

» Idź do

- Spis treści
- Przykładowy rozdział

» Katalog książek

- Katalog online
- Zamów drukowany katalog

» Twój koszyk

- Dodaj do koszyka

» Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

» Czytelnia

- Fragmenty książek online

» Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl
© Helion 1991-2008

PHP6 i MySQL 5. Dynammiczne strony www. Szybki start

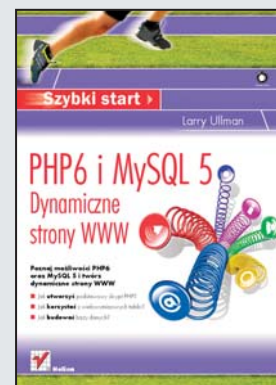
Autor: Larry Ullman

Tłumaczenie: Jaromir Senczyk

ISBN: 978-83-246-1723-4

Tytuł oryginału: [PHP 6 and MySQL 5
for Dynamic Web Sites: Visual QuickPro Guide](#)

Format: 170x230, stron: 640



Poznaj możliwości PHP6 oraz MySQL 5 i twórz dynamiczne strony WWW

- Jak utworzyć podstawowy skrypt PHP?
- Jak korzystać z wielowymiarowych tablic?
- Jak budować bazy danych?

Każda funkcjonalna i atrakcyjna dla użytkowników strona internetowa musi być na bieżąco aktualizowana, a umieszczone na niej interesujące informacje powinny być łatwo dostępne. Najpopularniejsze narzędzia typu open source, służące do tworzenia dynamicznych witryn, to język PHP i system zarządzania relacyjnymi bazami danych MySQL. Oba te narzędzia oferują wysoką wydajność, przenośność i niezawodność. Wśród wielu ogromnych możliwości oraz zalet PHP i MySQL mają także taką, że sprawne posługiwanie się nimi nie jest zbyt skomplikowane nawet dla początkujących.

Książka „PHP6 i MySQL 5. Dynamiczne strony WWW. Szybki start” zawiera precyzyjny opis czynności oraz bogato ilustrowane zrzutami ekranu niezbędne wskazówki i wyjaśnienia, ułatwiające samodzielne zbudowanie dynamicznej strony internetowej. Dzięki temu podręcznikowi nauczysz się wyszukiwać i usuwać błędy w skryptach PHP, tworzyć formularze w języku HTML oraz zapobiegać atakom na Twoje witryny. Poznasz także podstawowe i zaawansowane techniki tworzenia różnych aplikacji (na przykład stron wielojęzycznych lub obsługujących fora dyskusyjne).

- PHP i MySQL
- Tworzenie formularza w języku HTML
- Tablice i łańcuchy
- Tworzenie i wywoływanie własnych funkcji
- Wypełnianie baz danych
- Zabezpieczenia
- Stosowanie modyfikatorów
- Szyfrowanie danych
- Tworzenie uniwersalnych witryn
- Budowanie strony domowej
- Wielojęzyczna strona WWW
- Tworzenie kont użytkowników i nadawanie uprawnień

**Szybko i łatwo naucz się tworzyć funkcjonalne
oraz bezpieczne witryny internetowe**

Spis treści

	Wprowadzenie	9
	Czym są dynamiczne strony WWW?	10
	Co będzie Ci potrzebne?	16
	O tej książce	17
Rozdział 1.	Wprowadzenie do PHP	19
	Podstawy składni	20
	Przesyłanie danych do przeglądarki internetowej	24
	Wstawianie komentarzy	28
	Co to są zmienne?	32
	Łańcuchy	36
	Łączenie łańcuchów	39
	Liczby	41
	Stałe	45
	Apostrof kontra cudzysłów	48
Rozdział 2.	Programowanie w PHP	51
	Tworzenie formularza w języku HTML	52
	Obsługa formularza HTML	56
	Wyrażenia warunkowe i operatory	60
	Weryfikacja danych pochodzących z formularza	64
	Co to są tablice?	70
	Pętle for i while	88
Rozdział 3.	Tworzenie dynamicznych stron WWW	91
	Wykorzystywanie plików zewnętrznych	92
	Wyświetlanie i obsługa formularza przez jeden skrypt	102
	Tworzenie formularzy z pamięcią	107
	Tworzenie i wywoływanie własnych funkcji	110
Rozdział 4.	Wprowadzenie do MySQL	125
	Elementy bazy danych i ich nazwy	126
	Wybór typu kolumny	128
	Wybór innych właściwości kolumn	132
	Korzystanie z serwera MySQL-a	134

Rozdział 5.	Wprowadzenie do SQL	141
	Tworzenie baz danych i tabel.....	142
	Wprowadzanie rekordów.....	145
	Wybieranie danych.....	149
	Wyrażenia warunkowe.....	151
	Stosowanie LIKE i NOT LIKE.....	154
	Sortowanie wyników zapytania.....	156
	Ograniczanie wyników zapytania.....	158
	Uaktualnianie danych.....	160
	Usuwanie danych.....	162
	Funkcje.....	164
Rozdział 6.	Zaawansowany SQL i MySQL	175
	Projekt bazy danych.....	176
	Złączenia.....	191
	Grupowanie wyników zapytania.....	196
	Indeksy.....	198
	Stosowanie różnych typów tabeli.....	203
	Wyszukiwanie FULLTEXT.....	206
	Wykonywanie transakcji.....	212
Rozdział 7.	Obsługa i usuwanie błędów	217
	Ogólne typy błędów i ich usuwanie.....	218
	Wyświetlanie błędów PHP.....	224
	Sterowanie raportowaniem błędów PHP.....	226
	Tworzenie własnych funkcji obsługi błędów.....	229
	Techniki usuwania błędów z PHP.....	234
	Techniki usuwania błędów SQL i MySQL.....	238
Rozdział 8.	PHP i MySQL	241
	Modyfikacja szablonu.....	242
	Łączenie się z MySQL-em i wybieranie bazy.....	244
	Wykonywanie prostych zapytań.....	248
	Odczytywanie wyników zapytania.....	257
	Bezpieczeństwo zapytań.....	261
	Zliczanie zwróconych rekordów.....	267
	Uaktualnianie rekordów w PHP.....	269
Rozdział 9.	Tworzenie aplikacji internetowych	277
	Przekazywanie wartości do skryptu.....	278
	Stosowanie ukrytych pól formularza.....	282
	Edycja istniejących rekordów.....	288

	Stronicowanie wyników zapytań.....	295
	Wyświetlanie tabel z możliwością sortowania.....	303
Rozdział 10.	Tworzenie aplikacji internetowych	309
	Wysyłanie poczty elektronicznej	310
	Funkcje daty i czasu.....	316
	Obsługa przesyłania plików.....	320
	Skrypty PHP i JavaScript	333
	Nagłówki HTTP	340
Rozdział 11.	Sesje i „ciasteczka”	345
	Strona logowania	346
	Funkcje logowania	349
	Posługiwanie się ciasteczkami.....	354
	Sesje	367
	Zwiększanie bezpieczeństwa sesji	376
Rozdział 12.	Zabezpieczenia	379
	Zapobieganie spamowi	380
	Walidacja danych według typu	387
	Zapobieganie atakom XSS	392
	Zapobieganie wstrzykiwaniu poleceń SQL.....	395
	Szyfrowanie i bazy danych	401
Rozdział 13.	Wyrażenie regularne Perl	407
	Skrypt testujący.....	408
	Definiowanie prostych wzorców.....	412
	Stosowanie kwantyfikatorów	415
	Klasy znaków	418
	Wyszukiwanie wszystkich dopasowań.....	421
	Stosowanie modyfikatorów.....	425
	Dopasowywanie i zastępowanie wzorców.....	427
Rozdział 14.	Tworzenie uniwersalnych witryn	431
	Zbiory znaków i kodowanie	432
	Tworzenie wielojęzycznych stron WWW	434
	Unicode w PHP	438
	Uporządkowanie zbioru znaków w PHP.....	442
	Transliteracja w PHP	445
	Języki i MySQL.....	448
	Strefy czasowe i MySQL	452
	Lokalizatory	455

Rozdział 15.	Forum dyskusyjne — przykład	459
	Baza danych.....	460
	Szablony.....	469
	Strona domowa.....	478
	Strona forum.....	479
	Strona wątku.....	484
	Wstawianie wiadomości.....	489
Rozdział 16.	Rejestracja użytkowników — przykład	501
	Tworzenie szablonu	502
	Skrypty konfiguracyjne.....	508
	Tworzenie strony domowej	516
	Rejestracja	518
	Aktywacja konta	527
	Logowanie i wylogowywanie się	531
	Zarządzanie hasłami.....	537
Rozdział 17.	Sklep internetowy — przykład	547
	Tworzenie bazy danych	548
	Część administracyjna aplikacji	554
	Tworzenie szablonu części publicznej aplikacji.....	571
	Katalog produktów.....	575
	Koszyk.....	587
	Rejestrowanie zamówień.....	597
Dodatek A	Instalacja	605
	Instalacja w systemie Windows	606
	Definiowanie uprawnień MySQL.....	609
	Testowanie instalacji.....	613
	Konfigurowanie PHP	616
	Skorowidz	619

Zaawansowany SQL i MySQL

6

Rozdział ten zaczyna się w miejscu, w którym ostatni się kończył. Omówię w nim bardziej zaawansowane zagadnienia dotyczące SQL-a i MySQL-a. Poznałeś już podstawy obu tych technologii i z pewnością wystarczą Ci one do realizacji wielu projektów, ale dopiero ich bardziej wyszukane możliwości wyniosą Twe aplikacje internetowe na wyższy poziom.

Zacznę od szczegółowego omówienia procesu projektowania bazy danych, opierając się na przykładzie systemu zarządzania forum. Doprowadzi nas to oczywiście do tematu złączeń, będących integralną częścią każdej relacyjnej bazy danych. Następnie będę opisywał kolejną kategorię funkcji wbudowanych MySQL-a używanych do grupowania wyników zapytań.

Na zakończenie przejdę do bardziej zaawansowanych zagadnień. Powiem o indeksach, nauczę Cię zmieniać strukturę istniejących tabel oraz omówię typy tabel dostępne w MySQL-u. Rozdział zakończę omówieniem dwóch dodatkowych możliwości MySQL-a: przeszukiwania tekstów i transakcji.

Projekt bazy danych

Pierwsze, co musisz zrobić, gdy pracujesz z systemem zarządzania relacyjnymi bazami danych, takim jak MySQL, to utworzyć strukturę bazy (zwaną również *schematem* bazy danych). *Projektowanie* bazy danych lub inaczej *modelowanie* danych to niezbędny etap gwarantujący długotrwałe i bezproblemowe zarządzanie Twoimi informacjami. W procesie zwanym *normalizacją* eliminuje się niepotrzebne powtórzenia informacji i inne problemy, które zagrażają spójności danych.

Dzięki technikom, które poznasz w tym rozdziale, Twoje bazy będą wydajne i niezawodne. Jeden z przykładów, które zaprezentuję — forum, na którym użytkownicy mogą wymieniać się opiniami — będzie intensywnie wykorzystywany dopiero w rozdziale 15., „Forum dyskusyjne — przykład”. Jednak omówione przeze mnie zasady normalizacji odnoszą się do wszystkich aplikacji bazodanowych, jakie możesz utworzyć. (Przykład bazy sitename używany w poprzednich dwóch rozdziałach został poprawnie zaprojektowany również z punktu widzenia normalizacji, ale zagadnienie to nie zostało jeszcze omówione.)

