wyraźnie to zaznaczono.

O ile nie zaznaczono inaczej, wymienione tutaj zmienne są dostępne w wydaniu MySQL 5.5.0 lub nowszym. Wyraźnie zaznaczono zmienne wprowadzone w późniejszych wydaniach lub te, których znaczenie uległo zmianie od wymienionej wersji.

D.1. Zmienne systemowe

Zmienne systemowe dostarczają informacje o konfiguracji i możliwościach serwera. Większość zmiennych systemowych może być ustawiona w trakcie uruchamiania serwera. W trakcie działania serwera każda zmienna systemowa ma wartość globalną, sesji lub obie. Wiele zmiennych systemowych jest dynamicznych, czyli mogą być modyfikowane w trakcie działania serwera. Tego rodzaju informacje są podawane w opisie poszczególnych zmiennych, w tym samym wierszu, w którym znajduje się nazwa zmiennej:

- W przypadku zmiennych, które mogą być ustawione w trakcie uruchamiania serwera, pojawia się słowo „startowa”, a następnie „ustawiana bezpośrednio” lub „opcja”. Zwrot „ustawiana bezpośrednio” oznacza możliwość bezpośredniego ustawienia zmiennej w wierszu poleceń lub pliku opcji przy użyciu opcji o takiej samej nazwie jak nazwa zmiennej. (W podpunkcie F.2.1.2, zatytułowanym „Ustawianie zmiennych programu”, omówiono odpowiednią składnię). W przeciwnym razie po słowie „startowa” podano opcję używaną do ustawienia zmiennej. Na przykład, zmienną `time_zone` ustawia się za pomocą opcji `--default-time-zone`. Kiedy opcja zostanie podana, jej znaczenie znajdziesz w opisie programu `mysql` w dodatku F, zatytułowanym „Przewodnik po programie MySQL”.
- W celu wskazania zakresu zmiennej użyto słowa „zakres”, a następnie „globalny” lub „sesji”, wskazując tym samym zakres zmiennej: `GLOBAL`, `SESSION` lub oba. Zmienna, którą w trakcie działania serwera można tylko odczytać, ma jedynie wartość globalną.
- W przypadku zmiennych, które mogą być modyfikowane w trakcie działania serwera, pojawia się słowo „dynamiczna”.

Aby wyświetlić zmienne systemowe, należy wykonać zapytanie `SHOW VARIABLES` lub wydać polecenie `mysqladmin variables`. Istnieje także możliwość przeanalizowania tabel bazy danych `INFORMATION_SCHEMA` o nazwach `GLOBAL_VARIABLES` i `SESSION_VARIABLES`; znajdziesz w nich informacje o zmiennych systemowych. W celu wyświetlenia wartości poszczególnych zmiennych należy wykonać zapytanie `SELECT @@GLOBAL.nazwa_zmiennej` dla zmiennych globalnych lub `SELECT@@SESSION.nazwa_zmiennej` bądź `SELECT@@nazwa_zmiennej` dla zmiennych sesji.

Do ustawienia zmiennej systemowej służy zapytanie `SET`. Ustawienie zmiennej globalnej wymaga uprawnień `SUPER`. Z kolei ustawienie zmiennych sesji zwykle nie wymaga specjalnych uprawnień. Od tej reguły istnieje kilka wyjątków, o których wspomniano.

Nazwy zmiennych systemowych nie rozróżniają wielkości liter.

Więcej informacji na temat ustawiania zmiennych systemowych w trakcie działania serwera lub analizy ich wartości znajdziesz w punkcie 12.3.1, zatytułowanym „Sprawdzanie i ustawienie wartości zmiennych systemowych”.

Pewne omówione tutaj zmienne są dostępne jedynie w określonych konfiguracjach na pewnych platformach. Na przykład, zmienna `debug` jest dostępna jedynie wtedy, gdy serwer został skompilowany wraz z obsługą debugowania. Z kolei zmienna `named_pipe` jest dostępna jedynie w Windows.

Poniższa lista zawiera opis ogólnych zmiennych systemowych. Oddzielna sekcja będzie poświęcona zmiennym silnika bazy danych InnoDB.

- `auto_increment_increment` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Liczba, o jaką wartość `AUTO_INCREMENT` będzie zwiększana za każdym razem, gdy serwer generuje nową wartość sekwencji. Zakres wartości wynosi od 1 do 65535. Wartością domyślną jest 1.

- `auto_increment_offset` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Wartość początkowa dla sekwencji `AUTO_INCREMENT`. Zakres wartości wynosi od 1 do 65535. Wartością domyślną jest 1.

- `autocommit` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Tryb automatycznego zatwierdzania podczas przetwarzania transakcji. Wartość domyślna wynosi 1, czyli automatyczne zatwierdzanie jest włączone i zapytania są wykonywane natychmiast. W zasadzie oznacza to, że każde zapytanie jest oddzielną transakcją. Ustawienie wartości 0 powoduje wyłączenie automatycznego zatwierdzania i kolejne zapytania nie będą miały efektu w bazie danych aż do chwili przeprowadzenia zatwierdzenia (za pomocą zapytania `COMMIT` lub przez ustawienie wartości 1 zmiennej `autocommit`). Skutek zapytań wykonanych w ramach transakcji można wycofać za pomocą zapytania `ROLLBACK`, o ile transakcja nie została jeszcze zatwierdzona. Ustawienie wartości 1 zmiennej `autocommit` powoduje ponowne włączenie automatycznego zatwierdzania (i pośrednio zatwierdzenie wszystkich oczekujących transakcji).

- `automatic_sp_privileges` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Jeżeli wartość tej zmiennej wynosi 1 (to jest wartość domyślna), wtedy podczas tworzenia procedury składowanej serwer automatycznie nadaje uprawnienia `EXECUTE` i `ALTER ROUTINE` (o ile konieczne), aby później można było wykonywać, modyfikować lub usunąć daną procedurę. Po usunięciu procedury składowanej serwer automatycznie odbiera wymienione uprawnienia. Jeżeli wartością zmiennej `automatic_sp_privileges` jest 0, automatyczne nadawanie i odbieranie uprawnień nie występuje.

- `back_log` (startowa: ustawiana bezpośrednio; zakres: globalny)

Zmienna wskazuje maksymalną liczbę oczekujących żądań połączenia, które mogą być kolejgowane w trakcie przetwarzania bieżących połączeń.

- `basedir` (startowa: ustawiana bezpośrednio; zakres: globalny)

Ścieżka dostępu do katalogu głównego instalacji MySQL. Wiele innych ścieżek dostępu jest podawanych względem wymienionego katalogu, o ile nie są bezwzględными ścieżkami dostępu.

- `big_tables` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Ustawienie wartości 1 tej zmiennej powoduje, że ogromne zbiory wynikowe będą przetwarzane przez zapis całego wyniku tymczasowego na dysku zamiast jego przechowywania w pamięci. Oznacza to zmniejszenie wydajności, ale zapytania `SELECT` wymagające użycia ogromnych tabel tymczasowych nie będą generowały błędów informujących o zapelnieniu tabeli. Wartość domyślna zmiennej wynosi 0 (przechowywanie tabel tymczasowych w pamięci).

Normalnie nie musisz ustawiać tej zmiennej, ponieważ serwer automatycznie zapisuje wynik na dysku, jeśli zachodzi potrzeba.

- **bind_address** (startowa: ustawiana bezpośrednio; zakres: globalny)
Adres IP, którego serwer będzie używał w celu nasłuchiwania połączeń klientów przez TCP/IP. Począwszy od wydania MySQL 5.6.6, wartość domyślna to * (nasłuchiwanie na wszystkich interfejsach IPv4 i IPv6), natomiast we wcześniejszych wydaniach to 0.0.0.0 (nasłuchiwanie na wszystkich interfejsach IPv4). Więcej informacji znajdziesz w punkcie 12.2.4, zatytułowanym „W jaki sposób serwer nasłuchuje połączeń?”. Ta zmienna została wprowadzona w MySQL 5.6.1.
- **binlog_cache_size** (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Wielkość bufora używanego do przechowywania zapytań SQL będących częścią transakcji, zanim nie zostaną zapisane w binarnym dzienniku zdarzeń. (Wspomniany zapis następuje jedynie w chwili zatwierdzenia transakcji lub wykonywania zapytań uaktualniających tabele nietransakcyjne. Jeżeli nastąpi wycofanie transakcji wykorzystującej jedynie tabele transakcyjne, zapytania wykonane przez tę transakcję zostaną odrzucone).
- **binlog_checksum** (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Wartość zmiennej wskazuje, czy w binarnym dzienniku zdarzeń wraz ze zdarzeniami mają być zapisywane sumy kontrolne. Dozwolone wartości zmiennej to NONE (brak sum kontrolnych) i CRC32. Ta zmienna została wprowadzona w MySQL 5.6.2. Począwszy od wydania MySQL 5.6.6, jej wartością domyślną jest CRC32, we wcześniejszych NONE.
- **binlog_direct_non_transactional_updates** (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Transakcja przeprowadzająca uaktualnienia tabel zarówno transakcyjnych, jak i nietransakcyjnych może doprowadzić do powstania uaktualnień w systemie głównym różnych niż w systemie podległym, ponieważ uaktualnienia zapytań nietransakcyjnych staną się widoczne dla innych sesji, zanim pojawią się w binarnym dzienniku zdarzeń. Włączenie omawianej zmiennej (domyślnie jest wyłączona) powoduje, że uaktualnienia nietransakcyjne są natychmiast zapisywane w binarnym dzienniku zdarzeń, zamiast być buforowane aż do zatwierdzenia transakcji. Włączenie tej zmiennej ma wpływ jedynie na zapytania replikowane za pomocą rejestracji opartej na zapytaniach.
- **binlog_format** (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Format binarnego rejestrowania zdarzeń. Dostępne są następujące wartości dla tej zmiennej: STATEMENT (rejestracja oparta na zapytaniach), ROW (rejestracja oparta na rekordach) i MIXED (rejestracja mieszana). W przypadku opcji MIXED

serwer będzie automatycznie przełączał się między rejestracją opartą na zapytaniach i rekordach. Wartością domyślną jest STATEMENT. W trakcie działania serwera klient musi mieć uprawnienie SUPER, aby móc zmienić wartość tej zmiennej; dotyczy to nawet wartości sesji.

- `binlog_row_image` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

W przypadku rejestracji opartej na rekordach używane są „obrazy” przedstawiające stan tabel przed wstawieniem rekordów i po nim, co ma na celu opisanie rekordów początkowych i zmodyfikowanych. Zmienna `binlog_row_image` określa, ile kolumn mają wspomniane obrazy. Wartość FULL (to jest wartość domyślna) rejestruje wszystkie kolumny. Z kolei MINIMAL rejestruje jedynie kolumny wymagane do identyfikacji rekordów i zmodyfikowanych kolumn. Wartość NOBLOB działa jak FULL, ale nie obejmuje kolumn typów BLOB i TEXT. Ta zmienna została wprowadzona w MySQL 5.6.2.

- `binlog_rows_query_log_events` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

W przypadku rejestracji opartej na rekordach włączenie tej zmiennej powoduje umieszczenie w binarnym dzienniku zdarzeń informacji użytecznych podczas debugowania. Ta zmienna została wprowadzona w MySQL 5.6.2.

- `binlog_stmt_cache_size` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Wielkość bufora dla zapytań nietransakcyjnych wykonywanych w transakcji. Wartość domyślna wynosi 32 K. Ta zmienna została wprowadzona w MySQL 5.5.9.

- `bulk_insert_buffer_size` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Wielkość bufora używanego w celu pomocy podczas optymalizacji masowej operacji wstawiania danych do tabel MyISAM. Obejmuje zapytania LOAD DATA, wstawiające wiele rekordów zapytania INSERT oraz zapytania INSERT INTO ... SELECT. Wartość 0 zmiennej powoduje wyłączenie optymalizacji.

- `character_set_client` (zakres: globalny, sesji; dynamiczna)

Kodowanie znaków w zapytaniach wysyłanych serwerowi przez klienta.

- `character_set_connection` (zakres: globalny, sesji; dynamiczna)

Kodowanie znaków w połączeniu klient-serwer. To kodowanie znaków jest używane do interpretacji dosłownych ciągów tekstowych (za wyjątkiem rozpoczynających się od introducera) oraz jako kodowanie znaków dla ciągów tekstowych powstałych na skutek konwersji liczb na postać ciągów tekstowych.

- `character_set_database` (zakres: globalny, sesji; dynamiczna)

Kodowanie znaków domyślnej bazy danych, o ile taka istnieje. Jeżeli nie ma domyślnej bazy danych (na przykład klient nawiązuje połączenie i nie wybiera bazy danych),

wartość tej zmiennej będzie taka sama jak zmiennej `character_set_server`. Serwer powoduje ustawienie wartości zmiennej `character_set_database` za każdym razem, gdy wybierana jest inna baza danych.

- `character_set_filesystem` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Kodowanie znaków systemu plików. Jest ono wykorzystywane podczas analizy dosłownych ciągów tekstowych wskazujących nazwy plików, na przykład pliku danych w zapytaniach `LOAD DATA`. Przed uzyskaniem dostępu do pliku serwer konwertuje nazwę pliku z kodowania znaków wymienionego w zmiennej `character_set_client` na wskazane przez zmienną `character_set_filesystem`. Wartość domyślna zmiennej wynosi binary (brak konwersji).
- `character_set_results` (zakres: globalny, sesji; dynamiczna)
Kodowanie znaków wyników zapytania wysyłanych klientowi przez serwer.
- `character_set_server` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Domyślne kodowanie znaków serwera.
- `character_set_system` (zakres: globalny)
Systemowe kodowanie znaków. Wartością tej zmiennej zawsze jest `utf8`. To jest kodowanie znaków używane dla metadanych takich jak nazwy baz danych, tabel i kolumn. Jest stosowane także w funkcjach takich jak `DATABASE()`, `CURRENT_USER()`, `USER()` i `VERSION()`.
- `character_sets_dir` (startowa: ustawiana bezpośrednio; zakres: globalny)
Katalog, w którym znajdują się pliki opisujące kodowania znaków.
- `collation_connection` (zakres: globalny, sesji; dynamiczna)
Kolejność sortowania w kodowaniu znaków połączenia.
- `collation_database` (zakres: globalny, sesji; dynamiczna)
Kolejność sortowania w kodowaniu znaków bazy danych, o ile takie zostało ustawione. W przypadku braku domyślnej bazy danych (na przykład klient nawiązuje połączenie, ale nie wybiera bazy danych) wartość tej zmiennej będzie taka sama jak zmiennej `collation_server`. Serwer ustawia wartość `collation_database` za każdym razem, gdy wybrana zostanie inna baza danych.
- `collation_server` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Domyślna kolejność sortowania dla domyślnego kodowania znaków serwera.
- `completion_type` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Typ zakończenia transakcji. Wartością domyślną jest 0 (`NO_CHAIN`), nie zmienia ona zachowania zapytań `COMMIT` i `ROLLBACK`. Wartość 1 (`CHAIN`) powoduje, że wymienione zapytania są odpowiednikami `COMMIT AND CHAIN` i `ROLLBACK AND CHAIN`.

Z kolei wartość 2 (RELEASE) powoduje, że wymienione zapytania są odpowiednikami COMMIT RELEASE i ROLLBACK RELEASE. W przypadku AND CHAIN, gdy transakcja zostanie zakończona, serwer rozpoczyna nową na takim samym poziomie izolacji. Z kolei w przypadku AND RELEASE po zakończeniu transakcji serwer kończy sesję.

- **concurrent_insert** (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

W przypadku tabeli MyISAM nieposiadających „dziur” w środku pliku danych serwer może zezwalać na przeprowadzanie jednoczesnych operacji: wstawiania, polegających na dodawaniu nowych rekordów na końcu tabeli, i pobierania istniejących rekordów. Omawiana zmienna określa, czy serwer zezwala na operacje wstawiania przeprowadzane jednocześnie z operacjami odczytu danych.

Wartość 0 (lub NEVER) wyłącza tę możliwość, wartość 1 (lub AUTO) ją włącza. Z kolei wartość 2 (lub ALWAYS) włącza jednoczesne operacje wstawiania nawet dla tabel MyISAM posiadających „dziury” w pliku danych. Nowe rekordy zostają dodane na końcu używanej tabeli, jeśli tabela nie jest aktualnie w użyciu, wtedy nowe rekordy są umieszczane w „dziurach”. Wartością domyślną jest 1.

- **connect_timeout** (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Liczba sekund, przez które serwer mysqld czeka na pakiety w trakcie początkowej fazy operacji nawiązywania połączenia. Wartość domyślna wynosi 10.

- **core_file** (startowa: ustawiana bezpośrednio; zakres: globalny)

Zmienna określa, czy serwer generuje plik core, zanim zakończy działanie na skutek wystąpienia błędu o znaczeniu krytycznym. Ta zmienna została wprowadzona w MySQL 5.6.2.

- **datadir** (startowa: ustawiana bezpośrednio; zakres: globalny)

Ścieżka dostępu prowadząca do katalogu danych MySQL.

- **date_format** (zakres: globalny)

Ta zmienna jest nieużywana.

- **datetime_format** (zakres: globalny)

Ta zmienna jest nieużywana.

- **debug** (zakres: globalny, sesji; dynamiczna)

Rodzaj danych wejściowych w trakcie debugowania. Zapoznaj się z opisem opcji --debug w punkcie F.2.1, zatytułowanym „Opcje standardowe programu MySQL”. Ta zmienna jest dostępna tylko wtedy, gdy serwer został skompilowany wraz z obsługą debugowania.

- `default_storage_engine` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Domyślny silnik bazy danych dla tabel tworzonych bez opcji `ENGINE=nazwa_silnika` lub tych, dla których podano nieobsługiwaną wartość `nazwa_silnika`. W nazwie silnika wielkość liter nie ma znaczenia. Domyślnie to jest silnik InnoDB.

- `default_tmp_storage_engine` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Działanie tej zmiennej jest podobne do zmiennej `default_storage_engine`, ale dotyczy tabel tworzonych za pomocą zapytania `CREATE TEMPORARY TABLE`. Ta zmienna została wprowadzona w MySQL 5.6.3.

- `default_week_format` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Domyślny typ tygodnia dla funkcji `WEEK()` i `YEARWEEK()` wywoływanych bez opcjonalnego atrybutu `mode`.

- `delay_key_write` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Ta zmienna określa, czy serwer respektuje opóźniony zapis kluczy dla tabel MyISAM utworzonych wraz z opcją `DELAY_KEY_WRITE`. Zmienna może przyjąć jedną z trzech wartości:

- ◆ `ON` (wartość domyślna) oznacza, że serwer honoruje opcję `DELAY_KEY_WRITE` dla tabel zdefiniowanych z wymienioną opcją. Zapis kluczy będzie opóźniony dla tabel zdefiniowanych wraz z opcją `DELAY_KEY_WRITE=1`, ale nie dla tabel z opcją `DELAY_KEY_WRITE=0`.
- ◆ `OFF` oznacza, że zapis kluczy nigdy nie będzie opóźniony, niezależnie od sposobu zdefiniowania tabeli.
- ◆ `ALL` wymusza, aby zapis kluczy zawsze był opóźniony, niezależnie od sposobu zdefiniowania tabeli.

Bardzo często zdarza się, że w serwerach podległych replikacji zmienna `delay_key_write` ma ustawioną wartość `ALL`, co ma na celu zwiększenie wydajności dla tabel MyISAM na skutek opóźnionego zapisu kluczy niezależnie od sposobu początkowego utworzenia tabeli.

- `delayed_insert_limit` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Liczba rekordów z zapytań `INSERT DELAYED`, które procedura obsługi opóźnionego wstawiania rekordów wstawi do tabeli, zanim sprawdzi, czy nie pojawiły się jakiegokolwiek nowe zapytania `SELECT` do tej tabeli. Jeżeli pojawiły się nowe zapytania `SELECT`, wtedy procedura wstrzyma operację wstawiania danych, aby umożliwić wykonanie operacji pobierania danych.

- `delayed_insert_timeout` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Kiedy procedura dla operacji `INSERT DELAYED` zakończy wstawianie kolejkowanych rekordów, odczekuje podaną przez tę zmienną liczbę sekund, zanim sprawdzi, czy pojawiły się nowe zapytania `INSERT DELAYED`. Jeżeli nowe zapytania pojawiły się, zostaną wówczas wykonane. Jeśli nie ma nowych zapytań, działanie procedury zostanie zakończone.
- `delayed_queue_size` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Liczba rekordów, które mogą być kolejgowane dla zapytań `INSERT DELAYED` wykonywanych w danej tabeli. Jeżeli kolejka będzie pełna, dalsze zapytania `INSERT DELAYED` dla tabeli zostaną wstrzymane (zablokowane) aż do chwili, gdy w kolejce znajdzie się dla nich miejsce.
- `div_precision_increment` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
W przypadku dzielenia dwóch dokładnych liczb za pomocą operatora `/` ta zmiana określa liczbę cyfr skali, które powinny zostać dodane. Na przykład, operacja `.1/.7` daje wynik `.14286` lub `.1428571` dla wartości zmiennej `div_precision_increment` odpowiednio 4 i 6. Wartość może być z zakresu od 0 do 30, domyślnie to 4.
- `eq_range_index_dive_limit` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
W przypadku porównań równości przeprowadzanych za pomocą nieunikalnych indeksów, zmienna wskazuje liczbę zakresów, powyżej których optymalizator będzie dokonywał oszacowania rekordów na podstawie danych statystycznych indeksu zamiast indeksu. Wartość 0 oznacza wyłączenie użycia danych statystycznych. Wartością domyślną jest 10. Ta zmienna została wprowadzona w MySQL 5.6.5. W wersjach wcześniejszych niż 5.6.5 indeksy były używane niezależnie od liczby zakresów.
- `error_count` (zakres: sesji)
To jest zmienna tylko do odczytu o zasięgu sesji i wskazuje liczbę błędów wygenerowanych przez ostatnie zapytanie, które mogło wygenerować błędy.
- `event_scheduler` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Stan harmonogramu zdarzeń. Wartością zmiennej może być `OFF`, `ON` lub `DISABLED`. Jeżeli podczas uruchamiania serwera stan harmonogramu zdarzeń jest ustawiony jako `DISABLED`, tego stanu nie można już zmienić w trakcie działania serwera. Z kolei wartość `ON` lub `OFF` zmiennej `event_scheduler` może być zmieniona w trakcie działania serwera. Wartością domyślną jest `OFF`.

- `expire_logs_days` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Jeżeli wartość tej zmiennej jest większa niż domyślne 0, serwer automatycznie usuwa pliki binarnego dziennika zdarzeń starsze niż wskazana liczba dni i uaktualnia plik indeksu binarnego dziennika zdarzeń. Wartość tej zmiennej jest sprawdzana podczas uruchamiania serwera i w trakcie otwierania nowego pliku binarnego dziennika zdarzeń.

- `external_user` (zakres: sesji)

To jest zmienna tylko do odczytu o zasięgu sesji, która może być ustawiona przez wtyczkę uwierzytelniania używaną do uwierzytelniania bieżącego użytkownika. Ta zmienna została wprowadzona w MySQL 5.5.7.

- `flush` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Zmienna określa, czy serwer zapisuje zawartość tabel MyISAM po każdym uaktualnieniu. To zmniejsza niebezpieczeństwo uszkodzenia tabeli na wypadek awarii, ale jednocześnie negatywnie wpływa na wydajność. Dlatego też zmienna jest użyteczna jedynie w przypadku niestabilnego systemu. Wartością domyślną jest OFF.

- `flush_time` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Jeżeli wartość tej zmiennej jest niezerowa, serwer co `flush_time` sekund zamyka tabele i zapisuje na dysku wszystkie buforowane zmiany. Jeżeli system jest niestabilny i ma tendencję do częstego zawieszania się i ponownego uruchamiania, wymuszenie w ten sposób zapisu na dysku zmian wprowadzonych w tabelach zmniejsza wydajność, ale jednocześnie zmniejsza też ryzyko uszkodzenia tabeli lub utraty danych. Wartość domyślna wynosi 0 dla systemów UNIX i 1800 (30 minut) dla Windows.

- `foreign_key_checks` (zakres: globalny, sesji; dynamiczna)

Ustawienie tej zmiennej wartości 0 lub 1 powoduje odpowiednio wyłączenie lub włączenie sprawdzania klucza zewnętrznego w tabelach InnoDB. Domyślnie jest włączone sprawdzanie klucza zewnętrznego. Wyłączenie tego sprawdzenia może być użyteczne na przykład w celu przywrócenia danych z pliku, w którym polecenia tworzą i wczytują tabele w kolejności innej niż wymagana przez relacje klucza zewnętrznego. Po wczytaniu tego rodzaju danych można z powrotem włączyć sprawdzanie klucza zewnętrznego.

- `ft_boolean_syntax` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Lista operatorów obsługiwanych przez wyszukiwanie pełnego tekstu w trybie IN BOOLEAN MODE.

- `ft_max_word_len` (startowa: ustawiana bezpośrednio; zakres: globalny)

Maksymalna długość słów, które mogą znajdować się w indeksach typu FULLTEXT. Dłuższe słowa będą ignorowane. Jeżeli zmienisz wartość tej zmiennej,

wtedy powinien również ponownie utworzyć indeksy typu FULLTEXT każdej tabeli MyISAM, dla której zostały zdefiniowane. Wartość domyślna wynosi 84.

- **ft_min_word_len** (startowa: ustawiana bezpośrednio; zakres: globalny)
Minimalna długość słów, które mogą znajdować się w indeksach typu FULLTEXT. Krótsze słowa będą ignorowane. Jeżeli zmienisz wartość tej zmiennej, wtedy powinien również ponownie utworzyć indeksy typu FULLTEXT każdej tabeli MyISAM, dla której zostały zdefiniowane. Wartość domyślna wynosi 4.
- **ft_query_expansion_limit** (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna jest używana w wyszukiwaniu pełnego tekstu przeprowadzanym za pomocą klauzuli WITH QUERY EXPANSION. Określa liczbę „najtrafniejszych dopasowań”, które będą używane w drugiej fazie operacji wyszukiwania.
- **ft_stopword_file** (startowa: ustawiana bezpośrednio; zakres: globalny)
Zmienna wskazuje plik słów przestankowych dla indeksów typu FULLTEXT. Domyślnie używana jest wbudowana lista tego rodzaju słów. Aby wyłączyć słowa przestankowe, należy zmiennej przypisać pusty ciąg tekstowy. Jeżeli zmienisz wartość tej zmiennej, wtedy powinien również ponownie utworzyć indeksy typu FULLTEXT każdej tabeli MyISAM, dla której zostały zdefiniowane.
- **general_log** (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Zmienna określa, czy włączony jest ogólny dziennik zapytań. Jeżeli tak, jego położenie wskazuje zmienna log_output.
- **general_log_file** (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Zmienna zawiera nazwę pliku ogólnego dziennika zapytań, jeśli jako standardowe wyjście dla dziennika podano FILE. Wartość domyślna to umieszczany w katalogu danych MySQL plik o nazwie *HOSTNAME.log*, gdzie *HOSTNAME* oznacza nazwę komputera, w którym działa serwer. Jeżeli nazwa pliku zostanie podana w postaci względnej ścieżki dostępu, serwer będzie ją interpretował względem katalogu danych MySQL.
- **group_concat_max_len** (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Zmienna wskazuje maksymalną długość wartości, które powinny być zwracane przez funkcję GROUP_CONCAT(). Wartość domyślna to 1024.
- **have_compress** (zakres: globalny)
Zmienna wskazuje, czy biblioteka kompresji zlib jest dostępna. Serwer potrzebuje tej biblioteki do implementacji funkcji COMPRESS() i UNCOMPRESS(). Jeżeli biblioteka jest niedostępna, wymienionych funkcji nie będzie można używać.

- **have_crypt** (zakres: globalny)

Zmienna wskazuje, czy jest dostępne wywołanie systemowe `crypt()`, którego serwer potrzebuje do implementacji funkcji `CRYPT()`. Jeżeli wywołanie `crypt()` jest niedostępne, wymieniona funkcja nie będzie mogła być używana.

- **have_dynamic_loading** (zakres: globalny)

Wartość zmiennej określa, czy serwer obsługuje dynamiczne wczytywanie wtyczek.

- **have_geometry** (zakres: globalny)

Wartość zmiennej określa, czy mogą być używane przestrzenne typy danych.

- **have_openssl** (zakres: globalny)

Wartość zmiennej określa, czy serwer obsługuje zaszyfrowane połączenia z klientami za pomocą protokołu SSL. Zmienne `have_openssl` i `have_ssl` są synonimami.

- **have_query_cache** (zakres: globalny)

Wartość zmiennej określa, czy dostępny jest bufor zapytań.

- **have_rtree_keys** (zakres: globalny)

Wartość zmiennej określa, czy indeksy typu `Rtree` są dostępne dla indeksów `SPATIAL`.

- **have_ssl** (zakres: globalny)

Patrz opis zmiennej `have_openssl`.

- **have_symlink** (zakres: globalny)

Wartością tej zmiennej jest `DISABLED`, o ile serwer został uruchomiony wraz z opcją `--skip-symbolic-links`. W przeciwnym razie zmienna ma wartość `YES` lub `NO`, choć jej znaczenie zależy od platformy. W systemach `UNIX` zmienna wskazuje, czy dowiązania symboliczne są obsługiwane dla tabel `MyISAM`. Z kolei w `Windows` określa, czy obsługiwane są dowiązania symboliczne dla baz danych. Zapoznaj się z podrozdziałem 11.3, zatytułowanym „Przeniesienie zawartości katalogu danych”.

- **host_cache_size** (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Wskazuje wielkość bufora przeznaczonego do przechowywania informacji o próbach nawiązania połączenia przez klienta oraz wyniku tych prób. Komputery, z poziomu których nawiązanie połączenia często kończy się niepowodzeniem, są blokowane aż do chwili opróżnienia bufora. Zmiana wartości tej zmiennej w trakcie działania serwera powoduje opróżnienie bufora (taki sam efekt ma wykonanie zapytania `FLUSH HOSTS`). Wartość domyślna wynosi 128. Zmienna została wprowadzona w `MySQL 5.6.5`.

- **hostname** (zakres: globalny)
Zmienna przechowuje nazwę komputera serwera. Wartość tej zmiennej serwer MySQL sprawdza podczas jego uruchamiania.
- **identity** (zakres: globalny, sesji)
To jest synonim zmiennej sesji `last_insert_id`.
- **ignore_db_dirs** (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna jest ustawiana przez egzemplarze opcji `--ignore-db-dir` wskazujące podkatalogi w katalogu danych MySQL, które mają nie być traktowane jako bazy danych przez zapytanie `SHOW DATABASES` lub tabelę `INFORMATION_SCHEMA`. Wartość domyślna jest pusta. Ta zmienna została wprowadzona w MySQL 5.6.3.
- **init_connect** (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Niepusta wartość tej zmiennej wskazuje jedno lub więcej zapytań SQL rozdzielonych średnikami. Wspomniane zapytania będą wykonane przez każdego klienta nawiązującego połączenie z serwerem. Ta zmienna może być używana do modyfikacji początkowego środowiska sesji dla klienta. Zmienna `init_connect` jest ignorowana przez użytkowników z uprawnieniem `SUPER`. Ma to na celu uniknięcie wykonania nieprawidłowego lub niepożądanego zapytania, które mogłoby administratorom uniemożliwić nawiązanie połączenia z serwerem w celu usunięcia problemu.
- **init_file** (startowa: ustawiana bezpośrednio; zakres: globalny)
Niepusta wartość wskazuje nazwę pliku zawierającego zapytania SQL do wykonania przez serwer podczas jego uruchamiania. Jeżeli nazwa pliku zostanie podana w postaci względnej ścieżki dostępu, serwer zinterpretuje ją względem katalogu danych MySQL. Plik powinien zawierać po jednym zapytaniu w wierszu, a zapytania nie powinny na końcu zawierać średnika.
- **init_slave** (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
W przypadku serwera głównego replikacji, jeżeli ta zmienna będzie miała niepustą wartość, wtedy wskazuje zapytania do wykonania przez każdy serwer podległy po nawiązaniu przez nie połączenia. Wartością zmiennej powinno być jedno lub więcej zapytań SQL rozdzielonych średnikami.
- **innodb_XXX**
Patrz punkt D.1.1, zatytułowany „Zmienne systemowe InnoDB”.
- **insert_id** (zakres: sesji; dynamiczna)
Ustawienie tej zmiennej powoduje wskazanie wartości używanej przez kolejne zapytanie `INSERT` podczas wstawiania kolumny `AUTO_INCREMENT`. Zmienna jest używana do przetwarzania binarnego dziennika zdarzeń.

- `interactive_timeout` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Zmienna określa liczbę sekund braku aktywności ze strony klienta interaktywnego, zanim serwer uzna, że może zamknąć z nim połączenie. W przypadku klientów nieinteraktywnych używana jest wartość zmiennej `wait_timeout`.

- `join_buffer_size` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Zmienna wskazuje minimalną wielkość bufora stosowanego podczas złączeń przeprowadzanych bez użycia indeksów i wymagających pełnego skanowania tabeli i dla określonych typów skanowania indeksu.

- `keep_files_on_create` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Jeżeli w zapytaniu `CREATE TABLE` dla tabeli MyISAM zostały użyte opcje `DATA DIRECTORY` i `INDEX DIRECTORY`, a serwer znajdzie we wskazanym katalogu istniejący plik odpowiednio danych i indeksu, wtedy wygenerowany będzie błąd. Zmienna `keep_files_on_create` określa sposób zachowania serwera podczas tworzenia tabel MyISAM, gdy nie zostaną podane opcje `DATA DIRECTORY` lub `INDEX DIRECTORY` wskazujące miejsce położenia pliku danych i indeksu. Jeżeli wartością zmiennej `keep_files_on_create` będzie `OFF` (to jest wartość domyślna), a serwer znajdzie istniejące pliki danych (`.myd`) i indeksy (`.myi`), wówczas nadpisze je. Jeśli wartością zmiennej będzie `ON`, wtedy w omawianej sytuacji zostanie zgłoszony błąd.

- `key_buffer_size` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Zmienna określa wielkość bufora używanego do buforowania bloków indeksu tabeli MyISAM. Ten bufor jest współdzielony między sesjami.

Ta zmienna i inne dotyczące bufora kluczy (`key_cache_age_threshold`, `key_cache_block_size` i `key_cache_limit`) istnieją w postaci grupy i mogą być dostępne jako komponenty strukturalnej zmiennej systemowej. Istnieje możliwość utworzenia wielu buforów kluczy w celu zachowania większej kontroli nad użyciem bufora kluczy. Więcej informacji na ten temat znajdziesz w punkcie 12.7.2, zatytułowanym „Dostrajanie silnika bazy danych”.

- `key_cache_age_threshold` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Ta wartość określa czas, przez jaki nieużywane bloki pozostaną w części bufora zawierającej nowe wartości, zanim zostaną przeniesione do części bufora zawierającej stare wartości. Wyższa wartość pozwala blokom na dłuższe pozostanie w części zawierającej nowe wartości. Dla tej zmiennej wartością domyślną jest 300, natomiast minimalną 100.

- **key_cache_block_size** (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Zmienna określa wielkość bloku dla bufora kluczy MyISAM. Domyślnie blok ma wielkość 1024 bajtów.
- **key_cache_limit** (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Jeżeli ta zmienna będzie miała ustawioną wartość domyślną wynoszącą 100, bufor klucza używa strategii LRU w celu ustalenia buforów, które mają zostać ponownie wykorzystane. Wartość mniejsza niż 100 powoduje, że bufor kluczy używa strategii wstawiania pośrodku, dzieląc w ten sposób bufor na części nowych kluczy i starych. Wartość może być z zakresu od 1 do 100.
- **language** (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna jest zbędna. Zamiast niej należy używać zmiennych `lc_messages` i `lc_messages_dir`.
- **large_files_support** (zakres: globalny)
Ta zmienna wskazuje, czy serwer został skompilowany wraz z obsługą ogromnych plików.
- **large_page_size** (zakres: globalny)
Zmienna określa wielkość ogromnych stron pamięci, o ile ich obsługa została włączona. W przeciwnym razie wartością zmiennej jest 0.
- **large_pages** (startowa: ustawiana bezpośrednio; zakres: globalny)
Zmienna wskazuje, czy włączona jest obsługa ogromnych stron pamięci. Taka obsługa jest możliwa jedynie w systemie Linux.
- **last_insert_id** (zakres: sesji; dynamiczna)
Ustawienie tej zmiennej wskazuje wartość zwracaną przez funkcję `LAST_INSERT_ID()`. Ta zmienna jest używana podczas przetwarzania binarnego dziennika zdarzeń.
- **lc_messages** (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Zmienna definiuje ustawienia językowe dla komunikatów błędów. Serwer przekłada wskazaną nazwę ustawień językowych na nazwę języka i szuka w katalogu wymienionym w zmiennej `lc_messages_dir` katalogu o podanej nazwie języka i zawierającego plik komunikatu błędu. Więcej informacji znajdziesz w punkcie 12.6.3, zatytułowanym „Wybór języka wyświetlania komunikatów błędów”.
- **lc_messages_dir** (startowa: ustawiana bezpośrednio; zakres: globalny)
Zmienna wskazuje katalog zawierający pliki komunikatów błędów w danym języku.

- `lc_time_names` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Ustawienia językowe dla języka używanego do wyświetlania nazw dni tygodnia i miesięcy przez funkcje `DATE_FORMAT()`, `DAYNAME()` i `MONTHNAME()`. Domyślne ustawienia językowe to `en_US`, ale istnieje możliwość wyboru innych w stylu POSIX, na przykład `es_AR` (hiszpański/Argentyna) lub `zh_HK` (chiński/Hongkong). Więcej informacji znajdziesz w punkcie 12.6.4, zatytułowanym „Wybór ustawień językowych”.
- `license` (zakres: globalny)
Zmienna wskazuje typ licencji serwera, na przykład GPL, jeśli serwer działa w ramach licencji GNU.
- `local_infile` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Zmienna wskazuje, czy serwer zezwala na używanie modyfikatora `LOCAL` w zapytaniach `LOAD DATA`.
- `lock_wait_timeout` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Określa w sekundach czas oczekiwania na każdą blokadę metadanych. Jeżeli blokada nałożona przez zapytanie będzie trwała dłużej niż podana tutaj wartość, wtedy nastąpi wygenerowanie błędu. Dozwolone wartości są z zakresu od 1 do 31536000 (1 rok to jest wartość domyślna). Ta zmienna została wprowadzona w MySQL 5.5.3.
- `locked_in_memory` (startowa: użyj `--memlock`; zakres: globalny)
Zmienna wskazuje, czy serwer został zablokowany w pamięci za pomocą `--memlock`.
- `log` (zakres: globalny)
Ta zmienna jest zbędna i została usunięta w MySQL 5.6. Zamiast niej należy używać `general_log`.
- `log_bin` (zakres: globalny)
Zmienna wskazuje, czy włączony jest binarny dziennik zdarzeń. Zwróć uwagę, że opcja `--bin-log` ustawia `log_bin_basename`, a nie `log_bin`.
- `log_bin_basename` (zakres: globalny)
Pełna ścieżka dostępu do pliku binarnego dziennika zdarzeń. Ta zmienna została wprowadzona w MySQL 5.6.2.
- `log_bin_index` (zakres: globalny)
Nazwa pliku indeksu binarnego dziennika zdarzeń. Ta zmienna została wprowadzona w MySQL 5.6.2.

- `log_bin_trust_function_creators` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

W celu utworzenia lub zmiany funkcji składowanych konieczne jest posiadanie uprawnienia odpowiednio `CREATE ROUTINE` lub `ALTER ROUTINE`. Jednak jeśli włączony jest binarny dziennik zdarzeń, a wartością zmiennej `log_bin_trust_function_creators` jest 0 (to wartość domyślna), wtedy trzeba mieć jeszcze uprawnienie `SUPER` i zadeklarować, że funkcja jest deterministyczna lub nie modyfikuje danych. Aby wyłączyć wspomniane wymagania dodatkowe, należy zmiennej `log_bin_trust_function_creators` ustawić wartość 1.

- `log_error` (startowa: ustawiana bezpośrednio; zakres: globalny)

Nazwa pliku dziennika błędów. Jeżeli nie ustawisz tej zmiennej, serwer będzie przekazywał komunikaty błędów do standardowego wyjścia błędów, czyli terminala. Jeżeli podczas uruchamiania serwera ustawisz tę zmienną bez wartości, plikiem dziennika błędów będzie utworzony w katalogu danych MySQL plik o nazwie `HOSTNAME.err`, gdzie `HOSTNAME` oznacza nazwę komputera serwera. Jeśli nazwa pliku została podana w postaci względnej ścieżki dostępu, serwer będzie ją interpretował jako względem katalogu danych MySQL. Jeżeli podana nazwa nie zawiera rozszerzenia, `mysqld` dołączy rozszerzenie `.err`.

- `log_output` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Zmienna wskazuje standardowe wyjście dla ogólnego dziennika zapytań i dziennika wolno wykonywanych zapytań, o ile wymienione dzienniki są włączone. Wartość zmiennej powinna zawierać wyjścia rozdzielone przecinkami. Dozwolonymi wyjściami są `TABLE`, `FILE` i `NONE`. Jeżeli zmienna będzie zdefiniowana, wartość `NONE` wyłącza rejestrację zdarzeń i ma pierwszeństwo przed pozostałymi wartościami. Wartością domyślną jest `FILE`.

Zmienna systemowa `general_log` lub `slow_query_log` pozwala na włączenie lub wyłączenie dziennika odpowiednio ogólnego i wolno wykonywanych zapytań. Za pomocą zmiennych systemowych `general_log_file` i `slow_query_log_file` można wskazać nazwę odpowiedniego pliku dziennika.

- `log_queries_not_using_indexes` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Jeżeli włączony jest dziennik wolno wykonywanych zapytań, ta zmienna wskazuje, czy zapytania nieużywające indeksów do przeprowadzenia operacji wyszukiwania powinny być rejestrowane. To można znacznie zwiększyć rozmiary dziennika zdarzeń. Patrz także opis zmiennej `log_throttle_queries_not_using_indexes`.

- `log_slave_updates` (startowa: ustawiana bezpośrednio; zakres: globalny)

Ta zmienna określa, kiedy binarny dziennik zdarzeń serwera podległego będzie uaktualniany informacjami pochodzącymi z binarnego dziennika zdarzeń serwera głównego. Domyślnie uaktualnianie binarnego dziennika zdarzeń serwera

podległego jest wyłączone, ale można włączyć tę funkcję i wówczas serwer podległy będzie działał w charakterze serwera głównego dla innych serwerów podległych.

- `log_throttle_queries_not_using_indexes` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Zmienna określa maksymalną liczbę zapytań, jakie w ciągu minuty mogą być zapisane w dzienniku wolno wykonywanych zapytań, o ile została ustawiona zmienna `log_queries_not_using_indexes`. Wartość domyślna wynosi 0, co oznacza brak jakichkolwiek ograniczeń. Ta zmienna została wprowadzona w MySQL 5.6.5.

- `log_slow_queries` (startowa: ustawiana bezpośrednio; zakres: globalny)

Ta zmienna jest zbędna i została usunięta w wydaniu MySQL 5.6. Zamiast niej należy używać `slow_query_log`.

- `log_warnings` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Zmienna określająca poziom rejestracji w dzienniku błędów ostrzeżeń, które nie mają znaczenia krytycznego. Wartość 0 wyłącza wspomniane ostrzeżenia, natomiast 1 (wartość domyślna) je włącza. Wartości większe niż 1 zwiększają poziom rejestrowanych danych, aby zawierały informacje o przerwanych połączeniach i błędach związanych z odmową dostępu.

- `long_query_time` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Każde zapytanie, którego wykonanie będzie trwało dłużej niż wyrażona w sekundach wartość tej zmiennej (i analizujące przynajmniej `min_examined_row_limit` rekordów), zostanie uznane za „wolne” i spowoduje zwiększenie wartości zmiennej stanu o nazwie `Slow_queries`. Ponadto, jeśli włączony jest dziennik wolno wykonywanych zapytań, to zapytanie zostanie w nim zapisane.

Wartości minimalna i domyślna wynoszą odpowiednio 0 i 10. Wartość może zawierać część ułamkową z dokładnością do mikrosekund. Jednak część ułamkowa jest rejestrowana tylko wtedy, gdy dziennik jest zapisywany w pliku, a nie w tabeli `mysql.slow_log`.

- `low_priority_updates` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Jeżeli ta zmienna jest ustawiona, w przypadku silników bazy danych stosujących nakładanie blokad na poziomie tabeli uaktualnienia będą miały niższy priorytet niż operacje pobierania. Zapytania modyfikujące zawartość tabeli (DELETE, INSERT, REPLACE, UPDATE) będą musiały poczekać na zakończenie wykonywania wszelkich zapytań SELECT. Zapytania SELECT pojawiające się, gdy aktywne są inne, wykonywane są natychmiast po zakończeniu poprzedniego zapytania SELECT i nie czekają na wykonanie zapytań modyfikujących o mniejszym priorytecie.

Ustawienie tej zmiennej ma taki sam efekt jak użycie opcji `LOW_PRIORITY` w zapytaniach obsługujących tę opcję, na przykład `INSERT` i `UPDATE`. W przypadku poszczególnych zapytań `INSERT` można użyć modyfikatora `HIGH_PRIORITY` i tym samym zniwelować efekt omawianej zmiennej i zwiększyć ponad normalny priorytet operacji wstawiania danych.

■ `lower_case_file_system` (zakres: globalny)

Ta zmienna określa, czy w przypadku nazw plików w systemie plików zawierającym katalog danych wielkość liter ma znaczenie. Wartość `ON` oznacza, że wielkość liter w nazwach nie ma znaczenia. (Wartość `ON` zmiennej oznacza, że zapisana małymi i wielkimi literami nazwa wskazuje ten sam plik). Z kolei wartość `OFF` oznacza, że wielkość liter ma znaczenie.

■ `lower_case_table_names` (startowa: ustawiana bezpośrednio; zakres: globalny)

Ta zmienna określa, jak nazwy katalogów i plików odpowiadających nazwom baz danych i tabel są traktowane przez zapytania `CREATE DATABASE` i `CREATE TABLE`. Ponadto, zmienna określa również, jak serwer przeprowadza porównania nazw w trakcie wykonywania zapytań. Dozwolone wartości zmiennej zostały wymienione w tabeli D.1.

Tabela D.1. Wartości, które można stosować w zmiennej `lower_case_table_names`

Wartość	Opis
0	Nazwy plików tworzonych na dysku są takie same jak w zapytaniach <code>CREATE DATABASE</code> i <code>CREATE TABLE</code> . Wielkość liter ma znaczenie w trakcie operacji porównań. To jest ustawienie domyślne w systemach, które stosują systemy plików rozróżniające wielkość liter.
1	W trakcie tworzenia baz danych i tabel wymuszane jest stosowanie małych liter. Wielkość liter nie ma znaczenia w trakcie operacji porównań.
2	Wielkość liter zostaje zachowana, ale podczas operacji porównań nie ma znaczenia. Oznacza to, że nazwy są takie same jak podane w zapytaniach <code>CREATE DATABASE</code> i <code>CREATE TABLE</code> , ale w trakcie porównań wielkość liter pozostaje bez znaczenia. Tej wartości zmiennej należy używać jedynie w systemach plików, które nie rozróżniają wielkości liter.

Jeżeli zmienna `lower_case_table_names` nie została wyraźnie ustawiona, serwer automatycznie przypisuje jej wartość 2, o ile wielkość liter w nazwach nie ma znaczenia w systemie plików zawierającym katalog danych MySQL. Przypisanie zmiennej `lower_case_table_names` wartości niezerowanej powoduje także, że wielkość liter nie ma znaczenia w aliasach tabel.

- `master_info_repository` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Zmienna określa, czy serwer podległy zapisuje informacje dziennika głównego w pliku czy w tabeli. Jeżeli wartością zmiennej jest `FILE` (wartość domyślna), serwer podległy zapisuje informacje w pliku wskazanym przez opcję `--master-info-file`. Jeśli wartością zmiennej jest `TABLE`, serwer zapisuje informacje w tabeli `mysql.slave_master_info`. Ta zmienna została wprowadzona w MySQL 5.6.2.

- `master_verify_checksum` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Zmienna określa, czy serwer główny weryfikuje sumy kontrolne zdarzeń odczytywanych z binarnego dziennika zdarzeń przed ich przekazaniem do serwerów podległych. Wartością domyślną jest `OFF`. Ta zmienna została wprowadzona w MySQL 5.6.2.

- `max_allowed_packet` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Maksymalna wielkość bufora używanego do komunikacji między serwerem i klientem. Ten bufor jest początkowo zaalokowany jako o wielkości `net_buffer_length` bajtów, ale jeśli potrzeba, może osiągnąć wielkość `max_allowed_packet` bajtów. Wartość omawianej zmiennej określa także maksymalną długość ciągów tekstowych obsługiwanych przez serwer, na przykład przez funkcje SQL działające na ciągach tekstowych. Wartości domyślna oraz maksymalna zmiennej `max_allowed_packet` wynoszą odpowiednio 1 MB i 1 GB. Wartość sesji jest jedynie do odczytu.

- `max_binlog_cache_size` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Maksymalna wielkość bufora binarnego dziennika zdarzeń. Zapytania tworzące transakcję są przechowywane w wymienionym buforze i zapisywane w binarnym dzienniku zdarzeń w chwili zatwierdzenia transakcji. Jeżeli transakcja przekracza wielkość bufora, musi być zapisana w pliku tymczasowym na dysku. Zapytania wpływające na tabele nietransakcyjne są buforowane oddzielnie, patrz opis zmiennej `max_binlog_stmt_cache_size`.

- `max_binlog_size` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Maksymalna wielkość pliku binarnego dziennika zdarzeń. Jeżeli bieżący plik binarnego dziennika zdarzeń osiągnie tę wielkość, serwer zamyka go i tworzy kolejny. Dozwolony zakres wartości mieści się od 4 KB do 1 GB; wartość domyślna wynosi 1 GB.

Jeżeli wartością zmiennej `max_relay_log_size` jest 0, wtedy zmienna `max_binlog_size` ogranicza również wielkość plików dziennika przekazywania w serwerze podległym.

- `max_binlog_stmt_cache_size` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna jest podobna do `max_binlog_cache_size`, ale dotyczy zapytań wpływających na tabele nietransakcyjne. Tego rodzaju zapytania są buforowane oddzielnie. Ta zmienna została wprowadzona w wydaniu MySQL 5.5.9.
- `max_connect_errors` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Zmienna określa liczbę nieudanych połączeń z komputera, które mogą wystąpić, zanim serwer zablokuje możliwość nawiązywania połączenia z danego komputera. Ma to na celu ochronę przed sytuacją, w której użytkownik próbuje włamać się do serwera z poziomu pewnego komputera. W celu wyczyszczenia bufora i ponownego umożliwienia nawiązywania połączeń z zablokowanych komputerów należy wykonać zapytanie `FLUSH HOSTS` lub wydać polecenie `mysqladmin flush-hosts`. Patrz także zmienne stanu `Connect_errors_xxx`.
- `max_connections` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Maksymalna dozwolona liczba jednoczesnych połączeń klientów. Wartość domyślna wynosi 151.
- `max_delayed_threads` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Maksymalna liczba wątków, które będą utworzone w celu obsługi zapytań `INSERT DELAYED`. Wszelkie zapytania otrzymane po osiągnięciu wspomnianej wartości maksymalnej będą traktowane jako zapytania nie `DELAYED`. W swojej sesji klient może ustawić wartość 0 dla zmiennej `max_delayed_threads` i tym samym wyłączyć zapytania `INSERT DELAYED`.
- `max_error_count` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Maksymalna liczba błędów, ostrzeżeń i innych komunikatów przechowywanych przez serwer. (Tego rodzaju komunikaty zawsze są *liczone*; ta zmienna określa jedynie, ile komunikatów jest *przechowywanych* i dostępnych dla zapytań `SHOW ERRORS` i `SHOW WARNINGS`).
- `max_heap_table_size` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Zmienna określa maksymalną wielkość nowych tabel `MEMORY`. Zmiana wielkości tej zmiennej nie ma żadnego wpływu na istniejące tabele, o ile nie zostaną zmodyfikowane za pomocą zapytań `ALTER TABLE` lub `TRUNCATE TABLE`. Zmienna `max_heap_table_size` może być używana w celu uniemożliwienia serwerowi wykorzystywania ogromnych ilości pamięci. Ma również wpływ na sposób traktowania przez serwer wewnętrznych tabel w pamięci; patrz opis zmiennej `tmp_table_size`.

- `max_insert_delayed_threads` (startowa: użyj `--max_delayed-threads`; zakres: globalny, sesji; dynamiczna)

Ta zmienna jest synonimem zmiennej `max_delayed_threads`.

- `max_join_size` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Podczas przeprowadzania złączenia optymalizator MySQL szacuje liczbę kombinacji rekordów, które trzeba będzie przeanalizować. Jeżeli oszacowana wartość przekracza `max_join_size` rekordów, wtedy wystąpi błąd. Omawiana zmienna może być używana, jeśli użytkownicy mają tendencję do tworzenia niczym nieograniczonych zapytań `SELECT` zwracających ogromną liczbę rekordów. Ograniczenie nie ma zastosowania względem wyników zapytania przechowywanych w buforze zapytań, ponieważ buforowane wyniki mogą być zwrócone bez konieczności ponownego wykonywania zapytania.

Omawiana zmienna jest używana w połączeniu z `sql_big_selects`, jak wspomniano w opisie wymienionej zmiennej. Przypisanie zmiennej `max_join_size` wartości innej niż `DEFAULT` automatycznie powoduje przypisanie wartości 0 zmiennej `sql_big_selects`.

- `max_length_for_sort_data` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Optymalizator zapytań używa tej zmiennej w celu ustalenia typu operacji `filesort` do przeprowadzenia przez `ORDER BY`.

- `max_prepared_stmt_count` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Zmienna określa maksymalną liczbę zapytań preinterpretowanych, które jednocześnie mogą być obsługiwane przez serwer. Niższa wartość tej zmiennej ogranicza ilość pamięci zużywanej przez serwer. Dozwolona wartość może być z zakresu od 0 do 1000000, a wartość domyślna to 16382. Przypisanie zmiennej wartości 0 powoduje całkowite wyłączenie użycia zapytań preinterpretowanych.

- `max_relay_log_size` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Zmienna określa maksymalną wielkość pliku dziennika przekazywania serwera podległego. Jeżeli bieżący plik dziennika przekazywania osiągnie tę wielkość, serwer zamyka go i tworzy kolejny. W przypadku przypisania wartości 0 omawianej zmiennej, serwer będzie używał wartości zmiennej `max_binlog_size` w celu ograniczenia wielkości pliku dziennika przekazywania. Dozwolony zakres wartości niezerowanych mieści się od 4 KB do 1 GB, wartość domyślna wynosi 0.

- `max_seeks_for_key` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Optymalizator zapytań używa tej zmiennej podczas operacji wyszukiwania za pomocą klucza. Jeżeli indeks ma niski poziom różnorodności (niewiele unikalnych wartości), wtedy optymalizator może przyjąć założenie, że operacja wyszukiwania za pomocą klucza będzie wymagała wielu wyszukiwań i zamiast niej przeprowadzi skanowanie tabeli. Przypisanie tej zmiennej małej wartości wskazuje optymalizatorowi, że wymagana będzie co najwyżej wskazana liczba wyszukiwań indeksu. To powoduje użycie indeksu zamiast przeprowadzania skanowania tabeli.

- `max_sort_length` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Wartości danych są sortowane za pomocą pierwszych `max_sort_length` bajtów. Wartość domyślna wynosi 1024. Zmniejszenie tej wartości spowoduje skrócenie czasu porównywania bez zmniejszenia dokładności, o ile sortowane wartości są unikalne we wskazanej liczbie bajtów. Jeżeli wartości nie są unikalne we wskazanej liczbie bajtów, zwiększenie wartości tej zmiennej pozwala na ich lepsze rozróżnianie.

- `max_sp_recursion_depth` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Zmienna wskazuje maksymalną głębokość, do której rekurencję może przeprowadzić każda procedura składowana. Wartość domyślna wynosi 0 (brak zgody na rekurencję), natomiast wartość maksymalna to 255.

- `max_tmp_tables` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Ta zmienna jest nieużywana.

- `max_user_connections` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Maksymalna liczba jednocześnie dozwolonych połączeń klientów dla danego konta użytkownika. Wartość 0 oznacza brak ograniczeń. W każdym przypadku liczba połączeń jest określana globalnie przez wartość `max_connections`.

Wartość sesji omawianej zmiennej jest tylko do odczytu. Wartość sesji będzie taka sama jak globalna, o ile rekord konta użytkownika w tabeli `user` ma wartość niezerową dla `MAX_USER_CONNECTIONS`. W takim przypadku wartość sesji zmiennej jest pobierana ze wspomnianego rekordu konta użytkownika.

Aby zdefiniować ograniczenie liczby połączeń dla danego konta użytkownika, trzeba użyć zapytania `GRANT`.

- `max_write_lock_count` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Po nałożeniu wskazanej przez tę zmienną liczby blokad zapisu w tabeli serwer rozpocznie zwiększanie priorytetu zapytań oczekujących na nałożenie blokad odczytu w tabeli.
- `metadata_locks_cache_size` (startowa: ustawiana bezpośrednio; zakres: globalny)
Zmienna określa wielkość bufora używanego przez serwer dla blokad metadanych. Ta zmienna została wprowadzona w MySQL 5.5.19.
- `min_examined_row_limit` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Zmienna określa minimalną liczbę rekordów, jakie muszą być przeanalizowane przez zapytanie, aby kwalifikowało się ono do zarejestrowania jako wolno wykonywane. Wartość domyślna wynosi 0.
- `myisam_data_pointer_size` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Wyrażona w bajtach wielkość wskaźników rekordu w plikach indeksu MyISAM. Wartość może być z zakresu od 2 do 7, a wartość domyślna wynosi 6. Wielkość wskaźnika można zmienić dla poszczególnych tabel za pomocą opcji tabeli `MAX_ROWS`.
- `myisam_max_sort_file_size` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Operacja ponownego utworzenia indeksu tabeli MyISAM za pomocą zapytań takich jak `REPAIR TABLE`, `ALTER TABLE` lub `LOAD DATA` może używać pliku tymczasowego lub bufora kluczy. Wartość tej zmiennej wskazuje na wykorzystywaną metodę. Jeżeli plik tymczasowy będzie większy niż wartość omawianej zmiennej, zamiast niego zostanie użyty bufor kluczy.
- `myisam_mmap_size` (startowa: ustawiana bezpośrednio; zakres: globalny)
Zmienna określa maksymalną ilość pamięci używanej do mapowania skompresowanych plików tabel MyISAM. Ta zmienna została wprowadzona w MySQL 5.5.1.
- `myisam_recover_options` (startowa: ustawiana bezpośrednio; zakres: globalny)
Zmienna wskazuje tryb używany podczas automatycznej naprawy tabel MyISAM. Jeżeli zostanie ustawiona, gdy serwer otworzy tabelę MyISAM, wtedy przeprowadzi jej naprawę, jeśli tabela została oznaczona jako uszkodzona lub niezamknięta prawidłowo podczas ostatniego użycia. Wartością zmiennej powinna być rozdzielona przecinkami lista jednej lub więcej następujących opcji: `BACKUP` (utworzenie kopii zapasowej tabeli, jeśli operacja naprawy spowoduje jej modyfikację), `FORCE` (wymuszenie naprawy, nawet jeśli

doprowadzi ona do usunięcia więcej niż rekordu danych), QUICK (szybka naprawa), DEFAULT (naprawa bez innych specjalnych operacji) i OFF (brak naprawy, to jest wartość domyślna).

Dobrym rozwiązaniem jest włączenie automatycznej naprawy, jeśli serwer działa wraz z ustawioną zmienną systemową `delay_key_write` lub zawiera poszczególne tabele MyISAM skonfigurowane do zapisu kluczy z opóźnieniem.

- `myisam_repair_threads` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Zmienna określa maksymalną liczbę wątków używanych do utworzenia indeksów tabeli MyISAM w trakcie operacji naprawy. Wartością domyślną jest 1 dla naprawy przeprowadzanej przez pojedynczy wątek. Ustawienie wartości większej niż 1 w celu przeprowadzenia wielowątkowej operacji naprawy powinno być uznawane za eksperymentalne.

- `myisam_sort_buffer_size` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Wielkość bufora zaalokowanego na potrzeby sortowania indeksu tabel MyISAM w trakcie operacji takich jak `ALTER TABLE`, `CREATE INDEX` i `REPAIR TABLE`.

- `myisam_stats_method` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Ta zmienna określa, czy serwer powinien uznawać wartości NULL za takie same, czy inne podczas obliczania danych statystycznych dotyczących rozproszenia wartości klucza dla tabel MyISAM. Zmiennej można przypisać wartość `nulls_equal` (wszystkie wartości NULL znajdują się w tej samej grupie), `nulls_unequal` (każda wartość NULL tworzy oddzielną grupę) lub `nulls_ignored` (wartości NULL są ignorowane).

- `myisam_use_mmap` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Zmienna określa, czy serwer będzie używał mapowania pamięci w celu odczytu i zapisu tabel MyISAM. Wartość domyślna zmiennej to OFF.

- `named_pipe` (startowa: ustawiana bezpośrednio)

Zmienna wskazuje, czy serwer zezwala na używanie przez klienty połączeń poprzez nazwane potoki. Tego rodzaju połączenia są obsługiwane jedynie przez Windows. Wartość domyślna zmiennej wynosi OFF.

- `net_buffer_length` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Zmienna określa początkową wielkość buforów połączenia i wyniku używanych do komunikacji między serwerem i klientem. Wielkość bufora można zwiększyć do `max_allowed_packet` bajtów. Wartość zmiennej może być z zakresu od 1 KB do 1 MB, wartość domyślna wynosi 16 KB. Wartość sesji jest tylko do odczytu.

- `net_read_timeout` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Liczba sekund oczekiwania na dane od klienta, zanim nastąpi przekroczenie czasu oczekiwania.
- `net_retry_count` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Zmienna określa, ile razy będzie powtarzana przerwana operacja odczytu.
- `net_write_timeout` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Liczba sekund oczekiwania podczas przekazywania danych od klienta, zanim nastąpi przekroczenie czasu oczekiwania.
- `new` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Ta zmienna jest nieużywana.
- `old` (startowa: ustawiana bezpośrednio; zakres: globalny)
Zapewniająca zachowanie zgodności zmienna, która pozwala na włączenie starego zachowania dla niektórych funkcji. Obecnie powoduje, że wskazówki indeksu nie mają zastosowania w trakcie operacji `ORDER BY` lub `GROUP BY`.
- `old_alter_table` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
W przypadku niektórych operacji `ALTER TABLE` wymagających w MySQL tymczasowej kopii tabeli obecne wersje serwera nie wymagają tworzenia wspomnianej kopii tymczasowej. Włączenie tej zmiennej powoduje, że serwer będzie używał kopii tymczasowej.
- `old_passwords` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Począwszy od wydania MySQL 4.1, serwer obsługuje znacznie bezpieczniejszą metodę szyfrowania haseł niż stosowana we wcześniejszych wersjach. Istniejące konta użytkowników z hasłami zaszyfrowanymi w stary sposób nadal są obsługiwane, ale domyślnie omawiana zmienna jest wyłączona i nowe hasła są szyfrowane już za pomocą nowej metody. Włączenie omawianej zmiennej powoduje, że hasła będą szyfrowane za pomocą starej metody, stosowanej w wersjach MySQL wcześniejszych niż 4.1. To może być użyteczne w przypadku powrotu do użycia starszej wersji serwera lub w trakcie przenoszenia kont użytkowników do starszych wersji serwera.
- `open_files_limit` (startowa: ustawiana bezpośrednio; zakres: globalny)
Zmienna określa liczbę deskryptorów plików, które serwer próbuje zarezerwować. Jeżeli w trakcie uruchamiania serwera ta zmienna będzie miała wartość niezerową, ale rzeczywista wartość wyświetlana przez serwer będzie mniejsza niż wskazana, wartość wskazuje maksymalną liczbę deskryptorów plików dozwolonych przez

system operacyjny. (Jeżeli serwer wyświetla wartość 0, oznacza to, że system operacyjny nie pozwala serwerowi `mysqld` na zmianę liczby deskryptorów). Jeżeli omawianej zmiennej nie ustawisz w trakcie uruchamiania serwera lub przypiszesz jej wartość 0, wtedy serwer użyje większej spośród $\text{max_connections} \times 5$ i $\text{max_connections} + \text{table_open_cache} \times 2$ jako rezerwowanej liczby deskryptorów. Zmienna `open_files_limit` kontroluje alokację deskryptorów plików oddzielnie od kontrolowanej przez `innodb_open_files`.

- `optimizer_prune_level` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Optymalizator zapytań analizuje wiele różnych planów wykonania w celu wybrania najlepszego. Omawiana zmienna określa, w jaki sposób optymalizator obsługuje plany pośrednie. Jeżeli wartością zmiennej jest 1 (to wartość domyślna), optymalizator odrzuca plany pośrednie na podstawie oszacowanej liczby rekordów, które będą musiały być przeanalizowane. Jeżeli wartością zmiennej jest 0, optymalizator przeprowadza wyczerpującą analizę wszystkich planów.

- `optimizer_search_depth` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Zmienna określa głębokość, na jaką zejdzie optymalizator zapytań w poszukiwaniu planów wykonania. Wartość 0 zmiennej oznacza, że optymalizator automatycznie wybierze rozsądną wartość. Domyślnie przeprowadzane jest dokładne wyszukiwanie.

- `optimizer_switch` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Ta zmienna pozwala na włączenie lub wyłączenie strategii stosowanych przez optymalizator. Wartością zmiennej jest rozdzielona przecinkami lista ustawień *flaga=wartość*, gdzie *wartość* oznacza on lub off. Z kolei *flaga* może mieć przypisaną wartość `default`, powodującą zwrot wartości domyślnej danej flagi. Ewentualnie zmiennej `optimizer_switch` można przypisać wartość `default`, przywracającą wartości domyślne wszystkich flag. Opis poszczególnych strategii znajdziesz w podręczniku użytkownika MySQL.

- `optimizer_trace_xxx` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji)

Zmienne systemowe o nazwach rozpoczynających się od `optimizer_trace` są używane w celu śledzenia wewnętrznej działalności optymalizatora zapytań. Ta zmienna została wprowadzona w MySQL 5.6.3.

- `performance_schema_xxx`

Zmienne systemowe o nazwach rozpoczynających się od `performance_schema` są używane w celu zbierania i analizy danych dotyczących wydajności serwera. Więcej informacji na ten temat znajdziesz w podręczniku użytkownika MySQL.

- `pid_file` (startowa: ustawiana bezpośrednio; zakres: globalny)

Podczas uruchamiania serwera `mysqld`, identyfikator procesu (PID) jest zapisywany w tak zwanym pliku PID. Omawiana zmienna zawiera ścieżkę

dostępu do wspomnianego pliku PID. Plik może być używany przez inne procesy, aby sprawdzić numer procesu serwera MySQL, zwykle w celu wysłania mu sygnału. Na przykład, skrypt `mysql.server` odczytuje plik PID podczas wysyłania serwerowi sygnału nakazującego zakończenie jego działania. Jeżeli nazwa pliku PID zostanie podana w postaci względnej ścieżki dostępu, serwer będzie interpretował ją względem katalogu danych. Domyślnie przez tę zmienną tworzony jest plik o nazwie *komputera.pid*.

- **plugin_dir** (startowa: ustawiana bezpośrednio; zakres: globalny)
Zmienna przechowuje ścieżkę dostępu do katalogu zawierającego wtyczki serwera.
- **port** (startowa: ustawiana bezpośrednio; zakres: globalny)
Zmienna przechowuje numer portu TCP/IP serwera używanego w celu nasłuchiwania połączeń przychodzących od klientów.
- **preload_buffer_size** (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Ta zmienna określa wielkość bufora alokowanego podczas przygotowywania indeksów za pomocą zapytania `LOAD INDEX`.
- **protocol_version** (zakres: globalny)
Zmienna zawiera numer protokołu klient-serwer używanego przez serwer.
- **proxy_user** (zakres: sesji)
To jest zmienna sesji tylko do odczytu wskazująca użytkownika proxy, o ile stosowane jest proxy. W przeciwnym razie wartością zmiennej jest `NULL`. Ta zmienna została wprowadzona w MySQL 5.5.7.
- **pseudo_thread_id** (zakres: sesji; dynamiczna)
Ta zmienna jest używana wewnętrznie przez serwer.
- **query_alloc_block_size** (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Zmienna określa wielkość bloku alokowanego jako pamięć tymczasowa podczas przetwarzania i wykonywania zapytań.
- **query_cache_limit** (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Zmienna określa maksymalną wielkość buforowanych wyników zapytania. Wyniki większe niż wartość omawianej zmiennej nie będą buforowane. Wartość domyślna wynosi 1 MB.
- **query_cache_min_res_unit** (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Zmienna określa wielkość bloku alokowanego w pamięci i przeznaczonego do przechowywania wyników w buforze zapytań. Wartość domyślna wynosi 4 KB.

- `query_cache_size` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Zmienna określa ilość pamięci używanej do buforowania wyniku zapytania. Ustawienie tej zmiennej wartości 0 powoduje wyłączenie bufora, nawet jeśli wartością zmiennej `query_cache_type` nie jest OFF. Dlatego też przypisanie omawianej zmiennej wartości niezerowej powoduje alokację wskazanej ilości pamięci, nawet jeśli wartością zmiennej `query_cache_type` jest OFF. Wartość powinna być wielokrotnością 1024.
- `query_cache_type` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Zmienna określa tryb działania bufora zapytań, o ile wartość zmiennej `query_cache_size` jest większa niż 0. W tabeli D.2 wymieniono dozwolone wartości omawianej zmiennej.

Tabela D.2. Wartości dozwolone do przypisania zmiennej `query_cache_type`

Tryb	Opis
0	Wyniki nie będą buforowane, a więc nie będzie żadnych wyników do pobrania z bufora.
1	Buforowane będą wyniki zapytań możliwych do buforowania poza rozpoczynającymi się od <code>SELECT SQL_NO_CACHE</code> .
2	Buforowanie na żądanie jedynie zapytań możliwych do buforowania, które rozpoczynają się od <code>SELECT SQL_CACHE</code> .

Jeżeli zdecydujesz się na przypisanie zmiennej `query_cache_type` wartości za pomocą zapytania SET, wartościom symbolicznym OFF, ON i DEMAND odpowiadają wartości liczbowe 0, 1 i 2.

- `query_cache_wlock_invalidate` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Jeżeli omawiana zmienna ma przypisaną wartość 0 (to jest wartość domyślna), klienci mogą pobierać buforowane wyniki zapytania dla tabeli nawet wtedy, gdy inny klient nałożył blokadę WRITE na tę tabelę. Przypisanie omawianej zmiennej wartości 1 powoduje, że buforowane wyniki są unieważniane po nałożeniu blokady WRITE przez klienta, co wymusza na innych klientach poczekanie na zwolnienie wymienionej blokady.
- `query_prealloc_size` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Zmienna określa wielkość bufora alokowanego na potrzeby przetwarzania i wykonywania zapytań. Ten bufor nie jest opróżniany między kolejno wykonywanymi zapytaniami, w przeciwieństwie do bloków alokowanych przez zmienną `query_alloc_block_size`.

- `rand_seed1, rand_seed2` (zakres: sesji; dynamiczna)
Ta zmienna sesji tylko do odczytu jest używana wewnętrznie do powielania funkcji `RAND()`.
- `range_alloc_block_size` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Ta zmienna określa wielkość bloku alokowanego w pamięci podczas przeprowadzania optymalizacji zakresu.
- `read_buffer_size` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Ta zmienna określa wielkość bufora używanego przez wątki przeprowadzające sekwencyjne skanowanie tabeli. Bufor jest alokowany na żądanie dla poszczególnych klientów.
- `read_only` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa, czy serwer działa w trybie tylko do odczytu dla połączeń klientów. Domyślnie wartością zmiennej `read_only` jest `OFF`, uaktualnienia klientów są akceptowane w zwykły sposób (to znaczy mają odpowiednie uprawnienia do ich przeprowadzania). Po przypisaniu omawianej zmiennej wartości `ON` uaktualnienia są dozwolone do wykonania jedynie przez zapytania otrzymane z serwera głównego (w przypadku serwera podległego) lub wydane przez klientów z uprawnieniem `SUPER`. Wykonanie zapytania `SET PASSWORD` wymaga uprawnienia `SUPER`. Wartość zmiennej `read_only` nie ma zastosowania względem tabel tymczasowych.

Omawianej zmiennej nie można ustawić w przypadku nałożenia blokady na tabelę lub gdy przeprowadzana jest transakcja. Jeżeli spróbujesz ustawić zmienną `read_only` w chwili, gdy inne klienty mają nałożone blokady na tabelę lub przeprowadzają transakcję, takie żądanie będzie wstrzymane aż do chwili zwolnienia blokady i zakończenia transakcji. W trakcie blokady wymienionego żądania inne klienty również pozostają zablokowane, jeśli chcą nałożyć nowe blokady na tabelę lub rozpocząć nowe transakcje. Wymienione tutaj warunki dotyczące blokowania nie mają zastosowania względem blokady `FLUSH TABLES WITH READ LOCK`, która jest globalną blokadą odczytu, a nie blokadą na poziomie tabeli.
- `read_rnd_buffer_size` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Zmienna określa wielkość bufora używanego do odczytywania rekordów w kolejności po ich posortowaniu. Bufor jest alokowany na żądanie dla poszczególnych klientów.
- `relay_log` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna wskazuje nazwę pliku dziennika przekazywania.

- `relay_log_basename` (zakres: globalny)
Ta zmienna wskazuje pełną ścieżkę dostępu do pliku dziennika przekazywania. Omawiana zmienna została wprowadzona w MySQL 5.6.2.
- `relay_log_index` (zakres: globalny)
Ta zmienna wskazuje nazwę pliku indeksu dziennika przekazywania.
- `relay_log_info_file` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna wskazuje nazwę pliku informacyjnego dziennika przekazywania, domyślnie tworzony jest plik o nazwie *relay-log.info*.
- `relay_log_info_repository` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa miejsce zapisu informacji dziennika przekazywania w serwerze podległym: plik lub tabela. Jeżeli wartością zmiennej jest `FILE` (to wartość domyślna), wtedy dziennikiem zdarzeń jest plik wskazany przez opcję `--relay-log-info-file`. Z kolei wartość `TABLE` powoduje, że informacje dziennika zdarzeń są zapisywane w tabeli `mysql.slave_relay_log_info_file`. Omawiana zmienna została wprowadzona w MySQL 5.6.2.
- `relay_log_purge` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Kiedy zmienna ma przypisaną wartość 1 (to jest wartość domyślna), serwer podległy usuwa każdy plik dziennika przekazywania, gdy tylko nie będzie już potrzebny. W przypadku przypisania zmiennej wartości 0 pliki dziennika przekazywania nie są usuwane automatycznie.
- `relay_log_recovery` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna może okazać się użyteczna po wystąpieniu awarii serwera podległego. Ustawienie jej podczas uruchamiania powoduje, że serwer podległy usuwa wszelkie nieprzetworzone jeszcze dzienniki przekazywania i ponownie pobiera je z serwera głównego. Domyślnie omawiana zmienna nie jest ustawiona.
- `relay_log_space_limit` (startowa: ustawiana bezpośrednio; zakres: globalny)
Zmienna określa maksymalną ilość miejsca przeznaczoną na pliki dziennika przekazywania. Po osiągnięciu ustalonego limitu wątek wejścia-wyjścia w serwerze podległym czeka, aż wątek SQL przetworzy zdarzenia i zwolni miejsce przez usunięcie plików dziennika przekazywania.
- `report_host`, `report_password`, `report_port`, `report_user` (zakres: globalny)
W serwerze podległym replikacji wymienione zmienne mogą być ustawione w celu wskazania wartości, które serwer podległy powinien zgłaszać serwerowi głównemu po nawiązaniu połączenia. Wspomniane wartości nie muszą odpowiadać wartościom faktycznie używanym przez serwer podległy do nawiązania połączenia. Wartości omawianych zmiennych pojawiają się w danych wyjściowych zapytania `SHOW SLAVE HOSTS`, gdy zostanie ono

wykonane w serwerze głównym. Wartości zmiennych `report_user` i `report_password` nie są zgłaszane, o ile serwer główny nie zostanie uruchomiony wraz z opcją `--show-slave-auth-info`.

- **secure_auth** (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa, czy serwer zezwala na nawiązywanie połączeń jedynie z kontami użytkowników, dla których hasło zostało zaszyfrowane w formacie wprowadzonym w wydaniu MySQL 4.1. Przypisanie wartości `OFF` omawianej zmiennej powoduje, że serwer zezwala również na nawiązywanie połączeń z kontami użytkowników posiadającymi hasła w formacie stosowanym przed wydaniem MySQL 4.1. Wartością domyślną jest `OFF` w wersjach wcześniejszych niż MySQL 5.6.5 i `ON` w późniejszych.
- **secure_file_priv** (startowa: ustawiana bezpośrednio; zakres: globalny)
Kiedy ta zmienna ma wartość w postaci ścieżki dostępu do katalogu, wtedy serwer pozwala, aby zapytania `LOAD DATA` i `SELECT ... INTO OUTLINE`, a także funkcja `LOAD_FILE()` przeprowadzały operacje jedynie w wymienionym katalogu. Wartością domyślną omawianej zmiennej jest pusta wartość, która oznacza brak wymienionego ograniczenia.
- **server_id** (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Zmienna przechowuje identyfikator serwera stosowany w replikacji. Jeżeli wartością zmiennej będzie 0, serwer nie uczestniczy w replikacji. W przeciwnym razie wartość musi być liczbą całkowitą z zakresu od 1 do $2^{32}-1$ i unikalną wśród komunikujących się ze sobą serwerów replikacji.
- **server_uuid** (zakres: globalny)
Ta zmienna przechowuje automatycznie wygenerowany UUID dla danego serwera. Omawiana zmienna została wprowadzona w MySQL 5.6.0.
- **shared_memory** (startowa: ustawiana bezpośrednio)
Ta zmienna określa, czy serwer zezwala na nawiązywanie przez klienty połączeń za pomocą pamięci współdzielonej. Tego rodzaju połączenia są obsługiwane jedynie w Windows. Wartością domyślną zmiennej jest `OFF`.
- **shared_memory_base_name** (startowa: ustawiana bezpośrednio)
Zmienna zawiera nazwę pamięci współdzielonej dla połączeń nawiązywanych przez tę pamięć. Wartością domyślną jest `MYSQL` (wielkość liter ma znaczenie).
- **skip_external_locking** (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna określa, czy wyłączone będzie stosowanie zewnętrznego nakładania blokad (czyli przeprowadzanej przez system plików operacji nakładania blokad).
- **skip_name_resolve** (startowa: ustawiana bezpośrednio; zakres: globalny)
Domyślnie ta zmienna nie jest ustawiona. Jej ustawienie powoduje wyłączenie określania nazwy komputera i tabela uprawnień musi wówczas wymieniać komputery przez ich adresy IP lub jako `localhost`.

- **skip_networking** (startowa: ustawiana bezpośrednio; zakres: globalny)
Jeżeli ta zmienna nie jest ustawiona (tak jest domyślnie), serwer zezwala na połączenia TCP/IP. Ustawienie omawianej zmiennej uniemożliwia nawiązywanie połączeń TCP/IP. W takim przypadku klienci będą mogli nawiązywać połączenia jedynie z poziomu komputera lokalnego, używając interfejsu innego niż TCP/IP. Klienci systemów UNIX mogą wtedy nawiązywać połączenie za pomocą pliku gniazda UNIX. Z kolei klienci Windows mają do dyspozycji połączenia za pomocą pamięci współdzielonej lub nazwanego potoku, o ile dozwolone jest stosowanie wymienionego typu połączeń.
- **skip_show_database** (startowa: ustawiana bezpośrednio; zakres: globalny)
Jeżeli ta zmienna nie jest ustawiona (tak jest domyślnie), użytkownik może wykonywać zapytanie `SHOW DATABASES`. Wymienione zapytanie powoduje wyświetlenie wszystkich baz danych, o ile użytkownik ma uprawnienie `SHOW DATABASES`. W przeciwnym razie wyświetlane są jedynie te bazy danych, do których użytkownik ma uprawnienia. Jeżeli omawiana zmienna zostanie ustawiona, zapytanie `SHOW DATABASES` będzie mogło być wykonywane jedynie przez użytkowników posiadających uprawnienie `SHOW DATABASES` i spowoduje wyświetlenie wszystkich baz danych.
- **slave_allow_batching** (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Zmienna pozwala serwerowi podległemu na żądania wsadowe; ma zastosowanie jedynie w przypadku klastra MySQL.
- **slave_checkpoint_group** (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Jeżeli dozwolone jest wielowątkowe działanie serwera podległego (patrz zmienna `slave_parallel_threads`), ta zmienna określa maksymalną liczbę transakcji wykonywanych przez serwer podległy, zanim nastąpi utworzenie punktu kontrolnego. Wartością domyślną jest 512. Dozwolone są wartości z zakresu od 32 do 512 KB. Omawiana zmienna została wprowadzona w MySQL 5.6.3. Tworzenie punktu kontrolnego zależy od zmiennych `slave_checkpoint_group` i `slave_checkpoint_period`. Przekroczenie limitu powoduje utworzenie przez serwer podległy punktu kontrolnego i wyzerowanie liczników powiązanych z wymienionymi zmiennymi.
- **slave_checkpoint_period** (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Jeżeli dozwolone jest wielowątkowe działanie serwera podległego (patrz zmienna `slave_parallel_threads`), ta zmienna określa wyrażoną w milisekundach maksymalną ilość czasu wykonywania zapytań przez serwer podległy, zanim nastąpi utworzenie punktu kontrolnego. Wartością domyślną jest 300. Dozwolone są wartości z zakresu od 1 do 4 GB. Omawiana zmienna została wprowadzona w MySQL 5.6.3. Patrz także opis zmiennej `slave_checkpoint_group`.

- `slave_compressed_protocol` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa, czy powinna być używana kompresja w celu zmniejszenia ilości ruchu sieciowego generowanego między serwerami podległym i głównym. W takim przypadku wymagane jest, aby serwer zarówno główny, jak i podległy obsługiwały protokół skompresowany.
- `slave_exec_mode` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Zmienna określa tryb działania serwera podległego: `STRICT` (wartość domyślna) lub `IDEMPOTENT`. Drugi z wymienionych może być użyteczny w przypadku topologii replikacji obejmujących pętle lub wiele serwerów głównych, co chroni przed błędami typu powielenia kluczy lub niezalezionych kluczy.
- `slave_load_tmpdir` (startowa: ustawiana bezpośrednio; zakres: globalny)
Zmienna określa ścieżkę dostępu do katalogu używanego podczas przetwarzania zapytań `LOAD DATA`, jeśli serwer jest serwerem podległym replikacji. Wartością domyślną omawianej zmiennej jest zmienna systemowa `tmpdir`.
- `slave_max_allowed_packet` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Zmienna podobna do `max_allowed_packet`, ale dla wątków SQL i wejścia-wyjścia serwera podległego. Wartością domyślną jest 1 GB.
- `slave_net_timeout` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Zmienna określa liczbę sekund, przez które serwer podległy będzie oczekiwał na dane z serwera głównego, zanim nastąpi przekroczenie czasu oczekiwania.
- `slave_parallel_workers` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Przypisanie tej zmiennej wartości większej niż 0 w serwerze podległym powoduje włączenie replikacji wielowątkowej. Wątek SQL standardowo wykonuje wszystkie zdarzenia otrzymane z serwera podległego, zamiast stać się koordynatorem `slave_parallel_workers` wątków roboczych odpowiedzialnych za przetwarzanie zdarzeń. Wartością domyślną jest 0. Dozwolone wartości mieszczą się w zakresie od 0 do 1024. Ta zmienna została wprowadzona w MySQL 5.6.3.
- `slave_pending_jobs_size_max` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Jeżeli dozwolone jest wielowątkowe działanie serwera podległego (patrz zmienna `slave_parallel_threads`), ta zmienna określa maksymalną ilość pamięci przeznaczoną do buforowania nieprzetworzonych zdarzeń dla wątków roboczych. Wartością domyślną jest 16 MB. Dozwolone są wartości z zakresu od 1 KB do 18 EB. Omawiana zmienna została wprowadzona w MySQL 5.6.3.

- `slave_skip_errors` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna zawiera listę błędów, które serwer podległy powinien ignorować, zamiast wstrzymywać replikację po ich wystąpieniu. (Jednak zwykle najlepszym rozwiązaniem jest ustalenie przyczyny błędu i jego wyeliminowanie zamiast użycia omawianej zmiennej do zignorowania błędu). Wartość `all` oznacza, że wszystkie błędy powinny być zignorowane. Wartością zmiennej powinna być lista jednego lub więcej numerów błędów rozdzielonych przecinkami.
- `slave_sql_verify_checksum` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa, czy wątek SQL serwera podległego powinien weryfikować sumy kontrolne zdarzeń odczytanych z dziennika przekazywania. Jeżeli wystąpi błąd związany z sumą kontrolną, replikacja zostanie wstrzymana. Wartością domyślną zmiennej jest `OFF`. Omawiana zmienna została wprowadzona w MySQL 5.6.2.
- `slave_transaction_retries` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa, ile razy serwer podległy powinien ponawiać transakcję, której przeprowadzenie nie powiodło się ze względu na zakleszczenie lub przekroczenie czasu oczekiwania dla danego silnika bazy danych. Wartością domyślną jest `10`.
- `slave_type_conversions` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna wskazuje dozwolone typy konwersji w serwerze podległym dla replikacji opartej na rekordach. Wartością domyślną jest pusty ciąg tekstowy (brak zgody na konwersje). Niepusta wartość zmiennej powinna być rozdzieloną przecinkami listą jednej lub więcej wartości `ALL_LOSSY` (zezwolenie na utratę informacji w trakcie konwersji) lub `ALL_NON_LOSSY` (zezwolenie na konwersje, które nie wiążą się z utratą informacji). Omawiana zmienna została wprowadzona w MySQL 5.5.3.
- `slow_launch_time` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa liczbę sekund definiujących „powolne” tworzenie wątku. Jeżeli utworzenie wątku będzie trwało dłużej, nastąpi zwiększenie wartości zmiennej stanu `Slow_launch_threads`.
- `slow_query_log` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa, czy istnieje możliwość rejestracji informacji w dzienniku wolno wykonywanych zapytań. Jeśli tak, wtedy zmienna `log_output` wskazuje miejsce zapisu tych informacji.

- `slow_query_log_file` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Ta zmienna zawiera nazwę pliku dziennika wolno wykonywanych zapytań, który będzie używany w przypadku wybrania `FILE` jako miejsca zapisu informacji wymienionego dziennika. Domyślnie w katalogu danych MySQL tworzony jest plik o nazwie `HOSTNAME-slow.log`, gdzie `HOSTNAME` oznacza nazwę komputera, w którym działa serwer. Jeżeli nazwa zostanie podana w postaci względnej ścieżki dostępu, serwer będzie ją interpretował względem katalogu danych MySQL.

- `socket` (startowa: ustawiana bezpośrednio; zakres: globalny)

Zmienna zawiera ścieżkę dostępu do pliku gniazda domeny UNIX lub też nazwanego potoku, jeśli serwer działa w systemie Windows.

- `sort_buffer_size` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Ta zmienna określa wielkość bufora używanego przez wątki do przeprowadzania operacji sortowania (`GROUP BY` lub `ORDER BY`). Ten bufor jest alokowany na żądanie dla poszczególnych klientów. Normalnie, jeśli kilka klientów jednocześnie przeprowadza operację sortowania, nierozsądne jest nadmierne zwiększanie wartości omawianej zmiennej (czyli powyżej 1 MB).

- `sql_auto_is_null` (zakres: globalny, sesji; dynamiczna)

Jeżeli wartością tej zmiennej będzie 1, ostatnio wygenerowana wartość `AUTO_INCREMENT` będzie mogła być wybrana za pomocą klauzuli w formie `WHERE nazwa_kolumny IS NULL`, gdzie `nazwa_kolumny` jest nazwą kolumny `AUTO_INCREMENT`. Pewne programy ODBC używają tej funkcji. Aby ją wyłączyć, należy omawianej zmiennej przypisać wartość 0. Wartością domyślną jest 0 (w wersjach wcześniejszych niż MySQL 5.5.3 to wartość 1).

- `sql_big_selects` (zakres: globalny, sesji; dynamiczna)

Serwer używa tej zmiennej w połączeniu ze zmienną systemową `max_join_size`. Jeżeli wartością zmiennej `sql_big_selects` jest 1 (to wartość domyślna), serwer akceptuje zapytania, których wykonanie zwraca zbiór wynikowy o dowolnej wielkości. W przypadku gdy wartością zmiennej `sql_big_selects` jest 0, serwer odrzuca zapytania, które mogą zwrócić ogromną liczbę rekordów. W takim przypadku podczas przeprowadzania złączenia jest używana wartość zmiennej `max_join_size`: serwer szacuje konieczną liczbę kombinacji rekordów do przeanalizowania. Jeżeli oszacowana wartość jest większa niż `max_join_size`, wtedy serwer zgłasza błąd, zamiast wykonać zapytanie.

Przypisanie zmiennej `max_join_size` wartości innej niż `DEFAULT` automatycznie powoduje ustawienie wartości 0 zmiennej `sql_big_selects`.

- `sql_buffer_result` (zakres: globalny, sesji; dynamiczna)

Przypisanie wartości 1 tej zmiennej powoduje, że serwer używa wewnętrznych tabel tymczasowych do przechowywania wyników wykonania zapytań `SELECT`.

W efekcie serwer będzie mógł znacznie szybciej zwalniać blokady nakładane na tabele, na podstawie których ma zostać wygenerowany wynik zapytania. Wartością domyślną zmiennej jest 0.

■ **sql_log_bin** (zakres: globalny, sesji; dynamiczna)

Ustawienie wartości sesji 0 lub 1 tej zmiennej powoduje odpowiednio wyłączenie lub włączenie rejestracji danych w binarnym dzienniku zdarzeń dla bieżącej sesji klienta. W trakcie działania serwera klient musi mieć uprawnienie SUPER, aby móc zmienić wartość tej zmiennej, a nawet wartość sesji. Omawiana zmienna nie ma efektu, o ile nie zostanie włączony binarny dziennik zdarzeń.

■ **sql_log_off** (zakres: globalny, sesji; dynamiczna)

Ustawienie wartości sesji 0 lub 1 tej zmiennej powoduje odpowiednio włączenie lub wyłączenie rejestracji zapytań w ogólnym dzienniku zapytań dla bieżącej sesji klienta. W trakcie działania serwera klient musi mieć uprawnienie SUPER, aby móc zmienić wartość tej zmiennej, a nawet wartość sesji. Omawiana zmienna nie ma efektu, o ile nie zostanie włączony ogólny dziennik zapytań.

■ **sql_mode** (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Ta zmienna określa tryb SQL serwera. Powoduje modyfikację różnych aspektów zachowania serwera, które mogą spowodować, że będzie on działał zgodnie ze standardem SQL, zgodnie z innymi serwerami baz danych lub zgodnie ze starszymi wersjami serwerów MySQL. Wartość zmiennej powinna być w postaci pustego ciągu tekstowego (wyłączenie trybów) lub rozdzielonej przecinkami listy jednego lub więcej wymienionych dalej trybów. Począwszy od wydania MySQL 5.6.6, wartością domyślną jest NO_ENGINE_SUBSTITUTION, natomiast we wcześniejszych wersjach to pusty ciąg tekstowy. Pewne tryby są proste i włączają jedno konkretne zachowanie. Z kolei inne są trybami złożonymi i działają w charakterze skrótów pozwalających na łatwe włączenie pewnego zestawu trybów. Wartości określające nazwy trybów nie rozróżniają wielkości liter.

Pojęcie „tryb ścisły” odnosi się do ustawienia zmiennej sql_mode wartości STRICT_TRANS_TABLES lub STRICT_ALL_TABLES, co powoduje, że serwer ściśle sprawdza wstawiane dane. W podrozdziale 3.3, zatytułowanym „Jak MySQL obsługuje nieprawidłowe wartości danych?”, znajdziesz więcej informacji na temat trybu ścisłego oraz innych wpływających na obsługę danych wejściowych.

Poniżej wymieniono proste tryby SQL:

■ **ALLOW_INVALID_DATES**

W trybie ścisłym powoduje wyłączenie pełnego sprawdzania poprawności wartości typu DATE i DATETIME. Jedyne wymaganie jest takie, aby miesiąc był liczbą z zakresu od 1 do 12, natomiast dzień liczbą z zakresu od 1 do 31. Wartości typu TIMESTAMP muszą być prawidłowe niezależnie od wybranego trybu.

■ **ANSI_QUOTES**

Powoduje, że znaki cudzysłowu są traktowane jako znaki cytowania dla identyfikatorów takich jak nazwy baz danych, tabel i kolumn, a nie jako znaki cytowania ciągu tekstowego. (Odwrotne apostrofy są dozwolone do cytowania nazw niezależnie od wybranego trybu).

■ **ERROR_FOR_DIVISION_BY_ZERO**

W przypadku operacji wstawiania lub uaktualniania danych operacja dzielenia (lub modulo) o dzielniku 0 normalnie powoduje wygenerowanie wyniku NULL i brak ostrzeżenia, nawet w trybie ścisłym. Włączenie `ERROR_FOR_DIVISION_BY_ZERO` powoduje zmianę tego zachowania. Jeżeli tryb ścisły nie jest włączony, dzielenie przez zero nadal wygeneruje wynik NULL, ale także i ostrzeżenie. Z kolei w trybie ścisłym dzielenie przez zero w zapytaniu `INSERT` lub `UPDATE` spowoduje błąd i wykonanie zapytania zakończy się niepowodzeniem. Aby wyłączyć generowanie błędów dla operacji wstawiania lub uaktualniania danych i wyłączyć generowanie zamiast tego wartości NULL i ostrzeżenia, należy użyć `INSERT IGNORE` lub `UPDATE IGNORE`.

■ **HIGH_NOT_PRECEDENCE**

Ten tryb zmienia pierwszeństwo operatora `NOT` na takie samo jak operatora `!`.

■ **IGNORE_SPACE**

Ten tryb powoduje, że serwer ignoruje spacje między nazwami funkcji wbudowanych i nawiasem otwierającym, który poprzedza listę argumentów. Normalnie po nazwie funkcji natychmiast powinien znajdować się wspomniany nawias otwierający, bez spacji między nimi. Omawiany tryb powoduje, że nazwy funkcji są traktowane jako słowa zarezerwowane.

■ **NO_AUTO_CREATE_USER**

Ten tryb uniemożliwia zapytaniom `GRANT` tworzenie nowych kont użytkowników w postaci niezabezpieczonej. Oznacza to, że jeśli konto użytkownika nie istnieje, to wykonanie zapytania `GRANT` zakończy się niepowodzeniem i nie nastąpi utworzenie nowego konta użytkownika, o ile zapytanie nie zawiera klauzuli `IDENTIFIED BY` wskazującej niepuste hasło lub `IDENTIFIED WITH` wskazującej wtyczkę uwierzytelnienia.

■ **NO_AUTO_VALUE_ON_ZERO**

Normalnie, wstawienie wartości 0 w kolumnie `AUTO_INCREMENT` ma taki sam efekt jak wstawienie wartości NULL: MySQL wygeneruje kolejny numer w sekwencji i umieści go w kolumnie. Po włączeniu omawianego trybu wstawienie wartości 0 do kolumny `AUTO_INCREMENT` faktycznie spowoduje wstawienie zera do takiej kolumny.