Normalizacja

Proces normalizacji został wymyślony na początku lat siedemdziesiątych przez E.F. Codd, naukowca z firmy IBM, który wymyślił też relacyjne bazy danych. Tego rodzaju bazy to nic innego jak tylko zbiór danych ułożonych w pewien określony sposób. Istnieje szereg tak zwanych *postaci normalnych* (NF, z ang. *Normal Form*), które ułatwiają definiowanie struktury danych. W tym rozdziale omówię pierwsze trzy z nich, ponieważ w większości przypadków są one w pełni wystarczające.

Zanim zaczniesz normalizować swoją bazę danych, musisz określić, jakie funkcje będzie pełniła Twoja aplikacja. Być może będziesz musiał w tym celu porozmawiać z klientem lub samodzielnie zastanowić się nad tym zagadnieniem. W każdym razie struktura bazy danych zależy od sposobu, w jaki aplikacja będzie odwoływała się do zgromadzonych w niej informacji. Na tym etapie będziesz więc potrzebował raczej kartki i ołówka niż MySQL-a. Oczywiście, w ten sposób projektuje się wszystkie relacyjne bazy danych, nie tylko te działające w systemie MySQL.

W moim przykładowym systemie będę chciał stworzyć forum, na którym użytkownicy mogą zamieszczać swoje opinie i odpowiadać innym internautom. Aby korzystać z forum, użytkownik będzie musiał się zarejestrować, a następnie uwierzytelnić za pomocą kombinacji nazwy i hasła. Przewiduję również możliwość istnienia wielu forów poświęconych różnym tematom. W tabeli 6.1 pokazałem, jak wygląda przykładowy rekord. Baza danych będzie nosić nazwę forum.

Wskazówki

- Istnieje bardzo dobra metoda na określenie, jakiego rodzaju informacje powinny się znaleźć w bazie danych. Wystarczy, że zastanowisz się, jakiego rodzaju pytania o dane będą zadawane przez użytkowników i jakie dane będą musiały się znaleźć w odpowiedziach.
- Nauka normalizacji może sprawić Ci trudności, jeśli niepotrzebnie skoncentrujesz się na szczegółach. Każda z postaci normalnych jest zdefiniowana w dość skomplikowany sposób, a próba przełożenia tych definicji na język niefachowy może być myląca. Dlatego radzę Ci, abyś podczas lektury omówienia postaci normalnych skoncentrował się na ogólnym obrazie zmian zachodzących w schemacie bazy danych. Po zakończeniu normalizacji i otrzymaniu końcowej postaci bazy danych cały proces powinien stać się dla Ciebie wystarczająco zrozumiały.

Tabela 6.1. Przykładowy rekord pokazujący, jakiego rodzaju informacje chcę przechowywać w mojej bazie danych

Przykładowe dane forum	
Element	Przykład
username	jank
password	hasło
actual name	Jan Kowalski
user email	jank@example.com
forum	MySQL
message subject	Pytanie na temat normalizacji
message body	Nie rozumiem jednej rzeczy. Dla drugiej postaci normalnej (2NF) podano...
message date	2 lutego 2008 12:20

Klucze

W rozdziale 4., „Wprowadzenie do MySQL-a”, wspominałem już, że klucze są integralną częścią znormalizowanych baz danych. Spotkasz się z dwoma rodzajami kluczy, *głównymi* i *obcymi*. Klucz główny to unikatowy identyfikator, który podlega pewnym ściśle określonym regułom. Musi on:

- ◆ Zawsze mieć jakąś wartość (inną niż NULL).
- ◆ Mieć stałą wartość (taką, która nigdy nie ulega zmianie).
- ◆ Mieć inną wartość dla każdego rekordu w tabeli.

Najlepszym, z życia wziętym przykładem klucza głównego jest numer ubezpieczenia społecznego przydzielany obywatelom USA. Każdy ma tam własny, niezmienny, unikatowy numer polisy. Ułatwia to identyfikowanie osób. Wkrótce przekonasz się, że wielokrotnie sam będziesz tworzył we wszystkich tabelach klucze główne. Jest to po prostu dobra praktyka projektowa.

Drugi rodzaj kluczy stanowią klucze obce. Reprezentują one w tabeli B klucze główne tabeli A. Jeżeli masz na przykład bazę danych *filmy*, w której występują tabele *film* i *reżyser*, to klucz główny tabeli *reżyser* będzie pełnił rolę klucza obcego w tabeli *film*. Już niedługo zobaczysz, jak to wszystko działa w praktyce.

Tabela 6.2. *Dodałem do tabeli klucz główny, dzięki czemu będę mógł łatwo odwoływać się do rekordów*

Baza danych forum	
Element	Przykład
message ID	1
username	janek
password	hasło
actual name	Jan Kowalski
user email	jank@example.com
forum	MySQL
message subject	Pytanie na temat normalizacji
message body	Nie rozumiem jednej rzeczy. Dla drugiej postaci normalnej (2NF) podano...
message date	2 lutego 2008 12:20

Baza danych *forum* składa się w zasadzie tylko z jednej prostej tabeli (tabela 6.1), ale zanim rozpocznę proces normalizacji, chcę utworzyć w niej przynajmniej jeden klucz główny (klucze obce pojawiają się w następnych krokach).

Aby przypisać klucz główny:

1. Poszukaj pól, które spełniają wszystkie trzy warunki określone dla kluczy głównych.

W naszym przykładzie (tabela 6.1) żadna z kolumn nie spełnia kryteriów klucza głównego. Nazwa użytkownika i adres e-mail są unikalne dla każdego użytkownika forum, ale nie dla każdego rekordu bazy danych (ten sam użytkownik może umieszczać wiele wiadomości na forum). Również ten sam temat wiadomości może występować wiele razy. Tekst wiadomości będzie prawdopodobnie zawsze unikalny, ale może się zmieniać na skutek późniejszych poprawek, tym samym naruszając jedno z kryteriów wyboru klucza głównego.

2. Jeżeli nie istnieje żaden logiczny kandydat na klucz główny — wprowadź go! (tabela 6.2).

Sytuacja, w której musisz sam utworzyć klucz główny, ponieważ żadne z istniejących pól nie nadaje się do pełnienia tej roli, występuje dość często. W tym konkretnym przykładzie utworzę pole `message ID`.

Wskazówki

- Stosuję się do reguły, w myśl której w nazwach kluczy głównych powinna wystąpić przynajmniej część nazwy tabeli (np. `message`) i przyrostek `id`. Niektórzy projektanci dodają również skrót `pk` (ang. *primary key* — klucz główny).
- MySQL zezwala na stosowanie w każdej tabeli tylko jednego klucza głównego, choć możesz oprzeć go na kilku kolumnach (oznacza to, że kombinacja tych kolumn musi być unikatowa).
- Najlepiej byłoby, gdyby Twój klucz główny był zawsze liczbą całkowitą, ponieważ pozwala to MySQL-owi osiągnąć najwyższą możliwą wydajność.

Zależności

Mówiąc o zależnościach w bazach danych mam na myśli to, w jaki sposób dane z jednej tabeli są powiązane z danymi występującymi w drugiej. Zależności między dwiema tabelami mogą przybierać postać *jeden do jednego*, *jeden do wielu* lub *wiele do wielu*. (Dwie tablete tej samej bazy danych mogą być również wcale niepowiązane).

O zależności „jeden do jednego” mówimy wtedy, gdy jeden i tylko jeden element tabeli A odnosi się do jednego i tylko jednego elementu tabeli B (np. każdy mieszkaniec USA ma tylko jeden numer ubezpieczenia społecznego, a każdy numer polisy jest przyporządkowany tylko do jednego obywatela. Nikt nie może mieć dwóch polis, podobnie jak żaden numer ubezpieczenia nie może odnosić się do dwóch osób).

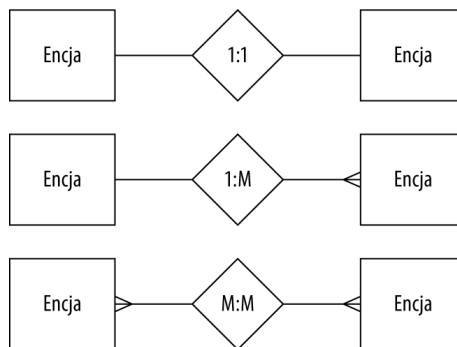
Zależność „jeden do wielu” występuje wtedy, gdy jakiś element tabeli A może odnosić się do kilku różnych elementów tabeli B. Na przykład, określenia *mężczyzna* i *kobieta* mogą być używane w odniesieniu do wielu osób, ale każdy człowiek może mieć tylko jedną płeć. Jest to najczęściej występująca zależność między tabelami w znormalizowanych bazach danych.

Istnieje też zależność „wiele do wielu”, w której kilka elementów tabeli A może odnosić się do kilku elementów tabeli B. Na przykład rekord płyty może zawierać piosenki wykonywane przez różnych artystów, a każdy artysta może mieć na swoim koncie kilka płyt. *W swoich projektach powinieneś unikać tego typu zależności*, ponieważ prowadzą one do problemów ze spójnością danych i sprzyjają ich powielaniu. Zamiast zależności „wiele do wielu” stworzysz *tabelę pośrednią*, dzięki której zależność „wiele do wielu” zostanie rozbita na dwie zależności „jeden do wielu”.

Temat zależności jest w pewien sposób związany z zagadnieniem kluczy, ponieważ klucze zdefiniowane w jednej tabeli odwołują się zazwyczaj do pól występujących w innych tabelach.

Wskazówki

- Modelowanie baz danych rządzi się pewnymi regułami. Istnieje określona konwencja reprezentowania struktury bazy (zostanie ona tutaj zachowana). Na rysunku 6.1 pokazałem symbole trzech rodzajów zależności.
- Proces projektowania bazy danych prowadzi do powstania diagramu zależności między encjami (ERD), na którym występują prostokąty reprezentujące tablete oraz symbole z rysunku 6.1.
- Istnieje wiele programów służących do tworzenia schematów baz danych. Jednym z nich jest MySQL Workbench (www.mysql.com).
- Termin „relacyjny” w określeniu „system zarządzania relacyjnymi bazami danych” odnosi się do tabel, które nazywa się *relacjami*.



Rysunek 6.1. Pokazane tu symbole często spotyka się na diagramach strukturalnych. Opisują one zależności występujące między tabelami

Pierwsza postać normalna

Jak już powiedziałem wcześniej, normalizacja bazy danych jest procesem dostosowywania struktury bazy danych do kilku *postaci*. Dostosowywanie to musi być precyzyjne i odbywać się w określonej kolejności.

Mówimy, że baza danych jest pierwszej postaci normalnej (1NF), jeżeli:

- ◆ Każda kolumna zawiera tylko jedną wartość (czyli jest *atomowa* lub *niepodzielna*).
- ◆ Żadna tabela nie ma powtarzających się kolumn dla danych pozostających w pewnej zależności.

Na przykład tabela zawierająca pole, w którym można umieścić kilka numerów telefonów (domowy, komórkowy, numer faksu itd.) *nie* jest zgodna z pierwszą postacią normalną, ponieważ w jednej kolumnie może się znaleźć więcej niż jedna wartość. Jeśli chodzi o drugi warunek, to tabela film zawierająca kolumny aktor1, aktor2, aktor3 i tak dalej, nie będzie w pierwszej postaci normalnej, ponieważ powtarzające się kolumny zawierają ten sam rodzaj informacji.

Proces normalizacji rozpoczynamy od sprawdzenia, czy aktualna struktura bazy (tabela 6.2) jest zgodna z 1NF. Kolumny, które nie są atomowe, zostaną rozbite na wiele kolumn. Jeśli tabela posiada powtarzające się, podobne kolumny, to zostaną one przekształcone w osobną tabelę.

Aby uczynić bazę zgodną z 1NF:

1. Zidentyfikuj pole, które może zawierać kilka informacji naraz.

Gdy spojrzysz jeszcze raz na tabelę 6.2, zobaczysz, że jedno z pól nie jest zgodne z 1NF: `actual name`. Zawiera ono zarówno imię jak i nazwisko użytkownika.

Pole `message date` (data dodania do bazy) przechowuje, co prawda, dzień, miesiąc i rok, ale raczej nie będzie nas interesować taki poziom szczegółowości i potraktujemy przechowywaną przez nie wartość jako niepodzielną. Zresztą pod koniec poprzedniego rozdziału pokazałem, że MySQL potrafi doskonale obsługiwać dane reprezentujące daty i czas za pomocą typu `DATETIME`.

Gdyby tabela zawierała na przykład jedną, wspólną kolumnę dla imienia i nazwiska zamiast dwóch osobnych albo przechowywała wiele numerów telefonów (stacjonarny, komórkowy, domowy, służbowy) w jednej kolumnie, to kolumny te również należałoby rozbić.

2. Rozbij wszystkie pola odszukane w kroku 1. na kilka mniejszych, niepodzielnych pól (tabela 6.3).

Tabela 6.3. Tabela z atomowymi kolumnami

Baza danych forum	
Element	Przykład
message ID	1
username	janek
password	hasło
first name	Jan
last name	Kowalski
user email	janek@example.com
forum	MySQL
message subject	Pytanie na temat normalizacji
message body	Nie rozumiem jednej rzeczy. Dla drugiej postaci normalnej (2NF) podano...
message date	2 lutego 2008 12:20

Aby poradzić sobie z tym problemem, utwórz osobne pola first name i last name, które będą zawierać tylko jedną wartość.

- Przekształć każdą grupę powtarzających się kolumn w osobną tabelę.

Problem ten nie występuje w bazie danych forum. Zademonstruję go zatem na przykładzie przedstawionym w tabeli 6.4. Powtarzające się kolumny zawierające informacje o aktorach powodują dwa problemy. Po pierwsze, dla każdego filmu można podać tylko ograniczoną liczbę aktorów. Nawet jeśli wprowadzimy kolumny aktor 1 aż do aktor 100, to ograniczeniem będzie stu aktorów. Po drugie, każdy rekord, który nie zawiera maksymalnej liczby aktorów, będzie mieć wartości NULL w nadmiarowych kolumnach. W schemacie bazy danych powinniśmy unikać kolumn zawierających wartości NULL. Dodatkowym problemem jest również to, że kolumny zawierające informacje o reżyserze i aktorach nie są atomowe.

Aby rozwiązać problemy występujące w tabeli film, utworzę dodatkową tabelę (tabela 6.5). Każdy jej rekord zawiera informacje o jednym autorze, co rozwiązuje problemy opisane powyżej. Także nazwiska i imiona aktorów są teraz przechowywane jako wartości atomowe. Zwróć również uwagę, że do nowej tabeli dodałem kolumnę indeksu głównego. Zasada, że każda tabela posiada klucz główny, wynika wprost z pierwszej postaci normalnej.

- Upewnij się, że wszystkie nowe pola utworzone w kroku 2. i 3. są zgodne z 1NF.

Wskazówki

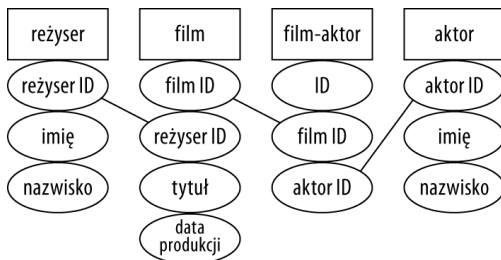
- Dostosowanie tabeli do 1NF wymaga analizy tabeli w poziomie. Wszystkie kolumny jednego rekordu przeglądamy pod kątem występowania w nich danych, które nie są atomowe oraz występowania powtarzających się, podobnych danych.
- Postacie normalne opisane są w różnych źródłach w różny sposób, często z użyciem bardziej technicznego żargonu. Najważniejszy jest jednak sens i końcowy rezultat procesu normalizacji, a nie słownictwo zastosowane do jego opisu.

Tabela 6.4. Tabela film nie jest zgodna z 1NF z dwóch powodów. Po pierwsze, zawiera powtarzające się kolumny podobnych danych (aktor1 itd.). Po drugie, kolumny aktor i reżyser nie zawierają wartości atomowych

Tabela film	
Kolumna	Wartość
film ID	976
tytuł filmu	Casablanca
rok produkcji	1943
reżyser	Michael Curtiz
aktor1	Humphrey Bogart
aktor2	Ingrid Bergman
aktor3	Peter Lorre

Tabela 6.5. Aby tabela film (tabela 6.4) była zgodna z 1NF, powiążę aktorów z filmami za pomocą tabeli film-aktor

Tabela film-aktor			
ID	Film	Imię aktora	Nazwisko aktora
1	Casablanca	Humphrey	Bogart
2	Casablanca	Ingrid	Bergman
3	Casablanca	Peter	Lorre
4	Sokół maltański	Humphrey	Bogart
5	Sokół maltański	Peter	Lorre



Rysunek 6.2. Aby baza danych o filmach była zgodna z 2NF, potrzebne są cztery tabele. Reżyserzy są reprezentowani w tabeli film za pomocą klucza reżyser ID; filmy są reprezentowane w tabeli film-aktor za pomocą klucza film ID; aktorzy są reprezentowani w tabeli film-aktor za pomocą klucza aktor ID

Druga postać normalna

Każda baza, która ma spełniać drugą postać normalną (2NF), musi być najpierw zgodna z 1NF (proces normalizacji trzeba przeprowadzać we właściwej kolejności). Następnie wszystkie kolumny, które nie zawierają kluczy (kluczy obcych), muszą być powiązane zależnością z kluczem głównym. Kolumny, które naruszają ten warunek, łatwo zidentyfikować po tym, że zawierają wartości niebędące kluczami i powtarzające się w różnych rekordach. Wartości te muszą zostać umieszczone w osobnej tabeli i być powiązane z tabelą wyjściową za pomocą klucza.

Przykładem może być fikcyjna tabela film (tabela 6.4), która zawierałaby ponad dwadzieścia razy nazwisko Martina Scorsese jako reżysera różnych filmów. Sytuacja taka jest niezgodna z drugą postacią normalną, ponieważ kolumna zawierająca informację o reżyserze nie jest kluczem i nie jest powiązana zależnością z kluczem głównym (film ID). Rozwiązanie polega na utworzeniu osobnej tabeli reżyserzy wyposażonej we własny klucz główny, który wystąpiłby jako klucz obcy w tabeli film, tworząc w ten sposób powiązanie obu tabel.

Patrząc na tabelę 6.5, można dostrzec dwa kolejne naruszenia drugiej 2NF — ani tytuły filmów, ani nazwiska aktorów nie są powiązane z kluczem głównym tabeli. Zatem baza danych o filmach w najprostszej postaci wymaga czterech tabel (rysunek 6.2). W ten sposób informacje o każdym filmie, aktorze i reżyserze są przechowywane tylko jeden raz, a każda kolumna niebędąca kluczem jest zależna od klucza głównego danej tabeli.

W zasadzie proces normalizacji można by określić jako tworzenie coraz większej liczby tabel, tak aby całkowicie wyeliminować możliwość powielenia jakichkolwiek danych.

Aby uczynić bazę zgodną z 2NF:

1. Zidentyfikuj kolumny, które nie są kluczami i które nie są powiązane z kluczem głównym.

Przyglądając się tabeli 6.3 zauważysz, że żadne z pól `username`, `first name`, `last name`, `email` i `forum` nie są kluczami (jedyną kolumną będącą kluczem jest na razie `message ID`) i żadne z nich nie jest powiązane zależnością z `message ID`. Natomiast `message subject`, `body` i `date` również nie są kluczami, ale ich wartości zależą od klucza `message ID`.

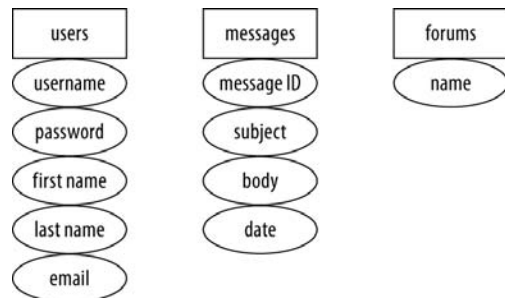
2. Utwórz odpowiednie tabele (patrz rysunek 6.3).

Najlogiczniejsza modyfikacja istniejącej struktury polega na utworzeniu trzech tabel: `users`, `forums` i `messages`.

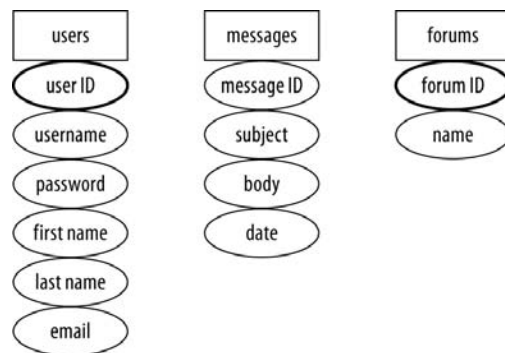
Na diagramie bazy danych każda tabela została przeze mnie oznaczona prostokątem. Jej nazwa pełni rolę nagłówka, pod którym wymienione są wszystkie kolumny (*atrybuty*) tabeli.

3. Przypisz lub utwórz nowe klucze główne (rysunek 6.4).

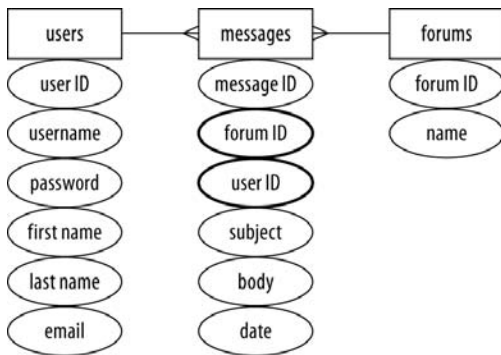
Posługując się technikami opisanymi wcześniej w tym rozdziale, upewnij się, że każda nowa tabela ma zdefiniowany klucz główny. W tabeli `users` stworzyłem klucz `user ID`, a w tabeli `forums` klucz `forum ID`. Ponieważ pole `username` w tabeli `users` oraz pole `name` w tabeli `forums` muszą być unikalne dla każdego rekordu i zawsze mieć wartość, to mógłbyś użyć ich jako kluczy głównych. Oznaczałoby to jednak, że wartości tych pól nie mogą się zmieniać (zgodnie z jednym z kryteriów wyboru klucza głównego). Użycie kluczy tekstowych zamiast numerycznych powodowałoby również nieco wolniejsze działanie bazy danych.



Rysunek 6.3. Aby baza danych forum była zgodna z 2NF, potrzebne są trzy tabele



Rysunek 6.4. Każda tabela powinna mieć własny klucz główny



Rysunek 6.5. Aby zdefiniować zależność między tymi trzema tabelami, dodałem do tabeli messages dwa klucze obce, z których każdy łączy ją z jedną z pozostałych dwóch tabel

4. Utwórz pomocnicze klucze obce, które połączą tabele zależnościami (rysunek 6.5).

Ostatni krok na drodze do zgodności z 2NF polega na dodaniu kluczy obcych, które określą powiązania między tabelami. Pamiętaj, że to, co w jednej tabeli jest kluczem głównym, w drugiej najprawdopodobniej będzie kluczem obcym.