■ **NO_BACKSLASH_ESCAPES**

Ten tryb powoduje, że ukośnik `\` nie będzie traktowany jako znak sterujący w ciągach tekstowych, ale jako zwykły znak bez żadnego specjalnego znaczenia.

■ **NO_DIR_IN_CREATE**

Tryb powoduje zignorowanie opcji `DATA DIRECTORY` i `INDEX DIRECTORY` w zapytaniach `CREATE TABLE` i `ALTER TABLE`.

■ **NO_ENGINE_SUBSTITUTION**

Ten tryb określa, w jaki sposób serwer będzie obsługiwał zapytania `CREATE TABLE` lub `ALTER TABLE` zawierające opcję `ENGINE` wymieniającą niedostępny silnik bazy danych. Jeżeli włączony będzie omawiany tryb, nastąpi wygenerowanie błędu, a tabela nie zostanie utworzona (lub zmodyfikowana). W przypadku wyłączenia trybu `NO_ENGINE_SUBSTITUTION` dozwolone będzie użycie domyślnego silnika bazy danych zamiast niedostępnego.

■ **NO_FIELD_OPTIONS**

Ten tryb powoduje, że dane wyjściowe zapytań `SHOW CREATE TABLE` są nieco bardziej przenośne dzięki wyłączeniu wyświetlania charakterystycznych dla MySQL opcji powiązanych z kolumnami.

■ **NO_KEY_OPTIONS**

Ten tryb powoduje, że dane wyjściowe zapytań `SHOW CREATE TABLE` są nieco bardziej przenośne dzięki wyłączeniu wyświetlania charakterystycznych dla MySQL opcji powiązanych z indeksami.

■ **NO_TABLE_OPTIONS**

Ten tryb powoduje, że dane wyjściowe zapytań `SHOW CREATE TABLE` są nieco bardziej przenośne dzięki wyłączeniu wyświetlania charakterystycznych dla MySQL opcji powiązanych z tabelami.

■ **NO_UNSIGNED_SUBTRACTION**

Domyślnie operacja odejmowania między operandami w postaci liczb całkowitych powoduje, że wynik nie ma znaku, jeśli którykolwiek z operandów także nie ma znaku. Włączenie tego trybu powoduje dodanie znaku do wyniku, co zapewnia zgodność z zachowaniem serwera MySQL w wersjach wcześniejszych niż 4.0.

■ **NO_ZERO_DATE**

W trybie ścisłym serwer traktuje wartość `'0000-00-00'` jako nieprawidłową i odrzuca ją. Normalnie MySQL zezwala na przechowywanie „zerowej” wartości daty. Ten tryb można nadpisać przez użycie `INSERT IGNORE` zamiast `INSERT`.

■ **NO_ZERO_IN_DATE**

W trybie ścisłym serwer odrzuca datę, w której dzień lub miesiąc ma wartość zero. (Dozwolone jest użycie zera dla roku). Normalnie MySQL zezwala na przechowywanie tego rodzaju wartości. W trybie nieścisłym lub po użyciu zapytania `INSERT IGNORE` serwer będzie wspomniane daty przechowywał jako `'0000-00-00'`.

■ **ONLY_FULL_GROUP_BY**

Normalnie serwer MySQL zezwala na wykonywanie zapytań SELECT wraz z nieagregacyjnymi kolumnami umieszczanymi na liście kolumn w danych wyjściowych. Podobna sytuacja dotyczy klauzul HAVING, które nie zostały wymienione w klauzuli GROUP BY. Na przykład:

```
SELECT a, b, COUNT(*) FROM t GROUP BY a;
```

Flaga ONLY_FULL_GROUP_BY wymaga, aby kolumny nieagregacyjne w danych wyjściowych (lub kolumny HAVING) zostały wymienione w klauzuli GROUP BY:

```
SELECT a, b, COUNT(*) FROM t GROUP BY a, b;
```

■ **PAD_CHAR_TO_FULL_LENGTH**

Normalnie serwer usuwa spacje znajdujące się na końcu wartości kolumn typu CHAR podczas ich pobierania. Włączenie omawianego trybu powoduje, że serwer nie będzie usuwał wspomnianych spacji i pobrane wartości będą miały długość pełnej kolumny.

■ **PIPES_AS_CONCAT**

Ten tryb powoduje, że operator || będzie traktowany jako operator konkatencji ciągów tekstowych, a nie jako logiczny operator OR.

■ **REAL_AS_FLOAT**

Ten tryb powoduje, że typ danych REAL stanie się synonimem typu FLOAT zamiast DOUBLE.

■ **STRICT_ALL_TABLES**

Ten tryb powoduje włączenie ścisłego sprawdzania wartości danych wejściowych we wszystkich silnikach baz danych i MySQL odrzuci większość nieprawidłowych wartości. Zapewnienie jeszcze ściślejszego sprawdzania jest możliwe przez włączenie trybu TRADITIONAL.

■ **STRICT_TRANS_TABLES**

Ten tryb powoduje włączenie ścisłego sprawdzania wartości danych wejściowych we wszystkich transakcyjnych silnikach baz danych i MySQL odrzuci większość nieprawidłowych wartości. Ponadto, powoduje włączenie ścisłego sprawdzania w silnikach nietransakcyjnych, gdzie tylko będzie to możliwe (na przykład w zapytaniach INSERT dotyczących pojedynczych rekordów). Zapewnienie jeszcze ściślejszego sprawdzania jest możliwe przez włączenie trybu TRADITIONAL.

W tabeli D.3 wymieniono listę złożonych trybów SQL i tworzące je tryby proste.

Nazwa trybu TRADITIONAL wzięła się stąd, że włącza on tryby powodujące obsługę wartości danych wejściowych w taki sposób, jak tradycyjne bazy danych odrzucające nieprawidłowe dane. Ten tryb jest jak tryb ścisły, ale obejmuje kilka dodatkowych ograniczeń, które powodują jeszcze ściślejsze sprawdzanie danych wejściowych.

Tabela D.3. Tryby złożone SQL i tworzące je tryby proste

Tryb złożony	Tryby proste tworzące tryb złożony
ANSI	ANSI_QUOTES, IGNORE_SPACE, PIPES_AS_CONCAT, REAL_AS_FLOAT
DB2	ANSI_QUOTES, IGNORE_SPACE, NO_FIELD_OPTIONS, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, PIPES_AS_CONCAT
MAXDB	ANSI_QUOTES, IGNORE_SPACE, NO_AUTO_CREATE_USER, NO_FIELD_OPTIONS, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, PIPES_AS_CONCAT
MSSQL	ANSI_QUOTES, IGNORE_SPACE, NO_FIELD_OPTIONS, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, PIPES_AS_CONCAT
MYSQL323	HIGH_NOT_PRECEDENCE, NO_FIELD_OPTIONS
MYSQL40	HIGH_NOT_PRECEDENCE, NO_FIELD_OPTIONS
ORACLE	ANSI_QUOTES, IGNORE_SPACE, NO_AUTO_CREATE_USER, NO_FIELD_OPTIONS, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, PIPES_AS_CONCAT
POSTGRESQL	ANSI_QUOTES, IGNORE_SPACE, NO_FIELD_OPTIONS, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, PIPES_AS_CONCAT
TRADITIONAL	ERROR_FOR_DIVISION_BY_ZERO, NO_AUTO_CREATE_USER, NO_ZERO_DATE, NO_ZERO_IN_DATE, STRICT_ALL_TABLES, STRICT_TRANS_TABLES

- **sql_notes** (zakres: globalny, sesji; dynamiczna)

Przypisanie wartości 0 lub 1 (wartość domyślna) tej zmiennej wskazuje, czy serwer wyłącza czy włącza rejestrowanie ostrzeżeń na poziomie Note.

- **sql_quote_show_create** (zakres: globalny, sesji; dynamiczna)

Ta zmienna określa, czy mają być cytowane identyfikatory (baz danych, tabel, kolumn i nazw indeksów) w danych wyjściowych zapytań `SHOW CREATE TABLE` i `SHOW CREATE DATABASE`. Wartością domyślną jest 1, czyli stosowanie cytowania. Wyłączenie cytowania przez przypisanie omawianej zmiennej wartości 0 może być użyteczne podczas tworzenia zapytań `CREATE TABLE` do użycia z innymi serwerami baz danych. Jeżeli cytowanie zostanie wyłączone, upewnij się, że tabele nie stosują nazw w postaci słów zarezerwowanych i nie zawierają znaków specjalnych.

Identyfikatory są cytowane za pomocą odwróconych apostrofów (```), jeśli wyłączony jest tryb SQL `ANSI_QUOTES`. Po włączeniu wymienionego trybu znaki cytowania to cudzysłowy (`"`).

- **sql_safe_updates** (zakres: globalny, sesji; dynamiczna)

Jeżeli wartością tej zmiennej będzie 1, wtedy serwer akceptuje zapytania `UPDATE` i `DELETE` tylko wtedy, gdy rekordy przeznaczone do modyfikacji są wskazane przez wartość kluczy lub jeśli użyta została klauzula `LIMIT`. Wartość domyślna 0 oznacza brak wymienionego ograniczenia.

- `sql_select_limit` (zakres: globalny, sesji; dynamiczna)
Ta zmienna określa maksymalną liczbę rekordów, jaką może zwrócić zapytanie SELECT. Wyraźne umieszczenie klauzuli LIMIT w zapytaniu ma pierwszeństwo przed wartością omawianej zmiennej. Wartością domyślną jest maksymalna liczba rekordów dozwolonych do umieszczenia w tabeli. Wartość DEFAULT powoduje przywrócenie wspomnianej wartości domyślnej, o ile została ona zmieniona. Ta zmienna nie ma efektu w procedurach składowanych i zapytaniach SELECT, które nie zwracają rekordów klientowi (na przykład używanych w charakterze podzapytań INSERT INTO ... SELECT i CREATE TABLE ... SELECT).
- `sql_slave_skip_counter` (zakres: globalny, sesji; dynamiczna)
Jeżeli masz uprawnienie SUPER, wtedy możesz ustawić tę zmienną jako GLOBAL o wartości *n* i nakazać serwerowi podległemu replikacji pominięcie kolejnych *n* zdarzeń otrzymanych z serwera głównego. Jeżeli wynikowe położenie znajdzie się w środku grupy zdarzeń (na przykład tworzących transakcję), wówczas serwer podległy pominię także pozostałe zdarzenia w danej grupie.
- `sql_warnings` (zakres: globalny, sesji; dynamiczna)
Jeżeli ta zmienna będzie miała ustawioną wartość 1, MySQL zgłosi ostrzeżenia nawet w przypadku operacji wstawiania pojedynczego rekordu. W przypadku wartości domyślnej wynoszącej 0 ostrzeżenia są zgłaszane tylko wtedy, gdy zapytanie INSERT obejmuje więcej niż jeden rekord.
- `ssl_XXX`
Zmienne `ssl_XXX` wskazują wartości odpowiadające opcjom `--ssl-XXX` użytym podczas uruchamiania serwera. (Na przykład `ssl_ca` odpowiada wartości opcji `--ssl-ca`). Wartość poszczególnych zmiennych jest pustym ciągiem tekstowym, jeśli odpowiadająca jej opcja nie została użyta. W przypadku braku obsługi SSL wartości zmiennych to NULL.
- `storage_engine` (zakres: globalny, sesji; dynamiczna)
Począwszy od wydania MySQL 5.5.3, ta zmienna jest uznawana za przestarzałą. Zamiast niej należy używać `default_storage_engine`.
- `stored_program_cache` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Zmienna określa liczbę procedur składowanych, jakie są buforowane przez serwer dla każdego połączenia. Wartość domyślna wynosi 256. Ta zmienna została wprowadzona w MySQL 5.5.21.
- `sync_binlog` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Kiedy ta zmienna ma przypisaną wartość zero (to jest wartość domyślna), serwer nie opróżnia binarnego dziennika zdarzeń. Przypisanie zmiennej wartości dodatniej *n* powoduje, że serwer opróżnia dziennik po każdym *n* zapisach w binarnym dzienniku zdarzeń. W takiej sytuacji niższa wartość zapewnia większe bezpieczeństwo w przypadku awarii, ale jednocześnie negatywnie wpływa na wydajność.

- `sync_frm` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Kiedy ta zmienna jest ustawiona (tak jest domyślnie), w trakcie tworzenia tabeli serwer opróżnia plik `.frm` dla każdej tabeli innej niż tymczasowa, aby zsynchronizować zawartość pliku.
- `sync_master_info` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
W przypadku serwera podległego, jeśli omawiana zmienna ma wartość 0 (to jest wartość domyślna), wtedy serwer podległy nie wymusza synchronizacji pliku `master.info`. Zamiast tego stosowane jest zwykle jego opróżnianie przez system plików. Wartość większa niż zero tej zmiennej powoduje, że serwer podległy synchronizuje wymieniony plik.
- `sync_relay_log` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
W przypadku serwera podległego, jeśli omawiana zmienna ma wartość 0 (to jest wartość domyślna), wtedy serwer podległy nie wymusza synchronizacji jego dziennika przekazywania. Zamiast tego stosowane jest zwykle jego opróżnianie przez system plików. Wartość większa niż zero tej zmiennej powoduje, że serwer podległy synchronizuje wspomniany dziennik przekazywania. Jeśli włączone jest automatyczne zatwierdzanie, operacja zapisu występuje raz dla danego polecenia, w przeciwnym razie raz dla transakcji.
- `sync_relay_log_info` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna ma działanie podobne do `sync_master_info`, ale dotyczy pliku `relay-log.info`.
- `system_time_zone` (zakres: globalny)
Strefa czasowa systemu serwera. Serwer próbuje ustalić wartość tej zmiennej podczas uruchamiania. Aby wyraźnie ustawić wartość zmiennej `system_time_zone`, należy ustawić zmienną środowiskową `TZ` lub użyć opcji `--timezone` w trakcie wywoływania skryptu `mysqld_safe`.
- `table_definition_cache` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Zmienna określa liczbę definicji tabel (znajdujących się w plikach `.frm`), które serwer będzie buforował w swoim buforze definicji.
- `table_open_cache` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Zmienna określa maksymalną liczbę tabel, jakie jednocześnie mogą być otwarte.
- `table_open_cache_instances` (startowa: ustawiana bezpośrednio; zakres: globalny)
Zmienna określa liczbę elementów używanych podczas partycjonowania bufora otwartych tabel. Wartość domyślna wynosi 1 (w zasadzie oznacza

brak partycjonowania). Podział bufora może zmniejszyć poziom rywalizacji między sesjami w przypadku dużego obciążenia serwera. Ta zmienna została wprowadzona w MySQL 5.6.6.

- **thread_cache_size** (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa maksymalną liczbę wątków obsługujących wątek bufora. Wątki „odzyskane” od klientów, które zamknęły połączenia, są umieszczane w buforze, o ile nie jest on jeszcze zapelniony. W ten sposób nowe połączenia mogą być obsługiwane za pomocą ponownego wykorzystania buforowanych wątków zamiast tworzenia nowych, oczywiście pod warunkiem dostępności wątków w buforze. Wspomniany bufor wątków jest używany wtedy, gdy serwer używa jednego wątku dla poszczególnych klientów, które nawiązały połączenie.
- **thread_concurrency** (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna jest zbędna.
- **thread_handling** (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna określa model wątku używanego przez serwer do obsługi połączeń z klientami. Wartością zmiennej może być `no-threads` (pojedynczy wątek dla połączenia) lub `one-thread-per-connection` (jeden wątek dla aktualnie połączonego klienta). Wartością domyślną zmiennej jest `one-thread-per-connection`.
- **thread_stack** (startowa: ustawiana bezpośrednio; zakres: globalny)
Zmienna wskazuje wielkość stosu dla poszczególnych wątków.
- **time_format** (zakres: globalny)
Ta zmienna jest nieużywana.
- **time_zone** (startowa: użyj `--default-time-zone`; zakres: globalny, sesji; dynamiczna)
Ta zmienna określa bieżącą strefę czasową serwera. Wartość `SYSTEM` oznacza, że serwer będzie używał wartości zmiennej `system_time_zone`. Klient może modyfikować wartość sesji tej zmiennej i ustawić strefę czasową dla własnej sesji.
- **timestamp** (zakres: sesji; dynamiczna)
Ustawienie tej zmiennej wskazuje wartość `TIMESTAMP` dla bieżącej sesji. Wspomniana wartość jest używana podczas przetwarzania binarnego dziennika zdarzeń. Wpływa również na wartość zwracaną przez funkcję `NOW()`, ale już nie na wartość zwrótną funkcji `SYSDATE()`.
- **tmp_table_size** (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Zmienna wskazuje maksymalną dozwoloną wielkość wewnętrznych tabel tymczasowych (to znaczy tabel automatycznie tworzonych przez serwer podczas przetwarzania zapytań). Jeżeli tabela tymczasowa będzie większa niż mniejsza

wartość spośród `max_heap_table_size` i `tmp_table_size`, wtedy serwer skonwertuje ją z postaci wewnętrznej tabeli w pamięci na tabelę MyISAM zapisaną na dysku. Jeżeli masz dostępną wolną pamięć, wyższa wartość omawianej zmiennej pozwala serwerowi na przechowywanie w pamięci większych tabel tymczasowych bez konieczności ich konwersji na format tabel zapisywanych na dysku.

- `tmpdir` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna zawiera ścieżkę dostępu do katalogu, w którym serwer tworzy pliki tymczasowe. Wartością tej zmiennej może być lista katalogów. W systemach UNIX nazwy katalogów powinny być rozdzielone dwukropkami, natomiast w Windows średnikami.
- `transaction_alloc_block_size` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Wielkość bloku alokowanego w pamięci i wymaganego do przetwarzania zapytań przechowywanych jako część transakcji, zanim transakcja zostanie zapisana w binarnym dzienniku zdarzeń po jej zatwierdzeniu.
- `transaction_prealloc_size` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Ta zmienna określa wielkość bufora alokowanego do przetwarzania zapytań będących częścią transakcji. W przeciwieństwie do bloków alokowanych pod kontrolą zmiennej `transaction_alloc_block_size` wspomniany bufor nie jest opróżniany między wykonaniem poszczególnych zapytań.
- `tx_isolation` (startowa: użyj `--transaction-isolation`; zakres: globalny, sesji; dynamiczna)
Ta zmienna wskazuje domyślny poziom izolacji transakcji.
- `tx_read_only` (startowa: użyj `--transaction-read-only`; zakres: globalny, sesji; dynamiczna)
Ta zmienna określa, czy domyślny tryb dostępu do transakcji jest tylko do odczytu. Wartością domyślną omawianej zmiennej jest `OFF`, czyli transakcja jest w trybie odczytu i zapisu. Ta zmienna została wprowadzona w MySQL 5.6.5.
- `unique_checks` (zakres: globalny, sesji; dynamiczna)
Przypisanie tej zmiennej wartości 0 lub 1 powoduje wyłączenie lub włączenie sprawdzania unikalności drugorzędnych indeksów w tabelach InnoDB. Wyłączenie wspomnianego sprawdzenia może zwiększyć wydajność działania podczas importu danych do tabel InnoDB. Jednak nie powinieneś wyłączać wspomnianego sprawdzenia, o ile nie masz pewności, że wartości spełniają wymagania dotyczące unikalności.
- `updatable_views_with_limit` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Kiedy ta zmienna nie jest ustawiona, serwer uniemożliwia uaktualnienia (zapytania `UPDATE` lub `DELETE`) widoków, które nie używają klucza podstawowego

w tabeli, nawet jeśli uaktualnienie zawiera klauzulę `LIMIT 1`, ograniczającą uaktualnienie do pojedynczego rekordu. Włączenie omawianej zmiennej (tak jest domyślnie) powoduje, że wspomniane uaktualnienie jest dozwolone, a serwer wygeneruje jedynie ostrzeżenie.

- **version** (zakres: globalny)

Ta zmienna wskazuje wersję serwera. Wartość zmiennej składa się z numeru wersji oraz ewentualnie jednego lub więcej przyrostków, takich jak `-log` (wskazuje włączoną rejestrację informacji) lub `-debug` (wskazuje uruchomienie serwera w trybie debugowania).

- **version_comment** (zakres: globalny)

Ta zmienna zawiera wartość opcji `-DWITH_COMMENT` podanej CMake w trakcie konfiguracji serwera przed jego kompilacją. Wartością domyślną omawianej zmiennej jest `Source distribution`, o ile żaden inny komentarz nie został podany w trakcie wspomnianej konfiguracji.

- **version_compile_machine** (zakres: globalny)

Typ komputera, w którym przeprowadzono kompilację serwera. Wartość tej zmiennej jest ustalana w trakcie procesu konfiguracji serwera przed jego kompilacją.

- **version_compile_os** (zakres: globalny)

System operacyjny, w którym przeprowadzono kompilację serwera. Wartość tej zmiennej jest ustalana w trakcie procesu konfiguracji serwera przed jego kompilacją.

- **wait_timeout** (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Zmienna określa liczbę sekund, przez jakie klient nieinteraktywny może pozostać bezczynny, zanim serwer uzna, że może zamknąć połączenie. W przypadku klientów interaktywnych używana jest wartość zmiennej `interactive_timeout`.

- **warning_count** (zakres: sesji)

To jest zmienna sesji tylko do odczytu; wskazuje liczbę błędów, ostrzeżeń i uwag wygenerowanych przez ostatnie zapytanie, które może wygenerować tego rodzaju komunikaty.

D.1.1.1. Zmienne systemowe InnoDB

Wymienione poniżej zmienne systemowe dotyczą silnika bazy danych InnoDB:

- **ignore_builtin_innodb** (zakres: globalny)

Ta zmienna jest nieużywana.

- `innodb_adaptive_flushing` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa, kiedy silnik InnoDB próbuje uniknąć operacji wejścia-wyjścia przez zastosowanie poziomów obciążenia w celu zmiany częstotliwości opróżniania „brudnych” stron w puli bufora. Wartością domyślną tej zmiennej jest 0N.
- `innodb_adaptive_flushing_lwm` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Zmienna określa punkt krytyczny wyrażający w procentach pojemność dziennika zdarzeń, po przekroczeniu którego InnoDB rozpocznie adaptacyjne opróżnianie dziennika. Wartością domyślną jest 10. Ta zmienna została wprowadzona w MySQL 5.6.6.
- `innodb_adaptive_hash_index` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa, kiedy silnik InnoDB używa adaptacyjnych indeksów hash, czyli kiedy rozpoczyna monitorowanie operacji wyszukiwania indeksu i tworzy indeksy typu hash „w locie”, jeśli to może poprawić wydajność. Wartością domyślną zmiennej jest 0N.
- `innodb_adaptive_max_sleep_delay` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa, kiedy silnik InnoDB używa poziomów obciążenia w celu dostosowania `innodb_thread_sleep_delay`. Wartość 0 zmiennej powoduje wyłączenie wspomnianego dostosowywania. Wartość niezerowa aż do maksymalnie 1000000 oznacza wyrażony w mikrosekundach limit dostosowywania `innodb_thread_sleep_delay` przez InnoDB. Wartość domyślna wynosi 150000. Ta zmienna została wprowadzona w MySQL 5.6.3.
- `innodb_additional_mem_pool_size` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna określa wielkość puli pamięci InnoDB przeznaczonej do przechowywania wewnętrznych struktur danych.
- `innodb_autoextend_increment` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Jeżeli systemowa przestrzeń tabel InnoDB jest skonfigurowana jako automatycznie rozszerzająca się po zapelnieniu, ta zmienna wskazuje wielkość, o jaką nastąpi zwiększenie przestrzeni tabel. Wartość zmiennej jest wyrażona w megabajtach, domyślnie wynosi 8, maksymalnie 1000.
- `innodb_autoinc_lock_mode` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna wskazuje algorytm nakładania blokad stosowany przez InnoDB podczas generowania wartości `AUTO_INCREMENT`. Dozwolone wartości wynoszą 0, 1 (wartość domyślna) i 2. Ogólnie rzecz biorąc, wymienione wartości pozwalają na zwiększenie poziomu skalowalności blokad, zapewnienie lepszej współbieżności

i zmniejszenie poziomu blokowania, gdy wiele transakcji jednocześnie generuje wartości `AUTO_INCREMENT`. W przypadku zapytań `INSERT` wstawiających wiele rekordów i tym samym generujących wiele wartości `AUTO_INCREMENT`, tryby 0 i 1 gwarantują otrzymanie kolejnych wartości w zapytaniu. Tryb 2 nie daje takiej gwarancji: pozwala na alokację wartości między zapytaniami i choć wartości wygenerowane dla danego zapytania będą monotoniczne, to niekoniecznie będą kolejnymi w sekwencji. To również ma wpływ na replikację. Tryby 0 i 1 są bezpieczne dla replikacji opartej na zapytaniach, ale alokacja wartości dla trybu 2 jest niedeterministyczna, a tym samym tryb nie jest bezpieczny. (Wszystkie tryby są bezpieczne dla replikacji opartej na rekordach).

- `innodb_buffer_pool_dump_at_shutdown` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Zmienna określa, czy silnik InnoDB będzie w trakcie zamykania serwera zapisywał pulę bufora na dysku. Wartością domyślną zmiennej jest `OFF`. Ta zmienna została wprowadzona w MySQL 5.6.3.
- `innodb_buffer_pool_dump_now` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Przypisanie wartości `ON` tej zmiennej powoduje, że silnik InnoDB przeprowadza pośrednie zapisanie puli bufora na dysku. Wartością domyślną zmiennej jest `OFF`. Ta zmienna została wprowadzona w MySQL 5.6.3.
- `innodb_buffer_pool_filename` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna wskazuje nazwę pliku używanego przez InnoDB podczas zapisu puli bufora na dysku oraz w trakcie jego wczytywania. Domyślnie to jest `ib_buffer_pool` w katalogu danych MySQL. Ta zmienna została wprowadzona w MySQL 5.6.3.
- `innodb_buffer_pool_instances` (startowa: ustawiana bezpośrednio; zakres: globalny)
Zmienna określa liczbę obszarów, na które zostanie podzielona pula bufora InnoDB, o ile wartość `innodb_buffer_pool_size` wynosi przynajmniej 1 GB. Wartością domyślną zmiennej jest 1 (pojedyncza pula), natomiast maksymalna to 64. W celu uzyskania najlepszych wyników należy w taki sposób ustalać wartości `innodb_buffer_pool_size` i `innodb_buffer_pool_instances`, aby każdy obszar miał wielkość co najmniej 1 GB. Ta zmienna została wprowadzona w MySQL 5.5.4.
- `innodb_buffer_pool_load_abort` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Przypisanie wartości `ON` tej zmiennej powoduje, że InnoDB przerwie aktualną operację wczytywania puli bufora z dysku. Wartością domyślną zmiennej jest `OFF`. Ta zmienna została wprowadzona w MySQL 5.6.3.

- `innodb_buffer_pool_load_at_startup` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna określa, czy silnik InnoDB będzie przeprowadzał operację wczytywania puli bufora z dysku podczas uruchamiania serwera. Wartością domyślną zmiennej jest OFF. Ta zmienna została wprowadzona w MySQL 5.6.3.
- `innodb_buffer_pool_load_now` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Przypisanie wartości ON tej zmiennej powoduje, że InnoDB przeprowadzi operację pośredniego wczytywania puli bufora z dysku. Wartością domyślną zmiennej jest OFF. Ta zmienna została wprowadzona w MySQL 5.6.3.
- `innodb_buffer_pool_size` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna określa wielkość bufora InnoDB przeznaczonego do buforowania danych tabeli i indeksów. Wartość domyślna zmiennej wynosi 128 MB.
- `innodb_change_buffer_max_size` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa maksymalną procentową wielkość puli bufora rezerwowaną na zmianę buforowania. Wartość domyślna wynosi 25. Dozwolone wartości mieszczą się w zakresie od 0 do 50. Ta zmienna została wprowadzona w MySQL 5.6.2.
- `innodb_change_buffering` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa, jak InnoDB buforuje zmiany w opóźnionych operacjach zapisu dla indeksów drugorzędnych. Dzięki buforowaniu operacji zapisu silnik InnoDB czasami może je buforować i przeprowadzać jako sekwencyjne, a nie losowe operacje wejścia-wyjścia. Operacje sekwencyjne charakteryzują się lepszą wydajnością. Dozwolone wartości omawianej zmiennej wymieniono w tabeli D.4.

Tabela D.4. Dozwolone wartości zmiennej `innodb_change_buffering`

Wartość	Opis
<code>all</code>	Buforowanie wszystkich zmian, to jest wartość domyślna.
<code>none</code>	Nie będą buforowane żadne zmiany.
<code>changes</code>	Buforowanie operacji wstawiania i usuwania danych.
<code>deletes</code>	Buforowanie zmian, które powodują oznaczenie rekordów indeksu jako przeznaczonych do usunięcia.
<code>inserts</code>	Buforowanie operacji wstawiania danych.
<code>purges</code>	Buforowanie operacji usuwania danych wskazanych przez mechanizm usuwania nieużytków jako przeznaczonych do usunięcia.

- `innodb_checksum_algorithm` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Silnik InnoDB zapisuje sumę kontrolną wartości w każdym bloku przestrzeni tabel. Ta zmienna wskazuje algorytm używany do obliczenia wspomnianej sumy kontrolnej. Dozwolone wartości zmiennej wymieniono w tabeli D.5. Wartości `strict_xxx` działają jak ich odpowiedniki bez przyrostka `strict_` za wyjątkiem faktu, że InnoDB zatrzymuje je, jeśli w przestrzeni tabel znajdzie wiele typów sum kontrolnych.

Tabela D.5. Dozwolone wartości zmiennej `innodb_checksum_algorithm`

Wartość	Opis
<code>none</code> , <code>strict_none</code>	Zapis stałej wartości jako sumy kontrolnej.
<code>crc32</code> , <code>strict_crc32</code>	Użycie algorytmu CRC32.
<code>innodb</code> , <code>strict_innodb</code>	Algorytm domyślny.

Omawiana zmienna została wprowadzona w MySQL 5.6.3.

- `innodb_checksums` (startowa: ustawiana bezpośrednio; zakres: globalny)
Zmienna wskazuje, czy włączone jest obliczanie sumy kontrolnej dla tabel InnoDB. Wartością domyślną zmiennej jest `ON`. Począwszy od wersji MySQL 5.6.3, zmienna jest uznana za przestarzałą, zamiast niej należy stosować `innodb_checksum_algorithm`.
- `innodb_commit_concurrency` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa, ile wątków może jednocześnie przeprowadzać zatwierdzenie transakcji. Wartość `0` (to jest wartość domyślna) oznacza brak ograniczenia.
- `innodb_concurrency_tickets` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Jeżeli wątek chce się dostać do InnoDB, może to zrobić tylko wtedy, gdy liczba wątków jest mniejsza niż wskazywana przez zmienną `innodb_thread_concurrency`. W przeciwnym razie wątek jest kolejkowany aż do chwili, gdy liczba wątków spadnie poniżej limitu wyznaczonego przez wymienioną zmienną. Jeżeli wątek może się dostać do InnoDB, ma możliwość opuszczenia i następnie ponownego wejścia do InnoDB bez żadnych ograniczeń i maksymalnie liczbę razy wskazaną przez zmienną `innodb_concurrency_tickets`. Wartością domyślną omawianej zmiennej jest `500`.
- `innodb_data_file_path` (startowa: ustawiana bezpośrednio; zakres: globalny)
Zmienna określa specyfikację dla plików komponentu przestrzeni tabel InnoDB. W podpunkcie 12.5.3.1, zatytułowanym „Konfiguracja przestrzeni tabel InnoDB”, znajdziesz informacje o formacie wartości omawianej zmiennej.

- `innodb_data_home_dir` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna zawiera ścieżkę dostępu do katalogu, względem którego będą umieszczane pliki komponentu przestrzeni tabel InnoDB. Jeżeli wartość tej zmiennej będzie pusta, nazwy plików komponentu będą interpretowane jako bezwzględne ścieżki dostępu.
- `innodb_doublewrite` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna określa, czy włączony jest bufor podwójnego zapisu InnoDB. Wartością domyślną zmiennej jest ON.
- `innodb_fast_shutdown` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Wartość 0 lub 1 tej zmiennej wskazuje, czy InnoDB używa szybszej metody zamknięcia, pomijającej pewne operacje, które normalnie są wykonywane. W przypadku wartości 2 omawianej zmiennej silnik InnoDB opróżnia dzienniki zdarzeń i kończy działanie.
- `innodb_file_format` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna wskazuje format używany dla nowych tabel InnoDB, jeśli ustawiona jest zmienna `innodb_file_per_table`. Formatem domyślnym jest Antelope, inna dozwolona wartość to Barracuda. Wartości Barracuda używaj do włączenia funkcji nieobsługiwanych przez format Antelope, na przykład formatu COMPRESSED dla rekordów.
- `innodb_file_format_check` (startowa: ustawiana bezpośrednio; zakres: globalny)
Systemowa przestrzeń tabel InnoDB zawiera flagę wskazującą najwyższy format pliku używany w przestrzeni tabel. Wartość omawianej zmiennej w momencie uruchamiania serwera określa, czy silnik InnoDB sprawdza wspomnianą flagę w celu ustalenia, czy format jest wyższy od wersji obsługiwanej przez InnoDB. Jeżeli omawiana zmienna jest ustawiona (tak jest domyślnie) i format będzie wyższy, wtedy wykonanie plików startowych zakończy się niepowodzeniem. Z kolei jeśli format nie jest wyższy, InnoDB ustawi zmiennej `innodb_file_format_max` wskazany format.
- `innodb_file_format_max` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Patrz opis zmiennej `innodb_file_format_check`.
- `innodb_file_io_threads` (startowa: ustawiana bezpośrednio)
Ta zmienna nie jest używana.
- `innodb_file_per_table` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Jeżeli wartość zmiennej wynosi 0 (to jest wartość domyślna), wtedy silnik InnoDB tworzy każdą nową tabelę w systemowej przestrzeni tabel. Wartość 1 omawianej

zmiennej powoduje, że silnik InnoDB używa oddzielnych przestrzeni tabel dla poszczególnych tabel. W takim przypadku każda nowa tabela otrzymuje własny plik *.ibd* w katalogu bazy danych, w wymienionym pliku będzie przechowywana zawartość tabeli. Wówczas systemowa przestrzeń tabel jest używana jedynie do przechowywania katalogu danych InnoDB, a nie dla danych lub indeksów. Omawiana zmienna wpływa jedynie na sposób tworzenia przez InnoDB *nowych* tabel. Dostęp do już istniejących w systemowej przestrzeni tabel bądź poszczególnych przestrzeniach tabel odbywa się niezależnie od zmian wartości zmiennej `innodb_file_per_table`.

- `innodb_flush_log_at_trx_commit` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Ta zmienna określa sposób, w jaki InnoDB opróżnia dzienniki zdarzeń. Dozwolone wartości zmiennej zostały wymienione w tabeli D.6.

Tabela D.6. Dozwolone wartości zmiennej `innodb_flush_log_at_trx_commit`

Wartość	Opis
0	Zapis dziennika zdarzeń i zapis jego zawartości na dysku raz na sekundę.
1	Zapis dziennika zdarzeń i zapis jego zawartości na dysku w trakcie każdego zatwierdzenia transakcji.
2	Zapis w dzienniku zdarzeń w trakcie każdego zatwierdzenia transakcji, natomiast zapis na dysku raz na sekundę.

Silnik InnoDB gwarantuje zachowanie właściwości ACID jedynie wtedy, gdy omawiana zmienna ma wartość 1 (to jest wartość domyślna). W przeciwnym razie w przypadku awarii można utracić transakcje przeprowadzone w ciągu ostatniej sekundy. Przypisanie zmiennej wartości 0 powoduje zmniejszenie ilości danych dziennika zdarzeń zapisywanych na dysku. Jednak to wiąże się z większym ryzykiem utraty ostatnio zatwierdzonych transakcji, jeśli wystąpi awaria.

- `innodb_flush_method` (startowa: ustawiana bezpośrednio; zakres: globalny)

Ta zmienna wskazuje używaną przez InnoDB metodę opróżniania plików. Ma zastosowanie jedynie w systemach UNIX. Dozwolone wartości zmiennej to: `fdatasync` (użycie funkcji `fsync()` do opróżniania plików danych i dzienników zdarzeń), `O_DSYNC` (użycie funkcji `fsync()` do opróżniania plików danych i `O_SYNC` do otwierania i opróżniania dzienników zdarzeń) i `O_DIRECT` (użycie funkcji `fsync()` do opróżniania plików danych i dzienników zdarzeń oraz `O_DIRECT` lub `directio()` do otwierania plików danych, o ile wymieniona funkcja będzie dostępna). Wartością domyślną zmiennej jest `fdatasync`. W systemie Windows wartością omawianej zmiennej zawsze jest `async_unbuffered`.

- `innodb_flush_neighbors` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Ta zmienna określa, czy silnik InnoDB opróżnia „brudne” strony z puli bufora oddzielnie, czy wraz z sąsiednimi stronami umieszczonymi w tej samej grupie. W przypadku usuwania stron wraz z sąsiednimi można połączyć operacje zapisu i tym samym zmniejszyć obciążenie związane z czasem wyszukiwania i przesunięcia głowicy do właściwego miejsca (to dotyczy urządzeń mechanicznych, takich jak dyski twarde). Ta zmienna została wprowadzona w MySQL 5.6.3 i ma wartość boolowską (domyślnie 0N). Począwszy od wydania MySQL 5.6.6, omawiana zmienna ma trzy stany dozwolonych wartości: 0 (brak opróżniania sąsiednich stron), 1 (opróżnianie sąsiadujących stron) i 2 (opróżnianie wszystkich sąsiednich stron do pewnej granicy).

- `innodb_force_load_corrupted` (startowa: ustawiana bezpośrednio; zakres: globalny)

Ta zmienna określa, czy silnik InnoDB będzie w trakcie uruchamiania wczytywać tabele oznaczone jako uszkodzone. Wartością domyślną zmiennej jest OFF, a więc tego rodzaju tabele są ignorowane. Ustawienie wartości 0N omawianej zmiennej może pozwolić na przeprowadzenie operacji odzyskania danych, do których nie ma dostępu, gdy zmienna ma wartość OFF. Po przeprowadzeniu operacji odzyskiwania danych zmiennej należy znów przypisać wartość OFF i ponownie uruchomić serwer. Więcej informacji na ten temat znajdziesz w punkcie 14.7.4, zatytułowanym „Rozwiązywanie problemów związanych z automatyczną naprawą w InnoDB”. Ta zmienna została wprowadzona w MySQL 5.5.18.

- `innodb_force_recovery` (startowa: ustawiana bezpośrednio; zakres: globalny)

Ta zmienna ma normalnie wartość 0, ale można jej przypisać wartość z zakresu od 1 do 6 i tym samym spowodować uruchomienie serwera po awarii, nawet jeśli operacja odzyskiwania danych zakończyła się niepowodzeniem. Informacje na temat sposobu użycia omawianej zmiennej znajdziesz w punkcie 14.7.4, zatytułowanym „Rozwiązywanie problemów związanych z automatyczną naprawą w InnoDB”.

- `innodb_ft_xxx`

Przed wydaniem MySQL 5.6.4 wyszukiwanie pełnego tekstu było możliwe jedynie w tabelach MyISAM. Zmienne systemowe rozpoczynające się od przyrostka `innodb_ft` dotyczą obsługi wyszukiwania pełnego tekstu w tabelach InnoDB, co wprowadzono w MySQL 5.6.4. Więcej informacji na ten temat znajdziesz w podręczniku użytkownika MySQL.

- `innodb_io_capacity` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Ta zmienna określa przybliżony limit operacji wejścia-wyjścia na sekundę przeprowadzanych przez InnoDB dla zadań wykonywanych w tle. Wartością

domyślną jest 200, minimum wynosi 100. W przypadku wolnych, tradycyjnych dysków twardych możesz zmniejszyć tę wartość. Natomiast w przypadku napędów SSD można zwiększyć wartość omawianej zmiennej. Patrz także opis zmiennej `innodb_io_capacity_max`.

- `innodb_io_capacity_max` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Jeżeli wartość zmiennej `innodb_io_capacity` nie jest wystarczająco wysoka podczas sytuacji wyjątkowych, zmienna `innodb_io_capacity_max` określa wartość maksymalną, do której InnoDB może podnieść limit. Wartość domyślna wynosi dwukrotność wartości domyślnej zmiennej `innodb_io_capacity`, przy czym serwer używa wartości minimum 2000. Omawiana zmienna została wprowadzona w MySQL 5.6.6.

- `innodb_large_prefix` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)

Zwykła maksymalna długość prefiksu dla indeksów InnoDB wynosi 767 bajtów. Przypisanie wartości tej zmiennej pozwala na stosowanie prefiksów o długości do 3072 bajtów dla tabel stosujących format rekordu COMPRESSED lub DYNAMIC. Wartością domyślną jest OFF. Ta zmienna została wprowadzona w MySQL 5.5.14.

- `innodb_lock_wait_timeout` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)

Ta zmienna określa liczbę sekund, przez jakie InnoDB czeka na nałożenie blokady dla transakcji. Jeżeli blokady nie uda się nałożyć, wtedy InnoDB wycofuje transakcję.

- `innodb_locks_unsafe_for_binlog` (startowa: ustawiana bezpośrednio; zakres: globalny)

Ta zmienna wpływa na to, jak silnik InnoDB obsługuje nakładanie blokad na rekordy indeksu, ale począwszy od wersji MySQL 5.6.3, jest uznawana za przestarzałą. Aby uzyskać taki sam efekt jak użycie omawianej zmiennej (mniej ściśle nakładanie blokad), ale w znacznie elastyczniejszy sposób (na poziomie sesji lub dla transakcji), należy wykonać zapytanie `SET TRANSACTION` i ustawić poziom izolacji o nazwie `READ COMMITTED`. Patrz punkt 2.12.3, zatytułowany „Izolacja transakcji”.

- `innodb_log_buffer_size` (startowa: ustawiana bezpośrednio; zakres: globalny)

Ta zmienna określa wielkość bufora dziennika zdarzeń transakcji. Jej wartość z reguły jest z zakresu od 1 MB do 8 MB. Wartość domyślna zmiennej wynosi 1 MB.

- `innodb_log_file_size` (startowa: ustawiana bezpośrednio; zakres: globalny)

Ta zmienna określa wielkość każdego pliku dziennika InnoDB. Iloczyn wartości zmiennych `innodb_log_file_size` i `innodb_log_files_in_group` wskazuje całkowitą wielkość dziennika zdarzeń InnoDB.

- `innodb_log_files_in_group` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna wskazuje liczbę plików dziennika InnoDB. Iloczyn wartości zmiennych `innodb_log_file_size` i `innodb_log_files_in_group` określa całkowitą wielkość dziennika zdarzeń InnoDB.
- `innodb_log_group_home_dir` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna wskazuje ścieżkę dostępu do katalogu, w którym silnik InnoDB powinien zapisywać pliki tworzące dziennik zdarzeń.
- `innodb_lru_scan_depth` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Silnik InnoDB wykorzystuje operacje działające w tle do wyszukiwania „brudnych” stron przeznaczonych do usunięcia z puli bufora. Ta zmienna określa głębokość, do jakiej na liście stron (posortowanych w kolejności od najrzadziej używanych) zejdzie operacja wyszukiwania. Wartość domyślna omawianej zmiennej wynosi 1024. Rozsądne będzie zmniejszenie tej wartości dla serwerów bardzo obciążonych i mających ogromną pulę bufora lub też jej zwiększenie dla serwerów oferujących większą przepustowość operacji wejścia-wyjścia. Omawiana zmienna została wprowadzona w MySQL 5.6.3.
- `innodb_max_dirty_pages_pct` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa procentową ilość „brudnych” stron dozwolonych przez InnoDB do istnienia w puli bufora, zanim wystąpi konieczność zapisu dziennika zdarzeń na dysku. Dozwolone wartości mieszczą się w zakresie od 0 do 100, wartość domyślna wynosi 90.
- `innodb_max_dirty_pages_pct_lwm` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Zmienna określa punkt krytyczny wyrażający w procentach ilość „brudnych” stron w puli bufora, po przekroczeniu którego InnoDB rozpocznie ich wcześniejszy zapis na dysku w celu zmniejszenia współczynnika. Wartością domyślną jest 0, która oznacza brak wcześniejszego zapisu. Dozwolone wartości są z zakresu od 0 do 99. Ta zmienna została wprowadzona w MySQL 5.6.6.
- `innodb_max_purge_lag` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Silnik InnoDB zawiera pewną grupę wątków odpowiedzialnych za usuwanie rekordów przeznaczonych do usunięcia na skutek przeprowadzonych operacji usunięcia lub uaktualnienia danych. W sytuacji, gdy mała grupa rekordów jest wstawiana i usuwana mniej więcej w takiej samej wielkości, istnieje prawdopodobieństwo, że wątek usuwający nie będzie nadążał z działaniem. Wówczas duża liczba rekordów przeznaczonych do usunięcia będzie

niepotrzebnie zajmowała miejsce w tabeli. Zmienna `innodb_max_purge_lag` określa opóźnienie wykonywania zapytań INSERT, UPDATE i DELETE, aby wątek odpowiedzialny za usuwanie rekordów mógł działać znacznie efektywniej. Wartość domyślna wynosi 0 (to znaczy brak opóźnienia). W przypadku wartości niezerowych opóźnienie jest proporcjonalne do $((n / \text{innodb_max_purge_lag}) \times 10) - 5$ milisekund, gdzie n oznacza liczbę transakcji, w których rekordy są oznaczane jako do usunięcia.

- `innodb_max_purge_lag_delay` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa w milisekundach opóźnienie, któremu może podlegać `innodb_max_purge_lag`. Wartość domyślna wynosi 0 (brak opóźnienia). Omawiana zmienna została wprowadzona w MySQL 5.6.5.
- `innodb_mirrored_log_groups` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna określa liczbę grup plików dziennika zdarzeń InnoDB, które będą obsługiwane. Wartość domyślna zawsze powinna wynosić 1.
- `innodb_monitor_disable`, `innodb_monitor_enable`, `innodb_monitor_reset`, `innodb_monitor_reset_all` (startowa: ustawiana bezpośrednio; zakres: globalny)
Wymienione zmienne nadzorują działanie tabeli `INFORMATION_SCHEMA.innodb_metrics`. Dokładne informacje na ten temat znajdziesz w podręczniku użytkownika MySQL. Zmienne zostały wprowadzone w MySQL 5.6.2.
- `innodb_old_blocks_pct` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa procentową wielkość bufora przeznaczoną na podlistę starych wartości. Domyślnie to 37% (3/8 puli). Dozwolone jest użycie wartości z zakresu od 5 do 95%. Więcej informacji na ten temat znajdziesz w podpunkcie 12.7.2.1, zatytułowanym „Konfiguracja puli bufora InnoDB”.
- `innodb_old_blocks_time` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa liczbę milisekund, które muszą upłynąć, zanim blok wstawiony do podlisty starych wartości po jego pierwszym użyciu pozostanie tam, nim zostanie przeniesiony do podlisty nowych w przypadku jego kolejnego użycia. Wartość domyślna wynosi zero. Bloki wstawione do podlisty starych wartości są natychmiast przenoszone do podlisty nowych po ich pierwszym użyciu. Więcej informacji na ten temat znajdziesz w podpunkcie 12.7.2.1, zatytułowanym „Konfiguracja puli bufora InnoDB”.
- `innodb_open_files` (startowa: ustawiana bezpośrednio; zakres: globalny)
Jeżeli wartość zmiennej `innodb_file_per_table` wynosi 1 w celu włączenia oddzielnych przestrzeni tabel dla poszczególnych tabel, wtedy zmienna `innodb_open_files` określa liczbę deskryptorów plików, jakie równocześnie

mogą być używane przez InnoDB, aby obsługiwać jednocześnie otwarte pliki *.ibd*. Wartość minimalna wynosi 10, natomiast wartość domyślna to 300. Zmienna `innodb_file_per_table` nadzoruje alokację deskryptorów plików oddzielnie od nadzorowanych przez zmienną `open_files_limit`. Deskryptory używane dla plików *.ibd* nie są używane przez bufor tabeli.

- `innodb_page_size` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna określa wielkość stron w przestrzeniach tabel InnoDB. Wartość domyślna wynosi 16 KB. Dozwolone wartości to 4 KB, 8 KB i 16 KB. Wartość omawianej zmiennej ma efekt tylko wtedy, gdy InnoDB inicjalizuje przestrzenie tabel. Dlatego też powinien ją ustawić jeszcze przez inicjalizację MySQL lub przed usunięciem i ponownym utworzeniem plików przestrzeni tabel InnoDB. Ta zmienna została wprowadzona w MySQL 5.6.4.
- `innodb_print_all_deadlocks` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa, czy silnik InnoDB będzie zapisywał w dzienniku błędów informacje diagnostyczne o zakleszczeniach w transakcji. Wartość domyślna to OFF. Ta zmienna została wprowadzona w MySQL 5.6.2.
- `innodb_purge_batch_size` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa liczbę rekordów, które powodują, że operacja usuwania danych spowoduje zapis na dysku zmienionych bloków puli bufora. Wartość domyślna wynosi 20. Ta zmienna została wprowadzona w MySQL 5.5.4.
- `innodb_purge_threads` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna określa, ilu wątków działających w tle silnik InnoDB używa do przeprowadzania operacji usuwania danych (czyli usuniętych rekordów, które nie są dłużej wymagane przez jakąkolwiek transakcję). Wartość domyślna wynosi 0. Ta zmienna została wprowadzona w MySQL 5.5.4.
- `innodb_random_read_ahead` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa, czy silnik InnoDB próbuje przewidzieć, kiedy strony znajdujące się w pewnym zakresie (grupa stron) będą potrzebne, i przeprowadzi asynchroniczną operację odczytu z wyprzedzeniem takich stron. Przewidywanie odbywa się na podstawie ustalenia, czy inne strony w danym zakresie zostały odczytane, niezależnie od kolejności takiego odczytu. Wartością domyślną zmiennej jest OFF.
- `innodb_read_ahead_threshold` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Jeżeli silnik InnoDB wykryje wzorec w sekwencyjnym dostępie do stron dla `innodb_read_ahead_threshold` lub większej liczby stron z tego samego zakresu (grupy stron), wtedy przeprowadzi asynchroniczną operację odczytu

z wyprzedzeniem stron znajdujących się w kolejnym zakresie. Wartością domyślną zmiennej jest 56. Dozwolone wartości zmiennej mieszczą się w przedziale od 0 do 64.

- `innodb_read_io_threads` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna określa liczbę wątków używanych przez InnoDB dla operacji odczytu. Zakres dozwolonych wartości wynosi od 1 do 64, wartość domyślna to 4.
- `innodb_replication_delay` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Jeżeli w serwerze podległym został osiągnięty limit wskazywany przez zmienną `innodb_thread_concurrency`, wtedy omawiana zmienna wyraża w milisekundach opóźnienie wątku replikacji. Wartością domyślną jest 0.
- `innodb_rollback_on_timeout` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna określa, co robi InnoDB, gdy nastąpi przekroczenie czasu transakcji. W przypadku wartości 0FF (to jest wartość domyślna) silnik InnoDB wycofa jedynie ostatnie zapytanie. Jeżeli wartością omawianej zmiennej będzie 0N, wtedy InnoDB wycofa całą transakcję.
- `innodb_rollback_segments` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa liczbę segmentów wycofania używanych przez InnoDB w systemowej przestrzeni tabel w ramach transakcji. Wartość domyślna wynosi 128. Ta zmienna została wprowadzona w MySQL 5.5.11, natomiast w wersji 5.6.3 została zastąpiona przez `innodb_undo_logs`.
- `innodb_sort_buffer_size` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna wyraża w bajtach wielkość bufora używanego przez InnoDB przez połączone sortowanie w trakcie tworzenia indeksu. Wartością domyślną jest 1 MB. Wartość minimalna wynosi 512 KB w MySQL 5.6.4 i 64 KB w wersji 5.6.5 oraz nowszych. Ta zmienna została wprowadzona w MySQL 5.6.4. Przed wydaniem MySQL 5.6.4 używana była stała wielkość wynosząca 1 MB.
- `innodb_spin_wait_delay` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa maksymalną długość oczekiwania między sprawdzaniem stanu blokady. Ta wartość jest pozbawiona jednostki, ale większa oznacza dłuższe oczekiwanie. Wartość domyślna wynosi 6, natomiast minimalna to 0.
- `innodb_stats_method` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa, czy serwer uznaje wartości NULL za równe czy nierówne sobie podczas obliczenia danych statystycznych dotyczących rozproszenia wartości klucza dla tabel InnoDB. Wartością zmiennej może być `nulls_equal`

(wszystkie wartości NULL są w tej samej grupie), `nulls_unequal` (każda wartość NULL tworzy oddzielną grupę) lub `nulls_ignored` (wartości NULL są ignorowane).

- `innodb_stats_on_metadata` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa, czy silnik InnoDB uaktualnia dane statystyczne dla zapytań uzyskujących dostęp do metadanych tabeli, na przykład `SHOW INDEX` lub `SHOW TABLE STATUS` bądź też uzyskujących dostęp do tabel `STATISTICS` i `INFORMATION_SCHEMA`. Efekt działania zmiennej jest podobny do wykonania zapytania `ANALYZE TABLE`. Wartość domyślna zmiennej wynosi 0N.
- `innodb_stats_persistent_sample_pages` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa liczbę stron indeksu próbkowanych przez InnoDB w celu oszacowania danych statystycznych. Wartość domyślna wynosi 20. Ta zmienna jest ignorowana, o ile nie zostanie ustawiona również zmienna `innodb_analyze_is_persistent`. W przeciwnym razie silnik InnoDB używa wartości `innodb_stats_transient_sample_pages`. Omawiana zmienna została wprowadzona w MySQL 5.6.2.
- `innodb_stats_sample_pages` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
W wydaniu MySQL 5.5 ta zmienna określa liczbę stron indeksu próbkowanych przez silnik InnoDB w celu oszacowania danych statystycznych. Wartość domyślna zmiennej wynosi 8. Począwszy od wersji MySQL 5.6.3, ta zmienna została uznana za przestarzałą i zastąpiona zmienną o nazwie `innodb_stats_transient_sample_pages`.
- `innodb_stats_transient_sample_pages` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa liczbę stron indeksu próbkowanych przez InnoDB w celu oszacowania danych statystycznych. Wartość domyślna wynosi 8. Ta zmienna jest ignorowana, o ile zmienna `innodb_analyze_is_persistent` nie będzie ustawiona. W przeciwnym razie silnik InnoDB używa wartości `innodb_stats_transient_sample_pages`. Omawiana zmienna została wprowadzona w MySQL 5.6.2.
- `innodb_strict_mode` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Ta zmienna określa, czy silnik InnoDB będzie zachowywał się ściślej w kwestii składni zapytań tworzenia tabeli i indeksów, a także ich zmiany. Jeżeli omawiana zmienna zostanie ustawiona, InnoDB wszelkie konflikty w klauzulach wspomnianych zapytań traktuje jako błędy. W przeciwnym razie są one traktowane jako ostrzeżenia. Działanie zmiennej jest podobne do włączenia trybu ścisłego SQL.

- `innodb_support_xa` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Ta zmienna określa, czy silnik InnoDB obsługuje dwufazowe zatwierdzenia w transakcjach XA. Wartość domyślna wynosi ON, ale zmiennej można przypisać wartość OFF, jeśli nie korzystasz z transakcji XA. Wartość OFF może nieco zwiększyć wydajność działania silnika.
- `innodb_sync_spin_loops` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa, ile razy wątek będzie oczekiwał, aż InnoDB zwolni muteks. Przekroczenie określonego czasu powoduje zawieszenie wątku.
- `innodb_table_locks` (startowa: ustawiana bezpośrednio; zakres: globalny, sesji; dynamiczna)
Ta zmienna określa sposób obsługi przez InnoDB zapytań `LOCK TABLE` odpowiedzialnych za nałożenie blokady zapisu na tabelę InnoDB, gdy wyłączony jest tryb automatycznego zatwierdzania. Wartość ON (to jest wartość domyślna) powoduje, że InnoDB nakłada wewnętrzną blokadę tabeli. Z kolei wartość OFF powoduje, że InnoDB czeka, aż wszystkie inne wątki zwolnią blokadę nałożoną na tabelę. Jeżeli omawiana zmienna nie będzie ustawiona, może uchronić przed niektórymi zakleszczeniami w przypadku aplikacji wykonujących zapytania `LOCK TABLES`, gdy wyłączony jest tryb automatycznego zatwierdzania.
- `innodb_thread_concurrency` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa maksymalną liczbę wątków, jakie próbuje obsługiwać InnoDB. Wartość 0 (to jest wartość domyślna) oznacza brak ograniczenia. Dozwolone są wartości z zakresu od 0 do 1000.
- `innodb_thread_sleep_delay` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna to wyrażony w mikrosekundach czas, przez jaki wątki InnoDB będą w stanie uśpienia przed ich umieszczeniem w kolejce InnoDB. Wartość domyślna wynosi 10000 (10 sekund), z kolei wartość 0 oznacza brak czasu uśpienia.
- `innodb_undo_directory` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna wskazuje nazwę katalogu, w którym silnik InnoDB będzie tworzył dzienniki zdarzeń cofnięcia dla przestrzeni tabel, o ile zmienne `innodb_undo_logs` i `innodb_undo_tablespaces` mają przypisane wartości niezerowe. Wartością domyślną zmiennej jest `.` (kropka), która oznacza domyślny katalog przeznaczony na inne pliki dzienników InnoDB. Omawiana zmienna została wprowadzona w MySQL 5.6.3.
- `innodb_undo_logs` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa liczbę segmentów wycofania używanych przez InnoDB w systemowej przestrzeni tabel w ramach transakcji. Wartość domyślna wynosi 128.

Omawiana zmienna została wprowadzona w MySQL 5.6.3 i zastąpiła zmienną `innodb_rollback_segments`.

- `innodb_undo_tablespace` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna określa liczbę plików przestrzeni tabel używanych przez InnoDB dla oddzielnych dzienników wycofania. Wartość domyślna wynosi 0.
Omawiana zmienna została wprowadzona w MySQL 5.6.3.
- `innodb_use_native_aio` (startowa: ustawiana bezpośrednio; zakres: globalny)
W systemie Linux ta zmienna wskazuje, czy używany będzie asynchroniczny podsystem operacji wejścia-wyjścia. Wartość domyślna wynosi 0N. Omawiana zmienna została wprowadzona w MySQL 5.5.4.
- `innodb_use_sys_malloc` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna określa, czy InnoDB używa systemowego mechanizmu alokacji pamięci. Wartość domyślna wynosi 0N. Przypisanie zmiennej wartości 0FF powoduje, że InnoDB użyje własnego mechanizmu alokacji.
- `innodb_version` (zakres: globalny)
Ta zmienna przechowuje numer wersji silnika bazy danych InnoDB.
- `innodb_write_io_threads` (startowa: ustawiana bezpośrednio; zakres: globalny)
Ta zmienna określa liczbę wątków używanych przez InnoDB podczas przeprowadzania operacji zapisu. Dozwolone są wartości z zakresu od 1 do 64. Wartość domyślna wynosi 4.
- `timed_mutexes` (startowa: ustawiana bezpośrednio; zakres: globalny; dynamiczna)
Ta zmienna określa, czy zbierane będą informacje dotyczące muteksu. Wartością domyślną zmiennej jest 0FF.

D.2. Zmienne stanu

Zmienne stanu dostarczają informacje o aktualnym stanie serwera. W celu wyświetlenia zmiennych stanu należy wykonać zapytanie `SHOW STATUS` lub wydać polecenie `mysqladmin extended-status`. Zmienne stanu (podobnie jak zmienne systemowe) mają wartości globalne i sesji. Przedstawiają one wartości odpowiednio dla wszystkich klientów oraz dla bieżącego klienta. Jeżeli zmienna ma jedynie wartość globalną, ta sama wartość będzie zwracana dla zmiennych globalnej i sesji. Informacje dotyczące zmiennej stanu można również pobrać z tabel o nazwach `GLOBAL_STATUS` i `SESSION_STATUS` bazy danych `INFORMATION_SCHEMA`.

Więcej informacji na temat analizy zmiennych stanu w trakcie działania serwera znajdziesz w punkcie 12.3.2, zatytułowanym „Sprawdzenie wartości zmiennej stanu”.

W nazwach zmiennych stanu wielkość liter nie ma znaczenia.

Poniższa lista wymienia ogólne zmienne stanu. Oddzielne sekcje przedstawiają zestawy powiązanych ze sobą zmiennych, między innymi dotyczących silnika bazy danych InnoDB, bufora zapytań i SSL.

- **Aborted_clients**
Ta zmienna określa liczbę połączeń klientów przerwanych na skutek braku poprawnego zamknięcia połączenia przez klienta.
- **Aborted_connects**
Ta zmienna określa liczbę nieudanych prób nawiązania połączenia z serwerem.
- **Binlog_cache_disk_use**
Ta zmienna określa liczbę transakcji, które używały pliku tymczasowego na dysku, ponieważ ich wielkość przekroczyła wartość zmiennej systemowej `binlog_cache_size`.
- **Binlog_cache_use**
Ta zmienna określa liczbę transakcji, które mogły być przechowywane w buforze binarnego dziennika zdarzeń, ponieważ ich wielkość nie przekroczyła wartości zmiennej systemowej `binlog_cache_size`.
- **Binlog_stmt_cache_disk_use**
Ta zmienna określa liczbę zapytań nietransakcyjnych przechowywanych w pliku tymczasowym z powodu wypełnienia bufora zapytań. Ta zmienna została wprowadzona w MySQL 5.5.9.
- **Binlog_stmt_cache_use**
Ta zmienna określa liczbę zapytań nietransakcyjnych przechowywanych w buforze zapytań. Ta zmienna została wprowadzona w MySQL 5.5.9.
- **Bytes_received**
Ta zmienna określa liczbę bajtów otrzymanych od wszystkich klientów (wartość globalna) lub bieżącego klienta (wartość sesji).
- **Bytes_sent**
Ta zmienna określa liczbę bajtów wysłanych do wszystkich klientów (wartość globalna) lub bieżącego klienta (wartość sesji).
- **Com_xxx**
Serwer obsługuje zestaw zmiennych stanu służących w charakterze liczników wskazujących liczbę wykonań danego typu zapytań (poleceń). Tego rodzaju zmiennych są dziesiątki, wszystkie mają podobne nazwy i nie będą tutaj wymienione. Każda nazwa zmiennej rozpoczyna się przedrostkiem `Com_` i ma przyrostek wskazujący typ zapytania, któremu odpowiada. Na przykład, zmienne `Com_select` i `Com_drop_table` odpowiadają liczbie wykonanych zapytań odpowiednio `SELECT` i `DROP TABLE`.
- **Compression**
Ta zmienna określa, czy ruch sieciowy protokołu klient-serwer będzie używał kompresji.

- **Connection_errors_xxx**
Tego rodzaju zmienne śledzą różnego rodzaju błędy pojawiające się w trakcie prób nawiązania połączenia przez klienty. Te zmienne zostały wprowadzone w MySQL 5.6.5.
- **Connections**
Ta zmienna określa liczbę prób nawiązania połączenia z serwerem (zarówno zakończonych powodzeniem, jak i niepowodzeniem).
- **Created_tmp_disk_tables**
Ta zmienna określa liczbę tabel tymczasowych utworzonych przez serwer na dysku podczas przetwarzania zapytań.
- **Created_tmp_files**
Ta zmienna określa liczbę utworzonych przez serwer plików tymczasowych.
- **Created_tmp_tables**
Ta zmienna określa liczbę wewnętrznych tabel tymczasowych utworzonych przez serwer podczas przetwarzania zapytań.
- **Delayed_errors**
Ta zmienna określa liczbę błędów, jakie wystąpiły podczas przetwarzania zapytań INSERT DELAYED.
- **Delayed_insert_threads**
Ta zmienna określa liczbę wątków obsługujących zapytania INSERT DELAYED.
- **Delayed_writes**
Ta zmienna określa liczbę rekordów wstawionych przez zapytania INSERT DELAYED.
- **Flush_commands**
Ta zmienna określa liczbę wykonanych operacji opróżnienia tabeli.
- **Handler_commit**
Ta zmienna określa liczbę żądań zatwierdzenia transakcji.
- **Handler_delete**
Ta zmienna określa liczbę żądań usunięcia rekordu z tabeli.
- **Handler_external_lock**
Ta zmienna jest powiązana z liczbą operacji nakładania blokad, zarówno na początku, jak i na końcu dostępu do tabeli. Liczbę operacji otrzymasz po podzieleniu tej wartości przez dwa. Ta zmienna została wprowadzona w MySQL 5.6.2.
- **Handler_mrr_init**
Pewne silniki bazy danych dostarczają własną optymalizację strategii Multi-Range Read. Ta zmienna określa, ile razy serwer użył takich implementacji. Omawiana zmienna została wprowadzona w MySQL 5.6.1.
- **Handler_prepare**
Ta zmienna określa liczbę przygotowań do dwufazowej operacji zatwierdzenia.

- **Handler_read_first**
Ta zmienna określa liczbę żądań odczytu pierwszego rekordu indeksu.
- **Handler_read_key**
Ta zmienna określa liczbę żądań odczytu rekordu na podstawie wartości indeksu.
- **Handler_read_last**
Ta zmienna określa liczbę żądań odczytu ostatniego rekordu indeksu.
Omawiana zmienna została wprowadzona w MySQL 5.5.7.
- **Handler_read_next**
Ta zmienna określa liczbę żądań odczytu następnego rekordu w kolejności indeksu.
- **Handler_read_prev**
Ta zmienna określa liczbę żądań odczytu poprzedniego rekordu w malejącej kolejności indeksu.
- **Handler_read_rnd**
Ta zmienna określa liczbę żądań odczytu rekordu na podstawie jego położenia.
- **Handler_read_rnd_next**
Ta zmienna określa liczbę żądań odczytu następnego rekordu. Jeżeli wartość tej zmiennej jest wysoka, prawdopodobnie wykonujesz wiele zapytań wymagających pełnego skanowania tabeli lub nie używasz prawidłowo indeksów.
- **Handler_rollback**
Ta zmienna określa liczbę żądań wycofania transakcji.
- **Handler_savepoint**
Ta zmienna określa liczbę żądań utworzenia punktu kontrolnego transakcji.
- **Handler_savepoint_rollback**
Ta zmienna określa liczbę żądań wycofania do punktu kontrolnego transakcji.
- **Handler_update**
Ta zmienna określa liczbę żądań uaktualnienia rekordu w tabeli.
- **Handler_write**
Ta zmienna określa liczbę żądań wstawienia rekordu do tabeli.
- **Innodb_xxx**
Patrz punkt D.2.1, zatytułowany „Zmienne stanu InnoDB”.
- **Key_blocks_not_flushed**
Ta zmienna określa liczbę bloków w buforze kluczy, które zostały zmodyfikowane, ale jeszcze niezapisane na dysku.
- **Key_blocks_unused**
Ta zmienna określa liczbę nieużywanych bloków w buforze kluczy.
- **Key_blocks_used**
Ta zmienna określa największą liczbę jednocześnie używanych bloków w buforze kluczy.

- **Key_read_requests**
Ta zmienna określa liczbę żądań odczytu bloku z bufora kluczy.
- **Key_reads**
Ta zmienna określa liczbę odczytów z dysku bloków indeksu.
- **Key_write_requests**
Ta zmienna określa liczbę żądań zapisu bloku w buforze kluczy.
- **Key_writes**
Ta zmienna określa liczbę zapisów na dysku bloków indeksu.
- **Last_query_cost**
Ta zmienna przedstawia przeprowadzone przez optymalizator zapytań obliczenie kosztu ostatniego zapytania. Ta wartość jest użyteczna jedynie w przypadku zapytań nieużywających klauzuli UNION lub podzapytań. Wartość 0 oznacza, że koszt nie został jeszcze obliczony. Taka wartość jest przypisywana zapytaniom obsłużonym za pomocą bufora zapytań.
- **Last_query_partial_plans**
W przypadku ostatnio wykonanego zapytania ta zmienna oznacza liczbę iteracji podczas przygotowywania planu wykonania przez optymalizator zapytań. Ta zmienna została wprowadzona w MySQL 5.6.5.
- **Max_used_connections**
Ta zmienna określa największą liczbę jednocześnie otwartych połączeń.
- **Not_flushed_delayed_rows**
Ta zmienna określa liczbę rekordów oczekujących na zapisanie na skutek wykonania zapytań INSERT DELAYED.
- **Open_files**
Ta zmienna określa liczbę otwartych plików.
- **Open_streams**
Ta zmienna określa liczbę otwartych strumieni. Strumień to plik otwarty za pomocą funkcji `fopen()`, to ma zastosowanie jedynie względem plików dzienników zdarzeń.
- **Open_table_definitions**
Ta zmienna określa liczbę buforowanych plików *.frm*.
- **Open_tables**
Ta zmienna określa liczbę otwartych tabel; tabele tymczasowe nie są uwzględniane.
- **Opened_files**
Ta zmienna wskazuje, ile razy serwer otworzył plik za pomocą jego wewnętrznej funkcji `my_open()`.
- **Opened_table_definitions**
Ta zmienna określa, ile razy serwer otworzył plik *.frm*.

- **Opened_tables**
Ta zmienna określa, ile razy serwer otworzył tabelę. Jeżeli wartość tej zmiennej jest wysoka, dobrym rozwiązaniem może być zwiększenie wielkości bufora tabeli.
- **Performance_schema_xxx**
Zmienne stanu rozpoczynające się przedrostkiem `Performance_schema` są używane do zbierania i analizowania danych dotyczących wydajności serwera. Więcej informacji na ten temat znajdziesz w podręczniku użytkownika MySQL.
- **Prepared_stmt_count**
Ta zmienna określa liczbę zapytań preinterpretowanych.
- **Qcache_xxx**
Patrz punkt D.2.2, zatytułowany „Zmienne stanu bufora zapytań”.
- **Questions**
Ta zmienna określa liczbę zapytań otrzymanych przez serwer (zarówno zakończonych powodzeniem, jak i niepowodzeniem). Współczynnik wartości zmiennej `Questions` do wartości zmiennej `Update` wskazuje liczbę zapytań na sekundę.
- **Select_full_join**
Ta zmienna określa liczbę „pełnych” złączeń, to znaczy przeprowadzonych bez użycia indeksów.
- **Select_full_range_join**
Ta zmienna określa liczbę złączeń przeprowadzonych za pomocą wyszukiwania zakresu w tabeli.
- **Select_range**
Ta zmienna określa liczbę złączeń przeprowadzonych za pomocą zakresu w pierwszej tabeli.
- **Select_range_check**
Ta zmienna określa liczbę złączeń, dla których wyszukiwanie zakresu zostało użyte w celu pobrania rekordów w drugiej tabeli.
- **Select_scan**
Ta zmienna określa liczbę złączeń, które spowodowały przeprowadzenie pełnego skanowania pierwszej tabeli.
- **Slave_heartbeat_period**
Ta zmienna wyraża w sekundach częstotliwość pulsu (ang. *heartbeat*) replikacji.
- **Slave_last_heartbeat**
To jest wartość typu `TIMESTAMP`, wskazująca datę i czas ostatniego pulsu przez serwer podległy z serwera głównego. Ta zmienna została wprowadzona w wersji MySQL 5.6.1.
- **Slave_open_temp_tables**
Ta zmienna określa liczbę tabel tymczasowych otwartych przez wątek SQL serwera podległego.

- **Slave_received_heartbeats**
Ta zmienna określa liczbę pulsów otrzymanych przez serwer podległy z serwera głównego od chwili wykonania ostatniego zapytania `CHANGE MASTER` lub ponownego uruchomienia serwera podległego.
- **Slave_retried_transactions**
Ta zmienna wskazuje, ile razy wątek SQL serwera podległego powtarzał transakcje.
- **Slave_running**
Ta zmienna informuje, czy działają wątki SQL i wejścia-wyjścia serwera podległego.
- **Slow_launch_threads**
Ta zmienna określa liczbę wątków, których utworzenie trwa dłużej niż wyrażona w sekundach wartość zmiennej `slow_launch_time`.
- **Slow_queries**
Ta zmienna określa liczbę zapytań, których wykonanie trwa dłużej niż wyrażona w sekundach wartość zmiennej `long_query_time`.
- **Sort_merge_passes**
Ta zmienna określa liczbę połączonych przejść przeprowadzonych przez algorytm sortowania.
- **Sort_range**
Ta zmienna określa liczbę operacji sortowania wykorzystujących zakres.
- **Sort_rows**
Ta zmienna określa liczbę posortowanych rekordów.
- **Sort_scan**
Ta zmienna określa liczbę operacji sortowania używających pełnego skanowania tabeli.
- **Ssl_xxx**
Patrz punkt D.2.3, zatytułowany „Zmienne stanu SSL”.
- **Table_locks_immediate**
Ta zmienna określa liczbę żądań nałożenia blokady tabeli spełnionych natychmiast, bez konieczności oczekiwania.
- **Table_locks_waited**
Ta zmienna określa liczbę żądań nałożenia blokady tabeli spełnionych jedynie po oczekiwaniu. Jeżeli wartość tej zmiennej jest wysoka, oznacza istnienie ogromnej rywalizacji w zakresie nakładania blokad na tabele.
- **Table_open_cache_hits**, **Table_open_cache_misses**,
Table_open_cache_overflows
Te zmienne dostarczają dane statystyczne dotyczące operacji otwierania bufora tabeli. Trafienia i nietrafienia to operacje wyszukiwania, które znalazły tabelę w buforze lub jej nie znalazły. Zmienna `Table_open_cache_overflow` informuje,

ile razy bufor był rozszerzany poza wartość wskazywaną przez zmienną `table_open_cache`. Omawiane zmienne zostały wprowadzone w MySQL 5.6.6.

- `Tc_log_max_pages_used`
Ta zmienna określa maksymalną liczbę stron używanych dla pliku dziennika koordynatora odzyskiwania transakcji.
- `Tc_log_page_size`
Ta zmienna określa wielkość strony pliku dziennika koordynatora odzyskiwania transakcji.
- `Tc_log_page_waits`
Ta zmienna wskazuje, ile razy dwufazowa operacja zatwierdzania musiała czekać na wolną stronę w pliku dziennika koordynatora odzyskiwania transakcji.
- `Threads_cached`
Ta zmienna określa liczbę wątków w buforze wątków.
- `Threads_connected`
Ta zmienna określa liczbę otwartych połączeń.
- `Threads_created`
Ta zmienna wskazuje, ile razy serwer utworzył wątek przeznaczony do obsługi połączeń klientów.
- `Threads_running`
Ta zmienna określa liczbę aktywnych (nieuśpionych) wątków.
- `Uptime`
Ta zmienna wskazuje liczbę sekund, które upłynęły od chwili uruchomienia serwera.
- `Uptime_since_flush_status`
Ta zmienna wskazuje liczbę sekund, które upłynęły od chwili ostatniego wykonania zapytania `FLUSH STATUS`.

D.2.1. Zmienne stanu InnoDB

Wymienione w tym punkcie zmienne wyświetlają informacje dotyczące działania silnika bazy danych InnoDB. Wiele przedstawionych tutaj zmiennych znajduje się w danych wyjściowych zapytania `SHOW ENGINE INNODB STATUS`, ale znacznie łatwiej je analizować w danych wyjściowych zapytania `SHOW STATUS`.

- `Innodb_available_undo_logs`
Ta zmienna określa liczbę dostępnych dzienników cofnięcia InnoDB. (Zmienna systemowa `innodb_undo_logs` wskazuje liczbę aktywnych dzienników tego rodzaju). Ta zmienna została wprowadzona w MySQL 5.6.5.

- **Innodb_buffer_pool_pages_data**
Ta zmienna określa liczbę stron w puli bufora InnoDB, które zawierają dane. Zmienna uwzględnia czyste, niezmodyfikowane strony oraz brudne strony zawierające zmodyfikowane dane.
- **Innodb_buffer_pool_dump_status**
Ta zmienna określa stan operacji zapisu puli bufora. Została wprowadzona w MySQL 5.6.3.
- **Innodb_buffer_pool_load_status**
Ta zmienna określa stan operacji wczytania puli bufora. Została wprowadzona w MySQL 5.6.3.
- **Innodb_buffer_pool_pages_dirty**
Ta zmienna określa liczbę stron w puli bufora InnoDB, które zawierają zmodyfikowane dane.
- **Innodb_buffer_pool_pages_flushed**
Ta zmienna określa liczbę stron puli bufora InnoDB, dla których zostały wydane żądania ich zapisu.
- **Innodb_buffer_pool_pages_free**
Ta zmienna określa liczbę wolnych stron w puli bufora InnoDB.
- **Innodb_buffer_pool_pages_latched**
Ta zmienna określa liczbę stron w puli bufora InnoDB w procesie odczytu lub zapisu bądź też tych, które nie mogą być zapisane i opróżnione w celu ponownego użycia. Wartość tej zmiennej jest wyświetlana tylko wtedy, gdy serwer MySQL został skompilowany wraz ze zdefiniowaną opcją `UNIV_DEBUG`.
- **Innodb_buffer_pool_pages_misc**
Ta zmienna określa liczbę stron zaalokowanych w puli bufora InnoDB dla wewnętrznych operacji.
- **Innodb_buffer_pool_pages_total**
Ta zmienna określa całkowitą liczbę stron w puli bufora InnoDB.
- **Innodb_buffer_pool_read_ahead**
Ta zmienna określa liczbę stron wczytanych do puli bufora InnoDB przez działający w tle wątek odczytu z wyprzedzeniem.
- **Innodb_buffer_pool_read_ahead_evicted**
Ta zmienna określa liczbę stron wczytanych do puli bufora InnoDB przez działający w tle wątek odczytu z wyprzedzeniem i usuniętych z niego bez użycia w zapytaniu.
- **Innodb_buffer_pool_read_requests**
Ta zmienna określa liczbę logicznych żądań odczytu wydanych przez InnoDB.
- **Innodb_buffer_pool_reads**
Ta zmienna określa liczbę operacji odczytu pojedynczej strony wykonanych na skutek braku możliwości przeprowadzenia logicznego odczytu z puli bufora InnoDB.

- `Innodb_buffer_pool_wait_free`
Ta zmienna wskazuje, ile razy silnik InnoDB musiał czekać z zapisem na opróżnienie puli bufora. Operacje zapisu są z reguły przeprowadzane w tle, ale jeśli aktualnie nie ma dostępnych stron, to InnoDB musi czekać na odczyt strony lub utworzyć nową.
- `Innodb_buffer_pool_write_requests`
Ta zmienna określa liczbę operacji zapisu puli bufora InnoDB.
- `Innodb_data_fsyncs`
Ta zmienna określa liczbę przeprowadzonych przez InnoDB operacji synchronizacji z dyskiem.
- `Innodb_data_pending_fsyncs`
Ta zmienna określa liczbę oczekujących na przeprowadzenie przez InnoDB operacji synchronizacji z dyskiem.
- `Innodb_data_pending_reads`
Ta zmienna określa liczbę oczekujących na wykonanie przez InnoDB operacji odczytu danych.
- `Innodb_data_pending_writes`
Ta zmienna określa liczbę oczekujących na wykonanie przez InnoDB operacji zapisu danych.
- `Innodb_data_read`
Ta zmienna określa liczbę bajtów odczytanych przez InnoDB.
- `Innodb_data_reads`
Ta zmienna określa liczbę przeprowadzonych przez InnoDB operacji odczytu danych.
- `Innodb_data_writes`
Ta zmienna określa liczbę przeprowadzonych przez InnoDB operacji zapisu danych.
- `Innodb_data_written`
Ta zmienna określa liczbę bajtów zapisanych przez InnoDB.
- `Innodb_dblwr_pages_written`
Ta zmienna określa liczbę stron zapisanych przez InnoDB w buforze podwójnego zapisu.
- `Innodb_dblwr_writes`
Ta zmienna określa liczbę operacji zapisu w buforze podwójnego zapisu InnoDB.
- `Innodb_have_atomic_builtins`
Ta zmienna wskazuje, czy serwer MySQL został skompilowany wraz z włączoną obsługą operacji niepodzielnych.

- **Innodb_log_waits**
Ta zmienna wskazuje, ile razy silnik InnoDB musiał oczekiwać z operacją zapisu na opróżnienie puli bufora.
- **Innodb_log_write_requests**
Ta zmienna określa liczbę żądań zapisu w pliku dziennika zdarzeń InnoDB.
- **Innodb_log_writes**
Ta zmienna określa liczbę operacji zapisu w pliku dziennika zdarzeń InnoDB.
- **Innodb_num_open_files**
Ta zmienna określa liczbę otwartych plików przez InnoDB. Została wprowadzona w MySQL 5.6.2.
- **Innodb_os_log_fsyncs**
Ta zmienna określa liczbę operacji synchronizacji z dyskiem pliku dziennika zdarzeń InnoDB.
- **Innodb_os_log_pending_fsyncs**
Ta zmienna określa liczbę oczekujących operacji synchronizacji z dyskiem pliku dziennika zdarzeń InnoDB.
- **Innodb_os_log_pending_writes**
Ta zmienna określa liczbę oczekujących operacji zapisu pliku dziennika zdarzeń InnoDB.
- **Innodb_os_log_written**
Ta zmienna określa liczbę bajtów zapisanych w pliku dziennika InnoDB.
- **Innodb_page_size**
Ta zmienna określa używaną przez InnoDB wielkość strony zdefiniowaną w trakcie kompilacji. Tej zmiennej można użyć do konwersji wartości mierzonych w stronach na jednostki wyrażone w bajtach. Wartość domyślna wynosi 16 KB.
- **Innodb_pages_created**
Ta zmienna określa liczbę stron utworzonych przez InnoDB.
- **Innodb_pages_read**
Ta zmienna określa liczbę stron odczytanych przez InnoDB.
- **Innodb_pages_written**
Ta zmienna określa liczbę stron zapisanych przez InnoDB.
- **Innodb_row_lock_current_waits**
Ta zmienna określa liczbę oczekujących na nałożenie blokad rekordów w InnoDB.
- **Innodb_row_lock_time**
Ta zmienna określa wyrażony w milisekundach całkowity czas poświęcony na nakładanie blokad rekordów w InnoDB.

- `Innodb_row_lock_time_avg`
Ta zmienna określa wyrażony w milisekundach średni czas potrzebny na nałożenie blokady rekordu w InnoDB.
- `Innodb_row_lock_time_max`
Ta zmienna określa wyrażony w milisekundach maksymalny czas potrzebny na nałożenie blokady rekordu w InnoDB.
- `Innodb_row_lock_waits`
Ta zmienna wskazuje, ile razy silnik InnoDB oczekiwał na nałożenie blokady na rekord.
- `Innodb_rows_deleted`
Ta zmienna określa liczbę rekordów usuniętych z tabel InnoDB.
- `Innodb_rows_inserted`
Ta zmienna określa liczbę rekordów wstawionych w tabelach InnoDB.
- `Innodb_rows_read`
Ta zmienna określa liczbę rekordów odczytanych z tabel InnoDB.
- `Innodb_rows_updated`
Ta zmienna określa liczbę rekordów uaktualnionych w tabelach InnoDB.
- `Innodb_truncated_status_writes`
Ta zmienna wskazuje, ile razy dane wyjściowe zapytania `SHOW ENGINE INNODB STATUS` były skracane. Omawiana zmienna została wprowadzona w MySQL 5.5.7.

D.2.2. Zmienne stanu bufora zapytań

Wymienione w tym punkcie zmienne wyświetlają informacje o działaniu bufora zapytań.

- `Qcache_free_blocks`
Ta zmienna określa liczbę wolnych bloków pamięci w buforze zapytań.
- `Qcache_free_memory`
Ta zmienna określa ilość wolnej pamięci w buforze zapytań.
- `Qcache_hits`
Ta zmienna określa liczbę zapytań, które zostały obsłużone przez zapytania przechowywane w buforze.
- `Qcache_inserts`
Ta zmienna określa liczbę zapytań, które kiedykolwiek zostały zarejestrowane w buforze zapytań.
- `Qcache_lowmem_prunes`
Ta zmienna określa liczbę buforowanych wyników zapytań, które zostały usunięte z bufora w celu zrobienia miejsca dla nowszych wyników.
- `Qcache_not_cached`
Ta zmienna określa liczbę zapytań, które były niemożliwe do buforowania lub ich buforowanie uniemożliwiało użycie opcji `SQL_NO_CACHE`.

- `Qcache_queries_in_cache`
Ta zmienna określa liczbę zapytań zarejestrowanych w buforze.
- `Qcache_total_blocks`
Ta zmienna wskazuje całkowitą liczbę bloków pamięci w buforze zapytań.

D.2.3. Zmienne stanu SSL

Wymienione w tym punkcie zmienne dostarczają informacje o kodzie zarządzającym SSL. Wiele z nich odzwierciedla stan bieżącej sesji i nie ma przypisanej wartości, dopóki sesja faktycznie nie korzysta z protokołu SSL. Wymienione tutaj zmienne są niedostępne, o ile serwer nie został skompilowany wraz z obsługą SSL.

- `Ssl_accept_renegotiates`
Ta zmienna określa liczbę rozpoczętych renegotjacji w trybie serwera.
- `Ssl_accepts`
Ta zmienna określa liczbę rozpoczętych połączeń SSL/TLS w trybie serwera.
- `Ssl_callback_cache_hits`
Ta zmienna określa liczbę sesji, które z powodzeniem zostały pobrane z zewnętrznego bufora sesji w trybie serwera.
- `Ssl_cipher`
Ta zmienna określa protokół szyfrowania SSL dla sesji (brak wartości, jeśli nie jest stosowane szyfrowanie). Wartość tej zmiennej można wykorzystać do sprawdzenia, czy sesja jest szyfrowana.
- `Ssl_cipher_list`
Ta zmienna wyświetla listę dostępnych algorytmów szyfrowania SSL.
- `Ssl_client_connects`
Ta zmienna określa liczbę rozpoczętych połączeń SSL/TLS w trybie klienta.
- `Ssl_connect_renegotiates`
Ta zmienna określa liczbę rozpoczętych renegotjacji w trybie klienta.
- `Ssl_ctx_verify_depth`
Ta zmienna określa głębokość weryfikacji kontekstu SSL.
- `Ssl_ctx_verify_mode`
Ta zmienna określa tryb weryfikacji kontekstu SSL.
- `Ssl_default_timeout`
Ta zmienna określa domyślny czas utraty ważności sesji SSL.
- `Ssl_finished_accepts`
Ta zmienna określa liczbę nawiązanych z powodzeniem sesji SSL/TLS w trybie serwera.
- `Ssl_finished_connects`
Ta zmienna określa liczbę nawiązanych z powodzeniem sesji SSL/TLS w trybie klienta.

- `Ssl_server_not_after`
Ta zmienna określa końcową datę ważności certyfikatu SSL.
Została wprowadzona w MySQL 5.6.3.
- `Ssl_server_not_before`
Ta zmienna określa początkową datę ważności certyfikatu SSL.
Została wprowadzona w MySQL 5.6.3.
- `Ssl_session_cache_hits`
Ta zmienna określa liczbę sesji SSL znalezionych w buforze sesji.
- `Ssl_session_cache_misses`
Ta zmienna określa liczbę sesji SSL niezalezionych w buforze sesji.
- `Ssl_session_cache_mode`
Ta zmienna określa typ buforowania SSL używanego przez serwer.
- `Ssl_session_cache_overflows`
Ta zmienna określa liczbę sesji usuniętych z bufora na skutek jego zapelnienia.
- `Ssl_session_cache_size`
Ta zmienna określa liczbę sesji, które mogą być przechowywane w buforze sesji SSL.
- `Ssl_session_cache_timeouts`
Ta zmienna wskazuje liczbę sesji, które wygasły.
- `Ssl_sessions_reused`
Ta zmienna określa, czy sesja została ponownie użyta z poprzedniej sesji.
- `Ssl_used_session_cache_entries`
Ta zmienna wskazuje liczbę sesji w buforze sesji.
- `Ssl_verify_depth`
Ta zmienna określa głębokość weryfikacji SSL.
- `Ssl_verify_mode`
Ta zmienna określa tryb weryfikacji SSL.
- `Ssl_version`
Ta zmienna określa wersję protokołu SSL dla danej sesji.

D.3. Zmienne zdefiniowane przez użytkownika

Zmienne zdefiniowane przez użytkownika (lub prościej „zmienne użytkownika”) mogą mieć przypisane wartości, do których później możesz się odwoływać w innych zapytaniach.

Nazwa zmiennej zdefiniowanej przez użytkownika składa się ze znaku @, po którym znajduje się identyfikator. Stosowane są te same reguły dotyczące prawidłowych nazw identyfikatorów (patrz podrozdział 2.2, zatytułowany „Identyfikatory składni MySQL

i reguły nadawania nazw”). Jednak, w przeciwieństwie do identyfikatorów, nazwa zmiennej użytkownika może zawierać kropkę bez konieczności cytowania. Wielkość liter nie ma znaczenia w nazwach zmiennych użytkownika.

Wartości zmiennym użytkownika mogą być przypisywane za pomocą operatorów = lub := w zapytaniach SET lub operatora := w innych zapytaniach, takich jak SELECT. W pojedynczym zapytaniu można przeprowadzić wiele operacji przypisania wartości.

```
mysql> SET @x = 0, @y = 2;
mysql> SET @color := 'red', @size := 'large';
mysql> SELECT @x, @y, @color, @size;
+-----+-----+-----+-----+
| @x   | @y   | @color | @size |
+-----+-----+-----+-----+
| 0    | 2    | red    | large |
+-----+-----+-----+-----+
mysql> SELECT @count := COUNT(*) FROM member;
+-----+
| @count := COUNT(*) |
+-----+
| 102                 |
+-----+
```

Zmiennym użytkownika mogą być przypisywane wartości w postaci liczb całkowitych, o stałej ilości cyfr, zmiennoprzecinkowych, ciągi tekstowe i wartości NULL. Ponadto, można przypisać dowolne wyrażenie, łącznie z odwołującymi się do innych zmiennych. Jeżeli uzyskujesz dostęp do zmiennej użytkownika, której nie została przypisana wartość, wtedy jej wartością jest NULL.

Wartość zmiennej użytkownika nie jest zachowywana między różnymi sesjami w serwerze. Oznacza to utratę wspomnianej wartości po zakończeniu sesji.

W zapytaniach SELECT zwracających wiele rekordów przypisanie zmiennej następuje dla każdego rekordu. Ostateczną wartością będzie ta, która została przypisana ostatniemu rekordowi.

Zmienne użytkownika o wartości w postaci ciągu tekstowego mają to samo kodowanie znaków i kolejność sortowania jak w przypisanej im wartości:

```
mysql> SET @s = CONVERT('abc' USING latin2) COLLATE latin2_czech_cs;
mysql> SELECT CHARSET(@s), COLLATION(@s);
+-----+-----+
| CHARSET(@s) | COLLATION(@s) |
+-----+-----+
| latin2     | latin2_czech_cs |
+-----+-----+
```


Przewodnik po składni SQL

W tym dodatku zostanie przedstawiona stosowana przez MySQL składnia zapytań SQL. O ile nie zaznaczono inaczej, zaprezentowane tutaj zapytania są dostępne w wydaniu MySQL 5.5.0 lub nowszym. Wyraźnie zaznaczono zmiany wprowadzone w późniejszych wydaniach.

Dodatek składa się z trzech części:

- Zapytania SQL inne niż zapytania złożone.
- Zapytania SQL przeznaczone do tworzenia zapytań złożonych, które są przygotowywane za pomocą słów kluczowych BEGIN i END oraz mogą być stosowane do tworzenia przechowywanych po stronie serwera programów składowanych: funkcji, procedur, wyzwalaczy i zdarzeń.
- Składnia umieszczania komentarzy w kodzie SQL. Komentarze są używane w charakterze opisowego tekstu ignorowanego przez serwer oraz do ukrywania charakterystycznych dla MySQL słów kluczowych, które będą wykonane przez MySQL, ale zignorowane przez inne serwery baz danych.

W opisie składni zapytań zastosowano następujące konwencje:

- Nawiasy kwadratowe ([]) oznaczają informacje opcjonalne.
- Pionowe kreski (|) rozdzielają alternatywne elementy listy. Jeżeli lista będzie znajdowała się w nawiasie kwadratowym, można z niej wybrać jeden element. Umieszczenie listy w nawiasie klamrowym ({}) oznacza konieczność wyboru jednego elementu.
- Wielokropek oznacza, że poprzedni element można powtórzyć.
- *n* oznacza liczbę całkowitą.
- 'str' oznacza wartość w postaci ciągu tekstowego. Wartość cytowana taka jak 'file_name' lub 'pattern' wskazuje znacznie dokładniejszy rodzaj wartości, na przykład nazwę pliku lub wzorzec.

Ten dodatek nie przedstawia zapytań i klauzul powiązanych z funkcjami zdefiniowanymi przez użytkownika (ang. *User-Defined Function* — UDF), transakcjami XA bądź też charakterystycznymi dla klastra MySQL lub rzadziej używanych silników bazy danych, takich jak FEDERATED.

Pewne ogólne synonimy zawsze są aktualne, dlatego zostały wymienione poniżej zamiast w każdym miejscu, w którym można ich użyć:

- Dowolny z poniższych formatów można zastosować w celu wskazania kodowania znaków:

```
CHARACTER SET charset
CHARSET [=] charset
```

Powyższe formy synonimów mogą być używane w definicjach tabel i kolumn oraz w zapytaniach CREATE DATABASE lub ALTER DATABASE.

- SCHEMA i SCHEMAS są synonimami dla odpowiednio DATABASE i DATABASES, mogą być dowolnie stosowane w zapytaniach, w których można stosować słowa kluczowe DATABASE i DATABASES. Na przykład, bazę danych można utworzyć za pomocą zapytania CREATE DATABASE lub CREATE SCHEMA.
- COLUMNS i FIELDS są synonimami.

E.1. Składnia zapytań SQL (zapytania niezłożone)

W tym podrozdziale zostanie przedstawiona składnia i znaczenie poszczególnych zapytań SQL innych niż przeznaczone do tworzenia zapytań złożonych, które z kolei będą omówione w podrozdziale E.2, zatytułowanym „Składnia zapytań SQL (zapytania złożone)”. Wykonanie zapytania zakończy się niepowodzeniem, jeśli nie masz odpowiednich uprawnień do jego wykonania. Na przykład, wykonanie zapytania USE *nazwa_bazy_danych* zakończy się niepowodzeniem, jeśli nie masz uprawnień do wskazanej bazy danych.

ALTER DATABASE

```
ALTER DATABASE [db_name] db_attr ...
ALTER DATABASE db_name UPGRADE DATA DIRECTORY NAME
```

To zapytanie powoduje zmianę atrybutów bazy danych lub uaktualnia kodowanie nazwy katalogu bazy danych. Wymagane jest posiadanie uprawnień ALTER do wskazanej bazy danych.

W przypadku pierwszej składni dozwolone wartości atrybutów bazy danych (*db_attr*) są takie same jak wymienione w zapytaniu CREATE DATABASE. Jeżeli pominięta zostanie nazwa bazy danych, zapytanie będzie dotyczyło domyślnej bazy danych. W przypadku braku domyślnej bazy danych nastąpi wygenerowanie błędu.

Składnia `UPGRADE DATA DIRECTORY NAME` jest używana wtedy, gdy chcesz przeprowadzić uaktualnienie z wersji starszej niż 5.1. Jeżeli to konieczne, powoduje ona ponowne zakodowanie nazwy katalogu bazy danych i zastosowanie kodowania systemu plików aktualnie używanego przez MySQL, jeśli nazwa zawiera znaki specjalne.