W naszym przykładzie kolumna user ID z tabeli users łączy się z kolumną user ID z tabeli messages. Dlatego też tabela users pozostaje w zależności „jeden do wielu” z tabelą messages (każdy użytkownik może umieścić na forum wiele wiadomości, ale każda wiadomość może mieć tylko jednego autora).

Połączone zostały również dwie kolumny forum ID, tworząc w ten sposób zależność „jeden do wielu” pomiędzy tabelami messages i forums (każda wiadomość może należeć tylko do jednego forum, ale dane forum może zawierać wiele wiadomości).

Wskazówki

- Inny sposób sprawdzenia, czy tabele spełniają drugą postać normalną, polega na przyjrzeniu się powiązaniom tabel. Idealna sytuacja polega na występowaniu samych zależności „jeden do wielu”. Tabele podlegające zależnościom „wiele do wielu” mogą wymagać restrukturyzacji.
- Jeśli przyjrzymy się jeszcze raz rysunkowi 6.2, możemy zauważyć, że tabela film-aktor spełnia funkcję tabeli pośredniczącej. Pozwala ona zamienić zależność „wiele do wielu” zachodzącą pomiędzy filmami i aktorami na dwie zależności „jeden do wielu”. Tabele pośredniczące łatwo rozpoznać po tym, że wszystkie ich kolumny są kluczami obcymi. W tym przypadku kolumna odgrywająca rolę klucza głównego nie jest potrzebna, ponieważ kluczem głównym może być kombinacja obu kolumn tabeli.
- W prawidłowo znormalizowanej bazie danych w jednej tabeli nie mają prawa pojawić się dwa takie same rekordy (dwa lub więcej rekordów, w których wartości występujące w poszczególnych kolumnach są identyczne).
- Aby łatwiej przyswoić sobie proces normalizacji, zapamiętaj, że pierwsza postać normalna wiąże się z analizą tabeli w poziomie, podczas gdy druga postać normalna powstaje na skutek analizy w pionie (poszukiwania wartości powtarzających się w wielu rekordach).

Trzecia postać normalna

Mówimy, że baza danych spełnia trzecią postać normalną (3NF), jeżeli jest ona zgodna z 2NF, a każda kolumna, która nie jest kluczem, jest zależna od klucza głównego. Jeżeli prawidłowo przeszedłeś przez pierwsze dwa etapy normalizacji, prawdopodobnie dostosowanie bazy danych do 3NF nie będzie już wymagać żadnych zmian. W moim przykładzie (patrz rysunek 6.5) również nie ma problemów, które należałoby rozwiązać, aby baza była zgodna z trzecią postacią normalną. Dlatego też przedstawię je omawiając hipotetyczną sytuację.

Weźmy na przykład pojedynczą tabelę przechowującą informacje o zarejestrowanych klientach: imię, nazwisko, e-mail, numer telefonu, adres pocztowy itp. Tabela taka nie jest zgodna z 3NF, ponieważ wiele kolumn nie będzie zależnych od klucza głównego. Nazwa ulicy będzie zależać od miasta, a miasto od stanu. Również kod pocztowy nie będzie zależać od tego, jaką osobę opisuje rekord. Aby znormalizować taką bazę danych, powinno się stworzyć osobną tabelę reprezentującą państwa, osobną reprezentującą miasta (połączoną kluczem obcym z tabelą państw) i jeszcze inną dla kodów. Wszystkie te tabele byłyby połączone z tabelą opisującą klientów.

Jeśli takie rozwiązanie wydaje Ci się przesadą, to masz rację. Wyższy stopień normalizacji często nie jest konieczny. Chociaż powinno się dążyć do jak najpełniejszej normalizacji, to czasami warto ją poświęcić na rzecz prostoty. (patrz ramka „Rezygnacja z normalizacji”). W praktyce stopień normalizacji zależy od potrzeb konkretnej aplikacji i szczegółów bazy danych.

Jak już wspomniałem, nasz przykład bazy *forum* nie wymaga dalszej normalizacji (jest już zgodny z 3NF) i dlatego proces dostosowywania do trzeciej postaci normalnej przedstawię właśnie na przykładzie omówionej wyżej bazy danych o klientach.

Aby uczynić bazę zgodną z 3NF:

1. Zidentyfikuj wszystkie pola, które nie są bezpośrednio powiązane z kluczem głównym.

Jak już wspomniałem, na tym etapie poszukujemy kolumn, które powiązane są między sobą zależnościami (tak jak miasto i stan) zamiast z kluczem głównym.

W bazie danych *forum* nie występowały takie zależności. Przyjrzyjmy się tabeli *messages*. Każda wartość pola *subject* jest specyficzna dla danego *message ID*, każda wartość pola *body* jest specyficzna dla wybranego *message ID* itd.

2. Utwórz odpowiednie tabele.

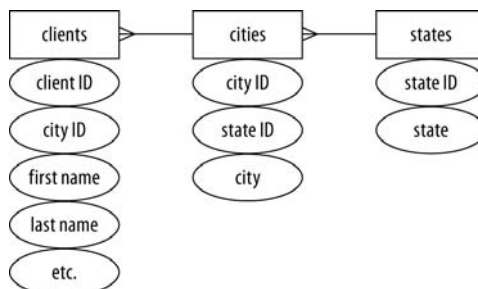
Jeśli w punkcie 1. udało się odnaleźć problematyczne kolumny, takie jak na przykład miasto i stan, to tworzymy dla nich osobne tabele *cities* i *states*.

3. Przypisz lub utwórz nowe klucze główne.

Każda tabela musi mieć klucz główny, wobec czego do nowych tabel dodajemy kolumny *city ID* i *state ID*.

4. Utwórz pomocnicze klucze obce, definiując tym samym zależności (rysunek 6.6).

Na zakończenie dodałem do tabeli *Cities* klucz obcy *State ID* oraz klucz obcy *City ID* do tabeli *Clients*. W efekcie uzyskałem połączenie rekordu klienta nie tylko z informacją o mieście, w którym jest zameldowany, ale również o stanie.



Rysunek 6.6. Uproszczona wersja hipotetycznej bazy *clients* będzie zawierać dwie nowe tabele przechowujące informacje o miastach i stanach



Wskazówka

- W praktyce nie normalizowałbym tabeli `Clients` aż do takiego stopnia. Gdybym pozostawił pola opisujące miasto i stan w tabeli `Clients`, to najgorszą rzeczą, jaka mogłaby się zdarzyć, byłaby zmiana nazwy miasta i związana z tym konieczność aktualizacji rekordów wszystkich klientów będących jego mieszkańcami. W rzeczywistości jednak sytuacja taka zdarza się bardzo rzadko.
- Mimo istnienia zasad normalizacji baz danych dwóch różnych projektantów może dokonać normalizacji tej samej bazy w nieco inny sposób. Projektowanie baz danych pozostawia pewien margines dla indywidualnych preferencji i interpretacji. Najważniejsze jest, by zaprojektowana baza danych nie naruszała postaci normalnej, gdyż prędzej czy później doprowadzi to do pojawienia się poważnych problemów.

Rezygnacja z normalizacji

Choć spełnienie trzeciej postaci normalnej jest korzystne, nie oznacza to jeszcze, że masz normalizować każdą bazę danych, z którą pracujesz. Jednocześnie powinieneś uzmysłwić sobie, że odejście od sprawdzonych metod może mieć w perspektywie dłuższego czasu katastrofalne skutki.

Z normalizacji rezygnuje się zazwyczaj z dwóch powodów — ze względu na wydajność i dla wygody. Dużo łatwiej jest ogarnąć mniejszą liczbę tabel, a poza tym — łatwiej się nimi zarządza. Ze względu na liczbę powiązań między tabelami, uaktualnianie, odczytywanie i modyfikowanie danych w znormalizowanych bazach danych trwa z reguły dłużej. Krótko mówiąc, normalizacja jest poświęceniem prostoty i szybkości na rzecz spójności i skalowalności. Należy jednak pamiętać, że istnieje znacznie więcej sposobów na przyspieszenie bazy danych niż na odzyskanie informacji utraconych na skutek przechowywania ich w źle zaprojektowanej bazie.

W miarę nabywania doświadczenia będziesz potrafił coraz lepiej modelować bazy danych, ale mimo wszystko staraj się trzymać po stronie normalizacji.

Tworzenie bazy danych

Aby zakończyć projekt bazy danych, musimy wykonać trzy ostatnie kroki:

1. Sprawdzić, czy w bazie znajdują się wszystkie potrzebne informacje.
2. Zidentyfikować typy kolumn.
3. Nazwać wszystkie elementy bazy danych.

Ostateczny projekt bazy danych został przedstawiony w tabeli 6.6. W porównaniu do rysunku 6.5 została dodana jeszcze jedna kolumna. Ponieważ wiadomość umieszczana na forum może być odpowiedzią na inną wiadomość, musimy jakoś reprezentować tę zależność w bazie. Rozwiązanie polega na dodaniu kolumny `parent_id` w tabeli `messages`. Jeśli wiadomość jest odpowiedzią, to jej pole `parent_id` będzie zawierać wartość pola `message_id` oryginalnej wiadomości (czyli `message_id` spełnia funkcję klucza obcego w tej samej tabeli). Jeśli pole `parent_id` ma wartość równą 0, oznacza to, że wiadomość stanowi początek nowego wątku, czyli nie jest odpowiedzią na żadną inną wiadomość.

Jeśli wprowadzasz jakiegokolwiek zmiany w strukturze tabel, powinieneś ponownie sprawdzić, czy spełniają one wszystkie postacie normalne, aby mieć pewność, że baza danych jest nadal znormalizowana.

Zagadnienie nazw tabel i kolumn oraz wyboru typów kolumn omówiłem już w rozdziale 4.

Gdy schemat jest gotowy, możesz utworzyć odpowiadającą mu bazę danych MySQL-a za pomocą poleceń omówionych w rozdziale 5., „Wprowadzenie do SQL”.

Aby utworzyć bazę danych:

1. Użyj wybranego klienta do dostępu do serwera MySQL-a.

Podobnie jak w poprzednim rozdziale, we wszystkich przykładach będziemy posługiwali się monitorem (klientem) `mysql`. Oczywiście możesz też śmiało korzystać z `phpMyAdmina` i innych narzędzi.

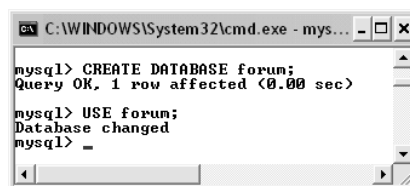
2. Utwórz bazę danych forum (rysunek 6.7).

```
CREATE DATABASE forum;
USE forum;
```

Możliwe, że Twoja konfiguracja nie pozwalała na tworzenie nowych baz danych. W takiej sytuacji po prostu wykorzystaj jakąś już istniejącą bazę i dodawaj do niej kolejne tabele.

Tabela 6.6. Ostateczny projekt bazy forum wraz z typami kolumn

forum		
Nazwa kolumny	Tabela	Typ kolumny
<code>forum_id</code>	<code>forums</code>	TINYINT
<code>name</code>	<code>forums</code>	VARCHAR(60)
<code>message_id</code>	<code>messages</code>	INT
<code>forum_id</code>	<code>messages</code>	TINYINT
<code>parent_id</code>	<code>messages</code>	INT
<code>user_id</code>	<code>messages</code>	MEDIUMINT
<code>subject</code>	<code>messages</code>	VARCHAR(100)
<code>body</code>	<code>messages</code>	LONGTEXT
<code>date_entered</code>	<code>messages</code>	TIMESTAMP
<code>user_id</code>	<code>users</code>	MEDIUMINT
<code>username</code>	<code>users</code>	VARCHAR(30)
<code>pass</code>	<code>users</code>	CHAR(40)
<code>first_name</code>	<code>users</code>	VARCHAR(20)
<code>last_name</code>	<code>users</code>	VARCHAR(40)
<code>email</code>	<code>users</code>	VARCHAR(80)



Rysunek 6.7. Najpierw musisz utworzyć i wybrać bazę danych

3. Utwórz tabelę forums (rysunek 6.8).

```
CREATE TABLE forums (
  forum_id TINYINT UNSIGNED
    NOT NULL AUTO_INCREMENT,
  name VARCHAR(60) NOT NULL,
  PRIMARY KEY (forum_id)
);
```

Kolejność, w jakiej tworzysz tabele, nie ma oczywiście znaczenia. Ja zacznę od forums. Pamiętaj, że zawsze możesz rozpisać polecenie SQL w kilku wierszach na ekranie, jeżeli tylko będzie Ci tak wygodniej.

```
C:\WINDOWS\System32\cmd.exe - mysql -u...
mysql> CREATE TABLE forums (
-> forum_id TINYINT UNSIGNED NOT NULL
-> AUTO_INCREMENT,
-> name VARCHAR(60) NOT NULL,
-> PRIMARY KEY (forum_id)
-> );
Query OK, 0 rows affected (0.16 sec)
mysql> -
```

Rysunek 6.8. Utwórz pierwszą tabelę

```
C:\WINDOWS\System32\cmd.exe - mysql -u root
mysql> CREATE TABLE messages (
-> message_id INT UNSIGNED
-> NOT NULL AUTO_INCREMENT,
-> forum_id TINYINT UNSIGNED NOT NULL,
-> parent_id INT UNSIGNED NOT NULL,
-> user_id MEDIUMINT UNSIGNED NOT NULL,
-> subject VARCHAR(100) NOT NULL,
-> body LONGTEXT NOT NULL,
-> date_entered TIMESTAMP NOT NULL,
-> PRIMARY KEY (message_id)
-> );
Query OK, 0 rows affected (0.48 sec)
mysql> -
```

Rysunek 6.9. Utwórz drugą tabelę

Tabela ta zawiera tylko dwie kolumny (sytuacja taka ma często miejsce w przypadku znormalizowanych baz danych). Ponieważ nie spodziewam się, że tabela ta będzie zawierać dużą liczbę rekordów, klucz główny otrzymał typ TINYINT. Jeśli chcesz, by baza zawierała również opis każdego forum, możesz dodać do tej tabeli kolumnę typu VARCHAR(255).

4. Utwórz tabelę messages (rysunek 6.9).

```
CREATE TABLE messages (
  message_id INT UNSIGNED
    NOT NULL AUTO_INCREMENT,
  forum_id TINYINT UNSIGNED
    NOT NULL,
  parent_id INT UNSIGNED NOT NULL,
  user_id MEDIUMINT UNSIGNED
    NOT NULL,
  subject VARCHAR(100) NOT NULL,
  body LONGTEXT NOT NULL,
  date_entered TIMESTAMP NOT NULL,
  PRIMARY KEY (message_id)
);
```

W tym przypadku klucz główny musi być zdecydowanie bardziej pojemny, ponieważ spodziewam się, że tabela ta będzie zawierać bardzo dużo rekordów. Trzy kolumny będące kluczami obcymi — `forum_id`, `parent_id` i `user_id` — będą mieć taki sam rozmiar i typ jak ich odpowiedniki będące kluczami głównymi w innych tabelach. Pole `subject` zostało ograniczone do 100 znaków, a pole `body` może zawierać znaczną ilość tekstu. Pole `date_entered` otrzymało typ `TIMESTAMP`. Będzie ono przechowywać datę i czas dodania rekordu. Jego wartość zostanie automatycznie zaktualizowana (bieżącą datą i czasem) w momencie wstawienia rekordu (właśnie w ten sposób zachowuje się typ `TIMESTAMP`).

5. Utwórz tabelę users (rysunek 6.10).

```
CREATE TABLE users (
  user_id MEDIUMINT UNSIGNED
    NOT NULL AUTO_INCREMENT,
  username VARCHAR(30) NOT NULL,
  pass CHAR(40) NOT NULL,
  first_name VARCHAR(20) NOT NULL,
  last_name VARCHAR(40) NOT NULL,
  email VARCHAR(80) NOT NULL,
  PRIMARY KEY (user_id)
);
```

Większość kolumn tej tabeli wykorzystuje definicje znane z tabeli users bazy sitename używanej w poprzednich dwóch rozdziałach. Kolumna pass jest typu CHAR(40), ponieważ dla haseł będą stosować funkcję SHA1(), która zwraca zawsze łańcuch o długości 40 znaków (patrz rozdział 5.).

6. Jeśli chcesz, możesz upewnić się, że baza danych ma taką strukturę jak powinna (rysunek 6.11).

```
SHOW TABLES;
SHOW COLUMNS FROM forums;
SHOW COLUMNS FROM messages;
SHOW COLUMNS FROM users;
```

Ten krok jest opcjonalny, ponieważ MySQL i tak wyświetla informacje o pomyślnym wykonaniu każdego wprowadzanego polecenia. Warto jednak przypomnieć sobie strukturę bazy danych.

Wskazówka

- W przypadku połączenia klucz główny-klucz obcy (na przykład forum_id tabeli forum z forum_id tabeli messages), obie kolumny muszą być tego samego typu (w naszym przykładzie: TINYINT UNSIGNED NOT NULL).

```
C:\WINDOWS\system32\cmd.exe - mysql -u root
mysql> CREATE TABLE users <
-> user_id MEDIUMINT UNSIGNED NOT NULL AUT
-> username VARCHAR(30) NOT NULL,
-> pass CHAR(40) NOT NULL,
-> first_name VARCHAR(20) NOT NULL,
-> last_name VARCHAR(40) NOT NULL,
-> email VARCHAR(80) NOT NULL,
-> PRIMARY KEY (user_id)
-> ;
Query OK, 0 rows affected (0.11 sec)
mysql> _
```

Rysunek 6.10. Trzecia i ostatnia tabela w bazie

```
C:\WINDOWS\system32\cmd.exe - mysql -u root
mysql> SHOW TABLES;
+-----+
| Tables_in_forum |
+-----+
| forums           |
| messages        |
| users           |
+-----+
3 rows in set (0.00 sec)

mysql> SHOW COLUMNS FROM forums;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| forum_id | tinyint(3) unsigned | NO | PRI | NULL | auto_increment |
| name | varchar(60) | NO | | | |
+-----+
2 rows in set (0.01 sec)

mysql> SHOW COLUMNS FROM messages;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| message_id | int(10) unsigned | NO | PRI | NULL | auto_increment |
| parent_id | int(10) unsigned | NO | | | |
| user_id | mediumint(8) unsigned | NO | | | |
| subject | varchar(100) | NO | | | |
| body | longtext | NO | | | |
| date_entered | timestamp | NO | | CURRENT_TIMESTAMP | |
+-----+
7 rows in set (0.02 sec)

mysql> SHOW COLUMNS FROM users;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| user_id | mediumint(8) unsigned | NO | PRI | NULL | auto_increment |
| username | varchar(30) | NO | | | |
| pass | char(40) | NO | | | |
| first_name | varchar(20) | NO | | | |
| last_name | varchar(40) | NO | | | |
| email | varchar(80) | NO | | | |
+-----+
6 rows in set (0.00 sec)

mysql> _
```

Rysunek 6.11. Sprawdź strukturę bazy za pomocą polecenia SHOW

Wypełnianie bazy danych

W rozdziale 15. napiszemy w PHP interfejs WWW dla bazy forums. Będzie on dostarczać standardowego sposobu wypełniania bazy danymi (poprzez rejestrowanie się użytkowników i umieszczanie wiadomości na forum). Zanim jednak dojdziemy do tego punktu, musisz się jeszcze sporo nauczyć. Dlatego na razie wypełnimy bazę danymi za pomocą klienta mysqła. Możesz wykonać po kolei poniższe kroki lub skorzystać z gotowych poleceń SQL-a dołączonych do przykładów zamieszczonych na serwerze ftp.

```

C:\WINDOWS\System32\cmd.exe - mysql -u root
mysql> INSERT INTO forums (name) VALUES
-> ('MySQL'), ('PHP'), ('Programowanie'),
-> ('HTML'), ('CSS'), ('Bazy danych');
Query OK, 6 rows affected (0.08 sec)
Records: 6 Duplicates: 0 Warnings: 0
mysql> _

```

Rysunek 6.12. Dodawanie rekordów do tabeli forums

```

C:\WINDOWS\System32\cmd.exe - mysql -u root
mysql> INSERT INTO users (username, pass,
-> first_name, last_name, email) VALUES
-> ('janek', SHA1('haslo'),
-> 'Jan', 'Kowalski', 'jank@example.com'),
-> ('ak', SHA1('haseiko'),
-> 'Arkadiusz', 'Kaczmarek', 'ak@example.com'),
-> ('GosiaM', SHA1('tajne'),
-> 'Małgorzata', 'Malinowska', 'mm@example.com');
Query OK, 3 rows affected (0.42 sec)
Records: 3 Duplicates: 0 Warnings: 0
mysql>

```

Rysunek 6.13. Dodawanie rekordów do tabeli users

Aby wypełnić bazę danymi:

1. Dodaj kilka nowych rekordów do tabeli forums (rysunek 6.12).

```

INSERT INTO forums (name) VALUES ('MySQL'),
('PHP'), ('Programowanie'), ('HTML'),
('CSS'), ('Bazy danych');

```

Ponieważ tablica messages jest zależna od wartości odczytywanych z tabel forums i users, wypełnię najpierw te ostatnie. W przypadku powyższego polecenia INSERT wystarczy jedynie podać wartość kolumny name (MySQL automatycznie nada wartość kolumnie forum_id).

2. Dodaj nowe rekordy do tabeli users (rysunek 6.13).

```

INSERT INTO users (username, pass,
first_name, last_name, email) VALUES
('janek', SHA1('haslo'),
'Jan', 'Kowalski', 'jank@example.com'),
('ak', SHA1('haseiko'),
'Arkadiusz', 'Kaczmarek', 'ak@example.com'),
('GosiaM', SHA1('tajne'), 'Małgorzata',
'Malinowska', 'mm@example.com');

```

Jeśli masz wątpliwości co do składni polecenia INSERT lub zastosowania funkcji SHA1(), skorzystaj z informacji podanych w rozdziale 5.

3. Wstaw nowe rekordy do tabeli messages (patrz rysunek 6.14).

```
SELECT * FROM forums;
SELECT user_id, username FROM users;
INSERT INTO messages (forum_id,
parent_id, user_id, subject, body)
VALUES
(1, 0, 1, 'Pytanie na temat normalizacji',
↳ 'Nie rozumiem jednej rzeczy. Dla drugiej
↳ postaci normalnej (2NF) podano...'),
(1, 0, 2, 'Projekt bazy danych', 'Projektuję
↳ nową bazę danych i natrafiłem na pewien
↳ problem. Ile tabel powinna mieć moja
↳ baza?...'),
(1, 2, 1, 'Projekt bazy danych', 'Liczba
↳ tabel w Twojej bazie danych...'),
(1, 3, 2, 'Projekt bazy danych', 'OK,
↳ dziękuję!'),
(2, 0, 3, 'Błędy PHP', 'Próbuję uruchomić
↳ skrypty z rozdziału 3. i pierwszy skrypt
↳ kalkulatora nie działa. Gdy akceptuję
↳ formularz...');
```

Ponieważ dwa pola tabeli messages (forum_id oraz user_id) odwołują się do wartości przechowywanych w innych tabelach, przed wstawieniem nowych rekordów dokonam wyboru tych wartości. Na przykład, gdy użytkownik jank utworzy nową wiadomość na forum MySQL-a, musisz użyć forum_id równy 1 i user_id równy 1.