ALTER EVENT

```
ALTER
  [DEFINER = definer_name]
  EVENT event_name
  [ON SCHEDULE schedule]
  [ON COMPLETION [NOT] PRESERVE]
  [RENAME TO new_event_name]
  [ENABLE | DISABLE [ON SLAVE]]
  [COMMENT 'str']
  [DO event_stmt]
```

Zapytanie zmienia istniejące zdarzenie na definicję wskazaną w zapytaniu. Klauzula `RENAME TO` powoduje zmianę nazwy zdarzenia. Pozostałe klauzule zostały opisane w podpunkcie dotyczącym zapytania `CREATE EVENT`. Musisz mieć uprawnienie `EVENT` do bazy danych, do której należy zdarzenie.

ALTER FUNCTION, ALTER PROCEDURE

```
ALTER {FUNCTION | PROCEDURE} routine_name [characteristic] ...
characteristic:
  COMMENT 'str'
  | {CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA}
  | LANGUAGE SQL
  | SQL SECURITY {DEFINER | INVOKER}
```

Wymienione zapytania zmieniają cechy charakterystyczne procedur składowanych. Wymagają uprawnienia `ALTER ROUTINE` do danej procedury. Wspomniane cechy charakterystyczne zostały opisane w podpunktach dotyczących zapytań `CREATE FUNCTION` i `CREATE PROCEDURE`.

ALTER TABLE

```
ALTER [IGNORE] TABLE tbl_name [action [, action] ...]
```

Zapytanie `ALTER TABLE` pozwala na zmianę nazwy tabeli lub modyfikację jej struktury. Aby użyć wymienionego zapytania, należy podać nazwę tabeli wraz z operacjami, które mają być przeprowadzone względem niej. Słowo kluczowe `IGNORE` jest stosowane, jeśli operacja może spowodować powstanie powielonych wartości kluczy w unikalnym indeksie po modyfikacji tabeli. W takim przypadku, jeśli zabraknie słowa kluczowego `IGNORE`, wykonywanie zapytania `ALTER TABLE` zostanie przerwane. Gdy użyjesz `IGNORE`, rekordy powielające wartości unikalnych kluczy zostaną usunięte.

W trakcie wykonywania zapytania `ALTER TABLE` inne klienty mogą odczytywać dane z tabeli poddawanej modyfikacji. Natomiast operacje, które próbują uaktualnić tabelę,

będą wstrzymane aż do chwili zakończenia wykonywania zapytania `ALTER TABLE` i dopiero wtedy uaktualnienia zostaną wprowadzone już w nowej tabeli.

Wartości *action* wskazują operacje przeprowadzane po kolei. Pewne operacje nie mogą być łączone z innymi, co zostało wymienione w opisie danej operacji. Jeżeli nie zostanie podana żadna operacja, wykonanie zapytania `ALTER TABLE` nie będzie miało żadnego skutku.

W przypadku operacji dotyczących indeksu i zawierających klauzule *index_option* pewne silniki baz danych pozwalają na wskazanie algorytmu indeksowania lub innego modyfikatora definicji indeksu. Informacje szczegółowe dotyczące wartości indeksu obsługiwanych w różnych wersjach MySQL znajdziesz w podpunkcie opisującym zapytanie `CREATE INDEX`. Więcej informacji na temat tworzenia indeksu przedstawiono w punkcie 2.6.4, zatytułowanym „Indeksowanie tabel”.

Wartość *action* może być dowolną z poniższych:

■ *table_option*

Wskazuje opcje tabeli rodzaju, których można użyć w części *table_option* zapytania `CREATE TABLE`. Można pominąć przecinki rozdzielające kolejne opcje tabeli.

```
ALTER TABLE score ENGINE=MyISAM CHECKSUM=1;
ALTER TABLE sayings CHARACTER SET utf8;
```

Wszelkie ograniczenia danej opcji tabeli charakterystyczne dla wersji lub silnika bazy danych zostały omówione w podpunkcie dotyczącym zapytania `CREATE TABLE`. Jeżeli próbujesz zmienić tabelę w celu użycia niedostępnego silnika bazy danych, efekt wykonania zapytania `ALTER TABLE` będzie zależał od ustawienia trybu `SQL NO_ENGINE_SUBSTITUTION`.

Opcja `[DEFAULT] CHARACTER SET` tabeli powoduje zmianę domyślnego kodowania znaków tabeli, ale nie konwertuje istniejących kolumn na wskazane kodowanie znaków. Aby przeprowadzić tego rodzaju operację, należy użyć akcji `CONVERT TO CHARACTER SET`.

Zapytanie `ALTER TABLE` ignoruje opcje `DATA DIRECTORY` i `INDEX DIRECTORY`.

■ `ADD [COLUMN] col_name col_definition [FIRST | AFTER col_name]`

Powoduje dodanie kolumny do tabeli. Nazwę dodawanej kolumny wskazuje *col_name*, definicję określa *col_definition*; ma taki sam format jak w zapytaniu `CREATE TABLE`. Podanie słowa kluczowego `FIRST` oznacza, że kolumna zostanie wstawiona jako pierwsza kolumna w tabeli. Z kolei użycie `AFTER col_name` powoduje wstawienie kolumny po wskazanej. Jeżeli miejsce umieszczenia kolumny nie zostanie wyraźnie podane, będzie ona wstawiona na końcu tabeli.

```
ALTER TABLE t ADD name CHAR(20);
ALTER TABLE t ADD id INT UNSIGNED NOT NULL PRIMARY KEY FIRST;
ALTER TABLE t ADD birth DATE AFTER name;
```

■ **ADD [COLUMN] (*create_definition*,...)**

Powoduje dodanie kolumn lub indeksów do tabeli. Każda *create_definition* jest definicją kolumny lub indeksu; ma taki sam format jak w zapytaniu CREATE TABLE.

■ **ADD [CONSTRAINT [*name*]] FOREIGN KEY [*fk_name*]
(*index_columns*) *reference_definition***

Powoduje dodanie do tabeli definicji klucza zewnętrznego. Ta akcja jest obsługiwana jedynie w przypadku tabel InnoDB. Klucz zewnętrzny jest oparty na kolumnach wymienionych w parametrze *index_columns*, który powinien zawierać jedną lub więcej nazw kolumn tabeli rozdzielonych przecinkami. Jeśli zostanie podana nazwa CONSTRAINT, wtedy będzie zignorowana. *fk_name* jest identyfikatorem klucza zewnętrznego. Jeżeli zostanie podany, to będzie zignorowany, o ile InnoDB automatycznie nie utworzy indeksu dla klucza zewnętrznego, w takim przypadku *fk_name* stanie się nazwą indeksu. Z kolei parametr *reference_definition* zawiera definicję wskazującą, jak klucz zewnętrzny jest związany z tabelą nadrzędną. Składnia została przedstawiona w podpunkcie dotyczącym zapytania CREATE TABLE.

■ **ALTER TABLE *child***

ADD FOREIGN KEY (*par_id*) REFERENCES *parent* (*par_id*) ON DELETE CASCADE;
Akcje ADD FOREIGN KEY i DROP FOREIGN KEY nie mogą znajdować się w tym samym zapytaniu ALTER TABLE.

■ **ADD FULLTEXT [INDEX | KEY] [*index_name*]
(*index_columns*) [*index_option*] ...**

Dodanie indeksu typu FULLTEXT do tabeli MyISAM lub (począwszy od MySQL 5.6.4) do tabeli InnoDB. Indeks opiera się na kolumnach wymienionych w parametrze *index_columns*, który powinien zawierać jedną lub więcej rozdzielonych przecinkami niebinarnych kolumn typu ciągu tekstowego. Parametr *index_name* jest podawany podobnie jak w przypadku akcji ADD INDEX.

ALTER TABLE *poetry* ADD FULLTEXT (*author,title,stanza*);

■ **ADD {INDEX | KEY} [*index_name*]
(*index_columns*) [*index_option*] ...**

Powoduje dodanie indeksu do tabeli. Indeks opiera się na kolumnach wymienionych w parametrze *index_columns*, który powinien zawierać jedną lub więcej rozdzielonych przecinkami kolumn tabeli. Jeżeli pominięta zostanie *index_name*, wtedy MySQL wybierze ją automatycznie na podstawie nazwy pierwszej kolumny indeksu.

■ **ADD [CONSTRAINT [*name*]] PRIMARY KEY
(*index_columns*) [*index_option*] ...**

Dodaje klucz podstawowy na wskazanych kolumnach. Kluczowi zostaje nadana nazwa PRIMARY. Parametr *index_columns* jest wskazany jak w akcji ADD INDEX.

Każda kolumna musi być zdefiniowana jako typu NOT NULL. Jeżeli klucz podstawowy już istnieje, wtedy wystąpi błąd.

```
ALTER TABLE president ADD PRIMARY KEY (last_name, first_name);
```

- **ADD SPATIAL** [**INDEX** | **KEY**] [*index_name*]
(*index_columns*) [*index_option*]...

Dodaje indeks typu SPATIAL do tabeli MyISAM. Dodawany indeks opiera się na kolumnach wymienionych w parametrze *index_columns*, który powinien zawierać jedną lub więcej rozdzielonych przecinkami kolumn tabeli. Każda kolumna musi być zdefiniowana jako typu NOT NULL. Parametr *index_columns* jest wskazany jak w akcji ADD_INDEX.

```
ALTER TABLE coordinates ADD SPATIAL (x,y);
```

- **ADD** [**CONSTRAINT** [*name*]] **UNIQUE** [**INDEX** | **KEY**]
[*index_name*]
(*index_columns*) [*index_option*] ...

Dodaje indeks unikalnych wartości do tabeli. Parametry *index_name* i *index_columns* są podane podobnie jak w akcji ADD_INDEX.

```
ALTER TABLE absence ADD UNIQUE id_date (student_id, date);
```

- **ALTER** [**COLUMN**] *col_name* {**SET** **DEFAULT** *value* | **DROP** **DEFAULT**}
- Modyfikuje wartość domyślną kolumny na wskazaną wartość lub usuwa bieżącą wartość domyślną. W tym drugim przypadku automatycznie może być przypisana nowa wartość domyślna, jak to zostało przedstawione w punkcie 3.2.3, zatytułowanym „Definiowanie wartości domyślnych kolumn”.

```
ALTER TABLE grade_event ALTER category SET DEFAULT 'Q';  
ALTER TABLE grade_event ALTER category DROP DEFAULT;
```

- **CHANGE** [**COLUMN**] *old_col_name* *new_col_name* *col_definition*
[**FIRST** | **AFTER** *col_name*]

Zmienia nazwę i definicję kolumny. Parametry *old_col_name* i *new_col_name* określają starą i nową nazwę kolumny, natomiast parametr *col_definition* wskazuje docelową definicję kolumny. Parametr *col_definition* ma taki sam format jak w przypadku zapytania CREATE_TABLE, łącznie z wszelkimi atrybutami kolumny, takimi jak NULL, NOT NULL i DEFAULT. Zwróć uwagę, że w celu zmiany definicji, ale nie nazwy konieczne jest dwukrotne podanie tej samej nazwy. Parametry FIRST i AFTER mają taki sam efekt jak w przypadku akcji ADD_COLUMN.

```
ALTER TABLE student CHANGE name name VARCHAR(40);  
ALTER TABLE student CHANGE name student_name CHAR(30) NOT NULL;
```

- **CONVERT TO CHARACTER SET** *charset* [**COLLATE** *collation*]
- Przeprowadza konwersję domyślnego kodowania znaków kolumny i zmienia na podane kodowanie znaków wszystkie znaki niebinarnych kolumn tabeli. Wartość binary dla *charset* powoduje konwersję kolumn na odpowiadające im binarne typy ciągów tekstowych, z kolei DEFAULT konwertuje tabelę do użycia kodowania znaków bazy danych. Klauzula COLLATE może zostać użyta w celu

wskazania kolejności sortowania. Pominięcie wymienionej klauzuli powoduje użycie domyślnej kolejności sortowania we wskazanym kodowaniu znaków.

■ **DISABLE KEYS**

W przypadku tabel MyISAM ta akcja powoduje wyłączenie uaktualniania nieunikalnych indeksów, co normalnie następuje w trakcie zmiany tabeli. Aby ponownie włączyć uaktualnianie indeksu, należy użyć akcji **ENABLE KEYS**.

```
ALTER TABLE score DISABLE KEYS;
```

■ **DISCARD TABLESPACE**

Ta akcja ma zastosowanie względem tabel InnoDB używających oddzielnych przestrzeni tabel. Powoduje usunięcie pliku *tbl_name.ibd*, przechowującego zawartość tabeli. Ta akcja nie może być używana w połączeniu z innymi akcjami.

■ **DROP [COLUMN] *col_name* [RESTRICT | CASCADE]**

Powoduje usunięcie kolumn z tabeli oraz z wszelkich indeksów będących jej częścią. Jeżeli usunięte zostaną wszystkie kolumny z indeksu, wtedy sam indeks również będzie usunięty.

```
ALTER TABLE president DROP suffix;
```

Słowa kluczowe **RESTRICT** i **CASCADE** są przetwarzane, ale pozostają ignorowane i nie mają efektu.

■ **DROP FOREIGN KEY *fk_name***

Usunięcie wskazanej definicji klucza zewnętrznego. Akcje **ADD FOREIGN KEY** i **DROP FOREIGN KEY** nie mogą znajdować się w tym samym zapytaniu **ALTER TABLE**.

■ **DROP {INDEX | KEY} *index_name***

Usunięcie indeksu z tabeli.

```
ALTER TABLE member DROP INDEX name;
```

■ **DROP PRIMARY KEY**

Usunięcie klucza podstawowego z tabel. Jeżeli tabela nie zawiera klucza podstawowego, wtedy wystąpi błąd.

```
ALTER TABLE president DROP PRIMARY KEY;
```

■ **ENABLE KEYS**

W przypadku tabeli MyISAM włącza ponowne uaktualnianie indeksów nieunikalnych, co zostało wcześniej wyłączone za pomocą akcji **DISABLE KEYS**.

```
ALTER TABLE score ENABLE KEYS;
```

■ **FORCE**

Przeprowadza operację „null”, która ponownie tworzy tabelę bez zmiany jej struktury. Przed wydaniem MySQL 5.5.11 ta akcja nie ma żadnego efektu.

■ **IMPORT TABLESPACE**

Ta akcja ma zastosowanie w przypadku tabel InnoDB, które używają oddzielnych przestrzeni tabel. Powoduje powiązanie pliku *tbl_name.ibd* w katalogu bazy

danych tabeli z tabelą. Plik *.ibd* musi być utworzony w tym samym serwerze, do którego jest importowany. (Przypuszczalnie, poprzedni plik *.ibd* tabeli został usunięty za pomocą `DISCARD TABLESPACE`). Ta akcja nie może być używana w połączeniu z innymi akcjami.

- **MODIFY** [**COLUMN**] *col_name col_definition* [**FIRST** | **AFTER** *col_name*]

Zmienia definicję kolumny. Nazwa kolumny do zmodyfikowania jest podawana w parametrze *col_name*. Z kolei parametr *col_definition* ma taki sam format jak w przypadku zapytania `CREATE TABLE`, łącznie z wszelkimi atrybutami kolumny, takimi jak `NULL`, `NOT NULL` i `DEFAULT`. Parametry `FIRST` i `AFTER` mają taki sam efekt jak w przypadku akcji `ADD COLUMN`.

```
ALTER TABLE student MODIFY name VARCHAR(40) DEFAULT '' NOT NULL;
```

- **ORDER BY** *col_list*

Sortuje rekordy w tabeli według kolumn wymienionych w parametrze *col_list*, który powinien zawierać nazwę jednej lub więcej rozdzielonych przecinkami nazw kolumn tabeli. W przypadku podania wielu nazw kolumn to powinna być ostatnia klauzula. Domyślnie stosowana jest rosnąca kolejność sortowania. Po nazwie kolumny można podać słowo kluczowe `ASC` lub `DESC`, wyraźnie wskazujące kolejność sortowania, odpowiednio rosnącą lub malejącą. Sortowanie tabeli w taki sposób może poprawić wydajność kolejnych zapytań pobierających rekordy w tej samej kolejności. To jest najbardziej użyteczne dla tabeli, która nie będzie później modyfikowana, ponieważ rekordy nie pozostają w kolejności, jeśli tabela zostanie zmodyfikowana po przeprowadzeniu akcji `ORDER BY`.

```
ALTER TABLE score ORDER BY event_id, student_id;
```

- **RENAME** [**TO** | **AS**] *new_tbl_name*

Powoduje zmianę nazwy tabeli na wskazaną. Jeżeli zmieniasz nazwę tabeli InnoDB, od której zależą inne tabele (relacja klucza zewnętrznego), InnoDB odpowiednio dopasowuje zależności, aby prowadziły do tabeli o nowej nazwie.

```
ALTER TABLE president RENAME TO prez;
```

Zapytanie `ALTER TABLE` obsługuje modyfikacje partycjonowania. W podpunkcie dotyczącym zapytania `CREATE TABLE` zdefiniowano znaczenie pojęć *partition_scheme* i *partition_definition*, które są stosowane w poniższych opisach akcji. Jeżeli którakolwiek z opcji partycjonowania pojawia się w zapytaniu `ALTER TABLE`, wtedy nie można użyć żadnej z pozostałych.

- *partition_scheme*

Partycjonowanie tabeli zgodnie ze wskazanym opisem partycjonowania. Jeżeli tabela nie jest partycjonowana, wtedy stanie się partycjonowana. W przeciwnym razie stary schemat partycjonowania zostanie zastąpiony nowym.

- **ADD PARTITION** (*partition_definition*)

Dodanie nowej tabeli do partycjonowanej.

- `{ANALYZE | CHECK | OPTIMIZE | REBUILD | REPAIR | TRUNCATE}`
`PARTITION { partition_name [, partition_name] ... | ALL }`
 Przeprowadza wskazaną akcję na wymienionych partycjach. Każda akcja pozwala na użycie słowa kluczowego ALL zamiast listy partycji i wtedy wpływa na wszystkie partycje. TRUNCATE nie działa z podpartycjami.
- `COALESCE PARTITION n`
 Powoduje, że partycjonowana tabela będzie miała *n* partycji mniej na skutek połączenia danych w usuwanych partycjach z istniejącymi. Takie rozwiązanie działa jedynie z partycjami HASH lub KEY. W celu usunięcia partycji LIST lub RANGE należy użyć DROP PARTITION.
- `DROP PARTITION partition_name [, partition_name] ...`
 Usunięcie wskazanej partycji. Ta akcja działa jedynie z partycjami LIST lub RANGE, dane w usuwanej partycji są tracone. Aby zmniejszyć liczbę partycji typu HASH lub KEY, należy użyć COALESCE PARTITION.
- `EXCHANGE PARTITION partition_name WITH TABLE tbl_name2`
 Zamienia wymienioną partycję z tabeli partycjonowanej o wskazanej nazwie na niepartycjonowaną tabelę *tbl_name2*. Pomijając partycjonowanie, obie tabele muszą mieć identyczną strukturę. W tabeli *tbl_name2* nie mogą znajdować się rekordy wykraczające poza definicję partycji i nie mogą znajdować się w żadnych odniesieniach klucza zewnętrznego.
- `REMOVE PARTITIONING`
 Usunięcie całego partycjonowania; skutkiem jest powstanie tabeli niepartycjonowanej.
- `REORGANIZE PARTITION partition_name [, partition_name] ...`
`INTO (partition_definition [, partition_definition] ...)`
 Ponowne partycjonowanie wskazanych partycji i użycie nowych definicji partycjonowania.

ALTER VIEW

ALTER

```
[ALGORITHM = {MERGE | TEMPTABLE | UNDEFINED}]
[DEFINER = definer_name]
[SQL SECURITY = {DEFINER | INVOKER}]
VIEW view_name [(col_list)] AS select_stmt
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

Zapytanie powoduje zmianę istniejącego widoku i nadanie mu wskazanej definicji. Zastosowane klauzule mają takie samo znaczenie jak opisane w zapytaniu CREATE VIEW.

Zapytanie ALTER VIEW wymaga uprawnień CREATE VIEW i DROP do modyfikowanego widoku oraz pewnych uprawnień do wszystkich kolumn użytych w zapytaniu SELECT definiującym widok. Zapytanie ALTER VIEW może być wykonane jedynie przez użytkownika, który zdefiniował dany widok lub posiada uprawnienie SUPER.

ANALYZE TABLE

```
ANALYZE
  [NO_WRITE_TO_BINLOG | LOCAL]
  {TABLE | TABLES} tbl_name [, tbl_name] ...
```

To zapytanie powoduje, że MySQL analizuje wszystkie wymienione tabele i rozproszenie wartości kluczy w każdym indeksie tabeli. Działa jedynie w przypadku tabel MyISAM i InnoDB. Zapytanie ANALYZE TABLE wymaga uprawnień SELECT i INSERT dla każdej tabeli.

Jeżeli włączony jest binarny dziennik zdarzeń, wykonanie zapytania ANALYZE TABLE zostanie w nim zapisane, o ile nie zostanie użyta opcja NO_WRITE_TO_BINLOG lub LOCAL.

Po przeprowadzeniu analizy kolumna Cardinality danych wyjściowych zapytania SHOW INDEX będzie zawierała przybliżoną liczbę unikalnych wartości w indeksach. Informacje pochodzące z analizy mogą być używane przez optymalizator w kolejnych zapytaniach w celu znacznie szybszego przeprowadzenia pewnego typu złączeń.

Analiza tabeli wymaga nałożenia blokady odczytu, która uniemożliwia modyfikację tabeli w trakcie operacji. Zapytanie ANALYZE TABLE nie powoduje żadnego efektu, jeśli tabela była wcześniej już analizowana i nie uległa zmianie od tamtej chwili.

Omawiane zapytanie ANALYZE TABLE generuje dane wyjściowe w formacie opisanym w podpunkcie dotyczącym zapytania CHECK TABLE.

BEGIN

```
BEGIN [WORK]
```

To zapytanie jest synonimem dla START TRANSACTION; zapoznaj się z podpunktem dotyczącym wymienionego zapytania.

BEGIN można używać także w połączeniu z END w programach składowanych w celu utworzenia zapytania złożonego. Zapoznaj się z podrozdziałem E.2, zatytułowanym „Składnia zapytań SQL (zapytania złożone)”.

BINLOG

```
BINLOG 'str'
```

Zapytania BINLOG są generowane przez narzędzie mysqlbinlog, na przykład '*str*' to zdarzenie binarnego dziennika zakodowane jako base64 i przedstawione w postaci możliwej do wyświetlenia. Po ponownym wykonaniu serwer dekoduje ciąg tekstowy i otrzymuje zdarzenie zmieniające dane. Omawiane zapytanie wymaga uprawnień SUPER.

CACHE INDEX

```
CACHE INDEX
  tbl_index_spec [, tbl_index_spec] ...
  IN cache_name
tbl_index_spec:
  tbl_name
```

```
[PARTITION ( partition_name [, partition_name] ... | ALL)]
[[INDEX | KEY] ( index_name [, index_name] ...)]
```

Powoduje utworzenie powiązania między jedną i więcej tabel MyISAM oraz wymienionym buforem kluczy, który musi już istnieć. Konieczne jest posiadanie uprawnień INDEX do każdej tabeli wymienionej w zapytaniu. Domyślny bufor kluczy ma nazwę default. Indeksy tabeli mogą być później wczytane do bufora za pomocą zapytania LOAD INDEX. Chociaż składnia zezwala na tworzenie tylko określonych indeksów, implementacja powiązuje wszystkie indeksy w tabelach z buforem.

Poniższe zapytanie powoduje buforowanie w buforze kluczy o nazwie member_cache indeksów dla zapytania member:

```
CACHE INDEX member IN member_cache;
```

W przypadku tabel partycjonowanych klauzula PARTITION zezwala na przypisanie bufora określonym partycjom.

Zapytanie CACHE INDEX generuje dane wyjściowe w formacie omówionym w podpunkcie dotyczącym zapytania CHECK TABLE.

Więcej informacji na temat zarządzania buforem kluczy MyISAM znajdziesz w punkcie 12.7.2, zatytułowanym „Dostrajanie silnika bazy danych”.

CALL

```
CALL routine_name([ proc_param [, proc_param] ...])
CALL routine_name(())
```

Powoduje wywołanie procedury składowanej. Opcjonalna lista parametrów składa się z jednej lub więcej rozdzielonych przecinkami wartości parametrów. Jeżeli którykolwiek z nich jest parametrem OUT lub INOUT, procedura może przy ich pomocy zwrócić wartość.

Po zakończeniu wykonywania procedury składowanej liczbę rekordów, na które wpływ miało wykonanie ostatniego zapytania w procedurze, można sprawdzić za pomocą wywołania funkcji ROW_COUNT(). Z poziomu kodu w języku C tę samą wartość można poznać za pomocą wywołania mysql_affected_rows().

Jeżeli procedura nie pobiera argumentów, nawias znajdujący się po jej nazwie jest opcjonalny.

CHANGE MASTER

```
CHANGE MASTER TO option [, option] ...
```

Zmienia parametry replikacji serwera podległego i pozwala na wskazanie używanego serwera głównego, sposobu nawiązywania z nim połączenia oraz wykorzystywanych dzienników zdarzeń. Serwer podległy zapisuje parametry w plikach *master.info* i *relay-log.info*, z których korzysta w trakcie kolejnych uruchomień. Wiele z wymienionych tutaj parametrów jest dostępnych także w danych wyjściowych zapytania SHOW SLAVE STATUS.

Każda *option* wskazuje definicję parametru w formacie parametr=wartość wybranego z poniższej listy:

- **IGNORE_SERVER_IDS** = (*server_id_list*)
Powoduje, że serwer podległy ignoruje zdarzenia pochodzące z dowolnych serwerów o identyfikatorach wymienionych na liście. Wspomniana lista składa się z zera lub więcej rozdzielonych przecinkami identyfikatorów serwerów. W celu wyzerowania listy ignorowanych serwerów należy po prostu podać pustą listę.
- **MASTER_BIND** = '*interface*'
Adres IP, który będzie przypisany po nawiązaniu połączenia z serwerem głównym. Ta opcja została wprowadzona w MySQL 5.6.1.
- **MASTER_CONNECT_RETRY** = *n*
Liczba sekund oczekiwania między kolejnymi próbami nawiązania połączenia z serwerem głównym.
- **MASTER_DELAY** = *n*
Liczba sekund opóźnienia replikacji. Serwer podległy będzie zwlekał z wykonaniem zdarzenia przynajmniej *n* sekund od chwili jego wykonania w serwerze głównym. Wartość domyślna wynosi 0. Ta opcja została wprowadzona w wydaniu MySQL 5.6.0.
- **MASTER_HEARTBEAT_PERIOD** = *interval*
Kiedy w serwerze głównym bez żadnego zdarzenia zapisanego w binarnym dzienniku zdarzeń mija przedział czasu oznaczający puls replikacji, serwer główny przekazuje wspomniany puls replikacji serwerowi podległemu. Ta opcja wyraża w sekundach stosowany przedział czasu. Wartość może zawierać część ułamkową wyrażoną w milisekundach. Wartość 0 powoduje wyłączenie pulsu replikacji. Wartość domyślna wynosi `slave_net_timeout/2`. Zapytanie RESET SLAVE powoduje wyzerowanie wartości pulsu replikacji do jej wartości domyślnej.
- **MASTER_HOST** = '*host_name*'
Oznacza komputer, w którym działa serwer główny replikacji.
- **MASTER_LOG_FILE** = '*file_name*'
Nazwa pliku binarnego dziennika zdarzeń w serwerze głównym, od którego będzie rozpoczynać się lub wznawiać replikacja.
- **MASTER_LOG_POS** = *n*
Położenie w pliku binarnego dziennika zdarzeń serwera głównego, od którego będzie rozpoczynać się lub wznawiać replikacja.
- **MASTER_PASSWORD** = '*pass_val*'
Hasło używane podczas nawiązywania połączenia z serwerem głównym.
- **MASTER_PORT** = *n*
Numer portu TCP/IP używanego podczas nawiązywania połączenia z serwerem głównym.

■ MASTER_RETRY_COUNT = *n*

Liczba prób nawiązania połączenia z serwerem głównym, zanim operacja zostanie uznana za zakończoną niepowodzeniem. Ta opcja została wprowadzona w MySQL 5.6.1.

■ MASTER_SSL = {0 | 1}
MASTER_SSL_CA = '*file_name*'
MASTER_SSL_CAPATH = '*dir_name*'
MASTER_SSL_CERT = '*file_name*'
MASTER_SSL_CIPHER = '*str*'
MASTER_SSL_CRL = '*file_name*'
MASTER_SSL_CRLPATH = '*dir_name*'
MASTER_SSL_KEY = '*file_name*'
MASTER_SSL_VERIFY_SERVER_CERT = {0 | 1}

Wymienione opcje pozwalają na określenie parametrów podczas nawiązywania połączenia SSL z serwerem głównym. Mają takie samo znaczenie jak odpowiadające im opcje --ssl-xxx omówione w podpunkcie F.2.1.1, zatytułowanym „Opcje standardowe SSL”. Serwer podległy zapisuje wartości tych opcji w swoim pliku *master.info*, ale nie mają one efektu, jeśli w serwerze podległym nie włączono obsługi SSL. Opcje MASTER_SSL_CRL i MASTER_CRLPATH zostały wprowadzone w wydaniu MySQL 5.6.3.

■ MASTER_USER = '*user_name*'

Nazwa użytkownika konta używanego podczas nawiązywania połączenia z serwerem głównym. Począwszy od MySQL 5.6.4, błędem jest przypisanie tej opcji wartości NULL lub pustego ciągu tekstowego bądź też pozostawienie nieustawionej mimo użycia MASTER_PASSWORD.

■ RELAY_LOG_FILE = '*file_name*'

Nazwa pliku dziennika przekazywania w serwerze podległym.

■ RELAY_LOG_POS = *n*

Bieżące położenie w dzienniku przekazywania w serwerze podległym.

Parametry, które nie zostały wymienione w zapytaniu, zachowują wartości bieżące, poza jednym wyjątkiem: zmiana MASTER_HOST lub MASTER_PORT zwykle oznacza przełączenie do innego serwera głównego. Dlatego też po podaniu jednej z wymienionych opcji, wartości opcji MASTER_LOG_FILE i MASTER_LOG_POS będą wskazywały początek pierwszego pliku binarnego dziennika zdarzeń serwera głównego.

W tym samym zapytaniu nie należy mieszać opcji MASTER_LOG_FILE i MASTER_LOG_POS z opcjami RELAY_LOG_FILE i RELAY_LOG_POS.

Zapytanie CHANGE MASTER powoduje usunięcie wszelkich istniejących plików dziennika przekazywania i rozpoczęcie tworzenia nowego, o ile nie zostały podane opcje RELAY_LOG_FILE i RELAY_LOG_POS.

CHECK TABLE

CHECK {TABLE | TABLES} *tbl_name* [, *tbl_name*] ... [*option*] ...

To zapytanie sprawdza tabelę pod kątem błędów. Działa z tabelami InnoDB, MyISAM, ARCHIVE i CSV. Zapytanie CHECK TABLE może również sprawdzić definicje widoku pod kątem problemów takich jak odwołania do nieistniejących tabel. Omawiane zapytanie wymaga uprawnienia SELECT do wszystkich sprawdzanych tabel lub analizowanego widoku.

W przypadku tabel InnoDB, jeśli znaleziony zostanie problem, wtedy serwer przerywa wykonywanie zapytania po zapisaniu komunikatu w dzienniku błędów, aby uniknąć powstania kolejnych błędów. Z kolei dla tabel MyISAM zapytanie CHECK TABLE uaktualnia także dane statystyczne indeksu.

Każda z wartości *option* może być jedną z wymienionych poniżej opcji. O ile nie zaznaczono inaczej, wymienione opcje mają zastosowanie względem tabel MyISAM. Pozostałe silniki bazy danych mogą je ignorować.

- Opcja CHANGED powoduje pominięcie sprawdzania tabeli, jeśli tabela została prawidłowo zamknięta i nie była modyfikowana od chwili jej ostatniego sprawdzenia.
- Opcja EXTENDED przeprowadza rozszerzone sprawdzenie i próbuje ustalić, czy tabela jest w pełni spójna. To jest najdokładniejszy z dostępnych typów sprawdzenia, a tym samym najwolniejszy. Na przykład, w trakcie tej operacji sprawdzane jest, czy każdy klucz w każdym indeksie prowadzi do rekordu danych.
- Opcja FAST przeprowadza operację sprawdzenia tabeli tylko wtedy, gdy nie została ona prawidłowo zamknięta.
- Opcja MEDIUM sprawdza indeks, skanuje rekordy danych pod kątem problemów i przeprowadza weryfikację sum kontrolnych. Jeżeli nie zostanie podana żadna opcja, przeprowadzony będzie ten typ sprawdzenia.
- Opcja QUICK powoduje przeskanowanie jedynie indeksów, a nie rekordów danych. Ma ona zastosowanie w przypadku tabel InnoDB i MyISAM.
- Opcja FOR UPGRADE określa, czy sprawdzana tabela jest zgodna z bieżącą wersją MySQL, a więc ta opcja jest użyteczna po przeprowadzeniu uaktualnienia oprogramowania serwera. Jeżeli występuje jakakolwiek niezgodność, wtedy serwer przeprowadza pełną operację sprawdzenia. Gdy wspomniana pełna operacja sprawdzenia zakończy się niepowodzeniem, wówczas powinienś spróbować naprawić tabelę. Serwer uaktualnia plik *.frm* tabeli bieżącą wersją MySQL, o ile nie wystąpiła niezgodność i przeprowadzona operacja pełnego sprawdzenia nie zakończyła się niepowodzeniem. Omawiana opcja nie dotyczy jedynie tabel MyISAM.

Jeżeli do sprawdzenia tabeli nie użyto opcji FOR UPGRADE i nie wskazano opcji QUICK, MEDIUM lub EXTENDED podczas sprawdzania tabeli MyISAM, wtedy zapytanie CHECK TABLE domyślnie użyje opcji MEDIUM, o ile tabela ma rekordy o zmiennej długości. W przypadku

rekordów o stałej długości domyślnie używaną opcją jest QUICK, jeśli podano CHANGED lub FAST, w przeciwnym razie domyślnie używana jest opcja MEDIUM.

Istnieje możliwość, że w pewnych sytuacjach zapytanie CHECK TABLE zmodyfikuje tabelę. Wspomniane modyfikacje polegają jedynie na ustawieniu wewnętrznej flagi. Na przykład, jeśli tabela jest oznaczona jako uszkodzona lub jako zamknięta nieprawidłowo, a operacja sprawdzenia nie znajdzie żadnych problemów, wtedy zapytanie CHECK TABLE oznacza tabelę jako dobrą.

Zapytanie CHECK TABLE zwraca informacje o wyniku operacji, na przykład:

```
mysql> CHECK TABLE t;
+-----+-----+-----+-----+
| Table | Op   | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.t | check | status   | OK       |
+-----+-----+-----+-----+
```

Zapytania ANALYZE TABLE, CACHE INDEX, LOAD INDEX INTO CACHE, OPTIMIZE TABLE i REPAIR TABLE również zwracają informacje w takim formacie. Kolumna Table wskazuje tabelę, na której została przeprowadzona operacja. W kolumnie Op podany jest typ operacji przeprowadzonej przez zapytanie. Z kolei kolumny Msg_type i Msg_text zawierają informacje o wyniku operacji. Jeżeli wartości kolumny Msg_type i Msg_text nie wskazują na poprawność tabeli, wtedy należy przeprowadzić jej naprawę.

CHECKSUM TABLE

```
CHECKSUM {TABLE | TABLES} tbl_name [, tbl_name] ...
[QUICK | EXTENDED]
```

Zapytanie podaje sumę kontrolną tabeli. W przypadku tabel partycjonowanych wartością zwrótną tego zapytania jest 0 w wersji serwera wcześniejszej niż MySQL 5.6.4, o ile nie zostanie użyta opcja EXTENDED. Zapytanie CHECKSUM TABLE wymaga uprawnienia SELECT do wszystkich sprawdzanych tabel.

```
mysql> CHECKSUM TABLE president;
+-----+-----+-----+
| Table | Checksum |
+-----+-----+
| sampdb.president | 3032762697 |
+-----+-----+
```

Jeżeli tabela nie istnieje, wtedy wartością kolumny Checksum jest NULL, a ponadto generowane będzie ostrzeżenie.

Domyślnie zapytanie podaje sumę kontrolną live, o ile taką możliwość obsługuje używany silnik bazy danych. (Suma kontrolna live to taka, która jest uaktualniana po każdej modyfikacji tabeli). W przypadku tabeli MyISAM można włączyć sumy kontrolne live przez użycie opcji CHECKSUM = 1 w zapytaniu CREATE TABLE lub ALTER TABLE.

Użycie opcji QUICK powoduje, że zapytanie zwraca sumę kontrolną live, o ile w tabeli istnieje choć jedna tego typu suma kontrolna; w przeciwnym razie wartością zwrótną jest NULL. Z kolei opcja EXTENDED powoduje, że podawana suma kontrolna jest obliczana przez odczyt całej tabeli. Im tabela większa, tym taka operacja staje się wolniejsza.

COMMIT

COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]

Zapytanie zatwierdza zmiany wprowadzone przez zapytania w bieżącej transakcji i na trwałe zapisuje je w bazie danych. Zapytanie COMMIT działa jedynie w transakcyjnych silnikach bazy danych. (W przypadku silników nietransakcyjnych zapytania są zatwierdzane w trakcie ich wykonywania).

Opcjonalne słowo kluczowe WORK nie ma żadnego efektu. Z kolei słowa kluczowe CHAIN i RELEASE wpływają na sposób obsługi przez serwer ukończenia transakcji. W przypadku AND CHAIN wraz z zakończeniem transakcji następuje rozpoczęcie kolejnej na tym samym poziomie izolacji. Z kolei w przypadku RELEASE zakończenie transakcji powoduje, że serwer kończy bieżącą sesję. Dodanie słowa kluczowego NO do CHAIN lub RELEASE nie powoduje odpowiednio rozpoczęcia nowej transakcji i przerwania bieżącej sesji. Zachowanie zapytania COMMIT w przypadku braku wymienionych klauzul jest określone przez wartość zmiennej systemowej *completion_type*. Domyślnie nie są stosowane słowa kluczowe CHAIN ani RELEASE.

Zapytanie COMMIT nie ma żadnego efektu, jeśli automatyczne zatwierdzanie zapytań nie zostało wyłączone za pomocą START TRANSACTION lub przez ustawienie wartości 0 zmiennej autocommit.

Pewne zapytania niejawnie mogą zakończyć bieżącą transakcję (podobnie jak to robi zapytanie COMMIT), ponieważ nie mogą być częścią transakcji. Ogólnie rzecz biorąc, to są zapytania typu DDL (ang. *Data Definition Language*), powodujące utworzenie, zmianę i usunięcie baz danych bądź ich obiektów, lub zapytania związane z blokadami. Na przykład, wykonanie dowolnego z poniższych zapytań w trakcie transakcji powoduje, że serwer zatwierdza transakcję przed wykonaniem zapytania:

```
ALTER TABLE
CREATE INDEX
DROP DATABASE
DROP INDEX
DROP TABLE
LOCK TABLES
RENAME TABLE
SET autocommit = 1 (jeśli ta zmienna nie ma jeszcze przypisanej wartości 1)
TRUNCATE TABLE
UNLOCK TABLES (jeżeli tabele są aktualnie zablokowane)
```

Pełną listę zapytań powodujących niejawnie zatwierdzenie transakcji znajdziesz w podręczniku użytkownika MySQL.

CREATE DATABASE

CREATE DATABASE [IF NOT EXISTS] *db_name* [*db_attr*] ...

db_attr:

```
[DEFAULT] CHARACTER SET [=] charset
| [DEFAULT] COLLATE [=] collation
```

Zapytanie powoduje utworzenie bazy danych o podanej nazwie. Konieczne jest posiadanie uprawnień CREATE dla bazy danych. Próba utworzenia bazy danych o istniejącej nazwie normalnie powoduje wygenerowanie błędu. Jeżeli zostanie użyta klauzula IF NOT EXISTS, wtedy baza danych nie zostanie utworzona, ale również nie będzie wygenerowany błąd.

Opcjonalne atrybuty CHARACTER SET i COLLATE mogą być podane w celu wskazania domyślnego kodowania znaków i kolejności sortowania w bazie danych. Wymienione atrybuty są używane dla tabel, dla których nie podano wyraźnie kodowania znaków lub kolejności sortowania. *charset* może być nazwą wybranego kodowania lub mieć wartość DEFAULT, która powoduje użycie bieżącego kodowania znaków serwera. Z kolei *collation* może być nazwą wybranej kolejności lub mieć wartość DEFAULT, oznaczającą użycie bieżącej kolejności sortowania serwera.

Jeżeli nie zostaną podane wymienione argumenty, użyte będą wartości zdefiniowane dla serwera. Podanie CHARACTER SET bez COLLATE powoduje użycie domyślnej kolejności sortowania dla wskazanego kodowania znaków. Natomiast podanie COLLATE bez CHARACTER SET powoduje określenie kodowania znaków na podstawie wskazanej kolejności sortowania. W przypadku podania zarówno CHARACTER SET, jak i COLLATE podana kolejność sortowania musi być zgodna z podanym kodowaniem znaków.

Atrybuty bazy danych są przez MySQL przechowywane w pliku *db.opt*, znajdującym się w katalogu bazy danych.

CREATE EVENT

```
CREATE
  [DEFINER = definer_name]
  EVENT [IF NOT EXISTS] event_name
  ON SCHEDULE schedule
  [ON COMPLETION [NOT] PRESERVE]
  [ENABLE | DISABLE | DISABLE ON SLAVE]
  [COMMENT 'str']
  DO event_stmt

schedule:
  AT datetime
  | EVERY expr interval [STARTS datetime] [ENDS datetime]
```

Zapytanie tworzy nowe zdarzenie o podanej nazwie i umieszcza je w harmonogramie zdarzeń. Konieczne jest posiadanie uprawnień EVENT dla bazy danych, do której będzie należało zdarzenie. Domyślnie zdarzenia są tworzone w domyślnej bazie danych. W celu utworzenia zdarzenia we wskazanej bazie danych należy użyć formatu *nazwa_bazy_danych.nazwa_zdarzenia*.

Klauzula DEFINER powoduje określenie kontekstu bezpieczeństwa (konta użytkownika sprawdzanego podczas weryfikacji uprawnień dostępu) podczas wykonywania zdarzenia, jak to omówiono w podrozdziale 4.3, zatytułowanym „Zapewnienie bezpieczeństwa widokom i programom składowanym”. Domyślnie używane jest konto użytkownika, który wykonuje zapytanie CREATE EVENT.

Klauzula `ON SCHEDULE` pozwala na zdefiniowanie harmonogramu wykonywania danego zdarzenia (przy założeniu, że harmonogram zdarzeń jest uruchomiony). W formatach dla tej klauzuli *datetime* oznacza wartość daty i godziny. Funkcję `CURRENT_TIMESTAMP()` lub jej synonim można wykorzystać do przedstawienia bieżącej daty i godziny. Wyrażenie *datetime* może używać `INTERVAL expr interval` w celu dodania lub odjęcia przedziału czasu. Taka składnia została omówiona w sekcji dotyczącej funkcji `DATE_ADD()` w punkcie C.2.5, zatytułowanym „Funkcje daty i godziny”. Wartość *interval* nie powinna używać żadnych specyfikatorów zawierających część ułamkową.

Dla klauzuli `ON SCHEDULE` typ `AT` powoduje zdefiniowanie zdarzenia wykonywanego jednokrotnie o wskazanej godzinie. Typ `EVERY` oznacza zdarzenie powtarzające się we wskazanym odstępie czasu. Człon definiujący powtarzanie zdarzenia składa się z liczby powtórzeń oraz modyfikatora *interval* wskazującego sposób interpretacji przedziału czasu (na przykład, `5 HOUR` lub `'1:30' MINUTE_SECOND`). Domyślnie, pierwsze wykonanie zdarzenia następuje po jego utworzeniu, a następnie w zdefiniowanych odstępach czasu. Jeśli podana będzie klauzula `STARTS`, określa datę i godzinę rozpoczęcia wykonywania zdarzenia. Z kolei klauzula `ENDS` wskazuje datę i godzinę, po której zdarzenie nie będzie dłużej uruchamiane. W klauzuli `ON SCHEDULE` nie należy używać odwołań do tabel, funkcji składowanych lub funkcji zdefiniowanych przez użytkownika.

Klauzula `DO` wskazuje zapytanie wykonywane po uruchomieniu zdarzenia. To powinno być pojedyncze zapytanie SQL. Aby móc użyć wielu zapytań, należy je umieścić w bloku `BEGIN` i `END`, tworząc w ten sposób zapytanie złożone; patrz podrozdział E.2, zatytułowany „Składnia zapytań SQL (zapytania złożone)”.

Domyślnie serwer usuwa zdarzenie po zakończeniu jego ostatniego uruchomienia. Klauzula `ON COMPLETION NOT PRESERVE` wyraźnie definiuje takie zachowanie. Natomiast klauzula `ON COMPLETION PRESERVE` oznacza zachowanie zdarzenia po jego ostatnim uruchomieniu.

Opcje `ENABLE` i `DISABLE` wskazują stan zdarzenia w trakcie jego tworzenia jako odpowiednio włączone (uruchamiane zgodnie z harmonogramem) lub wyłączone (nieuruchamiane). Opcja `DISABLE ON SLAVE` wskazuje, że zdarzenie jest włączone w serwerze, w którym zostało utworzone, ale jest wyłączone w serwerach podległych replikacji.

W trakcie tworzenia zdarzenia wartość bieżąca zmiennej systemowej `sql_mode` jest zapisywana w celu jej użycia podczas wykonywania zdarzenia.

Zdarzenia nie pobierają danych wejściowych i nie generują danych wyjściowych. Oznacza to brak możliwości przekazania parametrów do zdarzenia oraz odrzucenie danych wyjściowych zapytań generujących zbiór wynikowy, takich jak `SELECT`.

CREATE FUNCTION, CREATE PROCEDURE

```
CREATE
[DEFINER = definer_name]
FUNCTION routine_name ([func_param [, func_param] ...])
RETURNS type
[characteristic] ...
routine_stmt
```

```

CREATE
  [DEFINER = definer_name]
  PROCEDURE routine_name ([proc_param [, proc_param] ...])
  [characteristic] ...
  routine_stmt

func_param:
  param_name type

proc_param:
  [IN | OUT | INOUT] param_name type

characteristic:
  COMMENT 'str'
  | {CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA}
  | [NOT] DETERMINISTIC
  | LANGUAGE SQL
  | SQL SECURITY {DEFINER | INVOKER}

```

Wymienione zapytania powodują utworzenie nowej procedury składowanej (funkcji i procedury). Konieczne jest posiadanie uprawnień CREATE ROUTINE do danej procedury.

Domyślnie procedury są tworzone w domyślnej bazie danych. W celu utworzenia zdarzenia we wskazanej bazie danych należy użyć formatu *nazwa_bazy_danych.nazwa_procedury*. Funkcja i procedura mogą mieć takie same nazwy, ale w bazie danych nie mogą znajdować się dwie funkcje lub procedury o takiej samej nazwie.

Każdy parametr funkcji jest definiowany przez nadanie mu nazwy i typu. Typ powinien być dowolnym poprawnym typem danych obsługiwany przez MySQL. Parametry dostarczają wartości do funkcji w trakcie jej wywołania, ale zmiany parametrów wprowadzane przez funkcję nie są widoczne dla wywołującego po zakończeniu działania funkcji (czyli są traktowane jako parametry typu IN).

W przypadku funkcji polecenie RETURNS musi zawierać listę parametrów wskazujących typ danych dla wartości zwrótej.

Każdy parametr procedury także jest zdefiniowany za pomocą nazwy i typu, ale nazwa może być poprzedzona słowem kluczowym IN, OUT lub INOUT, wskazującym, że parametr jest odpowiednio tylko danych wejściowych, tylko danych wyjściowych lub danych zarówno wejściowych, jak i wyjściowych. Jeżeli żadne z wymienionych słów kluczowych nie zostanie podane, domyślnie użyte będzie IN.

- Parametr IN dostarcza wartość procedurze. Zmiany parametru wprowadzane wewnątrz procedury są niewidoczne dla wywołującego ją programu po zakończeniu działania procedury.
- Parametr OUT nie dostarcza wartości procedurze. Jego wartością początkową w procedurze jest NULL i może być ona modyfikowana wewnątrz procedury. Ostateczna wartość parametru jest widoczna dla wywołującego ją programu po zakończeniu działania procedury.
- Parametr INOUT dostarcza wartość procedurze, a wszelkie zmiany parametru wprowadzane wewnątrz procedury są widoczne dla wywołującego ją programu po zakończeniu działania procedury.

Istnieje możliwość podania jednej lub więcej wartości *characteristics*, które powinny być rozdzielone spacjami:

■ COMMENT

Opisowy komentarz danej procedury. Ten komentarz jest wyświetlany przez zapytania SHOW, wyświetlające informacje o procedurze.

■ CONTAINS SQL, NO SQL, READS SQL DATA, MODIFIES SQL DATA

Wymienione cechy charakterystyczne zapewniają wskazówki dotyczące uzyskiwanego przez procedurę sposobu dostępu do danych. W MySQL nie mają żadnego efektu w określeniu zapytań, na których wykonanie przez procedurę faktycznie zezwoli serwer.

- ◆ CONTAINS SQL — procedura zawiera zapytania SQL. To jest wartość domyślna, jeśli nie zostaną podane żadne cechy charakterystyczne wskazujące na sposób dostępu do bazy danych.
- ◆ NO SQL — procedura nie zawiera zapytań SQL.
- ◆ READS SQL DATA — procedura zawiera zapytania SQL przeznaczone do odczytu, ale już nie do modyfikacji danych.
- ◆ MODIFIES SQL DATA — procedura zawiera zapytania, które mogą modyfikować dane.

■ DETERMINISTIC, NOT DETERMINISTIC

DETERMINISTIC wskazuje, że funkcja zawsze generuje ten sam wynik po jej wywołaniu z tymi samymi wartościami parametru. Z kolei NOT DETERMINISTIC wskazuje sytuację zupełnie odwrotną. Na przykład, funkcja używająca NOW() jako wartości zwrotnej najprawdopodobniej będzie niedeterministyczna.

■ LANGUAGE SQL

Wskazuje język procedury. Ta wartość jest przetwarzana i ignorowana, MySQL obsługuje jedynie SQL jako język procedur składowanych.

■ SQL SECURITY

Ta cecha charakterystyczna w połączeniu z DEFINER określa kontekst bezpieczeństwa (konta użytkownika sprawdzanego podczas weryfikacji uprawnień dostępu) podczas wykonywania procedury, jak to omówiono w podrozdziale 4.3, zatytułowanym „Zapewnienie bezpieczeństwa widokom i programom składowanym”. W przypadku pominięcia klauzuli DEFINER domyślnie używane jest konto użytkownika, który wykonuje zapytanie CREATE. Wykorzystywane konto użytkownika musi posiadać uprawnienie EXECUTE, aby było możliwe wywołanie procedury składowanej. Domyślnie serwer MySQL automatycznie nadaje uprawnienia EXECUTE i ALTER ROUTINE twórcy procedury i odbiera wymienione uprawnienia po usunięciu procedury. W celu wyłączenia takiego zachowania należy ustawić wartość zero zmiennej systemowej `automatic_sp_privileges`.

routine_stmt to zapytanie SQL tworzące część główną procedury. To powinno być pojedyncze zapytanie SQL. Aby móc użyć wielu zapytań, należy je umieścić w bloku BEGIN i END, tworząc w ten sposób zapytanie złożone; patrz podrozdział E.2, zatytułowany „Składnia zapytań SQL (zapytania złożone)”.

Funkcja zwraca wywołującemu wartość, a więc musi posiadać przynajmniej jedno polecenie RETURN. Jednak funkcja nie może wykonywać zapytań generujących zbiór wynikowy.

W trakcie tworzenia procedury wartość bieżąca zmiennej systemowej `sql_mode` jest zapisywana w celu jej użycia podczas wykonywania procedury.

CREATE INDEX

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX index_name
ON tbl_name (index_columns) [index_option] ...
```

```
index_option:
    index_type
    | COMMENT 'str'
    | KEY_BLOCK_SIZE [=] n
    | WITH PARSER parser_name
index_type: USING {BTREE | HASH | RTREE}
```

Zapytanie dodaje wskazany indeks (*index_name*) do tabeli (*tbl_name*). Indeks jest oparty na kolumnach wymienionych w parametrze *index_columns*, który jest listą składającą się z jednej lub więcej kolumn w tabeli rozdzielonych przecinkami. Początkowo MySQL obsługuje to zapytanie jak ALTER TABLE. Informacje szczegółowe znajdziesz w podpunkcie dotyczącym zapytania ALTER TABLE. W celu utworzenia wielu indeksów w tabeli preferowanym rozwiązaniem jest użycie zapytania ALTER TABLE i dodanie wszystkich indeksów za pomocą pojedynczego zapytania, co jest szybsze od dodawania ich pojedynczo.

Domyślnie tworzony jest indeks nieunikalny. Słowa kluczowe UNIQUE, FULLTEXT lub SPATIAL, o ile zostaną podane, wskazują na utworzenie określonego rodzaju indeksu. Zapytanie CREATE INDEX nie może być używane do tworzenia PRIMARY KEY, zamiast tego należy skorzystać z ALTER TABLE.

Indeksy typu FULLTEXT są obsługiwane przez tabele MyISAM lub (począwszy od wydania MySQL 5.6.4) InnoDB, ale jedynie dla kolumn niebinarnych ciągów tekstowych (CHAR, VARCHAR, TEXT). Z kolei indeksy typu SPATIAL są obsługiwane jedynie dla tabel MyISAM i tylko dla kolumn NOT NULL przestrzennych typów danych.

Na końcu definicji indeksu dozwolone jest podanie następujących wartości *index_option*:

- *index_type* wskazuje algorytm indeksowania, który jest dozwolony dla niektórych silników bazy danych. Wartością algorytmu może być BTREE dla tabel InnoDB i MyISAM, HASH lub BTREE dla tabel MEMORY bądź też RTREE lub SPATIAL dla tabel MyISAM.
- COMMENT '*str*' pozwala na umieszczenie opisowego komentarza dla indeksu (maksymalnie 1024 znaki). Ta opcja została wprowadzona w MySQL 5.5.3.

- `KEY_BLOCK_SIZE [=] n` sugeruje wyrażoną w bajtach wielkość, której silnik bazy danych powinien użyć dla bloków kluczy w indeksie. Wartość 0 oznacza użycie wartości domyślnej.
- `WITH PARSER nazwa` jest dozwolone jedynie dla indeksów typu `FULLTEXT`. Wartość powinna zawierać nazwę wtyczki analizatora pełnego tekstu użytego dla indeksu. Więcej informacji na ten temat znajdziesz w podręczniku użytkownika MySQL.

Informacje dodatkowe dotyczące tworzenia indeksu znajdziesz również w punkcie 2.6.4, zatytułowanym „Indeksowanie tabel”.

CREATE TABLE

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
{
    (create_definition,...) [table_option] ...
    [partition_scheme] [trailing_select]
| (create_definition,...) [table_option] ...
    [partition_scheme] trailing_select
| LIKE tbl_name2
| (LIKE tbl_name2)
}
```

table_option: (patrz opis w tekście)

trailing_select:
[IGNORE | REPLACE] [AS] *select_stmt*

create_definition:
col_name col_definition [*reference_definition*]
| [CONSTRAINT [*name*]] PRIMARY KEY
 [*index_name*]
 (*index_columns*) [*index_option*] ...
| [CONSTRAINT [*name*]] UNIQUE [INDEX | KEY]
 [*index_name*]
 (*index_columns*) [*index_option*] ...
| {INDEX | KEY} [*index_name*]
 (*index_columns*) [*index_option*] ...
| {FULLTEXT | SPATIAL} [INDEX | KEY]
 [*index_name*] (*index_columns*) [*index_option*] ...
| [CONSTRAINT [*name*]] FOREIGN KEY [*fk_name*]
 (*index_columns*) [*reference_definition*]
| CHECK (*expr*)

col_definition:
data_type
[NOT NULL | NULL] [DEFAULT *default_value*]
[AUTO_INCREMENT] [PRIMARY KEY] [UNIQUE [KEY]]
[COMMENT '*str*']

index_option: (patrz opis w tekście)

reference_definition:
REFERENCES *tbl_name* (*index_columns*)
[MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]


```
[ON DELETE reference_action]
[ON UPDATE reference_action]
```

reference_action:

```
RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
```

partition_scheme:

```
PARTITION BY
{
    RANGE (expr)
  | RANGE COLUMNS (col_list)
  | LIST (expr)
  | LIST COLUMNS (col_list)
  | [LINEAR] HASH (expr)
  | [LINEAR] KEY (col_list)
}
[PARTITIONS n]
[SUBPARTITION BY
{
    [LINEAR] HASH (expr)
  | [LINEAR] KEY (col_list)
}
[SUBPARTITIONS n]
]
[( partition_definition [, partition_definition] ...)]
```

partition_definition:

```
PARTITION partition_name
[VALUES {LESS THAN {(expr) | MAXVALUE} | IN (value_list)}]
[partition_option] ...
[(subpartition_definition [, subpartition_definition] ...)]
```

subpartition_definition:

```
SUBPARTITION subpartition_name
[partition_option] ...
```

partition_option: (patrz opis w tekście)

Zapytanie CREATE TABLE powoduje utworzenie nowej tabeli o wskazanej nazwie (*tbl_name*) w domyślnej bazie danych. Aby utworzyć tabelę w konkretnej bazie danych, należy skorzystać z formatu *nazwa_bazy_danych.nazwa_tabeli*. Konieczne jest posiadanie uprawnień CREATE dla tabeli.

Normalnie próba utworzenia tabeli o nazwie istniejącej tabeli spowoduje wygenerowanie błędu. Wspomniany błąd nie zostanie wygenerowany w dwóch przypadkach. Po pierwsze, użycie klauzuli IF NOT EXISTS powoduje, że tabela nie będzie utworzona, a błąd nie zostanie wygenerowany. Po drugie, jeżeli użyte będzie słowo kluczowe TEMPORARY, a istniejąca tabela o takiej samej nazwie nie jest tabelą tymczasową, wtedy zostanie utworzona nowa tabela. Istniejąca tabela o takiej samej nazwie będzie ukryta przed klientem, dopóki istnieje tabela tymczasowa. Oryginalna tabela będzie jednak dostępna dla innych klientów, ponieważ tabela tymczasowa jest widoczna jedynie dla klienta, który ją utworzył. Natomiast dla bieżącego klienta oryginalna tabela stanie się ponownie widoczna po wykonaniu zapytania DROP TABLE dla tabeli tymczasowej lub po nadaniu jej innej nazwy. W celu utworzenia tabeli tymczasowej konieczne jest uprawnienie CREATE TEMPORARY TABLE.

Jeżeli użyte zostanie słowo kluczowe `TEMPORARY`, tak utworzona tabela istnieje jedynie do chwili zakończenia bieżącej sesji klienta lub wykonania zapytania `DROP TABLE` usuwającego tę tabelę.

Lista `create_definition` zawiera listę nazw kolumn i indeksów przeznaczonych do utworzenia. Ta lista jest opcjonalna, jeśli tworzysz tabelę na podstawie zapytania `SELECT` zdefiniowanego na końcu zapytania `CREATE TABLE`. Wartości `table_option` pozwalają na podanie właściwości tabeli. Z kolei `partition_scheme` definiuje cechy charakterystyczne partycjonowania, jeśli tabela ma być partycjonowana. W przypadku podania zapytania `select_stmt` (w postaci dowolnego zapytania `SELECT`) tabela zostanie utworzona na podstawie zbioru wynikowego zwróconego przez to zapytanie. Umieszczenie na końcu klauzuli `LIKE` powoduje utworzenie nowej tabeli jako pustej kopii istniejącej tabeli. W dalszej części dodatku wymienione klauzule zostaną omówione znacznie dokładniej.

Definicje kolumn i indeksu. Użyta w zapytaniu `create_definition` może być definicją kolumny lub indeksu, klauzulą `FOREIGN KEY` bądź też klauzulą `CHECK`. Klauzula `CHECK` jest przetwarzana, ale ignorowana. Z kolei klauzula `FOREIGN KEY` jest traktowana podobnie, ale za wyjątkiem tabel InnoDB.

Definicja kolumny `col_definition` rozpoczyna się od typu danych (`data_type`) i może zawierać wiele opcjonalnych słów kluczowych. Typem może być dowolny typ danych wymieniony w dodatku B, zatytułowanym „Przewodnik po typach danych”. Zajrzyj do wymienionego dodatku, jeśli chcesz poznać atrybuty charakterystyczne dla wybranego typu danych przypisanego definiowanym kolumnom. Poniżej wymieniono słowa kluczowe, które można podać po typie danych:

- **NULL, NOT NULL**

Wskazuje, czy kolumna może lub nie może zawierać wartości `NULL`. Jeżeli nie będzie podane, domyślnie przyjmuje się użycie `NULL`.

- **DEFAULT `default_value`**

Pozwala na podanie wartości domyślnej dla kolumny. To słowo kluczowe nie może być używane dla typów `BLOB`, `TEXT`, przestrzennych typów danych lub kolumn z atrybutem `AUTO_INCREMENT`. Za wyjątkiem `TIMESTAMP` i (począwszy od MySQL 5.6.5) `DATETIME`, wartości domyślne muszą być stałymi podanymi w postaci liczby, ciągu tekstowego lub wartości `NULL`. Reguły stosowane przez MySQL podczas przypisywania wartości domyślnej, jeśli nie użyto klauzuli `DEFAULT`, zostały omówione w punkcie 3.2.3, zatytułowanym „Definiowanie wartości domyślnych kolumn”.

- **AUTO_INCREMENT**

To słowo kluczowe ma zastosowanie jedynie do typów danych liczb całkowitych i zmiennoprzecinkowych. Kolumna `AUTO_INCREMENT` po wstawieniu do niej wartości `NULL` zachowuje się w sposób specjalny, to znaczy faktycznie wstawioną wartością jest kolejna wartość w sekwencji kolumny. Zwykle to będzie wartość o jeden większa od obecnej wartości maksymalnej w kolumnie. Domyślnie, wartości `AUTO_INCREMENT` rozpoczynają się od 1. (Pewne silniki baz danych pozwalają na podanie wartości początkowej dla `AUTO_INCREMENT`.)

Zapoznaj się z przedstawioną nieco dalej analizą dotyczącą opcji tabeli).

Kolumna musi być również indeksowana i powinna mieć zdefiniowaną opcję NOT NULL. W tabeli może znajdować się co najwyżej tylko jedna kolumna AUTO_INCREMENT.

■ [PRIMARY] KEY

Wskazuje, że kolumna jest kluczem podstawowym (PRIMARY KEY). Tego rodzaju kolumna musi być zdefiniowana jako NOT NULL, więc MySQL dodaje NOT NULL do definicji kolumny, jeśli samodzielnie tego nie zrobisz.

■ UNIQUE [KEY]

Wskazuje, że kolumna jest indeksem UNIQUE.

■ COMMENT '*str*'

Opisowy komentarz dla kolumny; może składać się z maksymalnie 1024 znaków. Ten atrybut jest wyświetlany przez zapytania SHOW CREATE TABLE i SHOW FULL COLUMNS.

Klauzule PRIMARY KEY, UNIQUE, INDEX, KEY, FULLTEXT i SPATIAL określają indeksy. PRIMARY KEY i UNIQUE oznacza, że indeksy muszą zawierać unikalne wartości. INDEX i KEY są synonimami, oznaczają, że indeksy mogą zawierać powtarzające się wartości. Sam indeks opiera się na kolumnach wymienionych w parametrze *index_columns*, który powinien zawierać jedną lub więcej rozdzielonych przecinkami kolumn tabeli. Jeżeli pominięta zostanie *index_name*, wtedy MySQL wybierze ją automatycznie na podstawie nazwy pierwszej kolumny indeksu.

Indeksy typu FULLTEXT są obsługiwane przez tabele MyISAM lub (począwszy od wydania MySQL 5.6.4) InnoDB, ale jedynie dla kolumn niebinarnych ciągów tekstowych (CHAR, VARCHAR, TEXT). Z kolei indeksy typu SPATIAL są obsługiwane jedynie dla tabel MyISAM i tylko dla kolumn NOT NULL przestrzennych typów danych.

Na końcu definicji indeksu dozwolone jest podanie wartości *index_option*, które zostały omówione w podpunkcie dotyczącym zapytania CREATE INDEX. Więcej informacji na temat tworzenia indeksów znajdziesz w punkcie 2.6.4, zatytułowanym „Indeksowanie tabel”.

Opcje tabeli. Każda wartość *table_option* wskazuje dodatkową cechę charakterystyczną tabeli wybraną z przedstawionej poniżej listy. Jeśli nie zostanie zaznaczone inaczej, każda opcja ma zastosowanie we wszystkich silnikach bazy danych. Opcje można rozdzielić przecinkami lub znakami odstępu.

■ AUTO_INCREMENT [=] *n*

Pierwsza wartość, AUTO_INCREMENT, jaka zostanie wygenerowana w tabeli. Ta opcja jest używana w tabelach InnoDB, MyISAM i MEMORY. W przypadku tabel InnoDB efekt użycia opcji będzie zniesiony, jeśli nastąpi ponowne uruchomienie serwera przed wygenerowaniem jakiegokolwiek wartości AUTO_INCREMENT.

■ AVG_ROW_LENGTH [=] *n*

Przybliżona średnia długość rekordu w tabeli. W przypadku tabel MyISAM serwer MySQL używa iloczynu wartości AVG_ROW_LENGTH i MAX_ROWS do ustalenia

maksymalnej wielkości pliku danych. Silnik bazy danych MyISAM może używać wewnętrznych wskaźników rekordu o wielkości od 2 do 7 bajtów. Domyślna wielkość wskaźnika jest wystarczająca do zapewnienia obsługi tabel o wielkości do 256 TB. Jeżeli musisz użyć większej tabeli (i używany system operacyjny obsługuje ogromne pliki), wtedy opcje `AVG_ROW_LENGTH` i `MAX_ROWS` zapewniają informacje pozwalające MyISAM na dostosowanie wielkości wewnętrznego wskaźnika. Ogromne iloczyny wymienionych wartości powodują użycie większych wskaźników, co z kolei oznacza możliwość obsługi plików o wielkości do 65536 TB. Natomiast mały iloczyn przekłada się na użycie mniejszych wskaźników. W przypadku pojedynczej małej tabeli oszczędność nie będzie duża, ale jeśli korzystasz z wielu tabel, wówczas skumulowane oszczędności mogą być już znaczne.

W celu bezpośredniego zdefiniowania wielkości wskaźnika danych należy przypisać odpowiednią wartość zmiennej systemowej `myisam_data_pointer_size` jeszcze przed utworzeniem tabeli.

■ **[DEFAULT] CHARACTER SET [=] *charset***

To jest domyślne kodowanie znaków w tabeli. Można podać wybrane kodowanie znaków (wartość *charset*) lub wartość `DEFAULT`, oznaczającą użycie kodowania znaków stosowanego w bazie danych. Omawiana opcja określa znaki, jakich będzie można używać w kolumnach typu ciągu tekstowego, jeśli dla danej kolumny nie podano wyraźnie kodowania znaków. W poniższym przykładzie kolumnom `c1` i `c2` zostały przypisane kodowania znaków odpowiednio `sjis` i `ujis`:

```
CREATE TABLE t
(
  c1 CHAR(50) CHARACTER SET sjis,
  c2 CHAR(50)
) CHARACTER SET ujis;
```

Omawiana opcja tabeli ma również zastosowanie względem kolejnych modyfikacji tabeli przeprowadzanych za pomocą zapytania `ALTER TABLE` dla zmian definicji kolumny typu ciągu tekstowego, w których nie są wyraźnie podane kodowania znaków.

■ **CHECKSUM [=] {0 | 1}**

Jeżeli tej opcji zostanie przypisana wartość 1, wtedy MySQL będzie obsługiwać sumę kontrolną live dla danej tabeli. Tego rodzaju suma kontrolna jest uaktualniana w trakcie każdej modyfikacji tabeli. Istnieje drobne opóźnienie w trakcie uaktualnienia tabeli, ale obecność sum kontrolnych usprawnia proces sprawdzania tabeli. (Tylko tabele MyISAM).

■ **[DEFAULT] COLLATE [=] *collation***

To kolejność sortowania domyślnego kodowania znaków tabeli. Wartość *collation* może być nazwą wybranej kolejności sortowania lub wartością `DEFAULT`, oznaczającą użycie domyślnej kolejności sortowania dla kodowania znaków tabeli.

■ COMMENT [=] 'str'

Opisowy komentarz dla kolumny, może składać się z maksymalnie 2048 znaków. Ten komentarz jest wyświetlany przez zapytania SHOW CREATE TABLE i SHOW TABLE STATUS.

■ DATA DIRECTORY [=] 'dir_name'

Ta opcja jest używana jedynie dla tabel MyISAM i tylko w systemach UNIX. Wskazuje katalog, w którym zapisywany będzie plik danych (.myd). Wartość *dir_name* musi być pełną ścieżką dostępu. Omawiana opcja działa jedynie po uruchomieniu serwera bez opcji --skip-symbolic-links. W pewnych systemach UNIX dowiązania symboliczne nie zapewniają bezpieczeństwa wątków i są domyślnie wyłączone. Ta opcja jest ignorowana przez tabele partycjonowane. Jeżeli zmienna systemowa keep_files_on_create została ustawiona, nastąpi wygenerowanie błędu, jeśli dla tabeli podanej w katalogu istnieje już plik .myd.

■ DELAY_KEY_WRITE [=] {0 | 1}

Jeżeli ta opcja ma przypisaną wartość 1, bufor kluczy dla tabeli będzie opróżniany okresowo zamiast po każdej operacji wstawienia danych. W ten sposób następuje poprawa wydajności działania, ale jednocześnie w przypadku wystąpienia awarii wymagane jest przeprowadzenie operacji naprawy tabeli. (Tylko tabele MyISAM).

■ ENGINE [=] engine_name

Silnik bazy danych używany przez tabelę. Jeżeli serwer nie zostanie skonfigurowany inaczej, domyślnym silnikiem bazy danych jest InnoDB. W celu uruchomienia serwera z innym silnikiem bazy danych należy skorzystać ze wskazówek przedstawionych w punkcie 12.5.2, zatytułowanym „Wybór domyślnego silnika bazy danych”. Nazwy obsługiwanych silników bazy danych można wyświetlić za pomocą zapytania SHOW ENGINES. Jeżeli spróbujesz utworzyć tabelę, używając nieobsługiwanego silnika bazy danych, efektem wykonania zapytania będzie ustawienie trybu SQL o nazwie NO_ENGINE_SUBSTITUTION. Silniki baz danych zostały omówione w punkcie 2.6.1, zatytułowanym „Cechy charakterystyczne silników bazy danych”.

■ INDEX DIRECTORY [=] 'dir_name'

Ta opcja działa podobnie jak DATA DIRECTORY (i ma takie same ograniczenia), ale wskazuje katalog, w którym zostanie zapisany plik indeksu (.myi).

■ INSERT_METHOD [=] {NO | FIRST | LAST}

Ta opcja jest używana przez tabele MERGE w celu wskazania sposobu wstawiania rekordów. Wartość NO całkowicie uniemożliwia wstawianie rekordów. Z kolei wartości FIRST i LAST wskazują, że rekordy powinny być wstawione do pierwszej lub ostatniej tabeli MyISAM tworzącej tabelę MERGE.

■ KEY_BLOCK_SIZE [=] n

Wyrażona w bajtach sugerowana wielkość, jakiej silnik bazy danych powinien użyć dla bloków kluczy w indeksach. Wartość 0 oznacza użycie wartości domyślnej. Wspomniana wartość domyślna tabeli może być nadpisana przez definicję indeksu zawierającą własną opcję KEY_BLOCK_SIZE.

■ **MAX_ROWS [=] *n***

Wskazówka udzielana silnikowi bazy danych, dotycząca maksymalnej liczby rekordów, jakie mają zostać umieszczone w tabeli. Tabela zostanie utworzona w taki sposób, aby umożliwić przechowywanie co najmniej podanej liczby rekordów. W opisie opcji `AVG_ROW_LENGTH` znajdziesz informacje o przeznaczeniu tej wartości.

■ **MIN_ROWS [=] *n***

Wskazówka udzielana silnikowi bazy danych, dotycząca minimalnej liczby rekordów, jakie mają zostać umieszczone w tabeli. Omawiana opcja może być używana dla tabel `MEMORY` i stanowi wskazówkę dla silnika bazy danych `MEMORY` odnośnie sposobu optymalizacji użycia pamięci.

■ **PACK_KEYS [=] {0 | 1 | DEFAULT}**

Ta opcja określa sposób kompresji indeksów `MyISAM`, co pozwala na kompresję podobnych wartości indeksu. Efektem jej użycia najczęściej jest skrócenie czasu pobierania i, niestety, wydłużenie czasu operacji uaktualniania. Wartość 0 oznacza brak kompresji indeksu. Wartość 1 oznacza kompresję indeksów w postaci ciągu tekstowego (`CHAR`, `VARCHAR`, `BINARY`, `VARBINARY`) i liczb. Z kolei wartość `DEFAULT` powoduje kompresję jedynie kolumn długich ciągów tekstowych.

■ **ROW_FORMAT [=]**

{`DEFAULT` | `COMPACT` | `COMPRESSED` | `DYNAMIC` | `FIXED` | `REDUNDANT`}

Ta opcja określa format rekordu. Wartość `DEFAULT` pozwala silnikowi bazy danych na wybór formatu domyślnego. Silnik bazy danych może zignorować tę opcję, jeśli podany format rekordu nie będzie mógł zostać użyty. Na przykład, formatu `FIXED` nie można użyć, jeśli tabela zawiera kolumny `BLOB` lub `TEXT`. Wykonaj zapytanie `SHOW TABLE STATUS` i sprawdź wartość `Row_format`, aby przekonać się, jaki format rekordu został wybrany przez silnik bazy danych.

W przypadku tabel `InnoDB` formatem domyślnym jest `COMPACT`. Starszy, początkowy format `InnoDB` można wybrać, podając wartość `REDUNDANT`. Jeżeli ustawisz zmienne systemowe pozwalające silnikowi `InnoDB` na tworzenie tabel w formacie pliku `BARRACUDA`, wtedy do wyboru masz jeszcze inne formaty rekordu. Musisz zacząć od ustawienia zmiennych systemowych `innodb_file_per_table=1` i `innodb_file_format=Barracuda`. (Patrz podpunkt 12.5.3.1.4, zatytułowany „Używanie oddzielnych przestrzeni tabel dla każdej tabeli `InnoDB`”). Następnie opcji `ROW_FORMAT` możesz przypisać wartość `COMPRESSED` lub `DYNAMIC`. Więcej informacji na temat cech charakterystycznych wymienionych formatów rekordu znajdziesz w podrozdziale 5.4, zatytułowanym „Wybór formatu tabeli dla efektywnych zapytań”.

W przypadku tabel `MyISAM` wartość `DYNAMIC` lub `FIXED` wskazuje format rekordu o odpowiednio zmiennej i stałej długości. Jeżeli użyjesz programu `mysampack` do kompresji tabeli `MyISAM` (który również powoduje, że jest ona tylko do odczytu), wtedy `mysampack` ustawia format rekordu jako `COMPRESSED`.

■ **UNION [=] (*tbl_list*)**

Ta opcja jest używana dla tabel MERGE. Zawiera rozdzieloną przecinkami listę tabel MyISAM tworzących tabelę typu MERGE.

Zapytanie SELECT na końcu. Jeżeli podana będzie klauzula *select_stmt* (jako zapytanie SELECT na końcu), wtedy tabela zostanie utworzona na podstawie zawartości zbioru wynikowego zwróconego przez to zapytanie. Rekordy powielające wartości w unikalnym indeksie zostaną zignorowane lub zastąpią istniejące rekordy, w zależności od użycia słowa kluczowego IGNORE lub REPLACE. Jeżeli żadne z wymienionych słów kluczowych nie zostanie podane, wykonanie zapytania będzie przerwane i nastąpi wygenerowanie błędu.

Klauzula LIKE na końcu. Podanie klauzuli LIKE *tbl_name2* na końcu powoduje utworzenie tabeli jako pustej kopii tabeli o podanej nazwie (*tbl_name2*). Konieczne jest posiadanie uprawnienia SELECT do *tbl_name2*. Kopia będzie zawierała te same definicje kolumn, indeksu i opcji tabeli, poza wymienionymi wyjątkami: opcje DATA DIRECTORY i INDEX DIRECTORY nie są kopiowane, podobnie jak definicje kluczy zewnętrznych.

Obsługa klucza zewnętrznego. Silnik InnoDB zapewnia obsługę klucza zewnętrznego. Wspomniany klucz zewnętrzny w tabeli potomnej jest wskazany przez FOREIGN KEY, opcjonalny identyfikator klucza zewnętrznego, listę kolumn tworzących klucz zewnętrzny oraz definicję REFERENCES. Identyfikator, o ile zostanie podany, będzie ignorowany, chyba że InnoDB automatycznie tworzy indeks dla klucza zewnętrznego. W takim przypadku *fk_name* staje się nazwą indeksu. Definicja REFERENCES zawiera nazwy tabeli nadrzędnej i kolumn, do których odwołuje się klucz zewnętrzny, a także wskazuje zachowanie po usunięciu rekordu w tabeli nadrzędnej. Domyślna akcja polega na uniknięciu operacji usunięcia lub uaktualnienia danych w tabelach nadrzędnych i potomnych, jeśli może to doprowadzić do uszkodzenia integralności odwołań. Akcje RESTRICT i NO ACTION mają taki sam efekt jak brak wskazania akcji. Klauzule ON DELETE i ON UPDATE można podać w celu wskazania dokładnych akcji. Akcje implementowane przez InnoDB to CASCADE (usunięcie lub uaktualnienie odpowiednich rekordów tabeli potomnej) i SET NULL (przypisanie wartości NULL kolumnom klucza zewnętrznego w odpowiadających rekordach tabeli potomnej). Akcja SET DEFAULT nie jest zaimplementowana i InnoDB powoduje wygenerowanie błędu.

Klauzule MATCH w definicjach REFERENCE są przetwarzane, ale nie używane. (Należy unikać klauzul MATCH, ponieważ powodują ignorowanie ON DELETE i ON UPDATE). Jeżeli podasz definicję klucza zewnętrznego dla silnika bazy danych innego niż InnoDB, cała definicja zostanie zignorowana.

Więcej informacji na temat kluczy zewnętrznych znajdziesz w podrozdziale 2.13, zatytułowanym „Klucze zewnętrzne i integralność odwołań”.

Opcje partycjonowania. Serwer MySQL obsługuje partycjonowanie tabel, czyli funkcję pozwalającą na zdefiniowanie podziału tabeli na różne sekcje. Poniżej przedstawiono krótkie podsumowanie składni definiowania partycji tabeli. W podpunkcie 2.6.2.5, zatytułowanym „Używanie tabel partycjonowanych”, znajdziesz dokładniejszą analizę i przykłady; po kolejnej informacji na ten temat możesz sięgnąć do podręcznika użytkownika MySQL.

Opis partycjonowania rozpoczyna się od `PARTITION BY` i funkcji partycjonującej obliczającej wartość dla każdego rekordu tabeli lub listy nazw kolumn. Wartość funkcji lub nazwy kolumn dla rekordu określają partycję, w której będzie przechowywany rekord. Opis może opcjonalnie zawierać wymienione poniżej komponenty:

- Klauzulę `PARTITIONS n`, wskazującą liczbę tworzonych partycji tabeli. Wartość *n* powinna być dodatnią liczbą całkowitą. Jeżeli użyta zostanie omawiana klauzula i dowolna klauzula *partition_definition*, wtedy musi być *n* tego typu definicji. Maksymalna liczba partycji wynosi 1024, włączając także podpartycje.
- Opis wskazujący sposób podziału partycji na podpartycje.
- Listę klauzul *partition_definition* dla partycji. Każda z wymienionych klauzul opisuje cechy charakterystyczne pojedynczej partycji. Zawiera jej nazwę, klauzule `VALUES`, mapujące wartości funkcji partycjonującej na partycje, a także inne opcje partycji i listę definicji podpartycji. Wszystkie klauzule *subpartition_definition* są podobne, ale opisują podpartycje i nie mogą zawierać klauzuli `VALUES` lub definicji podpartycji.

Poniższa lista pokazuje różne sposoby przypisywania rekordów tabeli podpartycjom. W przedstawionych przykładach *expr* jest wyrażeniem w postaci liczby całkowitej odwołującym się do jednej lub więcej kolumn w tabeli. Z kolei *col_list* to rozdzielona przecinkami lista zawierająca od jednej do szesnastu nazw kolumn. Nazwy kolumn mogą odwoływać się tylko do tworzonej tabeli, a nie do innych tabel.

- `RANGE (expr)` — partycjonowanie powiązuje każdą partycję z podzbiorem zakresu możliwych wartości wyrażenia *expr*. Ta funkcja musi być używana w połączeniu z definicjami partycji zawierającymi klauzule `VALUES LESS THAN`, wskazującymi wyrażony za pomocą liczby całkowitej górny zakres wartości mapowanych na daną partycję. (Wartość `NULL` nie może być ustalona jako zakres, wartości `NULL` są mapowane na pierwszą partycję). Wartości `VALUES` dla kolejnych partycji powinny mieć coraz większe wartości zakresu. Ostatnia partycja może używać `MAXVALUE`, która ma zastosowanie dla wszystkich wartości zbyt małych dla poprzednich partycji.

```
CREATE TABLE t (income BIGINT, ...)
PARTITION BY RANGE (income)
(
    PARTITION p0 VALUES LESS THAN (10000),
    PARTITION p1 VALUES LESS THAN (30000),
    PARTITION p2 VALUES LESS THAN (75000),
    PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

- `RANGE COLUMNS (col_list)` — partycjonowanie podobne do `RANGE(expr)`, ale zamiast pojedynczego wyrażenia do partycjonowania używana jest lista nazw kolumn. Klauzula `VALUES LESS THAN` dla każdej partycji wymienia dosłowne wartości inne niż `NULL` dla każdej kolumny, definiując górny zakres wartości dozwolonych w danej partycji. Na przykład, jeśli specyfikacja partycji rozpoczyna się od

PARTITION BY RANGE COLUMNS (*col1*, *col2*, *col3*), każda klauzula VALUES jest w postaci VALUES LESS THAN (*val1*, *val2*, *val3*), wskazując wartość maksymalną dla poszczególnych kolumn. Każda kolumna na liście (*col_list*) musi być typu liczby całkowitej, ciągu tekstowego, wartości daty i godzinny, bądź też inna niż typu BIT, BLOB, TEXT, ENUM lub SET. Każda dosłowna wartość w klauzuli VALUES LESS THAN musi mieć taki sam typ danych jak odpowiadająca jej kolumna na liście *col_list*.

- LIST (*expr*) — partycjonowanie powiązuje każdą partycję z listą wartości. Musi być używane w połączeniu z definicjami partycji zawierającymi klauzule VALUES IN, wymieniające listę wartości w postaci liczb całkowitych mapujących na partycje. Użycie wartości NULL jest dozwolone, natomiast MAXVALUE już nie. Jeżeli wartością wyrażenia *expr* będzie NULL, wtedy NULL zostanie dołączone jako jedna z wartości listy VALUES.

```
CREATE TABLE t (id INT NULL, ...)
PARTITION BY LIST(id)
(
    PARTITION p0 VALUES IN (1, 2, 3),
    PARTITION p1 VALUES IN (4, 5, 6, NULL)
);
```

- LIST COLUMNS (*col_list*) — partycjonowanie analogiczne dla RANGE COLUMNS(*col_list*), ale dotyczące list. Podobne warunki dotyczą dozwolonego typu kolumn. Zamiast klauzuli VALUES LESS THAN używana jest VALUES IN, ale podobne warunki mają zastosowanie względem typów podanych wartości. (W przeciwieństwie do partycjonowania LIST, wymagającego partycjonowania przez wartości liczb całkowitych, klauzule VALUES IN dla LIST COLUMNS mogą zawierać wartości inne niż liczby całkowite). Każda klauzula VALUES IN dostarcza jednej lub więcej list wartości ujętych w nawiasy. Wspomniane wartości na każdej liście muszą zawierać tę samą liczbę kolumn, ale liczba list w każdej klauzuli może być różna. Na przykład, jeśli tabela zawiera dwie kolumny INT o nazwach *i1* i *i2*, których chcesz użyć do partycjonowania, wtedy partycjonowanie LIST COLUMNS można przeprowadzić następująco:

```
PARTITION BY LIST COLUMNS (i1, i2)
(
    PARTITION p1 VALUES IN ((NULL, NULL)),
    PARTITION p2 VALUES IN ((0, 0), (0, 1), (0, 2)),
    PARTITION p3 VALUES IN ((1, 0), (1, 1))
);
```

- HASH (*expr*) — partycjonowanie powiązuje rekordy z partycjami na podstawie wartości wyrażenia *expr* obliczonych dla zawartości rekordu. Zazwyczaj partycjonowanie HASH() jest używane wraz z klauzulą PARTITIONS *n*, wskazującą liczbę tworzonych partycji. Przypisanie rekordu odbywa się na podstawie reszty z dzielenia *expr* przez *n*.

```
CREATE TABLE t (d DATE, ...)
PARTITION BY HASH(TO_DAYS(d))
PARTITIONS 5;
```

Wyrażenie partycjonowania `HASH()` może być poprzedzone słowem kluczowym `LINEAR`, które zmienia algorytm obliczania wartości hash. Jedną z zalet użycia `LINEAR` jest fakt, że pewne operacje zarządzania partycją stają się znacznie efektywniejsze, na przykład dodawanie lub usuwanie partycji za pomocą zapytania `ALTER TABLE`. Jednak istnieje również prawdopodobieństwo, że rekordy będą mniej równomiernie rozłożone w partycjach, jeśli słowo kluczowe `LINEAR` nie zostanie użyte.

- `KEY (col_list)` — partycjonowanie podobne do `HASH()`, ale wymieniasz kolumny tabeli, na podstawie których będzie obliczana wartość hash, a serwer dostarcza funkcję obliczającą tę wartość. `KEY()` można poprzedzić słowem kluczowym `LINEAR`.

Jeżeli dołączysz listę definicji partycji dla partycjonowania `HASH()` lub `KEY()`, wspomniane definicje nie powinny zawierać klauzul `VALUES`. Klauzule `VALUES` są używane jedynie z `RANGE()` i `LIST()`.

Wyrażenie *expr* musi być deterministyczne, czyli zawsze generować taki sam wynik dla konkretnych danych wejściowych. Na przykład, wyrażenie *expr* może używać funkcji `ABS()`, ale już nie `RAND()`. Zapytanie `CREATE TABLE` zwróci błąd, jeśli użyjesz niedozwolonej funkcji.

Dla partycjonowania `RANGE()` lub `LIST()` wyrażenie *expr* musi generować liczbę całkowitą lub wartość `NULL`. Z kolei dla partycjonowania `HASH()` wyrażenie *expr* musi generować wartość inną niż `NULL`, nieujemną liczbę całkowitą. Dlatego też, jeśli wyrażenie odwołuje się do kolumny typu innego niż liczba całkowita, wtedy jej wartość musi być skonwertowana na postać liczby całkowitej. Na przykład, jeśli *d* jest kolumną typu `DATE`, wówczas można użyć wywołania `TO_DAYS(d)` w celu konwersji daty na liczbę dni i `HASH(TO_DAYS(d))` stanie się prawidłową funkcją hash.

Dla `KEY()` argumentami są nazwy kolumn, ale wspomniane kolumny nie muszą być typu liczby całkowitej.

Każda wartość *partition_option* określa dodatkowe cechy charakterystyczne partycji wybrane z poniższej listy. (W opisach stosowane jest pojęcie „partycja”, ponieważ te opcje mogą być stosowane również w definicjach podpartycji).

- `COMMENT [=] 'str'`

Opisowy komentarz dla partycji.

- `DATA DIRECTORY [=] 'dir_name', INDEX DIRECTORY [=] 'dir_name'`

Te opcje są podobne do wcześniej omówionych opcji tabeli o takich samych nazwach. Wskazują miejsce przechowywania danych i indeksów partycji. Domyślne położenie to katalog bazy danych dla bazy danych zawierającej tabelę partycji.

- `MAX_ROWS [=] n, MIN_ROWS [=] n`

Te opcje są wskazówkami dla silnika bazy danych, wskazują maksymalną i minimalną liczbę rekordów, które mają być przechowywane w partycji. Wartość *n* powinna być dodatnią liczbą całkowitą.

■ [STORAGE] ENGINE [=] *engine_name*

Silnik bazy danych używany przez daną partycję. Nie ma możliwości stosowania różnych silników bazy danych w poszczególnych partycjach, więc jeśli zdecydujesz się na użycie omawianej klauzuli, tę samą nazwę silnika musisz podać dla wszystkich partycji.

Przedstawione poniżej zapytania pokazują niektóre sposoby wykonywania zapytań
CREATE TABLE:

- Utworzenie tabeli wraz z trzema kolumnami. Kolumna `id` jest typu `PRIMARY KEY`, a kolumny `last_name` i `first_name` są używane w indeksie obejmującym wiele kolumn:

```
CREATE TABLE customer
(
    id          SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    last_name   CHAR(30) NOT NULL,
    first_name  CHAR(20) NOT NULL,
    PRIMARY KEY (id),
    INDEX (last_name, first_name)
);
```

- Utworzenie tabeli tymczasowej typu `MEMORY` w celu osiągnięcia większej szybkości działania:

```
CREATE TEMPORARY TABLE tmp_table
(id MEDIUMINT NOT NULL UNIQUE, name CHAR(40))
ENGINE=MEMORY;
```

- Utworzenie tabeli jako pustej kopii innej tabeli:

```
CREATE TABLE prez_copy LIKE president;
```

- Utworzenie tabeli przy użyciu zawartości innej tabeli:

```
CREATE TABLE prez_copy SELECT * FROM president;
```

- Utworzenie tabeli przy użyciu jedynie częściowej zawartości innej tabeli:

```
CREATE TABLE prez_alive SELECT last_name, first_name, birth
FROM president WHERE death IS NULL;
```

Jeżeli definicje kolumn są podawane dla tabeli tworzonej i wypełnianej zawartością przez zapytanie `SELECT` umieszczone na końcu, wtedy wspomniane definicje są stosowane po wstawieniu zawartości tabeli. Na przykład, można zdefiniować, że wskazana kolumna powinna być indeksowana jako `PRIMARY KEY`:

```
CREATE TABLE new_tb1 (PRIMARY KEY (a)) SELECT a, b, c FROM old_tb1;
```

Istnieje możliwość podawania definicji dla kolumn w nowej tabeli nadpisujących definicje używane domyślnie i opierając się na cechach charakterystycznych zbioru wynikowego:

```
CREATE TABLE new_tb1
(a INT UNSIGNED NOT NULL AUTO_INCREMENT, b DATE, PRIMARY KEY (a))
SELECT a, b, c FROM old_tb1;
```

CREATE TRIGGER

```
CREATE
  [DEFINER = definer_name]
  TRIGGER trigger_name trigger_time trigger_event
  ON tbl_name
  FOR EACH ROW trigger_stmt
```

Zapytanie powiązuje wyzwalacz z tabelą. Na przykład, po wystąpieniu określonego zdarzenia w tabeli wyzwalacz aktywuje i wykonuje zdefiniowane zapytanie. Domyślnie przyjmuje się założenie, że wskazana tabela (*tbl_name*) znajduje się w domyślnej bazie danych. W celu wskazania tabeli w innej bazie danych należy jej nazwę podać w formacie *nazwa_bazy_danych.nazwa_tabeli*. Zapytanie CREATE TRIGGER wymaga uprawnienia TRIGGER do tabeli, z którą będzie powiązany wyzwalacz.

Kiedy następuje aktywacja wyzwalacza, klauzula DEFINER określa kontekst bezpieczeństwa (konto użytkownika sprawdzanego podczas weryfikacji uprawnień dostępu), jak to omówiono w podrozdziale 4.3, zatytułowanym „Zapewnienie bezpieczeństwa widokom i programom składowanym”. Domyślnie używane jest konto użytkownika, który wykonuje zapytanie CREATE TRIGGER. Wspomniane konto użytkownika musi mieć uprawnienie TRIGGER do tabeli, uprawnienie SELECT do tabeli *tbl_name*, o ile definicja wyzwalacza odwołuje się do którejkolwiek z jej kolumn za pomocą słów kluczowych NEW lub OLD, oraz uprawnienie UPDATE do tabeli *tbl_name*, jeśli definicja wyzwalacza modyfikuje którąkolwiek z jej kolumn za pomocą zapytania SET NEW.*nazwa_kolumny*. Ponadto, konto użytkownika musi mieć standardowe uprawnienia wymagane do wykonania zapytań podanych w definicji wyzwalacza.

Wartość *trigger_time* to BEFORE lub AFTER i wskazuje, że wykonanie zapytania wyzwalacza powinno nastąpić przed przetworzeniem lub po przetworzeniu rekordu przez zapytanie, które spowodowało aktywację wyzwalacza.

Wartością *trigger_event* powinno być INSERT, UPDATE lub DELETE w celu wskazania rodzaju zapytania powodującego aktywację wyzwalacza.

Z kolei *trigger_stmt* to zapytanie SQL przedstawiające część główną wyzwalacza. To powinno być pojedyncze zapytanie SQL. Aby móc użyć wielu zapytań, należy je umieścić w bloku BEGIN i END, tworząc w ten sposób zapytanie złożone; patrz podrozdział E.2, zatytułowany „Składnia zapytań SQL (zapytania złożone)”.

Składnia OLD.*nazwa_kolumny* może być używana w celu odwołania się do kolumn w starym rekordzie przeznaczonym do usunięcia lub uaktualnienia w wyzwalaczu DELETE lub UPDATE. Podobnie, składnia NEW.*nazwa_kolumny* służy do odwołania się do kolumn w nowym rekordzie wstawianym lub uaktualnianym przez wyzwalacz INSERT lub UPDATE. Wielkość liter w słowach kluczowych OLD i NEW nie ma znaczenia.

W wyzwalaczu BEFORE można zmienić wartości w nowym rekordzie, używając do tego zapytania SET:

```
SET NEW.col_name = value
```

W trakcie tworzenia wyzwalacza wartość bieżąca zmiennej systemowej sql_mode jest zapisywana w celu jej użycia podczas wykonywania wyzwalacza.

Wyzwalacze nie pobierają parametrów i podobnie jak funkcje składowane nie mogą wykonywać zapytań, które generują zbiór wynikowy.

CREATE USER

```
CREATE USER account [auth_info]
[, account [auth_info]] ...
auth_info:
    IDENTIFIED BY [PASSWORD] 'password'
    | IDENTIFIED WITH auth_plugin [AS 'auth_string']
```

Zapytanie tworzy jedno lub więcej kont użytkowników MySQL. Omawiane zapytanie wymaga globalnego uprawnienia CREATE USER lub uprawnienia INSERT dla bazy danych mysql.

Dla każdego tworzonego konta użytkownika następuje utworzenie rekordu w tabeli mysql.user pozbawionego uprawnień. Jeśli tworzone konto użytkownika już istnieje, wtedy zostanie wygenerowany błąd. Nazwę konta użytkownika trzeba podać w formacie 'nazwa_użytkownika'@'nazwa_komputera', jak omówiono w podpunkcie 13.2.1.1, zatytułowanym „Określanie nazw kont”.