Sprawę dodatkowo komplikuje kolumna parent_id. Musi ona zawierać wartość message_id tej wiadomości, na którą odpowiedź stanowi nowa wiadomość. Druga wiadomość umieszczona w bazie danych będzie mieć message_id równy 2 i wobec tego każda wiadomość stanowiąca odpowiedź na tę wiadomość będzie musiała mieć wartość parent_id równą 2.

W tworzonych przez Ciebie skryptach PHP, wszystko to będzie wyglądało znacznie prościej. Muszę teraz jednak wyłożyć Ci całą teorię, posługując się terminologią języka SQL.

Zwróć również uwagę, że nie wprowadziłem wartości dla pola date_entered. MySQL automatycznie umieści w nim bieżącą datę i czas, ponieważ jest to kolumna typu TIMESTAMP.

4. Powtórz kroki od 1. do 3., aby zapełnić bazę danymi.

We wszystkich kolejnych przykładach w tym rozdziale będę wykorzystywał bazę, którą właśnie zapełniłem. Jeśli chcesz, możesz wykonać u siebie te same polecenia INSERT co ja lub utworzyć swoje własne.

```
C:\WINDOWS\System32\cmd.exe - mysql -u root

mysql> SELECT * FROM forums;
+----+-----+
| forum_id | name |
+----+-----+
| 1 | MySQL |
| 2 | PHP |
| 3 | Programowanie |
| 4 | HTML |
| 5 | CSS |
| 6 | Bazy danych |
+----+-----+
6 rows in set (0.00 sec)

mysql> SELECT user_id, username FROM users;
+----+-----+
| user_id | username |
+----+-----+
| 1 | jank |
| 2 | ak |
| 3 | GosiaH |
+----+-----+
3 rows in set (0.00 sec)

mysql> INSERT INTO messages (forum_id,parent_id, user_id, subject, body) VALUES
-> (1, 0, 1, 'Pytanie na temat normalizacji', 'Nie rozumiem jednej rzeczy. Dla drugiej postaci normalnej (2NF) podano
o...'),
-> (1, 0, 2, 'Projekt bazy danych', 'Projektuję nową bazę danych i natrafiłem na pewien problem. Ile tabel powinna
mieć moja baza?...'),
-> (1, 2, 1, 'Projekt bazy danych', 'Liczba tabel w Twojej bazie danych...'),
-> (1, 3, 2, 'Projekt bazy danych', 'OK, dziękuję!'),
-> (2, 0, 3, 'Błędy PHP', 'Próbuję uruchomić skrypty z rozdziału 3. i pierwszy skrypt kalkulatora nie działa. Gdy ak
ceptuję formularz...');
Query OK, 5 rows affected (0.00 sec)
Records: 5 Duplicates: 0 Warnings: 0

mysql>
```

Rysunek 6.14. W przypadku znormalizowanych baz danych bardzo często spotkasz się z sytuacją, w której, aby ustawić jakiś rekord do tabeli, będziesz musiał znać wartości przechowywane w innych tabelach

Złączenia

Ponieważ relacyjne bazy danych mają złożoną strukturę, aby wydobyć te informacje, które najbardziej nas interesują, musimy czasem wykonać jakieś niestandardowe zapytanie. Na przykład, jeśli chcesz dowiedzieć się, jakie wiadomości zawiera forum MySQL-a, musisz najpierw dowiedzieć się, jaka jest wartość `forum_id` dla tego forum, a następnie użyć jej do pobrania wszystkich rekordów tabeli `message`, które mają taką wartość `forum_id`. Jak z tego wynika, to proste zadanie wymaga wykonania dwóch zapytań. Jednak stosując złączenie, możesz poradzić sobie z nim w jednym kroku.

Złączenie jest zapytaniem SQL-a używającym dwóch lub więcej tabel i tworzącym wirtualną tabelę wyników. W specyfikacji SQL-a występują dwa główne typy złączeń: wewnętrzne i zewnętrzne (oba mają szereg podtypów).

Złączenie wewnętrzne zwraca wszystkie rekordy podanych tabel, dla których zachodzi dopasowanie. Na przykład, aby uzyskać wszystkie wiadomości zamieszczone na forum MySQL, powinieneś użyć następującego złączenia wewnętrznego (rysunek 6.15):

```
SELECT * FROM messages
INNER JOIN forums
ON messages.forum_id =
forums.forum_id
WHERE forums.name = 'MySQL'
```

Złączenie to wybiera wszystkie kolumny obu tabel, jeśli spełnione są dwa warunki. Po pierwsze, kolumna `forums.name` musi zawierać wartość MySQL (której odpowiada wartość `forum_id` równa 1). Po drugie, wartość `forum_id` w tabeli `forums` musi odpowiadać wartości `forum_id` w tabeli `messages`. Ze względu na sprawdzenie równości dotyczące pól w różnych tabelach (`messages.forum_id = forums.forum_id`) złączenie takie nazywa się równościowym.

Złączenia wewnętrzne możesz również zapisywać bez stosowania klauzuli `INNER JOIN`:

```
SELECT * FROM messages, forums WHERE
messages.forum_id = forums.forum_id
AND forums.name = 'MySQL'
```

Jeżeli wybierasz dane z wielu tabel i kolumn, a w kilku tabelach występują kolumny o takiej samej nazwie, musisz zastosować notację z kropką (*tabela.kolumna*). W przypadku relacyjnych baz danych robi się to praktycznie zawsze, ponieważ klucz główny jednej tabeli ma taką samą nazwę jak obcy klucz drugiej. Jeżeli nie wskażesz jednoznacznie kolumny, do której się odwołujesz, zobaczysz komunikat o błędzie (rysunek 6.16).

```
C:\WINDOWS\system32\cmd.exe - mysql -u root
mysql> SELECT * FROM messages INNER JOIN forums
-> ON messages.forum_id = forums.forum_id
-> WHERE forums.name = 'MySQL';
+-----+-----+-----+-----+-----+-----+
| message_id | forum_id | parent_id | user_id | subject | body |
+-----+-----+-----+-----+-----+-----+
| 1 | 1 | 0 | 1 | Pytanie na temat normalizacji | Nie rozumiem jednej rzeczy. Dla drug |
| 2 | 1 | 0 | 2 | Projekt bazy danych | Projektuję nową bazę danych i natraf |
| 3 | 1 | 2 | 1 | Projekt bazy danych | Liczba tabel w Twojej bazie danych.. |
| 4 | 1 | 3 | 2 | Projekt bazy danych | OK, dziękuję! |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
mysql>
```

Rysunek 6.15. To złączenie zwraca kolumny obu tabel dla rekordów, dla których wartości `forum_id` są równe MySQL (1)

```
C:\WINDOWS\system32\cmd.exe - mysql -u root
mysql> SELECT * FROM messages, forums WHERE
-> forum_id = forum_id
-> AND forums.name = 'MySQL';
ERROR 1052 (23000): Column 'forum_id' in where clause is ambiguous
mysql>
```

Rysunek 6.16. Ogólne odwołanie do kolumny występującej w kilku tabelach spowoduje wystąpienie błędu niejednoznaczności

Złączenia zewnętrzne różnią się od złączeń wewnętrznych tym, że potrafią zwracać rekordy, które nie spełniają wyrażenia warunkowego. Istnieją trzy podtypy złączeń zewnętrznych: *left* (lewe), *right* (prawe) i *full* (pełne). Przykładem pierwszego podtypu jest następujące złączenie:

```
SELECT * FROM forums
LEFT JOIN messages ON
forums.forum_id = messages.forum_id;
```

Najważniejsze przy tego rodzaju złączeniach jest to, które tabele należy wymienić jako pierwsze. W powyższym przykładzie w przypadku udanego dopasowania zwrócone zostaną wszystkie rekordy tabeli forums wraz ze wszystkimi informacjami z tabeli messages. Jeśli dla danego rekordu tabeli forums nie można dopasować informacji w tabeli messages, to zamiast nich zostaną zwrócone wartości NULL (rysunek 6.17).

Jeżeli w warunku porównania złączenia zewnętrznego bądź wewnętrznego występuje kolumna o takiej samej nazwie w obu tabelach, możesz uprościć zapis zapytania, stosując klauzulę USING:

```
SELECT * FROM messages
INNER JOIN forums
USING (forum_id)
WHERE forums.name = 'MySQL'
SELECT * FROM forums
LEFT JOIN messages
USING (forum_id)
```

Zanim przejdę do przykładów, jeszcze dwie uwagi. Po pierwsze, ponieważ składnia wymagana przy tworzeniu złączeń jest dość skomplikowana, przy ich pisaniu przydają się aliasy omówione w rozdziale 5. Po drugie, ponieważ złączenia często zwracają dużo danych, to najlepiej zawsze określić, które kolumny nas interesują, zamiast wybierać wszystkie.

forum_id	name	message_id	forum_id	parent_id	user_id	subject	body
1	MySQL	1	1	0	1	Pytanie na temat normalizacji	Nie rozumiem
1	MySQL	2	1	0	2	Projekt bazy danych	Projektuję no
1	MySQL	3	1	2	1	Projekt bazy danych	Liczba tabel
1	MySQL	4	1	3	2	Projekt bazy danych	Oh, dziękuję!
2	PHP	5	2	0	3	Błędy PHP	Próbuje uruch
3	Programowanie	NULL	NULL	NULL	NULL	NULL	NULL
4	HTML	NULL	NULL	NULL	NULL	NULL	NULL
5	CSS	NULL	NULL	NULL	NULL	NULL	NULL
6	Bazy danych	NULL	NULL	NULL	NULL	NULL	NULL
7	Taniec nowoczesny	NULL	NULL	NULL	NULL	NULL	NULL

10 rows in set (0.42 sec)

Rysunek 6.17. To złączenie zewnętrzne zwraca więcej rekordów niż złączenie wewnętrzne, ponieważ zwraca wszystkie wiersze pierwszej tabeli. Zapytanie to zwraca również nazwy wszystkich forów, nawet jeśli nie zawierają one jeszcze żadnej wiadomości

Aby wykorzystać złączenia:

1. Pobierz nazwę forum i temat wiadomości dla każdego rekordu tabeli messages (patrz rysunek 6.18).

```
SELECT f.name, m.subject
FROM forums
AS f INNER JOIN messages AS m
USING (forum_id) ORDER BY f.name;
```

Zapytanie to zawiera złączenie wewnętrzne, które spowoduje w efekcie zastąpienie wartości forum_id z tabeli messages odpowiadającą jej wartością name z tabeli forums dla każdego rekordu tabeli messages. W wyniku otrzymasz zatem temat każdej wiadomości wraz z nazwą forum, do którego należy ta wiadomość.

Zwróć uwagę, że w połączeniach nadal możesz używać klauzul ORDER BY.

```
C:\WINDOWS\System32\cmd.exe - mysql -u root
mysql> SELECT f.name, m.subject FROM forums
-> AS f INNER JOIN messages AS m
-> USING (forum_id) ORDER BY f.name;
+----+-----+
| name | subject |
+----+-----+
| MySQL | Projekt bazy danych |
| MySQL | Projekt bazy danych |
| MySQL | Pytanie na temat normalizacji |
| MySQL | Projekt bazy danych |
| PHP | Błędy PHP |
+----+-----+
5 rows in set (0.06 sec)
mysql> _
```

Rysunek 6.18. Podstawowe złączenie wewnętrzne zwraca w tym przykładzie tylko dwie kolumny

2. Pobierz temat i datę wprowadzenia każdej wiadomości wysłanej przez użytkownika jankę (rysunek 6.19).

```
SELECT m.subject,
DATE_FORMAT(m.date_entered,
'%M %D, %Y') AS Date
FROM users AS u INNER JOIN
messages AS m
USING (user_id)
WHERE u.username = 'jank';
```

Również to złączenie używa dwóch tabel, users i messages. Złączenie obu tabel odbywa się za pomocą kolumny user_id i dlatego została ona umieszczona wewnątrz klauzuli USING. Wyrażenie warunkowe WHERE pozwala zidentyfikować interesującego nas użytkownika, a funkcja DATE_FORMAT sformatować wartość pola date_entered.

```
C:\WINDOWS\System32\cmd.exe - mysql -u root
mysql> SELECT m.subject, DATE_FORMAT(m.date_entered, '%M %D, %Y') AS Date FROM users
-> AS u INNER JOIN messages AS m
-> USING (user_id)
-> WHERE u.username = 'jank';
+----+-----+
| subject | Date |
+----+-----+
| Pytanie na temat normalizacji | October 30th, 2008 |
| Projekt bazy danych | October 30th, 2008 |
+----+-----+
2 rows in set (0.06 sec)
mysql> _
```

Rysunek 6.19. Nieco bardziej skomplikowana wersja złączenia wewnętrznego tabeli users i messages

3. Pobierz identyfikator wiadomości, temat oraz nazwę forum dla każdej wiadomości, której autorem jest użytkownik jank (patrz rysunek 6.20).

```
SELECT m.message_id, m.subject,
       f.name FROM users AS u
       INNER JOIN
       messages AS m USING (user_id)
       INNER JOIN forums AS f
       USING (forum_id)
       WHERE u.username = 'jank';
```

Powyższe złączenie jest podobne do zastosowanego w punkcie 2., ale idzie o krok dalej, dołączając trzecią tabelę. Zwróć szczególną uwagę na sposób konstrukcji złączenia wewnętrznego trzech tabel oraz zastosowanie aliasów w celu łatwiejszego odwoływania się do tabel i ich kolumn.

```
C:\WINDOWS\System32\cmd.exe - mysql -u root
mysql> SELECT m.message_id, m.subject,
-> f.name FROM users AS u INNER JOIN
-> messages AS m USING (user_id)
-> INNER JOIN forums AS f
-> USING (forum_id)
-> WHERE u.username = 'jank';
+-----+-----+-----+
| message_id | subject | name |
+-----+-----+-----+
| 1 | Pytanie na temat normalizacji | MySQL |
| 3 | Projekt bazy danych | MySQL |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> _
```

Rysunek 6.20. Złączenie wewnętrzne wszystkich trzech tabel bazy

4. Dla każdego użytkownika pobierz nazwę użytkownika, temat wiadomości i nazwę forum (patrz rysunek 6.21).

```
SELECT u.username, m.subject,
       f.name FROM users AS u LEFT JOIN
       messages AS m USING (user_id)
       LEFT JOIN forums AS f
       USING (forum_id);
```

Gdybyś zastosował podobne złączenie wewnętrzne, to użytkownicy, którzy nie są jeszcze autorami żadnej wiadomości, nie zostaliby uwzględnieni (patrz rysunek 6.22). Zatem jeśli interesują Cię wszyscy użytkownicy, powinieneś użyć złączenia zewnętrznego. Zwróć uwagę, że tabela, która ma być uwzględniona w całości (users w tym przypadku), musi być podana jako pierwsza tabela złączenia typu left.

```
C:\WINDOWS\System32\cmd.exe - mysql -u root
mysql> SELECT u.username, m.subject,
-> f.name FROM users AS u LEFT JOIN
-> messages AS m USING (user_id)
-> LEFT JOIN forums AS f
-> USING (forum_id);
+-----+-----+-----+
| username | subject | name |
+-----+-----+-----+
| jank | Pytanie na temat normalizacji | MySQL |
| jank | Projekt bazy danych | MySQL |
| ak | Projekt bazy danych | MySQL |
| ak | Projekt bazy danych | MySQL |
| GosiaM | Błędy PHP | PHP |
| tomekj | NULL | NULL |
+-----+-----+-----+
6 rows in set (0.01 sec)

mysql>
```

Rysunek 6.21. To złączenie zewnętrzne zwraca dla każdego użytkownika temat każdej wiadomości i nazwę każdego forum. Jeśli użytkownik nie umieścił jeszcze żadnej wiadomości na forum (tak jak tomekj widoczny na dole rysunku), to temat wiadomości i nazwa forum będą mieć dla niego wartości NULL

Wskazówki

- Możesz nawet złączyć tabelę z nią samą.
- Złączenia można tworzyć z wykorzystaniem wyrażeń warunkowych odwołujących się do dowolnych kolumn. Nie muszą one pełnić roli kluczy, choć sytuacja taka jest najczęstsza.
- Stosując składnię *bazadanych.tabela.kolumna*, możesz wykonywać złączenia tabeli należących do różnych baz danych, o ile tylko wszystkie znajdują się w tym samym serwerze. Ta metoda nie zadziała, jeżeli bazy komunikują się ze sobą za pośrednictwem sieci.
- Złączenia, które nie zawierają klauzuli WHERE (np. `SELECT * FROM urls, url_associations`), nazywamy *złączeniami pełnymi*. Zwracają one rekordy z obu tabel. W przypadku dużych tabel ilość zwracanych danych może stanowić pewien problem.
- Nigdy nie zostanie zwrócona wartość NULL występująca w kolumnie, do której odwołujemy się w złączeniu. To dlatego, że nie jest ona równa żadnej innej wartości, w tym także innym wartościom NULL.

```

C:\WINDOWS\System32\cmd.exe - mysql -u root
mysql> SELECT u.username, m.subject,
-> f.name FROM users AS u INNER JOIN
-> messages AS m USING (user_id)
-> INNER JOIN forum AS f
-> USING (forum_id);
+-----+-----+-----+
| username | subject | name |
+-----+-----+-----+
| jank     | Pytanie na temat normalizacji | MySQL |
| jank     | Projekt bazy danych | MySQL |
| ak       | Projekt bazy danych | MySQL |
| ak       | Projekt bazy danych | MySQL |
| GosiaM   | Błędy PHP | PHP |
+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> _

```

Rysunek 6.22. To złączenie wewnętrzne nie zwróci użytkownika, który nie umieścił jeszcze żadnej wiadomości na forum (tak jak tomekj widoczny na dole rysunku 6.21)

Grupowanie wyników zapytania

W poprzednim rozdziale przedstawiłem dwie różne klauzule — ORDER BY i LIMIT — mające wpływ na wynik zapytania. Pierwsza z nich umożliwia określenie uporządkowania zwracanych rekordów, a druga pozwala określić, które z rekordów będących wynikiem zapytania zostaną w rzeczywistości zwrócone. Kolejna klauzula, GROUP BY, grupuje zwracane dane w podobne do siebie bloki informacji. Na przykład, aby pogrupować wszystkie wiadomości według forum, wpisz:

```
SELECT * FROM messages
GROUP BY forum_id;
```

Zwracane dane nie będą miały postaci pojedynczych rekordów, tylko grup informacji. Zamiast dużej liczby wiadomości pochodzących z danego forum zobaczysz je wszystkie w postaci jednego rekordu. Ten przykład nie jest być może szczególnie przydatny, ale wystarczający do przedstawienia samej koncepcji.

Z klauzulą GROUP BY często używa się jednej z funkcji agregujących przedstawionych w tabeli 6.7 (funkcji tych można używać również niezależnie od klauzuli GROUP BY).

Na wyrażenie GROUP BY możesz oczywiście nakładać warunki WHERE, ORDER BY i LIMIT, tworząc zapytania w rodzaju:

```
SELECT kolumny FROM tabela WHERE warunek
GROUP BY kolumna ORDER BY kolumna
LIMIT x, y;
```

Aby zgrupować dane:

1. Policz zarejestrowanych użytkowników (patrz rysunek 6.23).

```
SELECT COUNT(user_id) FROM users;
```

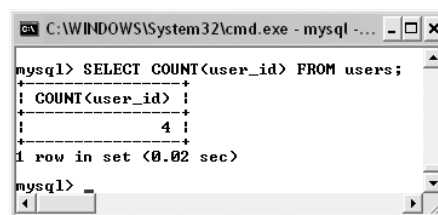
COUNT() jest prawdopodobnie najpopularniejszą z funkcji agregujących. Z jej pomocą możesz szybko zliczyć rekordy tak jak w przypadku tabeli users. Zwróć uwagę, że nie wszystkie zapytania stosujące funkcje agregujące używają klauzuli GROUP BY.

2. Policz, ile razy każdy użytkownik umieścił na forum swoją wiadomość (patrz rysunek 6.24).

```
SELECT username,
COUNT(message_id) AS Number
FROM users LEFT JOIN messages AS m
USING (user_id) GROUP BY (m.user_id);
```

Tabela 6.7. Funkcje grupujące MySQL-a

Funkcja	Przeznaczenie
AVG()	Zwraca średnią wartość z podanej kolumny.
MIN()	Zwraca najmniejszą wartość z podanej kolumny.
MAX()	Zwraca największą wartość z podanej kolumny.
SUM()	Zwraca sumę wszystkich wartości z danej kolumny.
COUNT()	Zwraca liczbę rzędów.
GROUP_CONCAT()	Zwraca konkatenaację wartości kolumny.



Rysunek 6.23. To zapytanie grupujące zwraca liczbę wartości user_id w tabeli users

Jest to rozszerzona wersja zapytania występującego w kroku 1., ale zamiast zliczać samych użytkowników, podaje liczbę wiadomości związanych z każdym użytkownikiem. Zastosowanie złączenia umożliwia dostęp do informacji w dwóch tabelach. Aby uwzględnić użytkowników, którzy nie byli jeszcze aktywni na forum, zastosowałem złączenie wewnętrzne.

3. Znajdź dwóch najaktywniejszych użytkowników forum (rysunek 6.25).

```
SELECT username,
       COUNT(message_id) AS Number
FROM users
LEFT JOIN messages AS m
USING (user_id)
GROUP BY (m.user_id)
ORDER BY Number DESC LIMIT 2;
```

```
mysql> SELECT username,
-> COUNT(message_id) AS Number
-> FROM users LEFT JOIN messages AS m
-> USING (user_id) GROUP BY (m.user_id);
+-----+-----+
| username | Number |
+-----+-----+
| tomekj   | 0      |
| jank     | 2      |
| ak       | 2      |
| GosiaM   | 1      |
+-----+-----+
4 rows in set (0.01 sec)

mysql>
```

Rysunek 6.24. To zapytanie *GROUP BY* podaje liczbę wiadomości umieszczonych na forum przez każdego użytkownika

```
mysql> SELECT username,
-> COUNT(message_id) AS Number
-> FROM users LEFT JOIN messages AS m
-> USING (user_id) GROUP BY (m.user_id)
-> ORDER BY Number DESC LIMIT 2;
+-----+-----+
| username | Number |
+-----+-----+
| ak       | 2      |
| jank     | 2      |
+-----+-----+
2 rows in set (0.01 sec)

mysql>
```

Rysunek 6.25. Klauzula *ORDER BY* pozwala posortować autorów na podstawie liczby wiadomości umieszczonych na forum. Klauzula *LIMIT* powoduje, że podane zostaną tylko dwa wyniki

W zapytaniach z klauzulą *GROUP BY* nadal możesz sortować wyniki zapytań. Proces ten możesz uprościć, stosując alias *Number* dla wartości *COUNT(message_id)*.