Klauzula auth_info, o ile zostanie podana, wskazuje konto użytkownika uwierzytelniane za pomocą hasła lub w inny sposób, na przykład przez wtyczkę uwierzytelniającą. Więcej informacji na ten temat znajdziesz w podpunkcie 13.2.1.3, zatytułowanym „Określanie sposobu uwierzytelniania użytkownika”. Jeśli nie zdefiniujesz klauzuli IDENTIFIED BY z niepustym ciągiem tekstowym hasła lub z klauzulą IDENTIFIED WITH, wtedy klienci mogą użyć konta użytkownika w celu nawiązania połączenia z serwerem bez przeprowadzania uwierzytelniania. Takie rozwiązanie jest niebezpieczne i należy go unikać.

```
CREATE USER 'myname'@'localhost' IDENTIFIED BY 'mypass';
```

CREATE VIEW

```
CREATE [OR REPLACE]
[ALGORITHM = {MERGE | TEMPTABLE | UNDEFINED}]
[DEFINER = definer_name]
[SQL SECURITY = {DEFINER | INVOKER}]
VIEW view_name [(col_list)] AS select_stmt
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

Zapytanie tworzy widok. Jeżeli widok o podanej nazwie już istnieje, domyślnie zostanie wygenerowany błąd. Natomiast po użyciu klauzuli OR REPLACE nowy widok zastąpi już istniejący.

Jeśli podany będzie parametr col_list, wtedy zawiera listę kolumn zwracanych przez widok. Konieczne jest podanie nazwy każdej kolumny. W przypadku pominięcia parametru col_list nazwy kolumn widoku zostaną określone na podstawie kolumn wybranych przez zapytanie SELECT umieszczone w definicji widoku.

Parametr select_stmt to zapytanie SELECT definiujące widok. Może się odwoływać do tabel oraz innych widoków.

W celu utworzenia widoku konieczne jest posiadanie do niego uprawnienia CREATE VIEW, pewnych uprawnień do każdej kolumny wybranej przez select_stmt oraz uprawnienia

SELECT do każdej kolumny wymienionej w *select_stmt*. Jeżeli używasz OR REPLACE, konieczne jest również posiadanie uprawnień DROP dla widoku.

Kiedy następuje wywołanie widoku, klauzula DEFINER określa kontekst bezpieczeństwa (konto użytkownika sprawdzanego podczas weryfikacji uprawnień dostępu), jak to omówiono w podrozdziale 4.3, zatytułowanym „Zapewnienie bezpieczeństwa widokom i programom składowanym”. Domyślnie używane jest konto użytkownika, który wykonuje zapytanie CREATE VIEW.

Klauzula ALGORITHM określa sposób przetwarzania widoku. W przypadku wartości MERGE wykonanie zapytania odwołującego się do widoku powoduje dołączenie definicji widoku do tego zapytania. Następnie zostaje wykonane zapytanie otrzymane w wyniku wspomnianego połączenia. Z kolei wartość TEMPTABLE oznacza użycie tabel tymczasowych podczas wykonywania widoku. Natomiast wartość UNDEFINED oznacza, że serwer wybiera używany algorytm. Wartością domyślną jest UNDEFINED.

Klauzula WITH CHECK OPTION ma zastosowanie względem widoków możliwych do uaktualnienia (czyli możliwych do użycia wraz z zapytaniami UPDATE lub innymi modyfikującymi tabelę tworzące widok). Zezwala na użycie widoku do wstawiania lub uaktualniania w tabelach tworzących widok tylko tych rekordów, dla których wartością klauzuli WHERE w zapytaniu SELECT jest true. Słowa kluczowe CASCADED i LOCAL mają zastosowanie w przypadku definicji widoku odwołującej się do innych widoków. CASCADED powoduje kaskadowe sprawdzenie widoków, do których następuje odwołanie. Z kolei LOCAL sprawdza jedynie widok bieżący. Jeżeli nie zostanie podana żadna, wówczas wartością domyślną jest CASCADED.

DEALLOCATE PREPARE

```
{DEALLOCATE | DROP} PREPARE stmt_name
```

Zapytanie powoduje dezaktywację zapytania preinterpretowanego *stmt_name*, które wcześniej przygotowano za pomocą zapytania PREPARE. Po dezaktywacji danego zapytania nie można więcej wykonać.

DELETE

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
FROM tbl_name
[PARTITION (partition_name [, partition_name] ...)]
[WHERE where_expr] [ORDER BY ...] [LIMIT n]
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
tbl_list
FROM tbl_refs
[WHERE where_expr]
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
FROM tbl_list
USING tbl_refs
[WHERE where_expr]
tbl_list:
tbl_name[.*]
[PARTITION (partition_name [, partition_name] ...)]
```

```
[, tbl_name[.*]
 [PARTITION (partition_name [, partition_name] ...)]
] ...
```

Zapytanie DELETE powoduje usunięcie rekordów z tabeli i zwrot liczby usuniętych rekordów. Począwszy od MySQL 5.6.2, zapytanie DELETE obsługuje klauzulę PARTITION dla tabel partycjonowanych, co pozwala na wskazanie partycji lub podpartycji, z których mają być usunięte rekordy. W takim przypadku, jeśli rekord przeznaczony do usunięcia nie znajduje się we wskazanej partycji, wtedy pozostanie nienaruszony.

Pierwsza forma zapytania DELETE powoduje usunięcie rekordów z tabeli *tbl_name*. Z kolei druga i trzecia forma pozwala na usuwanie rekordów z wielu tabel lub usuwanie rekordów na podstawie warunków obejmujących wiele tabel. Składnia *tbl_refs* jest taka sama jak w zapytaniu SELECT, za wyjątkiem braku możliwości podania podzapytania jako tabeli.

Rekordy do usunięcia to te, które dopasowują warunki zdefiniowane w klauzuli WHERE:

```
DELETE FROM score WHERE event_id = 14;
DELETE FROM member WHERE expiration < CURDATE();
```

W przypadku pominięcia klauzuli WHERE *zostaną usunięte wszystkie rekordy znajdujące się w tabeli*. (Innym sposobem opróżnienia tabeli, gdy nie trzeba zliczyć rekordów, jest użycie zapytania TRUNCATE TABLE).

Opcja LOW_PRIORITY powoduje, że wykonanie zapytania zostanie opóźnione aż do chwili, gdy inne klienty nie będą odczytywały danych z tabeli. Ta opcja jest efektywna jedynie w przypadku silników bazy danych stosujących nakładanie blokad na poziomie tabeli, na przykład MyISAM lub MEMORY.

W przypadku tabel MyISAM użycie opcji QUICK może spowodować szybsze wykonanie zapytania; silnik MyISAM nie będzie musiał go wykonywać w zwykły sposób, polegający na „łączeniu liści drzewa” indeksu.

Użycie modyfikatora IGNORE powoduje, że błędy występujące podczas usuwania rekordów będą ignorowane. Tego rodzaju błędy spowodują wygenerowanie ostrzeżeń.

Jeżeli podana będzie klauzula LIMIT, wtedy wartość *n* określa maksymalną liczbę rekordów do usunięcia.

Klauzula ORDER BY powoduje usuwanie rekordów w posortowanym zbiorze wynikowym. W połączeniu z klauzulą LIMIT daje to większą kontrolę nad usuwanymi rekordami. Klauzula ORDER BY ma taką samą składnię jak w przypadku zapytania SELECT.

Druga i trzecia forma zapytania DELETE pozwalają na jednoczesne usuwanie rekordów z wielu tabel. Umożliwiają również identyfikację rekordów do usunięcia na podstawie złączenia między tabelami. Nazwy na liście tabel, z których rekordy będą usuwane, mogą być podane w postaci *tbl_name* lub *tbl_name.**; druga z wymienionych postaci jest obsługiwana w celu zapewnienia zgodności z ODBC.

Klauzula *tbl_refs* wskazuje tabele do złączenia w celu określenia rekordów do usunięcia. Ta klauzula może zadeklarować aliasy dla tabel, przeznaczone do użycia w zapytaniu. Inne części zapytania mogą, ale nie muszą odwoływać się do zadeklarowanych aliasów.

W celu usunięcia rekordów w tabeli t1, których wartość id zostanie dopasowana do wartości w tabeli t2, należy użyć pierwszej składni zapytania obejmującego wiele tabel:

```
DELETE t1 FROM t1 INNER JOIN t2 WHERE t1.id = t2.id;
```

Ewentualnie można również użyć drugiej składni:

```
DELETE FROM t1 USING t1 INNER JOIN t2 WHERE t1.id = t2.id;
```

Zapytania DELETE obejmujące wiele tabel nie zezwalają na używanie klauzul ORDER BY lub LIMIT. Ponadto, klauzula WHERE nie może zawierać podzapytania wybierającego rekordy z tabeli, z której rekordy są usuwane.

DESCRIBE

```
{DESCRIBE | DESC} tbl_name [col_name | 'pattern']
{DESCRIBE | DESC} stmt
```

Zapytanie DESCRIBE wykonane względem nazwy tabeli lub widoku powoduje wygenerowanie takich samych danych wyjściowych jak SHOW COLUMNS. Więcej informacji znajdziesz w podpunkcie dotyczącym zapytania SHOW. Za pomocą tej składni wymienione na końcu nazwy kolumn ograniczają dane wyjściowe do informacji dotyczących jedynie wskazanych kolumn. Umieszczony na końcu ciąg tekstowy jest interpretowany jako wzorzec i podobnie jak operator LIKE ogranicza dane wyjściowe do kolumn posiadających nazwy dopasowane do wzorca.

- Wyświetlenie danych wyjściowych dla kolumny last_name tabeli president:

```
DESCRIBE president last_name;
```

- Wyświetlenie danych wyjściowych dla kolumn last_name i first_name tabeli president:

```
DESCRIBE president '%name';
```

Zapytanie DESCRIBE jest synonimem zapytania EXPLAIN. Przed wersją MySQL 5.6.3 zapytanie musiało być typu SELECT. Począwszy od MySQL 5.6.3, zapytanie może być typu SELECT, DELETE, INSERT, REPLACE lub UPDATE. Więcej informacji na ten temat znajdziesz w podpunkcie dotyczącym zapytania EXPLAIN.

DO

```
DO expr [, expr] ...
```

Zapytanie powoduje obliczenie wartości wyrażenia bez zwracania jakichkolwiek wyników. W ten sposób zapytanie DO jest znacznie wygodniejsze niż SELECT w przypadku obliczania wartości wyrażenia, ponieważ nie trzeba się zajmować zbiorem wynikowym. Na przykład, zapytanie DO można wykorzystać do ustawienia zmiennych lub wywołania funkcji, w przypadku których interesuje Cię ich efekt uboczny, a nie ich wartości zwrotne.

```
DO @sidea := 3, @sideb := 4, @sidec := SQRT(@sidea*@sidea+@sideb*@sideb);
DO RELEASE_LOCK('mylock');
```


DROP DATABASE

`DROP DATABASE [IF EXISTS] db_name`

Zapytanie powoduje usunięcie wskazanej bazy danych i jej zawartości. Konieczne jest posiadanie uprawnienia DROP do usuwanej bazy danych.

Klauzula `IF EXISTS` może być użyta w celu wyłączenia wyświetlania błędu, który standardowo jest wyświetlany, gdy nie istnieje baza danych przeznaczona do usunięcia. W takim przypadku zostanie wygenerowane ostrzeżenie.

Baza danych jest przedstawiana przez katalog znajdujący się w katalogu danych MySQL. Serwer usuwa jedynie pliki i katalogi, które zostały przez niego utworzone (na przykład pliki *.frm*). Nie powoduje usunięcia pozostałych plików i katalogów. Jeżeli w katalogu bazy danych umieściłeś pliki nieprzedstawiające tabel, tego rodzaju pliki nie zostaną usunięte przez zapytanie `DROP DATABASE`. Wynikiem jest nieudane usunięcie katalogu bazy danych, a tym samym niepowodzenie wykonania zapytania `DROP DATABASE`. W takim przypadku baza danych nadal będzie wyświetlana na liście danych wyjściowych wygenerowanych przez `SHOW DATABASES`. Rozwiązanie problemu polega na ręcznym usunięciu nadmiarowych (niestandardowych) plików i podkatalogów, a następnie ponownym wykonaniu zapytania `DROP DATABASE`.

Jeżeli wykonanie zapytania `DROP DATABASE` zakończy się powodzeniem, dane wyjściowe zapytania będą zawierały informacje o liczbie usuniętych tabel i widoków. (W rzeczywistości to jest liczba usuniętych plików *.frm* oznaczających dokładnie wymienione obiekty).

DROP EVENT

`DROP EVENT [IF EXISTS] event_name`

Zapytanie usuwa wskazane zdarzenie. Klauzula `IF EXISTS` może być użyta w celu wyłączenia wyświetlania błędu, który standardowo jest wyświetlany, gdy nie istnieje zdarzenie przeznaczone do usunięcia. W takim przypadku zostanie wygenerowane ostrzeżenie. Musisz mieć uprawnienie `EVENT` do bazy danych, do której należy zdarzenie.

DROP FUNCTION, DROP PROCEDURE

`DROP {FUNCTION | PROCEDURE} [IF EXISTS] routine_name`

Zapytanie powoduje usunięcie wskazanej funkcji lub procedury składowanej. Oba zapytania wymagają uprawnień `ALTER ROUTINE` do danej procedury.

Klauzula `IF EXISTS` może być użyta w celu wyłączenia wyświetlania błędu, który standardowo jest wyświetlany, gdy nie istnieje procedura przeznaczona do usunięcia. W takim przypadku zostanie wygenerowane ostrzeżenie.

DROP INDEX

```
DROP INDEX index_name ON tbl_name
```

Zapytanie usuwa indeks o wskazanej nazwie (*index_name*) z wymienionej tabeli (*tbl_name*). Wewnętrznie serwer MySQL obsługuje to zapytanie jako ALTER TABLE DROP INDEX. Informacje szczegółowe znajdziesz w podpunkcie dotyczącym zapytania ALTER TABLE. Aby użyć DROP INDEX w celu usunięcia PRIMARY KEY, nazwą indeksu musi być PRIMARY i trzeba ją cytować jak identyfikator:

```
DROP INDEX `PRIMARY` ON tbl_name;
```

DROP TABLE

```
DROP [TEMPORARY] {TABLE | TABLES} [IF EXISTS]
tbl_name [, tbl_name] ...
[RESTRICT | CASCADE]
```

Zapytanie powoduje usunięcie wskazanej tabeli z bazy danych, do której ona należy. Użycie słowa kluczowego TEMPORARY powoduje usunięcie jedynie tabel tymczasowych.

Klauzula IF EXISTS może być użyta w celu wyłączenia wyświetlania błędu, który standardowo jest wyświetlany, gdy nie istnieje tabela przeznaczona do usunięcia. W takim przypadku zostanie wygenerowane ostrzeżenie.

Słowa kluczowe RESTRICT i CASCADE są przetwarzane, ale ignorowane i nie mają żadnego efektu.

DROP TRIGGER

```
DROP TRIGGER [IF EXISTS] trigger_name
```

Zapytanie powoduje usunięcie wyzwalacza. Domyślnie wyzwalacz jest usuwany z domyślnej bazy danych. W celu usunięcia wyzwalacza w innej bazie danych należy jej nazwę podać w formie *nazwa_bazy_danych.nazwa_wyzwalacza*. Zapytanie DROP TRIGGER wymaga uprawnienia TRIGGER do tabeli, z którą jest powiązany wyzwalacz.

Klauzula IF EXISTS może być użyta w celu wyłączenia wyświetlania błędu, który standardowo jest wyświetlany, gdy nie istnieje wyzwalacz przeznaczony do usunięcia. W takim przypadku zostanie wygenerowane ostrzeżenie.

Jeżeli tabela ma wyzwalacze, usunięcie tabeli powoduje również usunięcie znajdujących się w niej wyzwalaczy.

DROP USER

```
DROP USER account [, account] ...
```

Zapytanie DROP USER powoduje usunięcie w tabelach uprawnień wszystkich rekordów powiązanych ze wskazanym kontem użytkownika. To zapytanie wymaga globalnego uprawnienia CREATE USER lub uprawnienia DELETE do bazy danych mysql.

Zapytanie DROP USER powoduje usunięcie konta użytkownika i wszelkich jego uprawnień, ale nie baz danych lub innych obiektów utworzonych przez usuwane

konto użytkownika. Nazwę konta użytkownika należy podać w formacie `'nazwa_użytkownika'@'nazwa_komputera'`, zgodnie z opisem przedstawionym w podpunkcie 13.2.1.1, zatytułowanym „Określanie nazw kont”. Jeżeli konto użytkownika podane w zapytaniu nie istnieje, wtedy zostanie wygenerowany błąd.

```
DROP USER 'myname'@'localhost';
```

DROP VIEW

```
DROP VIEW [IF EXISTS]
view_name [, view_name] ...
[RESTRICT | CASCADE]
```

Zapytanie usuwa wymieniony widok z bazy danych, do której on należy. Konieczne jest posiadanie uprawnienia DROP do usuwanego widoku.

Klauzula IF EXISTS może być użyta w celu wyłączenia wyświetlania błędu, który standardowo jest wyświetlany, gdy nie istnieje widok przeznaczony do usunięcia. W takim przypadku zostanie wygenerowane ostrzeżenie.

Słowa kluczowe RESTRICT i CASCADE są przetwarzane, ale ignorowane i nie mają żadnego efektu.

EXECUTE

```
EXECUTE stmt_name [USING @var_name [, @var_name] ...]
```

Zapytanie powoduje wykonanie zapytania preinterpretowanego o nazwie `stmt_name`, które wcześniej przygotowano za pomocą PREPARE. Klauzula USING musi być użyta, jeśli zapytanie preinterpretowane zawiera jakiekolwiek miejsca zarezerwowane. Wspomniana klauzula powinna zawierać rozdzieloną przecinkami listę wartości użytkownika dostarczających odpowiednie wartości dla kolejnych miejsc zarezerwowanych w zapytaniu.

EXPLAIN

```
EXPLAIN tbl_name [col_name | 'pattern']
EXPLAIN
[EXTENDED | PARTITIONS | FORMAT = {TRADITIONAL | JSON}]
{SELECT ... | DELETE ... | INSERT ... | REPLACE ... | UPDATE ...}
```

Pierwsza forma tego zapytania jest odpowiednikiem DESCRIBE `tbl_name`. Więcej informacji na temat wymienionego zapytania znajdziesz w podpunkcie dotyczącym zapytania DESCRIBE.

Druga forma zapytania EXPLAIN dostarcza informacje o planie wykonania, jaki optymalizator MySQL wygeneruje podczas przetwarzania zapytania znajdującego się po słowie kluczowym EXPLAIN:

```
EXPLAIN SELECT score.* FROM score INNER JOIN grade_event
ON score.event_id = grade_event.event_id AND grade_event.event_id = 14;
```

Przed wydaniem MySQL 5.6.3 zapytanie po EXPLAIN musiało być typu SELECT. Jednak począwszy od wersji 5.6.3, zapytanie może być typu SELECT, DELETE, INSERT, REPLACE lub UPDATE.

Po słowie kluczowym EXPLAIN można podać opcjonalny wskaźnik typu danych wyjściowych do wygenerowania:

- EXTENDED powoduje, że zapytanie EXPLAIN wygeneruje dodatkowe informacje o planie wykonania. Aby wyświetlić wspomniane informacje, wykonaj SHOW WARNINGS natychmiast po zapytaniu EXPLAIN.
- PARTITIONS wyświetla dodatkową kolumnę danych wyjściowych zawierającą informacje o partycjach.
- FORMAT wskazuje, czy dane wyjściowe powinny zostać wygenerowane w „tradycyjnym” (domyślnym) formacie, czy w formacie JSON. Dane wyjściowe JSON zawierają rozszerzone informacje, a także informacje o partycjach, o ile takie istnieją. Omawiana opcja została wprowadzona w wydaniu MySQL 5.6.5.

W domyślnym formacie danych wyjściowych zapytanie EXPLAIN generuje jeden lub więcej rekordów zawierających następujące kolumny:

- id
Identyfikator zapytania SELECT, którego dotyczy dany rekord danych wyjściowych. Jeżeli zapytanie zawiera podzapytania lub jest unią (UNION), to może być więcej niż tylko jedno zapytanie SELECT.
- select_type
Typ zapytania SELECT, którego dotyczy dany rekord danych wyjściowych. Typy zostały wymienione w tabeli E.1.

Tabela E.1. Typy zapytania SELECT w danych wyjściowych EXPLAIN

Typ	Opis
SIMPLE	Zapytanie SELECT bez części UNION lub podzapytań.
PRIMARY	Zewnętrzne lub pierwsze od lewej zapytanie SELECT.
UNION	Drugie lub późniejsze zapytanie SELECT w unii (UNION).
DEPENDENT UNION	Podobne jak UNION, ale zależne od zewnętrznego zapytania.
UNION RESULT	Wynik unii.
SUBQUERY	Pierwsze zapytanie SELECT w podzapytaniu.
DEPENDENT SUBQUERY	Podobne jak SUBQUERY, ale zależne od zewnętrznego zapytania.
DERIVED	Podzapytanie w klauzuli FROM.
UNCACHEABLE SUBQUERY	Wynik podzapytania niemożliwy do buforowania.
UNCACHEABLE UNION	Drugie lub późniejsze podzapytanie SELECT w unii (UNION), którego nie można buforować.

- table
Tabela, do której odnoszą się rekordy danych wyjściowych.

- **partitions**

Partycje, które będą używane. Ta kolumna jest wyświetlana w przypadku użycia opcji `PARTITIONS`. Dla tabel niepartycjonowanych wartością będzie `NULL`.

- **type**

Typ złączenia, jakie zostanie przeprowadzone przez MySQL. Począwszy od najlepszych do najgorszych, możliwymi typami złączeń są `system`, `const`, `eq_ref`, `ref`, `ref_or_null`, `index_merge`, `unique_subquery`, `index_subquery`, `range`, `index` i `ALL`. Lepsze typy są znacznie bardziej restrykcyjne, co oznacza, że MySQL sprawdza mniejszą liczbę rekordów w tabeli podczas pobierania danych.

- **possible_keys**

Indeksy uznawane przez MySQL jako przydatne podczas wyszukiwania rekordów w tabeli wymienionej w kolumnie `table`. Wartość `NULL` oznacza, że nie znaleziono żadnych indeksów.

- **key**

Indeks, który faktycznie będzie użyty do wyszukania rekordów w tabeli. (W tym miejscu może być wymienionych kilka indeksów, jeśli MySQL używa typu złączenia `index_merge`, ponieważ taki rodzaj optymalizacji wykorzystuje wiele indeksów do przetworzenia zapytania). Wartość `NULL` oznacza, że nie będzie użyty żaden indeks.

- **key_len**

Ilość wykorzystanego indeksu. Jeżeli MySQL używa lewego prefiksu indeksu, wtedy ta wartość będzie mniejsza niż pełna długość rekordu indeksu.

- **ref**

Wartości, do których MySQL będzie porównywać wartości indeksu. Słowo `const` lub `'???'` oznacza porównanie względem stałej, z kolei nazwa kolumny wskazuje na porównanie kolumn.

- **rows**

Oszacowana liczba rekordów w tabeli, które MySQL musi przeanalizować w celu wykonania zapytania. Wartość w tej kolumnie jest oszacowanym iloczynem kombinacji rekordów, które muszą być przeanalizowane we wszystkich tabelach zapytania.

- **filtered**

Oszacowana procentowa ilość rekordów, które będą złączone w poprzednich tabelach. Ta kolumna jest wyświetlana w przypadku użycia opcji `EXTENDED`.

- **Extra**

Inne informacje dotyczące planu wykonania. Ta wartość może być pusta lub zawierać jedną lub więcej wartości takich jak wymienione poniżej:

- ◆ **Using filesort**: wartości indeksu muszą być zapisane w pliku i posortowane, aby rekordy można było pobrać w kolejności posortowanej.
- ◆ **Using index**: MySQL może pobierać informacje z tabeli, używając jedynie danych indeksu, bez konieczności analizowania rekordów danych.

- ◆ `Using temporary`: konieczne jest utworzenie tabeli tymczasowej.
- ◆ `Using where`: do wybrania rekordów są używane informacje znajdujące się w klauzuli `WHERE` zapytania `SELECT`.

Inne, niewymienione tutaj wartości mogą pojawiać się w polu `Extra`. Aktualny zestaw dozwolonych wartości `Extra` znajdziesz w podręczniku użytkownika MySQL.

FLUSH

`FLUSH [NO_WRITE_TO_BINLOG | LOCAL] option [, option] ...`

Zapytanie powoduje opróżnienie wewnętrznych buforów używanych przez serwer. Konieczne jest posiadanie uprawnienia `RELOAD` i prawdopodobnie także innych uprawnień dla pewnych operacji, co zostanie wkrótce omówione.

Jeżeli włączony jest binarny dziennik zdarzeń, zapytania `FLUSH` są także w nim zapisywane, o ile nie zostanie użyta opcja `NO_WRITE_TO_BINLOG` lub `LOCAL`. Wyjątkami, które w żadnym przypadku nie będą rejestrowane w binarnym dzienniku zdarzeń, są zapytania `FLUSH LOGS`, `FLUSH MASTER`, `FLUSH SLAVE` oraz obie formy `FLUSH TABLES WITH READ LOCK`. Wymienione zapytania nie są rejestrowane w celu uniknięcia problemów z replikacją w serwerach podległych.

Każda wartość `option` powinna być jednym z elementów z poniższej listy. W przypadku opcji `TABLES`, `TABLE` jest synonimem. Żadna z opcji `TABLES` nie może być w tym samym zapytaniu `FLUSH` łączona z innymi opcjami.

- `DES_KEY_FILE`

Ponowne wczytanie pliku `DES` używanego przez funkcje `DES_ENCRYPT()` i `DES_DECRYPT()`.

- `HOSTS`

Wyczyszczenie bufora komputera. To może być konieczne, jeśli połączenia z pewnego komputera klienta powodują powstanie więcej niż `max_connect_errors` błędów i serwer rozpoczął blokowanie połączeń z wspomnianego komputera.

- `[log_type] LOGS`

Opróżnienie plików dziennika zdarzeń przez ich zamknięcie i ponowne otwarcie. Jeżeli włączony jest dziennik binarny lub przekazywania, takie zapytanie spowoduje otwarcie kolejnego pliku w ponumerowanej sekwencji plików dziennika. Począwszy od MySQL 5.5.3, słowo kluczowe `LOGS` może być poprzedzone specyfikatorem wskazującym konkretny dziennik do opróżnienia. `BINARY`, `ERROR`, `GENERAL`, `RELAY` i `SLOW` powoduje opróżnienie dziennika odpowiednio binarnego, błędów, ogólnego, przekazywania i wolno wykonywanych zapytań. `ENGINE` nakazuje silnikom bazy danych opróżnienie wszelkich dzienników zdarzeń, które obsługują i mogą opróżnić. Silnik `InnoDB` opróżnia swoje dzienniki zdarzeń i tworzy punkt kontrolny.

■ PRIVILEGES

Ponowne wczytanie tabel uprawnień. Nakazanie serwerowi ponownego wczytania tych tabel jest konieczne po przeprowadzeniu ich bezpośredniej modyfikacji za pomocą zapytań takich jak INSERT lub UPDATE. Jeżeli tabele uprawnień zostaną zmodyfikowane za pomocą zapytań GRANT lub REVOKE, wtedy serwer automatycznie ponownie wczytuje znajdujące się w pamięci kopie tabel uprawnień. Ta opcja wpływa również na nakładane na konta użytkownika ograniczenia w zakresie wykorzystania dostępnych zasobów. Omawiana opcja ma taki sam efekt jak opcja USER_RESOURCES.

■ QUERY CACHE

Opróżnienie bufora zapytania w celu jego defragmentacji bez usuwania zapytań znajdujących się w buforze. (W celu faktycznego usunięcia zawartości bufora należy użyć RESET QUERY CACHE).

■ STATUS

Ponowna inicjalizacja zmiennych stanu serwera.

■ TABLES [*tbl_name* [, *tbl_name*] ...]

W przypadku braku jakichkolwiek nazw tabel powoduje zamknięcie i otwarcie wszystkich tabel w buforze tabel. Po podaniu rozdzielonej przecinkami listy jednej lub więcej nazw tabel ta opcja spowoduje opróżnienie bufora wymienionych tabel zamiast całego bufora tabel. Począwszy od wersji MySQL 5.5.3, ta operacja nie może być używana, gdy aktywna jest jakakolwiek blokada typu READ nałożona przez LOCK TABLES. Użyj odpowiedniej składni WITH READ LOCK w celu nałożenia blokady odczytu i opróżnienia bufora tabel.

Jeżeli bufor zapytań jest w użyciu, FLUSH TABLES spowoduje także jego opróżnienie.

■ TABLES WITH READ LOCK

Opróżnia wszystkie tabele we wszystkich bazach danych, a następnie nakłada na nie globalną blokadę odczytu, która będzie obowiązywała aż do chwili wykonania zapytania UNLOCK TABLES. Wymienione zapytanie pozwala klientom na odczyt tabel, ale uniemożliwia wprowadzanie jakichkolwiek zmian. Takie rozwiązanie jest użyteczne podczas tworzenia kopii zapasowej całego serwera i pozwala zagwarantować, że w trakcie operacji żadna tabela nie ulegnie zmianie. Z punktu widzenia klienta wadą takiego rozwiązania jest wydłużenie okresu, w którym nie można wprowadzać zmian w tabelach.

■ TABLES *tbl_name* [, *tbl_name*] ... WITH READ LOCK

Opróżnienie wymienionych tabel i nałożenie na nie blokady odczytu, która będzie obowiązywała aż do chwili wykonania zapytania UNLOCK TABLES, LOCK TABLES lub START TRANSACTION. Poza uprawnieniem RELOAD konieczne jest również posiadanie uprawnienia LOCK TABLES do tabel wymienionych w zapytaniu. Operacja ma zastosowanie jedynie do tabel bazowych. Ignoruje tabele tymczasowe i generuje błąd w przypadku widoków. Omówiona składnia została wprowadzona w wersji MySQL 5.5.3.

■ USER_RESOURCES

Wyzerowanie liczników dla godzinnych ograniczeń dotyczących wykorzystania zasobów, nakładanych na konta użytkowników (przykładem może być MAX_QUERIES_PER_HOUR). Konto użytkownika, które osiągnęło nałożony mu limit, będzie mogło znów przeprowadzać operacje. Ta opcja nie wpływa na żadne ograniczenie MAX_USER_CONNECTIONS, ponieważ nie jest ono ograniczeniem godzinnym.

GRANT

```
GRANT priv_type [(col_list)] [, priv_type [(col_list)] ] ...
ON [TABLE | FUNCTION | PROCEDURE]
  {*. * | * | db_name. * | db_name.tbl_name
   | tbl_name | db_name.routine_name}
TO account [auth_info] [, account [auth_info] ] ...
[REQUIRE security_options]
[WITH grant_or_resource_options]
GRANT PROXY ON account
TO account [auth_info] [, account [auth_info] ] ...
[WITH GRANT OPTION]
auth_info:
  IDENTIFIED BY [PASSWORD] 'password'
  | IDENTIFIED WITH auth_plugin [AS 'auth_string']
```

Zapytanie GRANT nadaje uprawnienia dostępu dla jednego lub więcej kont użytkowników MySQL. W celu wykonania tego zapytania konieczne jest posiadanie zarówno uprawnienia GRANT OPTION, jak i uprawnień, które mają zostać nadane.

Składnia nadawania uprawnień PROXY jest znacznie bardziej restrykcyjna niż w przypadku innych uprawnień: nazwa uprawnienia musi być podana samodzielnie, nie jest dozwolone użycie klauzuli REQUIRE, a klauzula WITH jest dozwolona jedynie z GRANT OPTION.

Każda wartość *priv_type* wskazuje nadawane uprawnienie wybrane z tabeli E.2. Uprawnienia ALL i PROXY są używane samodzielnie. W przypadku pozostałych uprawnień można podać jedno lub więcej z nich w postaci listy rozdzielonej przecinkami. ALL oznacza połączenie wszystkich uprawnień dostępnych na danym poziomie, poza uprawnieniem GRANT OPTION, które musi być nadane oddzielnie, za pomocą klauzuli WITH GRANT OPTION. Na przykład, na poziomie tabeli ALL powoduje nadanie uprawnień, które mają zastosowanie jedynie do tabel.

Tabela E.2. Uprawnienia, które można nadać za pomocą zapytania GRANT

Nazwa uprawnienia	Operacja umożliwiana przez dane uprawnienie
ALTER	Zmiana tabel i indeksów.
ALTER ROUTINE	Zmiana lub usunięcie funkcji bądź procedury składowanej.
CREATE	Utworzenie baz danych i tabel.
CREATE ROUTINE	Utworzenie funkcji i procedur składowanych.
CREATE TABLESPACE	Utworzenie, usunięcie lub zmiana przestrzeni tabel.

Tabela E.2. Uprawnienia, które można nadać za pomocą zapytania GRANT (ciąg dalszy)

Nazwa uprawnienia	Operacja umożliwiana przez dane uprawnienie
CREATE TABLESPACE	Utworzenie, usunięcie lub zmiana przestrzeni tabel.
CREATE TEMPORARY TABLES	Utworzenie tabeli tymczasowej za pomocą słowa kluczowego TEMPORARY.
CREATE USER	Użycie wysokiego poziomu zapytań do zarządzania kontami użytkowników.
CREATE VIEW	Utworzenie widoków.
DELETE	Usunięcie rekordów z tabeli.
DROP	Usunięcie baz danych, tabel i innych obiektów.
EVENT	Utworzenie, usunięcie lub zmiana zdarzeń dla harmonogramu zdarzeń.
EXECUTE	Wykonanie procedur lub funkcji składowanych.
FILE	Odczyt i zapis plików w komputerze serwera.
GRANT OPTION	Nadanie uprawnień innym kontom użytkowników.
INDEX	Tworzenie i usuwanie indeksów.
INSERT	Wstawianie nowych rekordów do tabel.
LOCK TABLES	Wyraźne nakładanie blokad na tabele za pomocą zapytań LOCK TABLES.
PROCESS	Wyświetlenie informacji o wątkach działających w serwerze.
PROXY	Proxy użytkownika.
REFERENCES	Nie używane (zarezerwowane do użycia w przyszłości).
RELOAD	Ponowne wczytanie tabel uprawnień, opróżnienie dzienników zdarzeń lub buforów.
REPLICATION CLIENT	Sprawdzenie lokalizacji serwerów głównego i podległego.
REPLICATION SLAVE	Działanie w charakterze serwera podległego replikacji.
SELECT	Pobranie rekordów z tabel.
SHOW DATABASES	Wyświetlenie nazw wszystkich baz danych za pomocą zapytania SHOW DATABASES.
SHOW VIEW	Wyświetlenie definicji widoku za pomocą zapytania SHOW CREATE VIEW.
SHUTDOWN	Zamknięcie serwera.
SUPER	Zamknięcie wątków i wykonanie innych operacji administracyjnych.
TRIGGER	Utworzenie lub usunięcie wyzwalacza.
UPDATE	Modyfikacja rekordów tabeli.
ALL [PRIVILEGES]	Wszystkie operacje (poza GRANT i REVOKE).
USAGE	Upewnienie specjalne, oznaczające „brak uprawnień”.

Uprawnienie `PROXY` zostało wprowadzone w MySQL 5.5.7.

Uprawnienie `LOCK TABLES` może być nałożone jedynie na tabele, dla których masz uprawnienie `SELECT`; pozwala na nałożenie dowolnego rodzaju blokady, nie tylko blokady odczytu.

Zawsze można wyświetlać i zamykać własne wątki. Uprawnienia `PROCESS` i `SUPER` pozwalają na odpowiednio wyświetlanie i zamykanie wątków należących do dowolnego konta użytkownika, a nie tylko swoich. Uprawnienia `ALTER ROUTINE` i `CREATE ROUTINE` mają zastosowanie jedynie do procedur składowanych, a nie do funkcji zdefiniowanych przez użytkownika (ang. *User Defined Functions*, UDF).

Klauzula `ON` wskazuje zakres nadawanych uprawnień, jak przedstawiono w tabeli E.3.

Tabela E.3. Zakres nadawanych uprawnień

Specyfikator uprawnienia	Poziom, na którym zastosowanie ma uprawnienie
<code>ON *.*</code>	Uprawnienia globalne: wszystkie bazy danych, wszystkie obiekty w bazach danych.
<code>ON *</code>	Uprawnienia na poziomie bazy danych dla domyślnej bazy danych. Jeżeli nie ma domyślnej bazy danych, wtedy następuje wygenerowanie błędu.
<code>ON nazwa_bazy_danych.*</code>	Uprawnienia na poziomie bazy danych: wszystkie obiekty we wskazanej bazie danych.
<code>ON nazwa_bazy_danych.nazwa_tabeli</code>	Uprawnienia na poziomie tabeli: wszystkie kolumny we wskazanej tabeli.
<code>ON nazwa_tabeli</code>	Uprawnienia na poziomie tabeli: wszystkie kolumny we wskazanej tabeli znajdującej się w domyślnej bazie danych.
<code>ON nazwa_bazy_danych.nazwa_procedury</code>	Uprawnienia dla wskazanej procedury we wskazanej bazie danych.
<code>ON konto_uzytkownika</code>	Uprawnienia proxy: <i>konto_uzytkownika</i> wskazuje użytkownika otrzymującego uprawnienia.

W celu wyraźnego wskazania typu obiektu, dla którego będą stosowane uprawnienia (jeśli mamy do czynienia z niejasnością), można użyć słowa kluczowego `TABLE`, `FUNCTION` lub `PROCEDURE` (na przykład `ON TABLE mydb.mytbl` lub `ON FUNCTION mydb.myfunc`).

Aby nadać uprawnienia dla tabeli lub kolumny, tabela musi istnieć.

W przypadku użycia `ALL` jako nazwy uprawnień nadawane są jedynie uprawnienia dostępne na poziomie, dla którego są nadawane. Na przykład, uprawnienie `RELOAD` jest dostępne jedynie jako uprawnienie globalne, a więc będzie nadawane przez `GRANT ALL` dla specyfikatora `ON *.*`, ale już nie w przypadku specyfikatora `ON nazwa_bazy_danych.*`. W tym drugim przypadku nadawane są jedynie te uprawnienia, które mają zastosowanie względem baz danych. Słowo kluczowe `ALL` można wykorzystać jedynie podczas nadawania uprawnień globalnych, bazy danych, tabeli lub procedury.

`USAGE` oznacza „brak uprawnień”. Ma zastosowanie jedynie na poziomie globalnym.

GRANT OPTION ma zastosowanie dla wszystkich uprawnień nadawanych na konkretnym poziomie. Na przykład, jeżeli kontu użytkownika nadajesz uprawnienie SELECT i INSERT dla danej bazy danych, nie możesz spowodować, że tylko jedno z nich będzie możliwe do nadania przez to konto użytkownika.

Kiedy nazwa tabeli jest wymieniona w klauzuli ON, uprawnienie może być charakterystyczne dla kolumny przez podanie listy jednej lub więcej rozdzielonych przecinkami nazw kolumn w klauzuli *col_list*. (To ma zastosowanie tylko dla uprawnień INSERT, REFERENCES, SELECT i UPDATE, które są jedynymi możliwymi do nadania na poziomie kolumny).

Klauzula TO wskazuje jedno lub więcej kont użytkowników, którym powinny być nadane wymienione uprawnienia. Nazwę konta użytkownika trzeba podać w formacie '*nazwa_uzytkownika*'@'*nazwa_komputera*', jak omówiono w podpunkcie 13.2.1.1, zatytułowanym „Określanie nazw kont”.

Po każdej nazwie konta użytkownika można podać opcjonalną klauzulę *auth_info*, wskazującą, czy konto użytkownika jest uwierzytelniane za pomocą hasła, czy na przykład przez wtyczkę uwierzytelnienia. Składnia jest taka sama jak w przypadku zapytania CREATE USER; zapoznaj się z opisem wymienionego zapytania. Więcej informacji szczegółowych znajdziesz w podpunkcie 13.2.1.3, zatytułowanym „Określanie sposobu uwierzytelniania użytkownika”. Wszelkie hasła podane dla istniejących kont użytkowników spowodują zastąpienie ich istniejących haseł.

Jeżeli wymienione konto nie istnieje, wtedy zapytanie GRANT je utworzy. W takim przypadku klient może używać konta użytkownika w celu nawiązywania połączenia z serwerem bez uwierzytelniania się, jeśli nie podano informacji o sposobie uwierzytelniania (na przykład w postaci klauzuli IDENTIFIED BY wraz z niepustym hasłem lub klauzuli IDENTIFIED WITH). Aby uniemożliwić zapytaniu GRANT tworzenie konta użytkownika, o ile nie zostaną podane informacje o sposobie jego uwierzytelniania, należy włączyć tryb SQL o nazwie NO_AUTO_CREATE_USER.

Nazwy baz danych, tabel, kolumn i procedur, jeśli nie będą cytowane, to muszą być cytowane za pomocą znaków cytowania identyfikatora. Nazwy użytkowników i komputerów mogą być cytowane za pomocą znaków cytowania identyfikatorów lub ciągów tekstowych, na przykład:

```
GRANT INSERT ('mycol') ON `test`.`t` TO 'myuser'@'localhost';
```

Klauzula REQUIRE, jeżeli zostanie podana, pozwala na zdefiniowanie możliwości użycia połączenia szyfrowanego i wskazuje rodzaj informacji, które muszą być dostarczone przez klienta. Po słowie kluczowym REQUIRE można użyć jeszcze następującego:

- NONE. Bezpieczne połączenia nie są wymagane.
- SSL. Ogólny typ połączenia. Wymaga, aby połączenia za pomocą tego konta wykorzystywały SSL.
- X509. Użytkownik musi dostarczyć prawidłowy certyfikat X509. W takim przypadku klient może dostarczyć dowolny certyfikat X509; nie ma znaczenia jego treść, ważne jest jedynie, aby certyfikat był prawidłowy.

- Jedna lub więcej poniższych opcji jest wymaganych, aby nawiązane połączenie charakteryzowało się pewnymi cechami:
 - ◆ CIPHER '*str*'. Połączenie musi być nawiązane z użyciem szyfrowania wskazanego przez '*str*'.
 - ◆ ISSUER '*str*'. Certyfikat klienta musi mieć '*str*' jako wartość oznaczającą wystawcę certyfikatu.
 - ◆ SUBJECT '*str*'. Certyfikat klienta musi mieć '*str*' jako wartość oznaczającą przedmiot certyfikatu.

Jeżeli użyjesz więcej niż tylko jednej z powyższych opcji, mogą być one opcjonalnie rozdzielone przez AND. Kolejność opcji nie ma znaczenia.

Jeżeli zostanie podana klauzula WITH, wtedy będzie używana do wskazania konta użytkownika, które może posiadać uprawnienia nadawać innym kontom użytkowników i nakładać ograniczenia na zużycie zasobów przez dane konto użytkownika. Dozwolone opcje klauzuli WITH wymieniono poniżej. Istnieje możliwość podania więcej niż tylko jednej opcji, ich kolejność nie ma znaczenia.

- GRANT OPTION: dane konto użytkownika może nadawać innym kontom użytkownika posiadane uprawnienia, łącznie z prawem do nadawania uprawnień.
- MAX_CONNECTIONS_PER_HOUR *n*: dane konto może w ciągu godziny wykonać *n* połączeń z serwerem.
- MAX_QUERIES_PER_HOUR *n*: dane konto może w ciągu godziny wykonać *n* zapytań.
- MAX_UPDATES_PER_HOUR *n*: dane konto może w ciągu godziny wykonać *n* zapytań modyfikujących dane.
- MAX_USER_CONNECTIONS *n*: dane konto może w ciągu godziny wykonać maksymalnie *n* jednoczesnych połączeń z serwerem.

W przypadku opcji MAX_CONNECTIONS_PER_HOUR, MAX_QUERIES_PER_HOUR i MAX_UPDATES_PER_HOUR wartość 0 oznacza „brak ograniczeń”. Z kolei dla MAX_USER_CONNECTIONS wartość 0 oznacza użycie wartości zmiennej systemowej `max_user_connections`.

Przedstawione poniżej zapytania demonstrują pewne sposoby użycia zapytania GRANT. Inne przykłady znajdziesz w punkcie 13.2.2, zatytułowanym „Nadawanie uprawnień”. Z kolei informacje dotyczące włączenia SSL znajdziesz w podrozdziale 13.5, zatytułowanym „Konfiguracja bezpiecznych połączeń za pomocą SSL”. W poniższych przykładach nie użyto klauzuli IDENTIFIED, ponieważ przyjęto założenie, że konto zostało wcześniej utworzone wraz z hasłem za pomocą zapytania CREATE USER.

- Umożliwienie użytkownikowi paul uzyskania dostępu do wszystkich tabel w bazie danych sampdb z poziomu dowolnego komputera. Oba przedstawione poniżej zapytania są odpowiednikami, ponieważ pominięcie nazwy komputera w identyfikatorze konta jest odpowiednikiem podania % jako nazwy komputera:

```
GRANT ALL ON sampdb.* TO 'paul';
GRANT ALL ON sampdb.* TO 'paul'@'%';
```

- Nadanie uprawnień tylko do odczytu dla tabel w bazie danych menagerie. Użytkownik lookonly będzie mógł nawiązać połączenie z dowolnego komputera w domenę xyz.com:

```
GRANT SELECT ON menagerie.* TO 'lookonly'@'%.xyz.com';
```
- Nadanie kontu użytkownika pełnych uprawnień, ale jedynie do tabeli member w bazie danych sampdb. Użytkownik member_mgr może nawiązać połączenie z jednego, konkretnego komputera:

```
GRANT ALL ON sampdb.member TO 'member_mgr'@'boa.example.com';
```
- Nadanie kontu użytkownika uprawnień superużytkownika, łącznie z możliwością nadawania uprawnień innym użytkownikom. Użytkownik musi nawiązać połączenie z komputera lokalnego:

```
GRANT ALL ON *.* TO 'superduper'@'localhost' WITH GRANT OPTION;
```
- Nadanie użytkownikowi anonimowemu pełnych uprawnień do bazy danych menagerie:

```
GRANT ALL ON menagerie.* TO ''@'localhost';
```
- Nadanie kontu użytkownika odpowiedzialnemu za generowanie raportów uprawnień do wykonywania procedur składowanych znajdujących się w bazie danych admin:

```
GRANT EXECUTE ON admin.* TO 'report_generator'@'localhost';
```
- Nadanie kontu użytkownika pełnego dostępu do bazy danych privatedb, przy czym połączenia muszą być nawiązywane za pomocą protokołu SSL wraz z ważnym certyfikatem X509:

```
GRANT ALL ON privatedb.* TO 'paranoid'@'%.mydomain.com' REQUIRE X509;
```
- Nadanie kontu użytkownika ograniczonego dostępu, czyli możliwości wykonania jedynie 100 zapytań w trakcie godziny, z czego co najwyżej 10 może być uaktualnieniami:

```
GRANT ALL ON test.* TO 'caleb'@'localhost'
  WITH MAX_QUERIES_PER_HOUR 100 MAX_UPDATES_PER_HOUR 10;
```
- Nadanie użytkownikowi client uprawnień do działania w charakterze proxy dla użytkownika bart:

```
GRANT PROXY ON 'bart'@'localhost' TO 'clint'@'localhost';
```

HANDLER

```
HANDLER tbl_name OPEN [[AS] alias_name]
HANDLER tbl_name READ
  {FIRST | NEXT}
  [where_clause] [limit_clause]
HANDLER tbl_name READ index_name
  {FIRST | NEXT | PREV | LAST }
  [where_clause] [limit_clause]
HANDLER tbl_name READ index_name
  {< | <= | = | => | >} (expr_list)
```

```
[where_clause] [limit_clause]
HANDLER tbl_name CLOSE
```

Zapytanie HANDLER zapewnia interfejs niskiego poziomu dla silników bazy danych InnoDB i MyISAM, pozwalający na ominięcie optymalizatora i bezpośrednie uzyskanie dostępu do zawartości tabel. W celu uzyskania dostępu do tabeli za pomocą interfejsu HANDLER w pierwszej kolejności trzeba użyć HANDLER ... OPEN i ją otworzyć. Tabela pozostanie dostępna do użycia aż do chwili wykonania zapytania HANDLER ... CLOSE lub do zakończenia danej sesji. Kiedy tabela pozostaje otwarta, zapytanie HANDLER ... READ pozwala na uzyskanie dostępu do jej zawartości.

Interfejs HANDLER nie chroni przed jednoczesnymi operacjami uaktualnienia danych. Nie nakłada blokady na tabele, co oznacza możliwość modyfikacji tabeli utworzonej przez HANDLER. Dlatego też nie ma gwarancji, że wprowadzone modyfikacje będą odzwierciedlone w rekordach odczytywanych z tabeli.

Musisz mieć świadomość, że zapytania mogą wyzerować otwarty HANDLER lub spowodować utratę jego położenia. Dotyczy to zapytań w ramach sesji HANDLER używających dowolnej tabeli lub zapytań FLUSH TABLES w dowolnej sesji.

INSERT

```
INSERT [DELAYED | LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name
[PARTITION (partition_name [, partition_name] ...)]
[(col_list)]
{VALUES | VALUE} (expr [, expr] ...) [, (...)] ...
[ON DUPLICATE KEY UPDATE col_name=expr [, col_name=expr] ...]
INSERT [DELAYED | LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name
[PARTITION (partition_name [, partition_name] ...)]
SET col_name=expr [, col_name=expr] ...
[ON DUPLICATE KEY UPDATE col_name=expr [, col_name=expr] ...]
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name
[PARTITION (partition_name [, partition_name] ...)]
[(col_list)]
{SELECT ... | (SELECT ...)}
[ON DUPLICATE KEY UPDATE col_name=expr [, col_name=expr] ...]
```

Zapytanie powoduje wstawienie rekordu do istniejącej tabeli (*tbl_name*) i zwraca liczbę wstawionych rekordów. Począwszy od wersji MySQL 5.6.2, zapytanie INSERT obsługuje klauzulę PARTITION dla tabel partycjonowanych, pozwalającą na wskazanie partycji lub podpartycji, do której ma być wstawiony rekord. W takim przypadku, jeśli rekord nie będzie wstawiony do wskazanej partycji, zostanie wygenerowany błąd.

Składnia zapytania INSERT ma trzy formy.

Pierwsza forma zapytania INSERT wymaga podania listy VALUES(), wskazującej wszystkie wartości do wstawienia. Jeżeli nie będzie podana *col_list*, wtedy lista VALUES() musi zawierać po jednej wartości dla każdej kolumny tabeli. Natomiast jeśli będzie podana *col_list* składająca się z jednej lub wielu rozdzielonych przecinkami nazw kolumn, wówczas lista VALUES() musi zawierać po jednej wartości dla wymienionych kolumn.

Kolumnom, które nie zostały wymienione na liście kolumn, zostaną przypisane wartości domyślne. Istnieje możliwość podania wielu list wartości, co pozwala na wstawienie wielu rekordów za pomocą pojedynczego zapytania INSERT.

```
INSERT INTO absence (student_id, date) VALUES(14, '2011-11-03'), (34, NOW());
```

Listy *col_list* i *VALUES()* mogą być puste. To pozwala na utworzenie rekordu, w którym wszystkim kolumnom zostaną przypisane wartości domyślne:

```
INSERT INTO t () VALUES();
```

Druga forma zapytania INSERT powoduje wstawienie kolumn wymienionych w klauzuli SET wraz z wartościami podanymi w odpowiadających im wyrażeniach. Kolumnom niewymienionym zostaną przypisane wartości domyślne.

```
INSERT INTO absence SET student_id = 14, date = '2011-11-03';  
INSERT INTO absence SET student_id = 34, date = NOW();
```

Słowo kluczowe DEFAULT może być użyte na liście *VALUES()* lub w klauzuli SET w celu wyraźnego przypisania kolumnie jej wartości domyślnej, której dzięki temu nie trzeba znać. Ogólnie rzecz biorąc, w celu odniesienia się w wyrażeniu do wartości domyślnej kolumny można użyć *DEFAULT(col_name)*. Poniższe zapytanie powoduje ustawienie kolumnie *i* wartości 0, jeśli jej wartość domyślna wynosi NULL. W przeciwnym razie przypisywana jest wartość 1.

```
INSERT INTO t SET i = IF(DEFAULT(i) IS NULL, 1, 0);
```

Trzecia forma zapytania INSERT powoduje wstawienie do tabeli (*tbl_name*) rekordów pobranych przez zapytanie SELECT. Rekordy muszą zawierać tyle samo kolumn, ile zawiera tabela *tbl_name* lub ile wymieniono na liście *col_list*, o ile taka lista została podana. W przypadku użycia listy kolumn wszelkie kolumny niewymienione na liście będą miały przypisane ich wartości domyślne.

```
INSERT INTO score (student_id, score, event_id)  
SELECT student_id, 100 AS score, 15 AS event_id FROM student;
```

Nie ma możliwości użycia podzapytania w celu pobrania rekordów z tej samej tabeli, do której są wstawiane.

Jeżeli w trakcie wykonywania zapytania INSERT włączony jest tryb ścisły SQL, błędem jest pominięcie kolumny nieposiadającej klauzuli DEFAULT w jej definicji bądź podanie jej wartości za pomocą DEFAULT.

Jeżeli wstawienie rekordu spowoduje powielenie wartości klucza unikalnego indeksu, wtedy wykonywanie zapytania INSERT zostanie przerwane, nastąpi wygenerowanie błędu i więcej żaden rekord nie będzie wstawiony. Dodanie słowa kluczowego IGNORE powoduje, że wspomniane rekordy nie zostaną wstawione i nie będzie wygenerowany błąd. W trybie ścisłego SQL słowo kluczowe IGNORE powoduje, że błędy w trakcie konwersji danych lub inne błędy powodujące przerwanie wykonywania zapytania będą traktowane jako ostrzeżenia, które nie mają znaczenia krytycznego. W takim przypadku kolumnom zostaną przypisane najbliższe prawidłowe wartości.

Klauzula `ON DUPLICATE KEY UPDATE` ma zastosowanie względem rekordów, które mogą doprowadzić do powielenia wartości kluczy w unikalnym indeksie. W przypadku wymienionej klauzuli zapytania `INSERT` będą skonwertowane na zapytania `UPDATE` modyfikujące kolumny istniejącego rekordu przy użyciu danych podanych po słowie kluczowym `UPDATE`. Jeżeli uaktualnienie już nastąpiło, liczba rekordów, których dotyczyło zapytanie `INSERT`, będzie wynosiła 2 zamiast 1.

Opcje `DELAYED`, `LOW_PRIORITY` i `HIGH_PRIORITY` wpływają na harmonogram wykonywania zapytań:

- **DELAYED.** Opcja `DELAYED` powoduje, że rekordy zostaną umieszczone w kolejce do ich późniejszego wstawienia, a wykonanie zapytania zostanie zakończone, aby klient mógł kontynuować działanie bez zbędnego oczekiwania. Jednak w takim przypadku funkcja `LAST_INSERT_ID()` nie zwróci wartości `AUTO_INCREMENT` dla dowolnej kolumny `AUTO_INCREMENT` w tabeli. Opcja `DELAYED` działa w przypadku tabel `MyISAM`, `MEMORY` i `ARCHIVE`. Opcja `DELAYED` będzie ignorowana w zapytaniach `INSERT INTO ... SELECT` oraz `INSERT INTO ... ON DUPLICATE KEY UPDATE`. Opcja `DELAYED` jest ignorowana w przypadku stosowania jej w funkcjach składowanych lub wyzwalaczach, takich jak zapytanie `INSERT` odwołujące się do funkcji składowanych, które uzyskują dostęp do tabeli lub wyzwalaczy. Dotyczy to również zapytań `INSERT` wywoływanych wewnątrz funkcji składowanej lub wyzwalacza. Opcja `DELAYED` nie jest obsługiwana w widokach oraz tabelach partycjonowanych.

Opcja `DELAYED` nie jest obsługiwana dla `InnoDB`, czyli domyślnego silnika bazy danych. Ponadto, począwszy od wersji `MySQL 5.6.6`, jest ona uznawana za przestarzałą i zniknie w pewnym wydaniu. Dlatego też najlepszym rozwiązaniem jest jej unikanie.

- **LOW_PRIORITY.** Opcja powoduje, że wykonanie zapytania zostanie odroczone aż do chwili, gdy żaden inny klient nie będzie odczytywał danych z tabeli.
- **HIGH_PRIORITY.** Opcja powoduje, że efekt użycia opcji serwera `--low-priority-updates` zostanie zniesiony dla tego jednego zapytania. (Jeżeli serwer został uruchomiony z wymienioną opcją, wtedy zmniejsza priorytet zapytań `INSERT` oraz innych uaktualniających dane. Działa to w taki sam sposób jak opcja `LOW_PRIORITY` dla pojedynczego zapytania). Opcja `HIGH_PRIORITY` uniemożliwia także jednoczesne wykonywanie zapytań `INSERT` i `SELECT`, co może się zdarzyć, gdy omawiana opcja nie jest stosowana.

Opcje `LOW_PRIORITY` i `HIGH_PRIORITY` mają zastosowanie jedynie w silnikach bazy danych stosujących nakładanie blokad na poziomie tabeli, na przykład `MyISAM` i `MEMORY`.

INSTALL PLUGIN

```
INSTALL PLUGIN plugin_name SONAME 'plugin_lib'
```

Zapytanie powoduje instalację wtyczki z pliku obiektu biblioteki, który musi znajdować się w katalogu wskazywanym przez zmienną systemową `plugin_dir`. Przedstawione zapytanie rejestruje także wtyczkę w tabeli `mysql.plugin`, aby była wczytywana przez serwer w trakcie jego kolejnych uruchomień. Zapytanie `INSTALL PLUGIN` wymaga uprawnienia `INSERT` do tabeli `mysql.plugin`. Więcej informacji na ten temat znajdziesz w podrozdziale 12.4, zatytułowanym „Interfejs wtyczek”.

KILL

```
KILL [CONNECTION | QUERY] thread_id
```

Zapytanie powoduje zamknięcie wątku serwera o podanym identyfikatorze (*thread_id*). Aby zamknąć wątek, konieczne jest posiadanie uprawnienia `SUPER`, chyba że zamykasz własny wątek. Zapytanie `KILL` pozwala na podanie tylko jednego identyfikatora. Polecenie `mysqladmin kill` wykonuje tę samą operację, ale pozwala na podanie wielu identyfikatorów wątków w wierszu poleceń.

Opcja `CONNECTION` ma taki sam efekt jak brak tej opcji: zamknięcie wątku o podanym identyfikatorze. Z kolei opcja `QUERY` powoduje zakończenie wszystkich zapytań wykonywanych przez wątek, ale nie zamyka samego wątku.

LOAD DATA

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'
  [IGNORE | REPLACE]
  INTO TABLE tbl_name
  [PARTITION (partition_name [, partition_name] ...)]
  [CHARACTER SET charset]
  [field_options] [line_options]
  [IGNORE n {LINES | ROWS}]
  [(col_or_user_var_name, ...)]
  [SET col_name = expr [, col_name = expr] ...]
```

Zapytanie `LOAD DATA` odczytuje rekordy danych wejściowych z pliku o podanej nazwie, a następnie wstawia je do podanej tabeli za pomocą pojedynczej operacji. To rozwiązanie jest znacznie szybsze niż zestaw pojedynczych zapytań `INSERT`. Począwszy od MySQL 5.6.2, zapytanie `LOAD DATA` obsługuje także klauzulę `PARTITION` dla tabel partycjonowanych, co pozwala na wskazanie partycji lub podpartycji, do której mają być wstawione rekordy. W takim przypadku, jeśli rekord nie będzie mógł być wstawiony do wskazanej partycji, nastąpi wygenerowanie błędu.

Zapytanie `LOAD DATA` generuje dane wyjściowe w poniższym formacie:

```
Records: n Deleted: n Skipped: n Warnings:
```

Jeżeli licznik ostrzeżeń (`Warnings`) ma wartość niezerową, wykonaj zapytanie `SHOW WARNINGS`, aby przekonać się, w czym tkwi problem.

Opcja `LOW_PRIORITY` powoduje, że wykonanie zapytania zostanie odroczone aż do chwili, gdy żaden inny klient nie będzie odczytywał danych z tabeli. Ma ona zastosowanie jedynie w silnikach bazy danych stosujących nakładanie blokad na poziomie tabeli, na przykład `MyISAM` i `MEMORY`.

Opcja `CONCURRENT` ma zastosowanie jedynie dla tabel `MyISAM`. Jeżeli tabela nie ma przerw w środku, wtedy nowe rekordy zostaną wstawione na końcu tabeli. W takim przypadku inne klienty mogą pobierać dane z tabeli podczas wstawiania na jej końcu nowych rekordów.

Bez użycia słowa kluczowego `LOCAL` plik jest odczytywany bezpośrednio przez serwer w komputerze serwera. To wymaga uprawnienia `FILE`, a sam plik musi znajdować się w katalogu domyślnej bazy danych lub mieć uprawnienia pozwalające na jego odczyt innym użytkownikom. Jeżeli wartość zmiennej systemowej `secure_file_priv` jest niepusta, jej wartość powinna być katalogiem, a plik musi znajdować się w tym katalogu. W przypadku użycia słowa kluczowego `LOCAL` klient odczytuje plik po stronie klienta, czyli w komputerze klienta, a następnie przez sieć przekazuje jego zawartość serwerowi. Uprawnienie `FILE` nie jest więc wymagane, ale plik musi być możliwy do odczytu przez klienta. Słowo kluczowe `LOCAL` można stosować wedle potrzeb. Jeżeli nie zostanie użyte po stronie serwera, wtedy nie będzie go można użyć także po stronie klienta. Z kolei, jeśli zostało podane po stronie serwera, ale już nie klienta, wtedy trzeba je wyraźnie podać po stronie klienta. Na przykład, używając programu `mysql` można wykorzystać flagę `--local-infile` do włączenia `LOCAL`.

W przypadku pominięcia słowa kluczowego `LOCAL` w zapytaniu `LOAD DATA`, serwer lokalizuje plik w następujący sposób:

- Jeżeli `'file_name'` to bezwzględna ścieżka dostępu, serwer szuka pliku, poczynawszy od katalogu głównego.
- Jeżeli `'file_name'` to względna ścieżka dostępu, interpretacja zależy od tego, czy nazwa składa się z pojedynczego komponentu. Gdy tak jest, serwer szuka pliku w katalogu domyślnej bazy danych. Natomiast jeśli nazwa pliku zawiera wiele komponentów, wyszukiwanie pliku serwer rozpoczyna od katalogu danych serwera.

W przypadku podania słowa kluczowego `LOCAL` interpretacja nazwy pliku przedstawia się następująco:

- Jeżeli `'file_name'` to bezwzględna ścieżka dostępu, klient szuka pliku, poczynawszy od katalogu głównego.
- Jeżeli `'file_name'` to względna ścieżka dostępu, klient szuka pliku, poczynawszy od katalogu roboczego.

W systemie Windows ukośniki w nazwach pliku mogą być zapisane jako pojedyncze ukośniki slash (/) lub podwójne ukośniki backslash (\\).

Domyślnie zawartość pliku jest interpretowana za pomocą kodowania znaków wymienionego w zmiennej systemowej `character_set_database`. W celu wyraźnego wskazania kodowania znaków pliku należy użyć klauzuli `CHARACTER SET`. (Jednak

kodowaniem znaków nie może być ucs2, utf16, utf16e lub utf32, co oznacza również brak możliwości wczytania plików, których zawartość stosuje wymienione kodowania znaków).

Rekordy powielające wartości w unikalnym indeksie zostaną zignorowane lub zastąpią istniejące rekordy, w zależności od użycia słowa kluczowego IGNORE lub REPLACE. Jeżeli żadne z wymienionych słów kluczowych nie zostanie podane, pozostałe rekordy zostaną zignorowane i nastąpi wygenerowanie błędu. W przypadku użycia słowa kluczowego LOCAL nie można zakłócić transmisji pliku, więc zachowanie domyślne jest jak IGNORE, o ile nie zostanie podana żadna opcja dotycząca obsługi powielających się wartości.

Klauzule *field_options* i *line_options* wskazują format danych. (Opcje dostępne w tych klauzulach mają zastosowanie także dla odpowiadających im klauzul w zapytaniu SELECT ... INTO OUTFILE). Składnia dwóch wymienionych klauzul przedstawia się następująco:

```
field_options:
[FIELDS
 [TERMINATED BY 'str']
 [[OPTIONALLY] ENCLOSED BY 'char']
 [ESCAPED BY 'char' ] ]
```

```
line_options:
[LINES
 [STARTING BY 'str']
 [TERMINATED BY 'str'] ]
```

Wartości 'str' i 'char' mogą zawierać wymienione w tabeli E.4 sekwencje sterujące wskazujące znaki specjalne. Sekwencje sterujące powinny być podane za pomocą liter o wielkości przedstawionej w tabeli E.3.

Tabela E.4. Sekwencje sterujące, które można stosować w zapytaniach LOAD DATA

Sekwencja	Opis
\0	NUL (bajt o wartości zero).
\b	Backspace.
\n	Znak nowego wiersza (wysuw wiersza).
\r	Znak powrotu do początku wiersza.
\s	Spacja.
\t	Tabulator.
\'	Apostrof.
\"	Cudzysłów.
\\	Backslash.
\Z	Ctrl+Z (znak EOF w Windows).

Istnieje możliwość użycia wartości szesnastkowych w celu wskazania dowolnego znaku. Na przykład, LINES TERMINATED BY 0x02 oznacza, że wiersze będą kończyły się znakiem Ctrl+B (ASCII 2).

Jeżeli podana będzie klauzula `FIELDS`, wtedy musi być podana co najmniej jedna z wymienionych opcji: `TERMINATED BY`, `ENCLOSED BY` lub `ESCAPED BY`. Z kolei w przypadku podania wielu opcji mogą one być wymienione w dowolnej kolejności. Podobnie, jeśli podana będzie klauzula `LINES`, wtedy konieczne jest wskazanie co najmniej jednej opcji z wymienionych: `STARTING BY` lub `TERMINATED BY`. Jeśli chcesz podać obie, możesz je wymienić w dowolnej kolejności. W przypadku użycia obu klauzul, `FIELDS` i `LINES`, jako pierwszą trzeba podać `FIELDS`.

Poniżej wymieniono opcje dostępne dla klauzuli `FIELDS`:

- `TERMINATED BY`. Określa znak lub znaki, które będą ogranicznikami wartości w wierszu.
- `ENCLOSED BY`. Wskazuje znaki cytowania, które będą usunięte z obu stron wartości, o ile zostaną użyte. Odbywa się to niezależnie od ustawienia `OPTIONALLY`. W przypadku danych wyjściowych (`SELECT ... INTO OUTFILE`) znak zdefiniowany przez opcję `ENCLOSED BY` jest używany do ujmowania wartości w wierszach danych wyjściowych. Jeżeli zostanie użyte słowo kluczowe `OPTIONALLY`, wtedy wartości będą cytowane jedynie dla kolumn typu `CHAR` i `VARCHAR`.

W celu umieszczenia egzemplarza znaku `ENCLOSED BY` w wartości danych wejściowych należy podać go podwójnie lub poprzedzić znakiem zdefiniowanym przez `ESCAPED BY`. W przeciwnym razie zostanie zinterpretowany jako koniec wartości. Z kolei w danych wyjściowych wystąpienia znaku `ENCLOSED BY` w wartości są poprzedzane znakiem `ESCAPED BY`.

- `ESCAPED BY`. Określa sposób cytowania znaków specjalnych. W przedstawionych poniżej przykładach przyjęto założenie, że znakiem sterującym jest ukośnik (`\`). Dla danych wejściowych niecytowana sekwencja `\N` będzie interpretowana jako `NULL`. Z kolei sekwencja `\0` (backslash + ASCII 0) jest interpretowana jako bajt o wartości zero. W przypadku innych znaków sterujących znak sterujący zostanie usunięty, a znak znajdujący się za nim będzie użyty dosłownie. Na przykład, sekwencja `\"` jest interpretowana jako cudzysłów, nawet jeśli wartości będą ujęte w cudzysłów.

W danych wyjściowych znak sterujący jest używany do zakodowania `NULL` jako niecytowanej sekwencji `\N`, a bajty o wartości zero jako `\0`. Ponadto, wystąpienia znaków `ESCAPED BY` i `ENCLOSED BY` są poprzedzane znakiem sterującym (o ile to konieczne), ponieważ są pierwszymi znakami wiersza i znakami kończącymi ciągi tekstowe. Jeżeli `ESCAPED BY` jest pustym znakiem (`ESCAPED BY ''`), wtedy nie następuje zmiana znaczenia znaków. (W takim przypadku `NULL` jest zapisywane jako `NULL`, a nie jako `\N`). Aby użyć dosłownego znaku sterowania `\`, należy podać go dwukrotnie (`ESCAPED BY '\\'`).

Opcje dla klauzuli `LINES` są następujące:

- `STARTING BY`. Wartość wskazuje jeden lub więcej znaków, od których rozpoczyna się wiersz. Ta wartość oraz wszystko to, co znajduje się przed nią w wierszu, jest uznawane za początek wiersza.

- **TERMINATED BY.** Wartość wskazuje jeden lub więcej znaków oznaczających koniec wiersza.

Jeżeli nie zostanie podana żadna wartość **FIELDS** lub **LINES**, wtedy domyślnie są używane znaki, jakby zostały zdefiniowane następująco:

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
LINES STARTING BY '' TERMINATED BY '\n'
```

Innymi słowy, wartości w wierszu są ograniczane tabulatorami, nie są cytowane, ukośnik \ jest traktowany jako znak sterujący, a wiersz kończy się znakiem nowego wiersza.

Jeżeli wartości **TERMINATED BY** i **ENCLOSED BY** dla klauzuli **FIELDS** będą puste, wtedy używany jest format rekordu o stałej szerokości bez ograniczników między kolumnami. Wartości kolumn są odczytywane (lub w przypadku danych wyjściowych zapisywane) jako wystarczająco długie, aby pomieścić wszystkie wartości w kolumnie. Na przykład, kolumny **VARCHAR(15)** i **MEDIUMINT** są w przypadku danych wejściowych odczytywane jako wartości składające się z odpowiednio 15 i 8 znaków. Z kolei w przypadku danych wyjściowych są zapisywane z użyciem 15 i 8 znaków. Wartości **NULL** są zapisywane jako ciągi tekstowe znaków.

Wartości **NULL** w pliku danych wejściowych są wskazywane przez niecytowane sekwencje \N. Jeżeli znak **FIELDS ENCLOSED BY** nie jest pusty, wszystkie wartości danych wejściowych inne niż **NULL** muszą być cytowane za pomocą znaku **ENCLOSED BY**, a niecytowane słowo **NULL** również jest interpretowane jako wartość **NULL**.

Jeżeli użyta zostanie klauzula **IGNORE n LINES**, pierwsze *n* wierszy danych wejściowych zostanie odrzucone (**ROWS** to synonim **LINES**). Na przykład, jeśli plik danych ma rekord nagłówek kolumn, których nie chcesz wstawić do tabeli bazy danych, użyj klauzuli **IGNORE 1 LINES**:

```
LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl IGNORE 1 LINES;
```

Domyślnie przyjmuje się, że wiersze danych wejściowych zawierają po jednej wartości dla każdej kolumny tabeli. Jeżeli lista składa się z jednej lub więcej rozdzielonych przecinkami nazw kolumn, wtedy wiersze danych wejściowych powinny zawierać wartość dla każdej wymienionej kolumny. Kolumnom niewymienionym na liście przypisywane są ich wartości domyślne. Jeżeli wiersz danych wejściowych jest krótszy niż oczekiwana liczba wartości, kolumnom, dla których nie podano wartości, zostaną przypisane ich wartości domyślne.

Jeżeli w trakcie wykonywania zapytania **LOAD DATA** włączony jest tryb ścisły **SQL**, błędem będzie pominięcie wartości dla kolumny nieposiadającej klauzuli **DEFAULT** w jej definicji lub przypisanie wartości **NULL** kolumnie **NOT NULL**.

Lista może zawierać nazwy kolumn lub zmiennych użytkownika oraz klauzulę **SET** w celu przeprowadzenia dodatkowych operacji przetwarzania wartości danych wejściowych przed ich wstawieniem do tabeli. Na przykład, poniższe zapytanie powoduje wstawienie pierwszej kolumny danych wejściowych do **col1**, zignorowanie drugiej kolumny, wstawienie do **col2** sumy trzeciej i czwartej kolumny oraz użycie funkcji **UUID()** do wygenerowania wartości dla **col3**:

```
LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl
  (col1,@skip,@addend1,@addend2)
SET col2 = @addend1 + @addend2, col3 = UUID();
```

Klauzula SET składa się z jednego lub więcej rozdzielonych przecinkami wyrażeń przypisania wartości. Lewą stroną każdego przypisania musi być nazwa kolumny tabeli. Zmienne użytkownika są niedozwolone w formacie danych wejściowych o stałej długości, ponieważ nie ma możliwości ustalenia długości kolumny. Podzapytania skalarne można wykorzystać w celu dostarczenia wartości kolumny, ale nie można pobierać wartości z tej samej tabeli, do której są wstawiane dane.

Jeżeli masz utworzony w Windows plik tekstowy z wartościami rozdzielonymi tabulatorami, wtedy można użyć domyślnego separatora kolumn, ale wiersze prawdopodobnie będą kończyły się parą znaków nowego wiersza i powrotu na początek wiersza. Aby wczytać taki plik, trzeba w zapytaniu wskazać wymienione znaki jako znajdujące się na końcu wiersza:

```
LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl
  LINES TERMINATED BY '\r\n';
```

Jeżeli plik danych utworzony w Windows przez program używający starej konwencji MS-DOS umieszczania znaku *Ctrl+Z* na końcu pliku w celu wskazania EOF (ang. *End Of File*, koniec pliku) wczytasz do bazy danych, skutkiem może być uszkodzony rekord. Plik trzeba utworzyć za pomocą programu, który nie umieszcza wymienionego znaku na końcu, lub usunąć rekord po wczytaniu zawartości pliku do bazy danych.

Pliki w formacie CSV (ang. *Comma Separated Values*, wartości rozdzielone przecinkami) zawierają przecinki między wartościami, a same wartości mogą być ujmowane w cudzysłów. Przyjmując założenie, że wiersze mają na końcu znak nowego wiersza, zapytanie LOAD DATA wczytujące tego rodzaju plik będzie miało następującą postać:

```
LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl
  FIELDS TERMINATED BY ',' ENCLOSED BY ''';
```

Zapis szesnastkowy jest użyteczny podczas podawania dowolnych znaków sterujących. Poniższe zapytanie odczytuje plik, którego wartości są rozdzielone znakami *Ctrl+A* (ASCII 1), a wiersze kończą się znakami *Ctrl+B* (ASCII 2):

```
LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl
  FIELDS TERMINATED BY 0x01 LINES TERMINATED BY 0x02;
```

LOAD INDEX INTO CACHE

```
LOAD INDEX INTO CACHE
tbl_index_spec [, tbl_index_spec] ...
```

```
tbl_index_spec:
tbl_name
  [PARTITION (partition_name [, partition_name] ... | ALL)]
  [[INDEX | KEY] (index_name [, index_name] ...)]
  [IGNORE LEAVES] ...
```

Zapytanie wczytuje indeksy z wymienionej tabeli MyISAM do bufora kluczy przypisanego tabeli. To będzie domyślny bufor kluczy, o ile tabela nie została przypisana do innego za pomocą zapytania `CACHE INDEX`. Wszystkie bloki indeksu są domyślnie wczytywane, natomiast po podaniu klauzuli `IGNORE LEAVES` wczytywane są jedynie te bloki w drzewie indeksu, które nie są „liśćmi drzewa” indeksu.

W przypadku tabel partycjonowanych klauzula `PARTITION` pozwala na wczytanie wskazanych partycji.

Podobnie jak w przypadku zapytania `CACHE INDEX`, składnia `LOAD INDEX INTO CACHE` umożliwia wskazanie poszczególnych indeksów, ale implementacja powoduje wczytanie wszystkich indeksów tabeli.

Konieczne jest posiadanie uprawnień `INDEX` do każdej tabeli wymienionej w zapytaniu.

Zapytanie `LOAD INDEX INTO CACHE` powoduje wygenerowanie danych wyjściowych w formacie omówionym w podpunkcie poświęconym `CHECK TABLE`.

Więcej informacji dotyczących zarządzania buforem kluczy MyISAM znajdziesz w punkcie 12.7.2, zatytułowanym „Dostrajanie silnika bazy danych”.

LOAD XML

```
LOAD XML [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'
[IGNORE | REPLACE]
INTO TABLE tbl_name
[PARTITION (partition_name [, partition_name] ...)]
[CHARACTER SET charset]
[ROWS IDENTIFIED BY '<tag_name>']
[IGNORE n {LINES | ROWS}]
[(col_or_user_var_name, ...)]
[SET col_name = expr [, col_name = expr] ...]
```

To zapytanie jest podobne do `LOAD DATA`, ale różni się pod następującymi względami:

- Plik przeznaczony do wczytania powinien być plikiem XML.
- Klauzula `ROWS IDENTIFIED BY`, o ile zostanie podana, wymienia znacznik (łącznie z nawiasem ostrym) identyfikujący miejsce, w którym rozpoczynają i kończą się rekordy. Domyślnie to `'<row>'`.

Rekordy przeznaczone do wstawienia w tabeli mogą być w pliku XML danych wejściowych przedstawione za pomocą trzech formatów. Plik może używać różnych formatów dla różnych rekordów.

- W ramach elementów `<row>` nazwy kolumn i ich wartości są pobierane z nazw atrybutów i ich wartości.

```
<row col_name1="value1" col_name2="value2" ... />
```

- W ramach elementów `<row>` nazwy kolumn i ich wartości są pobierane z nazw znaczników podoelementów i ich zawartości.

```
<row>
  <col_name1>value1</col_name1>
  <col_name2>value2</col_name2>
  ...
</row>
```

- W ramach elementów <row> nazwy kolumn i ich wartości są pobierane z wartości i zawartości atrybutu podelementu.

```
<row>
  <field name="col_name1">value1</field>
  <field name="col_name2">value2</field>
  ...
</row>
```

LOCK TABLE

```
LOCK {TABLE | TABLES}
    tbl_name [[AS] alias_name] lock_type
[, tbl_name [[AS] alias_name] lock_type] ...
```

Zapytanie powoduje nałożenie blokady na wskazaną tabelę lub widok; być może będzie konieczne poczekanie na nałożenie wszystkich wymaganych blokad. W przypadku nakładania blokady na widok zapytanie powoduje zablokowanie tabel bazowych tworzących dany widok. Omawiane zapytanie wymaga uprawnień LOCK TABLES i SELECT na każdą tabelę lub widok, na które będzie nakładana blokada.

Każda wartość *lock_type* musi być jedną z wymienionych poniżej:

- READ [LOCAL]. Nałożenie blokady odczytu. Pozostałe klienty nie mogą zapisywać danych w tabeli, ale mogą odczytywać z niej dane.

Blokada typu READ LOCAL jest odmianą blokady READ przeznaczoną do obsługi wstawiania danych, gdy inne dane są odczytywane z tabeli. Ma zastosowanie jedynie względem tabel MyISAM, które nie mają żadnych dziur w środku powstałych na skutek usunięcia lub uaktualnienia rekordów. Blokada READ LOCAL pozwala na wyraźne nałożenie blokady na tabelę, ale nadal pozwala innym klientom na jednoczesne wstawianie danych. (Jeżeli w tabeli są dziury, blokada jest traktowana jako zwykła blokada READ).

- [LOW_PRIORITY] WRITE. Nałożenie blokady zapisu. W ten sposób pozostałe klienty nie mają możliwości odczytu i zapisu w tabeli. Modyfikator LOW_PRIORITY jest używany do zmiany zachowania podczas nakładania blokady, ale nie ma żadnego efektu w wersjach wcześniejszych niż MySQL 5.5.3.

Wykonanie zapytania LOCK TABLE powoduje zniesienie wszystkich istniejących aktualnie blokad. Dlatego też w celu nałożenia blokad na wiele tabel trzeba to zrobić za pomocą pojedynczego zapytania LOCK TABLE. Po nałożeniu blokady za pomocą LOCK TABLE nie ma możliwości odwołania się do jakiegokolwiek niezablokowanej tabeli. Po zakończeniu sesji serwer automatycznie zwalnia wszystkie blokady.

Zapytanie LOCK TABLE pozwala na stosowanie aliasu i tym samym można nałożyć blokadę w ramach aliasu, który ma być używany podczas odwoływania się do tabeli w kolejnych zapytaniach. Jeżeli w zapytaniu wielokrotnie odwołujesz się do tabeli, blokadę trzeba nałożyć na każdy egzemplarz tabeli, w razie konieczności blokując aliasy. Żądanie wszystkich blokad musi nastąpić w tym samym zapytaniu.


```
LOCK TABLE student READ, score WRITE, grade_event READ;  
LOCK TABLE member READ;  
LOCK TABLE t AS t1 READ, t AS t2 READ;
```

Jeżeli jest przeprowadzana transakcja, zapytanie `LOCK TABLE` powoduje jej zatwierdzenie. Blokady nałożone na tabelę za pomocą zapytania `LOCK TABLE` są znoszone po rozpoczęciu transakcji za pomocą `START TRANSACTION`.

OPTIMIZE TABLE

```
OPTIMIZE [NO_WRITE_TO_BINLOG | LOCAL]  
{TABLE | TABLES} tbl_name [, tbl_name] ...
```

Wykonywanie zapytań `DELETE`, `REPLACE` i `UPDATE` może doprowadzić do powstania w tabeli dużych ilości niewykorzystanego miejsca. Rozwiązaniem problemu jest wykonanie zapytania `OPTIMIZE TABLE`, które przeprowadza operacje optymalizacyjne. Omawiane zapytanie działa z tabelami InnoDB, MyISAM i ARCHIVE. Zapytanie `OPTIMIZE TABLE` wymaga uprawnień `SELECT` i `INSERT` do każdej używanej tabeli.

Jeżeli włączony jest binarny dziennik zdarzeń, MySQL zapisuje w nim zapytanie `OPTIMIZE TABLE`, o ile nie zostanie podana opcja `NO_WRITE_TO_BINLOG` lub `LOCAL`.

W przypadku tabeli InnoDB zapytanie `OPTIMIZE TABLE` jest mapowane na zapytanie `ALTER TABLE` uaktualniające dane statystyczne indeksu i zwalnijące nieużywaną przestrzeń w indeksie klastrowanym.

W przypadku tabeli MyISAM zapytanie `OPTIMIZE TABLE` wykonuje następujące operacje:

- Defragmentacja tabeli w celu eliminacji zmarnowanego miejsca i zmniejszenia wielkości tabeli.
- Połączenie zawartości rekordów o zmiennej długości, które stały się pofragmentowane. W wyniku połączenia powstają ciągłe fragmenty i cały rekord jest przechowywany w jednym miejscu.
- Sortowanie stron indeksu, o ile zachodzi konieczność.
- Uaktualnienie wewnętrznych danych statystycznych tabeli.

Wykonanie zapytania `OPTIMIZE TABLE` dla tabeli MyISAM jest jak uruchomienie narzędzia `myisamchk` wraz z opcjami `--check-only-changed`, `--quick`, `--sort-index` i `--analyze`. Jednak w przypadku narzędzia `myisamchk` trzeba samodzielnie zadbać o to, aby serwer nie uzyskiwał dostępu do tabeli w trakcie działania `myisamchk`. Gdy wykonywane jest zapytanie `OPTIMIZE TABLE`, serwer nie przeprowadza innych operacji i gwarantuje, że inne klienty nie będą modyfikować tabeli w trakcie jej optymalizacji.

W przypadku tabeli ARCHIVE zapytanie `OPTIMIZE TABLE` przeprowadza analizę tabeli i ponownie ją kompresuje w celu zmniejszenia ilości zajmowanego przez nią miejsca.

Zapytanie `OPTIMIZE TABLE` generuje dane wyjściowe w formacie opisanym w podpunkcie poświęconym zapytaniu `CHECK TABLE`.

PREPARE

```
PREPARE stmt_name FROM {'str' | @var_name}
```

Zapytanie przygotowuje inne zapytanie i przypisuje mu nazwę *stmt_name*. Takie zapytanie może być później wykonane za pomocą EXECUTE lub usunięte za pomocą DEALLOCATE PREPARE. Jeżeli wcześniej istniało zapytanie preinterpretowane o takiej samej nazwie, zostanie ono usunięte i dopiero później nastąpi zdefiniowanie nowego. Wielkość liter w nazwach zapytań nie ma znaczenia.

Nazwa dla zapytania preinterpretowanego powinna być podana jako dosłowny ciąg tekstowy lub jako zmienna użytkownika. Zestaw zapytań, których można używać w PREPARE, znacznie się rozrósł na przestrzeni czasu. Początkowo obejmował zapytania CREATE TABLE, DELETE, DO, INSERT, REPLACE, SELECT, SET, UPDATE oraz większość wariantów SHOW. Następnie dodano możliwość użycia innych zapytań. Listę zapytań dozwolonych do wykorzystania z PREPARE znajdziesz w podręczniku użytkownika MySQL dla używanej wersji serwera. Nie ma możliwości użycia zapytań PREPARE, EXECUTE i DEALLOCATE PREPARE wraz z PREPARE.

Zapytanie preparowane może zawierać znaki ? działające w charakterze miejsc zarezerwowanych. W trakcie późniejszego wykonywania zapytania należy dostarczyć wartości danych dołączane do wspomnianych miejsc zarezerwowanych. Miejsca zarezerwowane pozwalają na parametryzację zapytań, co pozwala na użycie tego samego zapytania preinterpretowanego z różnymi wartościami danych w trakcie kolejnych jego wywołań. Miejsca zarezerwowane nie mogą być używane dla identyfikatorów lub słów kluczowych SQL.

Zapytania PREPARE, EXECUTE i DEALLOCATE zapewniają działający na poziomie SQL interfejs dla zapytań preinterpretowanych. Nie są takie same i nie są tak samo efektywne jak binarne API dla zapytań preinterpretowanych, które przedstawiono w rozdziale 7., zatytułowanym „Tworzenie programów MySQL przy użyciu języka C”.

PURGE BINARY LOGS

```
PURGE {BINARY | MASTER} LOGS {TO 'log_name' | BEFORE 'date'}
```

Zapytanie powoduje usunięcie w serwerze wszystkich plików binarnego dziennika zdarzeń, które zostały wygenerowane wcześniej niż wskazany plik lub data (podana w formacie 'CCYY-MM-DD HH:mm:ss'). Plik indeksu binarnego dziennika zdarzeń zostaje wyzerowany i zawiera jedynie pozostałe pliki binarnego dziennika zdarzeń. Normalnie, omawiane zapytanie jest stosowane po wykonaniu w każdym serwerze podległym serwerowi głównemu zapytania SHOW SLAVE STATUS, pozwalającego na określenie pozostałych w użyciu plików dziennika zdarzeń. Wymagane jest posiadanie uprawnień SUPER.

Poniższe zapytanie powoduje usunięcie plików od *binlog.000001* do *binlog.000009*, o ile istnieją, i powoduje, że *binlog.000010* staje się pierwszym z pozostałych plików binarnego dziennika zdarzeń:

```
PURGE BINARY LOGS TO 'binlog.000010';
```

RELEASE SAVEPOINT

`RELEASE SAVEPOINT savepoint_name`

Zwolnienie punktu kontrolnego o podanej nazwie z punktów kontrolnych dla bieżącej transakcji. Jeśli punkt kontrolny nie istnieje, wtedy zostanie wygenerowany błąd. Nie jest przeprowadzana żadna operacja zatwierdzenia lub wycofania transakcji.

RENAME TABLE

`RENAME {TABLE | TABLES} tbl_name TO new_tbl_name
[, tbl_name TO new_tbl_name] ...`

Zapytanie powoduje zmianę nazwy jednej lub więcej tabel. Jego działanie jest podobne do zapytania `ALTER TABLE ... RENAME`, za wyjątkiem faktu, że ma możliwość zmiany nazwy jednocześnie wielu tabel i nałożenia na nie blokady w trakcie operacji. To będzie zaletą, jeśli chcesz przeprowadzić „niepodzielną” operację zmiany nazwy, która uniemożliwi innym klientom uzyskanie dostępu do tabeli w trakcie całej operacji.

Jeżeli zmienisz nazwę tabeli InnoDB, do której odwołują się inne tabele przez relację klucza zewnętrznego, silnik InnoDB automatycznie zmodyfikuje zależność, aby wskazywana była tabela o zmienionej nazwie.

Zapytanie `RENAME TABLE` nie może być używane względem tabel tymczasowych.

Jeżeli spróbujesz zmienić nazwę tabeli i podać nazwę innej bazy danych niż tej, w której ją utworzono, wtedy zostanie wygenerowany błąd.

Zapytania `RENAME TABLE` można używać także względem widoków, o ile nie próbujesz podać nazwy widoku w innej bazie danych.

RENAME USER

`RENAME USER from_account TO to_account
[, from_account TO to_account] ...`

Zapytanie powoduje zmianę nazwy jednego lub więcej kont użytkowników MySQL. Wymagane jest posiadanie globalnego uprawnienia `CREATE USER` lub `UPDATE` do bazy danych `mysql`.

Każda nazwa *from_account* jest zmieniana na *to_account*. Jeżeli *form_account* nie istnieje lub *to_account* już istnieje, wtedy zostanie wygenerowany błąd. Nazwę konta użytkownika trzeba podać w formacie '*nazwa_użytkownika*'@'*nazwa_komputera*', jak omówiono w podpunkcie 13.2.1.1, zatytułowanym „Określanie nazw kont”.

`RENAME USER 'myname'@'localhost' TO 'yourname'@'localhost';`

Zapytanie `RENAME USER` powoduje uaktualnienie uprawnień posiadanych przez oryginalne konto użytkownika i ich przekazanie nowemu kontu użytkownika. Jednak zapytanie nie wprowadza żadnych zmian w definicjach obiektu odwołujących się do oryginalnego konta użytkownika. Na przykład, każdy program składowany wraz z klauzulą `DEFINER` odwołującą się do oryginalnego konta użytkownika musi zostać uaktualniony, aby odwoływał się do nowego konta użytkownika.

REPAIR TABLE

```
REPAIR [NO WRITE TO BINLOG | LOCAL]
{TABLE | TABLES} tbl_name [, tbl_name] ... [option] ...
```

To zapytanie przeprowadza operację naprawy tabeli; działa wraz z tabelami MyISAM, ARCHIVE i CSV. Konieczne jest posiadanie uprawnień SELECT i INSERT do każdej tabeli.

Jeżeli włączony jest binarny dziennik zdarzeń, wykonanie zapytania REPAIR TABLE zostanie w nim zapisane, o ile nie zostanie użyta opcja NO_WRITE_TO_BINLOG lub LOCAL.

Zapytanie REPAIR TABLE bez opcji powoduje przeprowadzenie zwykłej operacji naprawy tabeli, która może usunąć większość najczęściej spotykanych problemów. Wyjątkiem są problemy związane z wystąpieniami powielonych wartości w indeksie, który powinien zawierać jedynie unikalne wartości. Przedstawiona poniżej lista wymienia dozwolone wartości *option*. Wspomniane opcje mają zastosowanie względem tabel MyISAM, pozostałe silniki baz danych mogą je ignorować.

- EXTENDED. Ta opcja powoduje przeprowadzenie rozszerzonej naprawy, w trakcie której następuje ponowne utworzenie indeksów.
- QUICK. Ta opcja próbuje szybkiej naprawy jedynie indeksów, plik danych pozostaje nienaruszony.
- USE_FRM. Ta opcja używa pliku *.frm* tabeli w celu ponownej inicjalizacji pliku indeksu oraz określenia sposobu interpretacji pliku danych, aby na nowo mogły zostać utworzone indeksy. Wykorzystanie omawianej opcji może być użyteczne w przypadku utraty lub uszkodzenia indeksu. Jednak tę opcję należy traktować jako ostatnią deskę ratunku i używać *tylko* wtedy, gdy tabela została utworzona za pomocą tej samej wersji MySQL. W przeciwnym razie ryzykujesz dalsze uszkodzenie tabeli. Opcji USE_FRM nie można używać w przypadku tabel partycjonowanych.

Zapytanie REPAIR TABLE powoduje wygenerowanie danych wyjściowych w formacie opisanym w podpunkcie dotyczącym CHECK TABLE.

REPLACE

```
REPLACE [DELAYED | LOW_PRIORITY]
[INTO] tbl_name
[PARTITION (partition_name [, partition_name] ...)]
[(col_list)]
{VALUES | VALUE} (expr [, expr] ...) [, (...)] ...

REPLACE [DELAYED | LOW_PRIORITY]
[INTO] tbl_name
[PARTITION (partition_name [, partition_name] ...)]
[(col_list)]
{SELECT ... | (SELECT ...)}

REPLACE [DELAYED | LOW_PRIORITY]
[INTO] tbl_name
[PARTITION (partition_name [, partition_name] ...)]
SET col_name=expr [, col_name=expr] ...
```

Podstawowe działanie zapytania REPLACE przypomina zapytanie INSERT, ale za wyjątkiem faktu, że rekord przeznaczony do wstawienia ma dla unikalnego indeksu wartość powielającą wartość w rekordzie znajdującym się już w tabeli. Przed wstawieniem nowego, stary rekord zostanie usunięty. Z tego powodu w składni zapytania REPLACE nie jest dostępna klauzula IGNORE. Ponadto, zapytanie REPLACE nie obsługuje ON DUPLICATE KEY UPDATE. Więcej informacji znajdziesz w podpunkcie poświęconym zapytaniu INSERT. Zapytanie REPLACE wymaga uprawnień INSERT i DELETE do tabeli.

Począwszy od wydania MySQL 5.6.2, zapytanie REPLACE obsługuje klauzulę PARTITION dla tabel partycjonowanych, co pozwala na wskazanie partycji lub podpartycji, do której mają być wstawione rekordy. W takim przypadku, jeśli rekord nie będzie mógł być wstawiony do wskazanej partycji, nastąpi wygenerowanie błędu.

Jeśli tabela zawiera wiele unikalnych indeksów, istnieje możliwość, że zapytanie DELETE usunie więcej niż tylko jeden rekord. Taka sytuacja zdarza się, jeśli nowy rekord spowoduje dopasowanie wartości w kilku unikalnych indeksach. Wówczas wszystkie dopasowane rekordy zostaną usunięte przed wstawieniem nowego.

RESET

`RESET option [, option] ...`

Zapytanie RESET jest podobne do FLUSH pod tym względem, że wpływa na dzienniki zdarzeń i buforowanie. Wykonanie zapytania RESET wymaga uprawnień RELOAD.

Wartość *option* powinna być wybrana z poniższej listy:

- **MASTER.** Usunięcie istniejących plików binarnego dziennika zdarzeń w serwerze głównym replikacji, utworzenie nowego pliku z numerem sekwencji 000001 i wyzerowanie pliku indeksu dziennika binarnego, aby zawierał jedynie nazwę nowego pliku.
- **QUERY CACHE.** Opróżnienie bufora zapytań i usunięcie wszystkich aktualnie zarejestrowanych w nich zapytań. Aby przeprowadzić defragmentację bufora bez jego opróżniania, należy użyć zapytania FLUSH QUERY CACHE.
- **SLAVE.** Jeżeli serwer jest serwerem podległym replikacji, to otrzymuje polecenie usunięcia wszystkich istniejących plików dziennika przekazywania, utworzenia nowego pliku dziennika przekazywania i wyzerowania koordynatu replikacji (to znaczy bieżącej nazwy pliku binarnego dziennika replikacji i położenia w tym pliku).

REVOKE

```
REVOKE priv_type [(col_list)] [, priv_type [(col_list)] ...]
ON [TABLE | FUNCTION | PROCEDURE]
  {*. * | * | db_name.* | db_name.tbl_name
   | tbl_name | db_name.routine_name}
  | tbl_name | db_name.routine_name}
FROM account [, account] ...
REVOKE ALL [PRIVILEGES], GRANT OPTION
FROM account [, account] ...
```

```
REVOKE PROXY ON account
FROM account [, account] ...
```

Zapytanie REVOKE powoduje odebranie uprawnień wskazanemu kontu użytkownika. Nazwę konta użytkownika trzeba podać w formacie '*nazwa_użytkownika*'@'*nazwa_komputera*', jak omówiono w podpunkcie 13.2.1.1, zatytułowanym „Określanie nazw kont”.

Jeżeli wskazane konto nie istnieje, wówczas zostanie wygenerowany błąd.

W pierwszej składni zapytania *priv_type*, *col_list* i klauzula ON są podawane w dokładnie taki sam sposób jak w zapytaniu GRANT. W celu wykonania zapytania REVOKE trzeba mieć uprawnienie GRANT OPTION oraz uprawnienia, które są odbierane.

Druga składnia ma stałą listę uprawnień oraz jest pozbawiona klauzuli ON. Powoduje odebranie wszystkich uprawnień posiadanych przez wymienione konta użytkownika. Druga składnia wymaga posiadania globalnego uprawnienia CREATE USER lub UPDATE do tabeli mysql.

Trzecia składnia powoduje odebranie kontu użytkownika w klauzuli ON uprawnienia PROXY dotyczącego kont użytkowników wskazanych w klauzuli FROM.

Zapytanie REVOKE nie powoduje usunięcia rekordu konta użytkownika z tabeli uprawnień mysql.user. Oznacza to, że danego konta można nadal używać w celu nawiązania połączenia z serwerem MySQL, nawet jeśli temu kontu użytkownika odebrane zostały wszystkie uprawnienia. Aby całkowicie usunąć konto użytkownika z serwera, należy wykonać zapytanie DROP USER (lub ręcznie usunąć rekord konta użytkownika z tabeli mysql.user).

- Odebranie uprawnień pozwalających użytkownikowi member_mgr na modyfikację tabeli member w bazie danych sampdb odbywa się po wykonaniu poniższego zapytania:

```
REVOKE INSERT,DELETE,UPDATE ON sampdb.member
FROM 'member_mgr'@'boa.example.com';
```

- Odebranie użytkownikowi anonimowemu w komputerze lokalnym wszystkich uprawnień dla pojedynczej tabeli w bazie danych menagerie odbywa się po wykonaniu poniższego zapytania:

```
REVOKE ALL ON menagerie.pet FROM ''@'localhost';
```

- Opcja ALL odbiera wszystkie uprawnienia poza GRANT OPTION. Aby odebrać także i wymienione uprawnienie, trzeba je wyraźnie podać:

```
REVOKE GRANT OPTION ON menagerie.pet FROM ''@'localhost';
```

- Odebranie użytkownikowi superduper@localhost wszystkich uprawnień na wszystkich poziomach odbywa się po wykonaniu poniższego zapytania:

```
REVOKE ALL, GRANT OPTION FROM 'superduper'@'localhost';
```

- Odebranie użytkownikowi proxy_user uprawnienia PROXY dla proxied_user odbywa się po wykonaniu poniższego zapytania:

```
REVOKE PROXY ON proxied_user FROM proxy_user;
```

ROLLBACK

```
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [WORK] TO [SAVEPOINT] savepoint_name
```

Zapytanie ROLLBACK wycofuje zmiany wprowadzone przez zapytania będące częścią bieżącej transakcji. Omawiane zapytanie działa jedynie w transakcyjnych silnikach bazy danych. (W przypadku nietransakcyjnych silników bazy danych zapytania są zatwierdzane w trakcie ich wykonywania i dlatego nie mogą być wycofane).

Opcjonalne słowo kluczowe WORK nie ma efektu. Klauzule CHAIN i RELEASE mają taki sam efekt, jak opisany w podpunkcie dotyczącym zapytania COMMIT.

W przypadku użycia klauzuli TO SAVEPOINT zapytanie ROLLBACK spowoduje wycofanie transakcji jedynie do wskazanego punktu kontrolnego.

Zapytanie ROLLBACK nie ma nic do zrobienia, jeśli tryb automatycznego zatwierdzania nie został wyłączony za pomocą START TRANSACTION lub przez przypisanie wartości 0 zmiennej autocommit.

SAVEPOINT

```
SAVEPOINT savepoint_name
```

Utworzenie w transakcji punktu kontrolnego o podanej nazwie. Istniejący punkt kontrolny o wskazanej nazwie zostanie usunięty. Zapytania wykonane w dalszej części transakcji będą mogły być wycofane do punktu kontrolnego za pomocą zapytania ROLLBACK TO SAVEPOINT.

SELECT

```
SELECT
  [select_option] ...
select_expr [, select_expr] ...
  [FROM tbl_refs]
  [WHERE where_expr]
  [GROUP BY {col_name | expr | position}
    [ASC | DESC], ... [WITH ROLLUP]]
  [HAVING where_expr]
  [ORDER BY {col_name | expr | position}
    [ASC | DESC], ...]
  [LIMIT {[skip_count,] show_count | show_count OFFSET skip_count}]
  [PROCEDURE procedure_name([param_list])]
  [
    INTO OUTFILE 'file_name'
      [CHARACTER SET charset]
      [field_options] [line_options]
    | INTO DUMPFILE 'file_name'
    | INTO var_name [, var_name] ...
  ]
[FOR UPDATE | LOCK IN SHARE MODE] ]
```

Zapytanie SELECT jest normalnie używane w celu pobrania rekordów z jednej lub więcej tabel. Ponieważ wszystko poza słowem kluczowym SELECT i co najmniej jednym

wyrażeniem `select_expr` jest opcjonalne, istnieje możliwość tworzenia zapytań przeznaczonych po prostu do obliczania wartości wyrażenia, na przykład:

```
SELECT 'one plus one =', 1+1;
```

W celu zachowania zgodności z systemami baz danych wymagającymi klauzuli `FROM` w zapytaniu `SELECT`, MySQL obsługuje także pseudotabelę `DUAL`:

```
SELECT 'one plus one =', 1+1 FROM DUAL;
```

Podzapytanie to zapytanie `SELECT` zagnieżdżone w innym; przykłady znajdziesz w podrozdziale 2.9, zatytułowanym „Pobieranie informacji z wielu tabel za pomocą podzapytań”. Podzapytania mogą być również stosowane w klauzuli `WHERE` zapytań `DELETE` i `UPDATE` oraz w zapytaniach `INSERT` i `REPLACE`. Jednak nie ma możliwości użycia podzapytania do pobrania rekordów z modyfikowanej tabeli.

Każda wartość `select_option` może być jedną z opcji pobranych z poniższej listy:

- **ALL, DISTINCT, DISTINCTROW.** To słowo kluczowe określa, kiedy zwracane będą powielone rekordy. Opcja `ALL` powoduje zwrot wszystkich rekordów, to jest opcja domyślna. Z kolei `DISTINCT` i `DISTINCTROW` wskazują, że powielone rekordy muszą być eliminowane ze zbioru wynikowego.
- **HIGH_PRIORITY.** Opcja `HIGH_PRIORITY` powoduje nadanie zapytaniu wyższego priorytetu, jeśli normalnie miałyby oczekiwać na wykonanie. Jeżeli inne zapytania, takie jak `INSERT` lub `UPDATE`, czekają na przeprowadzenie operacji zapisu w tabelach wymienionych w `SELECT`, ponieważ inne klienty odczytują dane z tych tabel, wtedy opcja `HIGH_PRIORITY` powoduje, że zapytanie `SELECT` będzie miało wyższy priorytet niż wspomniane zapytania zapisujące dane. Takie rozwiązanie należy stosować jedynie do zapytań `SELECT`, o których wiadomo, że zostaną wykonane szybko. Musi być to zrobione natychmiast, ponieważ spowalnia wykonanie zapytań zapisu danych. Ta opcja jest efektywna jedynie dla silników bazy danych stosujących nakładanie blokad na poziomie tabeli, na przykład `MyISAM` lub `MEMORY`.
- **SQL_BUFFER_RESULT.** Ta opcja nakazuje serwerowi buforowanie wyniku zapytania w oddzielnej tabeli tymczasowej zamiast zachowania nałożonych blokad na tabele wymienione w zapytaniu `SELECT` w oczekiwaniu, aż cały wynik zapytania zostanie przekazany klientowi. Dzięki takiemu rozwiązaniu można wcześniej zwolnić blokady, co z kolei pozwala innym klientom znacznie szybciej uzyskać dostęp do tabel. Jednak użycie omawianej opcji wiąże się również z większym zużyciem miejsca na dysku oraz pamięci.
- **SQL_CACHE, SQL_NO_CACHE.** Jeżeli wynik zapytania można buforować, a bufor zapytań działa w trybie na żądanie, wtedy opcja `SQL_CACHE` powoduje buforowanie wspomnianego wyniku zapytania. Z kolei opcja `SQL_NO_CACHE` wyłącza wszelkie buforowanie wyniku zapytania. Obie wymienione opcje są wzajemnie wykluczające się i nie mogą być stosowane w podzapytaniach lub zapytaniach innych niż pierwsze `SELECT` w `UNION`.

- **SQL_CALC_FOUND_ROWS**. Normalnie liczba rekordów z zapytania zawierającego klauzulę **LIMIT** jest liczbą faktycznie zwróconych rekordów. Opcja **SQL_CALC_FOUND_ROWS** nakazuje serwerowi ustalenie wielkości wyniku zapytania, gdyby nie zawierało klauzuli **LIMIT**. Aby poznać tę wielkość, należy wykonać zapytanie **SELECT FOUND_ROWS()** natychmiast po początkowym zapytaniu **SELECT**.
- **SQL_BIG_RESULT**, **SQL_SMALL_RESULT**. Wymienione słowa kluczowe zawierają wskazówkę informującą, że zbiór wynikowy będzie odpowiednio ogromny lub mały. W ten sposób optymalizator otrzymuje informacje, które może wykorzystać do znacznie efektywniejszego przetworzenia zapytania.
- **STRAIGHT_JOIN**. Ta opcja wymusza, aby tabele były złączane w kolejności podanej w klauzuli **FROM**. Opcja może być użyteczna, jeśli jesteś przekonany, że optymalizator nie dokonuje najlepszych wyborów.

Wyrażenie *select_expr* zawiera rozdzieloną przecinkami listę kolumn, które mają znajdować się w danych wyjściowych. Kolumny mogą być odniesieniami do kolumn tabel lub wyrażeniami (włączając w nie podzapytania skalarne). Dowolnej kolumnie może być przypisany alias kolumny za pomocą składni **AS alias_name** (słowo kluczowe **AS** jest opcjonalne). Alias staje się nazwą kolumny w danych wyjściowych i można się do niego odwoływać w klauzulach **GROUP BY**, **ORDER BY** i **HAVING**. Natomiast do aliasów kolumn nie można się odwoływać w klauzuli **WHERE**.

Zapis specjalny ***** oznacza „wszystkie kolumny z tabel wskazanych w klauzuli **FROM**”, natomiast *tbl_name*.***** oznacza „wszystkie kolumny z wymienionej tabeli”.

Klauzula **FROM** zawiera nazwę jednej lub więcej kolumn, z których powinny być pobrane rekordy. Serwer MySQL obsługuje następującą składnię złączenia:

```
tbl_refs:
    tbl_ref [, tbl_ref] ...

tbl_ref:
    tbl_factor
    | join_tbl

tbl_factor:
    tbl_name
    | (subquery) [AS] alias_name
    | (tbl_refs)
    | {OJ tbl_ref LEFT OUTER JOIN tbl_ref ON conditional_expr}

join_tbl:
    tbl_ref [INNER | CROSS] JOIN tbl_factor [join_condition]
    | tbl_ref STRAIGHT_JOIN tbl_factor [ON conditional_expr]
    | tbl_ref {LEFT | RIGHT} [OUTER] JOIN tbl_ref join_condition
    | tbl_ref NATURAL [{LEFT | RIGHT} [OUTER]] JOIN tbl_factor

join_condition:
    ON conditional_expr
    | USING (col_list)
```

Każdej *tbl_name* może towarzyszyć wskazówka w postaci aliasu lub indeksu. Ponadto, począwszy od wydania MySQL 5.6.2, zapytanie SELECT obsługuje klauzulę PARTITION dla tabel partycjonowanych w celu wskazania partycji lub podpartycji, z której mają być pobrane rekordy. Pełna składnia odniesienia się do tabeli w rzeczywistości przedstawia się następująco:

```
tbl_name
[PARTITION (partition_name [, partition_name] ...)]
[[AS] alias_name]
[[USE | IGNORE | FORCE] {INDEX | KEY}
[FOR {JOIN | ORDER BY | GROUP BY}]
(index_list)]
```

Tabelom mogą być w klauzuli FROM przypisane aliasy za pomocą składni *tbl_name nazwa_aliasu* lub *tbl_name AS nazwa_aliasu*. Alias zapewnia alternatywną nazwę, za pomocą której można odwoływać się do kolumn tabeli w dowolnym miejscu zapytania.

Dozwolone jest również podanie tabeli w klauzuli FROM w postaci podzapytania ujętego w nawias, o ile zostanie użyty alias pozwalający na odwołanie się do tabeli w dowolnym miejscu zapytania:

```
SELECT * FROM (SELECT 1) AS t;
```

Użycie klauzul USE INDEX, IGNORE INDEX i FORCE INDEX dostarcza pewnych wskazówek optymalizatorowi. Takie rozwiązanie może być użyteczne w sytuacji, gdy optymalizator nie dokonuje najlepszego wyboru indeksu podczas złączenia. Klauzula USE INDEX nakazuje optymalizatorowi wybór indeksu jedynie spośród wymienionych na liście *index_list*. Z kolei klauzula IGNORE INDEX wskazuje optymalizatorowi indeksy, które nie powinny być używane. Natomiast FORCE INDEX jest jak USE INDEX, ale nakazuje optymalizatorowi uznanie skanowania tabeli za bardzo kosztowne rozwiązanie w porównaniu do użycia indeksu.

Lista *index_list* powinna zawierać nazwę jednego lub wielu rozdzielonych przecinkami indeksów. W przypadku klauzuli USE lista *index_list* może być pusta i oznacza wówczas „nie używaj indeksu”. Każdy indeks powinien być nazwą indeksu w tabeli lub słowem kluczowym PRIMARY oznaczającym PRIMARY KEY dla danej tabeli.

Wskazówki dotyczące indeksu mają zastosowanie jedynie podczas pobierania rekordów i złączeń tabel, a nie podczas przetwarzania rekordów za pomocą klauzul ORDER BY lub GROUP BY. Aby zastosować wskazówki jedynie do wybierania i złączania, użyj FOR JOIN.

Dla tabeli dozwolone jest stosowanie wielu wskazówek dotyczących indeksu, ale w tym samym odwołaniu do tabeli nie można używać jednocześnie USE INDEX i FORCE INDEX.

W przypadku indeksów typu FULLTEXT wskazówki dotyczące indeksu nie mają żadnego efektu poza wyszukiwaniem w trybie boolowskim z modyfikatorem FOR JOIN lub bez modyfikatora FOR.

Złączenie pobiera rekordy z wymienionych tabel zgodnie z przedstawionymi poniżej opisami. Liczba rekordów faktycznie zwracanych klientowi może być ograniczona za pomocą klauzul WHERE, HAVING lub LIMIT.

- W przypadku pojedynczej tabeli wymienionej z nazwy zapytanie SELECT pobiera rekordy ze wskazanej tabeli.

- Jeżeli wymienionych zostanie więcej tabel, których nazwy rozdzielono przecinkami, zapytanie SELECT zwróci wszystkie możliwe kombinacje rekordów z tabel. Użycie JOIN, CROSS JOIN lub INNER JOIN jest podobne do użycia przecinka, jeśli nie ma klauzuli ON lub USING. Zastosowanie STRAIGHT_JOIN jest podobne, ale wymusza na optymalizatorze połączenie tabel w kolejności ich wymienienia. Takie rozwiązanie okaże się użyteczne, jeśli jesteś przekonany, że optymalizator nie dokonuje najlepszych wyborów.
- W przeciwieństwie do operatora przecinka, połączenia przeprowadzane za pomocą JOIN, CROSS JOIN lub INNER JOIN mogą być podane w klauzuli ON lub USING() w celu ograniczenia dopasowań między tabelami. Dopasowanie rekordów odbywa się według warunków wymienionych w klauzuli ON *conditional_expr* lub USING(*col_list*). Klauzula *conditional_expr* jest wyrażeniem w postaci, którą można stosować w klauzuli WHERE. Z kolei *col_list* składa się z jednej lub wielu rozdzielonych przecinkami nazw kolumn, z których każda musi być kolumną istniejącą w obu łączanych tabelach.
- LEFT JOIN pobiera rekordy ze łączanych tabel, ale wymusza wygenerowanie rekordu dla każdego rekordu znajdującego się lewej tabeli, nawet jeśli dla niego nie istnieje dopasowany rekord w prawej tabeli. W przypadku braku dopasowania, kolumny z prawej tabeli będą zwracane w postaci wartości NULL. Klauzula ON lub USING() znajdująca się po nazwie tabel jest stosowana jak w przypadku JOIN, CROSS JOIN lub INNER JOIN. Klauzula LEFT OUTER JOIN jest odpowiednikiem LEFT JOIN, podobnie jak składnia rozpoczynająca się od OJ, która została dołączona w celu zapewnienia zgodności z ODBC. Nawiasy klamrowe dla składni OJ nie są metaznakami — to dosłowne znaki, które muszą istnieć w zapytaniu.
- NATURAL LEFT JOIN jest odpowiednikiem LEFT JOIN USING(*col_list*), gdzie *col_list* zawiera wszystkie kolumny występujące w obu tabelach.
- Typ RIGHT JOIN jest podobny do LEFT JOIN, ale role tabel są odwrócone.
- Połączenie za pomocą przecinka ma mniejsze pierwszeństwo niż pozostałe typy połączeń. Stosowanie połączenia za pomocą przecinka wraz z innym typem połączenia może doprowadzić do wygenerowania błędów informujących o nieznannej kolumnie. W takich przypadkach zastąpienie przecinka klauzulą INNER JOIN najczęściej rozwiązuje problem.

Klauzula WHERE określa wyrażenie, które będzie zastosowane względem rekordów pobranych z tabeli wymienionej w klauzuli FROM. Rekordy niespełniające kryteriów zdefiniowanych w wyrażeniu zostaną odrzucone. Zbiór wynikowy może być jeszcze bardziej ograniczony za pomocą klauzul HAVING i LIMIT. W klauzuli WHERE nie można odwoływać się do aliasów kolumn.

Klauzule GROUP BY i ORDER BY mają podobną składnię. Klauzula GROUP BY *col_list* jest używana do grupowania zbioru wynikowego względem kolumn wymienionych na podanej liście. Ta klauzula jest używana podczas stosowania funkcji podsumowania, takich jak COUNT() lub MAX() w wyrażeniu *select_expr*. Z kolei klauzula ORDER BY wskazuje,

że zbiór wynikowy powinien być sortowany na podstawie wymienionych kolumn. W przypadku obu omawianych klauzul do kolumn można się odwoływać za pomocą ich nazw, aliasów lub położenia na liście wyrażenia *select_expr*. Położenie kolumny jest wskazywane przez liczbę całkowitą bez znaku, pierwszą kolumnę oznacza liczba 1. Warto pamiętać, że stosowanie położenia kolumn jest niestandardowym rozwiązaniem i dlatego uznany za przestarzałe i niezalecane. W celu grupowania lub sortowania wyniku działania wyrażenia można również użyć wyrażań. Na przykład, `ORDER BY RAND()` powoduje posortowanie rekordów w losowo wybranej kolejności.

W klauzuli `GROUP BY` lub `ORDER BY` po dowolnej kolumnie na liście można podać słowo kluczowe `ASC` lub `DESC`, wskazując tym samym, że kolumna powinna być sortowana w kolejności odpowiednio rosnącej lub malejącej. Jeżeli nie zostanie użyte żadne z wymienionych słów, domyślnie sortowanie odbywa się w kolejności rosnącej. Specyfikatory sortowania są dozwolone w klauzulach `GROUP BY`, ponieważ w serwerze MySQL klauzula `GROUP BY` nie tylko grupuje rekordy, ale również sortuje wynik. Kolejność danych wyjściowych ustalona przez `GROUP BY` jest nadpisywana przez dowolną podaną klauzulę `ORDER BY`. Aby uniemożliwić niejawnie sortowanie przez klauzulę `GROUP BY` (i tym samym nie powodować obciążenia związanego z operacją sortowania), należy użyć `ORDER BY NULL`.

Klauzula `WITH ROLLUP` może być używana na końcu `GROUP BY`. Powoduje, że dane wyjściowe będą zawierały rekordy podsumowania dla połączeń wysokiego poziomu zgrupowanych kolumn plus dodatkowe podsumowanie ogólne na końcu.

Klauzula `HAVING` określa drugorzędne wyrażenie stosowane do ograniczenia liczby rekordów, które spełniły warunki zdefiniowane w klauzuli `WHERE` oraz po ich pogrupowaniu zgodnie z klauzulą `GROUP BY`. Rekordy niespełniające warunku `HAVING` zostaną odrzucone. Klauzula `HAVING` jest użyteczna dla wyrażań obejmujących funkcje podsumowania, które nie mogą zostać zastosowane w klauzuli `WHERE`. Jednak jeśli umieszczenie warunku jest poprawne zarówno w klauzuli `WHERE`, jak i `HAVING`, wtedy najlepiej podać go w klauzuli `WHERE`, ponieważ będzie tam analizowany przez optymalizator.

Klauzula `LIMIT` może być używana w celu wybrania pewnych rekordów ze zbioru wynikowego. Pobiera ona jeden lub dwa argumenty, które muszą być w postaci liczb całkowitych. Klauzula `LIMIT n` zwraca pierwsze n rekordów, podczas gdy `LIMIT m, n` pomija pierwsze m rekordów i zwraca następne n rekordów.

`PROCEDURE` wskazuje nazwę procedury, do której dane znajdujące się w zbiorze wynikowym zostaną wysłane przez zwróceniem ich klientowi. Opcjonalny parametr *param_list* to rozdzielona przecinkami lista wartości przekazywanych procedurze. Istnieje możliwość użycia `PROCEDURE ANALYSE()` w celu uzyskania informacji o cechach charakterystycznych danych znajdujących się w kolumnach wymienionych na liście kolumn. Zapoznaj się z podrozdziałem 5.3, zatytułowanym „Wybór typu danych zapewniającego efektywne wykonywanie zapytań”.

Klauzula `INTO` wskazuje miejsce docelowe dla wyników zapytania. Alternatywne miejsce dla `INTO` to umieszczenie tej klauzuli wcześniej w zapytaniu po liście *select_expr*. Jeżeli używasz klauzuli `INTO`, zapytanie nie może być zagnieżdżonym zapytaniem `SELECT`.

Wynik wykonania zapytania SELECT można zapisać w pliku o podanej nazwie (*file_name*) za pomocą klauzuli INTO outfile '*file_name*'. Aby wskazać kodowanie znaków dla zapisywanych wartości, trzeba użyć klauzuli CHARACTER SET. W przypadku pominięcia wymienionej klauzuli lub użycia wartości binary dla kodowania znaków, nie będzie przeprowadzana żadna konwersja. Składnia *field_options* i *line_options* są takie same jak odpowiadających im klauzul w zapytaniu LOAD DATA. Więcej informacji znajdziesz w podpunkcie dotyczącym zapytania LOAD DATA.

Klauzula INTO DUMPFILE '*file_name*' jest podobna do INTO outfile, ale zapisuje jedynie pojedynczy rekord, a dane wyjściowe zupełnie nie są interpretowane. Oznacza to zapis niezmodyfikowanych wartości bez ograniczników, cytowania lub znaków ograniczających. Takie rozwiązanie może być użyteczne w przypadku zapisu w pliku danych BLOB, takich jak obraz lub inne dane binarne.

W przypadku zarówno INTO outfile, jak i INTO DUMPFILE położenie pliku podlega takim samym regułom, jakie mają zastosowanie podczas odczytu plików innych niż LOCAL za pomocą zapytania LOAD DATA. Konieczne jest posiadanie uprawnienia FILE, plik danych wyjściowych nie może istnieć, a sam plik zostanie utworzony w serwerze i będzie dostępny dla innych użytkowników. Właścicielem pliku będzie użytkownik, którego konto zostało użyte do uruchomienia serwera.

Po klauzuli INTO znajduje się rozdzielona przecinkami lista nazw zmiennych przechowujących wyniki zapytania SELECT. Każda z wspomnianych zmiennych może być zmienną zdefiniowaną przez użytkownika (w postaci @nazwa_zmiennej) lub w programie składowanym parametrem bądź zmienną lokalną. Zapytanie musi pobierać pojedynczy rekord wartości, a ponadto musi podawać po jednej nazwie zmiennej dla kolumny danych wyjściowych.

Klauzule FOR UPDATE i LOCK IN SHARE MODE powodują nałożenie blokad na rekordy analizowane podczas wykonywania zapytania. Blokada pozostaje aktywna aż do chwili zatwierdzenia aktualnej transakcji lub jej wycofania. Wspomniane blokady mogą być użyteczne w transakcjach obejmujących wiele zapytań. Jeżeli użyjesz FOR UPDATE w tabeli, dla której silnik bazy danych stosuje nakładanie blokad na poziomie rekordów (InnoDB), wtedy analizowane rekordy będą miały nałożoną blokadę zapisu na wyłączność. Klauzula LOCK IN SHARE MODE powoduje nałożenie blokady odczytu na rekordy, co pozwala innym klientom na odczyt rekordów, ale już nie na ich modyfikacje. Warto dodać, że jeśli optymalizator zapytania nie znajdzie indeksu do użycia dla analizowanych rekordów, wówczas musi przeprowadzić skanowanie wszystkich rekordów w tabeli, co oznacza nałożenie na nie blokady.

Poniższe zapytania pokazują pewne sposoby wykorzystania zapytania SELECT. Więcej innych przykładów znajdziesz w rozdziałach 1., „Rozpoczęcie pracy z MySQL”, i 2., „Użycie MySQL do zarządzania danymi”.

- Pobranie całej zawartości tabeli:

```
SELECT * FROM president;
```

- Pobranie całej zawartości, ale posortowanej według nazwiska i imienia:

```
SELECT * FROM president ORDER BY last_name, first_name;
```

- Pobranie rekordów prezydentów urodzonych po '1900-01-01':

```
SELECT * FROM president WHERE birth >= '1900-01-01';
```
- Taka sama operacja, ale sortowanie odbywa się w kolejności daty urodzenia:

```
SELECT * FROM president WHERE birth >= '1900-01-01' ORDER BY birth;
```
- Określenie stanów przedstawianych przez rekordy tabeli member:

```
SELECT DISTINCT state FROM member;
```
- Pobranie rekordów z tabeli member i zapis kolumn w pliku w postaci wartości rozdzielonych przecinkami:

```
SELECT * INTO OUTFILE '/tmp/member.txt'
  FIELDS TERMINATED BY ',' FROM member;
```
- Pobranie pięciu najlepszych wyników ze wskazanej tabeli:

```
SELECT * FROM score WHERE event_id = 9 ORDER BY score DESC LIMIT 5;
```

SET

```
SET assignment [, assignment ] ...
assignment: var_name {= | :=} expr
```

Zapytanie SET powoduje przypisanie wartości zmiennym systemowym, zmiennym zdefiniowanym przez użytkownika oraz parametrom lub zmiennym lokalnym w programach składowanych. W dodatku D, zatytułowanym „Przewodnik po zmiennych systemowych, stanu i użytkownika”, znajdziesz więcej informacji na temat zmiennych systemowych i zdefiniowanych przez użytkownika. Natomiast w punkcie E.2.2, zatytułowanym „Zapytania obsługi deklaracji”, omówiono składnię deklaracji dla zmiennych lokalnych programu składowanego. Zapytanie SET jest również stosowane w wielu różnych ustawieniach wspomnianych w dalszej części tego podpunktu.

Inne zapytania rozpoczynające się od SET (SET PASSWORD i SET TRANSACTION) zostały przedstawione w oddzielnych podpunktach w dalszej części dodatku.

Kiedy zapytanie SET jest używane w celu przypisania wartości zmiennym, *var_name* w każdym przypisaniu wskazuje zmienną, której jest przypisana wartość. Z kolei *expr* oznacza wyrażenie wskazujące wartość przypisywaną zmiennej. Operatorem przypisania w zapytaniu SET może być = lub :=.

Zapytanie SET można wykorzystać także do przypisania wartości zmiennym zdefiniowanym przez użytkownika, które mają nazwy w postaci @*var_name*:

```
SET @day = CURDATE(), @time = CURTIME();
```

Zapytanie SET jest używane również do przypisywania wartości zmiennym systemowym, z których wiele jest dynamicznych i może być zmienianych w trakcie działania serwera. Dynamiczne zmienne systemowe istnieją na dwóch poziomach. Pierwszy to globalne zmienne systemowe, które mają zasięg całego serwera i wpływają na wszystkie klienty. Drugi poziom to zmienne systemowe sesji (nazywane również lokalnymi zmiennymi systemowymi), które dotyczą jedynie danej sesji klienta. W przypadku zmiennych istniejących na obu wymienionych poziomach, dla danej sesji klienta, podczas nawiązywania przez

niego połączenia zmienne są inicjalizowane wraz z wartościami odpowiadających im zmiennych globalnych. Klient musi mieć uprawnienie SUPER, aby zmodyfikować zmienną globalną. Natomiast do modyfikacji zmiennych we własnej sesji nie są wymagane żadne uprawnienia.

Składnia przypisywania wartości zmiennym systemowym ma kilka form. W celu ustawienia zmiennej globalnej, na przykład wartości zmiennej `sql_mode`, należy wykonać zapytanie w jednej z poniższych postaci:

```
SET GLOBAL sql_mode = 'ANSI_QUOTES';c
SET @@GLOBAL.sql_mode = 'ANSI_QUOTES';
```

Z kolei przypisanie wartości zmiennej sesji wymaga zastąpienia słowa GLOBAL słowem SESSION:

```
SET SESSION sql_mode = 'ANSI_QUOTES';
SET @@SESSION.sql_mode = 'ANSI_QUOTES';
```

Istnieje również możliwość użycia słowa LOCAL, które jest synonimem dla SESSION.

Jeżeli nie zostanie podany żaden specyfikator zakresu (GLOBAL, SESSION lub LOCAL), wtedy zapytanie SET powoduje modyfikację wartości zmiennej na poziomie sesji:

```
SET sql_mode = 'ANSI_QUOTES';
SET @@sql_mode = 'ANSI_QUOTES';
```

Aby sprawdzić wartości zmiennych systemowych, należy wykonać zapytanie SHOW VARIABLES. Inną możliwością jest analiza tabel o nazwach GLOBAL_VARIABLES i SESSION_VARIABLES w bazie danych INFORMATION_SCHEMA. W celu wyświetlenia wartości pojedynczej zmiennej należy wykonać zapytanie SELECT:

```
SELECT @@GLOBAL.sql_mode, @@SESSION.sql_mode, @@LOCAL.sql_mode;
```

Więcej informacji na temat użycia zmiennych systemowych znajdziesz w punkcie 12.3.1, zatytułowanym „Sprawdzanie i ustawienie wartości zmiennych systemowych”.

Poniższa lista wymienia pewne ustawienia, których wartość można kontrolować za pomocą zapytania SET:

- SET CHARACTER SET {*charset* | DEFAULT}

Przypisanie wartości w postaci wskazanych kodowań znaków zmiennym sesji o nazwie `character_set_client` i `character_set_results`. Ponadto, zmiennej sesji `character_set_connection` zostanie przypisana wartość zmiennej `character_set_database`. Wymienione zmienne wpływają na konwersję znaków danych wysyłanych do serwera i otrzymywanych z niego. Wartością `charset` nie może być `ucs2`, `utf16`, `utf16le` lub `utf32`.

Zapytanie SET CHARACTER SET DEFAULT powoduje przywrócenie domyślnego mapowania kodowania znaków.

- SET NAMES {*charset* [COLLATE *collation*] | DEFAULT}

Przypisanie zmiennym sesji `character_set_client`, `character_set_connection` i `character_set_results` wskazanego kodowania znaków, natomiast zmiennej `collation_connection` domyślnej kolejności sortowania dla danej wartości `character_set_connection`. Klauzula COLLATE może zostać użyta, aby wyraźnie

wskazać wybraną kolejność sortowania. Wartości `charset` i `collation` mogą być w postaci cytowanych lub niecytowanych ciągów tekstowych. Wymienione zmienne wpływają na konwersję znaków danych wysyłanych do serwera i otrzymywanych z niego. Wartością `charset` nie może być `ucs2`, `utf16`, `utf16le` lub `utf32`.

Zapytanie `SET NAMES DEFAULT` powoduje przywrócenie domyślnego mapowania kodowania znaków.

SET PASSWORD

```
SET PASSWORD [FOR account] = PASSWORD('pass_val')
SET PASSWORD [FOR account] = OLD_PASSWORD('pass_val')
SET PASSWORD [FOR account] = 'encrypted_pass_val'
```

Zapytanie `SET PASSWORD` zmienia hasło do konta użytkownika MySQL. Użytkownik zawsze ma możliwość zmiany hasła, o ile połączenie nie zostało nawiązane za pomocą konta użytkownika anonimowego. W celu zmiany hasła dla innego konta użytkownika trzeba mieć uprawnienie `UPDATE` do bazy danych `mysql`. Jeżeli zmienna systemowa `read_only` ma przypisaną wartość 1, konieczne jest również posiadanie uprawnienia `SUPER`.

W przypadku braku klauzuli `FOR` zapytanie spowoduje ustawienie hasła dla bieżącego konta użytkownika. Z kolei użycie klauzuli `FOR` powoduje ustawienie hasła dla wskazanego konta użytkownika. Nazwę konta użytkownika trzeba podać w formacie `'nazwa_uzytkownika'@'nazwa_komputera'`, jak omówiono w podpunkcie 13.2.1.1, zatytułowanym „Określanie nazw kont”.

Hasło, czyli `'pass_val'`, powinno być zaszyfrowane za pomocą funkcji `PASSWORD()`, która jest standardową funkcją szyfrującą. Jeśli chcesz zaszyfrować hasło w formacie stosowanym w wersji wcześniejszej niż MySQL 4.1, należy użyć funkcji `OLD_PASSWORD()`. Jeżeli nie będzie użyta żadna z wymienionych funkcji, trzeba podać zaszyfrowane hasło (`'encrypted_pass_val'`) w postaci ciągu tekstowego już zaszyfrowanego hasła.

```
SET PASSWORD = PASSWORD('secret');
SET PASSWORD FOR 'paul' = PASSWORD('secret');
SET PASSWORD FOR 'paul'@'localhost' = PASSWORD('secret');
SET PASSWORD FOR 'bill'@'%.bigcorp.com' = PASSWORD('old-sneep');
```

SET TRANSACTION

```
SET [GLOBAL | SESSION] TRANSACTION
  trans_characteristic [, trans_characteristic] ...
  trans_characteristic:
    ISOLATION LEVEL level
    | READ WRITE
    | READ ONLY
```

Zapytanie powoduje zdefiniowanie cech charakterystycznych przetwarzania transakcji. Pobiera opcjonalny specyfikator zakresu, określający, jak wspomniane cechy charakterystyczne będą stosowane dla transakcji.

Zakres transakcji można ustawić w następujący sposób:

- Opcja GLOBAL powoduje globalne (zasięg całego serwera) ustawienie cech charakterystycznych; stają się one poziomem domyślnym dla wszystkich klientów, które od tej chwili nawiążą połączenie z serwerem.
- Opcja SESSION powoduje ustawienie cech charakterystycznych dla danej sesji (czyli dla klienta); mają one zastosowanie dla kolejnych transakcji przeprowadzanych w bieżącej sesji.
- Jeżeli nie zostanie użyta żadna z powyższych opcji, zapytanie definiuje cechy charakterystyczne dla kolejnej transakcji w bieżącej sesji. To niedozwolone, jeśli aktualnie jest przeprowadzana aktywna transakcja.

Użycie opcji GLOBAL wymaga uprawnień SUPER. Każdy klient może zmienić cechy charakterystyczne następnej transakcji lub transakcji przeprowadzanych w bieżącej sesji.

Począwszy od wydania MySQL 5.6.3, omawiane zapytanie pobiera jedną lub więcej cech charakterystycznych rozdzielonych przecinkami. Natomiast wydania wcześniejsze niż 5.6.3 akceptowały jedynie ISOLATION LEVEL.

W przypadku podania ISOLATION LEVEL poziom transakcji wskazywany przez parametr *level* powinien mieć jedną z wymienionych poniżej wartości:

- READ UNCOMMITTED. Transakcja widzi modyfikacje wprowadzone przez inne transakcje, nawet jeśli nie zostały jeszcze zatwierdzone.
- READ COMMITTED. Transakcja widzi modyfikacje wprowadzone przez inne transakcje tylko wtedy, gdy zostały zatwierdzone.
- REPEATABLE READ. Jeżeli transakcja dwukrotnie wykonuje dane zapytanie SELECT, wynik jest powtarzalny. Oznacza to, że ten sam wynik jest otrzymywany za każdym razem, nawet jeśli w międzyczasie inne transakcje zmieniły bądź wstawiły rekordy.
- SERIALIZABLE. Poziom izolacji podobny do REPEATABLE READ, ale całkowicie izoluje transakcje. Rekordy wybrane przez jedną transakcję nie mogą być modyfikowane przez inne transakcje aż do chwili zakończenia pierwszej transakcji.

Wartość ISOLATION LEVEL ma zastosowanie do silnika bazy danych InnoDB. Domyślnym poziomem izolacji jest REPEATABLE READ. Nietransakcyjne silniki bazy danych nie mają poziomów izolacji.

Więcej informacji na temat izolacji transakcji i poziomów izolacji znajdziesz w punkcie 2.12.3, zatytułowanym „Izolacja transakcji”.

Wartości READ WRITE i READ ONLY określają tryb dostępu do transakcji. Innymi słowy, wskazują, czy transakcja zezwala na modyfikacje tabel. (Tabele tymczasowe mogą być modyfikowane niezależnie od trybu dostępu). Wymienione wartości są wzajemnie wykluczające się i dlatego nie mogą być jednocześnie używane w tym samym zapytaniu. Wprowadzono je w wydaniu MySQL 5.6.5.

SHOW

```

SHOW {BINARY | MASTER} LOGS
SHOW BINLOG EVENTS
SHOW CHARACTER SET
SHOW COLLATION
SHOW COLUMNS
SHOW CREATE DATABASE
SHOW CREATE EVENT
SHOW CREATE {FUNCTION | PROCEDURE}
SHOW CREATE TABLE
SHOW CREATE TRIGGER
SHOW CREATE VIEW
SHOW DATABASES
SHOW ENGINE
SHOW ENGINES
SHOW ERRORS
SHOW EVENTS
SHOW {FUNCTION | PROCEDURE} STATUS
SHOW GRANTS
SHOW INDEX
SHOW MASTER STATUS
SHOW OPEN TABLES
SHOW PLUGINS
SHOW PRIVILEGES
SHOW PROCESSLIST
SHOW SLAVE HOSTS
SHOW SLAVE STATUS
SHOW STATUS
SHOW TABLE STATUS
SHOW TABLES
SHOW TRIGGERS
SHOW VARIABLES
SHOW WARNINGS

```

Zapytanie `SHOW` dostarcza informacje o bazach danych i znajdujących się w nich obiektach, takich jak tabele i programy składowane, a także informacje o działaniu serwera. Wiele zapytań pobiera opcjonalną klauzulę `FROM nazwa_bazy_danych`, pozwalającą na wskazanie bazy danych, o której mają być wyświetlone informacje. Jeżeli klauzula nie zostanie podana, wtedy będzie użyta domyślna baza danych. W każdym zapytaniu obsługującym klauzulę `FROM` do wskazania nazwy tabeli lub bazy danych synonimem wymienionej klauzuli jest `IN`.

Pewne formy pozwalają na użycie opcjonalnej klauzuli `LIKE 'pattern'`, ograniczającej dane wyjściowe do wartości dopasowanych do wzorca. Wymieniony `'pattern'` jest interpretowany jako wzorzec SQL i może zawierać znaki wieloznaczne `%` i `_`.

Baza danych `INFORMATION_SCHEMA` to inny sposób uzyskania metadanych o bazie danych, a wiele tabel w bazie danych `INFORMATION_SCHEMA` zawiera informacje podobne do wyświetlanych przez zapytania `SHOW`. Ponadto, wymienione tutaj zapytania `SHOW` obsługujące klauzulę `LIKE 'pattern'` mogą być zapisane w postaci klauzuli `WHERE`, zamiast wskazywać rekordy do wyświetlenia. Więcej informacji na ten temat znajdziesz w podrozdziale 2.7, zatytułowanym „Pobieranie metadanych bazy danych”.

SHOW BINARY LOGS

SHOW {BINARY | MASTER} LOGS

Zapytanie wyświetla nazwy i wielkości bieżących plików binarnego dziennika zdarzeń. W serwerze głównym replikacji wykonanie tego zapytania może być użyteczne przed wykonaniem zapytania PURGE BINARY LOGS, ale po wykonaniu SHOW SLAVE STATUS w każdym serwerze podległym — dzięki temu można określić pliki binarnego dziennika zdarzeń aktualnie przetwarzane przez serwery podległe.

Zapytanie wymaga uprawnień SUPER bądź — począwszy od wydania MySQL 5.5.25 — uprawnień SUPER lub REPLICATION CLIENT.

SHOW BINARY EVENTS

SHOW BINLOG EVENTS [IN '*file_name*'] [FROM *position*]
[LIMIT [*skip_count*,] *show_count*]

Zapytanie wyświetla zdarzenia z binarnego dziennika zdarzeń. Wymagane jest posiadanie uprawnień REPLICATION SLAVE. W celu wyświetlenia zdarzeń z dziennika przekazywania serwera podległego należy wykonać zapytanie SHOW RELAYLOG EVENTS.

Zdarzenia z grubsza odpowiadają zapytaniom SQL. Jeżeli pominięta zostanie nazwa pliku, użyty będzie pierwszy plik binarnego dziennika zdarzeń. Pominięcie położenia powoduje rozpoczęcie odczytu zdarzeń od początku pliku. Z kolei podanie klauzuli LIMIT ogranicza liczbę wyświetlanych rekordów. Składnia klauzuli LIMIT jest dokładnie taka sama jak w zapytaniu SELECT.

Dane wyjściowe zapytania SHOW BINLOG EVENTS zawierają wymienione poniżej kolumny:

- Log_name. Nazwa pliku binarnego dziennika zdarzeń.
- Pos. Położenie zdarzenia w pliku binarnego dziennika zdarzeń.
- Event_type. Typ zdarzenia, na przykład Query dla zapytania, które ma zostać wykonane.
- Server_id. Identyfikator serwera, który zarejestrował zdarzenie.
- End_log_pos. Położenie następnego po zdarzeniu bajta w pliku binarnego dziennika zdarzeń.
- Info. Informacje o zdarzeniu, na przykład tekst zapytania w przypadku zdarzenia typu Query.

SHOW CHARACTER SET

SHOW CHARACTER SET [LIKE '*pattern*' | WHERE *where_expr*]

Wyświetla listę obsługiwanych przez serwer kodowań znaków. Klauzula LIKE powoduje ograniczenie danych wyjściowych do rekordów zawierających nazwy kodowań znaków dopasowane do podanego wzorca. Z kolei klauzula WHERE ogranicza dane wyjściowe do rekordów spełniających warunek zdefiniowany w podanym wyrażeniu.

Dane wyjściowe zapytania `SHOW CHARACTER SET` zawierają wymienione poniżej kolumny:

- **Charset.** Krótka nazwa kodowania znaków. Ta nazwa jest stosowana w zapytaniach SQL.
- **Description.** Dłuższa, opisowa nazwa kodowania znaków.
- **Default collation.** Nazwa domyślnej kolejności sortowania w danym kodowaniu znaków.
- **Maxlen.** To jest wyrażona w bajtach długość „najszerzego” znaku w danym kodowaniu znaków. W przypadku wielobajtowych kodowań znaków ta wartość będzie większa niż 1. Z kolei w przypadku jednobajtowych kodowań znaków każdy znak zajmuje jeden bajt, a więc wartością tej kolumny będzie 1.

SHOW COLLATION

`SHOW COLLATION [LIKE 'pattern' | WHERE where_expr]`

Wyświetla listę dostępnych dla danego kodowania znaków kolejności sortowania. Klauzula `LIKE` powoduje ograniczenie danych wyjściowych do kolejności sortowania o nazwach dopasowanych do podanego wzorca. Z kolei klauzula `WHERE` ogranicza dane wyjściowe do rekordów spełniających warunek zdefiniowany w podanym wyrażeniu.

Dane wyjściowe zapytania `SHOW COLLATION` zawierają wymienione poniżej kolumny:

- **Collation.** Nazwa kolejności sortowania.
- **Charset.** Nazwa kodowania znaków danej kolejności sortowania.
- **Id.** Identyfikator kolejności sortowania.
- **Default.** Wartość `Yes`, jeśli dana kolejność sortowania jest domyślna w kodowaniu znaków. W przeciwnym razie kolumna nie ma wartości.
- **Compiled.** Wartość `Yes`, jeśli dana kolejność sortowania została wkompilowana w serwer. W przeciwnym razie kolumna nie ma wartości.
- **Sortlen.** Współczynnik kosztu względem ilości pamięci, która musi zostać zaalokowana dla wewnętrznych operacji konwersji ciągu tekstowego, gdy dana kolejność sortowania będzie używana do sortowania wartości.

SHOW COLUMNS

`SHOW [FULL] COLUMNS {FROM | IN} tbl_name
[({FROM | IN} db_name) [LIKE 'pattern' | WHERE where_expr]`

Zapytanie wyświetla kolumny danej tabeli lub widoku. Dane wyjściowe zawierają tylko te kolumny, do których masz pewne uprawnienia. Zapytanie `SHOW FIELDS` jest synonimem dla `SHOW COLUMNS`. Użycie słowa kluczowego `FULL` powoduje, że dane wyjściowe zapytania będą zawierały kolumny `Collation`, `Privilege` i `Comment`. Klauzula `LIKE` powoduje ograniczenie danych wyjściowych do rekordów dla kolumn o nazwach dopasowanych do podanego wzorca. Z kolei klauzula `WHERE` ogranicza dane wyjściowe do rekordów spełniających warunek zdefiniowany w podanym wyrażeniu.

Aby wskazać bazę danych zawierającą tabelę, należy użyć klauzuli FROM *nazwa_bazy_danych* lub podać nazwę tabeli w formacie *nazwa_bazy_danych.nazwa_tabeli*:

```
SHOW COLUMNS FROM president;  
SHOW COLUMNS FROM president FROM sampdb;  
SHOW COLUMNS FROM sampdb.president;
```

Dane wyjściowe zapytania SHOW COLUMNS zawierają wymienione poniżej informacje o każdej tabeli kolumny. Wartości Collation, Privileges i Comment są wyświetlane jedynie po użyciu słowa kluczowego FULL:

- **Field.** Nazwa kolumny.
- **Type.** Typ danych kolumny. Po nazwie typu mogą być podane atrybuty typu.
- **Collation.** Nazwa kolejności sortowania dla kolumn typu niebinarnego ciągu tekstowego, w przypadku pozostałych kolumn to NULL. Nazwa kolejności sortowania wskazuje nazwę kodowania znaków.
- **Null.** Wartość YES, jeśli kolumna może przechowywać wartości NULL, w przeciwnym razie NO.
- **Key.** Wskazuje, czy kolumna jest zindeksowana. PRI oznacza dowolną kolumnę w PRIMARY KEY, UNI to pierwsza kolumna w indeksie typu UNIQUE, MUL to pierwsza kolumna nieunikalnego indeksu pozwalającego na istnienie wielu wystąpień danej wartości w kolumnie. Brak wartości oznacza, że kolumna nie jest zindeksowana lub jest zindeksowana, ale nie kwalifikuje się do żadnego z pozostałych desygnatorów.
- **Default.** Wartość domyślna kolumny. NULL oznacza dosłowną wartość NULL albo fakt, że definicja kolumny nie zawiera klauzuli DEFAULT.
- **Extra.** Informacje dodatkowe o kolumnie. Wartość auto_increment wskazuje kolumnę z atrybutem AUTO_INCREMENT, on update CURRENT_TIMESTAMP wskazuje kolumnę z atrybutem ON UPDATE CURRENT_TIMESTAMP. W pozostałych przypadkach kolumna nie ma wartości.
- **Privileges.** Uprawnienia użytkownika do danej kolumny.
- **Comment.** Wartość atrybutu COMMENT podanego w definicji kolumny.

SHOW CREATE

```
SHOW CREATE DATABASE [IF NOT EXISTS] db_name  
SHOW CREATE EVENT event_name  
SHOW CREATE FUNCTION func_name  
SHOW CREATE PROCEDURE proc_name  
SHOW CREATE TABLE tbl_name  
SHOW CREATE TRIGGER trigger_name  
SHOW CREATE VIEW view_name
```

Zapytania SHOW CREATE *typ_obiektu* wyświetlają zapytania CREATE *typ_obiektu* użyte do utworzenia wskazanego obiektu. Nie będziesz mieć wyświetlenia zapytania CREATE dla obiektu, o ile nie masz pewnych uprawnień do danego obiektu. Kilka zapytań może

wyświetlać także inne informacje o obiekcie, na przykład wartość, jaką miała zmienna `sql_mode` w trakcie tworzenia wskazanego obiektu.

W przypadku zapytania `SHOW CREATE DATABASE`, jeśli zapytanie zawiera klauzulę `IF NOT EXISTS`, wtedy dane wyjściowe `CREATE DATABASE` również ją zawierają.

SHOW DATABASES

`SHOW DATABASES [LIKE 'pattern' | WHERE where_expr]`

Zapytanie wyświetla bazy danych dostępne w komputerze serwera. Klauzula `LIKE` powoduje ograniczenie danych wyjściowych do rekordów dla baz danych o nazwach dopasowanych do podanego wzorca. Z kolei klauzula `WHERE` ogranicza dane wyjściowe do rekordów spełniających warunek zdefiniowany w podanym wyrażeniu.

W przypadku braku uprawnienia `SHOW DATABASES` zapytanie wyświetli jedynie te bazy danych, do których masz pewne uprawnienia dostępu. Po uruchomieniu serwera z opcją `--skip-show-database` wyświetlone będą wszystkie bazy danych, jeśli masz uprawnienie `SHOW DATABASES`, w przeciwnym razie żadna nie zostanie wyświetlona.

SHOW ENGINE

`SHOW ENGINE engine_name info_type`

Zapytanie wyświetla informacje o silnikach bazy danych. Wymagane jest posiadanie uprawnienia `PROCESS`.

- `SHOW ENGINE INNODB MUTEX`

Zapytanie wyświetla informacje o muteksach InnoDB.

- `SHOW ENGINE INNODB STATUS`

Wyświetlenie informacji o wewnętrznym działaniu silnika bazy danych InnoDB.

- `SHOW ENGINE PERFORMANCE_SCHEMA STATUS`

Wyświetlenie informacji o wewnętrznym działaniu zmiennych systemowych typu Performance Schema (zaimplementowane na poziomie silnika bazy danych).

SHOW ENGINES

`SHOW [STORAGE] ENGINES`

Zapytanie wyświetla silniki bazy danych obsługiwane przez serwer. Dla każdego silnika dane wyjściowe zawierają wymienione poniżej kolumny, wskazujące na poziom obsługi i zapewniające krótki opis cech charakterystycznych poszczególnych silników:

- `Engine`. Nazwa silnika bazy danych (InnoDB, MyISAM itd.).
- `Support`. Poziom obsługi danego silnika bazy danych. Wartość `YES` oznacza obsługiwany, `NO` oznacza nieobsługiwany, `DISABLED` oznacza obsługiwany, ale wyłączony w trakcie działania serwera, natomiast `DEFAULT` oznacza, że dany silnik jest domyślny. Domyślny silnik bazy danych zawsze jest włączony.

- **Comment.** Opisowy tekst dotyczący danego silnika bazy danych.
- **Transactions.** Wartość wskazuje, czy silnik obsługuje transakcje.
- **XA.** Wartość wskazuje, czy silnik obsługuje transakcje rozproszone.
- **Savepoints.** Wartość wskazuje, czy silnik obsługuje częściowe wycofanie transakcji.

SHOW ERRORS

```
SHOW ERRORS [LIMIT [ skip_count,] show_count]  
SHOW COUNT(*) ERRORS
```

Zapytanie `SHOW ERRORS` przypomina `SHOW WARNINGS`, ale wyświetla jedynie komunikaty, które mają wagę błędów. Zapytanie `SHOW COUNT(*) ERRORS` przypomina `SHOW COUNT(*) WARNINGS`, ale wyświetla wartość zmiennej systemowej `error_count` zamiast zmiennej `warning_count`. Więcej informacji znajdziesz w podpunkcie dotyczącym zapytania `SHOW WARNINGS`.

SHOW EVENTS

```
SHOW EVENTS [{FROM | IN} db_name]  
[LIKE 'pattern' | WHERE where_expr]
```

Zapytanie wyświetla informacje o zdarzeniach znajdujących się w domyślnej bazie danych lub wskazanej w klauzuli `FROM`. Konieczne jest posiadanie uprawnień `EVENT` do bazy danych zawierającej zdarzenia. Klauzula `LIKE` powoduje ograniczenie danych wyjściowych do rekordów dla zdarzeń o nazwach dopasowanych do podanego wzorca. Z kolei klauzula `WHERE` ogranicza dane wyjściowe do rekordów spełniających warunek zdefiniowany w podanym wyrażeniu.

SHOW FUNCTION STATUS, SHOW PROCEDURE STATUS

```
SHOW {FUNCTION | PROCEDURE} STATUS  
[LIKE 'pattern' | WHERE where_expr]
```

Zapytanie wyświetla informacje o funkcjach lub procedurach składowanych znajdujących się w domyślnej bazie danych. Klauzula `LIKE` powoduje ograniczenie danych wyjściowych do rekordów dla procedur o nazwach dopasowanych do podanego wzorca. Z kolei klauzula `WHERE` ogranicza dane wyjściowe do rekordów spełniających warunek zdefiniowany w podanym wyrażeniu.

SHOW GRANTS

```
SHOW GRANTS [FOR account]
```

Zapytanie wyświetla informacje o wskazanym koncie użytkownika. Nazwę konta użytkownika trzeba podać w formacie `'nazwa_użytkownika'@'nazwa_komputera'`, jak omówiono w podpunkcie 13.2.1.1, zatytułowanym „Określanie nazw kont”.

```
SHOW GRANTS FOR 'root'@'localhost';
SHOW GRANTS FOR ''@'cobra.example.com';
```

Możesz wykonać dowolne z poniższych zapytań w celu wyświetlenia uprawnień nadanych kontu użytkownika, za pomocą którego nawiązano połączenie z serwerem:

```
SHOW GRANTS FOR CURRENT_USER();
SHOW GRANTS FOR CURRENT_USER;
SHOW GRANTS;
```

W przypadku formatów zapytania `SHOW GRANTS` powodujących wyświetlenie uprawnień bieżącego użytkownika dane wyjściowe procedury składowanej wykonywanej w kontekście `SQL SECURITY DEFINER` odpowiadają definiującemu procedurę, a nie wywołującemu ją.

SHOW INDEX

```
SHOW {INDEX | INDEXES | KEYS} {FROM | IN} tbl_name
    [{FROM | IN} db_name] [WHERE where_expr]
```

Zapytanie wyświetla informacje o indeksach tabeli. Konieczne jest posiadanie pewnych uprawnień do kolumn znajdujących się w tabeli.

Aby wskazać bazę danych zawierającą tabelę, należy użyć klauzuli `FROM nazwa_bazy_danych` lub podać nazwę tabeli w formacie `nazwa_bazy_danych.nazwa_tabeli`:

```
SHOW INDEX FROM score;
SHOW INDEX FROM score FROM sampdb;
SHOW INDEX FROM sampdb.score;
```

Klauzula `WHERE` ogranicza dane wyjściowe do rekordów spełniających warunek zdefiniowany w podanym wyrażeniu.

Dane wyjściowe zapytania `SHOW INDEX` zawierają wymienione poniżej kolumny:

- **Table.** Nazwa tabeli zawierającej indeks.
- **Non_unique.** Wartość 1, jeśli indeks może zawierać powtarzające się wartości, 0 w przeciwnym razie.
- **Key_name.** Nazwa indeksu.
- **Seq_in_index.** Numer kolumny w indeksie. Kolumny indeksu są numerowane od 1.
- **Column_name.** Nazwa kolumny w indeksie, którego rekord został wyświetlony w danych wyjściowych.
- **Collation.** Kolejność sortowania kolumn w indeksie. Wartościami mogą być: A (rosnąco), D (malejąco) lub NULL (brak sortowania). Obecnie nie ma możliwości sortowania kluczy w kolejności malejącej.
- **Cardinality.** Przybliżona liczba unikalnych wartości w indeksie. Narzędzie `mysamchk` uaktualnia tę wartość dla tabel `MyISAM` po jego uruchomieniu wraz z opcją `--analyze`. Z kolei zapytanie `ANALYZE TABLE` powoduje uaktualnienie tej wartości dla tabel `InnoDB` i `MyISAM`. Natomiast zapytanie `OPTIMIZE TABLE` uaktualnia omawianą wartość dla tabel `MyISAM`.

- **Sub_part.** Wyrażona w bajtach długość prefiksu, o ile indeksowany jest jedynie prefiks kolumny. W przypadku indeksowania całej kolumny wyświetlona będzie tutaj wartość NULL.
- **Packed.** Wskazuje sposób kompresji kluczy. Brak kompresji kluczy oznacza wartość NULL.
- **Null.** Wartość YES oznacza, że kolumna może zawierać wartości NULL. W przeciwnym razie nie będzie tutaj wyświetlona żadna wartość.
- **Index_type.** Algorytm używany do indeksowania kolumny, na przykład BTREE, FULLTEXT lub HASH.
- **Comment.** Pole zarezerwowane dla wewnętrznych komentarzy dotyczących indeksu.
- **Index_comment.** Treść komentarza umieszczonego w definicji indeksu.

SHOW MASTER STATUS

SHOW MASTER STATUS

Zapytanie wykonywane w serwerach głównych replikacji. Dane wyjściowe zawierają wymienione poniżej kolumny wraz z informacjami o stanie binarnego dziennika zdarzeń:

- **File.** Nazwa pliku binarnego dziennika zdarzeń.
- **Position.** Bieżące położenie, w którym serwer zapisuje dane w pliku.
- **Binlog_Do_DB.** Rozdzielona przecinkami lista baz danych, które są replikowane za pomocą binarnego dziennika zdarzeń dzięki użyciu opcji `--binlog-do-db`. Jeśli nie ma żadnego tego rodzaju serwera, w tym miejscu jest wyświetlana pusta wartość.
- **Binlog_Ignore_DB.** Rozdzielona przecinkami lista baz danych, które są wykluczone z replikacji za pomocą binarnego dziennika zdarzeń dzięki użyciu opcji `--binlog-ignore-db`. Jeśli nie ma żadnego tego rodzaju serwera, w tym miejscu jest wyświetlana pusta wartość.

SHOW OPEN TABLES

```
SHOW OPEN TABLES [{FROM | IN} db_name]
[LIKE 'pattern' | WHERE where_expr]
```

Zapytanie wyświetla listę otwartych tabel innych niż tymczasowe, zarejestrowanych w buforze tabel, do których masz pewne uprawnienia. Klauzula LIKE powoduje ograniczenie danych wyjściowych do rekordów dla tabel o nazwach dopasowanych do podanego wzorca. Z kolei klauzula WHERE ogranicza dane wyjściowe do rekordów spełniających warunek zdefiniowany w podanym wyrażeniu.

Dane wyjściowe zapytania SHOW OPEN TABLES zawierają wymienione poniżej kolumny:

- **Database.** Nazwa bazy danych zawierającej daną tabelę.
- **Table.** Nazwa tabeli.

- `In_use`. Wartość wskazująca, ile razy tabela była aktualnie w użyciu.
- `Name_locked`. Wartość wskazuje, czy tabela ma nałożoną blokadę, na przykład przez zapytanie wymagające użycia tabeli bez uzyskania dostępu do jej zawartości (to może być zapytanie takie jak `RENAME TABLE`).

SHOW PLUGINS

SHOW PLUGINS

Zapytanie wyświetla informacje o zainstalowanych wtyczkach.

Dane wyjściowe zapytania `SHOW PLUGINS` zawierają wymienione poniżej kolumny:

- `Name`. Nazwa wtyczki.
- `Status`. Stan wtyczki: `ACTIVE`, `INACTIVE`, `DISABLED`, `DELETED`.
- `Type`. Typ wtyczki, na przykład `STORAGE ENGINE`.
- `Library`. Plik obiektu biblioteki wtyczki. Jeśli wtyczka jest wbudowana, wtedy wartością będzie tutaj `NULL`.
- `License`. Typ licencji, na której jest udostępniana wtyczka.

SHOW PRIVILEGES

SHOW PRIVILEGES

Zapytanie wyświetla uprawnienia, które mogą być nadane przez danego użytkownika.

Ponadto, wyświetlane są informacje o przeznaczeniu poszczególnych uprawnień.

SHOW PROCESSLIST

SHOW [FULL] PROCESSLIST

Zapytanie wyświetla informacje o bieżącej aktywności serwera. Jeżeli masz uprawnienie `PROCESS`, wtedy zapytanie wyświetla wszystkie informacje. W przeciwnym razie wyświetlane są jedynie informacje o aktywności bieżącego użytkownika.

Dane wyjściowe zapytania `SHOW PROCESSLIST` zawierają wymienione poniżej kolumny:

- `Id`. Identyfikator procesu klienta.
- `User`. Nazwa użytkownika konta powiązanego z procesem.
- `Host`. Komputer, z poziomu którego klient nawiązał połączenie.
- `db`. Domyślna baza danych dla procesu. W przypadku braku takiej bazy danych wyświetlona będzie tutaj wartość `NULL`.
- `Command`. Typ wykonywanego zapytania.
- `Time`. Wyrażony w sekundach czas, przez który proces pozostawał w jego bieżącym stanie.

- State. Informacje o działaniach podejmowanych przez MySQL podczas przetwarzania zapytania SQL.
- Info. Pierwsze 100 znaków wykonywanego zapytania. Użycie słowa kluczowego FULL powoduje wyświetlenie w tym miejscu całego zapytania.

SHOW RELAYLOG EVENTS

```
SHOW RELAYLOG EVENTS [IN 'file_name'] [FROM position]
[LIMIT [ skip_count,] show_count]
```

Zapytanie podobne do SHOW BINLOG EVENTS, ale wyświetlające zawartość dziennika przekazywania serwera podległego replikacji.

SHOW SLAVE HOSTS

```
SHOW SLAVE HOSTS
```

Zapytanie stosowane w serwerach głównych replikacji w celu wyświetlania informacji o serwerach podległych, które aktualnie są zarejestrowane. Konieczne jest posiadanie uprawnień REPLICATION SLAVE.

Dane wyjściowe zapytania SHOW SLAVE HOSTS zawierają wymienione poniżej kolumny:

- Server_id. Identyfikator serwera podległego.
- Host. Komputer serwera podległego.
- User. Nazwa użytkownika konta, którego serwer podległy używa w celu nawiązania połączenia z serwerem głównym.
- Password. Hasło do konta użytkownika, którego serwer podległy używa w celu nawiązania połączenia z serwerem głównym.
- Port. Port, na którym nasłuchuje serwer podległy. Wartością kolumny Port jest 0 (3306 w wersjach MySQL wcześniejszych niż 5.5.23), o ile serwer podległy nie został uruchomiony wraz z ustawioną zmienną systemową report_port.
- Master_id. Identyfikator serwera głównego.
- Slave_UUID. Identyfikator UUID serwera podległego. Ta kolumna została wprowadzona w wydaniu MySQL 5.6.0.

SHOW SLAVE STATUS

```
SHOW SLAVE STATUS
```

Zapytanie używane w serwerach podległych, powoduje wyświetlenie informacji o stanie replikacji w danym serwerze. Wiele z wyświetlanych wartości odzwierciedla parametry podane w zapytaniu CHANGE MASTER. Dane wyjściowe zapytania SHOW SLAVE STATUS zawierają wymienione poniżej kolumny:

- **Slave_IO_State.** Stan wątku wejścia-wyjścia serwera podległego. Ta wartość jest taka sama jak wartość State wyświetlana przez zapytanie `SHOW PROCESSLIST` dla wątku wejścia-wyjścia.
- **Master_Host.** Komputer serwera głównego, z którym połączony jest serwer podległy.
- **Master_User.** Nazwa użytkownika konta, którego serwer podległy używa w celu nawiązania połączenia z serwerem głównym.
- **Master_Port.** Numer portu używany podczas nawiązania połączenia z serwerem głównym.
- **Connect_Retry.** Wyrażony w sekundach czas oczekiwania między kolejnymi próbami nawiązania połączenia z serwerem głównym.
- **Master_Log_File.** Nazwa bieżącego pliku binarnego dziennika zdarzeń w serwerze głównym.
- **Read_Master_Log_Pos.** Bieżące położenie wątku wejścia-wyjścia serwera podległego w binarnym dzienniku zdarzeń serwera głównego, z którego odczytuje on dane.
- **Relay_Log_File.** Nazwa bieżącego pliku dziennika przekazywania.
- **Relay_Log_Pos.** Bieżące położenie wątku SQL serwera podległego w pliku dziennika przekazywania.
- **Relay_Master_Log_File.** Nazwa pliku binarnego dziennika zdarzeń w serwerze głównym zawierającego ostatnie zdarzenie wykonane przez wątek SQL.
- **Slave_IO_Running.** Wartość wskazuje, czy wątek wejścia-wyjścia serwera podległego działa i jest połączony z serwerem głównym.
- **Slave_SQL_Running.** Wartość wskazuje, czy wątek SQL serwera podległego działa.
- **Replicate_Do_DB, Replicate_Ignore_DB, Replicate_Do_Table, Replicate_Ignore_Table, Replicate_Wild_Do_Table, Replicate_Wild_Ignore_Table.** Rozdzielona przecinkami lista baz danych lub tabel, dla których wyraźnie włączono lub wyłączyło replikację za pomocą opcji `--replicate-do-db`, `--replicate-ignore-db`, `--replicate-do-table`, `--replicate-ignore-table`, `--replicate-wild-do-table` i `--replicate-wild-ignore-table`. W przypadku braku tego rodzaju obiektów w tym miejscu nie jest wyświetlana żadna wartość.
- **Last_Errno, Last_Error.** Wymienione kolumny są aliasami dla `Last_SQL_Errno` i `Last_SQL_Error`.
- **Skip_Counter.** Aktualna liczba zdarzeń z serwera głównego, które serwer podległy powinien pominąć. (Aby pomijać zdarzenia w serwerze podległym, konieczne jest ustawienie w nim globalnej zmiennej systemowej `sql_slave_skip_counter`).
- **Exec_Master_Log_Pos.** Aktualne położenie w pliku binarnego dziennika zdarzeń serwera głównego, który jest przetwarzany przez wątek SQL serwera podległego.

- **Relay_Log_Space.** Łączna wielkość plików dziennika przekazywania.
- **Until_Condition.** Warunek zdefiniowany w klauzuli UNTIL zapytania START SLAVE, wskazujący, kiedy wątek SQL powinien wstrzymać odczytywanie i wykonywanie zdarzeń.
 - ◆ **None.** Brak zdefiniowanej klauzuli UNTIL.
 - ◆ **Master.** Serwer podległy będzie odczytywał zdarzenia aż do chwili, gdy wątek SQL dotrze do wskazanego położenia w pliku binarnego dziennika zdarzeń serwera głównego.
 - ◆ **Relay.** Serwer podległy będzie odczytywał zdarzenia aż do chwili, gdy wątek SQL dotrze do wskazanego położenia w pliku przekazywania serwera podległego.

Jeżeli wartością `Until_Condition` jest `Master` lub `Relay`, wtedy kolumny `Until_Log_File` i `Until_Log_Pos` będą wskazywały nazwę pliku i położenie, w którym wątek SQL wstrzymał działanie.

- **Until_Log_File, Until_Log_Pos.** Patrz opis `Until_Condition`.
- **Master_SSL_Allowed.** Wartość wskazuje, czy protokół SSL jest używany podczas nawiązywania połączenia z serwerem głównym. `YES` oznacza możliwość użycia SSL, `NO` oznacza brak takiej możliwości, `Ignored` oznacza, że połączenia SSL są dozwolone, ale serwer nie został skompilowany wraz z obsługą SSL.
- **Master_SSL_CA_File, Master_SSL_CA_Path, Master_SSL_Cert, Master_SSL_Cipher, Master_SSL_Key, Master_SSL_Verify_Server_Cert, Master_SSL_Crl, Master_SSL_Crlpath.** To są parametry SSL użyte podczas nawiązywania połączenia z serwerem głównym. Parametry `Master_SSL_Crl` i `Master_SSL_Crlpath` zostały wprowadzone w wersji MySQL 5.6.3.
- **Seconds_Behind_Master.** Wyrażona w sekundach różnica między aktualną godziną i znacznikiem czasu zapisywanym w zdarzeniu serwera głównego ostatnio wykonanego przez wątek SQL serwera podległego. Ta wartość wynosi zero, jeśli wątek SQL nadąży za wątkiem wejścia-wyjścia i obecnie znajduje się w stanie bezczynności. Jeżeli nie zostało wykonane żadne zdarzenie lub parametry serwera podległego zostały zmienione za pomocą zapytania `CHANGE MASTER` lub `RESET SLAVE`, wtedy wartością wyświetlaną przez tę kolumnę jest `NULL`.
- **Last_IO_Errno, Last_IO_Error.** Numer i komunikat ostatniego błędu wątku wejścia-wyjścia. Wartościami będą 0 i pusty ciąg tekstowy, jeśli jeszcze nie wystąpił żaden błąd. Niepuste komunikaty błędów są przez serwer zapisywane w dzienniku błędów.
- **Last_SQL_Errno, Last_SQL_Error.** Podobnie jak `Last_IO_Errno` i `Last_IO_Error`, ale dotyczą wątku SQL.
- **Replicate_Ignore_Server_Ids.** Identyfikatory serwerów, z których zdarzenia mają być ignorowane.

- **Master_Server_Id.** Wartość `server_id` serwera głównego.
- **Master_UUID.** Wartość `server_uuid` serwera głównego. Ta kolumna została wprowadzona w MySQL 5.6.1.
- **Master_Info_File.** Nazwa pliku *master.info* w serwerze podległym. Ta kolumna została wprowadzona w MySQL 5.6.1.
- **SQL_Delay.** Wyrażone w sekundach opóźnienie replikacji. Ta kolumna została wprowadzona w MySQL 5.6.1.
- **Slave_Remaining_Delay.** Wyrażone w sekundach opóźnienie serwera podległego oczekującego na wykonanie kolejnego zdarzenia. Jeżeli serwer nie czeka na kolejne zdarzenie, wartością jest NULL. Ta kolumna została wprowadzona w MySQL 5.6.1.
- **Slave_SQL_Running_State.** Stan wątku SQL serwera podległego. Ta wartość jest dokładnie taka sama jak wartość `State` wyświetlana w danych wyjściowych zapytania `SHOW PROCESSLIST` dla wątku SQL. Ta kolumna została wprowadzona w MySQL 5.6.1.
- **Master_Retry_Count.** Wartość wskazująca liczbę prób połączenia z serwerem głównym, jakie powinien podjąć serwer podległy, zanim zgłosi niepowodzenie operacji. Ta kolumna została wprowadzona w MySQL 5.6.1.
- **Master_Bind.** Interfejs sieciowy używany przez serwer podległy. Ta kolumna została wprowadzona w MySQL 5.6.1.
- **Last_IO_Error_Timestamp.** Data i godzina wystąpienia ostatniego błędu wątku wejścia-wyjścia. Ta kolumna została wprowadzona w MySQL 5.6.2. W wersjach wcześniejszych niż 5.6.2 znacznik czasu był częścią wartości `Last_IO_Error`.
- **Last_SQL_Error_Timestamp.** Data i godzina wystąpienia ostatniego błędu wątku SQL. Ta kolumna została wprowadzona w MySQL 5.6.2. W wersjach wcześniejszych niż 5.6.2 znacznik czasu był częścią wartości `Last_SQL_Error`.

SHOW STATUS

```
SHOW [GLOBAL | SESSION ] STATUS
[LIKE 'pattern' | WHERE where_expr]
```

Zapytanie wyświetla zmienne stanu serwera oraz ich wartości. Wspomniane zmienne dostarczają informacje o działaniu serwera. Klauzula `LIKE` powoduje ograniczenie danych wyjściowych do rekordów dla zmiennych o nazwach dopasowanych do podanego wzorca. Z kolei klauzula `WHERE` ogranicza dane wyjściowe do rekordów spełniających warunek zdefiniowany w podanym wyrażeniu.

Serwer może wyświetlać wartości zmiennych stanu o zasięgu globalnym (dla całego serwera) lub sesji (dla danego klienta). Przedstawiają one wartości odpowiednio dla wszystkich klientów i bieżącego klienta. Domyślnie zapytanie `SHOW` wyświetla wartość sesji dla wskazanej zmiennej. W celu wyświetlenia wartości globalnej lub wyraźnego wskazania, że ma być wyświetlona wartość sesji, należy użyć specyfikatora zakresu:

```
SHOW GLOBAL STATUS;  
SHOW SESSION STATUS;
```

Jeżeli zmienna ma jedynie wartość globalną, użycie specyfikatorów GLOBAL i SESSION powoduje wyświetlenie tej samej wartości. Słowo kluczowe LOCAL jest synonimem dla słowa kluczowego SESSION.

Informacje o stanie można również pobrać z tabel GLOBAL_STATUS i SESSION_STATUS bazy danych INFORMATION_SCHEMA.

Więcej informacji na temat zmiennych stanu znajdziesz w punkcie 12.3.2, zatytułowanym „Sprawdzenie wartości zmiennej stanu”. Opis poszczególnych zmiennych stanu znajdziesz w dodatku D, zatytułowanym „Przewodnik po zmiennych systemowych, stanu i użytkownika”.

SHOW TABLE STATUS

```
SHOW TABLE STATUS [{FROM | IN} db_name]  
[LIKE 'pattern' | WHERE where_expr]
```

Zapytanie wyświetla opisowe informacje o znajdujących się w bazie danych tabelach innych niż tymczasowe, do których masz pewne uprawnienia. To zapytanie wyświetla również widoki w bazie danych, ale wszystkie kolumny mają wartość NULL, za wyjątkiem kolumny Name, wyświetlającej nazwę widoku, i Comment, wyświetlającej wartość view. Klauzula LIKE powoduje ograniczenie danych wyjściowych do rekordów dla tabel o nazwach dopasowanych do podanego wzorca. Z kolei klauzula WHERE ogranicza dane wyjściowe do rekordów spełniających warunek zdefiniowany w podanym wyrażeniu.

Dane wyjściowe zapytania SHOW TABLE STATUS zawierają wymienione poniżej kolumny:

- Name. Nazwa tabeli.
 - Engine. Nazwa silnika bazy danych (InnoDB, MyISAM itd.).
 - Version. Wersja pliku *.frm* tabeli.
 - Row_format. Format rekordu. Ta wartość pochodzi z opcji ROW_FORMAT podanej w zapytaniu CREATE TABLE.
 - Rows. Liczba rekordów w tabeli. W przypadku niektórych silników bazy danych, takich jak InnoDB, to będzie wartość przybliżona.
 - Avg_row_length. Średnia liczba bajtów używanych przez rekordy tabeli.
 - Data_length. Wyrażona w bajtach rzeczywista wielkość pliku danych tabeli.
 - Max_data_length. Maksymalna wielkość, do jakiej może zostać zwiększony plik danych tabeli.
 - Index_length. Wyrażona w bajtach rzeczywista wielkość pliku indeksu tabeli.
 - Data_free. Liczba niewykorzystanych bajtów w pliku danych. Jeżeli ta liczba jest bardzo duża, wtedy dobrym rozwiązaniem będzie wykonanie zapytania OPTIMIZE TABLE w celu przeprowadzenia operacji defragmentacji tabeli.
- W przypadku tabel InnoDB ta kolumna wskazuje ilość wolnego miejsca

w przestrzeni tabel InnoDB, w której znajduje się dana tabela. (To może być systemowa bądź oddzielna przestrzeń tabel).

- **Auto_increment.** Następną wartość AUTO_INCREMENT, która zostanie wygenerowana dla tabeli.
- **Create_time.** Data i godzina utworzenia tabeli.
- **Update_time.** Data i godzina ostatniej modyfikacji tabeli. Jeżeli silnik bazy nie przechowuje takich danych, wtedy wartością tej kolumny będzie NULL.
- **Check_time.** Data i godzina ostatniej operacji sprawdzenia lub naprawy tabeli. Jeżeli silnik bazy nie przechowuje takich danych bądź też tabela nigdy nie była sprawdzana lub naprawiana, wtedy wartością tej kolumny będzie NULL.
- **Collation.** Kolejność sortowania w tabeli. Nazwa kolejności sortowania wskazuje nazwę używanego kodowania znaków.
- **Checksum.** Suma kontrolna tabeli; jeśli nie może być obliczona, wartością tej kolumny będzie NULL.
- **Create_options.** Opcje dodatkowe podane jako wartości *table_option* w zapytaniu CREATE TABLE użytym do utworzenia tabeli lub w kolejnych zapytaniach ALTER TABLE.
- **Comment.** Tekst dowolnego komentarza podanego w trakcie tworzenia tabeli.

SHOW TABLES

```
SHOW [FULL] TABLES [{FROM | IN} db_name]
[LIKE 'pattern' | WHERE where_expr]
```

Zapytanie wyświetla nazwy znajdujących się w bazie danych tabel innych niż tymczasowe, do których masz pewne uprawnienia. To zapytanie wyświetla również nazwy widoków. Istnieje możliwość podania słowa kluczowego FULL w celu wyświetlenia kolumny wskazującej, czy nazwa rekordu odwołuje się do tabeli (BASE_TABLE), czy do widoku (VIEW). Klauzula LIKE powoduje ograniczenie danych wyjściowych do rekordów dla tabel o nazwach dopasowanych do podanego wzorca. Z kolei klauzula WHERE ogranicza dane wyjściowe do rekordów spełniających warunek zdefiniowany w podanym wyrażeniu.

SHOW TRIGGERS

```
SHOW TRIGGERS [{FROM | IN} db_name]
[LIKE 'pattern' | WHERE where_expr]
```

Zapytanie wyświetla informacje o wyzwalaczach znajdujących się w domyślnej bazie danych lub wskazanej w klauzuli FROM. Dane wyjściowe zawierają jedynie wyniki dla tabel, dla których masz uprawnienie TRIGGER. Klauzula LIKE powoduje ograniczenie danych wyjściowych do rekordów dla tabel o nazwach dopasowanych do podanego wzorca. Z kolei klauzula WHERE ogranicza dane wyjściowe do rekordów spełniających warunek zdefiniowany w podanym wyrażeniu.

SHOW VARIABLES

```
SHOW [GLOBAL | SESSION ] VARIABLES  
[LIKE 'pattern' | WHERE where_expr]
```

Zapytanie wyświetla listę zmiennych systemowych oraz ich wartości. Wspomniane zmienne dostarczają informacje o konfiguracji i możliwościach serwera. Klauzula LIKE powoduje ograniczenie danych wyjściowych do rekordów dla zmiennych o nazwach dopasowanych do podanego wzorca. Z kolei klauzula WHERE ogranicza dane wyjściowe do rekordów spełniających warunek zdefiniowany w podanym wyrażeniu.

Serwer może wyświetlać wartości zmiennych systemowych o zasięgu globalnym (dla całego serwera) lub sesji (dla danego klienta). Domyślnie zapytanie SHOW VARIABLES wyświetla wartość sesji dla wskazanej zmiennej. (Wersje wcześniejsze niż MySQL 5.5.3 wyświetlały wartość sesji, w przypadku jej braku wyświetlana była wartość globalna). W celu wyświetlenia wartości globalnej lub wyraźnego wskazania, że ma być wyświetlona wartość sesji, należy użyć specyfikatora zakresu:

```
SHOW GLOBAL VARIABLES;  
SHOW SESSION VARIABLES;
```

Słowo kluczowe LOCAL jest synonimem dla słowa kluczowego SESSION. Istnieje również możliwość pobrania wartości poszczególnych zmiennych dynamicznych za pomocą zapytania SELECT:

```
SELECT @@GLOBAL.sql_mode, @@SESSION.sql_mode, @@LOCAL.sql_mode;
```

Użycie zapytania SELECT ma tę zaletę, że pozwala na znacznie łatwiejsze operowanie wynikiem zapytania w określonych kontekstach.

Informacje o zmiennych systemowych można również pobrać z tabel GLOBAL_VARIABLES i SESSION_VARIABLES bazy danych INFORMATION_SCHEMA.

Więcej informacji na temat użycia zmiennych systemowych znajdziesz w punkcie 12.3.1, zatytułowanym „Sprawdzanie i ustawienie wartości zmiennych systemowych”. Opis poszczególnych zmiennych stanu znajdziesz w dodatku D, zatytułowanym „Przewodnik po zmiennych systemowych, stanu i użytkownika”.

SHOW WARNINGS

```
SHOW WARNINGS [LIMIT [skip_count,] show_count]  
SHOW COUNT(*) WARNINGS
```

Zapytanie SHOW WARNINGS powoduje wyświetlenie błędów, ostrzeżeń i informacji wygenerowanych przez ostatnie zapytanie tworzących tego rodzaju komunikaty. Jeżeli wykonanie zapytania zakończy się powodzeniem, wówczas wartością zwrótną SHOW WARNINGS jest pusty zbiór wyników.

Zapytanie SHOW COUNT(*) WARNINGS wyświetla wartość zmiennej systemowej warning_count, która przechowuje liczbę wygenerowanych komunikatów. (Powiązana z nią zmienna o nazwie error_count przechowuje jedynie liczbę błędów). Istnieje możliwość, że wartość zmiennej warning_count będzie większa niż liczba komunikatów wyświetlanych

przez zapytanie `SHOW WARNINGS`. Zmienna systemowa `max_error_count` ogranicza liczbę komunikatów, które mogą być przechowywane do wyświetlenia przez zapytanie `SHOW WARNINGS`. Natomiast zmienna `warning_count` przechowuje liczbę wszystkich wygenerowanych komunikatów, niezależnie od tego, czy są one przechowywane.

Jeżeli zostanie użyta klauzula `LIMIT`, powoduje ona ograniczenie liczby wyświetlanych rekordów. Składnia klauzuli `LIMIT` jest dokładnie taka sama jak w zapytaniu `SELECT`.

START SLAVE

```
START SLAVE [thread_option [, thread_option] ...]
START SLAVE [SQL_THREAD] UNTIL
    MASTER_LOG_FILE = 'file_name', MASTER_LOG_POS = position
START SLAVE [SQL_THREAD] UNTIL
    RELAY_LOG_FILE = 'file_name', RELAY_LOG_POS = position
START SLAVE [USER = 'user_name'] [PASSWORD = 'pass_val']
    [DEFAULT_AUTH = 'auth_plugin'] [PLUGIN_DIR = 'dir_name']
```

To zapytanie w połączeniu ze `STOP SLAVE` nadzoruje działanie wątków replikacji w serwerze podległym. Wykonanie zapytania `START SLAVE` bez opcji powoduje inicjalizację wątków wejścia-wyjścia i SQL, natomiast zapytania `STOP SLAVE` zamyka oba wymienione wątki. Wartość *thread_option* pozwala na wskazanie uruchamianego lub zamykanego wątku:

- **IO_THREAD.** Uruchomienie lub zatrzymanie wątku wejścia-wyjścia odczytującego zdarzenia z serwera głównego i przechowującego je w dzienniku przekazywania.
- **SQL_THREAD.** Uruchomienie lub zatrzymanie wątku SQL odczytującego zdarzenia z dziennika przekazywania i wykonującego je.

Jeżeli nie zostanie podany żaden wątek lub podana zostanie wartość `SQL_THREAD`, wtedy można użyć klauzuli `UNTIL`. W zależności od podanej w klauzuli pary opcji (plik dziennika zdarzeń i położenie) serwer podległy będzie działał, dopóki wątek SQL nie dotrze do wskazanego położenia w binarnym dzienniku zdarzeń serwera głównego lub dziennika przekazywania serwera podległego. Jeżeli wątek SQL już działa, wtedy serwer ignoruje klauzulę `UNTIL` i generuje komunikat ostrzeżenia. Z kolei, jeśli klauzula zawiera opcję `SQL_THREAD`, serwer uruchomi jedynie wątek SQL. W przeciwnym razie zostaną uruchomione oba wątki.

Począwszy od MySQL 5.6.4, zapytanie `START SLAVE` może zawierać parametry pozwalające na użycie wtyczki uwierzytelniania. Składnia dopuszcza wykorzystanie wymienionych poniżej opcji, które muszą być użyte w podanej kolejności:

- **USER.** Nazwa użytkownika. To nie może być wartość `NULL` lub pusty ciąg tekstowy. Nazwa użytkownika jest wymagana w przypadku użycia opcji `PASSWORD`.
- **PASSWORD.** Hasło użytkownika.
- **DEFAULT_AUTH.** Nazwa wtyczki uwierzytelniania. Jeżeli zostanie pominięta, wtedy będzie użyte domyślne w MySQL uwierzytelnianie za pomocą hasła.
- **PLUGIN_DIR.** Nazwa katalogu zawierającego wtyczkę uwierzytelniania.

START TRANSACTION

```
START TRANSACTION
    [trans_characteristic [, trans_characteristic ] ...]
trans_characteristic:
    WITH CONSISTENT SNAPSHOT
    | READ WRITE
    | READ ONLY
```

Zapytanie rozpoczyna transakcję przez wyłączenie trybu automatycznego zatwierdzania aż do chwili wykonania zapytania COMMIT lub ROLLBACK. Zapytania wykonywane przy wyłączonym trybie automatycznego zatwierdzania będą zatwierdzone lub wycofane w postaci pojedynczej jednostki.

Po zatwierdzeniu lub wycofaniu transakcji następuje przywrócenie stanu trybu automatycznego zatwierdzania, jaki obowiązywał przed wykonaniem zapytania START TRANSACTION. W celu jasnego operowania trybem automatycznego zatwierdzania należy ustawić wartość zmiennej systemowej autocommit zgodnie z opisem przedstawionym w punkcie 2.12.1, zatytułowanym „Użycie transakcji do zapewnienia bezpiecznego środowiska wykonywania”.

Jeżeli zostanie użyta opcja WITH CONSISTENT SNAPSHOT, powoduje rozpoczęcie transakcji jako jednolitej operacji odczytu. W przypadku silnika InnoDB wymieniona klauzula nie powoduje zmiany aktualnego poziomu izolacji, więc efekt ma jedynie w przypadku poziomu REPEATABLE READ lub SERIALIZABLE.

Wartości READ WRITE i READ ONLY określają tryb dostępu do transakcji. Innymi słowy, wskazują, czy transakcja zezwala na modyfikacje tabel. (Tabele tymczasowe mogą być modyfikowane niezależnie od trybu dostępu). Wymienione wartości są wzajemnie wykluczające się i dlatego nie mogą być jednocześnie używane w tym samym zapytaniu. Wprowadzono je w wydaniu MySQL 5.6.5.

Zapytanie START TRANSACTION niejawnie znosi wszelkie blokady tabel nałożone przez klienta za pomocą zapytania LOCK TABLE, ale jeszcze niezniesione. Wykonanie zapytania START TRANSACTION w trakcie trwania innej transakcji powoduje jej niejawne zatwierdzenie.

STOP SLAVE

```
STOP SLAVE [thread_option [, thread_option] ...]
```

To zapytanie w połączeniu z START SLAVE nadzoruje działanie wątków replikacji w serwerze podległym. Informacje szczegółowe na temat dozwolonych wartości opcji thread_option znajdziesz w podpunkcie poświęconym zapytaniu START SLAVE.

TRUNCATE TABLE

```
TRUNCATE [TABLE] tbl_name
```

Zapytanie TRUNCATE TABLE przeprowadza operację szybkiego opróżnienia zawartości tabeli przez jej usunięcie i ponowne utworzenie. To jest znacznie szybsze rozwiązanie niż usuwanie poszczególnych rekordów. Konieczne jest posiadanie uprawnień DROP do wskazanej tabeli.

W przypadku silnika bazy danych InnoDB omawiane zapytanie jest niedozwolone i zwraca błąd, jeśli tabela zawiera jakiegokolwiek odwołania do kluczy zewnętrznych.

Zapytanie `TRUNCATE TABLE` nie jest bezpieczne dla transakcji i jego wykonanie w ramach transakcji lub w sytuacji, gdy nałożona została blokada na tabelę, spowoduje wygenerowanie błędu.

UNINSTALL PLUGIN

```
UNINSTALL PLUGIN plugin_name
```

Zapytanie powoduje odinstalowanie i wyrejestrowanie wskazanej wtyczki z tabeli `mysql.plugin` (o ile informacje o wtyczce się w niej znajdują). Dzięki temu wtyczka nie będzie wczytywana w trakcie kolejnych uruchomień serwera. Omawiane zapytanie wymaga uprawnień `DELETE` do tabeli `mysql.plugin`. Więcej informacji znajdziesz w podrozdziale 12.4, zatytułowanym „Interfejs wtyczek”.

UNION

```
select_stmt
UNION [DISTINCT | ALL] select_stmt
[UNION [DISTINCT | ALL] select_stmt] ...
[ORDER BY col_list] [LIMIT [skip_count,] show_count]
```

Zapytanie `UNION` powoduje połączenie wyników wykonania wielu zapytań `SELECT`. Każde z wspomnianych zapytań `SELECT` musi generować tę samą liczbę kolumn w zbiorze wynikowym. Nazwy kolumn z pierwszego zapytania `SELECT` będą nazwami kolumn w ostatecznym zbiorze wynikowym. Typ danych kolumn jest określany przez uwzględnienie wszystkich wartości w odpowiadających sobie kolumnach wskazanych tabel.

Po słowie kluczowym `UNION` można podać `DISTINCT` w celu usunięcia powtarzających się rekordów lub `ALL` w celu zachowania duplikatów i zwrotu wszystkich rekordów. Domyślnie, w przypadku braku któregośkolwiek z wymienionych słów kluczowych następuje usunięcie duplikatów. Każda operacja typu `DISTINCT` (zarówno jawna, jak i niejawna) ma pierwszeństwo przez każdą operacją `ALL` po jej lewej stronie:

```
mysql> SELECT 1 UNION ALL SELECT 2 UNION ALL SELECT 1;
+---+
| 1 |
+---+
| 1 |
| 2 |
| 1 |
+---+
mysql> SELECT 1 UNION ALL SELECT 2 UNION SELECT 1;
+---+
| 1 |
+---+
| 1 |
| 2 |
+---+
```

W celu użycia klauzul `ORDER BY` i `LIMIT` wraz z poszczególnymi zapytaniami `SELECT` należy każde zapytanie `SELECT` ująć w nawias. (Klauzule `ORDER BY` w poszczególnych zapytaniu `SELECT` są używane jedynie wtedy, gdy obecna jest również klauzula `LIMIT`, co ma na celu ustalenie rekordów, do których zastosowanie ma klauzula `LIMIT`. Nie ma wpływu na kolejność rekordów pojawiających się w ostatecznym wyniku wykonania zapytania `UNION`). W celu zastosowania klauzuli `ORDER BY` lub `LIMIT` na zapytaniu `UNION` jako całości należy poszczególne zapytania `SELECT` ująć w nawiasy i dodać klauzulę `ORDER BY` lub `LIMIT` po ostatnim nawiasie zamykającym. W takim przypadku wszelkie kolumny wymienione w klauzuli `ORDER BY` powinny odwoływać się do nazw kolumn użytych w pierwszym zapytaniu `SELECT`.

UNLOCK TABLE

```
UNLOCK {TABLE | TABLES}
```

To zapytanie powoduje zniesienie wszelkich blokad tabeli nałożonych przez bieżącego klienta. Serwer automatycznie znosi blokady nałożone przez klienta, jeśli klient rozpocznie transakcję lub zakończy sesję.

UPDATE

```
UPDATE [LOW_PRIORITY] [IGNORE]
tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    SET col_name=expr [, col_name=expr ] ...
    [WHERE where_expr] [ORDER BY ... ] [LIMIT n]
UPDATE [LOW_PRIORITY] [IGNORE] tbl_refs
    SET col_name=expr [, col_name=expr ] ...
    [WHERE where_expr] [ORDER BY ... ] [LIMIT n]
```

W pierwszej składni zapytanie `UPDATE` powoduje modyfikację zawartości istniejących rekordów w podanej tabeli (*tbl_name*). Począwszy od wersji MySQL 5.6.2, zapytanie `UPDATE` obsługuje klauzulę `PARTITION` dla tabel partycjonowanych, co pozwala na wskazanie partycji lub podpartycji, w której mają być uaktualnione rekordy. W takim przypadku, jeśli rekord przeznaczony do uaktualnienia nie znajduje się we wskazanej partycji, wtedy pozostanie nienaruszony.

Druga składnia zapytania `UPDATE` jest podobna do pierwszej, ale pozwala na wskazanie wielu nazw tabel i przeprowadzenie operacji uaktualnienia wielu tabel. Składnia *tbl_refs* jest taka sama jak w zapytaniu `SELECT` (łącznie z obsługą klauzuli `PARTITION` po nazwie każdej tabeli), za wyjątkiem faktu, że nie można użyć podzapytania jako nazwy tabeli.

Rekordy przeznaczone do uaktualnienia są wybierane przez wyrażenie zdefiniowane w klauzuli `WHERE`. Dla wybranych rekordów każda kolumna wymieniona w klauzuli `SET` będzie miała wartość odpowiadającą jej wyrażeniu.

```
UPDATE member SET expiration = NULL, phone = '197-602-4832'
WHERE member_id = 14;
```

Klauzula `WHERE` może zawierać podzapytania, ale nie ma możliwości pobierania rekordów z uaktualnianej tabeli.

W przypadku pominięcia klauzuli *WHERE* *uaktualnione zostaną wszystkie rekordy w tabeli*.

Domyślnie zapytanie *UPDATE* zwraca liczbę uaktualnionych rekordów. Jednak rekord nie jest uznawany za uaktualniony, o ile faktycznie nie nastąpi zmiana wartości pewnej kolumny. Przypisanie kolumnie jej aktualnej wartości nie jest uważane za modyfikację rekordu. Jeżeli aplikacja rzeczywiście wymaga, aby zapytanie *UPDATE* zwróciło liczbę rekordów dopasowanych do klauzuli *WHERE* niezależnie od tego, czy nastąpi faktyczna zmiana ich wartości, wtedy należy podać flagę *CLIENT_FOUND_ROWS* podczas nawiązywania połączenia z serwerem.

Opcja *LOW_PRIORITY* powoduje, że wykonanie zapytania zostanie opóźnione aż do chwili, gdy inne klienty nie będą odczytywały danych z tabeli. Ta opcja jest efektywna jedynie w przypadku silników bazy danych stosujących nakładanie blokad na poziomie tabeli, na przykład *MyISAM* lub *MEMORY*.

Jeżeli uaktualnienie rekordu spowoduje powielenie wartości klucza unikalnego indeksu, wtedy wykonywanie zapytania *UPDATE* zostanie przerwane, nastąpi wygenerowanie błędu i więcej żaden rekord nie będzie uaktualniony. Dodanie słowa kluczowego *IGNORE* powoduje, że wspomniane rekordy nie zostaną uaktualnione i nie będzie wygenerowany błąd. W trybie ścisłego SQL słowo kluczowe *IGNORE* powoduje, że błędy w trakcie konwersji danych lub inne błędy powodujące przerwanie wykonywania zapytania będą traktowane jako ostrzeżenia, które nie mają znaczenia krytycznego. W takim przypadku kolumnom zostaną przypisane najbliższe prawidłowe wartości.

Klauzula *ORDER BY* powoduje uaktualnianie rekordów zgodnie z kolejnością sortowania wyniku. Ta klauzula ma taką samą składnię jak w przypadku zapytania *SELECT*.

Jeżeli zostanie podana klauzula *LIMIT*, wtedy wartość *n* oznacza maksymalną liczbę uaktualnianych rekordów.

W przypadku uaktualniania wielu tabel za pomocą zapytania *UPDATE*, klauzula *WHERE* może wskazywać warunki, na podstawie których tabele będą łączone, a klauzula *SET* może uaktualnić kolumny w wielu tabelach. Na przykład, poniższe zapytanie spowoduje uaktualnienie rekordów w tabeli *t1* mających wartości *id* dopasowane do odpowiadających im wartości w *t2* i skopiowanie wartości *quantity* z *t2* do *t1*:

```
UPDATE t INNER JOIN t2 SET t.quantity = t2.quantity WHERE t.id = t2.id;
```

USE

USE db_name

Zapytanie powoduje wybranie wskazanej bazy danych jako domyślnej (baza danych dla tabel, widoków i procedur składowanych, gdy nie będzie wyraźnie podana nazwa bazy danych). Jeżeli wykonanie zapytania *USE* zakończy się powodzeniem, serwer przypisze w sesji zmiennym systemowym *character_set_database* i *collation_database* kodowanie znaków i kolejność sortowania używane w bazie danych.

Wykonanie zapytania *USE* kończy się niepowodzeniem, jeśli wskazana baza danych nie istnieje lub jeśli użytkownik nie ma do niej uprawnień.

E.2. Składnia zapytań SQL (zapytania złożone)

W tym podrozdziale zostanie przedstawiona składnia zapytań stosowanych w zapytaniach złożonych, które są tworzone za pomocą słów kluczowych BEGIN i END, a także używanych do tworzenia programów składowanych: funkcji, procedur, wyzwalaczy i zdarzeń przechowywanych po stronie serwera.

Każde zapytanie w części głównej programu musi być zakończone średnikiem. Jeżeli używasz klienta mysql do tworzenia procedury składowanej zawierającej wiele zapytań, powinieneś tymczasowo zmienić ogranicznik zapytania w mysql, aby wymieniony klient nie interpretował znaków średnika. Do tego celu służy zapytanie DELIMITER. Upewnij się, że wybrałeś ogranicznik niewystępujący w zapytaniach tworzących daną procedurę. Więcej informacji na temat definiowania programów składowanych i ich przykłady znajdziesz w punkcie 4.2.1, zatytułowanym „Zapytania złożone i ograniczniki zapytań”.

E.2.1. Polecenia struktury kontrolnej

Zapytania omówione w tym punkcie tworzą bloki i zapewniają konstrukcje przebiegu działania programu. Każde wystąpienie *stmt_list* w składni zapytań wskazuje na listę jednego lub więcej zapytań, z których każde jest zakończone średnikiem.

Pewne konstrukcje mogą być oznaczone etykietami (BEGIN, LOOP, REPEAT i WHILE). Wspomniane etykiety nie rozróżniają wielkości liter i muszą spełniać wymienione poniżej kryteria:

- Jeżeli etykieta znajduje się na początku konstrukcji, wtedy etykieta o takiej samej nazwie może pojawiać się także na końcu konstrukcji.
- Etykieta nie może pojawić się na końcu konstrukcji, jeśli odpowiadająca jej etykieta nie znajduje się na początku konstrukcji.

BEGIN ... END

```
BEGIN [stmt_list] END  
label: BEGIN [stmt_list] END [label]
```

Konstrukcja BEGIN ... END tworzy blok pozwalający na grupowanie wielu zapytań. Jeżeli część główna programu składowanego zawiera więcej niż tylko jedno zapytanie, muszą być one grupowane w ramach bloku BEGIN. Ponadto, jeśli blok BEGIN zawiera jakiegokolwiek zapytania DECLARE, muszą się one znajdować na początku bloku.

CASE

```
CASE [expr]  
  WHEN expr1 THEN stmt_list1  
  [WHEN expr2 THEN stmt_list2] ...  
  [ELSE stmt_list]  
END IF
```

Zapytanie CASE pozwala na utworzenie konstrukcji rozgałęziającej przebieg działania programu. To zapytanie różni się od operatora CASE, omówionego w punkcie C.1.4, zatytułowanym „Operatory porównania”.

Jeżeli podano wyrażenie początkowe (*expr*), wtedy CASE porównuje je z wyrażeniem znajdującym się po każdym słowie kluczowym WHEN. Po znalezieniu pierwszego dopasowania następuje wykonanie listy zapytań odpowiadających dopasowanej wartości THEN. Takie rozwiązanie jest użyteczne podczas porównywania danej wartości ze zbiorem wartości.

Jeżeli nie podano wyrażenia początkowego (*expr*), wtedy CASE oblicza wyrażenie WHEN. Po znalezieniu pierwszego dopasowania następuje wykonanie listy zapytań odpowiadających dopasowanej wartości THEN. Takie rozwiązanie jest użyteczne podczas przeprowadzania testów nierówności lub sprawdzania dowolnych warunków.

W przypadku braku dopasowanego wyrażenia WHEN wykonane będą zapytania znajdujące się w klauzuli ELSE, o ile taka została zdefiniowana.

IF

```
IF expr1 THEN stmt_list1
  [ELSEIF expr2 THEN stmt_list2] ...
  [ELSE stmt_list]
END IF
```

Zapytanie IF pozwala na utworzenie konstrukcji rozgałęziającej przebieg działania programu. To zapytanie różni się od funkcji IF(), omówionej w punkcie C.2.1, zatytułowanym „Funkcje porównania”.

Jeżeli wyrażenie znajdujące się po słowie kluczowym IF przyjmuje wartość true, wykonane będą zapytania znajdujące się po pierwszej klauzuli WHEN. W przeciwnym razie nastąpi obliczenie wyrażeń zdefiniowanych w klauzulach ELSEIF. Po znalezieniu pierwszego, które przyjmuje wartość true, zostaną wykonane odpowiadające mu zapytania. Jeżeli żadne z wyrażeń nie przyjmuje wartości true, nastąpi wykonanie zapytań znajdujących się w klauzuli ELSE, o ile taka została zdefiniowana.

ITERATE

```
ITERATE label
```

Zapytanie ITERATE jest używane w konstrukcjach pętli do rozpoczęcia kolejnej iteracji pętli. Może być stosowane w pętlach LOOP, REPEAT i WHILE.

LEAVE

```
LEAVE label
```

Zapytanie LEAVE powoduje opuszczenie oznaczonej etykietą konstrukcji sterującej przebiegiem działania programu. To zapytanie musi zostać wykonane w konstrukcji, której przypisano etykietę.

LOOP

```
LOOP stmt_list END LOOP
label: LOOP stmt_list END LOOP [label]
```

Zapytanie powoduje zdefiniowanie i wykonanie pętli. Zapytania znajdujące się w pętli są powtarzane do chwili przekazania kontroli nad programem poza pętlę.

REPEAT

```
REPEAT stmt_list UNTIL expr END REPEAT
label: REPEAT stmt_list UNTIL expr END REPEAT [label]
```

Zapytanie powoduje zdefiniowanie i wykonanie pętli. Zapytania znajdujące się w pętli są powtarzane aż do chwili, gdy podane wyrażenie (*expr*) przyjmie wartość true.

RETURN

```
RETURN expr
```

Zapytanie RETURN jest używane jedynie w funkcjach składowanych (nie w procedurach składowanych, wyzwalaczach lub zdarzeniach). Podczas wykonania powoduje zakończenie działania funkcji, a wartość wyrażenia *expr* staje się wartością zwracaną zapytaniu, które wywołało daną funkcję. Funkcja musi zawierać przynajmniej jedno zapytanie RETURN.

WHILE

```
WHILE expr DO stmt_list END WHILE
label: WHILE expr DO stmt_list END WHILE [label]
```

Zapytanie powoduje zdefiniowanie i wykonanie pętli. Zapytania znajdujące się w pętli są powtarzane, dopóki podane wyrażenie (*expr*) przyjmuje wartość true.

E.2.2. Zapytania obsługi deklaracji

DECLARE

```
DECLARE var_name [, var_name] ... type [DEFAULT value]
DECLARE condition_name CONDITION FOR named_condition
named_condition: {SQLSTATE [VALUE] sqlstate_value | mysql_errno}
DECLARE cursor_name CURSOR FOR select_stmt
DECLARE handler_type
    HANDLER FOR handler_condition [, handler_condition] ...
statement
handler_type: {CONTINUE | EXIT}
handler_condition:
    SQLSTATE [VALUE] sqlstate_value
    | mysql_errno
    | condition_name
    | SQLWARNING
    | NOT FOUND
    | SQLEXCEPTION
```

Warianty zapytania DECLARE pozwalają na deklarowanie zmiennych lokalnych, warunków, kursorów i procedur obsługi. Zapytanie DECLARE może znajdować się jedynie na początku bloku BEGIN. Jeżeli blok zawiera wiele deklaracji, wtedy muszą być podane w następującej kolejności:

1. Deklaracje zmiennych i warunków.
2. Deklaracje kursorów.
3. Deklaracje procedur obsługi.

Po słowie kluczowym DECLARE powinna znajdować się lista rozdzielonych przecinkami deklaracji zmiennych lokalnych przeznaczonych do użycia w procedurze. Zmienna lokalna jest dostępna w bloku BEGIN, w którym została zdefiniowana, oraz w jego blokach zagnieźdżonych, ale nie w żadnym bloku zewnętrznym.

Zmienna lokalna może być w zapytaniu DECLARE zainicjalizowana wraz z klauzulą DEFAULT. Jeżeli definicja nie zawiera klauzuli DEFAULT, wtedy wartością początkową jest NULL. W celu przypisania wartości zmiennej lokalnej w dalszej części procedury należy użyć zapytania SET, SELECT ... INTO lub FETCH ... INTO.

Zapytanie DECLARE ... CONDITION tworzy nazwę dla warunku. Do *condition_name* można się odwoływać w zapytaniach DECLARE ... HANDLER, SIGNAL i RESIGNAL. Wartość *named_condition* może być wartością SQLSTATE w postaci pięciodziankowego, cytowanego ciągu tekstowego lub charakterystycznym dla MySQL numerem błędu.

Zapytanie DECLARE ... CURSOR powoduje zadeklarowanie kursora oraz powiązanie go ze wskazanym zapytaniem SELECT, które nie powinno zawierać klauzuli INTO. Kursor można otworzyć za pomocą OPEN, w zapytaniach FETCH używać do pobierania rekordów i zamykać za pomocą CLOSE.

Zapytanie DECLARE ... HANDLER powoduje powiązanie jednego lub więcej warunków z zapytaniem przeznaczonym do wykonania po spełnieniu dowolnego z wspomnianych warunków (zapytanie może być również zapytaniem złożonym zdefiniowanym za pomocą słów kluczowych BEGIN i END). Wartość *handler_type* wskazuje, co się stanie po wykonaniu procedury obsługi. Opcja CONTINUE powoduje kontynuację działania. Z kolei opcja EXIT kończy działanie bloku BEGIN, w którym została zdefiniowana dana procedura obsługi.

handler_condition może być dowolnym z wymienionych poniżej typów wartości:

- Wartość SQLSTATE w postaci pięciodziankowego ciągu tekstowego. Ta wartość nie powinna rozpoczynać się od '00', ponieważ przedstawia wówczas sukces, a nie błąd.
- Charakterystyczny dla MySQL numer błędu. Ta wartość nie powinna być zerem, ponieważ przedstawia wówczas sukces, a nie błąd.
- Warunek wymieniony wcześniej w DECLARE ... CONDITION.
- Wartość SQLWARNING, oznaczająca dowolną wartość SQLSTATE rozpoczynającą się od '01'.

- Wartość NOT FOUND, oznaczająca dowolną wartość SQLSTATE rozpoczynającą się od '02'.
- Wartość SQLEXCEPTION, oznaczająca dowolną wartość SQLSTATE, która nie rozpoczyna się od '00', '01' lub '02'.

Przykłady dla poszczególnych typów zapytania DECLARE znajdziesz w następnym punkcie.

E.2.3. Zapytania obsługi kursora

Zapytania przedstawione w tym punkcie pozwalają na otwieranie i zamykanie kursorów oraz używanie ich do pobierania rekordów, gdy kursor jest otwarty. Kursory w serwerze MySQL są tylko do odczytu i mogą być używane jedynie do poruszania się do przodu w zbiorze wynikowym, jednorazowo o jeden rekord (czyli nie obsługują przewijania). Przedstawiony poniżej fragment kodu pokazuje użycie zapytań dotyczących kursorów. Zaprezentowano wszystkie warianty zapytania DECLARE, które bardzo często są używane w połączeniu z pętlami kursorów.

```
CREATE PROCEDURE p ()
BEGIN
    DECLARE more_data INT DEFAULT TRUE;
    DECLARE id INT;
    DECLARE no_data CONDITION FOR SQLSTATE '02000';
    DECLARE curs CURSOR FOR SELECT member_id FROM sampdb.member;
    DECLARE CONTINUE HANDLER FOR no_data
    BEGIN
        SET more_data = FALSE;
    END;
    -- Otworzenie kursora, pobieranie rekordów za pomocą kursora aż do wystąpienia No Data
    OPEN curs;
    WHILE more_data DO
        FETCH curs INTO id;
    END WHILE;
    CLOSE curs;
END;
```

CLOSE

```
CLOSE cursor_name
```

Zapytanie zamyka wskazany kursor, który musi być otwarty. Otwarty kursor zostaje automatycznie zamknięty po zakończeniu bloku BEGIN, w którym został on zadeklarowany.

FETCH

```
FETCH [[NEXT] FROM] cursor_name INTO var_name [, var_name] ...
```

Zapytanie pobiera następny rekord ze wskazanego kursora i umieszcza w podanej zmiennej lub zmiennych. Kursor musi być otwarty. Jeżeli nie ma dostępnego rekordu, zostanie wygenerowany błąd o kodzie SQLSTATE 02000 oznaczający brak danych (No Data).

OPEN

OPEN *cursor_name*

Zapytanie otwiera wskazany kursor, który następnie może być użyty w zapytaniu FETCH.

E.2.4. Zapytania obsługi warunków

W trakcie wykonywania zapytań SQL generowane są przez nie informacje diagnostyczne dotyczące występujących warunków. Tego rodzaju informacje można przeanalizować za pomocą zapytania GET DIAGNOSTICS. Istnieje możliwość jawnego zgłoszenia warunku za pomocą SIGNAL i RESIGNAL. W tym punkcie zostaną omówione wymienione zapytania. Składnię deklarowania warunków i procedur obsługi warunków przedstawiono w punkcie E.2.2, zatytułowanym „Zapytania obsługi deklaracji”.

GET DIAGNOSTICS

```
GET [CURRENT] DIAGNOSTICS
{
  var_name = stmt_info_type
    [, var_name = stmt_info_type] ...
  | CONDITION n
  var_name = condition_info_type
    [, var_name = condition_info_type] ...
}
stmt_info_type:
  NUMBER
  | ROW_COUNT
condition_info_type:
  CLASS_ORIGIN
  | SUBCLASS_ORIGIN
  | RETURNED_SQLSTATE
  | MESSAGE_TEXT
  | MYSQL_ERRNO
  | CONSTRAINT_CATALOG
  | CONSTRAINT_SCHEMA
  | CONSTRAINT_NAME
  | CATALOG_NAME
  | SCHEMA_NAME
  | TABLE_NAME
  | COLUMN_NAME
  | CURSOR_NAME
```

Zapytanie pozwala na analizę zawartości obszaru diagnostycznego, który składa się z dwóch rodzajów informacji wygenerowanych przez zapytanie. Pierwszy to informacje o samym zapytaniu, na przykład liczba rekordów, których dotyczyło zapytanie, lub liczba spełnionych warunków. Drugi to informacje o warunkach wygenerowanych przez zapytanie, na przykład kody i komunikaty błędów. Obszar diagnostyczny może zawierać informacje o wielu warunkach. Zapytanie GET DIAGNOSTICS zostało wprowadzone w MySQL 5.6.4.

Słowo kluczowe `CURRENT` nie ma znaczenia w MySQL i może zostać pominięte. Standard SQL posiada stos obszarów diagnostycznych, natomiast MySQL ma tylko jeden, który zawsze jest obszarem bieżącym.

Każde wywołanie `GET DIAGNOSTICS` zwraca informacje o zapytaniu lub po użyciu klauzuli `CONDITION` informacje o warunkach. Na przykład, w celu pobrania i przypisania zmiennym zdefiniowanym przez użytkownika informacji o liczbie rekordów, na które wpłynęło zapytanie, oraz o warunkach należy wykonać poniższe zapytanie:

```
GET DIAGNOSTICS
  @affected_rows = ROW_COUNT, @condition_count = NUMBER;
```

Pobranie wartości `SQLSTATE` i komunikatu błędu dla warunku 2 umożliwia następujące zapytanie:

```
GET DIAGNOSTICS CONDITION 2
  @sqlstate = RETURNED_SQLSTATE, @message = MESSAGE_TEXT;
```

Numer warunku w klauzuli `CONDITION` musi być liczbą całkowitą z zakresu od 1 do ilości warunków. Tę liczbę można podać jako dosłowną wartość, zmienną zdefiniowaną przez użytkownika, zmienną systemową, zmienną lokalną programu składowanego bądź jako parametr funkcji lub procedury składowanej. Wartość zmiennej systemowej `max_error_count` określa liczbę warunków możliwych do przechowywania w obszarze diagnostycznym. Zmienna `warning_count` wskazuje liczbę błędów, ostrzeżeń i powiadomień, które wystąpiły. Ta wartość może być większa niż `NUMBER`, ponieważ zlicza warunki niezależnie od tego, czy jest miejsce na ich przechowywanie. Zmienna `error_count` jest podobna, ale zlicza jedynie warunki błędów.

Pozostała część zapytania, wskazująca typ pobieranych informacji, składa się z jednej lub więcej zmiennych rozdzielonych przecinkami. Każda zmienna `var_name` może być zmienną zdefiniowaną przez użytkownika, zmienną lokalną programu składowanego bądź też parametrem funkcji lub procedury składowanej. Poniżej przedstawiono dozwolone nazwy dla typów informacji przypisywanych zmiennym. Elementy liczbowe są wartościami w postaci liczb całkowitych, pozostałe wartości są ciągami tekstowymi.

Dla znajdujących się w obszarze diagnostycznym informacji o zapytaniach dostępne są następujące elementy:

- **NUMBER.** Liczba warunków w obszarze diagnostycznym. Istnieje możliwość, że nie będzie żadnych warunków, i wtedy wartością `NUMBER` jest 0. Aby ustalić liczbę warunków, należy wykonać zapytanie takie jak przedstawiono poniżej:

```
GET DIAGNOSTICS @condition_count = NUMBER;
```

- **ROW_COUNT.** Liczba rekordów, na które wpłynęło zapytanie.

Dla znajdujących się w obszarze diagnostycznym informacji o warunkach dostępne są wymienione poniżej elementy. Za wyjątkiem `RETURNED_SQLSTATE` mogą być one wymienione w klauzuli `SET` zapytań `SIGNAL` i `RESIGNAL`. Oba wymienione zapytania pośrednio ustawiają wartość `RETURNED_SQLSTATE` za pomocą nazwy warunku.

- **CLASS_ORIGIN, SUBCLASS_ORIGIN.** Klasa i podklasa wartości **RETURNED_SQLSTATE**. Jeżeli ta wartość została zdefiniowana w dokumencie standardu ISO 9075-2, klasą będzie 'ISO 9075', w przeciwnym razie 'MYSQL'. Jeżeli klasą będzie 'ISO 9075' lub **RETURNED_SQLSTATE** kończy się na '000', wtedy podklasą będzie 'ISO 9075', w przeciwnym razie 'MYSQL'.
- **RETURNED_SQLSTATE.** Ciąg tekstowy **SQLSTATE**.
- **MESSAGE_TEXT.** Komunikat błędu.
- **MYSQL_ERRNO.** Numer błędu w MySQL.
- **CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME.** Katalog, schemat i nazwa złamanego ograniczenia. Ta wartość jest zawsze pusta w MySQL.
- **CATALOG_NAME, SCHEMA_NAME, TABLE_NAME, COLUMN_NAME.** Katalog, schemat, tabela i kolumna dla warunku. Ta wartość jest zawsze pusta w MySQL.
- **CURSOR_NAME.** Nazwa kursora dla warunku. Ta wartość jest zawsze pusta w MySQL.

Niektóre z wymienionych powyżej wartości są zawsze puste w **GET DIAGNOSTICS**. Mogą mieć niepuste wartości w zapytaniach **SIGNAL** i **RESIGNAL**, ponieważ wymienione zapytania przypisują wartości elementom obszaru warunku.

Zapytania używające tabel czyszczą zawartość obszaru diagnostycznego w trakcie ich wykonywania. Pozostałe zapytania czyszczą obszar diagnostyczny warunków z poprzednich zapytań, jeśli warunek wystąpił w trakcie wykonywania. Wyjątkami są zapytania **GET DIAGNOSTICS** i **RESIGNAL**, które dodają warunki do obszaru diagnostycznego, zamiast go czyścić. Zapytania **GET DIAGNOSTICS**, **SHOW WARNINGS** i **SHOW ERRORS** pobierają lub wyświetlają informacje o warunkach bez czyszczenia obszaru diagnostycznego.

RESIGNAL

```
RESIGNAL [ condition_value]
[SET condition_info_type = value
[, condition_info_type = value] ...]
condition_value, condition_info_type, value:
Patrz opis zapytania SIGNAL.
```

Zapytanie **RESIGNAL** jest podobne do **SIGNAL**, ale używane do przekazania wraz z wcześniej wygenerowanymi informacjami o warunkach, prawdopodobnie modyfikując je lub dodając do nich. Takie rozwiązanie może być użyteczne w przypadku procedury obsługi warunków po odkryciu, że dany warunek jest lepiej obsługiwany w zewnętrznym kontekście, a nie w bieżącym. W przeciwieństwie do **SIGNAL**, zapytanie **RESIGNAL** może być używane jedynie wtedy, gdy aktywne są pewne procedury obsługi warunku.

Składnię wyrażeń używanych w **RESIGNAL** znajdziesz w podpunkcie poświęconym zapytaniu **SIGNAL**. Samo zapytanie **RESIGNAL** jest praktycznie takie samo, ale wartość *condition_value* jest opcjonalna:

- Zapytanie **RESIGNAL** bez *condition_value* jest przekazywane wraz istniejącymi informacjami warunków. Jeżeli użyta zostanie klauzula **SET**, przypisania w klauzuli spowodują modyfikacje wymienionych elementów informacji o warunkach.

- Zapytanie RESIGNAL wraz z *condition_value* umieszcza na końcu obszaru diagnostycznego kopię ostatniego warunku i modyfikuje go zgodnie z *condition_value* oraz wszelkimi przypisaniami istniejącymi w klauzuli SET, o ile została podana.

SIGNAL

```
SIGNAL condition_value
  [SET condition_info_type = value
    [, condition_info_type = value] ...]
condition_value:
  SQLSTATE [VALUE] sqlstate_value
  | condition_name
condition_info_type:
  CLASS_ORIGIN
  | SUBCLASS_ORIGIN
  | MESSAGE_TEXT
  | MYSQL_ERRNO
  | CONSTRAINT_CATALOG
  | CONSTRAINT_SCHEMA
  | CONSTRAINT_NAME
  | CATALOG_NAME
  | SCHEMA_NAME
  | TABLE_NAME
  | COLUMN_NAME
  | CURSOR_NAME
```

To zapytanie pozwala na sygnalizowanie błędów, co może być pomocne podczas obsługi problemów występujących podczas wykonywania zapytania i dostarcza informacje na temat tego, co się wydarzyło.

Wartość *condition_value* umieszczona po słowie kluczowym SIGNAL wskazuje warunek, który ma być sygnalizowany. Jest on podany w postaci pięciodziankowego, dosłownego ciągu tekstowego SQLSTATE lub nazwy warunku zadeklarowanego za pomocą zapytania DECLARE ... CONDITION. Więcej informacji na temat wartości SQLSTATE i składni zapytań DECLARE znajdziesz w punkcie E.2.2, zatytułowanym „Zapytania obsługi deklaracji”.

Opcjonalna klauzula SET powoduje przypisanie wartości elementom informacji o warunku, co pozwala na jego „opisanie”. Klauzula SET składa się z jednego lub więcej rozdzielonych przecinkami przypisań, każde wskazuje typ informacji i przypisaną wartość. Przedstawiony poniżej kod sprawdza dzielnik w operacji dzielenia. Jeżeli ma dojść do dzielenia przez zero, wtedy kod sygnalizuje błąd oraz ustawia numer kodu błędu w MySQL i komunikat błędu:

```
IF divisor <> 0 THEN
  SET ratio = numerator / divisor;
ELSE
  SIGNAL SQLSTATE '22012'
    SET MYSQL_ERRNO = 1365, MESSAGE_TEXT = 'Próba dzielenia przez zero';
END IF;
```

W żadnym przypisaniu wartości *value* nie może być ogólne wyrażenie. To może być jedynie dosłowna wartość, zmienna zdefiniowana przez użytkownika, zmienna systemowa, zmienna lokalna programu składowanego bądź też parametr funkcji lub procedury składowanej.

Listę dozwolonych nazw typów warunków i ich znaczenie znajdziesz w podpunkcie poświęconym zapytaniu GET DIAGNOSTICS.

E.3. Składnia komentarzy

MySQL pozwala na umieszczanie komentarzy w kodzie SQL, co może być użyteczne w celu dokumentowania zapytań przechowywanych w plikach. Serwer MySQL obsługuje trzy rodzaje zapytań:

- Wszystko to, co znajduje się między znakiem # i końcem wiersza, jest traktowane jako komentarz. Ta składnia jest taka sama jak w większości powłok systemu UNIX oraz w wielu językach skryptowych, na przykład Perl, PHP i Ruby.

To jest pojedynczy wiersz komentarza.

- Wszystko to, co znajduje się między znakami /* i */, jest traktowane jako komentarz. Komentarz w takiej postaci może obejmować wiele wierszy. Składnia jest taka sama jak w języku programowania C.

```
/* To jest pojedynczy wiersz komentarza. */
/* To jest
   komentarz obejmujący
   wiele wierszy.
*/
```

- Komentarz można rozpocząć także od dwóch myślników i spacji lub dwóch myślników i znaku kontrolnego, na przykład w postaci znaku nowego wiersza. Wszystko to, co znajduje się między znakami -- i końcem wiersza, jest traktowane jako komentarz.

```
--
-- To jest komentarz.
--
```

Obsługiwany przez MySQL styl w postaci dwóch myślników różni się od stylu komentarzy w standardzie SQL — również rozpoczynają się od dwóch myślników, ale nie wymagają umieszczenia po nich spacji. MySQL wymaga użycia spacji. To jest rodzaj pomocy w celu uniknięcia niejasności. W przeciwnym razie wyrażenia takie jak 5--7 mogą zostać uznane za zawierające sekwencję rozpoczynającą komentarz. Istnieje niewielkie prawdopodobieństwo, że wymienione wyrażenie będzie zapisywane w postaci 5-- 7, więc przyjęte rozwiązanie jest użyteczne. Pomimo tego nadal powinieneś zachować ostrożność podczas importowania do MySQLa innych systemów baz danych kodu SQL zawierającego komentarze rozpoczynające się od dwóch myślników.

Podczas wykonywania zapytań serwer ignoruje komentarze, za wyjątkiem komentarzy w stylu języka C, rozpoczynających się sekwencją specjalną `/*!`. Istnieje możliwość „ukrycia” charakterystycznych dla MySQL słów kluczowych właśnie w komentarzach w stylu C, rozpoczynających się od sekwencji `/*!` zamiast `/*`. MySQL sprawdza ten typ specjalny komentarzy i interpretuje je jako słowa kluczowe, natomiast inne serwery bazy danych traktują je jako część komentarza. W ten sposób zwiększa się poziom przenośności kodu, przynajmniej dla innych serwerów obsługujących komentarze w stylu języka C: istnieje możliwość tworzenia kodu wykorzystującego funkcje charakterystyczne dla MySQL podczas wykonywania kodu w serwerze MySQL i uruchamiania tego kodu bez żadnych modyfikacji w innych serwerach bazy danych. Przedstawione poniżej zapytania są takie same dla serwerów innych niż MySQL, natomiast drugie z wymienionych jest przez MySQL wykonywane jako `INSERT LOW_PRIORITY`.

```
INSERT INTO mytbl (id,date) VALUES(13,'2013-09-28');
INSERT /*! LOW_PRIORITY */ INTO mytbl (id,date) VALUES(13,'2013-09-28');
```

Komentarze w stylu języka C mogą być charakterystyczne dla konkretnej wersji. Jeżeli po sekwencji początkowej `/*!` znajduje się pięciocyfrowy numer wersji, wtedy serwer zignoruje taki komentarz, o ile nie będzie co najmniej we wskazanej wersji. Komentarz w poniższym zapytaniu `DELETE` będzie ignorowany, jeśli serwer nie jest w wersji 5.6.2 lub nowszej (serwer obsługuje klauzulę `PARTITION` zapytania `DELETE`, dopiero począwszy od wersji MySQL 5.6.2):

```
DELETE FROM mytbl /*!50602 PARTITION (p1) */ WHERE date < '2010-01-01';
```


Przewodnik po programie SQL

W tym dodatku znajdują się ogólne informacje dotyczące wywoływania programów MySQL. Ponadto, nieco dokładniej omówiono programy wymienione na poniższej liście. Omówienie każdego programu zawiera jego przeznaczenie, składnię wywołania, obsługiwane opcje oraz opis wszelkich zmiennych wewnętrznych, o ile takie zawiera. Jeżeli nie zaznaczono inaczej, zaprezentowane tutaj zapytania są dostępne w wydaniu MySQL 5.5.0 lub nowszym. Wyraźnie zaznaczono zmiany wprowadzone w późniejszych wydaniach.

- `myisamchk`
Program umożliwia sprawdzanie i naprawę tabel MyISAM, przeprowadzanie analizy rozproszenia kluczy, włączenie oraz wyłączenie indeksów.
- `mysql`
Program umożliwia prowadzenie komunikacji z serwerem MySQL oraz wykonywanie zapytań z pliku (w trybie wsadowym).
- `mysql.server`
Program umożliwia uruchamianie i zatrzymywanie serwera MySQL.
- `mysql_config`
Program wyświetla prawidłowe flagi dla kompilacji programów opartych na MySQL.
- `mysql_install_db`
Program umożliwia inicjalizację katalogu danych serwera oraz tabel uprawnień.
- `mysql_upgrade`
Program umożliwia uaktualnienie baz danych po instalacji nowej wersji MySQL.
- `mysqladmin`
Program umożliwia przeprowadzanie operacji administracyjnych.
- `mysqlbinlog`
Program wyświetla w formacie tekstowym pliki dziennika binarnego i przekazywania.

- `mysqlcheck`
Program umożliwia sprawdzanie, naprawę, optymalizację i analizę tabel.
- `mysqld`
Serwer MySQL; ten program musi działać, aby klienci miały dostęp do baz danych.
- `mysqld_multi`
Program umożliwia uruchamianie i zatrzymywanie wielu serwerów MySQL.
- `mysqld_safe`
Program umożliwia uruchamianie i monitorowanie serwera MySQL.
- `mysqldump`
Program umożliwia utworzenie kopii zapasowej zawartości bazy danych.
- `mysqlimport`
Program umożliwia grupowe wczytanie danych do tabel.
- `mysqlshow`
Program wyświetla informacje o bazach danych i tabelach.
- `perror`
Program wyświetla znaczenie kodów błędów.

Nawiasy kwadratowe ([]) w opisach składni oznaczają informacje opcjonalne.

F.1. Wyświetlanie komunikatu pomocy programu

Każdy opis programów przedstawionych w dalszej części dodatku zawiera opcje obsługiwane przez dany program. Jeżeli program nie rozpoznaje opcji wymienionej w opisie, być może korzystasz ze starszej wersji programu lub nowej, w której dana opcja została uznana za przestarzałą bądź po prostu usunięta.

Aby pobrać listę obsługiwanych opcji, należy sprawdzić komunikat pomocy programu, który pozwala na szybkie uzyskanie informacji o samym programie. W przypadku serwera (`mysqld`) można użyć zarówno opcji `--version`, jak i `--help`. Z kolei w pozostałych programach należy użyć opcji `--help`. Na przykład, jeśli nie jesteś pewien, jak używać programu `mysqlimport`, wtedy wywołaj go w następujący sposób:

```
% mysqlimport --help
```

Opcja `-?` ma takie samo znaczenie jak `--help`, ale powłoka (interpreter poleceń) może potraktować znak zapytania jako znak wieloznaczny w nazwie pliku:

```
% mysqlimport -?  
mysqlimport: No match.
```

W takim przypadku polecenie powinno zostać wydane w następujący sposób:

```
% mysqlimport -\?
```

Pewne opcje są wyświetlane w komunikacie pomocy jedynie w określonych sytuacjach. Na przykład, opcje dotyczące SSL pojawiają się tylko wtedy, gdy serwer MySQL został skompilowany wraz z obsługą SSL. Z kolei opcje dostępne dla systemu Windows, na przykład `--pipe`, są wyświetlane jedynie w systemach Windows.

Komunikat pomocy programu MySQL może wyświetlać także katalogi domyślnie sprawdzane przez program w poszukiwaniu plików opcji, a także informować o obsługiwanych zmiennych.

F.2. Określanie opcji programu

Większość programów MySQL obsługuje wiele opcji wpływających na ich działanie. Wspomniane opcje mogą być podawane w wierszu poleceń lub pliku opcji. Ponadto, pewne opcje można wskazać przez ustawienie odpowiednich zmiennych środowiskowych. Opcje podane w wierszu poleceń mają pierwszeństwo przed podanymi w inny sposób. Z kolei opcje w plikach opcji mają pierwszeństwo przez wartościami zmiennych środowiskowych.

Większość opcji posiada zarówno formę długą (pełne słowo), jak i krótką (pojedyncza litera). Przykładem może być wymieniona wcześniej para opcji `--help` i `-?`. Po długiej formie opcji wartość jest podawana w następujący sposób: `--nazwa=wartość` lub `--nazwa wartość`, gdzie *nazwa* oznacza nazwę opcji, natomiast *wartość* jej wartość. Z kolei w krótkiej formie wartość jest podawana po literze opcji i najczęściej jest oddzielona spacją. Na przykład, przy podawaniu nazwy użytkownika opcja `-usampadm` jest odpowiednikiem `-u sampadm`. Wyjątkiem jest opcja `-p` (hasło) — wartość hasła jest opcjonalna, ale *musi* być podana natychmiast po `-p` i bez spacji.

Nazwy opcji rozróżniają wielkość liter, natomiast wartości mogą ją rozróżniać, choć nie muszą. Na przykład, wartości takie jak nazwa użytkownika i hasła rozróżniają wielkość liter, natomiast wartość opcji `--protocol` już nie. Aby nawiązać połączenie TCP/IP, można użyć opcji `--protocol=tcp` lub `--protocol=TCP`.

Wiele opcji jest typu boolowskiego i mają wartość włączającą lub wyłączającą daną opcję. Tego rodzaju opcje mają podstawową formę i standardowy zestaw powiązanych ze sobą form, jak przedstawiono w tabeli F.1.

Tabela F.1. Formy stosowane przez opcje typu boolowskiego

Opcja	Opis
<code>--nazwa</code>	Podstawowa forma opcji, powoduje jej włączenie.
<code>--enable-nazwa</code>	Prefiks <code>--enable-</code> powoduje włączenie opcji.
<code>--disable-nazwa</code>	Prefiks <code>--disable-</code> powoduje wyłączenie opcji.
<code>--skip-nazwa</code>	Prefiks <code>--skip-</code> powoduje wyłączenie opcji.
<code>--nazwa=1</code>	Przyrostek <code>=1</code> powoduje włączenie opcji.
<code>--nazwa=0</code>	Przyrostek <code>=0</code> powoduje wyłączenie opcji.

Na przykład, wiele programów klienckich MySQL pozwala na wskazanie użycia kompresji w protokole klient-serwer. W przypadku tego rodzaju programów użycie opcji `--compress` powoduje włączenie kompresji, natomiast pominięcie wymienionej opcji oznacza nieużywanie kompresji. Istnieje jednak wiele różnych sposobów na osiągnięcie tego samego celu: `--enable-compress` i `--compress=1` również włączają kompresję, natomiast `--disable-compress`, `--skip-compress` i `--compress=0` powodują, że kompresja nie będzie stosowana.

Formaty jawnie wyłączające opcje są szczególnie użyteczne w przypadku domyślnie włączonych opcji. Jeśli chodzi o kompresję protokołu, to można ją wyłączyć, po prostu pomijając opcję `--compress`. Takie rozwiązanie jednak nie działa dla opcji, które domyślnie są włączone. Na przykład, opcja `--quote-names` dla `mysqldump` jest domyślnie włączona. Nie można wyłączyć cytowania przez pominięcie opcji, ale można to zrobić za pomocą jednej z wymienionych opcji: `--skip-quote-names`, `--disable-quote-names` lub `--quote-names=0`.

W przedstawionych w tym dodatku opisach programów wyrażenie (*boolean*) wskazuje opcje, które są interpretowane w omówiony powyżej sposób, czyli opcje obsługują prefiksy i przyrostki wymienione w tabeli F.1.

W razie wątpliwości zawsze możesz zajrzeć do komunikatu pomocy programu i sprawdzić obsługiwane przez niego opcje (patrz podrozdział F.1, zatytułowany „Wyświetlanie komunikatu pomocy programu”).

Programy MySQL mają jeszcze inne standardowe funkcje związane z przetwarzaniem opcji:

- Długie opcje mogą być skracane do jednoznacznych prefiksów, które są łatwiejsze do użycia w przypadku opcji posiadających bardzo długie nazwy. Jeżeli użyty prefiks nie będzie wystarczająco długi, aby był jednoznaczny, wywołany program poinformuje o tym oraz wyświetli listę opcji dopasowanych do prefiksu:

```
% mysql --h
mysql: ambiguous option '--h' (help, html)
```

- Zmienne programu można ustawić z poziomu wiersza zapytań lub w plikach opcji, traktując nazwy zmiennych jako nazwy opcji. Wprawdzie nazwy zmiennych stosują znak podkreślenia zamiast myślnika, ale można używać myślnika lub znaku podkreślenia w nazwach opcji bądź zmiennych podawanych w wierszu poleceń lub plikach opcji. (Znajdujące się na początku długiej nazwy opcji w wierszu poleceń dwa myślniki nie mogą być użyte). Więcej informacji znajdziesz w podpunkcie F.2.1.2, zatytułowanym „Ustawianie zmiennych programu”.
- MySQL obsługuje prefiks `--loose-`, ułatwiający użycie różnych wersji programów nieobsługujących tego samego zestawu opcji. W przypadku `--loose-` nierozpoznana opcja skutkuje jedynie wygenerowaniem ostrzeżenia, a nie zakończeniem działania programu wraz z błędem. Na przykład, klient `mysql`, począwszy od wersji 5.5.3, akceptuje opcję `--auto-vertical-output`, natomiast starsze wersje

nie rozpoznają tej opcji. Użycie opcji `--loose-auto-vertical-output` powoduje, że każda wersja `mysql` zaakceptuje tę opcję, pewne wersje jej użyją, starsze po prostu zignorują.

- Serwer MySQL (`mysqld`) obsługuje prefiks `--maximum-`, pozwalający na wskazanie wartości maksymalnej, do której mogą być ustawione zmienne modyfikowalne przez użytkownika. Na przykład, serwer pozwala użytkownikom na ustawienie wielkości ich bufora sortowania za pomocą zmiennej `sort_buffer_size`. Aby nałożyć ograniczenie do 64 MB wielkości wymienionej zmiennej, należy uruchomić serwer wraz z opcją `--maximum-sort_buffer_size=64M`.

F.2.1. Opcje standardowe programu MySQL

Wiele opcji ma standardowe znaczenie, a większość lub nawet wszystkie programy MySQL interpretują je w dokładnie taki sam sposób. Zamiast powtarzać ten sam opis podczas omawiania wielu programów, wspomniane opcje omówiono jednokrotnie w tym miejscu, a punkt „Opcje standardowe obsługiwane przez ...” w każdym programie wskazuje, które z omówionych tutaj opcji są obsługiwane przez dany program. W tym miejscu wymieniono jedynie długie nazwy opcji, ale o ile nie wspomniano inaczej, to programy obsługują również odpowiadające im krótkie nazwy opcji.

Poniższa lista opisuje opcje standardowe. Wymienione wartości domyślne mają zastosowanie, o ile nie zostały zmienione w trakcie kompilacji serwera MySQL.

- `--bind-address=addr`

Do programów klienckich ta opcja została wprowadzona w MySQL 5.6.1 i wskazuje adres IP, który ma zostać przypisany w trakcie nawiązywania połączenia z serwerem. To może być użyteczne w komputerach posiadających wiele interfejsów sieciowych. W przypadku serwera omawiana opcja ma o wiele bardziej skomplikowane znaczenie. Patrz punkt F.12.2, zatytułowany „Opcje charakterystyczne dla `mysqld`”.

- `--character-sets-dir=dir_name`

Nazwa katalogu przechowującego pliki kodowań znaków.

- `--compress, -C (boolean)`

Ta opcja jest stosowana jedynie przez programy klienckie i oznacza żądanie użycia kompresji w protokole komunikacyjnym klient-serwer, o ile jest on obsługiwany zarówno przez klienta, jak i serwer.

- `--debug=debug_options, -# debug_options`

Włączenie danych wyjściowych debugowania. Ta opcja nie ma żadnego efektu, dopóki serwer MySQL nie zostanie skompilowany z włączoną obsługą debugowania. Ciąg tekstowy `debug_options` składa się z rozdzielonych dwukropkami opcji. Typowa wartość wynosi `d:t:o,nazwa_pliku` i powoduje włączenie debugowania, włączenie śledzenia rozpoczęcia i zakończenia działania funkcji oraz umieszczenie danych wyjściowych we wskazanym pliku.

Jeżeli planujesz dość często korzystać z debugowania, powinieneś przeanalizować podręcznik użytkownika biblioteki DBUG i zapoznać się z opisami wszystkich używanych przez siebie opcji. Wspomniany podręcznik użytkownika znajdziesz w katalogu *debug* w dystrybucji kodu źródłowego MySQL.

- **--debug-check** (*boolean*)
Opcja pozwala na sprawdzenie użycia pamięci i otwartych plików w trakcie kończenia działania programu.
- **--debug-info** (*boolean*)
Opcja działa jak **--debug-check**, ale wyświetla również informacje o użyciu pamięci i procesora.
- **--default-auth=plugin_name**
W przypadku klientów ta opcja wskazuje nazwę używanej wtyczki uwierzytelniania. Użycie tej opcji jest raczej niepotrzebne, ponieważ serwer wysyła nazwę wtyczki uwierzytelniania na podstawie danych podanych przez klienta. Ta opcja została wprowadzona w MySQL 5.5.7.
- **--default-character-set=charset**
Opcja pozwala na zdefiniowanie domyślnego kodowania znaków. Wartością *charset* nie może być *ucs2*, *utf16*, *utf16le* i *utf32*.
- **--help, -?**
Opcja wyświetla komunikat pomocy i kończy działanie programu. Patrz także podrozdział F.1, zatytułowany „Wyświetlanie komunikatu pomocy programu”.
- **--host=host_name, -h host_name**
Ta opcja jest używana jedynie przez programy klienckie. Wskazuje komputer, z którym ma zostać nawiązane połączenie (czyli komputer zawierający działający serwer). Wartością domyślną jest *localhost*.
- **--password[=pass_val], -p[pass_val]**
Ta opcja jest używana jedynie przez programy klienckie. Wskazuje hasło, które ma zostać użyte podczas nawiązania połączenia z serwerem. Jeżeli po nazwie opcji nie podasz hasła, program wyświetli komunikat proszący o jego wprowadzenie. Jeśli zaś chcesz podać hasło, musi się ono znajdować natychmiast po **-p**, bez spacji między **-p** i hasłem. Innymi słowy, hasło musi być podane jako *-password*, a nie jako *-p password*.
W celu wyraźnego wskazania braku hasła należy użyć opcji **--skip-password**. To jest użyteczne rozwiązanie, gdy plik opcji zawiera hasło, które chcesz zignorować.
- **--pipe, -W**
Opcja pozwala na podanie nazwanego potoku używanego podczas nawiązywania połączenia z serwerem. Jest używana jedynie przez programy klienckie działające w systemie Windows i tylko do nawiązywania połączenia z serwerami Windows posiadającymi włączoną obsługę nazwanych potoków.

■ `--plugin-dir=dir_name`

W przypadku serwera ta opcja wskazuje katalog zawierający wtyczki działające po stronie serwera. Z kolei dla klientów wskazuje katalog zawierający wtyczki działające po stronie klienta. Ta opcja została wprowadzona w MySQL 5.5.7.

■ `--port=port_num, -P port_num`

Dla mysqld ta opcja wskazuje numer portu, na którym będą nasłuchiwane połączenia TCP/IP. Domyślny numer portu to 3306. Z kolei w przypadku klientów to numer portu używany podczas nawiązywania połączenia z serwerem za pomocą protokołu TCP/IP.

• `--protocol=protocol_type`

Ta opcja jest używana jedynie przez programy klienckie i wskazuje rodzaj nawiązywanego połączenia z serwerem. Wartością *protocol_type* może być *tcp* (użycie protokołu TCP/IP), *socket* (użycie pliku gniazda systemu UNIX), *pipe* (użycie nazwanego potoku w Windows) lub *memory* (użycie pamięci współdzielonej). Wartość nie rozróżnia wielkości liter.

Pewne rodzaje połączeń są charakterystyczne dla danej platformy lub możliwe do użycia podczas nawiązywania połączenia z serwerem lokalnym działającym w tym samym komputerze, w którym został uruchomiony klient:

- ◆ Połączenia za pomocą gniazda, nazwanego potoku i pamięci współdzielonej mogą być nawiązywane jedynie z serwerem lokalnym.
- ◆ Połączenia za pomocą gniazda mogą być stosowane jedynie w systemach UNIX.
- ◆ Połączenia za pomocą nazwanego potoku i pamięci współdzielonej mogą być stosowane jedynie w systemach Windows.
- ◆ Połączenia TCP/IP mogą być stosowane na dowolnej platformie do nawiązywania połączeń z serwerami lokalnymi lub zdalnymi.

Opcja `--protocol` może być stosowana w połączeniu z innymi opcjami dostarczającymi informacje o połączeniu z serwerem:

- ◆ W przypadku połączeń TCP/IP opcje `--host` i `--port` wskazują nazwę komputera oraz numer portu TCP/IP.
- ◆ W przypadku połączeń za pomocą gniazda lub nazwanego potoku opcja `--socket` określa nazwę pliku gniazda systemu UNIX lub nazwanego potoku w Windows.
- ◆ W przypadku połączeń za pomocą pamięci współdzielonej opcja `--shared-memory-base-name` określa nazwę pamięci współdzielonej.

■ `--shared-memory-base-name=name`

Opcja wskazuje nazwę pamięci współdzielonej do użycia w połączeniach za pomocą pamięci współdzielonej. Wartością domyślną jest `MYSQL`; wielkość liter ma znaczenie.

- `--silent, -s`

Opcja ograniczenia liczby komunikatów generowanych przez program. To niekoniecznie oznacza całkowite wyłączenie generowania komunikatów, program po prostu generuje ich mniej. Niektóre programy pozwalają na kilkukrotne użycie tej opcji, co prowadzi do coraz większego ograniczenia liczby generowanych przez nie komunikatów.

- `--socket=file_name, -S file_name`

W przypadku klientów w systemie UNIX ta opcja zawiera pełną ścieżkę dostępu do pliku gniazda używanego podczas nawiązywania połączenia z serwerem localhost. Domyślna ścieżka dostępu do pliku gniazda systemu UNIX to `/tmp/mysql.sock`. Wielkość liter ma znaczenie, jeśli rozróżnia ją system plików w komputerze, w którym działa serwer MySQL. Z kolei dla klientów Windows to nazwa potoku używanego podczas połączenia z serwerem za pomocą nazwanego potoku. Domyślną nazwą potoku jest MySQL. Nazwy potoków nie rozróżniają wielkości liter.

- `--user=user_name, -u user_name`

W przypadku `mysqld` ta opcja wskazuje nazwę lub identyfikator użytkownika konta systemu UNIX, w ramach którego został uruchomiony serwer. Aby ta opcja działała, serwer musi być uruchomiony przez użytkownika root, ponieważ tylko on może zmienić identyfikator użytkownika na wskazany. Z kolei dla programów klienckich ta opcja wskazuje nazwę użytkownika konta MySQL używanego podczas nawiązywania połączenia z serwerem. Domyślnie to nazwa logowania do systemu UNIX lub ODBC w Windows.

- `--verbose, -v`

Opcja nakazuje programowi generowanie rozbudowanych komunikatów, program generuje więc więcej danych wyjściowych niż zwykle. Niektóre programy pozwalają na kilkukrotne użycie tej opcji, co prowadzi do kolejnego zwiększania liczby generowanych przez nie komunikatów.

- `--version, -V`

Opcja powoduje wyświetlenie ciągu tekstowego zawierającego informacje o wersji programu, a następnie zakończenie działania programu.

F.2.1.1. Opcje standardowe SSL

Omówione poniżej opcje są używane do nawiązywania bezpiecznych połączeń. Będą dostępne, jeśli serwer MySQL został skompilowany wraz z obsługą SSL. Więcej informacji na temat nawiązywania bezpiecznych połączeń znajdziesz w podrozdziale 13.5, zatytułowanym „Konfiguracja bezpiecznych połączeń za pomocą SSL”.

- `--ssl (boolean)`

Włączenie obsługi połączeń SSL. Użycie `--ssl` jest niejawnie wskazywane przez pozostałe opcje SSL; znacznie częściej spotyka się ją jako `--skip-ssl` w celu wyłączenia obsługi SSL.

- `--ssl-ca=file_name`
Ścieżka dostępu prowadząca do pliku zawierającego certyfikat.
- `--ssl-capath=dir_name`
Ścieżka dostępu prowadząca do katalogu zawierającego zaufane certyfikaty używane podczas weryfikacji certyfikatu.
- `--ssl-cert=file_name`
Ścieżka dostępu prowadząca do pliku certyfikatu.
- `--ssl-cipher=str`
Ciąg tekstowy wymieniający algorytm używany przez SSL do szyfrowania ruchu sieciowego w połączeniu. Wartość tej opcji powinna zawierać nazwę jednego lub więcej rozdzielonych przecinkami algorytmów szyfrowania.
- `--ssl-crl=file_name`
Ścieżka dostępu prowadząca do pliku zawierającego listę unieważnionych certyfikatów. Ta opcja została wprowadzona w MySQL 5.6.3.
- `--ssl-crlpath=dir_name`
Ścieżka dostępu prowadząca do katalogu zawierającego pliki unieważnionych certyfikatów. Ta opcja została wprowadzona w MySQL 5.6.3.
- `--ssl-key=file_name`
Ścieżka dostępu prowadząca do pliku klucza.
- `--ssl-verify-server-cert (boolean)`
Ta opcja jest stosowana jedynie w programach klienckich i nakazuje klientowi sprawdzenie wartości „Common Name” w certyfikacie otrzymanym z serwera. Jeżeli ta wartość różni się od komputera serwera, z którym klient nawiązał połączenie, wtedy próba połączenia zostanie odrzucona.

F.2.1.2. Ustawianie zmiennych programu

Wiele programów MySQL ma zmienne (parametry operacyjne), których wartość można ustawić. Jednym ze sposobów ustawienia zmiennej jest potraktowanie jej nazwy jako opcji. Na przykład, wywołanie `mysql` i przypisanie wartości 10 zmiennej `connect_timeout` odbywa się za pomocą poniższego polecenia:

```
% mysql --connect_timeout=10
```

Ta składnia pozwala także na zastąpienie w nazwach zmiennych znaków podkreślenia znakami myślnika, co powoduje, że opcja bardziej przypomina opcję:

```
% mysql --connect-timeout=10
```

W przypadku zmiennych przedstawiających wielkość bufora lub długość, wartości są podawane w bajtach, o ile nie dołączono żadnego przyrostka. Istnieje również możliwość dołączenia przyrostka K, M lub G do wartości i tym samym podania wartości w kilobajtach, megabajtach lub gigabajtach. Wielkość liter w przyrostkach nie ma znaczenia, więc równie dobrze można je podać jako k, m i g.

Zmienne programów zostały w tym dodatku wymienione w opisach poszczególnych programów, a ponadto są wyświetlane w komunikacie pomocy danej aplikacji (patrz podrozdział F.1, zatytułowany „Wyświetlanie komunikatu pomocy programu”).

F.2.2. Pliki opcji

Większość programów MySQL obsługuje pliki opcji. Dzięki wspomnianym plikom można przechowywać opcje programu bez konieczności ich ręcznego wprowadzania w wierszu poleceń podczas każdego uruchamiania programu. Każdą opcję podaną w pliku opcji można nadpisać opcją podaną w wierszu poleceń wraz z inną wartością. Wyjątek: `mysqld` używa *pierwszego* wystąpienia opcji `--user`, aby uniemożliwić nadpisanie jej wartości w pliku opcji przez podaną w wierszu poleceń.

Programy obsługujące pliki opcji szukają ich w kilku miejscach, ale brak pliku najczęściej nie powoduje wygenerowania łądów. Oznacza to konieczność samodzielnego utworzenia pliku opcji, jeśli chcesz z takiego korzystać. Plik opcji musi być zwykłym plikiem tekstowym. Dlatego też jeśli stworzysz go za pomocą procesora tekstów, upewnij się, że zapisałeś plik w formacie zwykłego tekstu, a nie w rodzimym formacie procesora tekstów.

W systemach UNIX pliki opcji znajdują się w miejscach wymienionych w tabeli F.2 i są odczytywane w przedstawianej kolejności, o ile w ogóle istnieją. Ponadto, jeśli nazwa pliku opcji zostanie podana wraz z `--defaults-extra-file`, wtedy zostanie on odczytany przed plikiem `~/my.cnf`.

Tabela F.2. Pliki opcji używane w systemie UNIX

Nazwa pliku	Opis
<code>/etc/my.cnf</code>	Opcje globalne.
<code>/etc/mysql/my.cnf</code>	Opcje globalne.
<code>SYSCONFDIR/my.cnf</code>	Opcje globalne.
<code>\$MYSQL_HOME/my.cnf</code>	Opcje charakterystyczne dla serwera.
<code>~/my.cnf</code>	Opcje charakterystyczne dla użytkownika.

Tylda (~) oznacza ścieżkę dostępu do katalogu domowego użytkownika. Wartość `SYSCONFDIR` pochodzi z opcji `-DSYSCONFDIR` podanej `CMake` podczas konfiguracji i kompilacji MySQL. Jej wartością domyślną jest katalog `etc` znajdujący się w katalogu instalacyjnym podanym w trakcie kompilacji. Z kolei `$MYSQL_HOME` to zmienna środowiskowa, która może być zdefiniowana do użycia przez `mysqld_safe` i wskazuje katalog zawierający plik opcji charakterystyczny dla danego serwera. Jeżeli wymieniona zmienna nie będzie zdefiniowana, wtedy skrypt `mysqld_safe` automatycznie spróbuje odszukać plik `my.cnf` w katalogu instalacji MySQL.

W systemie Windows pliki opcji znajdują się w miejscach wymienionych w tabeli F.3 i są odczytywane w przedstawianej kolejności, o ile w ogóle istnieją. Ponadto, jeśli nazwa pliku opcji zostanie podana wraz z `--defaults-extra-file`, zostanie on odczytany po plikach wymienionych w tabeli F.3.

Tabela F.3. Pliki opcji używane w systemie Windows

Nazwa pliku	Opis
<i>WINDIR\my.ini</i> , <i>WINDIR\my.ini</i>	Opcje globalne.
<i>C:\my.ini</i> , <i>C:\my.cnf</i>	Opcje globalne.
<i>INSTALLDIR\my.ini</i> , <i>INSTALLDIR\my.ini</i>	Opcje globalne.

WINDIR oznacza ścieżkę dostępu do katalogu Windows, natomiast *INSTALLDIR* to ścieżka dostępu do katalogu instalacji MySQL.

Opcje globalne są używane przez wszystkie programy MySQL obsługujące pliki opcji. Z kolei pliki charakterystyczne dla użytkownika są w systemach UNIX odczytywane przez programy uruchomione przez danego użytkownika.

Podczas używania plików opcji użytkownicy systemów Windows powinni zwrócić szczególną uwagę na wymienione poniżej kwestie:

- Komponenty ścieżki dostępu w Windows są rozdzielane znakami \, które MySQL traktuje jako znaki sterujące. Rozwiązaniem tego problemu dla opcji pobierających wartości w postaci ścieżki dostępu jest stosowanie ukośników / lub podwójnych znaków \\.
- W systemie Windows nazwy plików mogą być wyświetlane wraz z ukrytymi rozszerzeniami. Jeżeli utworzysz plik opcji o nazwie *my.cnf*, wyświetloną nazwą może być po prostu *my*. Warto w tym miejscu wspomnieć o ważnej kwestii — próba zmiany tej nazwy na *my.cnf* spowoduje, że plik opcji nie będzie działał. Powód jest prosty: w rzeczywistości spowodujesz zmianę nazwy pliku z *my.cnf* na *my.cnf.cnf*.

Kilka opcji powiązanych z przetwarzaniem pliku opcji jest standardowych wśród większości programów MySQL i ma przedstawione poniżej znaczenie. Jeśli zdecydujesz się na użycie którejkolwiek z nich, to musi być pierwsza opcja w wierszu poleceń. Wyjątkiem jest możliwość umieszczenia `--print-defaults` natychmiast po `--defaults-file` lub `--defaults-extra-file`.

- `--defaults-extra-file=file_name`

Opcja pozwala na podanie pliku opcji odczytywanego poza standardowymi plikami opcji. Ten plik jest odczytywany po odczycie wszystkich plików opcji globalnych i charakterystycznych dla serwera, ale jeszcze przed odczytaniem plików opcji charakterystycznych dla użytkownika. Podany tutaj plik musi istnieć i być możliwy do odczytu, w przeciwnym razie wystąpi błąd.

- `--defaults-file=file_name`

Opcja pozwala na podanie jednego pliku, z którego będą odczytywane opcje. Standardowo programy szukają plików opcji w kilku katalogach (zostały wcześniej wymienione), ale użycie opcji `--defaults-file` powoduje, że odczytywany będzie jedynie wymieniony plik. Podany tutaj plik musi istnieć i być możliwy do odczytu, w przeciwnym razie wystąpi błąd.

- `--defaults-group-suffix=suffix`

Opcja pozwala na odczyt grupy opcji o standardowych nazwach oraz tych, których standardowe nazwy połączono ze wskazanym przyrostkiem.

- `--no-defaults`

Ta opcja powoduje wyłączenie użycia jakiegokolwiek pliku opcji. Ponadto, powoduje także brak obsługi innych opcji związanych z przetwarzaniem plików opcji, na przykład `--defaults-file`.

- `--print-defaults`

Wyświetla wartości opcji, które będą używane w przypadku uruchomienia programu bez podania opcji w wierszu poleceń. Omawiana opcja wyświetla wartości odczytane z plików opcji (oraz zmiennych środowiskowych). Okazuje się niezwykle użyteczna do sprawdzania poprawności konfiguracji pliku opcji. Ponadto, przydaje się, jeśli podejrzewasz, że program MySQL używa opcji, których nigdy nie podałeś. W takim przypadku wystarczy użyć opcji `--print-defaults` i przekonać się, które opcje są odczytywane z pliku opcji.

Wyświetlany przez program komunikat pomocy zawiera także listę katalogów sprawdzanych przez program w poszukiwaniu plików opcji (patrz także podrozdział F.1, zatytułowany „Wyświetlanie komunikatu pomocy programu”). Na domyślny zestaw odczytywanych plików wpływ ma użycie opcji `--defaults-file`, `--defaults-extra-file` i `--no-defaults`.

Opcje w plikach opcji są pogrupowane (tworzą sekcje). Poniżej przedstawiono przykładowy plik opcji:

```
[client]
user=sampadm
password=secret

[mysql]
skip-auto-rehash

[mysqlshow]
status
```

Nazwy grup są umieszczane w nawiasach kwadratowych i nie rozróżniają wielkości liter. Grupa specjalna `[client]` pozwala na podanie opcji, które będą stosowane przez wszystkie programy klienckie. Nazwy pozostałych grup muszą odpowiadać nazwom programów klienckich. W przedstawionym powyżej przykładzie `[mysql]` wskazuje grupę opcji dla klienta `mysql`, natomiast `[mysqlshow]` wskazuje grupę opcji dla `mysqlshow`. Standardowe programy klienckie MySQL sprawdzają zarówno grupę `[client]`, jak i grupę o takiej samej nazwie jak nazwa klienta. Na przykład, `mysql` sprawdza grupy `[client]` i `[mysql]`, natomiast `mysqlshow` sprawdza grupy `[client]` i `[mysqlshow]`.

Zachowaj ostrożność podczas umieszczania w grupie `[client]` opcji obsługiwanych tylko przez jednego klienta. Na przykład, opcja `skip-auto-rehash` jest charakterystyczna dla `mysql`. Jeżeli umieścisz tę opcję w grupie `[client]`, przekonasz się, że inne programy klienckie, takie jak `mysqlimport`, nie działają. (Próba ich uruchomienia zakończy się

wyświetleniem komunikatu błędu wraz z komunikatem pomocy programu). W takim przypadku opcję `skip-auto-rehash` należy umieścić w grupie `[mysql]`.

Wszystkie opcje umieszczone po nazwie grupy są z nią powiązane. Plik opcji może zawierać dowolną liczbę grup, a grupy znajdujące się dalej mają pierwszeństwo przed wcześniejszymi. Jeżeli dana opcja będzie użyta kilkakrotnie w grupach sprawdzanych przez program, zastosowana będzie wartość wymieniona jako ostatnia. Wyjątek: `mysql` używa *pierwszego* wystąpienia opcji `--user`, aby uniemożliwić nadpisanie jej wartości w pliku opcji przez podaną w wierszu poleceń.

Każda opcja powinna być umieszczona w oddzielnym wierszu. Pierwsze słowo w wierszu jest nazwą opcji w długim formacie, bez myślników na początku. Na przykład, w celu wskazania w wierszu poleceń użycia kompresji należy podać opcję `-C` lub `--compress`, natomiast w pliku opcji podaje się jedynie `compress`. Każda opcja w formacie długiej nazwy obsługiwana przez program może być umieszczona w pliku opcji. Jeżeli opcja wymaga podania wartości, nazwę i wartość trzeba rozdzielić znakiem równości.

Spójrz na poniższe polecenie wydawane w wierszu poleceń:

```
% mysql --compress --user=sampadm --max_allowed_packet=16M
```

Aby te same informacje podać w pliku opcji za pomocą grupy `[mysql]`, należy je zdefiniować następująco:

```
[mysql]
compress
user=sampadm
max_allowed_packet=16M
```

Wartość opcji można ująć w cudzysłów lub apostrofy. To jest użyteczne rozwiązanie, jeśli wartość zawiera spację.

W wierszu pliku opcji spacje znajdujące się na początku wiersza są ignorowane, podobnie jak spacje wokół znaku równości rozdzielające nazwę opcji i jej wartość. Wiersze puste bądź też rozpoczynające się od znaku `#` lub `;` są traktowane jako komentarz i ignorowane. Komentarz może rozpoczynać się także w środku wiersza, ale wówczas trzeba użyć znaku `#` (nie można zaś znaku `;`).

W celu wskazania w wartościach pliku opcji znaków specjalnych można użyć znaków sterujących wymienionych w tabeli F.4.

Tabela F.4. Znaki sterujące, które mogą być używane w plikach opcji

Sekwencja	Opis
<code>\b</code>	Backspace.
<code>\n</code>	Znak nowego wiersza (wysuw wiersza).
<code>\r</code>	Znak powrotu na początek wiersza.
<code>\s</code>	Spacja.
<code>\t</code>	Tabulator.
<code>\\</code>	Backslash.

Pliki opcji mogą zawierać dyrektywy pozwalające na odczyt innych plików opcji:

- `!include file_name`

Opcja powoduje odczyt wskazanego pliku opcji.

- `!includedir dir_name`

Opcja powoduje odczyt wszystkich plików opcji znajdujących się we wskazanym katalogu. Pliki opcji są identyfikowane za pomocą rozszerzenia `.cnf` (systemy UNIX) bądź też `.ini` lub `.cnf` (system Windows). Kolejność odczytu plików jest niezdefiniowana.

Dołączone pliki stosują standardową składnię plików opcji. Użyte zostaną opcje jedynie z grupy opcji, w której nastąpiło dołączenie pliku.

F.2.2.1. Zachowanie prywatności opcji danego użytkownika

W systemach UNIX plik opcji dla danego użytkownika to `.my.cnf` i znajduje się w katalogu domowym. Właścicielem tego pliku powinien być dany użytkownik, a uprawnienia pliku należy zdefiniować jako 600 lub 400, aby inni użytkownicy nie mogli go odczytywać. Nie chcesz przecież udostępnić innym użytkownikom nazwy użytkownika konta w serwerze MySQL i zdefiniowanego dla niego hasła. Prywatność pliku opcji można sobie zapewnić, wykonując jedno z poniższych poleceń w katalogu domowym:

```
% chmod 600 .my.cnf
% chmod go-rwx .my.cnf
```

To samo dotyczy pliku historii tworzonego przez program kliencki `mysql` — wspomniany plik to `mysql_history` i znajduje się w katalogu domowym użytkownika. Ten plik powinien być chronić tak samo, jak plik opcji.

F.2.2.2. Użycie narzędzia `my_print_defaults` w celu sprawdzenia opcji

Narzędzie `my_print_defaults` jest użyteczne w celu ustalenia opcji odczytywanych przez program z pliku opcji. Wymienione narzędzie wyszukuje pliki opcji i wyświetla te, które są zdefiniowane dla jednej lub więcej grup opcji. Na przykład, program `mysql` używa opcji z grup `[client]` i `[mysql]`. Aby dowiedzieć się, które opcje zdefiniowane w plikach opcji będą stosowane w `mysql`, należy omawiane narzędzie wywołać w następujący sposób:

```
% my_print_defaults client mysql
```

Podobnie, serwer `mysqld` używa opcji w grupach `[mysqld]` i `[server]`. W celu sprawdzenia opcji zdefiniowanych dla niego w plikach opcji należy wydać poniższe polecenie:

```
% my_print_defaults mysqld server
```


F.2.3. Zmienne środowiskowe

Programy MySQL sprawdzają wartości w wielu zmiennych środowiskowych w celu pobrania wartości dla opcji. Wspomniane zmienne środowiskowe mają niski priorytet, określone za ich pomocą wartości opcji mogą być nadpisane przez opcje podane w pliku opcji lub wierszu poleceń.

Programy MySQL sprawdzają następujące zmienne środowiskowe:

- **LANG, LC_ALL**

Jeżeli którakolwiek z wymienionych zmiennych środowiskowych została ustawiona i wskazuje ustawienia językowe, programy klienckie MySQL będą używały tej wartości do zdefiniowania domyślnego kodowania znaków. To jak użycie opcji `--default-character-set`.

- **MYSQL_DEBUG**

Opcje do użycia podczas debugowania. Ta zmienna nie ma żadnego efektu, o ile serwer MySQL nie został skompilowany wraz z włączoną obsługą debugowania. Ustawienie zmiennej `MYSQL_DEBUG` jest jak użycie opcji `--debug`.

- **MYSQL_PWD**

Hasło używane podczas nawiązywania połączenia z serwerem MySQL. Ustawienie zmiennej `MYSQL_PWD` jest jak użycie opcji `--password`.

Użycie zmiennej `MYSQL_PWD` do przechowywania hasła stanowi ryzyko, ponieważ inni użytkownicy systemu mogą bardzo łatwo poznać wartość wymienionej zmiennej. Na przykład, w niektórych systemach narzędzie ps wyświetla ustawienia zmiennych środowiskowych dla innych użytkowników.

- **MYSQL_TCP_PORT**

W przypadku serwera `mysqld` ta zmienna wskazuje port, na którym serwer powinien nasłuchiwać połączeń TCP/IP. Z kolei dla programów klienckich to jest numer portu używany podczas nawiązywania połączeń TCP/IP z serwerem. Ustawienie zmiennej `MYSQL_TCP_PORT` jest jak użycie opcji `--port`.

- **MYSQL_UNIX_PORT**

W przypadku serwera `mysqld` ta zmienna wskazuje plik gniazda, który serwer powinien sprawdzać, oczekując na połączenia lokalne. Z kolei dla programów klienckich to jest ścieżka dostępu do pliku gniazda systemu UNIX używanego podczas nawiązywania połączeń lokalnych z serwerem działającym jako `localhost`. Ustawienie zmiennej `MYSQL_UNIX_PORT` jest jak użycie opcji `--socket`.

- **TMPDIR**

To jest ścieżka dostępu do katalogu, w którym powinny być tworzone pliki tymczasowe. Ustawienie omawianej zmiennej jest jak użycie opcji `--tmpdir`, za wyjątkiem faktu, że należy podać nazwę tylko jednego katalogu.

- **USER**

Nazwa użytkownika konta używanego do nawiązywania połączenia z serwerem. Ta zmienna jest używana jedynie przez programy klienckie w systemie Windows. Jej ustawienie jest jak użycie opcji `--user`.

Klient `mysql` sprawdza wartość jeszcze trzech dodatkowych zmiennych środowiskowych:

■ **MYSQL_HISTFILE**

W systemach UNIX to nazwa pliku używanego do przechowywania historii w trakcie interaktywnego użycia klienta. Jeżeli ta zmienna nie została ustawiona, domyślnie będzie używany plik `.mysql_history`, znajdujący się w katalogu domowym.

■ **MYSQL_HOST**

Komputer zawierający działający serwer MySQL, z którym będzie nawiązane połączenie. Ustawienie omawianej zmiennej jest jak użycie opcji `--host`.

■ **MYSQL_PS1**

Ciąg tekstowy używany zamiast `mysql>` jako podstawowy ciąg tekstowy. Ten ciąg tekstowy może zawierać specjalne sekwencje sterujące wymienione w punkcie F.4.5, zatytułowanym „Sekwencje definiujące znak zachęty `mysql`”.

F.3. Narzędzie `myisamchk`

Narzędzie `myisamchk` przeprowadza operacje związane z obsługą tabel MyISAM. (Nie działa z tabelami partycjonowanymi; w takim przypadku należy użyć `mysqlcheck`). Omawiane narzędzie sprawdza i naprawia uszkodzone tabele, wyświetla informacje o tabeli, przeprowadza analizę rozproszenia wartości kluczy indeksu, a także włącza i wyłącza indeksy.

Tabele zarządzane przez silnik bazy danych MyISAM mają pliki danych i indeksów o rozszerzeniach odpowiednio `.myd` i `.myi`. Jeżeli spróbujesz użyć `myisamchk` względem tabeli niewłaściwego typu, narzędzie wyświetli odpowiedni komunikat ostrzeżenia i zignoruje tabelę.

Narzędzie `myisamchk` należy wywołać wraz z nazwami tabel przeznaczonymi do sprawdzenia

```
myisamchk [options] tbl_name[.myi] ...
```

W przypadku braku opcji, `myisamchk` sprawdzi wskazaną tabelę pod kątem błędów. W przeciwnym razie przetworzy ją zgodnie z podanymi opcjami. Jeżeli przeprowadzasz operację, która może modyfikować tabelę, dobrym rozwiązaniem będzie utworzenie najpierw jej kopii zapasowej.

Argument wskazujący tabelę może być nazwą tabeli (`tbl_name`) lub indeksu (`tbl_name.myi`). Wymienione poniżej polecenia są równoważne:

```
% myisamchk member
% myisamchk member.myi
```

Użycie nazw plików indeksów jest wygodne, jeśli interpreter poleceń rozwija wzorce nazw plików. Aby sprawdzić wszystkie tabele MyISAM w katalogu bieżącym, można użyć znaku wieloznacznego wskazującego wszystkie pliki indeksu:

```
% myisamchk *.myi
```

Narzędzie `mysamchk` nie wie, gdzie znajdują się pliki tabel. Jeżeli nie ma ich w katalogu bieżącym, trzeba podać ścieżkę dostępu do tabel. Ponieważ narzędzie nie zakłada, że pliki tabel znajdują się w katalogu danych serwera MySQL, można je skopiować do innego katalogu, a następnie operację przeprowadzić na kopiach zamiast oryginałach.

Wiele operacji `mysamchk` może zostać przeprowadzonych również przez wykonanie zapytań SQL do serwera. Wspomniane zapytania obejmują `ANALYZE TABLE`, `CHECK TABLE`, `OPTIMIZE TABLE` i `REPAIR TABLE`. Te zapytania można wykonać bezpośrednio lub użyć programu `mysamcheck`, który zapewnia interfejs wiersza poleceń służący do wydania wielu zapytań SQL związanych z obsługą tabel. Ogólnie rzecz biorąc, łatwiej i bezpieczniej jest używać wymienionych zapytań lub `mysqlcheck` zamiast `mysamchk`. Więcej informacji znajdziesz w podrozdziale 14.6, zatytułowanym „Sprawdzanie i naprawianie tabel bazy danych”.

Warto pamiętać o jednym: podczas używania narzędzia `mysamchk` do przeprowadzania operacji na tabelach należy uniemożliwić serwerowi uzyskiwanie do nich dostępu. To jest niezbędne, ponieważ serwer i narzędzie `mysamchk` uzyskują bezpośredni dostęp do plików tabel. Jeżeli jednocześnie uzyskają dostęp do plików, wtedy mogą zniszczyć tabelę. Przed użyciem `mysamchk` warto zajrzeć do podrozdziału 14.2, zatytułowanego „Obsługa bazy danych w działającym serwerze”, w którym omówiono sposoby uniemożliwienia serwerowi dostępu do tabel, gdy są używane przez narzędzie `mysamchk`.

Szczególną ostrożność należy zachować podczas używania narzędzia `mysamchk` wraz z tabelami zawierającymi indeksy typu `FULLTEXT`, gdy oba wymienione poniżej warunki są prawdziwe:

- Narzędzia `mysamchk` używasz do przeprowadzania operacji modyfikującej indeksy. To obejmuje operacje analizy i naprawy.
- Korzystasz z serwera używającego niedomyślnych wartości dowolnych zmiennych systemowych powiązanych z indeksami typu `FULLTEXT`:
`ft_max_word_len`, `ft_min_word_len` lub `ft_stopword_file`.

Kiedy oba wymienione warunki są prawdziwe, należy użyć odpowiednich opcji narzędzia `mysamchk` i wskazać mu właściwe parametry indeksu typu `FULLTEXT`, ponieważ narzędzie nie zna wartości stosowanych przez serwer. Jeżeli tego nie zrobisz, `mysamchk` utworzy indeksy typu `FULLTEXT`, używając innych wartości parametrów niż oczekiwane przez serwer, a operacja wyszukiwania pełnego tekstu będzie zwracała nieprawidłowe wyniki. Przyjmujemy założenie, że serwer używa przedstawionych poniżej niedomyślnych ustawień opcji dla minimalnej długości słowa oraz pliku słów przestankowych:

```
[mysqld]
ft_min_word_len=2
ft_stopword_file=/var/mysql/data/my-stopwords
```

W takim przypadku te same wartości należy wskazać narzędziu `mysamchk` podczas przeprowadzania wszelkich operacji zmiany indeksu w tabelach zawierających indeksy typu `FULLTEXT`. Można to zrobić za pomocą opcji wiersza poleceń (`--ft_min_word_len` i `--ft_stopword_list`), ale lepszym rozwiązaniem jest ich umieszczenie w pliku opcji, aby nie zapomnieć o ich użyciu. Skorzystaj z grupy opcji podobnej do używanej przez serwer:

```
[myisamchk]
ft_min_word_len=2
ft_stopword_file=/var/mysql/data/my-stopwords
```

Aby całkowicie uniknąć problemu z parametrem FULLTEXT, obsługę tabel przeprowadzaj za pomocą zapytań SQL, takich jak REPAIR TABLE lub ANALYZE TABLE. W takim przypadku modyfikacje będą przeprowadzane przez serwer. Ponieważ serwer zna używane parametry indeksów typu FULLTEXT, zastosuje je podczas operacji obsługi tabel zawierających tego typu indeksy.

F.3.1. Opcje standardowe obsługiwane przez myisamchk

```
--character-sets-dir  --help                --verbose
--debug              --silent              --version
```

OGólny opis powyższych opcji znajdziesz w punkcie F.2.1, zatytułowanym „Opcje standardowe programu MySQL”.

Opcja --print oznacza, że wyświetlane będą jedynie komunikaty błędów. Opcja --verbose wyświetla większą ilość informacji, jeśli zostanie użyta wraz z --check, --description lub --extend-check. W celu osiągnięcia większego efektu opcje --silent i --verbose mogą być podane kilkakrotnie.

Standardowa opcja --help powoduje wyświetlenie komunikatu pomocy wraz z opcjami pogrupowanymi wedle funkcji. Narzędzie myisamchk obsługuje również opcje --HELP i -H wyświetlające wszystkie opcje w kolejności alfabetycznej na pojedynczej liście.

F.3.2. Opcje charakterystyczne dla myisamchk

Pewne opcje odwołują się do numerów indeksu. Wspomniane numery rozpoczynają się od 1. Aby ustalić numerowanie indeksu dla określonej tabeli, należy wykonać zapytanie SHOW INDEX lub użyć polecenia mysql show --keys. Kolumna Key_name w danych wyjściowych wyświetla indeksy w tej samej kolejności, w której są widziane przez myisamchk.

■ --analyze, -a

Opcja pozwala na przeprowadzenie analizy rozproszenia kluczy. To może pomóc serwerowi w znacznie szybszym przeprowadzaniu operacji wyszukiwania na podstawie indeksu oraz złączeń. Aby uzyskać informacje o poziomie rozproszenia kluczy po analizie, należy uruchomić narzędzie myisamchk wraz z opcjami --description i --verbose.

■ --backup, -B

W przypadku opcji modyfikujących plik danych (.myd) omawiana opcja powoduje utworzenie dla pliku jego kopii zapasowej w postaci *tbl_name-time.bak*. Człon *time* przedstawia znacznik czasu. Narzędzie myisamchk tworzy plik kopii zapasowej w katalogu, w którym znajdują się pliki tabeli.

■ `--block-search=n, -b n`

Opcja powoduje wyświetlenie rekordu początkowego tabeli zawierającego blok startowy w bloku *n*. Omawiana opcja jest wykorzystywana jedynie podczas debugowania.

■ `--check, -c`

Sprawdzenie tabeli pod kątem błędów. To jest domyślne działanie, jeśli nie zostanie podana żadna opcja.

■ `--check-only-changed, -C`

Sprawdzanie tabel tylko wtedy, jeśli zostały zmodyfikowane od chwili ostatniego sprawdzenia.

■ `--correct-checksum`

W przypadku tabel utworzonych wraz z opcją `CHECKSUM = 1` omawiana opcja gwarantuje zachowanie poprawności informacji o sumie kontrolnej w tabeli.

■ `--data-file-length=n, -D n`

Wyrażona w bajtach maksymalna wielkość, do której może wzrosnąć plik danych podczas jego ponownego tworzenia. (Taka sytuacja zdarza się, gdy plik osiąga limit narzucony przez MySQL lub przez ograniczenia dotyczące wielkości pliku nakładane przez system operacyjny. Powodem wystąpienia wspomnianej sytuacji może być również osiągnięcie maksymalnej liczby rekordów określonej przez wewnętrzne struktury danych). Omawiana opcja jest efektywna jedynie podczas jej używania w połączeniu z `--recover` lub `--safe-recover`.

■ `--description, -d`

Wyświetlenie opisowych informacji o tabeli.

■ `--extend-check, -e`

Przeprowadzenie rozszerzonej operacji sprawdzania tabeli. Rzadko powinna występować potrzeba użycia tej opcji, ponieważ narzędzie `myisamchk` zwykle znajduje wszelkie błędy w trakcie pracy w jednym z mniej dokładnych trybów działania.

■ `--fast, -F`

Opcja powoduje sprawdzenie tabel tylko wtedy, gdy nie zostały prawidłowo zamknięte. Taka sytuacja może się zdarzyć na przykład po wystąpieniu awarii komputera serwera, w chwili gdy `mysqld` miał otwarte tabele, a tym samym serwer `mysqld` nie miał możliwości ich zamknięcia.

■ `--force, -f`

Opcja wymusza sprawdzenie lub naprawę tabeli nawet w przypadku istnienia pliku tymczasowego dla tabeli. Normalnie, jeśli narzędzie `myisamchk` znajdzie plik `tbl_name.tmd`, po prostu wyświetla komunikat błędu i kończy pracę, ponieważ wymieniony plik może wskazywać na działanie innego egzemplarza narzędzia. Jednak plik tymczasowy może istnieć również z powodu zamknięcia

poprzedniego egzemplarza `mysamchk` w trakcie jego działania i wtedy wspomniany plik tymczasowy można bezpiecznie usunąć. Jeżeli jesteś pewien, że masz do czynienia z tego typu sytuacją, to użyj opcji `--force`, wymuszając tym samym uruchomienie `mysamchk` pomimo istnienia pliku tymczasowego. (Alternatywnym rozwiązaniem jest ręczne usunięcie pliku tymczasowego). Jeżeli użyjesz opcji `--force` podczas sprawdzania tabel, narzędzie `mysamchk` automatycznie ponownie uruchomi się wraz z opcją `--recover` dla każdej tabeli, w której zostały znalezione problemy. Ponadto, `mysamchk` uaktualnia stan tabeli dokładnie w ten sam sposób, jak robi to opcja `--update-state`.

■ `--information, -i`

Opcja powoduje wyświetlenie informacji statystycznych dotyczących zawartości tabeli.

■ `--keys-used=n, -k n`

Opcja używana wraz z `--recover`. Wartość *n* jest mapą bitową wskazującą używane indeksy. Pierwszy indeks to bit zero. (Na przykład, wartość 6 to dwójkowo 110 i wskazuje użycie drugiego oraz trzeciego indeksu). Wartość 0 powoduje wyłączenie wszystkich indeksów. Takie rozwiązanie może być użyte w celu poprawienia wydajności operacji INSERT, DELETE i UPDATE. Włączenie indeksów z powrotem przywraca zwykle zachowanie indeksowania (należy podać maskę bitową zawierającą włączone bity dla wszystkich indeksów).

■ `--max-record-length=n`

Opcja powoduje zignorowanie rekordów większych niż *n* bajtów, jeśli nie można dla nich zaalokować pamięci.

■ `--medium-check, -m`

Sprawdzenie tabel za pomocą metody szybszej niż `--extended-check`, ale nieco mniej dokładnej. Ten tryb sprawdzania powinien być wystarczający w większości przypadków. Sprawdzenie odbywa się przez obliczenie wartości CRC dla kluczy w indeksie oraz porównanie ich z wartościami CRC obliczonymi w pliku danych na podstawie zindeksowanych kolumn.

■ `--parallel-recover, -p`

Naprawa zostanie przeprowadzona w ten sam sposób jak w przypadku opcji `--recover`, ale ponowne utworzenie indeksów będzie następowało z wykorzystaniem wielu wątków. Takie rozwiązanie jest szybsze niż użycie tylko jednego wątku, ale należy uznawać je za eksperymentalne.

■ `--quick, -q` (*boolean*)

Ta opcja jest używana w połączeniu z `--recover` i przeprowadza szybszą naprawę niż przy użyciu jedynie opcji `--recover`. Podanie obu wymienionych opcji powoduje, że plik danych nie będzie używany. Aby wymusić na programie modyfikację pliku danych w przypadku znalezienia powielających się wartości kluczy, należy dwukrotnie podać opcję `--quick`.

- `--read-only, -T`

Opcja powoduje, że tabele nie będą oznaczone jako sprawdzone.

- `--recover, -r`

Przeprowadzenie standardowej operacji naprawy. Pozwala ona na usunięcie większości problemów, poza występowaniem powtarzających się wartości indeksu, który powinien zawierać tylko unikalne wartości.

- `--safe-recover, -o`

Przeprowadzenie naprawy za pomocą metody wolniejszej niż po użyciu opcji `--recover`, ale pozwalającej na usunięcie pewnych problemów, których nie usunie opcja `--recover`. Opcja `--safe-recover` wymaga także mniejszej ilości miejsca na dysku niż `--recover`.

- `--set-auto-increment [=n], -A[n]`

Ustawienie licznika `AUTO_INCREMENT` w taki sposób, aby kolejne wartości sekwencji rozpoczynały się od n (lub większej wartości, jeśli tabela zawiera już rekordy o wartościach `AUTO_INCREMENT` o wielkości n). Jeżeli nie zostanie podana wartość n , omawiana opcja powoduje ustawienie wartości `AUTO_INCREMENT` jako o jeden większej niż bieżąca wartość maksymalna przechowywana w tabeli.

W przypadku podania wartości n po `-A` między nimi nie może znajdować się spacja, ponieważ wtedy wartość zostanie zinterpretowana nieprawidłowo.

Aby ustawić wartość `AUTO_INCREMENT` dla tabeli `MyISAM` bez konieczności użycia narzędzia `myisamchk`, należy wykonać zapytanie w poniższej postaci:

```
ALTER TABLE tbl_name AUTO_INCREMENT = n;
```

- `--set-collation=collation`

Kolejność sortowania używana podczas ponownego tworzenia i sortowania elementów indeksu tabeli. Nazwa kolejności sortowania wskazuje również nazwę kodowania znaków.

- `--sort-index, -S`

Opcja powoduje sortowanie bloków indeksu w celu przyspieszenia sekwencyjnego odczytu bloków w kolejnych operacjach odczytu.

- `--sort-records=n, -R n`

Opcja przeprowadza sortowanie rekordów danych zgodnie z kolejnością, w jakiej są wymienione w indeksie n . Kolejne oparte na indeksie operacje pobierania danych powinny być szybsze. Pierwsze przeprowadzenie tego rodzaju operacji na tabeli może być bardzo wolne, ponieważ rekordy będą nieuporządkowane. Zapytanie `ALTER TABLE ... ORDER BY` wykonuje takie samo zadanie jak `--sort-records`, ale zwykle szybciej.

- `--sort-recover, -n`

Opcja wymusza naprawę, nawet jeśli plik tymczasowy niezbędny do przeprowadzenia operacji sortowania może stać się ogromny.

- `--start-check-pos=n`

Położenie *n*, od którego rozpocznie się odczyt w pliku danych. Ta opcja jest używana jedynie podczas debugowania.

- `--tmpdir=dir_name, -t dir_name`

Ścieżka dostępu do katalogu używanego przez pliki tymczasowe. Wartością domyślną tej opcji jest wartość zmiennej środowiskowej `TMPDIR` lub katalog `/tmp`, jeśli wymieniona zmienna nie została ustawiona. Wartością opcji może być również lista katalogów. W systemach UNIX nazwy katalogów powinny być rozdzielone dwukropkami, natomiast w Windows średnikami.

- `--unpack, -u`

Rozpakowanie tabeli, która została spakowana za pomocą narzędzia `mysampack`. Ta opcja może być używana do konwersji skompresowanej tabeli tylko do odczytu na postać umożliwiającą modyfikację tabeli. Nie może być używana wraz z opcjami `--quick` lub `--sort-records`.

- `--update-state, -U`

Uaktualnienie wewnętrznej flagi przechowywanej w tabeli w celu określania jej stanu. Tabele prawidłowe są oznaczone jako dobre, natomiast tabele z błędami są oznaczone jako wymagające naprawy. Użycie omawianej opcji powoduje, że kolejne wywołania `mysamchk` wraz z opcją `--check-only-changed` są znacznie efektywniejsze w przypadku tabel niewymagających naprawy.

- `--wait, -w`

Jeżeli tabela ma nałożoną blokadę, ta opcja powoduje czekanie narzędzia, aż tabela stanie się dostępna. Bez opcji `--wait` narzędzie `mysamchk` czeka 10 sekund na możliwość nałożenia blokady, a następnie wyświetla komunikat błędu, jeśli nałożenie blokady zakończyło się niepowodzeniem.

F.3.3. Zmienne narzędzia `mysamchk`

Wymienione tutaj zmienne narzędzia `mysamchk` mogą być ustawione za pomocą poleceń podanych w podpunkcie F.2.1.2, zatytułowanym „Ustawianie zmiennych programu”.

W przypadku tabel zawierających indeksy typu `FULLTEXT` należy zwrócić uwagę na ostrzeżenie przedstawione na początku opisu programu `mysamchk`.

- `decode_bits`

Liczba bitów używanych podczas dekodowania skompresowanych tabel. Większe wartości mogą oznaczać szybszą operację, ale kosztem większego zużycia pamięci. Ogólnie rzecz biorąc, wartość domyślna, wynosząca 9, jest wystarczająca.

- `ft_max_word_len`

Maksymalna długość słów, które mogą znajdować się w indeksach typu `FULLTEXT`. Dłuższe słowa będą ignorowane. Wartość domyślna wynosi 84.

- `ft_min_word_len`
Minimalna długość słów, które mogą znajdować się w indeksach typu FULLTEXT. Krótsze słowa będą ignorowane. Wartość domyślna wynosi 4.
- `ft_stopword_file`
Plik słów przestankowych dla indeksów typu FULLTEXT. Domyślnie używana jest wbudowana lista tego rodzaju słów.
- `key_buffer_size`
Wielkość bufora używanego dla bloków indeksu. (Będzie wykorzystywany wraz z opcją `--safe-recover`, ale już nie z `--recover` lub `--sort-recover`). Domyślnie to 512 KB.
- `key_cache_block_size`
Wielkość bloków w buforze kluczy. Wartość domyślna wynosi 1 MB.
- `myisam_block_size`
Wielkość bloku używanego w plikach indeksu pliku `.myi`. Wartość domyślna wynosi 1 MB.
- `read_buffer_size`
Wielkość bufora odczytu. Wartość domyślna wynosi 256 KB.
- `sort_buffer_size`
Wielkość bufora używanego podczas operacji sortowania wartości kluczy. (Będzie wykorzystywany wraz z opcją `--recover` lub `--sort-recover`, ale już nie z `--safe-recover`). Domyślnie to 2 MB.
- `sort_key_blocks`
Ta zmienna jest powiązana z głębokością struktury B-tree używanej przez indeks. Wartość domyślna wynosi 16 i nie powinna występować potrzeba jej zmiany.
- `stats_method`
Ta zmienna określa, czy serwer powinien uznawać wartości NULL za takie same, czy inne podczas obliczania danych statystycznych dotyczących rozproszenia wartości klucza. Zmiennej można przypisać wartość `nulls_equal` (wszystkie wartości NULL znajdują się w tej samej grupie), `nulls_unequal` (każda wartość NULL tworzy oddzielną grupę) lub `nulls_ignored` (wartości NULL są ignorowane).
- `write_buffer_size`
Wielkość bufora zapisu. Wartość domyślna wynosi 256 KB.

F.4. mysql

Program kliencki `mysql` umożliwia nawiązanie połączenia z serwerem, wykonywanie zapytań SQL oraz przeglądanie ich wyników.

```
mysql [options] [db_name]
```

Jeżeli podany zostanie argument *db_name*, wskazana baza danych stanie się domyślną w trakcie sesji. W przypadku braku wymienionego argumentu program *mysql* zostanie uruchomiony bez domyślnej bazy danych i trzeba będzie stosować w pełni kwalifikowane nazwy tabel (wraz z nawami baz danych) lub wykonać zapytanie *USE db_name* w celu wskazania domyślnej bazy danych.

Program *mysql* może być uruchomiony w trybie interaktywnym. Inną możliwością jest jego uruchomienie w trybie wsadowym, co pozwala na wykonanie zapytań zapisanych w pliku. Wymaga to przekierowania danych wejściowych polecenia w taki sposób, aby były odczytywane z wspomnianego pliku, na przykład:

```
% mysql -u sampadm -p -h cobra.example.com sampdb < my_sql_file
```

Kiedy program *mysql* jest używany w trybie interaktywnym, wtedy wyświetla znak zachęty *mysql>*, wskazujący oczekiwanie na dane wejściowe. W celu wykonania zapytania należy je wprowadzić (jeśli zachodzi potrzeba, można to zrobić w wielu wierszach), a następnie wskazać koniec zapytania za pomocą średnika lub *\g*. Program *mysql* wysyła zapytanie do serwera, wyświetla wynik jego wykonania, a następnie ponownie wyświetla znak zachęty, oznaczający gotowość do przyjęcia kolejnego zapytania. Polecenie *\G* również oznacza koniec zapytania, ale powoduje „pionowe” sformatowanie danych wyjściowych (to znaczy umieszczenie po jednej kolumnie w każdym wierszu danych wyjściowych).

Program *mysql* stosuje różne znaki zachęty, wskazując dane, na które oczekuje (patrz tabela F.5). Podstawowy znak zachęty to *mysql>*, wyświetlany na początku każdego zapytania. Pozostałe znaki zachęty są drugorzędne, wyświetlane w celu pobrania dodatkowych wierszy bieżącego zapytania.

Tabela F.5. Znaki zachęty stosowane przez program *mysql*

Znak zachęty	Opis
<i>mysql></i>	Oczekiwanie na pierwszy wiersz nowego zapytania.
<i>-></i>	Oczekiwanie na kolejny wiersz bieżącego zapytania.
<i>'></i>	Oczekiwanie na dokończenie w bieżącym zapytaniu ciągu tekstowego ujętego w apostrofy.
<i>"></i>	Oczekiwanie na dokończenie w bieżącym zapytaniu ciągu tekstowego ujętego w cudzysłów.
<i>`></i>	Oczekiwanie na dokończenie w bieżącym zapytaniu cytowanego identyfikatora.
<i>/*></i>	Oczekiwanie na dokończenie komentarza w postaci <i>/* ... */</i> .

Znaki zachęty *'>* i *">* wskazują na rozpoczęcie w poprzednim wierszu ciągu tekstowego ujętego odpowiednio w apostrofy lub cudzysłów i niewprowadzenie dotąd zamykającego znaku cytowania. Podobnie, znak zachęty *`>* oznacza brak znaku zamykającego cytowany identyfikator. Z kolei */** wskazuje początek komentarza typu */* ... */*, który nie został jeszcze zakończony. Kiedy widzisz wymienione znaki zachęty, najczęściej oznacza to, że

zapomniałeś zakończyć cytowanie ciągu tekstowego, identyfikatora lub umieścić sekwencję kończącą komentarz. Jeśli faktycznie tak jest, aby wyjść z trybu pobierania ciągu tekstowego, wprowadź odpowiedni znak sterujący lub sekwencję kończącą komentarz, a następnie `\c` w celu przerwania bieżącego zapytania.

W systemach UNIX, gdy program `mysql` jest używany w trybie interaktywnym, zapytania są zapisywane w pliku historii. Domyślna nazwa wspomnianego pliku to `.mysql_history`; znajduje się on w katalogu domowym i można go wyraźnie zdefiniować za pomocą zmiennej środowiskowej `MYSQL_HISTFILE`. Plik historii powinien być dostępny jedynie dla użytkownika, który go utworzył, inni nie powinni mieć możliwości przeglądania jego zawartości — patrz podpunkt F.2.2.1, zatytułowany „Zachowanie prywatności opcji danego użytkownika”. Jeżeli nie chcesz korzystać z pliku historii, usuń go, a następnie utwórz dowiązanie symboliczne o nazwie `.mysql_history` prowadzące do `/dev/null` lub przypisz zmiennej `MYSQL_HISTFILE` wartość `/dev/null`.

Pewne opcje powodują wyłączenie użycia pliku historii. Ogólnie rzecz biorąc, to są opcje wskazujące na interaktywne użycie programu `mysql`, na przykład `--batch`, `--html` lub `--quick`.

W systemach obsługujących bibliotekę edycji danych wejściowych zapytania mogą być ponownie wywoływane z historii poleceń i wykonywane po dodatkowej edycji lub bez niej. W systemie Windows program `mysql` nie umożliwia edycji wiersza danych wejściowych, ale sam system Windows oferuje kilka poleceń edycji, które są dostępne dla `mysql`. Więcej informacji na temat edycji zapytań znajdziesz w podpunkcie 1.5.2.1, zatytułowanym „Użycie oferowanych przez `mysql` możliwości edycji wiersza danych wejściowych”.

F.4.1. Opcje standardowe obsługiwane przez `mysql`

<code>--bind-address</code>	<code>--help</code>	<code>--silent</code>
<code>--character-sets-dir</code>	<code>--host</code>	<code>--socket</code>
<code>--compress</code>	<code>--password</code>	<code>--user</code>
<code>--debug</code>	<code>--pipe</code>	<code>--verbose</code>
<code>--debug-check</code>	<code>--port</code>	<code>--version</code>
<code>--debug-info</code>	<code>--protocol</code>	
<code>--default-character-set</code>	<code>--shared-memory-base-name</code>	

Ogólny opis powyższych opcji znajdziesz w punkcie F.2.1, zatytułowanym „Opcje standardowe programu MySQL”. Program `mysql` obsługuje także standardowe opcje SSL wymienione w podpunkcie F.2.1.1, zatytułowanym „Opcje standardowe SSL”.

W celu osiągnięcia większego efektu opcje `--silent` i `--verbose` mogą być podane kilkukrotnie.

Opcja `-I` jest synonimem opcji `--help`.

F.4.2. Opcje charakterystyczne dla mysql

■ --auto-rehash (*boolean*)

Podczas uruchamiania programu `mysql` może on utworzyć wartość hash na podstawie nazw bazy danych, tabel i kolumn i tym samym przygotować strukturę danych pozwalającą na szybkie uzupełnianie nazw. Wprowadzasz początkową część nazwy, następnie naciskasz klawisz `Tab`, a `mysql` uzupełnia nazwę. Jeśli nazwa nie jest jednoznaczna, kolejne naciśnięcie klawisza `Tab` wyświetla dostępne uzupełnienia.

Tworzenie wartości hash na podstawie nazw jest włączone domyślnie, ale nie będzie stosowane aż do chwili wskazania domyślnej bazy danych.

Opcja `--skip-auto-rehash` powoduje wyłączenie tworzenia wspomnianych wartości hash. Dzięki temu program `mysql` będzie uruchomiony znacznie szybciej, zwłaszcza w przypadku posiadania wielu tabel.

Jeżeli tworzenie wartości hash jest wyłączone, rozpoczęcie uzupełniania nazw po uruchomieniu `mysql` jest możliwe, jeśli wydasz polecenie `rehash` po znaku zachęty `mysql>`.

■ --auto-vertical-output (*boolean*)

Opcja powoduje automatyczne wyświetlanie danych wyjściowych w układzie pionowym dla zapytań, których wyniki będą dłuższe niż szerokość okna terminala. Patrz także opis opcji `--vertical`. Ta opcja została wprowadzona w wersji MySQL 5.5.3.

■ --batch, -B

Uruchomienie programu w trybie wsadowym. Program `mysql` wyświetla wyniki zapytań w formie wartości rozdzielonych tabulatorami (każdy rekord w oddzielnym wierszu, wartości kolumn rozdzielone tabulatorami).

Takie rozwiązanie jest szczególnie wygodne podczas generowania danych wyjściowych przeznaczonych do zaimportowania przez inny program, na przykład arkusz kalkulacyjny. Wyniki zapytań domyślnie zawierają także rekord początkowy wraz z nagłówkami kolumn. Jeśli nie chcesz wspomnianego rekordu z nagłówkami kolumn, użyj opcji `--skip-column-names`.

■ --binary-mode (*boolean*)

Wyłączenie konwersji `\r\n` na `\n` i interpretacji `\0` jako znacznika końca zapytania.

Omawiana opcja jest przeznaczona do stosowania w sytuacjach, w których program `mysql` jest używany do przetwarzania danych wyjściowych narzędzia `mysqlbinlog`, mogących zawierać dowolne wartości danych, na przykład BLOB.

Kiedy program `mysql` jest używany w trybie nieinteraktywnym, ta opcja powoduje również wyłączenie poleceń `charset` i `delimiter`. Opcja została wprowadzona w MySQL 5.6.3.

- `--column-names` (*boolean*)
Wyświetlenie nazw kolumn jako nagłówków kolumn w wynikach zapytania. Wyłączenie nazw kolumn umożliwia opcja `--skip-column-names`. Ten sam efekt można uzyskać po dwukrotnym podaniu opcji `--silent`.
- `--column-type-info` (*boolean*)
Opcja powoduje umieszczenie metadanych zbioru wynikowego w danych wyjściowych zapytania.
- `--comments, -c` (*boolean*)
W przypadku zapytań zawierających komentarze ta opcja powoduje dołączenie komentarzy do zapytania wysłanego do serwera. Domyślnie komentarze są usuwane (to odpowiada użyciu opcji `--skip-comments`).
- `--database=db_name, -D db_name`
Opcja powoduje użycie wymienionej bazy danych jako domyślnej.
- `--delimiter=str`
Opcja pozwala na ustawienie ogranicznika zapytania. Domyślnym ogranicznikiem zapytania jest średnik.
- `--execute=stmt, -e stmt`
Opcja powoduje wykonanie zapytania i zakończenie działania programu. Zapytanie należy ująć w znaki cytowania, aby powłoka (interpreter wiersza poleceń) nie potraktowała go jako wiele argumentów polecenia. W celu podania wielu zapytań należy je w wartości `stmt` rozdzielić średnikami.
- `--force, -f` (*boolean*)
Standardowo, gdy program `mysql` odczytuje zapytania z pliku, kończy działanie, jeśli wystąpi błąd. Użycie tej opcji powoduje, że `mysql` kontynuuje przetwarzanie zapytań niezależnie od ewentualnych błędów.
- `--html, -H` (*boolean*)
Opcja powoduje wygenerowanie danych wyjściowych w postaci kodu HTML.
- `--i-am-a-dummy` (*boolean*)
Ta opcja jest synonimem opcji `--safe-updates`.
- `--ignore-spaces, -i`
Opcja powoduje, że serwer ignoruje spacje między nazwami funkcji wbudowanych i nawiasem otwierającym `(`, który rozpoczyna listę argumentów. Normalnie, po nazwie funkcji natychmiast powinien znajdować się wspomniany nawias otwierający, bez żadnej spacji. Omawiana opcja powoduje również, że nazwy funkcji są traktowane jako słowa zarezerwowane.
- `--init-command=stmt`
Opcja pozwala na wskazanie zapytanie wykonywanego po nawiązaniu połączenia z serwerem MySQL lub po ponownym nawiązaniu połączenia, jeśli włączona jest opcja automatycznego ponownego nawiązywania połączeń. Informacje dotyczące ciągu tekstowego zapytania znajdziesz w opisie opcji `--execute`.

- `--line-numbers` (*boolean*)

Opcja powoduje wyświetlanie numerów wierszy w komunikatach błędów. To jest zachowanie standardowe. Jeśli chcesz wyłączyć wyświetlanie wspomnianych numerów wierszy, użyj opcji `--skip-line-numbers`.

- `--local-infile` (*boolean*)

Opcja włącza lub wyłącza `LOAD DATA LOCAL`. Możliwość użycia `LOCAL` może być dostępna, ale standardowo wyłączona. Jeżeli wykonanie zapytania `LOAD DATA LOCAL` skutkuje wygenerowaniem błędu, spróbuj ponownie je wykonać po uruchomieniu programu `mysql` wraz z opcją `--local-infile`. Omawiana opcja może być również użyta do wyłączenia `LOCAL` (na przykład za pomocą `--disable-local-infile`), jeśli włączono tę możliwość.

Opcja nie ma żadnego efektu, jeśli serwer został skonfigurowany w sposób uniemożliwiający użycie `LOCAL`.

- `--named-commands, -G` (*boolean*)

Opcja powoduje włączenie długich form wewnętrznych poleceń `mysql` na początku dowolnego wiersza danych wejściowych. Jeżeli wspomniana możliwość została wyłączona za pomocą `--skip-named-commands`, wtedy długie nazwy poleceń można stosować jedynie po podstawowym znaku zachęty, a nie po drugorzędnych. Oznacza to, że są niedostępne w drugim i kolejnych wierszach zapytań obejmujących wiele wierszy.

- `--no-auto-rehash, -A`

Ta opcja działa tak samo jak `--skip-auto-rehash`. Patrz także opis opcji `--auto-rehash`.

- `--no-beep, -b` (*boolean*)

Opcja powoduje wyłączenie generowania dźwięku po wystąpieniu błędu.

- `--one-database, -o`

Ta opcja jest używana podczas uaktualniania baz danych na podstawie zawartości pliku binarnego dziennika zdarzeń. Nakazuje programowi `mysql` uaktualnienie jedynie domyślnej bazy danych (czyli wymienionej w wierszu poleceń) i zignorowanie uaktualnień pozostałych baz danych. Jeżeli w wierszu poleceń nie podano bazy danych, nie zostaną przeprowadzone żadne uaktualnienia.

- `--pager[=program]`

Nazwa programu stronicującego używanego do wyświetlania ogromnych wyników zapytań w postaci jednej strony w danej chwili. Przykłady programów stronicujących to `/bin/more` i `/bin/less`. Jeżeli *program* nie zostanie podany, serwer będzie używał programu zdefiniowanego w zmiennej środowiskowej `PAGER`. Stronicowanie danych wyjściowych jest niedostępne w trybie wsadowym i nie działa w systemie Windows. Aby wyłączyć stronicowanie, należy użyć opcji `--skip-pager`.

■ `--prompt=str`

Opcja pozwala na zmianę podstawowego znaku zachęty z postaci `mysql>` na ciąg tekstowy zdefiniowany przez `str`. Wspomniany ciąg tekstowy może zawierać sekwencje specjalne wymienione w punkcie F.4.5, zatytułowanym „Sekwencje definiujące znak zachęty mysql”.

■ `--quick, -q`

Normalnie program `mysql` pobiera z serwera cały zbiór wynikowy zapytania, a dopiero później go wyświetla. Ta opcja powoduje wyświetlanie rekordów tuż po ich pobraniu, co powoduje znaczne zmniejszenie ilości wymaganej pamięci i pozwala na wykonanie z powodzeniem ogromnych zapytań, które w przeciwnym razie kończą się niepowodzeniem. Omawiana opcja nie powinna być stosowana w trybie interaktywnym, ponieważ jeśli użytkownik wstrzyma wyświetlanie danych wyjściowych lub działanie `mysql`, wtedy serwer również wstrzyma się z przekazywaniem danych wyjściowych i tym samym zablokuje inne klienty.

■ `--raw, -r (boolean)`

Opcja powoduje zapis wartości kolumn bez stosowania znaków sterujących dla jakichkolwiek znaków specjalnych. Ta opcja jest stosowana w połączeniu z `--batch`.

■ `--reconnect (boolean)`

- Opcja powoduje automatyczne ponowne nawiązanie połączenia, jeśli zostało ono przerwane. Ta opcja jest włączona domyślnie; jeśli chcesz ją wyłączyć, użyj `--skip-reconnect`.

W pewnych sytuacjach automatyczne ponowne nawiązywanie połączenia może powodować problemy. Na przykład, następuje wycofanie wszystkich aktualnie przeprowadzanych transakcji, a zmienne ustawione w trakcie sesji zostają wyzerowane bez żadnego powiadomienia o tym fakcie.

■ `--safe-updates, -U (boolean)`

Ta opcja nakłada ograniczenia na pewne operacje i może okazać się użyteczna dla nowych użytkowników MySQL:

- ◆ Uaktualnienia (zapytania modyfikujące dane) są dozwolone tylko wtedy, gdy rekordy przeznaczone do modyfikacji są wskazane przez wartości kluczy lub jeśli używana jest klauzula `LIMIT`. To chroni przed wykonaniem pomyłkowo wydanych zapytań zmieniających lub usuwających całą tabelę bądź jej ogromną część.
- ◆ Zbiory wynikowe wygenerowane przez operacje nieużywające złączeń są ograniczone do tysiąca rekordów, o ile nie zostanie użyta klauzula `LIMIT`. Z kolei zapytania wykorzystujące złączenia są odrzucane, jeśli optymalizator oszacuje, że konieczne będzie przeanalizowanie ponad miliona rekordów. To chroni przed niezamierzonym spadkiem wydajności podczas generowania naprawdę ogromnych zbiorów wynikowych.

Wymienione ograniczenia mogą być zmienione za pomocą zmiennych `select_limit` i `max_join_size`.

■ **--secure-auth** (*boolean*)

Opcja uniemożliwia nawiązanie połączenia z serwerem, jeśli nie obsługuje on algorytmu szyfrowania haseł wprowadzonego w wersji MySQL 4.1. Ta opcja jest domyślnie włączona, począwszy od MySQL 5.6.7, natomiast we wcześniejszych wersjach jest wyłączona.

■ **--show-warnings** (*boolean*)

Opcja powoduje automatyczne wyświetlanie wszelkich komunikatów ostrzeżeń tak, jakby po wykonaniu każdego zapytania następowало wykonanie `SHOW WARNINGS`.

■ **--sigint-ignore** (*boolean*)

Opcja powoduje zignorowanie sygnałów SIGINT, zwykle wysyłanych po naciśnięciu klawiszy `Ctrl+C`. Wymieniona kombinacja klawiszy nakazuje programowi `mysql` zakończenie bieżącego zapytania. Program zakończy działanie, jeżeli zapytanie nie może być zakończone lub jeśli nastąpi ponowne naciśnięcie klawiszy `Ctrl+C` przed zakończeniem zapytania. Użycie `--sigint-ignore` uniemożliwia programowi `mysql` interpretację naciśnięcia `Ctrl+C` w opisany powyżej sposób.

■ **--skip-column-names**, `-N`

Patrz opis opcji `--column-names`.

■ **--skip-line-numbers**, `-L`

Patrz opis opcji `--line-numbers`.

■ **--table**, `-t` (*boolean*)

Opcja powoduje wygenerowanie danych wyjściowych w formacie tabeli, w której poszczególne wartości są ograniczone pionowymi kreskami i wyrównane pionowo. To jest domyślny format wyświetlania danych wyjściowych, gdy program `mysql` nie został uruchomiony w trybie wsadowym.

■ **--tee=***file_name*

Ta opcja powoduje dołączenie kopii wszystkich danych wyjściowych do wskazanego pliku. W celu wyłączenia kopiowania danych wyjściowych do pliku należy użyć `--skip-tee`. Omawiana opcja nie działa w trybie wsadowym.

■ **--unbuffered**, `-n` (*boolean*)

Po każdym zapytaniu nastąpi opróżnienie bufora używanego podczas komunikacji z serwerem.

■ **--vertical**, `-E`

Opcja powoduje pionowe wyświetlanie danych wyjściowych zapytań, to znaczy każdy rekord będzie sformatowany jako zestaw wierszy danych wyjściowych, po jednej kolumnie w każdym wierszu. (Każdy wiersz składa się z nazwy kolumny oraz jej wartości). Wyświetlenie każdego rekordu będzie poprzedzone wierszem

wskazującym numer rekordu w zbiorze wynikowym. Pionowy format wyświetlania danych wyjściowych może być użyteczny, gdy zapytanie generuje bardzo długie wiersze.

Jeżeli omawiana opcja nie została podana, zastosowanie pionowego formatu danych wyjściowych można włączyć dla poszczególnych zapytań, kończąc je znakami \G zamiast średnikiem. Patrz także opis opcji `--auto-vertical-output`.

- `--wait, -w`

Opcja powoduje ponowienie próby nawiązania połączenia z serwerem po krótkiej chwili.

- `--xml, -X (boolean)`

Opcja powoduje wygenerowanie danych wyjściowych w formacie XML.

F.4.3. Zmienne programu mysql

Wymienione tutaj zmienne programu mysql mogą być ustawione za pomocą poleceń podanych w podpunkcie F.2.1.2, zatytułowanym „Ustawianie zmiennych programu”.

- `connect_timeout`

Liczba sekund oczekiwania, zanim nastąpi przekroczenie czasu oczekiwania podczas próby nawiązania połączenia z serwerem. Wartość domyślna wynosi 0.

- `max_allowed_packet`

Maksymalna wielkość bufora używanego podczas komunikacji między serwerem i klientem. Wartość domyślna wynosi 16 MB, natomiast wartość maksymalna to 1 GB.

- `max_join_size`

Ograniczenie liczby rekordów podczas przeprowadzania złączeń, gdy użyta została opcja `--safe-updates`. Serwer odrzuca złączenia, jeśli oszacuje, że będzie musiał przeanalizować większą liczbę rekordów niż określona w zmiennej `max_join_size`. Wartość domyślna wynosi 1000000.

- `net_buffer_length`

Początkowa wielkość bufora używanego podczas komunikacji między serwerem i klientem. Wielkość tego bufora może wzrosnąć do określonej przez zmienną `max_allowed_packet`. Wartość domyślna wynosi 16 KB.

- `select_limit`

Ograniczenie liczby rekordów zwracanych przez zapytania SELECT, gdy użyta została opcja `--safe-updates`. Wartość domyślna wynosi 1000.

F.4.4. Polecenia programu mysql

Poza możliwością wysyłania zapytań SQL do serwera, program mysql implementuje również wiele własnych poleceń. Każde z nich musi zostać podane w pojedynczym wierszu. Większość poleceń ma długą formę, składającą się ze słowa, oraz krótką,

składającą się z ukośnika \ i pojedynczej litery. Wielkość liter w długich nazwach poleceń nie ma znaczenia. Z kolei w krótkiej formie poleceń trzeba używać wielkości liter podanej na poniższej liście. Średnik na końcu wiersza nie jest konieczny, ale dozwolony dla poleceń w długiej formie. Nie należy go stosować w krótkiej formie poleceń.

Jeżeli wyłączyłeś długie nazwy poleceń (na przykład za pomocą opcji `--skip-named-commands`), długie nazwy poleceń będą rozpoznawane jedynie po podstawowym znaku zachęty. Jeżeli program jest używany w trybie nieinteraktywnym i wywołany z opcją `--binary-mode`, wyłączone są wszystkie polecenia poza `charset` i `delimiter`.

- `clear, \c`

Polecenie powoduje przerwanie bieżącego zapytania, to znaczy aktualnie wprowadzanego. To polecenie nie spowoduje przerwania wykonywania zapytania wysłanego do serwera i tego, dla którego program `mysql` wyświetla dane wyjściowe.

- `charset [charset], \C [charset]`

Polecenie zmienia domyślne kodowanie znaków i wykonuje zapytanie `SET NAMES`, aby serwer również używał wskazanego kodowania znaków.

- `connect [db_name [host_name]], \r [db_name [host_name]]`

Nawiązanie połączenia (lub ponowne nawiązanie połączenia) z podaną bazą danych we wskazanym komputerze. Jeśli pominiesz nazwę bazy danych lub komputera, zostaną zastosowane ostatnio użyte wartości w bieżącej sesji `mysql`.

- `delimiter str, \d str`

Polecenie definiuje ogranicznik zapytania. Ogranicznikiem domyślnym jest średnik. Analizator programu składowanego rozpoznaje jedynie średnik jako ogranicznik zapytania, a więc omawiane polecenie może być używane do zmiany ogranicznika zapytania dla programu `mysql` podczas definiowania programu składowanego. Przykład użycia polecenia znajdziesz w punkcie 4.2.1, zatytułowanym „Zapytania złożone i ograniczniki zapytań”.

Najlepiej unikać ukośników w ogranicznikach, ponieważ MySQL traktuje ukośniki jako znaki sterujące.

- `edit, \e`

Polecenie pozwala na edycję bieżącego zapytania. Program `mysql` próbuje określić używany edytor przez analizę zmiennych środowiskowych `EDITOR` i `VISUAL`. Jeżeli żadna z wymienionych zmiennych nie została ustawiona, wtedy `mysql` używa edytora `vi`. Omawiane polecenie jest niedostępne w systemie Windows.

- `ego, \G`

Polecenie wysyła bieżące zapytanie do serwera, a następnie wyświetla jego wyniki w formacie pionowym.

- `exit`

Polecenie działa tak samo jak `quit`.

- `go, \g, ;`
Polecenie wysyła bieżące zapytanie do serwera, a następnie wyświetla jego wyniki.
- `help [topic], \h [topic], ? [topic], \? [topic]`
Polecenie wyświetla komunikat pomocy dostępny dla poleceń mysql.
Jeżeli wczytane zostały tabele pomocy w bazie danych mysql, wtedy polecenie `help` można wykorzystać do otrzymania pomocy ze strony serwera. Wydanie polecenia `help contents` pobiera listę kategorii pomocy, `help category` wyświetla pomoc dotyczącą wskazanej kategorii, natomiast `help keyword` wyświetla pomoc dotyczącą wskazanego słowa kluczowego (na przykład `SELECT` lub `UPDATE`). Informacje dotyczące wczytywania tabel pomocy znajdziesz w punkcie A.3.6, zatytułowanym „Inicjalizacja innych tabel systemowych”.
- `nopager, \n`
Polecenie powoduje wyłączenie użycia programu stronicującego i oznacza wysłanie danych do standardowego wyjścia. Omawiane polecenie jest niedostępne w systemie Windows.
- `notee, \t`
Polecenie zatrzymuje zapis danych wyjściowych do pliku.
- `nowarning, \w`
Polecenie zatrzymuje automatyczne wyświetlanie wszelkich ostrzeżeń generowanych przez zapytania.
- `pager [program], \P [program]`
Polecenie powoduje wyświetlenie danych wyjściowych za pomocą wskazanego programu stronicującego. Jeżeli *program* nie zostanie podany, serwer będzie używał programu zdefiniowanego w zmiennej środowiskowej `PAGER`. Omawiane polecenie jest niedostępne w systemie Windows.
- `print, \p`
Polecenie wyświetla bieżące zapytanie (sam tekst zapytania, a nie wyniki otrzymane po jego wykonaniu).
- `prompt [arguments], \R [arguments]`
Polecenie pozwala na zmianę podstawowego znaku zachęty `mysql>`. To wszystko, co zostanie umieszczone po pierwszej spacji znajdującej się za poleceniem `prompt`, staje się częścią ciągu tekstowego znaku zapytania (spacje również są uwzględniane). Ciąg tekstowy może zawierać sekwencje specjalne wymienione w punkcie F.4.5, zatytułowanym „Sekwencje definiujące znak zachęty mysql”. Aby przywrócić domyślną postać znaku zachęty, należy wydać polecenie `prompt` lub podać `\R` bez argumentów.

- `quit, \q`
Zakończenie pracy klienta `mysql`.
- `rehash, \#`
Ponowne wygenerowanie informacji potrzebnych do uzupełniania nazw baz danych, tabel i kolumn. Patrz także opis opcji `--auto-rehash`.
- `source file_name, \. file_name`
Polecenie odczytuje i wykonuje zapytania znajdujące się we wskazanym pliku. W przypadku ścieżek dostępu systemu Windows zawierających separatory w postaci ukośników `\` zamiast `/`, wspomniane ukośniki należy podać podwójnie (`\\`) lub użyć ukośnika `/`.
- `status, \s`
Polecenie pobiera z serwera i wyświetla informacje o stanie. Omawiane polecenie będzie użyteczne podczas sprawdzania wersji serwera, domyślnej bazy danych, szyfrowania SSL itd.
- `system command, \! command`
Wykonanie wskazanego polecenia (`command`) za pomocą domyślnego interpretera poleceń. Polecenie `system` jest niedostępne w systemie Windows.
- `tee [file_name], \T [file_name]`
Polecenie powoduje skopiowanie danych wyjściowych na koniec wskazanego pliku. W przypadku pominięcia nazwy pliku dane wyjściowe zostaną umieszczone w ostatnio użytym pliku.
- `use db_name, \u db_name`
Polecenie wybiera wskazaną bazę danych, która staje się domyślną.
- `warnings, \W`
Polecenie automatycznie wyświetla wszelkie ostrzeżenia wygenerowane przez zapytania.

F.4.5. Sekwencje definiujące znak zachęty `mysql`

Zmienna środowiskowa `MYSQL_PS1`, opcja `--prompt` i polecenie `prompt` mogą zostać użyte do zmiany podstawowego znaku zachęty `mysql>` wyświetlanego przez klienta `mysql`.

Na przykład, aby znak zachęty zawierał nazwę domyślnej bazy danych, należy użyć polecenia `prompt` w przedstawiony poniżej sposób. Następnie możesz wybierać różne bazy danych i zobaczyć, jak zmienia się znak zachęty:

```
% mysql
mysql> prompt \d>\_
PROMPT set to '\d>\_'
(none)> USE sampdb;
Database changed
sampdb> USE test;
Database changed
test>
```

Po słowie kluczowym prompt należy podać ciąg tekstowy definiujący znak zachęty. W definicji sekwencje sterujące rozpoczynające się od ukośnika wskazują opcje specjalne znaku zachęty. Sekwencje \d i _ wskazują odpowiednio nazwę domyślnej bazy danych i spację. (Jeżeli znak zachęty definiujesz za pomocą zmiennej środowiskowej lub opcji --prompt, wtedy konieczne może okazać się użycie podwójnych ukośników w ciągu tekstowym znaku zachęty). Dostępne opcje podczas definiowania znaku zachęty wymieniono w tabeli F.6.

Tabela F.6. Opcje dostępne podczas definiowania znaku zachęty

Sekwencja	Opis
\c	Numer bieżącego wiersza danych wejściowych.
\d	Nazwa domyślnej bazy danych lub "(none)", jeśli nie została wybrana.
\D	Pełna data i godzina.
\h	Bieżący komputer.
\l	Bieżący ogranicznik.
\m	Minuta.
\o	Numer miesiąca.
\O	Nazwa miesiąca podana za pomocą trzech liter.
\p	Bieżący numer portu, gniazda, nazwanego potoku lub pamięci współdzielonej.
\P	Użycie am lub pm podczas wyświetlania godziny.
\r	Godzina (format 12-godzinny).
\R	Godzina (format 24-godzinny).
\s	Sekunda.
\S	Średnik.
\t	Tabulator.
\u	Bieżąca nazwa użytkownika, bez nazwy komputera.
\U	Bieżąca nazwa użytkownika wraz z nazwą komputera.
\v	Wersja serwera.
\w	Dzień tygodnia podany za pomocą trzech liter.
\y	Rok (dwie cyfry).
\Y	Rok (cztery cyfry).
\'	Apostrof.
\"	Cudzysłów.
_	Spacja.

Tabela F.6. Opcje dostępne podczas definiowania znaku zachęty (ciąg dalszy)

Sekwencja	Opis
\	Spacja (sekwencja to ukośnik i spacja).
\\	Dosłowny ukośnik \.
\n	Znak nowego wiersza (wysuw wiersza).
\x	Dosłowny znak <i>x</i> dla dowolnego znaku <i>x</i> , który nie został wyżej wymieniony.

F.5. mysql.server

`mysql.server` to dostępny w systemach UNIX skrypt powłoki odpowiedzialny za uruchamianie i zatrzymywanie serwera za pomocą `mysqld_safe`.

Skrypt `mysql.server` akceptuje argument wiersza powłoki `start` lub `stop`:

```
mysql.server start
mysql.server stop
```

Normalnie skrypt `mysql.server` jest instalowany w katalogu poziomu działania w systemach UNIX posiadających tego rodzaju podkatalogi w */etc*. (Zainstalowana wersja najczęściej ma nazwę `mysql` zamiast `mysql.server`). System uruchamia serwer przez wywołanie skryptu `mysql.server` wraz z argumentem `start` w trakcie procesu uruchamiania systemu. Natomiast zamknięcie serwera przez system następuje przez wywołanie skryptu `mysql.server` wraz z argumentem `stop` w trakcie procesu zamykania systemu. Skrypt `mysql.server` można wywołać także ręcznie z odpowiednim argumentem i tym samym uruchomić lub zamknąć serwer.

F.5.1. Opcje obsługiwane przez mysql.server

Oferowana przez `mysql.server` obsługa standardowych opcji MySQL jest dość ograniczona. Skrypt nie odczytuje żadnej standardowej opcji z wiersza poleceń. W grupach opcji `[mysql_server]` i `[mysql.server]` znajdujących się w plikach opcji skrypt odczytuje opcje `basedir`, `datadir` oraz `pid-file` i przekazuje je `mysqld_safe`. Odczytuje także opcję `service-startup-timeout`, pobierającą wartość liczbową wskazującą liczbę sekund, przez które skrypt powinien poczekać na uruchomienie serwera. Wartość domyślna wynosi 900. Wartość 0 oznacza brak oczekiwania, natomiast wartość ujemna oznacza oczekiwanie w nieskończoność.

F.6. mysql_config

Narzędzie `mysql_config` pomaga w opracowywaniu programów MySQL w języku C. Może być wywołane w celu pobrania prawidłowych flag niezbędnych do kompilacji plików źródłowych C lub dołączania w bibliotekach MySQL:

```
mysql_config [options]
```

Przykład użycia wymienionego narzędzia znajdziesz w podrozdziale 7.1, zatytułowanym „Kompilacja i linkowanie programów klienckich”.

F.6.1. Opcje charakterystyczne dla mysql_config

- **--cflags**
Opcja wyświetla flagi katalogu plików nagłówkowych, niezbędna w celu uzyskania dostępu do plików nagłówkowych MySQL, oraz inne flagi kompilatora C, które mogą okazać się konieczne.
- **--cxxflags**
Opcja działa podobnie do **--cflags**, ale jest przeznaczona dla kompilatora C++. Omawiana opcja została wprowadzona w MySQL 5.6.4.
- **--embedded, --embedded-libs**
Te opcje są synonimami dla **--libmysqld-libs**.
- **--include**
Opcja wyświetla flagi katalogu plików nagłówkowych niezbędne w celu uzyskania dostępu do plików nagłówkowych MySQL.
- **--libmysqld-libs**
Opcja wyświetla flagi biblioteki niezbędne do dołączenia w **libmysqld**, osadzonej bibliotece serwera.
- **--libs**
Opcja wyświetla flagi niezbędne do dołączenia w bibliotece klienta.
- **--libs_r**
Opcja wyświetla flagi niezbędne do dołączenia w bibliotece klienta zapewniającej bezpieczeństwo wątków.
- **--plugindir**
Opcja wyświetla domyślny katalog wtyczek.
- **--port**
Opcja wyświetla domyślny numer portu TCP/IP.
- **--socket**
Opcja wyświetla domyślną ścieżkę dostępu do pliku gniazda systemu UNIX.
- **--variable=var_name**
Opcja wyświetla wartość wskazanej zmiennej, którą może być **pkgincludedir**, **pkglibdir** lub **plugindir**.
- **--version-**
Opcja wyświetla ciąg tekstowy przedstawiający wersję MySQL.

F.7. Skrypt `mysql_install_db`

Skrypt `mysql_install_db` powoduje utworzenie katalogu danych serwera, inicjalizację bazy danych `mysql` zawierającej tabelę uprawnień oraz utworzenie pustej bazy danych o nazwie `test`:

```
mysql_install_db [options]
```

Skrypt `mysql_install_db` wypełnia tabelę uprawnień początkowymi kontami dla użytkowników `root` i anonimowego. W rozdziale 12., zatytułowanym „Ogólna administracja bazą danych MySQL”, znajdziesz informacje szczegółowe na temat wymienionych kont użytkowników oraz dowiesz się, jak zabezpieczyć instalację MySQL przez zdefiniowanie haseł.

Skrypt `mysql_install_db` jest niedostępny w systemie Windows, ponieważ dystrybucje MySQL przeznaczone dla Windows mają preinstalowany katalog danych.

F.7.1. Opcje standardowe obsługiwane przez `mysql_install_db`

```
--help            --user            --verbose
```

Ogólny opis powyższych opcji znajdziesz w punkcie F.2.1, zatytułowanym „Opcje standardowe programu MySQL”.

W systemach UNIX opcja `--user` powoduje uruchomienie serwera w ramach konta logowania wskazanego użytkownika. Użyj tej opcji, jeśli chcesz uruchomić `mysql_install_db` jako użytkownik UNIX `root`, co zagwarantuje, że wszystkie katalogi i pliki utworzone przez serwer będą własnością wymienionego użytkownika.

F.7.2. Opcje charakterystyczne dla `mysql_install_db`

Opcji wymienionych w tym punkcie można używać w wierszu poleceń; wartości dla wielu tych opcji mogą być definiowane przez umieszczenie odpowiednich wpisów w grupie `[mysqld]` pliku opcji. Skrypt odczytuje także grupę opcji `[mysql_install_db]`, która będzie użyteczna dla opcji takich jak `--ldata` i `--force`, obsługiwanych jedynie przez `mysql_install_db`, a nie przez `mysqld`.

Wszystkie nierozpoznane opcje `mysql_install_db` przekazuje do `mysqld`.

- `--basedir=dir_name`

Ścieżka dostępu do katalogu bazowego serwera MySQL.

- `--datadir=dir_name, --ldata=dir_name`

Ścieżka dostępu do katalogu danych serwera MySQL.

- `--force`

Opcja powoduje uruchomienie skryptu, nawet jeśli nie można określić nazwy komputera. Adres IP komputera jest używany do utworzenia wpisów w tabelach

uprawnień. Dlatego też w celu użycia programów klienckich konieczne jest podanie adresu IP zamiast nazwy komputera, za wyjątkiem połączeń z localhost.

■ **--skip-name-resolve**

Opcja powoduje użycie jedynie adresów IP z tabel uprawnień zamiast nazw komputerów. Użycie omawianej opcji może okazać się konieczne, jeśli serwer DNS nie działa prawidłowo lub w ogóle.

F.8. mysql_upgrade

Ten program powinien być używany po przeprowadzeniu uaktualnienia serwera MySQL do nowszej wersji. Domyślnie nawiązuje on połączenie z serwerem lokalnym jako użytkownik MySQL root, więc należy go uruchomić wraz z hasłem użytkownika root:

```
% mysql_upgrade --password=rootpass
```

Program mysql_upgrade sprawdza bazy danych instalacji MySQL, a następnie próbuje znaleźć wszelkie niezgodności z nową wersją serwera i wyeliminować je, o ile możliwe. Przeprowadzane operacje obejmują uaktualnienie tabel w bazie danych mysql (na przykład w celu zapewnienia obsługi nowych uprawnień) i naprawę tabel zawierających dane w formatach, które mogą powodować problemy w nowym serwerze. Po zakończeniu działania przez mysql_upgrade należy ponownie uruchomić serwer, aby zmiany wprowadzone przez wymieniony program zostały uwzględnione.

Program mysql_upgrade wywołuje mysqlcheck w celu przeprowadzenia operacji sprawdzenia i naprawy tabel. Więcej informacji dotyczących możliwości mysqlcheck znajdziesz w poświęconym mu podpunkcie.

F.8.1. Opcje standardowe obsługiwane przez mysql_upgrade

--character-sets-dir	--default-character-set	--port
--compress	--help	--protocol
--debug	--host	--shared-memory-base-name
--debug-check	--password	--socket
--debug-info	--pipe	--user
--default-auth	--plugin-dir	--verbose

Ogólny opis powyższych opcji znajdziesz w punkcie F.2.1, zatytułowanym „Opcje standardowe programu MySQL”. Program mysql_upgrade obsługuje także standardowe opcje SSL wymienione w podpunkcie F.2.1.1, zatytułowanym „Opcje standardowe SSL”.

W przeciwieństwie do większości programów MySQL, wartością domyślną --user jest root. Opcja --verbose jest włączona domyślnie; aby ją wyłączyć, należy użyć --skip-verbose.

F.8.2. Opcje charakterystyczne dla `mysql_upgrade`

- `--basedir=dir_name, -b dir_name`
Ta opcja jest nieużywana.
- `--datadir=dir_name, -d dir_name`
Ta opcja jest nieużywana.
- `--force, -f (boolean)`
Opcja powoduje wymuszenie uaktualnienia, nawet jeśli program `mysql_upgrade` został uruchomiony dla bieżącej wersji MySQL.
- `--tmpdir=dir_name, -t dir_name`
Opcja określa ścieżkę dostępu do katalogu, który powinien być używany dla plików tymczasowych.
- `--upgrade-system-tables, -s (boolean)`
Ta opcja powoduje uaktualnienie jedynie tabel w bazie danych `mysql`.
- `--write-binlog (boolean)`
Ta opcja powoduje zapis w binarnym dzienniku zdarzeń zapytań SQL wykonanych w trakcie działania programu `mysql_upgrade`. Użycie omawianej opcji powoduje, że wspomniane zapytania SQL będą replikowane.

F.9. `mysqladmin`

Klient `mysqladmin` prowadzi komunikację z serwerem MySQL w celu wykonywania zadań administracyjnych. Program `mysqladmin` można wykorzystać do pobrania z serwera informacji dotyczących jego działania, do zdefiniowania haseł, do utworzenia lub usunięcia baz danych:

```
mysqladmin [options] command ...
```

F.9.1. Opcje standardowe obsługiwane przez `mysqladmin`

<code>--bind-address</code>	<code>--default-character-set</code>	<code>--protocol</code>
<code>--character-sets-dir</code>	<code>--help</code>	<code>--shared-memory-base-name</code>
<code>--compress</code>	<code>--host</code>	<code>--silent</code>
<code>--debug</code>	<code>--password</code>	<code>--socket</code>
<code>--debug-check</code>	<code>--pipe</code>	<code>--user</code>
<code>--debug-info</code>	<code>--plugin-dir</code>	<code>--verbose</code>
<code>--default-auth</code>	<code>--port</code>	<code>--version</code>

Ogólny opis powyższych opcji znajdziesz w punkcie F.2.1, zatytułowanym „Opcje standardowe programu MySQL”. Program `mysql_upgrade` obsługuje także standardowe opcje SSL wymienione w podpunkcie F.2.1.1, zatytułowanym „Opcje standardowe SSL”.

Opcja `--silent` powoduje, że `mysqladmin` zakończy działanie bez żadnego komunikatu, jeśli nie uda się nawiązać połączenia z serwerem. W przypadku kilku poleceń opcja `--verbose` powoduje, że `mysqladmin` będzie wyświetlać większą ilość informacji.

F.9.2. Opcje charakterystyczne dla `mysqladmin`

- `--count=n, -c n`

Ta opcja określa liczbę iteracji do wykonania, gdy użyta została opcja `--sleep`. W przypadku podania `--sleep`, ale bez `--count`, program `mysqladmin` będzie wykonywał iteracje w nieskończoność (lub do chwili zakończenia jego działania).

- `--force, -f (boolean)`

Ta opcja ma dwa efekty — program `mysqladmin` nie będzie żądał potwierdzenia opcji `drop nazwa_bazy_danych`, a poza tym w przypadku podania wielu poleceń w wierszu poleceń, `mysqladmin` spróbuje wykonać je wszystkie, nawet jeśli wystąpią błędy. Standardowo `mysqladmin` kończy działanie po wystąpieniu pierwszego błędu.

- `--no-beep, -b (boolean)`

Ta opcja powoduje wyłączenie generowania dźwięku po wystąpieniu błędu.

- `--relative, -r (boolean)`

Ta opcja pokazuje różnice między wartościami bieżącą i poprzednią, gdy program jest uruchomiony wraz z opcją `--sleep`. Omawiana opcja działa jedynie z poleceniem `extended-status`.

- `--sleep=n, -i n`

Ta opcja powoduje cykliczne wykonywanie poleceń podanych w wierszu poleceń z opóźnieniem n sekund między poszczególnymi wykonaniami.

- `--vertical, -E (boolean)`

Ta opcja działa podobnie jak `--relative`, ale pionowo formatuje dane wyjściowe.

- `--wait[=n], -w[n]`

Liczba sekund oczekiwania przed ponowieniem próby nawiązania połączenia, jeśli jeszcze nie udało się go nawiązać. Wartość domyślna n wynosi 1. W przypadku podania n po `-w` między nimi nie może być spacji, w przeciwnym razie wartość nie zostanie prawidłowo zinterpretowana.

F.9.3. Zmienne dla `mysqladmin`

Wymienione tutaj zmienne mogą być ustawione za pomocą poleceń podanych w podpunkcie F.2.1.2, zatytułowanym „Ustawianie zmiennych programu”.

- `connect_timeout`

Liczba sekund oczekiwania, zanim nastąpi przekroczenie czasu oczekiwania podczas próby nawiązania połączenia z serwerem. Wartość domyślna wynosi 43200.

- `shutdown_timeout`

W przypadku poleceń `shutdown` to liczba sekund oczekiwania na zakończone powodzeniem zamknięcie połączenia. Wartość domyślna wynosi 3600.

F.9.4. Polecenia `mysqladmin`

Po dowolnych opcjach w wierszu poleceń można podać jedno lub więcej wymienionych poniżej poleceń. Nazwa każdego polecenia może być skrócona do jednoznacznego skrótu. Na przykład, polecenie `processlist` można skrócić do `process` lub `proc`, ale już nie do `p`.

Kilka z wymienionych tutaj poleceń ma odpowiedniki w postaci zapytań SQL, o czym wspomniano w opisach. Więcej informacji na temat wspomnianych zapytań SQL znajdziesz w dodatku E, zatytułowanym „Przewodnik po składni SQL”.

- `create db_name`

Utworzenie nowej bazy danych o podanej nazwie. To polecenie działa jak zapytanie `CREATE DATABASE nazwa_bazy_danych`.

- `debug`

Polecenie nakazuje serwerowi zapis w pliku dziennika błędów informacji debugowania.

- `drop db_name`

Usunięcie bazy danych o wskazanej nazwie oraz wszystkich tabel, które mogą się w niej znajdować. Narzędzie `mysqladmin` prosi o potwierdzenie wykonania tego polecenia, o ile nie zostanie użyta opcja `--force`. Omawiane polecenie działa jak zapytanie `DROP DATABASE nazwa_bazy_danych`.

- `extended-status`

Polecenie powoduje wyświetlenie nazw i wartości zmiennych stanu serwera. To polecenie działa jak zapytanie `SHOW STATUS`.

- `flush-hosts`

Opróżnienie bufora hostów. To polecenie działa jak zapytanie `FLUSH HOSTS`.

- `flush-logs`

Opróżnienie (zamknięcie i ponowne otwarcie) plików dzienników zdarzeń. To polecenie działa jak zapytanie `FLUSH LOGS`.

- `flush-privileges`

Ponowne wczytanie tabel uprawnień. To polecenie działa jak zapytanie `FLUSH PRIVILEGES`.

- `flush-status`

Wyczyszczenie zmiennych stanu. (Polecenie powoduje wyzerowanie wielu liczników). To polecenie działa jak zapytanie `FLUSH STATUS`.

- `flush-tables`

Opróżnienie bufora tabeli. To polecenie działa jak zapytanie `FLUSH TABLES`.

- `flush-threads`

Opróżnienie bufora wątków.

- `kill id,id,...`

Zamknięcie w serwerze wątków o podanych identyfikatorach. Jeżeli chcesz podać więcej niż jeden identyfikator, wtedy lista nie może zawierać spacji, aby identyfikatory nie zostały pomyłone z innymi poleceniami po `kill`. Aby dowiedzieć się, jakie wątki aktualnie działają w serwerze, wydaj polecenie `mysqladmin processlist`. To polecenie jest jak wykonanie zapytania `KILL` wraz z każdym identyfikatorem wątku.

- `old-password new_password`

To polecenie działa podobnie jak `password`, za wyjątkiem faktu, że hasła są przechowywane w formacie stosowanym przez MySQL w wersji wcześniejszej niż 4.1.

- `password new_password`

Polecenie pozwala na zmianę hasła konta użytkownika wykorzystywanego do nawiązania połączenia z serwerem. (Możliwość nawiązania połączenia z serwerem za pomocą tego konta służy potwierdzeniu, że znasz bieżące hasło konta użytkownika). Argumentem polecenia jest definiowane hasło. Omawiane polecenie działa jak zapytanie `SET PASSWORD`.

W systemach UNIX można użyć apostrofów i cudzysłowu w poleceniu `mysqladmin` do cytowania hasła, jeśli zawiera ono znaki, które wiersz poleceń mógłby zinterpretować jako specjalne. Z kolei w Windows do cytowania hasła należy używać jedynie cudzysłowu. Wiersz poleceń w Windows nie rozpoznaje apostrofów jako znaków cytowania argumentu, więc jeśli ich użyjesz, apostrofy będą uznane za część hasła!

Począwszy od MySQL 5.5.3, jeśli `password` jest ostatnim poleceniem w wierszu poleceń, wtedy wartość `new_password` jest opcjonalna; w przypadku braku hasła `mysqladmin` poprosi o jego podanie.

- `ping`

Polecenie pozwala na sprawdzenie, czy serwer MySQL działa.

- `processlist`

Wyświetlenie listy operacji aktualnie wykonywanych przez serwer. To polecenie działa jak zapytanie `SHOW PROCESSLIST`, a po użyciu opcji `--verbose` działa jak `SHOW FULL PROCESSLIST`.

- `refresh`

Polecenie powoduje opróżnienie bufora tabeli i tabel uprawnień oraz zamknięcie i ponowne otwarcie plików dzienników zdarzeń. Jeżeli serwer jest serwerem głównym replikacji, wtedy omawiane polecenie powoduje usunięcie plików binarnego dziennika zdarzeń wymienionych w pliku indeksu oraz opróżnienie samego indeksu. Jeśli serwer jest serwerem podległym replikacji, omawiane polecenie powoduje, że serwer zapomina położenie w pliku binarnego dziennika zdarzeń serwera głównego.

- **reload**
Polecenie powoduje ponowne wczytanie tabel uprawnień. Omawiane polecenie działa jak zapytanie `FLUSH PRIVILEGES`.
- **shutdown**
Polecenie powoduje zamknięcie serwera.
- **start-slave**
Uruchomienie serwera podległego replikacji. To polecenie działa jak zapytanie `START SLAVE`.
- **status**
Polecenie wyświetla krótki komunikat stanu serwera.
- **stop-slave**
Zatrzymanie serwera podległego replikacji. To polecenie działa jak zapytanie `STOP SLAVE`.
- **variables**
Polecenie powoduje wyświetlenie nazw i wartości zmiennych serwera. To polecenie działa jak zapytanie `SHOW GLOBAL VARIABLES`.
- **version**
Polecenie powoduje pobranie i wyświetlenie ciągu tekstowego zawierającego informacje o wersji serwera. Te same informacje zwraca funkcja SQL o nazwie `VERSION()`.

F.10. mysqlbinlog

Program `mysqlbinlog` wyświetla w czytelnym formacie zawartość pliku binarnego dziennika zdarzeń:

```
mysqlbinlog [options] file_name ...
```

Domyślnie program `mysqlbinlog` odczytuje lokalne pliki dziennika zdarzeń bez nawiązywania połączenia z serwerem. Istnieje również możliwość nawiązania połączenia z serwerem i pobrania za jego pomocą wspomnianych plików dzienników zdarzeń. Patrz opis opcji `--read-from-remote-server`.

Format binarnego dziennika zdarzeń zmienia się od czasu do czasu. Aby uniknąć problemów związanych ze zgodnością, konieczne może okazać się użycie programu `mysqlbinlog` co najmniej w tej samej wersji, w której jest serwer.

Program `mysqlbinlog` odczytuje pliki dziennika przekazywania tworzonego przez serwer podległy replikacji, ponieważ dzienniki binarny i przekazywania mają ten sam format.

F.10.1. Opcje standardowe obsługiwane przez mysqlbinlog

--bind-address	--help	--shared-memory-base-name
--character-sets-dir	--host	--socket
--debug	--password	--user
--debug-check	--plugin-dir	--verbose
--debug-info	--port	--version
--default-auth	--protocol	

Ogólny opis powyższych opcji znajdziesz w punkcie F.2.1, zatytułowanym „Opcje standardowe programu MySQL”.

Opcja --verbose powoduje, że program mysqlbinlog wyświetla zdarzenia rekordów jako umieszczone w komentarzach zapytania SQL. Dwukrotne podanie wymienionej opcji generuje także komentarze opisujące metadane kolumn.

F.10.2. Opcje charakterystyczne dla mysqlbinlog

■ --base64-output=*value*

Opcja określa, że dane wyjściowe zapytań BINLOG mają być wyświetlane w postaci zakodowanej jako base64. Tego rodzaju zapytania są zwykle używane dla zdarzeń rekordów nieopisywanych w kategoriach SQL. Wartości dozwolone dla omawianej opcji wymieniono poniżej:

- ◆ AUTO lub UNSPEC. Automatyczne użycie zapytań BINLOG, gdy zajdzie potrzeba.
- ◆ NEVER. Zapytania BINLOG nie będą generowane. Jeżeli zdarzenie rekordu nie będzie mogło być wyświetlone w postaci innej niż BINLOG, nastąpi wygenerowanie błędu, a program zakończy działanie.
- ◆ DECODE_ROWS. Wartość używana, gdy podana jest także opcja --verbose. Omawiana wartość ma takie samo działanie jak NEVER, ale nie powoduje zakończenia działania programu z błędem.
- ◆ ALWAYS. Zapytania BINLOG będą zawsze wyświetlane, o ile istnieje możliwość. Począwszy od MySQL 5.6.1, omawiana wartość nie jest dłużej dostępna.

Wielkość liter w wartościach opcji --base64-output nie ma znaczenia. Jeżeli omawiana opcja nie zostanie podana, domyślnie używaną wartością jest AUTO. To także jedyna bezpieczna wartość, jeśli planujesz ponowne wykonanie danych wyjściowych za pomocą programu mysqlbinlog. Pozostałe wartości powinny być uznawane za przeznaczone jedynie do debugowania lub celów testowych.

■ --binlog-row-event-max-size=*n*

Maksymalna dozwolona wielkość dla zdarzeń opartych na rekordach. Program mysqlbinlog, gdy tylko to możliwe, próbuje grupować rekordy w zdarzenia nie większe niż wymieniona wartość. Podana tutaj wartość będzie skrócona do najbliższej niezerowej wielokrotności 256. Wartością domyślną jest 4 GB. Ta opcja została wprowadzona w MySQL 5.6.0.

■ `--database=db_name, -d db_name`

Wyodrębnienie z pliku dziennika zdarzeń tylko tych zapytań, które dotyczą wskazanej bazy danych. Ta opcja działa jedynie podczas odczytu lokalnych plików dzienników zdarzeń.

■ `--disable-log-bin, -D (boolean)`

Opcja powoduje umieszczenie w danych wyjściowych zapytań uniemożliwiających rejestrowanie w dzienniku zdarzeń zapytań uaktualniających dane. Dzięki temu wspomniane zapytania uaktualniające dane nie będą ponownie zarejestrowane podczas kolejnego wykonania zapytań dziennika.

■ `--force-if-open, -F (boolean)`

Opcja wymusza odczyt plików binarnego dziennika zdarzeń nawet wtedy, gdy nie zostały prawidłowo zamknięte lub są aktualnie w użyciu. Ta opcja jest włączona domyślnie. Aby ją wyłączyć, należy użyć `--skip-force-if-open`.

■ `--force-read, -f (boolean)`

Ta opcja określa zachowanie programu `mysqlbinlog` podczas odczytu z binarnego dziennika zdarzeń nierozpoznanego zdarzenia. Domyślnie następuje zatrzymanie działania programu. Po użyciu omawianej opcji program `mysqlbinlog` będzie kontynuował działanie i jedynie wygeneruje ostrzeżenie dotyczące nierozpoznanego zdarzenia.

■ `--hexdump, -H (boolean)`

Opcja powoduje umieszczenie w danych wyjściowych kodu zdarzenia w postaci szesnastkowej/ASCII.

■ `--local-load=dir_name, -l dir_name`

Opcja wskazuje katalog, w którym będą tworzone tymczasowe pliki danych podczas przetwarzania zapytań `LOAD DATA LOCAL`. Program `mysqlbinlog` nie usuwa wspomnianych plików, ponieważ być może nie będziesz chciał natychmiast ponownie wykonać jego danych wyjściowych, a wspomniane pliki będą konieczne w trakcie kolejnego wykonania zapytań. Pliki tymczasowe usuń ręcznie, gdy na pewno nie będą już potrzebne.

■ `--offset=n, -o n`

Opcja powoduje pominięcie pierwszych n zdarzeń w pliku dziennika zdarzeń.

■ `--raw (boolean)`

Ta opcja jest używana w połączeniu z `--read-from-remote-server` w celu żądania plików binarnego dziennika zdarzeń ze wskazanego serwera w ich oryginalnym formacie zamiast domyślnym formacie tekstowym. Jedną z sytuacji, w której omawiana opcja pokazuje swoją użyteczność, jest tworzenie kopii zapasowej plików binarnego dziennika zdarzeń serwera. (W takim przypadku za pomocą opcji `--result-file` można określić nazwę pliku danych wyjściowych). Omawiana opcja została wprowadzona w MySQL 5.6.0.

■ `--read-from-remote-server, -R (boolean)`

Opcja powoduje odczyt plików binarnego dziennika zdarzeń przez połączenie sieciowe z serwerem i nakazuje wysłanie wspomnianych przez to połączenie. Należy użyć opcji `--read-from-remote-server` i podać wartości opcji `--host`, `--password`, `--port`, `--protocol`, `--socket` i `--user`, wskazując tym samym parametry połączenia. Bez `--read-from-remote-server` wymienione opcje są ignorowane.

■ `--result-file=file_name, -r file_name`

Opcja powoduje zapis danych wyjściowych do wskazanego pliku, o ile nie zostanie podana opcja `--raw`. W przypadku użycia opcji `--raw` omawiana opcja wskazuje nazwę lokalnych kopii plików binarnego dziennika zdarzeń pobranych z serwera. W takim przypadku pliki danych wyjściowych muszą mieć takie same nazwy jak pliki dzienników zdarzeń w serwerze głównym, wraz z wszelkimi prefiksami wskazanymi przez wartość opcji `--result-file`. Wspomniana wartość może zaczynać się od nazwy katalogu, wtedy pliki będą zapisywane w podanym katalogu. Na przykład, pliki kopii zapasowej utworzone dnia 2012-01-28 mogą być oznaczone jako `--result-file=2012-01-28-`, a pliki danych wyjściowych będą rozpoczynały się od podanej daty i znajdą się w katalogu bieżącym. Aby zapisać pliki w katalogu `/var/backup`, należy użyć opcji w postaci `--result-file=/var/backup/2012-01-28-`.

■ `--server-id=n`

Opcja powoduje zapis jedynie zdarzeń utworzonych przez serwer o podanym identyfikatorze.

■ `--set-charset=charset`

Opcja powoduje umieszczanie w danych wyjściowych zapytań SET NAMES.

■ `--short-form, -s`

Opcja powoduje wyświetlanie jedynie zapytań istniejących w dzienniku zdarzeń. Wszelkie informacje dodatkowe znajdujące się w dzienniku zdarzeń i powiązane z zapytaniami będą pominięte, podobnie jak zdarzenia oparte na rekordach.

■ `--start-datetime=date_time`

Opcja powoduje, że program `mysqlbinlog` rozpocznie odczyt zdarzeń binarnego dziennika, począwszy od pierwszego zdarzenia, które ma datę i godzinę takie jak podane w opcji lub późniejsze. Wartość *date_time* powinna być podana w poprawnym formacie DATETIME, w czasie lokalnym dla komputera, w którym uruchomiono program `mysqlbinlog`. Jeśli to konieczne, należy cytować tę wartość.

■ `--start-position=n, -j n`

Opcja powoduje odczyt zdarzeń binarnego dziennika, począwszy od wskazanego położenia w pierwszym pliku dziennika wymienionym w wierszu poleceń.

- `--stop-datetime=date_time`

Opcja powoduje zatrzymanie odczytu zdarzeń binarnego dziennika, począwszy od pierwszego zdarzenia, które ma datę i godzinę takie jak podane w opcji lub późniejsze. Wartość `date_time` powinna być podana w poprawnym formacie DATETIME, w czasie lokalnym dla komputera, w którym uruchomiono program `mysqlbinlog`. Jeśli to konieczne, należy cytować tę wartość.

- `--stop-never (boolean)`

Ta opcja jest używana w połączeniu `--read-from-remote-server` w celu zachowania połączenia z serwerem. Można ją wykorzystać do wyświetlania zdarzeń dziennika w trakcie ich zapisywania przez serwer. Omawianej opcji można również użyć wraz z opcją `--raw` i tym samym utworzyć nieustannie uaktualniany dziennik kopii zapasowej. Ta opcja została wprowadzona w MySQL 5.6.0.

- `--stop-never-slave-server-id=server_id`

Opcja wskazuje identyfikator zgłaszany serwerowi podczas użycia opcji `--stop-never`. Identyfikator domyślny to 65535, ale może wystąpić konieczność wyraźnego zdefiniowania identyfikatora, gdy serwer podległy lub inny egzemplarz `mysqlbinlog` używa identyfikatora domyślnego. Ta opcja została wprowadzona w MySQL 5.6.0.

- `--stop-position=n`

Ta opcja powoduje zatrzymanie odczytu zdarzeń binarnego dziennika w podanym położeniu w ostatnim pliku dziennika wskazanym w wierszu poleceń.

- `--to-last-log, -t (boolean)`

Podczas odczytu plików dziennika z serwera (co wymaga opcji `--read-from-remote-server`) ta opcja powoduje, że pliki binarnego dziennika zdarzeń będą odczytywane aż do ostatniego pliku znajdującego się w serwerze zamiast do końca wskazanego pliku. Aby upewnić się, że zostały pobrane z serwera wszystkie informacje binarnego dziennika zdarzeń, należy użyć opcji `--to-last-log`. (Warto w tym miejscu pamiętać o jednym: jeśli zdarzenia są przekazywane do przetworzenia w tym samym serwerze, można doprowadzić do powstania pętli działającej w nieskończoność).

- `--verify-binlog-checksum, -c (boolean)`

Opcja powoduje włączenie sprawdzania sum kontrolnych w zdarzeniach. Ta opcja została wprowadzona w wersji MySQL 5.6.1.

F.10.3. Zmienne dla `mysqlbinlog`

Wymienione tutaj zmienne mogą być ustawione za pomocą poleceń podanych w podpunkcie F.2.1.2, zatytułowanym „Ustawianie zmiennych programu”.

■ `open_files_limit`

Zmienna określa liczbę zarezerwowanych deskryptorów plików.

Wartość domyślna wynosi 64.

F.11. mysqlcheck

`mysqlcheck` to program kliencki przeznaczony do sprawdzania i naprawy tabel. Zapewnia on interfejs wiersza poleceń dla następujących zapytań administracyjnych: `CHECK TABLE`, `ANALYZE TABLE`, `OPTIMIZE TABLE` i `REPAIR TABLE`.

Wszystkie opcje `mysqlcheck` są obsługiwane przez table MyISAM. Program ma również możliwość sprawdzania i analizowania tabel InnoDB.

Program `mysqlcheck` może być uruchomiony w jednym z trzech trybów:

```
mysqlcheck [options] db_name [tbl_name] ...
mysqlcheck [options] --databases db_name ...
mysqlcheck [options] --all-databases
```

W pierwszym trybie program `mysqlcheck` sprawdza wskazane table w podanej bazie danych. Jeżeli nie zostanie podana żadna tabela, program `mysqlcheck` sprawdza wszystkie table w bazi danych. W drugim trybie wszystkie argumenty są traktowane jako nazwy baz danych, a program sprawdza wszystkie table w wymienionych bazach danych. Z kolei w trzecim trybie program `mysqlcheck` sprawdza wszystkie table we wszystkich bazach danych.

F.11.1. Opcje standardowe obsługiwane przez mysqlcheck

<code>--bind-address</code>	<code>--default-character-set</code>	<code>--protocol</code>
<code>--character-sets-dir</code>	<code>--help</code>	<code>--shared-memory-base-name</code>
<code>--compress</code>	<code>--host</code>	<code>--silent</code>
<code>--debug</code>	<code>--password</code>	<code>--socket</code>
<code>--debug-check</code>	<code>--pipe</code>	<code>--user</code>
<code>--debug-info</code>	<code>--plugin-dir</code>	<code>--verbose</code>
<code>--default-auth</code>	<code>--port</code>	<code>--version</code>

Ogólny opis powyższych opcji znajdziesz w punkcie F.2.1, zatytułowanym „Opcje standardowe programu MySQL”. Program `mysqlcheck` obsługuje także standardowe opcje SSL wymienione w podpunkcie F.2.1.1, zatytułowanym „Opcje standardowe SSL”.

F.11.2. Opcje charakterystyczne dla mysqlcheck

Program `mysqlcheck` obsługuje wymienione poniżej opcje wpływające na sposób przetwarzania tabel. W dalszej części podrozdziału znajdziesz opis odpowiedników między wymienionymi opcjami i odpowiadającymi im zapytaniami SQL.

- `--all-databases, -A` (*boolean*)

Przetworzenie wszystkich tabel we wszystkich bazach danych.

- `--analyze, -a`

Przeprowadzenie analizy tabeli za pomocą zapytania `ANALYZE TABLE`. (Na przykład, spowoduje to przeprowadzenie analizy rozproszenia wartości kluczy). Wyniki analizy mogą pomóc optymalizatorowi zapytań w znacznie szybszym wykonywaniu operacji wyszukiwania na podstawie indeksu oraz operacji złączeń.

- `--all-in-1, -1` (*boolean*)

Bez tej opcji program `mysqlcheck` wykonuje oddzielne zapytania do poszczególnych tabel. Omawiana opcja powoduje, że `mysqlcheck` grupuje tabele według baz danych i wymienia w pojedynczym zapytaniu wszystkie tabele konkretnej bazy danych.

- `--auto-repair` (*boolean*)

Jeżeli w którejkolwiek ze sprawdzanych tabel zostaną znalezione problemy, po fazie sprawdzania nastąpi przejście do drugiej fazy, czyli naprawy.

- `--check, -c`

Wykonanie zapytania `CHECK TABLE` w celu sprawdzenia pod kątem błędów. To jest domyślnie podejmowana akcja, jeśli żadna nie zostanie wyraźnie podana w poleceniu.

- `--check-only-changed, -C`

Opcja powoduje sprawdzenie jedynie tabel, które uległy zmianie od chwili ich ostatniego sprawdzenia lub nie zostały prawidłowo zamknięte.

- `--check-upgrade, -g`

Opcja powoduje sprawdzenie, czy tabele są zgodne z bieżącą wersją MySQL. Wraz z opcją `--auto-repair`, po znalezieniu niezgodności następuje próba automatycznej naprawy tabeli. Omawiana opcja włącza opcje `--fix-db-names` i `--fix-table-names`.

- `--databases, -B` (*boolean*)

Wszystkie argumenty zostaną zinterpretowane jako nazwy baz danych, a program `mysqlcheck` sprawdzi wszystkie tabele we wszystkich bazach danych.

- `--extended, -e` (*boolean*)

Przeprowadzenie znacznie dokładniejszej operacji sprawdzenia tabeli. Jeżeli opcja zostanie użyta wraz z `--repair`, użyta będzie o wiele bardziej rozbudowana, choć jednocześnie wolniejsza metoda naprawy niż w przypadku zastosowania samej opcji `--repair`.

- `--fast, -F` (*boolean*)

Opcja powoduje sprawdzenie jedynie tabel, które nie zostały prawidłowo zamknięte.

- **--fix-db-names** (*boolean*)
Sprawdzenie nazw baz danych i konwersja ich kodowania znaków w nazwach ze względu na zmiany wprowadzone między wersjami MySQL 5.0 i MySQL 5.1.
- **--fix-table-names** (*boolean*)
Sprawdzenie nazw tabel i widoków oraz konwersja ich kodowania znaków w nazwach ze względu na zmiany wprowadzone między wersjami MySQL 5.0 i MySQL 5.1.
- **--force, -f** (*boolean*)
Opcja powoduje, że program `mysqlcheck` będzie kontynuował działanie nawet po wystąpieniu błędów.
- **--medium-check, -m**
Przeprowadzenie operacji sprawdzania tabel za pomocą metody szybszej niż w przypadku opcji `--extended`, ale nieco mniej dokładnej. Ten rodzaj operacji sprawdzenia powinien być wystarczający w większości przypadków.
- **--optimize, -o**
Przeprowadzenie operacji optymalizacji tabeli za pomocą zapytania `OPTIMIZE TABLE`.
- **--quick, -q** (*boolean*)
W przypadku operacji sprawdzania tabeli ta opcja powoduje pominięcie sprawdzania łączy w rekordach danych. Jeśli zostanie użyta wraz z `--repair`, wtedy spowoduje naprawę jedynie pliku indeksu, natomiast plik danych pozostanie nienaruszony.
- **--repair, -r**
Przeprowadzenie operacji naprawy za pomocą zapytania `REPAIR TABLE`. Tego rodzaju naprawa z reguły usuwa większość problemów za wyjątkiem występowania zduplikowanych wartości w indeksie, który powinien zawierać jedynie unikalne wartości.
- **--tables**
Nadpisuje `--databases` i powoduje, że znajdujące się po niej argumenty będą interpretowane jako nazwy tabel.
- **--use-frm** (*boolean*)
Kiedy ta opcja zostanie użyta wraz z `--repair`, wtedy przeprowadza operację naprawy tabeli używającą pliku `.frm` do ponownej inicjalizacji pliku indeksu oraz do ustalenia sposobu interpretacji zawartości pliku danych pozwalającej na ponowne utworzenie indeksów. Omawiana opcja może być użyteczna, gdy indeks został usunięty lub nieodwracalnie uszkodzony. Jednak tę opcję należy traktować jako ostatnią deskę ratunku i używać jej *tylko* wtedy, gdy tabela została utworzona za pomocą tej samej wersji MySQL. W przeciwnym razie ryzykujesz dalsze uszkodzenie tabeli.

■ `--write-binlog` (*boolean*)

Opcja powoduje zapis zapytań `ANALYZE TABLE`, `OPTIMIZE TABLE` i `REPAIR TABLE` w binarnym dzienniku zdarzeń, co oznacza, że będą przekazywane do serwerów podległych replikacji. Ta opcja jest domyślnie włączona. Aby ją wyłączyć, należy użyć `--skip-write-binlog`.

Przedstawione poniżej tabele pokazują relacje między opcjami programu `mysqlcheck` i wykonywanymi przez ten program zapytaniami SQL.

Opcje sprawdzania tabeli (tylko dla tabel InnoDB i MyISAM) wymieniono w tabeli F.7.

Tabela F.7. Opcje sprawdzania tabel InnoDB i MyISAM

Opcja	Zapytanie odpowiadające opcji
<code>--check</code>	<code>CHECK TABLE lista_tabel</code>
<code>--check-only-changed</code>	<code>CHECK TABLE lista_tabel CHANGED</code>
<code>--extended</code>	<code>CHECK TABLE lista_tabel EXTENDED</code>
<code>--fast</code>	<code>CHECK TABLE lista_tabel FAST</code>
<code>--medium-check</code>	<code>CHECK TABLE lista_tabel MEDIUM</code>
<code>--quick</code>	<code>CHECK TABLE lista_tabel QUICK</code>

Silnik bazy danych InnoDB nie obsługuje różnego rodzaju operacji sprawdzania. Dlatego też w przypadku tabel InnoDB program `mysqlcheck` traktuje wszystkie opcje w tabeli F.7 jako `--check`.

Opcje analizy tabeli (tylko tabele InnoDB i MyISAM) wymieniono w tabeli F.8.

Tabela F.8. Opcje analizy tabel InnoDB i MyISAM

Opcja	Zapytanie odpowiadające opcji
<code>--analyze</code>	<code>ANALYZE TABLE lista_tabel</code>

Opcje naprawy tabeli (tylko dla tabel MyISAM) wymieniono w tabeli F.9.

Tabela F.9. Opcje naprawy tabel MyISAM

Opcja	Zapytanie odpowiadające opcji
<code>--repair</code>	<code>RAPAIR TABLE lista_tabel</code>
<code>--repair --quick</code>	<code>RAPAIR TABLE lista_tabel QUICK</code>
<code>--repair --extended</code>	<code>RAPAIR TABLE lista_tabel EXTENDED</code>
<code>--repair --use-frm</code>	<code>RAPAIR TABLE lista_tabel USE_FRM</code>

Opcje optymalizacji tabeli (tylko dla tabel MyISAM) wymieniono w tabeli F.10.

Tabela F.10. Opcje optymalizacji tabel MyISAM

Opcja	Zapytanie odpowiadające opcji
--optimize	OPTIMIZE TABLE <i>lista_tabel</i>

F.12. mysqld

mysqld to serwer MySQL. Programom klienckim zapewnia dostęp do baz danych, a więc musi być uruchomiony, ponieważ w przeciwnym razie klienci nie będą mogli korzystać z baz danych zarządzanych przez serwer. Podczas uruchamiania mysqld otwiera interfejsy sieciowe, a na których nasłuchuje i oczekuje połączeń klientów. Serwer mysqld jest wielowątkowy, to znaczy zapewnia współbieżność między klientami oraz wykorzystuje wiele wątków do przetwarzania połączeń klientów.

Standardowym sposobem uruchomienia serwera jest po prostu podanie jego nazwy oraz listy wybranych opcji:

```
mysqld [options]
```

W systemie Windows serwer może być zainstalowany do działania jako usługa. Na przykład, serwer można skonfigurować do uruchamiania automatycznie wraz z systemem lub usunąć jako usługę:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqld" --install
C:\> mysqld --remove
```

Polecenie instalacyjne używa pełnej ścieżki dostępu do serwera. Jeżeli serwer został zainstalowany w innym katalogu, należy odpowiednio zmodyfikować ścieżkę dostępu. Domyślną nazwą usługi jest MySQL. Istnieje możliwość samodzielnego zdefiniowania nazwy usługi lub jej usunięcia za pomocą poniższych poleceń (każde polecenie powinno być wpisane w jednym wierszu):

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqld" --install service_name
C:\> mysqld --remove service_name
```

Dzięki takiemu rozwiązaniu wiele serwerów może działać pod różnymi nazwami usług. W przypadku braku argumentu *service_name* lub podania MySQL serwer użyje MySQL jako nazwy usługi i odczyta grupę [mysqld] ze standardowych plików opcji. Podanie dla argumentu *service_name* innej nazwy niż MySQL powoduje użycie przez serwer wskazanej nazwy i odczyt grup [mysqld] i [nazwa_usługi] w standardowych plikach opcji.

Opcja --defaults-file jest dozwolona po nazwie usługi i umożliwia wskazanie dodatkowego pliku opcji odczytywanego przez serwer podczas jego uruchamiania (polecenie powinno być wprowadzone w jednym wierszu):

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqld" --install
service_name --defaults-file=file_name
```

W takim przypadku argument *service_name* jest wymagany.

Przedstawione wcześniej informacje o opcji --install mają zastosowanie także względem opcji --install-manual.

F.12.1. Opcje standardowe obsługiwane przez mysqld

```
--character-sets-dir      --port                      --user
--debug                  --shared-memory-base-name --verbose
--help                   --socket                   --version
```

Ogólny opis powyższych opcji znajdziesz w punkcie F.2.1, zatytułowanym „Opcje standardowe programu MySQL”. Serwer `mysqld` obsługuje także standardowe opcje SSL wymienione w podpunkcie F.2.1.1, zatytułowanym „Opcje standardowe SSL”.

Opcja `--help` wyświetla jedynie krótkie informacje dotyczące sposobu użycia `mysqld`. Aby wyświetlić pełny komunikat pomocy, należy wydać poniższe polecenie:

```
% mysqld --verbose --help
```

Pełny komunikat pomocy wyświetla także zmienne systemowe, które mogą być ustawione z poziomu wiersza poleceń. Omówienie wspomnianych zmiennych znajdziesz w dodatku D, zatytułowanym „Przewodnik po zmiennych systemowych, stanu i użytkownika”.

Wprawdzie opcja `--socket` jest obsługiwana, ale odpowiadająca jej krótka forma (`-S`) już nie. W systemie Windows opcja `--socket` powoduje ustawienie nazwy potoku, o ile serwer obsługuje tego rodzaju połączenia.

W systemach UNIX, jeśli zostanie podana opcja `--user`, to wskazuje nazwę użytkownika lub liczbowy identyfikator konta użytkownika używanego do uruchomienia serwera. W takim przypadku podczas uruchamiania serwera będzie on szukał w pliku haseł wartości określających nazwy użytkownika i grupy, a następnie zmieni nazwę użytkownika i grupy na znalezione wartości. W ten sposób serwer będzie działał wraz z uprawnieniami wskazanego użytkownika, a nie użytkownika `root`. (Aby możliwe było użycie opcji `--user`, serwer musi być uruchomiony jako `root`. W przeciwnym razie nie może zmienić identyfikatora użytkownika i generuje komunikat ostrzeżenia).

F.12.2. Opcje charakterystyczne dla mysqld

Ze wszystkich programów MySQL, `mysqld` ma najbardziej rozbudowany zestaw opcji. Jednak jeśli porównasz dane wyjściowe polecenia `mysqld --verbose --help` z przedstawioną tutaj listą opcji, przekonasz się, że wspomniany komunikat pomocy zawiera informacje o wielu niewymienionych tutaj opcjach. Wynika to z faktu umieszczenia w komunikacie pomocy także wielu „opcji” będących tak naprawdę zmiennymi systemowymi, których wartość można ustawić podczas uruchamiania serwera. W celu zaoszczędzenia miejsca, jeśli opcja ma taką samą nazwę jak zmienna systemowa, została tutaj pominięta. Jeżeli na poniższej liście nie znajdujesz szukanej opcji serwera, sprawdź dodatek D, zatytułowany „Przewodnik po zmiennych systemowych, stanu i użytkownika”. Na przykład, komunikat pomocy `mysqld` zawiera opcję `--general-log`, pozwalającą na włączenie ogólnego dziennika zapytań. Informacje na jej temat możesz znaleźć w opisie zmiennej systemowej `general_log`.

Pierwsza przedstawiona tutaj lista zawiera opis opcji ogólnych. Następnie znajdują się listy opcji charakterystycznych dla Windows oraz opcji replikacji.

■ **--allow-suspicious-udfs** (*boolean*)

Umożliwia serwerowi wczytywanie funkcji zdefiniowanych przez użytkownika (UDF), które mogą definiować jedynie symbol odpowiadający nazwie funkcji, ale nie żadne symbole powiązane ze standardowymi procedurami. Możliwość wczytywania UDF jest domyślnie wyłączona jako forma zabezpieczenia przed wczytywaniem funkcji, które tak naprawdę nie będą funkcjami zdefiniowanymi przez użytkownika.

■ **--ansi, -a**

Opcja nakazuje serwerowi zastosowanie dla określonych typów składni standardowego zachowania SQL zamiast charakterystycznego dla MySQL. Ta opcja może być używana w celu zapewnienia większej zgodności serwera ze standardami.

Omawiana opcja jest odpowiednikiem ustawienia zmiennej systemowej `sql_mode` w taki sposób, aby zawierała wartości trybów `REAL_AS_FLOAT`, `PIPES_AS_CONCAT`, `ANSI_QUOTES`, `IGNORE_SPACE` i `ONLY_FULL_GROUP_BY`.

■ **--archive[=*state*]**

Opcja nadzoruje stan aktywacji wtyczki silnika bazy danych ARCHIVE, o ile został on dołączony do dystrybucji MySQL. Listę dozwolonych wartości *state* znajdziesz w opisie opcji `--plugin_name`.

■ **--basedir=*dir_name*, -b *dir_name***

Ścieżka dostępu do katalogu głównego instalacji MySQL. Wiele innych ścieżek dostępu jest określanych względem wskazanego tutaj katalogu, o ile wspomniane ścieżki są względnymi ścieżkami dostępu.

■ **--bind-address=*addr***

Opcja powoduje, że wskazany adres IP będzie nasłuchiwany pod kątem połączeń klientów za pomocą TCP/IP. Począwszy od wersji MySQL 5.6.6, wartością domyślną tej opcji jest `*` (nasłuchiwanie na wszystkich interfejsach IPv4 i IPv6). Przed wydaniem MySQL 5.6.6 wartością domyślną był adres `0.0.0.0` (nasłuchiwanie na wszystkich interfejsach IPv4). Więcej informacji na ten temat znajdziesz w punkcie 12.2.4, zatytułowanym „W jaki sposób serwer nasłuchuje połączeń?”.

■ **--binlog-row-event-max-size=*n***

Maksymalna dozwolona wielkość dla zdarzeń opartych na rekordach. Gdy tylko to możliwe, `mysqld` próbuje grupować rekordy w zdarzenia nie większe niż podana tutaj wartość. Podana tutaj wartość będzie skrócona do najbliższej niezerowej wielokrotności 256. Począwszy od MySQL 5.6.6, wartością domyślną jest 8 KB, natomiast we wcześniejszych 1 KB.

■ **--blackhole[=*state*]**

Opcja nadzoruje stan aktywacji wtyczki silnika bazy danych BLACKHOLE, o ile został on dołączony do dystrybucji MySQL. Listę dozwolonych wartości *state* znajdziesz w opisie opcji `--plugin_name`.

- **--bootstrap**
Ta opcja jest używana przez skrypty instalacyjne podczas pierwszej instalacji MySQL.
- **--character-set-client-handshake** (*boolean*)
Ta opcja nakazuje serwerowi użycie dostarczanych przez klienta informacji o kodowaniu znaków. Omawiana opcja jest włączona domyślnie; użycie **--skip-character-set-client-handshake** powoduje zignorowanie informacji podanych przez klienta, co jest zachowaniem w wersji MySQL 4.0.
- **--character-set-server=charset, -C charset**
Ta opcja określa domyślne kodowanie znaków serwera.
- **--chroot=dir_name, -r dir_name**
Uruchomienie serwera MySQL przypisanego do wskazanego katalogu, który będzie uważany za katalog główny serwera. Więcej informacji na temat uruchamiania programów w środowisku chroot() znajdziesz w podręczniku użytkownika systemu UNIX.
- **--collation-server=collation**
Domyślna kolejność sortowania dla domyślnego kodowania znaków w serwerze.
- **--core-file**
Po wystąpieniu błędów o znaczeniu krytycznym, przed zakończeniem działania serwer wygeneruje plik core.
- **--datadir=dir_name, -h dir_name**
Ścieżka dostępu do katalogu danych MySQL.
- **--default-time-zone=tz_name**
Przypisanie *tz_name* jako domyślnej strefy czasowej serwera. Dostępne wartości wskazujące strefy czasowe omówiono w punkcie 12.6.1, zatytułowanym „Konfiguracja obsługi stref czasowych”. Omawiana opcja powoduje ustawienie zmiennej systemowej *time_zone*, a nie *system_time_zone*.
- **--delay-key-write[=val]**
Ustawienie trybu używanego przez serwer do obsługi opóźnionego zapisu kluczy dla plików MyISAM. Wartością *val* może być ON (opóźniony zapis kluczy dla poszczególnych tabel, zgodnie z wartością *DELAY_KEY_WRITE* podaną podczas tworzenia tabel, to jest wartość domyślna, gdy nie zostanie podana żadna wartość), OFF (wyłączenie opóźnionego zapisu kluczy dla tabel MyISAM) lub ALL (opóźniony zapis kluczy dla wszystkich tabel MyISAM). Wartości OFF i ALL wymuszają zastosowanie ustawienia niezależnie od definicji *DELAY_ON_WRITE* użytej podczas tworzenia tabel.
- **--des-key-file=file_name**
Nazwa pliku zawierającego klucze DES dla funkcji *DES_ENCRYPT()* i *DES_DECRYPT()*. Opis formatu pliku znajdziesz w punkcie poświęconym funkcji *DES_ENCRYPT()* w dodatku C, zatytułowanym „Przewodnik po operatorach i funkcjach”.

■ `--exit-info[=n], -T[n]`

Opcja powoduje, że serwer będzie generował informacje debugowania w trakcie jego zamykania. Jeżeli wartość *n* będzie podana po -T, między nimi nie może znajdować się spacja, ponieważ wtedy wartość *n* nie zostanie prawidłowo zinterpretowana.

■ `--external-locking` (*boolean*)

Włączenie zewnętrznego nakładania blokad (na poziomie systemu plików) w przypadku systemów takich jak Linux, w których zewnętrzne nakładanie blokad jest domyślnie wyłączone.

Zewnętrzne nakładanie blokad jest problematyczne, ponieważ w niektórych systemach nie działa niezawodnie, a poza tym jest efektywne jedynie dla operacji odczytujących tabele, na przykład podczas sprawdzania tabeli.

■ `--federated[=state]`

Opcja nadzoruje stan aktywacji wtyczki silnika bazy danych FEDERATED, o ile został on dołączony do dystrybucji MySQL. Listę dozwolonych wartości *state* znajdziesz w opisie opcji `--plugin_name`.

■ `--gdb`

Konfiguracja procedur obsługi sygnałów użytecznych podczas debugowania za pomocą gdb.

■ `--ignore-db-dir=dir_name`

Opcja pozwala wskazać nazwę znajdującego się w katalogu danych MySQL podkatalogu, który nie będzie traktowany jako katalog bazy danych przez zapytanie `SHOW DATABASES` lub tabele `INFORMATION_SCHEMA`. Omawianą opcję można podać wielokrotnie. Katalogi wymienione w tej opcji są dostępne w trakcie działania serwera jako wartości zmiennej systemowej `ignore_db_dirs`. Warto pamiętać, że użycie omawianej opcji wraz z pustą wartością spowoduje wyczyszczenie listy i nadpisanie wszelkich wcześniejszych egzemplarzy. Ta opcja została wprowadzona w MySQL 5.6.3.

■ `--innodb[=state]`

Opcja nadzoruje stan aktywacji wtyczki silnika bazy danych InnoDB. Listę dozwolonych wartości *state* znajdziesz w opisie opcji `--plugin_name`. Ponieważ InnoDB jest domyślnym silnikiem bazy danych, pozostaje włączony domyślnie. Jeżeli nie używasz tabel InnoDB, możesz go wyłączyć za pomocą opcji `--innodb=OFF`. W takim przypadku trzeba ustawić również zmienną systemową `default_storage_engine` (począwszy od MySQL 5.6.3, także zmienną `default_tmp_storage_engine`) i podać inny domyślny silnik bazy danych, ponieważ w przeciwnym razie serwer nie będzie uruchamiany. Patrz punkt 12.5.2, zatytułowany „Wybór domyślnego silnika bazy danych”.

■ `--innodb-status-file` (*boolean*)

Opcja powoduje, że informacje generowane przez zapytanie `SHOW INNODB STATUS` będą okresowo zapisywane w pliku o nazwie `innodb_status.nnnnnnn`,

gdzie *nnnnnn* to identyfikator procesu. Te pliki stanu będą usuwane jedynie podczas prawidłowego zamknięcia serwera. Jednak od czasu do czasu powinien je usuwać, jeśli nie są dłużej potrzebne.

■ **--innodb-xxx**

Wiele innych parametrów InnoDB jest dostępnych w postaci zmiennych systemowych, które mogą być ustawione podczas uruchamiania serwera. Patrz punkt D.1.1, zatytułowany „Zmienne systemowe InnoDB”.

■ **--language=*lang_name*, -l *lang_name***

Opcja pozwala na wybór języka, w którym będą wyświetlane komunikaty. Ta opcja jest uznawana za przestarzałą. W celu wyboru języka dla komunikatów błędów i katalogu zawierającego pliki ustawień językowych należy użyć zmiennych systemowych odpowiednio *lc_messages* i *lc_messages_dir*.

■ **--log[=*file_name*], -l [*file_name*]**

Opcja pozwala na włączenie ogólnego dziennika zapytań. Została uznana za przestarzałą i usunięta w MySQL 5.6. Zamiast niej należy używać zmiennych systemowych *general_log* i *general_log_file*.

■ **--log-bin[=*file_name*]**

Opcja powoduje włączenie binarnego dziennika zdarzeń. Wartość *file_name* określa nazwę bazową dla plików binarnego dziennika zdarzeń. Jeżeli nie zostanie podana, nazwą pliku dziennika zdarzeń będzie *HOSTNAME-bin.nnnnnn*, gdzie *nnnnnn* to numer sekwencji zwiększany przez serwer o jeden po utworzeniu każdego kolejnego pliku. Wspomniane pliki będą tworzone w katalogu danych. Natomiast jeśli wartość *file_name* zostanie podana jako względna ścieżka dostępu, serwer będzie ją interpretował względem katalogu danych MySQL.

■ **--log-bin-index=*file_name***

Włączenie pliku indeksu binarnego dziennika zdarzeń. Jeżeli wartość *file_name* nie zostanie podana, domyślna nazwa jest taka sama jak nazwa bazowa plików binarnego dziennika zdarzeń i ma rozszerzenie *.index*. Natomiast jeśli wartość *file_name* zostanie podana jako względna ścieżka dostępu, serwer będzie ją interpretował względem katalogu danych MySQL.

■ **--log-isam[=*file_name*]**

Włączenie rejestracji pliku dziennika. Ta opcja jest używana jedynie w operacjach debugowania MyISAM. Jeżeli nie podasz wartości *file_name*, domyślnie będzie to plik *myisam.log* tworzony w katalogu danych MySQL.

■ **--log-raw (*boolean*)**

Opcja wyłącza zapis zapytań w ogólnym dzienniku zapytań, dzienniku wolno wykonywanych zapytań oraz binarnym dzienniku zdarzeń, aby uniemożliwić pojawianie się haseł w postaci zwykłego tekstu. Takie rozwiązanie jest przeznaczone do stosowania jedynie w celach debugowania i testowania — przecież nie chcesz, aby hasła były umieszczane w dziennikach zdarzeń serwera. Omawiana opcja została wprowadzona w MySQL 5.6.3.

- `--log-short-format` (*boolean*)
Opcja powoduje zapis informacji w dziennikach binarnym oraz wolno wykonywanych zapytań, o ile wspomniane dzienniki są dostępne.
- `--log-slow-admin-statements` (*boolean*)
Operacje administracyjne, na przykład wykonywane przez zapytania ALTER TABLE lub OPTIMIZE TABLE, mogą być wolne, ale domyślnie wolno wykonywane zapytania nie są rejestrowane w dzienniku. Ta opcja powoduje ich zarejestrowanie w dzienniku, jeśli są wykonywane wolno.
- `--log-slow-queries[=file_name]`
Opcja powoduje włączenie dziennika wolno wykonywanych zapytań. Została uznana za przestarzałą i usunięta w MySQL 5.6. Zamiast niej należy używać zmiennych systemowych `slow_query_log` i `slow_query_log_file`.
- `--log-tc=file_name`
Ścieżka dostępu do pliku koordynatora transakcji (dla transakcji XA). Ta opcja jest nieużywana.
- `--log-tc-size=n`
Wielkość pliku koordynatora transakcji.
- `--log-warnings[=n], -W[n]`
Opcja powoduje zapis w dzienniku błędów pewnych ostrzeżeń, które nie mają znaczenia krytycznego. Ta opcja jest domyślnie włączona. Jeżeli omawiana opcja zostanie podana bez żadnej wartości, nastąpi włączenie ostrzeżeń. Z kolei wartość 0 lub 1 powoduje odpowiednio wyłączenie i włączenie ostrzeżeń. Podanie opcji bez wartości lub z wartością 2 powoduje włączenie rejestrowania komunikatów dotyczących przerwanych połączeń oraz błędów związanych z brakiem uprawnień dostępu. Jeżeli wartość *n* będzie podana po -W, między nimi nie może znajdować się spacja, ponieważ wtedy wartość *n* nie zostanie prawidłowo zinterpretowana.
- `--memlock` (*boolean*)
Zablokowanie serwera w pamięci, o ile to możliwe. Ta opcja jest efektywna jedynie w systemach takich jak Solaris lub Linux, które mogą zablokować procesy w pamięci. Może wystąpić konieczność uruchomienia serwera przez użytkownika root.
- `--myisam-block-size=size`
Wielkość bloku indeksu tabeli MyISAM.
- `--old-style-user-limits` (*boolean*)
Dla kont użytkowników MySQL można zdefiniować pewne ograniczenia dotyczące ich aktywności, co zostało omówione w podpunkcie 13.2.2.5, zatytułowanym „Ograniczanie zasobów dostępnych dla użytkownika”. Ograniczenia względem konta są nakładane z jednoczesnym ignorowaniem komputera, z którego nawiązano połączenie. Jednak za pomocą opcji `--old-style-user-limits`

można włączyć starą metodę nakładania ograniczeń. (Przed wersją MySQL 5.0.3 ograniczenia były nakładane oddzielnie względem komputerów, z których nawiązano połączenie z danym kontem użytkownika).

■ **--plugin-load=plugin_list**

Ta opcja powoduje wczytanie wtyczek wskazanych przez wartości opcji, którą powinna być lista składająca się z jednego lub więcej rozdzielonych przecinkami specyfikatorów `lib_name`. Podanie nazwy wtyczki i biblioteki powoduje, że serwer wczytuje jedynie wskazaną wtyczkę z biblioteki. Z kolei podanie nazwy biblioteki bez wtyczki powoduje, że serwer wczytuje wszystkie wtyczki ze wskazanej biblioteki. Opcja `--plugin-load` zeruje listę wtyczek przeznaczonych do wczytania. Dlatego też w przypadku kilkukrotnego użycia omawianej opcji efekt przyniesie tylko ostatnie jej wystąpienie. Więcej informacji znajdziesz w podrozdziale 12.4, zatytułowanym „Interfejs wtyczek”.

■ **--plugin-load-add=plugin_list**

Opcja podobna w działaniu do `--plugin-load`, ale dodaje elementy do listy wtyczek przeznaczonych do wczytania bez wcześniejszego zerowania wspomnianej listy. Ta opcja została wprowadzona w MySQL 5.6.3.

■ **--plugin_name[=state]**

Jeżeli podczas uruchamiania serwera wczyta on wtyczkę, ponieważ została wbudowana, zarejestrowana w tabeli `mysql.plugin` bądź też podana za pomocą opcji `--plugin-load` lub `--plugin-load-add`, wtedy istnieje możliwość nadzorowania stanu aktywacji wtyczki. W tym celu należy użyć opcji o nazwie takiej samej jak nazwa wtyczki. Na przykład, opcja `--innodb` zapewnia możliwość kontroli aktywacji silnika bazy danych InnoDB. Dozwolone wartości `state` to `OFF` (wtyczka nie będzie aktywowana), `ON` (wtyczka aktywowana; to wartość domyślna w przypadku pominięcia `state`), `FORCE` (aktywacja wtyczki; w przypadku wystąpienia błędu próba uruchomienia serwera zakończy się niepowodzeniem) i `FORCE_PLUS_PERMANENT` (podobnie jak `FORCE`, ale uniemożliwia wyłączenie wtyczki w trakcie działania serwera). Więcej informacji znajdziesz w podrozdziale 12.4, zatytułowanym „Interfejs wtyczek”.

■ **--port-open-timeout=n**

Opcja określa w sekundach czas, przez który serwer powinien czekać na udostępnienie portu TCP/IP podczas uruchamiania. Wartością domyślną jest 0 (brak oczekiwania).

■ **--safe-mode**

Ta opcja jest przestarzała i zbędna.

■ **--safe-user-create (boolean)**

Ta opcja uniemożliwia tworzenie kont użytkowników przez użytkowników, którzy nie mają uprawnienia `INSERT` do tabeli uprawnień `mysql.user`.

■ **--skip-grant-tables** (*boolean*)

Wyłączenie użycia tabel uprawnień podczas weryfikacji połączeń klientów. W ten sposób każdy klient uzyskuje pełny dostęp do serwera MySQL. Ta opcja wyłącza również zapytania CREATE USER, DROP USER, RENAME USER, GRANT, REVOKE i SET PASSWORD, a także uniemożliwia serwerowi wczytanie jakichkolwiek wtyczek zarejestrowanych w tabeli mysql.plugin. Serwerowi można nakazać ponowne używanie tabel uprawnień, wykonując zapytanie FLUSH PRIVILEGES, polecenie mysqladmin flush-privileges lub ponownie go uruchamiając bez opcji --skip-grant-tables.

■ **--skip-host-cache**

Ta opcja powoduje wyłączenie bufora nazw komputerów. Począwszy od MySQL 5.6.5, jej użycie ma taki sam efekt jak ustawienie host_cache_size=0. Różnica polega na tym, że po użyciu omawianej opcji bufor nazw komputerów nie może być włączony w trakcie działania serwera.

■ **--skip-stack-trace**

Opcja powoduje wyłączenie wyświetlania stosu wywołań po wystąpieniu błędu.

■ **--symbolic-links, -s** (*boolean*)

W systemach UNIX ta opcja powoduje włączenie możliwości użycia dowiązań symbolicznych dla plików indeksu i danych tabel MyISAM (za pomocą opcji DATA DIRECTORY i INDEX DIRECTORY stosowanych podczas tworzenia tabeli). Z kolei w systemie Windows powoduje włączenie możliwości użycia dowiązań symbolicznych dla katalogów baz danych. Wymienione tutaj techniki zostały omówione w rozdziale 11., zatytułowanym „Katalog danych w MySQL”. Obsługa dowiązań symbolicznych dla baz danych w systemie Windows jest włączona domyślnie. Aby ją wyłączyć, należy użyć --skip-symbolic-links.

■ **--sysdate-is-now** (*boolean*)

Funkcja SYSDATE() zwraca datę i godzinę jej wywołania, podczas gdy NOW() zwraca godzinę, w której rozpoczęło się wykonywanie zapytania. Użycie omawianej opcji powoduje, że funkcja SYSDATE() będzie działała tak samo jak NOW().

■ **--tc-heuristic-recover=str**

Ta opcja jest nieużywana.

■ **--temp-pool** (*boolean*)

Ta opcja powoduje, że serwer używa małego zestawu nazw dla plików tymczasowych, zamiast tworzyć unikalne dla każdego pliku. W ten sposób można uniknąć pewnych problemów związanych z buforowaniem w systemie Linux, który jest jedynym systemem obsługującym tę opcję. Została ona włączona domyślnie; aby ją wyłączyć, należy użyć --skip-temp-pool.

■ **--tmpdir=dir_name, -t dir_name**

Ścieżka dostępu do katalogu przeznaczonego do przechowywania plików tymczasowych. Wartością tej opcji może być lista katalogów. W systemach UNIX nazwy katalogów należy rozdzielić dwukropkami, natomiast w Windows średnikami.

- `--transaction-isolation=level`

Opcja pozwala na ustawienie domyślnego poziomu izolacji transakcji. Dozwolone wartości *level* to READ-UNCOMMITTED, READ-COMMITTED, REPEATABLE-READ i SERIALIZABLE.

- `--transaction-read-only (boolean)`

Ta opcja określa, czy domyślny tryb dostępu do transakcji to tylko do odczytu. Omawiana opcja jest domyślnie wyłączona, co oznacza, że domyślny dostęp do transakcji to tryb odczyt/zapis. Tę opcję wprowadzono w MySQL 5.6.5.

F.12.2.1. Opcje w systemie Windows

Opcje wymienione w tym podpunkcie są dostępne w serwerach MySQL uruchomionych w systemie Windows. Wielkość liter w nazwach usług i nazwanych potokach nie ma znaczenia. Z kolei wielkość liter w nazwach pamięci współdzielonej ma znaczenie.

- `--console (boolean)`

Opcja powoduje wyświetlenie okna konsoli dla komunikatów błędów. Jeżeli użyta zostanie również opcja `--log-error`, wtedy komunikaty będą umieszczane w pliku dziennika zamiast w konsoli.

- `--install [service_name]`

Opcja powoduje instalację serwera jako usługi uruchamianej automatycznie wraz z systemem Windows. Jeżeli nazwa usługi nie zostanie podana, wtedy domyślnie będzie użyta nazwa MySQL.

- `--install-manual [service_name]`

Opcja powoduje instalację serwera jako usługi, ale nieuruchamianej automatycznie wraz z systemem Windows. W takim przypadku trzeba ręcznie uruchomić usługę. Jeżeli nazwa usługi nie zostanie podana, wtedy domyślnie będzie użyta nazwa MySQL.

- `--named-pipe (boolean)`

W przypadku serwerów MySQL zawierających obsługę nazwanych potoków możliwość nawiązywania tego rodzaju połączeń jest domyślnie wyłączona. Omawiana opcja włącza możliwość nawiązywania połączeń za pomocą nazwanych potoków. Domyślną stosowaną nazwą dla nazwanego potoku jest MySQL. Tę nazwę można zmienić za pomocą opcji `--socket`.

- `--remove [service_name]`

Opcja powoduje usunięcie serwera jako usługi. Jeżeli nazwa usługi nie zostanie podana, wtedy domyślnie będzie użyta nazwa MySQL.

- `--shared-memory (boolean)`

Opcja włącza możliwość nawiązywania połączeń za pomocą pamięci współdzielonej. Domyślną nazwą pamięci współdzielonej jest MySQL. Tę nazwę można zmienić za pomocą opcji `--shared-memory-base-name`.

- `--standalone`

Ta opcja powoduje uruchomienie serwera jako samodzielnego programu, a nie jako usługi.

F.12.2.2. Opcje replikacji

Opcje wymienione w tym podpunkcie dotyczą możliwości serwera MySQL w zakresie replikacji.

Opcja `--show-slave-auth-info` ma wpływ na dane wyjściowe generowane przez zapytanie `SHOW SLAVE HOSTS` wykonane w serwerze głównym, co omówiono w dodatku E, zatytułowanym „Przewodnik po składni SQL”.

- `--abort-slave-event-count=n`

Ta opcja jest używana przez zestaw testowy MySQL w celu przetestowania replikacji.

- `--binlog-do-db=db_name`

Opcja nakazuje serwerowi głównemu replikacji rejestrację uaktualnień tylko dla wskazanej bazy danych. Żadne inne bazy danych nie będą replikowane. W celu rejestrowania uaktualnień dla wielu baz danych tej opcji należy użyć wielokrotnie, za każdym razem podając nazwę tylko jednej bazy danych.

- `--binlog-ignore-db=db_name`

Opcja nakazuje serwerowi głównemu replikacji nieregistrowanie uaktualnień dla wskazanej bazy danych. W celu ignorowania uaktualnień dla wielu baz danych tej opcji należy użyć wielokrotnie, za każdym razem podając nazwę tylko jednej bazy danych.

Zwróć uwagę na pewien fakt: użycie tej opcji powoduje, że binarny dziennik zdarzeń nie będzie zawierał informacji wymaganych do przeprowadzenia operacji odzyskania zawartości bazy danych w przypadku awarii. Rozwiązaniem tego problemu jest zamiast omawianej opcji użycie `--replicate-ignore-db` w serwerze podległym replikacji.

- `--disconnect-slave-event-count=n`

Ta opcja jest używana przez zestaw testowy MySQL w celu przetestowania replikacji.

- `--master-info-file=file_name`

W przypadku serwera podległego replikacji ta opcja zawiera nazwę pliku przechowującego informacje o bieżącym stanie replikacji. Zawartością omawianego pliku są koordynaty replikacji (nazwa pliku binarnego dziennika zdarzeń w serwerze głównym replikacji oraz położenie w tym pliku), nazwa serwera głównego replikacji, nazwa użytkownika, hasło, numer portu, czas między kolejnymi próbami nawiązania połączenia oraz wartości opcji dotyczących SSL. Nazwą domyślną pliku jest *master.info* i znajduje się on w katalogu danych MySQL. Jeżeli wartość *file_name* zostanie podana jako względna ścieżka dostępu, serwer będzie ją interpretował względem katalogu danych MySQL.

- `--master-retry-count=n`

W przypadku serwera podległego replikacji ta opcja określa liczbę prób nawiązania połączenia z serwerem głównym replikacji, zanim operacja zakończy się

niepowodzeniem. W MySQL 5.6 ta opcja została uznana za zbędną, zamiast niej należy użyć opcji `MASTER_RETRY_COUNT` zapytania `CHANGE MASTER TO`.

- `--max-binlog-dump-events=n`

Ta opcja jest używana przez zestaw testowy MySQL w celu przetestowania replikacji.

- `--relay-log=file_name`

W przypadku serwera podległego replikacji ta opcja określa nazwę bazy dla plików dziennika przekazywania. (Wątek wejścia-wyjścia przechowuje w dzienniku przekazywania uaktualnienia odczytane z serwera głównego, natomiast wątek SQL odczytuje zapytania z dziennika przekazywania i wykonuje je). Domyślnie pliki dziennika przekazywania mają nazwy w postaci *HOSTNAME-relay-bin.nnnnnnn*, gdzie *nnnnnnn* to numer sekwencji zwiększany przez serwer o jeden po utworzeniu każdego kolejnego pliku. Wspomniane pliki będą tworzone w katalogu danych.

- `--relay-log-index=file_name`

W przypadku serwera podległego replikacji ta opcja określa nazwę pliku indeksu dziennika przekazywania. Domyślnie pliki dziennika przekazywania mają nazwy w postaci *HOSTNAME-relay-bin.index*, gdzie *HOSTNAME* to nazwa komputera, w którym działa serwer. Wspomniane pliki będą tworzone w katalogu danych. Jeśli wartość *file_name* zostanie podana jako względna ścieżka dostępu, serwer będzie ją interpretował względem katalogu danych MySQL.

- `--relay-log-info-file=file_name`

W przypadku serwera podległego replikacji ta opcja określa nazwę pliku informacyjnego dziennika przekazywania. Domyślną nazwą pliku jest *relay-log.info*, a sam plik znajduje się w katalogu danych MySQL.

- `--replicate-do-db=db_name`

Opcja nakazuje serwerowi podległemu replikację tylko wskazanej bazy danych. W celu replikacji wielu baz danych tej opcji należy użyć wielokrotnie, za każdym razem podając nazwę tylko jednej bazy danych.

- `--replicate-do-table=db_name.tbl_name`

Opcja nakazuje serwerowi podległemu replikację tylko wskazanej tabeli, która powinna być podana w formacie *nazwa_bazy_danych.nazwa_tabeli*. W celu replikacji wielu tabel tej opcji należy użyć wielokrotnie, za każdym razem podając nazwę tylko jednej tabeli.

- `--replicate-ignore-db=db_name`

Opcja nakazuje serwerowi podległemu niereplikowanie wskazanej bazy danych. W celu zignorowania wielu baz danych tej opcji należy użyć wielokrotnie, za każdym razem podając nazwę tylko jednej bazy danych.

- `--replicate-ignore-table=db_name.tbl_name`

Opcja nakazuje serwerowi podległemu niereplikowanie wskazanej tabeli. W celu zignorowania wielu tabel tej opcji należy użyć wielokrotnie, za każdym razem podając nazwę tylko jednej tabeli.

■ `--replicate-rewrite-db=master_db->slave_db`

Ta opcja nakazuje serwerowi podległemu replikacji traktowanie jednej bazy danych jak innej. Uaktualnienia wprowadzane w oryginalnej bazie danych *master_db* w serwerze głównym będą replikowane jako uaktualnienia w bazie danych *slave_db* w serwerze podległym. Operacja przepisywania jest stosowana tylko wtedy, gdy *master_db* jest domyślną bazą danych oraz jedynie dla zapytań operujących na tabelach w tej bazie danych. Kiedy opcja zostanie podana w wierszu poleceń, jej wartość powinna być cytowana, aby uniknąć interpretacji znaku > przez wiersz poleceń jako operatora przekierowania danych wyjściowych. Omawianą opcję można podać wielokrotnie. Serwer stosuje je w podanej kolejności i używa pierwszej reguły, dla której zostanie dopasowana wartość *master_db*.

Ta opcja będzie stosowana przed akcjami wskazanymi przez inne opcje `--replicate-xxx`. Dlatego też jeśli korzystasz z omawianej opcji, wspomniane opcje `--replicate-xxx` zawsze powinny wskazywać *slave_db* jako nazwę bazy danych.

■ `--replicate-same-server-id (boolean)`

Jeżeli ta opcja jest włączona, serwer nie będzie pomijał zdarzeń replikacji zawierających jego własny identyfikator. Ta opcja jest domyślnie wyłączona, aby uniknąć powstania pętli replikacji. W pewnych sytuacjach szczególnych można ją włączyć.

■ `--replicate-wild-do-table=pattern`

Ta opcja nakazuje serwerowi podległemu replikację jedynie tabel o nazwach dopasowanych do podanego wzorca. Aby ograniczyć replikację do zestawu wzorców, należy omawianą opcję podać wielokrotnie, za każdym razem definiując jeden wzorec.

■ `--replicate-wild-ignore-table=pattern`

Ta opcja nakazuje serwerowi podległemu replikacji ignorowanie tabel o nazwach dopasowanych do podanego wzorca. Aby zignorować wiele wzorców, należy omawianą opcję podać wielokrotnie, za każdym razem definiując jeden wzorec.

■ `--show-slave-auth-info (boolean)`

Ta opcja powoduje, że serwer główny replikacji będzie w danych wyjściowych zapytania `SHOW SLAVE HOSTS` wyświetlał nazwę użytkownika i hasło konta użytkownika w serwerze podległym replikacji.

■ `--skip-slave-start`

Ta opcja powoduje, że serwer nie będzie automatycznie uruchamiał wątków serwera podległego. Wspomniane wątki będą musiały być uruchomione ręcznie za pomocą zapytania `START SLAVE`.

■ `--sporadic-binlog-dump-fail (boolean)`

Ta opcja jest używana przez zestaw testowy MySQL w celu przetestowania replikacji.

F.12.3. Zmienne dla mysqld

W celu wyświetlenia pełnego komunikatu pomocy zawierającego wartości zmiennych systemowych domyślnie używane przez `mysqld` należy wydać poniższe polecenie:

```
% mysqld --verbose --help
```

Aby wyświetlić wartości zmiennych systemowych aktualnie używanych przez `mysqld`, należy wydać poniższe polecenie:

```
% mysqladmin variables
```

Istnieje możliwość sprawdzenia bieżących wartości zmiennych systemowych za pomocą zapytania `SHOW VARIABLES` lub przez spojrzenie do tabel `GLOBAL_VARIABLES` i `SESSION_VARIABLES` w bazie danych `INFORMATION_SCHEMA`. Poszczególne zmienne systemowe zostały omówione w dodatku D, zatytułowanym „Przewodnik po zmiennych systemowych, stanu i użytkownika”. Wartości zmiennych systemowych mogą być ustawione także podczas uruchamiania serwera; patrz informacje przedstawione w podpunkcie F.2.1.2, zatytułowanym „Ustawianie zmiennych programu”. Ponadto, wiele zmiennych systemowych można modyfikować dynamicznie; patrz punkt 12.3.1, zatytułowany „Sprawdzanie i ustawienie wartości zmiennych systemowych”. Omówienie zapytania `SET` znajdziesz w dodatku E, zatytułowanym „Przewodnik po składni SQL”.

F.13. mysqld_multi

Skrypt `mysqld_multi` ułatwia uruchamianie wielu serwerów `mysqld` w pojedynczym komputerze. Pozwala na uruchamianie i zatrzymywanie serwerów oraz ustalanie, czy są uruchomione:

```
mysqld_multi [options] command server_list
```

Poleceniem może być `start`, `stop` lub `report`. Począwszy od MySQL 5.6.3, dozwolone jest stosowanie również polecenia `reload` (zatrzymanie i ponowne uruchomienie serwera). Argument `server_list` wskazuje serwery, których dotyczą polecenia. Więcej informacji na temat użycia `mysqld_multi` znajdziesz w punkcie 12.9.4, zatytułowanym „Użycie skryptu `mysqld_multi` do zarządzania serwerem”.

F.13.1. Opcje standardowe obsługiwane przez mysqld_multi

```
--help      --silent    --verbose
--password  --user      --version
```

Ogólny opis powyższych opcji znajdziesz w punkcie F.2.1, zatytułowanym „Opcje standardowe programu MySQL”.

Skrypt `mysqld_multi` przekazuje narzędziu `mysqladmin` wartości opcji `--user` i `--password`, gdy musi zatrzymać serwery lub ustalić, czy są one uruchomione.

W przeciwieństwie do większości programów MySQL, w przypadku opcji `--password` podanie hasła jest wymagane.

F.13.2. Opcje charakterystyczne dla `mysqld_multi`

- `--example`
Wyświetlenie przykładowego pliku opcji, demonstrującego grupy pliku opcji odpowiednie do użycia wraz ze skryptem `mysqld_multi`.
- `--log=file_name`
Nazwa pliku dziennika, w którym skrypt `mysqld_multi` powinien rejestrować swoje działania. Jeżeli podany plik już istnieje, dane wyjściowe będą do niego dołączane. Domyślnie to plik o nazwie *mysqld_multi.log* umieszczony w katalogu danych MySQL. W celu wyłączenia rejestrowania należy użyć opcji `--no-log`.
- `--mysqladmin=file_name`
Ścieżka dostępu do pliku binarnego `mysqladmin` używanego przez skrypt. Ta opcja może być użyteczna, jeśli skrypt `mysqld_multi` samodzielnie nie znajduje pliku wykonywalnego `mysqladmin` lub gdy chcesz użyć jego konkretnej wersji.
- `--mysqld=file_name`
Ścieżka dostępu do pliku binarnego `mysqld` używanego przez skrypt. Ta opcja może być użyteczna, jeśli skrypt `mysqld_multi` samodzielnie nie znajduje pliku wykonywalnego `mysqld` lub gdy chcesz użyć jego konkretnej wersji. Dozwolone jest podanie ścieżki dostępu do pliku `mysqld` lub `mysqld_safe`.
- `--no-log`
Opcja powoduje wyświetlenie danych wyjściowych dziennika zdarzeń zamiast ich zapisu w dzienniku. Aby wyświetlić wspomniane dane na ekranie, trzeba użyć omawianej opcji, ponieważ domyślnie zdarzenia są zapisywane w dzienniku zdarzeń.
- `--tcp-ip`
Domyślnie skrypt `mysqld_multi` próbuje nawiązać połączenie z serwerem za pomocą pliku gniazda systemu UNIX. Ta opcja powoduje, że próba połączenia będzie wykorzystywała protokół TCP/IP. To może być użyteczne rozwiązanie, jeśli serwer działa, ale jego plik gniazda został usunięty i serwer jest dostępny jedynie przez protokół TCP/IP.

F.14. `mysqld_safe`

Skrypt `mysqld_safe` uruchamia serwer `mysqld` i monitoruje go:

```
mysqld_safe [options]
```

Jeżeli serwer zakończy działanie, skrypt `mysqld_safe` ponownie go uruchomi. Ten skrypt powłoki jest dostępny w systemach UNIX.

F.14.1. Opcje standardowe obsługiwane przez `mysqld_safe`

`--help` `--plugin-dir`

Ogólny opis powyższych opcji znajdziesz w punkcie F.2.1, zatytułowanym „Opcje standardowe programu MySQL”.

F.14.2. Opcje charakterystyczne dla `mysqld_safe`

Jeżeli w powłoce podasz opcje nieobsługiwane przez skrypt `mysqld_safe`, zostaną one przekazane serwerowi `mysqld`.

Skrypt `mysqld_safe` obsługuje następujące opcje:

- `--basedir=dir_name`
Ścieżka dostępu do katalogu bazowego MySQL.
- `--core-file-size=n`
Ograniczenie wielkości plików core do *n* bajtów, jeśli serwer ulegnie awarii.
- `--datadir=dir_name`
Ścieżka dostępu do katalogu danych MySQL.
- `--ledir=dir_name`
Katalog *libexec*, w którym należy szukać serwera.
- `--log-error[=file_name]`
Plik używany jako dziennik błędów. Ta opcja jest interpretowana w taki sam sposób, jak `mysqld` interpretuje wartość zmiennej systemowej `log_error`.
- `--malloc-lib[=lib_name]`
Biblioteka, której serwer `mysqld` powinien używać dla swojej implementacji `malloc()` zamiast biblioteki systemowej. Ta opcja modyfikuje zmienną środowiskową `LD_PRELOAD` w celu wpłynięcia na dynamiczne wiązanie dla `mysqld`. W przypadku systemu Linux wartością `lib_name` może być `tcmalloc`, co powoduje użycie implementacji biblioteki dostarczanej wraz z dystrybucją MySQL. Jeżeli wartością omawianej opcji jest ścieżka dostępu wskazująca katalog, wtedy skrypt `mysqld_safe` modyfikuje `LD_PRELOAD` przez umieszczenie wymienionego katalogu na początku jej wartości.
- `--mysqld=file_name`
Ścieżka dostępu do programu `mysqld`.
- `--mysqld-version=suffix`
Wartością tej opcji jest ciąg tekstowy przyrostka. Jeżeli opcja zostanie podana, wskazany przyrostek zostanie dodany do nazwy bazowej `mysqld` wraz z myślnikiem między nimi. W ten sposób powstanie nazwa serwera, który powinien być uruchomiony przez skrypt `mysqld`.

- **--nice=*N***
Opcja oznacza użycie programu nice do ustawienia priorytetu serwera jako wartości *N*.
- **--open-files-limit=*n***
Opcja wskazuje liczbę deskryptorów plików, które powinny być zarezerwowane przez mysqld.
- **--pid-file=*file_name***
Opcja określa nazwę pliku PID procesu mysqld.
- **--port=*port_num***
Opcja wskazuje numer portu, na którym serwer powinien nasłuchiwać połączeń TCP/IP.
- **--skip-kill-mysqld**
Nie próbuj zamknąć jakichkolwiek działających procesów mysqld, zanim nie uruchomisz nowego. Ta opcja może być użyteczna w przypadku uruchamiania wielu egzemplarzy tego samego programu binarnego mysqld. Omawiana opcja jest używana jedynie w systemie Linux.
- **--skip-syslog**
Opcja określa, że dane wyjściowe błędów nie powinny być kierowane do syslog; zamiast tego ma być użyty plik dziennika zdarzeń. Domyślnie używany jest plik dziennika zdarzeń.
- **--socket=*file_name***
Ścieżka dostępu do pliku gniazda systemu UNIX.
- **--syslog**
Opcja wskazuje, że dane wyjściowe błędów mają być kierowane do syslog w systemach posiadających program logger.
- **--syslog-tag=*tag***
Kiedy dane wyjściowe błędów są przekazywane do syslog, komunikaty ze skryptów mysqld_safe i mysqld są oznaczane prefiksem w postaci nazwy programu. Opcja --syslog-tag modyfikuje prefiks, aby miał postać odpowiednio mysqld_safe-tag i mysqld-tag.
- **--timezone=*tz_name***
Opcja powoduje ustawienie strefy czasowej serwera jako *tz_name*. To może być użyteczne, jeśli serwer nie jest w stanie automatycznie określić systemowej strefy czasowej.
- **--user=*user_name*, --user=*uid***
Ta opcja wskazuje nazwę lub identyfikator użytkownika konta systemu UNIX, w ramach którego został uruchomiony serwer.

F.15. mysqldump

Program `mysqldump` pozwala na zapisanie w plikach tekstowych zawartości tabel bazy danych. Wspomniane pliki mogą być wykorzystane do wielu celów, na przykład jako kopia zapasowa bazy danych, w celu przeniesienia bazy danych do innego serwera lub przygotowania testowej bazy danych na podstawie zawartości istniejącej.

Domyślnie dane wyjściowe każdej tabeli zapisanej w pliku zawierają zapytanie `CREATE TABLE` tworzące tę tabelę oraz zestaw zapytań `INSERT` wstawiających zawartość tabeli. Jeżeli użyta zostanie opcja `--tab`, wtedy zawartość tabeli zostanie zapisana w pliku danych jako wartości rozdzielone tabulatorami, po jednym rekordzie w wierszu, a samo zapytanie SQL tworzące tabelę znajdzie się w oddzielnym pliku.

Program `mysqldump` może być uruchomiony w jednym z trzech trybów:

```
mysqldump [options] db_name [tbl_name] ...
mysqldump [options] --databases db_name ...
mysqldump [options] --all-databases
```

W pierwszym trybie program `mysqldump` zapisuje w pliku zawartość wskazanych tabel w podanej bazie danych. Jeżeli nie zostanie podana żadna tabela, wtedy program `mysqldump` zapisuje w pliku zawartość wszystkich tabel w bazie danych. W drugim trybie wszystkie argumenty są traktowane jako nazwy baz danych, a program `mysqldump` zapisuje w pliku wszystkie tabele w wymienionych bazach danych. Z kolei w trzecim trybie program `mysqldump` zapisuje w pliku wszystkie tabele we wszystkich bazach danych. W przypadku użycia opcji `--databases` lub `--all-databases`, dane wyjściowe będą zawierały zapytania `CREATE DATABASE IF NOT EXISTS` i `USE` przed zapytaniami dotyczącymi poszczególnych tabel we wszystkich bazach danych.

Jeden z najczęściej stosowanych sposobów użycia programu `mysqldump` przedstawia się następująco:

```
% mysqldump db_name > backup_file
```

Aby utworzony wcześniej plik kopii zapasowej z powrotem wczytać do MySQL, należy użyć klienta `mysql`, a nie narzędzia `mysqlimport`:

```
% mysql db_name < backup_file
```

Program `mysqldump` ignoruje zawartość bazy danych `INFORMATION_SCHEMA` i nie zapisuje jej w pliku, o ile jej nazwa nie zostanie wyraźnie podana w wierszu poleceń. W takim przypadku trzeba również użyć opcji `--skip-lock-tables`.

F.15.1. Opcje standardowe obsługiwane przez mysqldump

<code>--bind-address</code>	<code>--default-character-set</code>	<code>--protocol</code>
<code>--character-sets-dir</code>	<code>--help</code>	<code>--shared-memory-base-name</code>
<code>--compress</code>	<code>--host</code>	<code>--socket</code>
<code>--debug</code>	<code>--password</code>	<code>--user</code>
<code>--debug-check</code>	<code>--pipe</code>	<code>--verbose</code>


```
--debug-info          --plugin-dir      --version
--default-auth        --port
```

Ogólny opis powyższych opcji znajdziesz w punkcie F.2.1, zatytułowanym „Opcje standardowe programu MySQL”. Program `mysqlcheck` obsługuje także standardowe opcje SSL wymienione w podpunkcie F.2.1.1, zatytułowanym „Opcje standardowe SSL”.

F.15.2. Opcje charakterystyczne dla mysqldump

Wymienione poniżej opcje kontrolują sposób działania programu `mysqldump`. W punkcie F.15.3, zatytułowanym „Oferowane przez mysqldump opcje formatu danych”, omówiono używane w połączeniu z `--tab` opcje służące do wskazania formatu plików danych.

- `--add-drop-database` (*boolean*)
Opcja powoduje dodanie zapytania `DROP DATABASE IF EXISTS` przed każdym zapytaniem `CREATE DATABASE`.
- `--add-drop-table` (*boolean*)
Opcja powoduje dodanie zapytania `DROP TABLE IF EXISTS` przed każdym zapytaniem `CREATE TABLE`.
- `--add-drop-trigger` (*boolean*)
Opcja powoduje dodanie zapytania `DROP TRIGGER IF EXISTS` przed każdym zapytaniem `CREATE TRIGGER`. Ta opcja została wprowadzona w MySQL 5.6.0.
- `--add-locks` (*boolean*)
Ta opcja powoduje dodanie zapytań `LOCK TABLE` i `UNLOCK TABLE` wokół zapytań `INSERT` dla wszystkich tabel.
- `--all-databases, -A` (*boolean*)
Ta opcja powoduje zapisanie w pliku zawartości wszystkich tabel wszystkich baz danych. Omawiana opcja powoduje również umieszczenie zapytań `CREATE DATABASE IF NOT EXISTS` i `USE` dla każdej bazy danych.
- `--all-tablespaces, -Y` (*boolean*)
Ta opcja powoduje zapis w pliku zawartości wszystkich przestrzeni tabel. Jest dostępna jedynie dla klastra MySQL.
- `--allow-keywords` (*boolean*)
Opcja pozwala na stosowanie nazw kolumn, które są słowami zarezerwowanymi.
- `--apply-slave-statements` (*boolean*)
Ta opcja jest stosowana w połączeniu z `--dump-slave`. Powoduje, że dane wyjściowe zawierają zapytania `STOP SLAVE` przed `CHANGE MASTER` oraz zapytanie `START SLAVE` na końcu. Omawiana opcja została wprowadzona w MySQL 5.5.3.

■ `--comments, -i` (*boolean*)

Ta opcja powoduje, że dane wyjściowe będą zawierały komentarze z informacjami dodatkowymi, takimi jak wersja programu `mysqldump`, tabele, których dotyczy zapytania `INSERT`, itd. Omawiana opcja jest włączona domyślnie. Aby ją wyłączyć, należy użyć `--skip-comments`.

■ `--compact` (*boolean*)

Opcja powoduje wygenerowanie zwięźlejszych danych wyjściowych pozbawionych komentarzy, między innymi definiowanych przez zmienne systemowe komentarzy o wersji. Omawiana opcja powoduje również włączenie opcji `--skip-add-drop-table`, `--skip-set-charset`, `--skip-disable-keys` i `--skip-add-locks`.

■ `--compatible=mode`

Opcja powoduje, że program `mysqldump` modyfikuje dane wyjściowe w taki sposób, aby zachowywały zgodność ze standardem SQL, innymi serwerami baz danych lub starszymi wersjami serwera MySQL. Wartość *mode* wskazuje tryb zgodności. Wartość omawianej opcji może składać się z jednej lub wielu rozdzielonych przecinkami wartości spośród wymienionych w tabeli F.11.

Tabela F.11. Wartości, które można przypisać opcji `--compatible`

Opcja	Opis
ANSI	Zgodność z ANSI.
DB2	Zgodność z DB2.
MAXDB	Zgodność z MAXDB.
MSSQL	Zgodność z MS SQL Server.
MYSQL323	Zgodność z MySQL 3.23.
MYSQL40	Zgodność z MySQL 4.0.
ORACLE	Zgodność z ORACLE.
POSTGRESQL	Zgodność z PostgreSQL.
NO_FIELD_OPTIONS	Wyłączenie charakterystycznych dla MySQL opcji dotyczących kolumn.
NO_KEY_OPTIONS	Wyłączenie charakterystycznych dla MySQL opcji dotyczących indeksów.
NO_TABLE_OPTIONS	Wyłączenie charakterystycznych dla MySQL opcji dotyczących tabel.

Ta opcja nie ma żadnego efektu w serwerach starszych niż MySQL 4.1.

■ `--complete-insert, -c` (*boolean*)

Opcja powoduje umieszczanie w zapytaniach `INSERT` nazwy wszystkich wstawianych kolumn.

■ **--create-options, -a (*boolean*)**

Opcja powoduje umieszczanie informacji dodatkowych w generowanych przez program mysqldump zapytaniach CREATE TABLE. Wspomniane informacje dotyczą między innymi silnika bazy danych, początkowej wartości AUTO_INCREMENT itd. Tego rodzaju informacje można podać jako opcje tabeli w zapytaniu CREATE TABLE. (Patrz także dodatek E, zatytułowany „Przewodnik po składni SQL”).

Omawiana opcja jest włączona domyślnie. Aby ją wyłączyć, należy użyć --skip-create-options.

■ **--databases, -B (*boolean*)**

Ta opcja powoduje zapisanie interpretacji wszystkich argumentów jako nazw danych i zapis w pliku zawartości wszystkich tabel wszystkich baz danych.

Omawiana opcja powoduje również umieszczenie zapytań CREATE DATABASE IF NOT EXISTS i USE dla każdej bazy danych.

■ **--delayed-insert (*boolean*)**

Ta opcja powoduje zapis w pliku zapytań INSERT DELAYED zamiast INSERT. Jeżeli plik kopii zapasowej zawierającej tabelę MyISAM wczytujesz do innej bazy danych i chcesz zminimalizować wpływ operacji na inne zapytania, które w tej chwili mogą być wykonywane w bazie danych, wtedy użycie opcji --delayed-insert pomoże w osiągnięciu zamierzonego celu.

■ **--delete-master-logs**

Opcja powoduje usunięcie w serwerze plików binarnego dziennika zdarzeń i utworzenie nowych za pomocą zapytania FLUSH MASTER wykonanego po wygenerowaniu kopii zapasowej. Nie używaj tej opcji, jeśli nie jesteś pewien, że chcesz usunąć istniejące pliki binarnego dziennika zdarzeń. Omawiana opcja powoduje włączenie opcji --master-data.

■ **--disable-keys, -K (*boolean*)**

Opcja powoduje dodanie zapytań ALTER TABLE ... DISABLE KEYS i ALTER TABLE ... ENABLE KEYS do danych wyjściowych, aby uniemożliwić w ten sposób uaktualnienia nieunikalnych indeksów podczas przetwarzania zapytań INSERT. To powoduje przyspieszenie operacji tworzenia indeksu dla każdej tabeli MyISAM, ponieważ wspomniana operacja odbywa się jednorazowo po wczytaniu całej tabeli.

■ **--dump-date (*boolean*)**

Opcja powoduje wstawienie na końcu pliku komentarza zawierającego datę utworzenia danych wyjściowych.

■ **--dump-slave[=*n*]**

Ta opcja działa jak --master-data, ale jest używana do utworzenia kopii zapasowej serwera podległego replikacji i generuje w danych wyjściowych zapytanie CHANGE MASTER wskazujące koordynaty binarnego dziennika zdarzeń w serwerze głównym replikacji, a nie w serwerze podległym. W punkcie

poświęconym opcji `--master-data` znajdziesz omówienie użycia argumentu opcji. Omawiana opcja została wprowadzona w MySQL 5.5.3.

■ `--events, -E` (*boolean*)

Opcja powoduje umieszczenie w danych wyjściowych zdarzeń zdefiniowanych w harmonogramie.

■ `--extended-insert, -e` (*boolean*)

Ta opcja powoduje tworzenie zapytań `INSERT` wstawiających wiele rekordów. To pozwala na znacznie efektywniejsze wstawianie danych niż za pomocą zapytań `INSERT` wstawiających tylko pojedyncze rekordy.

■ `--flush-logs, -F` (*boolean*)

Ta opcja powoduje opróżnianie plików dzienników zdarzeń serwera przed rozpoczęciem tworzenia kopii zapasowej. Domyślnie dzienniki zdarzeń są opróżniane dla każdej bazy danych w celu utworzenia punktu kontrolnego. To znacznie ułatwia przeprowadzenie operacji przywrócenia danych, ponieważ wiadomo, że pliki binarnego dziennika zdarzeń utworzone po punkcie kontrolnym powstały po wykonaniu kopii zapasowej bazy danych. W połączeniu z opcjami `--lock-all-tables` lub `--master-data` pliki dzienników zdarzeń są opróżniane jedynie po nałożeniu blokad na tabele. Omawiana opcja wymaga uprawnienia `RELOAD`.

■ `--flush-privileges` (*boolean*)

Jeżeli w kopii zapasowej znajduje się również zawartość bazy danych `mysql`, ta opcja powoduje umieszczenie zapytania `FLUSH PRIVILEGES` w danych wyjściowych po wspomnianej bazie danych.

■ `--force, -f` (*boolean*)

Ta opcja powoduje kontynuację działania programu, nawet pomimo wystąpienia błędów.

■ `--hex-blob` (*boolean*)

Opcja powoduje, że kolumny typu `BINARY`, `VARBINARY` i `BLOB` będą zapisane w postaci wartości szesnastkowych. Na przykład, użycie omawianej opcji powoduje zapisanie przez program `mysql dump` wartości MySQL jako `0x4D7953514C`.

■ `--ignore-table=db_name.tbl_name`

Opcja powoduje pominięcie danych należących do wskazanej tabeli lub widoku. W celu zignorowania wielu tabel tej opcji należy użyć wielokrotnie, za każdym razem podając nazwę tylko jednej tabeli.

■ `--include-master-host-port` (*boolean*)

Dla zapytania `CHANGE MASTER` umieszczanego w danych wyjściowych wygenerowanych przez opcję `--dump-slave` omawiana opcja powoduje dołączenie `MASTER_HOST` i `MASTER_PORT`, pozwalających na podanie nazwy komputera i numeru portu serwera głównego replikacji. Ta opcja została wprowadzona w MySQL 5.5.3.

- `--insert-ignore` (*boolean*)

Ta opcja powoduje umieszczenie zapytań `INSERT IGNORE` zamiast `INSERT`.

- `--lock-all-tables, -x` (*boolean*)

Opcja powoduje wykonanie zapytania `FLUSH TABLES WITH READ LOCK` do nałożenia blokad na wszystkie tabele we wszystkich bazach danych. Ta opcja powoduje wyłączenie `--single-transaction` i `--lock-tables`.

- `--lock-tables, -l` (*boolean*)

Dla każdej bazy danych umieszczonej w danych wyjściowych powoduje wykonanie zapytania `LOCK TABLES ... READ LOCAL` w celu nałożenia blokad na wszystkie tabele przed ich zapisaniem w pliku. Omawiana opcja jest dobra dla tabel MyISAM, ponieważ blokada `READ LOCK` pozwala na współbieżne operacje wstawiania danych podczas tworzenia kopii zapasowej. Z kolei dla tabel InnoDB preferowane jest użycie opcji `--single-transaction`.

- `--log-error=filename`

Zapisanie komunikatów ostrzeżeń i błędów na końcu wskazanego pliku.

- `--master-data[=value]`

Ta opcja ułatwia utworzenie kopii zapasowej, którą można wykorzystać do konfiguracji serwera podległego replikacji. Dzięki tej opcji narzędzie `mysqldump` wykonuje zapytanie `SHOW MASTER STATUS` w celu pobrania informacji o aktualnie używanym pliku binarnego dziennika zdarzeń oraz położeniu w tym pliku. Otrzymane dane są następnie używane do przygotowania i wykonania zapytania `CHANGE MASTER`, którego dane wyjściowe zawierają ten sam plik i położenie. Dlatego też po wczytaniu pliku kopii zapasowej w serwerze podległym następuje jego synchronizacja z właściwymi koordynatami replikacji, co pozwala na wznowienie replikacji od miejsca, w którym utworzono kopię zapasową. Ta opcja nie ma żadnego efektu, jeśli w serwerze nie włączono rejestracji zdarzeń w binarnym dzienniku.

Domyślnie zapytanie `CHANGE MASTER` jest zapisywane w postaci niekomentowanej.

Opcja `--master-data` pobiera opcjonalną wartość wyraźnie kontrolującą stosowanie komentarza dla zapytania. Wartość 1 oznacza wygenerowanie niekomentowanego zapytania, natomiast wartość 2 generuje zapytanie w komentarzu.

Opcja `--master-data` wymaga uprawnień `RELOAD`. Ta opcja automatycznie włącza `--lock-all-tables`, jeśli nie podano `--single-transaction`.

- `--no-autocommit` (*boolean*)

Opcja powoduje, że dla każdej tabeli zapytania `INSERT` są umieszczane w transakcji. Wygenerowane dane wyjściowe mogą być wczytane znacznie efektywniej niż przy wykonywaniu każdego zapytania w trybie automatycznego zatwierdzania.

- `--no-create-db, -n` (*boolean*)

Opcja powoduje, że zapytania `CREATE DATABASE` nie będą umieszczane w pliku kopii zapasowej. (Normalnie wspomniane zapytania są dodawane automatycznie po użyciu opcji `--databases` lub `--all-databases`).

- `--no-create-info, -t` (*boolean*)

Opcja powoduje, że zapytania `CREATE TABLE` nie będą umieszczane w pliku kopii zapasowej. To jest użyteczne w celu umieszczenia w kopii zapasowej jedynie danych tabel.

- `--no-data, -d` (*boolean*)

Opcja powoduje, że w pliku kopii zapasowej nie będą umieszczone dane tabel. Ta jest użyteczne w celu umieszczenia w kopii zapasowej jedynie zapytań `CREATE TABLE`.

- `--no-set-names, -N`

To jest synonim opcji `--skip-set-charset`.

- `--no-tablespaces, -y` (*boolean*)

Opcja powoduje, że przestrzeń tabel nie zostanie umieszczona w kopii zapasowej. Ta opcja ma zastosowanie jedynie w przypadku klastra MySQL.

- `--opt`

Opcja powoduje optymalizację szybkości i wygenerowanie pliku, który będzie optymalny podczas jego ponownego wczytywania. Ta opcja powoduje włączenie `--add-drop-table`, `--add-locks`, `--create-options`, `--disable-keys`, `--extended-insert`, `--lock-tables`, `--quick` i `--set-charset`. Omawiana opcja jest włączona domyślnie; aby ją wyłączyć, należy użyć `--skip-opt`.

- `--order-by-primary` (*boolean*)

Rekordy tabeli zostaną zapisane w kolejności klucza podstawowego lub pierwszego unikalnego indeksu, o ile taki istnieje. W ten sposób kopia zapasowa będzie zawierała posortowane dane każdej tabeli, ale kosztem spadku wydajności operacji.

- `--quick, -q` (*boolean*)

Domyślnie narzędzie `mysql dump` odczytuje całą zawartość tabeli, umieszcza ją w pamięci, a następnie zapisuje w pliku. Ta opcja powoduje, że każdy rekord zostanie zapisany w pliku natychmiast po jego odczytaniu z serwera. Dzięki temu operacja wymaga mniejszej ilości pamięci. Jeśli jednak zdecydujesz się na użycie tej opcji, nie powinieneś wstrzymywać działania `mysql dump`, ponieważ to zmusi serwer do czekania na `mysql dump`, co z kolei zakłóci współpracę serwera z innymi klientami.

- `--quote-names, -Q` (*boolean*)

Opcja powoduje cytowanie nazw tabel i kolumn przez ich ujęcie w odwrotne apostrofy (```). To użyteczne, jeśli nazwy są słowami zarezerwowanymi lub zawierają znaki specjalne. Opcja `--quote-names` jest włączona domyślnie; aby ją wyłączyć, należy użyć `--skip-quote-names`.

- **--replace**
Opcja powoduje wygenerowanie zapytań REPLACE zamiast INSERT.
- **--result-file=***file_name*, **-r** *file_name*
Opcja powoduje zapis danych wyjściowych do wskazanego pliku. Omawiana opcja jest przeznaczona dla systemu Windows, w którym uniemożliwia przeprowadzenie konwersji znaków wysuw wiersza na parę powrót na początek wiersza/wysuw wiersza.
- **--routines, -R** (*boolean*)
Opcja powoduje umieszczenie w danych wyjściowych procedur i funkcji składowanych.
- **--set-charset** (*boolean*)
Opcja powoduje umieszczenie w danych wyjściowych zapytań SET NAMES *charset*, gdzie *charset* to domyślnie utf8. Kodowanie znaków można zmienić za pomocą opcji **--default-character-set**. Opcja **--set-charset** jest włączona domyślnie; aby ją wyłączyć, należy użyć **--skip-set-charset**.
- **--single-transaction** (*boolean*)
Ta opcja umożliwia utworzenie spójnej kopii zapasowej tabel InnoDB. Idea polega na zapisaniu zawartości wszystkich tabel w ramach jednej transakcji. Narzędzie mysqldump używa poziomu izolacji REPEATABLE READ do wygenerowania spójnej kopii zapasowej bez jednoczesnego blokowania dostępu innym klientom. (W przypadku tabel nietransakcyjnych w trakcie operacji tworzenia kopii zapasowej mogą być wprowadzane zmiany). Omawiana opcja powoduje wyłączenie **--lock-all-tables**.
- **--skip-opt**
Działanie tej opcji ma odwrotny efekt do działania opcji **--opt**, która jest domyślnie włączona.
- **--tab=***dump_dir*, **-T** *dump_dir*
Opcja powoduje zapis przez narzędzie mysqldump po dwa pliki dla każdej tabeli przy użyciu wartości *dump_dir* jako miejsca położenia plików. Wskazany katalog musi już istnieć. Dla każdej tabeli (*tbl_name*) będą utworzone dwa pliki: *dump_dir/tbl_name.txt*, zawierający dane tabeli, oraz *dump_dir/tbl_name.sql*, zawierający zapytanie CREATE TABLE tworzące tę tabelę. Użycie omawianej opcji wymaga uprawnienia FILE.

Domyślnie pliki danych są zapisywane jako wiersze zakończone znakami nowego wiersza i zawierające rozdzielone tabulatorami wartości kolumn. Ten format można zmienić za pomocą opcji omówionych w podrozdziale F.15.3, zatytułowanym „Oferowane przez mysqldump opcje formatu danych”.
Efekt użycia opcji **--tab** może być niezrozumiały, o ile nie będziesz dokładnie wiedział, w jaki sposób omawiana opcja działa:

- ◆ Pewne pliki są zapisywane w komputerze serwera, z kolei inne w komputerze klienta. Wartość `dump_dir` wskazuje katalog w komputerze serwera przeznaczony dla plików `*.txt` oraz katalog w komputerze klienta przeznaczony dla plików `*.sql`. Jeżeli to będą dwa różne komputery, wtedy dane wyjściowe powstaną w różnych komputerach. Aby uniknąć niepewności w zakresie miejsca utworzenia plików danych wyjściowych, najlepszym rozwiązaniem jest uruchamianie narzędzia `mysql dump` w komputerze serwera.
- ◆ Właścicielem plików `*.txt` jest użytkownik, którego konto wykorzystano do uruchomienia serwera, natomiast właścicielem plików `*.sql` jest klient. To jest konsekwencja faktu, że serwer samodzielnie zapisuje pliki `*.txt`, podczas gdy zapytania `CREATE TABLE` są wysyłane przez serwer do narzędzia `mysql dump`, które następnie zapisuje je w plikach `*.sql`.
- `--tables`
Opcja nadpisuje `--databases` i powoduje, że znajdujące się po niej argumenty będą interpretowane jako nazwy tabel.
- `--triggers` (*boolean*)
Opcja powoduje umieszczenie wyzwalaczy w danych wyjściowych. Wyzwalacze są domyślnie umieszczane w danych wyjściowych. Jeśli chcesz je pominąć, to należy użyć opcji `--skip-triggers`.
- `--tz-utc` (*boolean*)
Opcja powoduje ustawienie strefy czasowej UTC po nawiązaniu połączenia z serwerem i umieszczenie zapytania `SET TIME_ZONE='+00:00'` w danych wyjściowych. Efektem jest wyłączenie konwersji z lokalnej strefy czasowej oraz na lokalną strefę czasową podczas tworzenia kopii zapasowej i wczytywania danych. Wartości `TIMESTAMP` nie ulegają zmianie, jeśli dane są wczytywane w innej strefie czasowej niż stosowana w trakcie tworzenia kopii zapasowej. Omawiana opcja jest domyślnie włączona; aby ją wyłączyć, należy użyć `--skip-tz-utc`.
- `--where=where_expr`, `-w where_expr`
Opcja powoduje umieszczenie w kopii zapasowej tylko tych rekordów, które zostały wybrane przez warunek `WHERE` w postaci `where_expr`. Warunek powinien być cytowany, aby wiersz poleceń nie potraktował go jako wiele argumentów wiersza poleceń.
- `--xml`, `-X`
Ta opcja powoduje wygenerowanie danych wyjściowych w formacie XML zamiast jako zestawu zapytań SQL.

F.15.3. Oferowane przez mysqldump opcje formatu danych

W przypadku użycia opcji `--tab` lub `-T` do wygenerowania oddzielnego pliku danych dla każdej tabeli dodatkowo można zastosować jeszcze kilka innych opcji. Może wystąpić konieczność cytowania wartości opcji. Wymienione tutaj opcje są analogiczne do opcji formatu danych dla zapytania `LOAD DATA`. Zapoznaj się z podpunktem poświęconym zapytaniu `LOAD DATA` w dodatku E, zatytułowanym „Przewodnik po składni SQL”.

- `--fields-enclosed-by=char`

Opcja oznacza, że wartości kolumn powinny być ujęte we wskazane znaki, najczęściej to znaki cytowania. Domyślnie nie ma konieczności ujmowania wartości kolumn w jakiegokolwiek znaki. Ta opcja wyklucza użycie `--fields-optionally-enclosed-by`.

- `--fields-escaped-by=char`

Opcja pozwala na zdefiniowanie znaku sterującego używanego do obsługi znaków specjalnych. Domyślnie nie jest zdefiniowany żaden znak specjalny.

- `--fields-optionally-enclosed-by=char`

Opcja wskazuje, że wartości kolumn powinny być ujęte we wskazane znaki, najczęściej to znaki cytowania. Wspomniane znaki są używane dla kolumn o wartościach innych niż liczbowe. Domyślnie nie ma konieczności ujmowania wartości kolumn w jakiegokolwiek znaki. Ta opcja wyklucza użycie `--fields-enclosed-by`.

- `--fields-terminated-by=str`

Opcja pozwala na wskazanie znaku lub znaków rozdzielających wartości kolumn w plikach danych. Domyślnie wartości są rozdzielone tabulatorami.

- `--lines-terminated-by=str`

Opcja pozwala na wskazanie znaku lub znaków umieszczanych na końcu wierszy danych wyjściowych. Domyślnie umieszczane są znaki nowego wiersza.

F.15.4. Zmienne dla mysqldump

Wymienione tutaj zmienne narzędzia `mysqldump` mogą być ustawione za pomocą poleceń podanych w podpunkcie F.2.1.2, zatytułowanym „Ustawianie zmiennych programu”.

- `max_allowed_packet`

Maksymalna wielkość bufora używanego podczas komunikacji między serwerem i klientem. Domyślnie to 24 MB, natomiast maksymalnie 1 GB.

- `net_buffer_length`

Początkowa wielkość bufora używanego podczas komunikacji między serwerem i klientem. Wielkość bufora może wzrosnąć aż do wyrażonej w bajtach wartości zdefiniowanej przez zmienną `max_allowed_packet`. Domyślnie to nieco mniej niż 1 MB.

F.16. mysqlimport

Narzędzie `mysqlimport` pozwala na wstawienie do bazy danych ogromnej ilości danych przez odczytanie zawartości tabel z plików tekstowych. Omawiane narzędzie to interfejs wiersza poleceń dla zapytania SQL `LOAD DATA` i zarazem efektywny sposób wstawienia rekordów do tabel.

```
mysqlimport [options] db_name file_name ...
```

Argument `db_name` określa bazę danych zawierającą tabele, do których mają być wstawione dane. Same tabele są ustalane na podstawie argumentów narzędzia. Dla każdej nazwy pliku następuje usunięcie rozszerzenia, poczynawszy od pierwszej kropki w nazwie, a tak powstała nazwa bazowa jest używana jako nazwa tabeli, w której powinny być wstawione dane. Na przykład, narzędzie `mysqlimport` umieści zawartość pliku `president.txt` w tabeli o nazwie `president`.

Narzędzie `mysqlimport` jest przeznaczone jedynie do odczytu pliku danych, *nie* zostało przystosowane do odczytu plików kopii zapasowej utworzonych przez `mysqldump`. Do drugiego z wymienionych celów służy klient `mysql`.

F.16.1. Opcje standardowe obsługiwane przez mysqlimport

<code>--bind-address</code>	<code>--default-character-set</code>	<code>--protocol</code>
<code>--character-sets-dir</code>	<code>--help</code>	<code>--shared-memory-base-name</code>
<code>--compress</code>	<code>--host</code>	<code>--silent</code>
<code>--debug</code>	<code>--password</code>	<code>--socket</code>
<code>--debug-check</code>	<code>--pipe</code>	<code>--user</code>
<code>--debug-info</code>	<code>--plugin-dir</code>	<code>--verbose</code>
<code>--default-auth</code>	<code>--port</code>	<code>--version</code>

Ogólny opis powyższych opcji znajdziesz w punkcie F.2.1, zatytułowanym „Opcje standardowe programu MySQL”. Program `mysqlcheck` obsługuje także standardowe opcje SSL wymienione w podpunkcie F.2.1.1, zatytułowanym „Opcje standardowe SSL”.

F.16.2. Opcje charakterystyczne dla mysqlimport

Wymienione poniżej opcje kontrolują sposób przetwarzania plików danych wejściowych przez program `mysqlimport`. W punkcie F.16.3, zatytułowanym „Oferowane przez `mysqlimport` opcje formatu danych”, omówiono opcje używane do wskazania formatu plików danych wejściowych.

■ `--columns=col_list, -C col_list`

Opcja pozwala na podanie listy kolumn tabeli, którym będą odpowiadały kolumny w pliku danych. Wartości znajdujące się w rekordach danych wejściowych narzędzie `mysqlimport` wczytuje do wskazanych kolumn, a pozostałym kolumnom przypisuje ich wartości domyślne. Argument `col_list` to jedna lub więcej rozdzielonych przecinkami nazw kolumn.

- **--delete, -d** (*boolean*)
Opcja powoduje opróżnienie każdej tabeli przed wstawieniem do niej jakichkolwiek danych.
- **--force, -f** (*boolean*)
Opcja powoduje kontynuowanie operacji wstawiania rekordów, nawet jeśli wystąpi błąd.
- **--ignore, -i**
Kiedy rekord danych wejściowych zawiera dla unikalnego klucza wartość już istniejącą w tabeli, dotychczasowy rekord zostanie zachowany, natomiast rekord danych wejściowych będzie odrzucony. Opcje `--ignore` i `--replace` wzajemnie się wykluczają.
- **--ignore-lines=*n***
Opcja powoduje zignorowanie pierwszych *n* wierszy pliku danych. Może być użyta do pominięcia na przykład początkowego rekordu zawierającego etykiety kolumn.
- **--local, -L** (*boolean*)
Domyślnie narzędzie `mysqlimport` pozwala serwerowi na odczyt pliku danych, co oznacza konieczność umieszczenia pliku w komputerze serwera oraz posiadania uprawnień `FILE`. Użycie opcji `--local` nakazuje narzędziu `mysqlimport` samodzielny odczyt pliku danych i przekazanie jego zawartości serwerowi. Takie rozwiązanie jest wolniejsze, ale działa w przypadku uruchomienia narzędzia `mysqlimport` w innym komputerze niż komputer z działającym serwerem MySQL, a także sprawdza się, jeśli nie masz uprawnień `FILE`. Omawiana opcja nie powoduje żadnego efektu, jeśli serwer został skonfigurowany w sposób uniemożliwiający użycie `LOAD DATA LOCAL`.
- **--lock-tables, -l** (*boolean*)
Nałożenie blokad na każdą tabelę przed wstawieniem w niej danych.
- **--low-priority** (*boolean*)
Opcja powoduje użycie modyfikatora `LOW_PRIORITY` dla wygenerowanych zapytań odpowiedzialnych za wstawianie danych do tabeli.
- **--replace, -r** (*boolean*)
Kiedy rekord danych wejściowych zawiera dla unikalnego klucza wartość już istniejącą w tabeli, dotychczasowy rekord zostanie zastąpiony nowym rekordem danych wejściowych. Opcje `--ignore` i `--replace` są wzajemnie wykluczające się.
- **--use-threads=*n***
Opcja powoduje użycie *n* wątków w celu równoległego wczytywania danych z wielu plików.

F.16.3. Oferowane przez `mysqlimport` opcje formatu danych

Domyślnie narzędzie `mysqlimport` przyjmuje założenie, że pliki danych zawierają wiersze kończące się znakami nowego wiersza, natomiast wartości danych są rozdzielone tabulatorami. Ten oczekiwany format można zmienić za pomocą wymienionych poniżej opcji. Może wystąpić konieczność cytowania wartości opcji. Wymienione tutaj opcje są analogiczne do opcji formatu danych dla zapytania `LOAD DATA`. Zapoznaj się z podpunktem poświęconym zapytaniu `LOAD DATA` w dodatku E, zatytułowanym „Przewodnik po składni SQL”.

- `--fields-enclosed-by=char`
Opcja oznacza, że wartości kolumn powinny być ujęte we wskazane znaki, najczęściej to znaki cytowania. Domyślnie nie ma konieczności ujmowania wartości kolumn w jakiegokolwiek znaki. Ta opcja wyklucza użycie `--fields-optionally-enclosed-by`.
- `--fields-escaped-by=char`
Opcja pozwala na zdefiniowanie znaku sterującego używanego do obsługi znaków specjalnych. Domyślnie nie jest zdefiniowany żaden znak specjalny.
- `--fields-optionally-enclosed-by=char`
Opcja wskazuje, że wartości kolumn powinny być ujęte we wskazane znaki, najczęściej to znaki cytowania. Ta opcja wyklucza użycie `--fields-enclosed-by`.
- `--fields-terminated-by=str`
Opcja pozwala na wskazanie znaku lub znaków rozdzielających wartości kolumn w plikach danych. Domyślnie wartości są rozdzielone tabulatorami.
- `--lines-terminated-by=str`
Opcja pozwala na wskazanie znaku lub znaków umieszczanych na końcu wierszy danych wyjściowych. Domyślnie umieszczane są znaki nowego wiersza.

F.17. `mysqlshow`

Narzędzie `mysqlshow` wyświetla bazy danych, tabele w bazach danych bądź też informacje o kolumnach lub indeksach znajdujących się w tabeli. Dostarcza interfejs wiersza poleceń dla zapytania SQL `SHOW`:

```
mysqlshow [options] [db_name [tbl_name [col_name]]]
```

Jeżeli nie zostanie podana żadna nazwa bazy danych, `mysqlshow` wyświetli wszystkie bazy danych w komputerze serwera. W przypadku podania nazwy bazy danych, ale bez tabel, wyświetlone będą wszystkie tabele znajdujące się we wskazanej bazie danych. Jeśli podane zostaną nazwy bazy danych i tabeli, ale bez kolumn, wówczas narzędzie wyświetli wszystkie kolumny w podanej tabeli. Po podaniu wszystkich nazw narzędzie `mysqlshow` wyświetli informacje o wskazanej kolumnie.

Ostatni argument może zawierać znaki wieloznaczne SQL % i _; efekt będzie taki sam jak przy użyciu operatora LIKE. Dane wyjściowe zostaną ograniczone do wartości dopasowanych do podanych znaków wieloznacznych. Jeżeli ostatni argument zawiera znaki wieloznaczne powłoki (* i ?), wtedy będą one traktowane jako znaki odpowiednio % i _.

F.17.1. Opcje standardowe obsługiwane przez mysqlshow

--bind-address	--default-character-set	--protocol
--character-sets-dir	--help	--shared-memory-base-name
--compress	--host	--socket
--debug	--password	--user
--debug-check	--pipe	--verbose
--debug-info	--plugin-dir	--version
--default-auth	--port	

Ogólny opis powyższych opcji znajdziesz w punkcie F.2.1, zatytułowanym „Opcje standardowe programu MySQL”. Program mysqlcheck obsługuje także standardowe opcje SSL wymienione w podpunkcie F.2.1.1, zatytułowanym „Opcje standardowe SSL”.

Opcja --verbose powoduje umieszczenie w danych wyjściowych dodatkowych kolumn (tabele w bazie danych, rekordy w tabeli itd.). Tę opcję można podać wielokrotnie.

F.17.2. Opcje charakterystyczne dla mysqlshow

- --count (*boolean*)

Opcja powoduje, że dane wyjściowe będą zawierały informacje o liczbie rekordów w tabeli. W pewnych silnikach bazy danych zliczanie rekordów może być wolne.

- --keys, -k (*boolean*)

Opcja powoduje, że oprócz informacji o kolumnach tabeli będą wyświetlone również informacje o indeksach tabeli. Ta opcja ma znaczenie jedynie w przypadku podania nazwy tabeli jako argumentu mysqlshow.

- --show-table-type, -t (*boolean*)

Opcja powoduje umieszczenie w danych wyjściowych kolumny wskazującej typ każdej tabeli (BASE TABLE lub VIEW), podobnie jak w przypadku wykonania zapytania SHOW FULL TABLES.

- --status, -i (*boolean*)

Opcja powoduje wyświetlenie tego samego rodzaju informacji o tabeli, jakie można uzyskać po wykonaniu zapytania SHOW TABLE STATUS.

F.18. perror

Narzędzie perror wyświetla komunikaty błędów dla danego kodu błędu:

```
perror [options] [err_code] ...
```

To narzędzie można wykorzystać do otrzymania konkretnych informacji na temat błędów zgłaszanych przez programy MySQL.

```
% perror 142
```

```
MySQL error: 142 = Unknown character set used
```

F.18.1. Opcje standardowe obsługiwane przez perror

```
--help      --silent    --verbose    --version
```

Ogólny opis powyższych opcji znajdziesz w punkcie F.2.1, zatytułowanym „Opcje standardowe programu MySQL”.

Opcja `--silent` powoduje wyświetlenie jedynie komunikatu błędu, bez jego kodu. Domyślnie jest stosowana opcja `--verbose`, która wyświetla zarówno komunikat błędu, jak i jego kod.

Opcje `--info` i `-I` są synonimami opcji `--help`.

Skorowidz

A

- ACID, 189
- administracja bazą danych, 591, 621
- adres IP, 763, 917
- algorytm wyświetlania, 412
- aliasy powłoki, 115
- alokacja uchwytu zapytania, 432
- analiza skryptów DBI, 447
- API, 359
 - C, 359, 360
 - Perl DBI, 360
 - PHP, 359, 362
- aplikacje sieciowe, 510
- architektura klient-serwer, 45
- argument undef, 473
- atrybut
 - ASCII, 254
 - AUTO_INCREMENT, 72, 214, 238, 272–277
 - BINARY, 254
 - NULL, 60, 264
 - RaiseError, 488
 - SIGNED, 237
 - UNICODE, 254
 - UNSIGNED, 237
 - ZEROFILL, 237
- atrybuty
 - połączenia, 450, 451
 - typu ciągu tekstowego, 251
 - typu daty, 262
 - typu liczbowego, 237

- automatyczna
 - konwersja typu, 337
 - naprawa, 783, 784, 807
- automatyczne
 - właściwości typu daty, 263
 - zatwierdzanie, 576

B

- baza danych, DB, 42, 323
 - INFORMATION_SCHEMA, 164, 166
 - sampdb, 523
- bezpieczeństwo, 320, 386, 524, 715–774, 911
 - serwera, 594
 - systemu plików, 594
 - transakcji, 160
- bezpieczne
 - połączenia, 770
 - środowisko wykonywania, 190
- biblioteka
 - info_tables.so, 652
 - libmysqlclient, 538
 - mysqlnd, 538
- bieżące ustawienia, 133
- binarne kopie zapasowe, 786, 816
- blok try/catch, 544
- blokada globalna, 782
- blokowanie tabel, 353, 779–782
- błąd Access denies, 398
- błędy, 268, 377, 453, 478, 561
- błędy składni, 53, 401
- BMP, Basic Multilingual Plane, 130

bufor
kluczy, 677
zapytań, 680

C

cechy skryptu, 446
Certificate Authority, CA, 771
ciąg tekstowy, 217, 416, 878–891
 binarny, 219, 221, 223
 niebinarny, 219, 223
CRC, Cyclic Redundancy Check, 874
cyfry znaczące, 235
cytowanie, 730
 ciągów tekstowych, 467
 danych, 559

D

dane binarne, 405, 418
data i godzina, 92, 892–907
DBD, Database Driver, 361
DBI, Database Interface, 361
DDL, Data Definition Language, 192
definicja
 FOREIGN KEY, 74, 198
 kolumny, 230
 kolumny member_id, 61
 PRIMARY KEY, 74
definiowanie
 haseł, 622
 uprawnień, 734
 widoków, 308
defragmentowanie tabel, 345
dezaktywacja indeksów, 351
dezinstalacja wtyczki, 653
długość
 identyfikatora, 127
 prefiksu, 330
 rekordów indeksu, 157
DNS, 730
dodawanie rekordów, 75
dokument HTML, 507
dokumentacja Perl DBI, 445

dołączanie
 do zmiennych, 474
 wyniku zapytania, 474
domyślne kodowanie znaków, 667
dopasowywanie wzorca, 95, 285, 532, 865
dostęp do
 CGI.pm, 514
 danych, 357, 358, 600
 metadanych, 482
 MySQL, 537, 538
 odczytu i zapisu, 780
 plików, 778
 pliku nagłówkowego, 546
 pliku opcji, 383
 tabel, 353, 778
dostrajanie
 serwera, 668, 670
 silnika bazy danych, 673
dystrybucja sampdb, 548, 819
działanie optymalizatora zapytań, 333, 334
dziennik
 błędów, 687, 688
 przekazywania, 692
 zapytań, 688, 689
 zdarzeń, 612, 686, 693, 701
 zdarzeń binarny, 690, 815
 zdarzeń serwera, 684

E

edycja
 rekordu, 500
 wiersza danych, 116
efektywne
 wczytywanie danych, 349
 wykonywanie zapytań, 344
efektywność indeksu, 324
elementy struktury my_option, 390

F

format
 rejestracji danych, 815
 RTF, 39, 489, 492

- formaty DSN, 450
 - formularze, 574, 588
 - fsp, 831
 - funkcja
 - AGAINST(), 207
 - ASCII(), 296
 - AVG(), 107
 - CAST(), 251
 - CHARSET(), 130
 - check_response(), 581
 - COLLATION(), 130
 - CONVERT(), 222, 297
 - COUNT(), 98
 - COUNT(*), 207
 - CURRENT_USER(), 321
 - DATE_ADD(), 94, 296
 - DATE_SUB(), 94
 - DAYOFYEAR(), 109
 - display_events(), 529
 - display_scores(), 530
 - display_table_contents(), 526
 - display_table_names(), 525
 - escape(), 517
 - get_one_option(), 424
 - htmlspecialchars(), 552
 - insert_rows(), 432, 435
 - LAST_INSERT_ID(), 279
 - LENGTH(), 222
 - li(), 526
 - load_defaults(), 383, 386–389
 - load_image(), 419
 - main(), 375
 - MIN(), 102
 - MONTH(), 109
 - my_print_help(), 394
 - mysql_close(), 376, 379
 - mysql_error(), 379
 - mysql_fetch_row(), 402
 - mysql_field_count(), 406
 - mysql_field_seek(), 413
 - mysql_free_result(), 402
 - mysql_init(), 375
 - mysql_library_end(), 376
 - mysql_library_init(), 376
 - mysql_more_results(), 427
 - mysql_real_connect(), 375
 - mysql_real_escape_string(), 417
 - mysql_sqlstate(), 379
 - mysql_ssl_set(), 425
 - mysql_stmt_init(), 432
 - mysql_store_result(), 402, 406, 409–411
 - mysql_use_result(), 409–411
 - NOW(), 151
 - password_field(), 585
 - print_dashed(), 414
 - print_error(), 380, 456
 - process_call_result(), 443
 - process_result_set(), 404, 412, 420, 444
 - RAND(), 151
 - referential integrity, 72
 - sampdb_connect(), 545, 547, 551
 - search_members(), 532, 533
 - select_rows(), 437, 438
 - TIMESTAMPDIFF(), 93
 - TO_DAYS(), 94
 - update_entry(), 589
 - funkcje, 869
 - agregujące, 177, 907
 - ciągu tekstowego, 878–91
 - daty i godziny, 892–907
 - liczbowe, 873
 - nakładania blokad, 915
 - podsumowań, 907
 - porównań, 870
 - przestrzenne, 920
 - różne, 920
 - rzutowania, 872
 - składowane, 311, 314
 - XML, 919
 - zapewnienia bezpieczeństwa, 911
 - związane z adresem IP, 917
- ## G
- GA, General Availability, 27
 - generowanie
 - danych wyjściowych, 514
 - dokumentów HTML, 518
 - catalogu, 488, 492
 - listy, 491

- generowanie
 - łącza, 530
 - podsumowania, 97
 - sekwencji, 278
 - strony HTML, 515
- globalizacja, 664
- globalne ustawienia, 125
- graficzny interfejs użytkownika, GUI, 45
- grupowanie, 221, 853

H

- harmonogram, 352
- harmonogram zdarzeń, 318
- hasła, 622, 627, 745, 747
- hermetyzacja kodu, 545
- historia poleceń, 114
- HTML, 515

I

- identyfikatory, 126, 127
- implementacje SQL, 361
- indeks
 - typu BTREE, 157, 332
 - typu FULLTEXT, 155, 205, 211, 534
 - typu HASH, 155, 331
 - typu PRIMARY KEY, 155
 - typu SPATIAL, 155
 - typu UNIQUE, 155
- indeksowanie, 324, 327
 - krótkich wartości, 329
 - prefiksów, 330
 - tabel, 153
- indeksy
 - klastrowanie, 330
 - syntetyczne, 346
 - unikalne, 154
 - w silnikach, 154
 - zwykłe, 154
- informacje
 - o bazach danych, 164, 166
 - o błędzie, 563
 - o kolumnie, 414
 - o strefie czasowej, 665
 - z wielu tabel, 104, 183

- inicjalizacja
 - katalogu danych, 825
 - tabel systemowych, 827
 - tabel uprawnień, 825
- InnoDB, 27
- instalacja
 - MySQL, 822, 823
 - PDO, 829
 - Perl DBI, 828
 - PHP, 829
- integralność odwołań, 197, 198
- interaktywne wykonywanie zapytań, 420
- interfejs
 - dostępu, 537
 - sieciowy, 703
 - wtyczek, 651
- izolacja transakcji, 194

J

- jednoczesne
 - operacje zapisu, 353
 - wykonywanie zapytań, 426
- język
 - C, 369–444
 - Perl, 39
 - Perl DBI, 445–536
 - PHP, 537–89
 - SQL, *Patrz* SQL
- języki interpretowane, 366

K

- katalog
 - danych, 593, 597, 614, 825
 - Ligi Historycznej, 488
 - phpapi, 537
 - ssl, 772
 - ushl, 553
- klauzula
 - CHANGE, 159
 - CHARSET, 131
 - CONSTRAINT, 198
 - DEFAULT, 254, 262, 832
 - DEFINER, 321

- DO, 320
- ENGINE, 604
- FOREIGN KEY, 74, 198
- FROM, 81, 104, 168
- GROUP BY, 99
- HAVING, 101
- IDENTIFIED BY, 746, 747
- IDENTIFIED WITH, 751
- LEFT JOIN, 108
- LIKE, 132, 163, 285, 532
- LIMIT, 89, 101, 186, 527
- MODIFY, 159
- ON, 104, 106, 170
- ON DELETE, 199
- ON UPDATE, 199
- ON UPDATE CASCADE, 201
- ORDER BY, 87, 100, 174, 185
- PRIMARY KEY, 61
- REFERENCES, 199
- REQUIRE, 741, 742
- RETURNS, 312
- SET, 77
- USING, 187
- USING BTREE, 157
- USING(), 170
- WHERE, 82, 308
- WHERE FALSE, 485
- WITH ROLLUP, 102
- klient
 - MySQL, 52, 382, 389, 394, 592
 - mysqladmin, 1154
 - z obsługą SSL, 421
- klucz zewnętrzny, 75, 197, 200
- kod
 - HTML, 515
 - XHTML, 516
- kodowanie
 - Unicode, 130, 132, 134
 - znaków, 130–132, 219, 222, 517
 - znaków specjalnych, 416, 417
- kolejność
 - dołączania plików, 374
 - operatorów, 287, 852
 - sortowania, 131, 667
- kolumna
 - AUTO_INCREMENT, 275–278
 - GROUP BY, 106
- kolumny
 - uprawnień, 758
 - uwierzytelnienia, 759
 - zarządzania, 760
 - zasięgu, 762, 764, 766
- komentarze, 1112
- kompilacja, 370
 - pliku, 421, 426
 - serwerów, 705
- kompilator gcc, 372
- komponenty MySQL, 592
- kompresja, 911
- komunikacja z serwerem, 46
- komunikat
 - błędu, 456, 667
 - pomocy, 1116
 - Using where, 341
- konfiguracja
 - Apache, 512
 - bufora kluczy, 677
 - haseł, 627
 - mysqld, 630
 - obsługi stref czasowych, 664
 - połączeń, 770
 - puli bufora InnoDB, 675
 - serwerów, 705
 - serwerów replikacji, 809
 - silnika, 654, 656
 - silnika wyszukiwania, 211
 - struktury MYSQL_BIND, 435
 - systemowej przestrzeni tabel, 657–661
 - tablicy MYSQL_BIND, 436, 439
 - zapytań preinterpretowanych, 431
 - zmiennej PATH, 824
- koniec zapytania, 52
- konstrukcja
 - AS, 91
 - new PDO(), 543
 - qq{}, 469
 - switch, 585
 - try/catch, 544

konto, 623

- anonymowe, 625
- logowania, 823
- MySQL, 727
- początkowe, 624
- użytkownika, 725, 747
- w serwerze, 48

kontrola

- dostępu, 594, 715, 746, 754, 761
- nad serwerem, 641
- współbieżności, 45

konwersja

- podzapytań, 182
- tabeli, 160
- typu, 289, 294
- zapytania, 181

kopia

- zapasowa, 785, 790, 794, 808, 815
- zapasowa binarna, 793

kopiowanie baz danych, 796, 798

koszt indeksowania, 327

kradzież danych, 717

kwalifikator, 171

kwalifikatory identyfikatora, 127

kwalifikowane odwołania, 170

L

liczba

- sekwencji, 276
- zwracanych rekordów, 462

liczbowe typy danych, 228, 232, 234, 236

liczby, 215, 873

liczby całkowite, 234

liczebność kolumny, 329

linkowanie programów klienckich, 370

lista

- argumentów, 385
- przypisać, 77
- zdarzeń, 530

Ł

łącze do skryptu, 530

M

makra przenośności, 374

mapowanie zapytań, 604

mechanizm cron, 784

metadane

- bazy danych, 161
- zbioru wynikowego, 411, 482

metoda

- col_prompt(), 503
- connect(), 450
- do(), 456, 479
- errorCode(), 562
- exec(), 554
- execute(), 471
- fetch(), 552, 555
- fetchall_arrayref(), 466
- fetchrow_array(), 452, 458, 463–465
- fetchrow_arrayref(), 460, 465
- fetchrow_hashref(), 461
- finish(), 452
- interpret_argument(), 497
- prepare(), 451, 471
- query(), 544, 555
- quote(), 471, 560
- rollback(), 487
- rtf_format_entry(), 508
- search_members(), 535

metody pobierania rekordu, 461

miejsca zarezerwowane, 470

moduł CGI.pm, 445, 513, 519

moduły MySQL, 361

modyfikator IF EXISTS, 152

MySQL, 20–22

MySQL jako usługa, 48

MySQL Workbench, 45

N

nadawanie uprawnień, 732, 743

nakładanie blokad, 354, 779, 780, 915

naprawianie tabel, 798, 801

narzędzia

- bezpośredniego dostępu, 601
- MySQL, 592

narzędzie, *Patrz* program
 nasłuchiwanie połączeń, 639
 nawiasy
 klamrowe, 469
 kwadratowe, 831, 1003
 nawiązywanie połączenia, 113, 373, 521, 545, 641, 741
 nazwa komputera, 730
 nazwy, 128
 baz danych, 605
 kont użytkowników, 728
 kolumn danych wyjściowych, 90
 niepowtarzalne odczyty, 195
 niezatwierdzone odczyty, 195

O

obiekt PDO, 554
 obliczanie
 szerokości kolumny, 413
 wartości, 90
 wyrażeń, 280
 obsługa
 bazy danych, 595, 775, 777
 błędów, 377, 453, 561
 dopełnień, 256
 kodowania znaków, 130, 219
 opcji hasła, 394
 preinterpretowanego zapytania, 441
 przyrostków, 491
 skryptów CGI, 512
 SSL, 421
 stref czasowych, 664
 Unicode, 133
 wartości nieprawidłowych, 267
 wyjątków, 548, 563
 odbieranie uprawnień, 744
 odczyty widma, 195
 odsłona reklamy, hit, 42
 odwołania
 do kolumn, 170
 do tabel, 127
 ograniczenia
 danych wejściowych, 269
 relacji, 72
 systemu operacyjnego, 605
 wyników zapytania, 89
 zasobów, 743
 ograniczniki zapytań, 310
 określanie opcji, 1117
 opcja, 1117
 CLIENT_MULTI_STATEMENTS, 426
 host, 386
 register_globals, 567
 opcje
 analizy tabel, 1166
 charakterystyczne dla serwera, 1124
 charakterystyczne dla użytkownika, 1124
 dzienników zdarzeń, 704
 formatu danych, 1193
 globalne, 1124
 mysql, 49
 mysql.server, 1150
 mysql_config, 1151
 naprawy tabel, 801, 1166
 narzędzia mysqlcheck, 802
 optymalizacji tabel, 1167
 replikacji, 1177
 sprawdzania tabel, 1166
 standardowe myisamchk, 1132
 standardowe mysql, 1139
 standardowe MySQL, 1119
 standardowe mysql_install_db, 1152
 standardowe mysql_upgrade, 1153
 standardowe mysqladmin, 1154
 standardowe mysqlbinlog, 1159
 standardowe mysqlcheck, 1163
 standardowe mysqld, 1168
 standardowe mysqld_multi, 1180
 standardowe mysqld_safe, 1182
 standardowe mysqldump, 791, 1184
 standardowe mysqlimport, 1194
 standardowe mysqlshow, 1197
 standardowe perror, 1198
 standardowe SSL, 1122
 startowe, 707
 startowe serwera, 637
 tabeli, 144
 zapytania CHECK TABLE, 802

- operacje na danych, 92
- operandy, 289
- operator, 852
 - ALL, 179
 - ANY, 179
 - COLLATE, 130, 132
 - EXISTS, 179
 - IN, 178
 - LIKE, 95
 - NOT EXISTS, 179
 - NOT IN, 177
 - NOT LIKE, 95
 - REGEXP, 337
 - SOME, 179
- operatory
 - arytmetyczne, 84, 282, 854
 - bitowe, 284, 292, 861
 - dopasowania wzorca, 865
 - grupowania, 853
 - logiczne, 84, 283, 862
 - porównania, 84, 176, 284, 856
 - rzutowania, 864
- opis
 - modułu CGI.pm, 513
 - Perl DBI, 447
 - PHP, 539
 - typów danych, 228, 831
- opóźniony zapis kluczy, 351
- oprogramowanie, 819
- oprogramowanie MySQL, 47
- optymalizacja zapytań, 333, 334
 - indeksowanie, 324
 - wczytywanie danych, 349
 - wybór formatu tabeli, 347
 - wybór typu danych, 344
 - zapytanie EXPLAIN, 333, 338
- optymalizacje sprzętowe, 682
- organizacja repozytorium, 42

P

- pakiety, 821
- pakowanie danych, 346

- parametry
 - danych wejściowych, 567
 - połączenia, 382, 393, 475, 478, 545
 - procedury składowanej, 315
- partycjonowanie tabeli, 150
- PDO, PHP Data Objects, 538, 829
- Perl DBI, 39, 445–536
- pętla pobierania
 - DBI, 464
 - rekordu, 458
- PHP, 40
- plik, *Patrz także* skrypt
 - .my.cnf, 1128
 - .mysql_history, 118
 - connect2.c, 395
 - crontab, 630, 784
 - dump_members.pl, 464
 - exec_stmt_ssl.c, 422, 426
 - fill_help_tables.sql, 828
 - index.php, 539
 - kopii zapasowej, 797
 - Makefile, 372
 - my_global.h, 374
 - my_sys.h, 374
 - mysql.h, 374
 - opcji, 796
 - php.ini, 546
 - PID, 612
 - prepared_call.c, 444
 - sampdb.cnf, 522
 - sampdb_pdo.php, 548
 - show_opt.c, 388
 - sslopt-case.h, 424
 - sslopt-vars.h, 424
- pliki
 - .csm, 139
 - .csv, 139
 - .frm, 139, 603, 789, 796
 - .ibd, 139, 796
 - .idb, 139
 - .my.cnf, 772
 - .myd, 139
 - .myi, 139
 - .php, 538

- .pl, 446
- .sql, 54
- certyfikatu, 771, 773
- dziennika zdarzeń, 611, 695, 698, 796
- kopii zapasowej, 788
- mysql, 537
- mysqli, 538
- nagłówkowe, 371, 546
- opcji, 113, 383, 477, 1124–1128
- PDO, 538, 554
- startowe powłoki, 481
- systemowej przestrzeni tabel, 796
- wsadowe, 115
- pobieranie
 - danych, 167, 566
 - informacji, 80, 82, 183
 - metadanych, 161–166
 - parametrów połączenia, 382
 - pojedynczego rekordu, 458, 462, 473
 - rekordów, 556
 - serwera MySQL, 820
- podsumowanie, 97, 907
- podzapytania, 175, 182
 - kolumny, 175
 - rekordu, 175
 - skalarne, 175, 176
 - skorelowane, 180
 - tabeli, 175
 - w klauzuli FROM, 181
- podzapytanie
 - ALL, 178
 - ANY, 178
 - EXISTS, 179
 - IN, 177
 - NOT EXISTS, 179
 - NOT IN, 177
 - SOME, 178
- pojedynczy rekord, 462
- pola formularza, 588
- polecenia
 - edycji wiersza danych, 116, 117
 - programu mysql, 1145
- polecenie
 - chgrp, 826
 - chown, 826
 - include, 548
 - mysql, 48
 - print, 478
 - quit, 51
 - require, 548
 - use strict, 449
 - use warnings, 449
- połączenie
 - localhost, 729
 - z serwerem, 49, 55, 373, 395, 425, 521, 543, 561, 770
- położenie katalogu danych, 598
- ponowne wykonywanie zapytań, 805
- porównywanie, 221, 856, 870
 - kolumn, 335
 - wartości, 336
- poziom
 - DBD, 361
 - DBI, 361
- poziomy izolacji transakcji, 195, 196
- prefiks, 157, 330
- preinterpretowane zapytania, 440
- procedura składowana, 313
- procedury, 311, 314
- program
 - connect1, 373, 376
 - connect2, 399
 - make, 372
 - metadata, 416
 - my_print_defaults, 1128
 - myisamchk, 211, 592, 601, 1130, 1136
 - myisamchkmysql, 1137
 - opcje, 1139–1144
 - polecenia, 1145
 - zmienne, 1145
 - znaki zachęty, 1138, 1148
 - mysql_config, 372, 1150
 - mysql_upgrade, 1153
 - mysqladmin, 592
 - opcje, 1154, 1155
 - polecenia, 1156
 - zmienne, 1155
 - mysqlbinlog, 803, 805, 807
 - opcje, 1159–1162
 - zmienne, 1162

program

- mysqlcheck, 592, 801
- opcje, 1163–1165
- mysqldump, 167, 592, 786–790, 1184
- opcje, 1185
- opcje formatu danych, 1193
- zmienne, 1193
- mysqlimport, 1194
- opcje, 1194, 1195
- opcje formatu danych, 1196
- mysqlshow, 166, 1196
- perror, 1198
- show_argv, 393
- show_opt, 387

programowanie MySQL, 355

programy składowane, 305, 309

projekt

- „Liga Historyczna USA”, 37
- „Oceny uczniów”, 40, 564

protokół

- SSL, 370, 421, 741, 759, 770
- TCP/IP, 762

prywatność, 1128

przeglądarka

- bazy danych, 523
- tabel, 528

przenoszenie

- katalogu danych, 614, 615, 617
- plików dzienników, 619
- stanu, 619
- systemowej przestrzeni tabel, 618
- tabel, 618

przenośność, 367

- skryptu DBI, 361
- tabel, 788

przepisywanie podzapytań, 181, 182

przeprowadzanie transakcji, 188, 486

przestrzeń tabel InnoDB, 657–662

przetwarzanie

- argumentów, 387
- opcji, 395
- opcji polecenia, 377
- transakcyjne, 189, 486
- wyników zapytania, 554
- zapytań, 399, 407

przyrostki, 491

przywracanie

- bazy danych, 79, 803
- tabel, 804

punkty pośrednie transakcji, 194

puzzle uprawnień, 767

Q

quiz, 577

R

RDBMS, 19, 33, 42

reguły nadawania nazw, 128

rejestracja zdarzeń, 692, 815

rekord, 75

relacja

- klucza zewnętrznego, 199
- typu główny – podległy, 811

relacyjna baza danych, 42

replikacja, 595, 809–815

reprezentacja tabeli, 139

rodzaje

- indeksów, 154
- kont, 623

root, 624

rotacja

- plików dzienników zdarzeń, 695
- tabel dzienników zdarzeń, 701

RTF, Rich Text Format, 39, 489

rzutowanie, 864, 872

S

sekwencje, 270, 276

serwer

- dodatkowy, 627
- główny, 810
- MySQL, 123, 592
- mysqld, 1167
- opcje, 1168–1176
- opcje replikacji, 1177
- zmienne, 1180

- podległy, 812, 815
- replikacji, 809
- WWW, 357
- silnik
 - InnoDB, 72, 140, 274, 348, 656, 796
 - MEMORY, 141, 275, 348
 - MyISAM, 140, 273
 - NDB, 142
- silniki
 - baz danych, 27, 137, 603, 655, 673
 - domyślne, 656
 - wyszukiwania, 211
- skanowanie
 - binarne, 325
 - liniowe, 325
- składnia
 - komentarzy, 1112
 - SQL, 1003–1113
- skrypt
 - count_members.pl, 462
 - db_browse.pl, 523–525
 - dump_members.php, 552
 - dump_members.pl, 451, 453, 550
 - edit_member.php, 582, 583, 589
 - edit_member.pl, 500, 501, 503, 504
 - flip_flop.pl, 520
 - gen_dir.pl, 490, 494, 510
 - index.php, 565
 - interests.pl, 506
 - mysql.server, 632, 633, 1150
 - mysql_install_db, 825, 1152
 - mysqld_multi, 632, 708
 - mysqld_safe, 632, 1181
 - need_renewal.pl, 495
 - pres_quiz.php, 577, 582
 - renewal_notify.pl, 497, 500
 - score_browse.pl, 528, 530
 - score_entry.php, 564, 568, 574
 - show_member.pl, 484, 499, 501
 - tabular.pl, 484
 - ushl_browse.pl, 532, 533
 - ushl_ft_browse.pl, 535
- skrypty
 - CGI, 512
 - DBI, 448
 - Perl, 446
 - PHP, 538, 541
 - powłoki, 115
 - sieciowe, 445
 - wiersza poleceń, 521
- słowo kluczowe
 - AS, 91
 - BEGIN, 311
 - DISTINCT, 97, 98
 - END, 311
 - GLOBAL, 125
 - LOCAL, 79
 - prompt, 1149
 - SCHEMA, 134
 - STRAIGHT_JOIN, 335
 - TEMPORARY, 153
 - UNION, 183
 - UNSIGNED, 235
- sortowanie, 131, 220, 250, 667
- sortowanie wyników zapytania, 87
- spacje, 53
- specyfikatory poziomu uprawnień, 738
- sprawdzanie, 572
 - konwersji typu, 294
 - opcji, 1128
 - operacji optymalizatora, 338
 - optymalizatora, 340
 - parametrów, 514
 - tabel, 798, 800, 801
 - wartości zwrotnej, 378
 - wyników podzapytania, 175
 - sprawdzenie wartości NULL, 558
- SQL, Structured Query Language, 21, 33, 44, 592, 1003–1113
- SSL, Secure Sockets Layer, 370, 741
- stan MySQL, 611
- status GA, 27
- strefy czasowe, 664
- strona pobierająca dane, 550
- struktura
 - katalogu, 599, 609
 - my_option, 390
 - MYSQL_BIND, 439
- superużytkownik, 748

system plików, 603, 614, 716
 szerokość kolumny, 413
 szyfrowanie, 774

Ś

śledzenie, 480, 481

T

tabela, 603

- absence, 75
- child, 203
- ENGINES, 138
- grade_event, 73
- member, 57
- parent, 203
- score, 73
- student, 71
- uprawnień, 825
- user, 761

tabele

- CSV, 789
- INFORMATION_SCHEMA, 165
- InnoDB, 796
- MEMORY, 789
- nadrzędne, 198, 202
- nietransakcyjne, 268
- partycjonowane, 150
- potomne, 198, 200
- transakcyjne, 197, 268
- tymczasowe, 145
- uprawnień, 624, 754–765

tablica

- \$score, 576
- atrybutów połączenia, 450
- hash, 450, 462
- MYSQL_BIND, 435, 439

tekst, 878–891

transakcje, 188, 190, 192, 486

tryb

- boolowski, 208
- obsługi błędów, 562
- PIPES_AS_CONCAT, 283
- SQL serwera

ANSI, 125

ANSI_QUOTES, 125, 126

PIPES_AS_CONCAT, 125

STRICT_ALL_TABLES, 124, 269

STRICT_TRANS_TABLES, 124, 269

TRADITIONAL, 124, 269

tryby pobierania rekordów, 556

tworzenie

- aplikacji, 364
- bazy danych, 54, 135
- binarnej kopii zapasowej, 793
- indeksów, 154
- klienta, 421
- konta logowania, 823
- kopii, 595
- kopii binarnej, 793, 794, 795
- kopii zapasowej, 785, 790, 795, 815
- programów, 372
- programów klienckich, 370
- programów MySQL
 - przy użyciu C, 369–444
 - przy użyciu Perl DBI, 445–536
 - przy użyciu PHP, 537–589
- sekwencji, 277
- skryptu, 455
- stron, 518
- tabel, 55, 58, 64, 143
- warunkowe tabeli, 145
- wyrażeń, 281
- wyzwalaczy, 318
- zdarzenia, 320
- zmiennej, 97

typ danych, 213, 227, 831

- BINARY, 244
- BIT, 236, 837
- BLOB, 244
- CHAR, 242
- DATE, 59, 92, 259
- DATETIME, 259, 263
- ENUM, 71, 245, 845
- MYSQL_ROW, 403
- SET, 245, 247, 845
- TEXT, 244
- TIME, 259
- TIMESTAMP, 260, 263, 264

VARBINARY, 244
 VARCHAR, 59, 242, 347
 YEAR, 261

typy

indeksów, 155
 operatorów, 282
 rzutowania, 150

typy danych

ciągi tekstowe, 219, 229, 303, 241, 838
 ciągi tekstowe binarne, 840
 ciągi tekstowe niebinarne, 842
 DBI, 447
 daty i godziny, 845
 liczbowe, 228, 232, 236, 833, 836
 wyliczeniowe, 71

U

uaktualnianie

MySQL, 712
 rekordów, 110, 186, 202

uchwyt połączenia, 406

uchwyty, 447

udostępnienie katalogu, 507

układ tabeli, 65

ukryte pola formularza, 582

Unicode, 130, 132, 133

uprawnienia, 726, 732–744, 748–750, 764, 767, *Patrz także* tabele uprawnień, puzzle uprawnień

administracyjne, 743
 funkcji składowanych, 314
 procedur, 314

uprawnienie

EXECUTE, 320
 FILE, 748, 749
 TRIGGER, 318
 USAGE, 741

uruchamianie

klientów, 712
 plików skryptów, 118
 serwera, 628, 707, 826
 jako usługi Windows, 635
 nasłuchiwanie połączeń, 639
 opcje startowe, 637

w systemach UNIX, 631
 w systemach Windows, 634
 wielu serwerów, 701, 710

ustawianie zmiennych, 1123

ustawianie zmiennych systemowych, 647, 648

ustawienia

bieżące, 132
 językowe, 668

usuwanie

bazy danych, 136
 błędów, 478, 480
 indeksu, 158
 konta użytkownika, 746
 rekordów, 110, 186
 tabel, 75, 152

UTC, Universal Coordinated Time, 260

uwierzytelnianie użytkownika, 731, 751, 759

użytkownik

proxy, 751
 root, 625, 629

używanie

bufora zapytań, 680
 indeksów, 327
 metadanych, 411
 programów składowanych, 309
 widoków, 306

W

wartości

boolowskie, 227
 ciągu tekstowego, 217
 daty i godziny, 226, 230, 256, 265, 293
 domyślnych kolumn, 231
 dopasowane, 181
 liczbowe, 215
 niedopasowane, 182
 pól bitowych, 216
 przestrzenne, 226
 zmiennych systemowych, 646

wartość

arg_type, 392
 NULL, 86, 227, 466, 558
 var_type, 391
 zmiennej stanu, 649

- weryfikacja uprawnień, 764
- widoki, 305, 306, 604
- wielkość
 - liter, 128, 129
 - tabeli, 607
- wiersz poleceń, 166, 373, 387, 391
- własne aplikacje MySQL, 355
- właściwości
 - ACID, 189
 - AUTO_INCREMENT, 270, 272
- wprowadzanie wyników, 568
- współbieżność, 352
- wstawianie rekordu, 76
- wstawienie rekordów z pliku, 78
- wtyczka, 651
 - LOCKS, 652
 - unix_auth, 752
- wtyczki uwierzytelnienia, 751
- wybór
 - API, 363
 - bazy danych, 134
 - danych z widoku, 308
 - domyślnego silnika, 656
 - formatu tabeli, 347
 - indeksów, 328
 - silnika, 655
 - systemu operacyjnego, 40
 - typu ciągu tekstowego, 254
 - typu danych, 297, 299, 344
 - typu liczbowego, 239
 - ustawień językowych, 668
- wydajność, 365
 - działania aplikacji, 365, 366
 - serwera, 429
 - systemu, 609
- wyjątek PDOException, 562
- wykonywanie
 - wielu zapytań, 426
 - zapytań, 51
- wymuszanie konwersji typu, 294
- wyniki zapytania, 554
- wyrażenia, 280, 281
- wyrażenie MATCH, 207
- wystąpienia znaków cytowania, 467
- wysyłanie przypomnień, 495
- wyszukiwanie, 531, 532, 534
 - członków, 506
 - pełnego tekstu, 204–208
 - rozszerzone, 210
 - w języku naturalnym, 208
- wyświetlanie
 - bieżących ustawień, 133
 - danych binarnych, 405
 - danych wejściowych, 584
 - informacji, 582
 - komunikatów błędów, 667
 - komunikatu pomocy, 1116
 - metadanych, 416
 - ocen, 528
 - struktury tabeli, 61
 - tablicy argumentów, 385
 - uprawnień, 744
 - wyników, 568
- wywoływanie programów MySQL, 1115
- wyzwalacze, 316, 604, 737
- wzorce, 95, 162, 285, 532, 763, 806, 865

X

- XHTML, 516
- XML, 919

Z

- zabezpieczanie
 - dostępu, 716
 - instalacji, 622, 718
 - plików opcji, 724
 - pliku gniazda, 723
- zakres wartości, 303
- zakresy typów liczbowych, 233
- zalety
 - architektury klient-serwer, 45
 - indeksowania, 324
 - prefiksów, 330
- zamykanie serwera, 628
- zapytania, 51, 1003–1113
 - generujące zbiór wyników, 555
 - modyfikujące rekordy, 401, 456, 554
 - niezłożone, 1004

- obsługi deklaracji, 1105
- obsługi kursora, 1107
- obsługi warunków, 1108
- ogólnego przeznaczenia, 405
- preinterpretowane, 428–432, 440, 470, 559
- typu DDL, 192
- złożone, 310, 1103
- zwracające dane, 402, 458
- zapytanie
 - ALTER DATABASE, 130, 134, 136
 - ALTER TABLE, 158, 273, 608
 - ANALYZE TABLE, 343
 - CALL, 440, 441, 442
 - CHANGE MASTER, 812, 813
 - CHARACTER SET, 254
 - CHECK TABLE, 777, 800
 - COMMIT, 192
 - CREATE DATABASE, 54, 130, 602
 - CREATE INDEX, 156
 - CREATE TABLE, 58, 137
 - CREATE TABLE ... LIKE, 147
 - CREATE TABLE ... SELECT, 147
 - CREATE TRIGGER, 316
 - DELETE, 110, 112, 186
 - DELIMITER, 310
 - DESCRIBE, 62, 81, 162
 - DROP, 136
 - DROP DATABASE, 134, 602, 806
 - DROP INDEX, 158, 351
 - DROP TABLE, 152
 - EXPLAIN, 333, 338, 340–343
 - FLUSH, 782
 - FLUSH PRIVILEGES, 627
 - GRANT USAGE ON, 774
 - INSERT, 76, 213, 350
 - INSTALL PLUGIN, 653
 - LOAD DATA, 76, 147, 349
 - LOAD DATA LOCAL, 79
 - LOCK TABLES, 353
 - PROCEDURE ANALYSE(), 345
 - PURGE BINARY LOGS, 814
 - RENAME USER, 727
 - REPAIR TABLE, 777, 801
 - REPLACE, 576
 - ROLLBACK, 190
 - SAVEPOINT, 194
 - SELECT, 80, 82, 167
 - SET, 97
 - SET PASSWORD, 627
 - SHOW, 131, 161
 - SHOW COLLATION, 220
 - SHOW COLUMNS, 253
 - SHOW DATABASES, 63, 602
 - SHOW FULL COLUMNS, 62, 64
 - SHOW SLAVE STATUS, 814
 - SHOW STATUS, 649
 - SHOW TABLES, 163, 164, 604
 - START SLAVE, 814
 - START TRANSACTION, 190
 - STOP SLAVE, 814
 - UNION, 168, 183, 184
 - UNION DISTINCT, 185
 - UNLOCK TABLES, 353
 - UPDATE, 110, 112, 186, 278, 402, 487
 - USE, 55
- zarządzanie
 - dziennikami zdarzeń, 693
 - kontami, 725, 727
- zasięg udzielanego dostępu, 757
- zastosowania MySQL, 35
- zatrzymanie serwera, 640
- zdarzenia, 318
- zdarzenie jednokrotne, 320
- złączenie, 108
 - CROSS JOIN, 170
 - INNER JOIN, 70, 169, 172
 - JOIN, 170
 - LEFT JOIN, 171–173
 - NATURAL LEFT JOIN, 173
 - OUTER JOIN, 171
 - RIGHT JOIN, 172
- zmiana
 - bazy danych, 136
 - hasła, 745
 - nazwy tabeli, 161
 - struktury tabeli, 158
 - typu danych, 159
 - właściciela, 826

zmienna, 96, 1123

- error_count, 645
- event_scheduler, 319
- ft_min_word_len, 211
- innodb_buffer_pool_size, 645
- log_output, 693
- max_connections, 810
- myisam_recover_options, 351
- query_cache_type, 681
- sql_mode, 124, 645
- time_zone, 666

zmienna środowiskowa

- DBI_DSN, 475
- DBI_TRACE, 481
- PATH, 824

zmienne

- bufora kluczy, 678
- narzędzia myisamchk, 1136
- programu mysql, 1145
- silnika, 663
- stanu, 644, 650, 987–994
 - bufora zapytań, 998
 - InnoDB, 994
 - SSL, 999
- systemowe, 223, 644–648, 927–972
 - InnoDB, 972–87
 - ogólnego przeznaczenia, 670
- środowiskowe, 386, 1129
- w PHP, 544
- zdefiniowane przez użytkownika, 1000

znak

- !, 114
- %, 95, 337, 767
- _, 95
- apostrofu, 468
- cudzysłowu, 468
- gwiazdki, 81
- nowego wiersza, 78
- odwrotnego apostrofu, 230
- pionowej kreski, 1003
- plus, 209
- średnika, 52
- wielokropka, 1003
- zachęty, 53, 54, 1148

znaki

- BMP, 130
- cytowania, 467
- specjalne, 416, 467, 730
- sterujące, 1127
- w identyfikatorach, 126
- wieloznaczne, 62, 209, 286, 336, 769
- zachęty, 1138

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION

- 
- A background image showing four hands holding four puzzle pieces. Three pieces are olive green and one is red. The hands are positioned around the pieces, suggesting they are being assembled.
- 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>