Wskazówki

- Jak już miałeś okazję się przekonać, *NULL* jest bardzo specyficzną wartością. Interesujące jest to, że wyrażenie *GROUP BY* umieszcza wszystkie wartości *NULL* w tej samej grupie, ponieważ wszystkie one wykazują taki sam „brak wartości”.
- Funkcja *COUNT()* zlicza jedynie wystąpienia wartości różnych od *NULL*. Dlatego pamiętaj, aby użyć jej dla każdej kolumny (*) lub dla kolumn, które nie zawierają wartości *NULL* (na przykład klucza głównego). Jeśli zapytanie użyte w punkcie 2. (rysunek 6.24) stosowałoby funkcję *COUNT()* dla wszystkich kolumn (*) zamiast tylko dla *message_id*, to pokazałoby błędną wartość równą 1 dla wszystkich użytkowników, którzy nie napisali jeszcze żadnej wiadomości. Stałoby się tak, ponieważ zapytanie zwróciłoby dokładnie jeden rekord dla każdego z tych użytkowników.
- Opanowanie klauzuli *GROUP BY* i funkcji omówionych w tym podrozdziale zajmie Ci trochę czasu. Za każdym razem, gdy użyjesz nieprawidłowej składni, zostanie zgłoszony błąd. Poeksperymentuj trochę z monitorem myślenia, aby zapamiętać, jak dokładnie muszą być sformułowane wszystkie zapytania, których będziesz chciał używać w swoich aplikacjach.
- Z klauzulą *GROUP BY* związana jest klauzula *HAVING*, która zachowuje się jak klauzula *WHERE* zastosowana do grupy rekordów.

Indeksy

Mechanizm *indeksów* to specjalny system wykorzystywany w bazach danych do poprawy ich wydajności. Zakładając na swych tabelach indeksy, sprawiasz, że MySQL przywiązuje wagę do określonych kolumn. Aby indeksy mogły być wykorzystywane w sposób maksymalnie efektywny, MySQL przechowuje je w osobnych plikach.

Na każdej tabeli można założyć co najmniej 16 indeksów, a każdy indeks może obejmować do 15 kolumn. Choć rozciąganie indeksu na kilka kolumn może wydawać się zastanawiające, przydaje się to podczas częstego przeszukiwania tego samego zestawu kolumn (np. imion i nazwisk, miast i województw).

Nie powinieneś jednak przesadzać z indeksowaniem. Choć przyspiesza ono odczytywanie informacji z bazy, to jednocześnie spowalnia ich modyfikowanie (ponieważ informacje o zmianach muszą trafić do indeksów). Najlepszym wyjściem jest zakładanie indeksów na kolumnach, które:

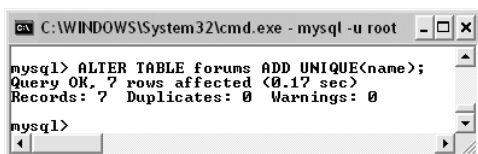
- ◆ Często wymieniane są w zapytaniach po klauzulach WHERE.
- ◆ Często wymieniane są w zapytaniach po klauzulach ORDER BY.
- ◆ Często wykorzystywane są jako główny punkt złączenia.
- ◆ Mają wiele różnych wartości (nie powinno się indeksować kolumn zawierających wiele powtarzających się wartości).

W MySQL-u występują cztery rodzaje indeksów: INDEX (standardowy), UNIQUE (który wymaga, aby w każdym rzędzie występowały inne wartości), FULLTEXT (do wyszukiwań FULLTEXT) oraz PRIMARY KEY (klucz główny, będący szczególnym przypadkiem indeksu UNIQUE). Pamiętaj, że dana kolumna może być tylko raz zaindeksowana i wobec tego wybierz dla niej najodpowiedniejszy rodzaj indeksu.

Wyposażony w te wiadomości zmodyfikuję bazę forum, dodając do niej indeksy. W tabeli 6.8 podałem indeksy utworzone dla poszczególnych kolumn za pomocą polecenia ALTER omówionego w ramce.

Tabela 6.8. Indeksy używane przez bazę danych forum. Nie wszystkie kolumny są indeksowane, a dwa indeksy zostały utworzone dla par kolumn: *user.pass* i *user.username* oraz *messages.body* i *messages.subject*

Indeksy bazy danych forum		
Nazwa kolumny	Tabela	Typ indeksu
forum_id	forums	PRIMARY
name	forums	UNIQUE
message_id	messages	PRIMARY
forum_id	messages	INDEX
parent_id	messages	INDEX
user_id	messages	INDEX
body/subject	messages	FULLTEXT
date_entered	messages	INDEX
user_id	users	PRIMARY
username	users	UNIQUE
pass/username	users	INDEX
email	users	UNIQUE



```

C:\WINDOWS\System32\cmd.exe - mysql -u root
mysql> ALTER TABLE forums ADD UNIQUE(name);
Query OK, 7 rows affected (0.17 sec)
Records: 7 Duplicates: 0 Warnings: 0
mysql>

```

Rysunek 6.26. Dla kolumny `name` dodano nowy indeks, który poprawi efektywność zapytań i zapobiegnie wprowadzaniu powtarzających się wartości

Aby dodać indeks do istniejącej tabeli:

- I. Załóż indeks na kolumnie `name` w tabeli `forums` (patrz rysunek 6.26).

```
ALTER TABLE forums ADD UNIQUE (name);
```

Tabela `forums` ma już założony indeks typu *klucz główny* na kolumnie `forum_id`. Ponieważ `name` również jest polem, do którego będą częste odwołania i którego wartość musi być unikatowa w każdym rekordzie, zakładam na nim indeks `UNIQUE`.

Modyfikowanie tabel

Polecenie `ALTER` służy przede wszystkim do modyfikowania struktury tabeli w bazie danych. Najczęściej oznacza to dodawanie, usuwanie bądź modyfikację kolumn, ale również dodawanie indeksów. Polecenia `ALTER` można również użyć do zmiany nazwy tabeli. Teoretycznie przy dobrym projekcie bazy jej struktura nie powinna stwarzać żadnych problemów. Jednak w praktyce często zdarza się wprowadzać w niej pewne zmiany. Składnia polecenia `ALTER` jest następująca:

```
ALTER TABLE nazwa_tabeli KLAUZULA;
```

Ponieważ klauzul, które można zastosować, jest bardzo wiele, w tabeli 6.9 wymieniłem tylko najczęściej stosowane. Pełną listę znajdziesz w podręczniku `MySQL-a`.

Tabela 6.9. Popularne warianty polecenia `ALTER` (gdzie *t* reprezentuje nazwę tabeli, *c* nazwę kolumny, a *i* nazwę indeksu. Pełną specyfikację polecenia znajdziesz w dokumentacji `MySQL-a`)

Klauzule, które można stosować w poleceniach `ALTER TABLE`

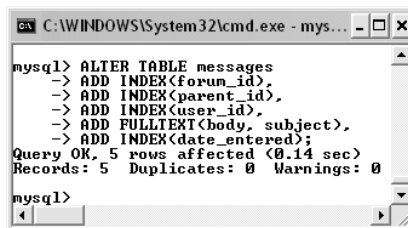
Klauzula	Sposób użycia	Znaczenie
<code>ADD COLUMN</code>	<code>ALTER TABLE t ADD COLUMN c TYP</code>	Dodaje nową kolumnę na końcu tabeli.
<code>CHANGE COLUMN</code>	<code>ALTER TABLE t CHANGE COLUMN c c TYP</code>	Pozwala zmienić typ danych i właściwości kolumny.
<code>DROP COLUMN</code>	<code>ALTER TABLE t DROP COLUMN c</code>	Usuwa kolumnę z tabeli razem z wszystkimi jej danymi.
<code>ADD INDEX</code>	<code>ALTER TABLE t ADD INDEX i (c)</code>	Zakłada nowy indeks na kolumnie <i>c</i> .
<code>DROP INDEX</code>	<code>ALTER TABLE t DROP INDEX i</code>	Usuwa istniejący indeks.
<code>RENAME AS</code>	<code>ALTER TABLE t RENAME AS nowa_t</code>	Zmienia nazwę tabeli.

2. Załóż indeksy dla tabeli messages (patrz rysunek 6.27).

```
ALTER TABLE messages
ADD INDEX(forum_id),
ADD INDEX(parent_id),
ADD INDEX(user_id),
ADD FULLTEXT(body, subject),
ADD INDEX(date_entered);
```

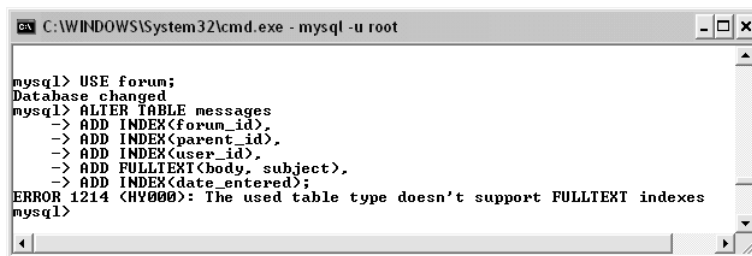
Ta tabela będzie mieć najwięcej indeksów, ponieważ jest najważniejszą tabelą w bazie i zawiera trzy klucze obce (forum_id, parent_id i user_id), dla których należy założyć osobne indeksy. Dodatkowo pola body i subject otrzymują indeks FULLTEXT używany podczas wyszukiwania typu FULLTEXT w dalszej części tego rozdziału. Również kolumna date_entered otrzymuje indeks, ponieważ będzie używana przez klauzule ORDER BY (do sortowania wiadomości po dacie).

Jeśli podczas wykonywania tego zapytania pojawi się komunikat o błędzie stwierdzający, że typ tabeli nie obsługuje indeksów FULLTEXT (patrz rysunek 6.28), na razie opuść ten wiersz zapytania i sprawdź w następnym podrozdziale, w jaki sposób zmienić typ tabeli.



```
C:\WINDOWS\System32\cmd.exe - mys...
mysql> ALTER TABLE messages
-> ADD INDEX(forum_id),
-> ADD INDEX(parent_id),
-> ADD INDEX(user_id),
-> ADD FULLTEXT(body, subject),
-> ADD INDEX(date_entered);
Query OK, 5 rows affected (0.14 sec)
Records: 5 Duplicates: 0 Warnings: 0
mysql>
```

Rysunek 6.27. Do tabeli messages dodano kilka indeksów naraz. MySQL raportuje wykonanie operacji oraz podaje liczbę rekordów, których dotyczyła (powinno to być liczba wszystkich rekordów tabeli)



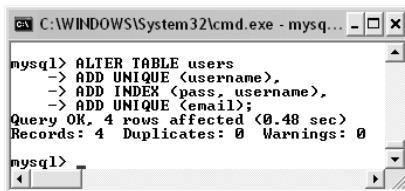
```
C:\WINDOWS\System32\cmd.exe - mysql -u root
mysql> USE forum;
Database changed
mysql> ALTER TABLE messages
-> ADD INDEX(forum_id),
-> ADD INDEX(parent_id),
-> ADD INDEX(user_id),
-> ADD FULLTEXT(body, subject),
-> ADD INDEX(date_entered);
ERROR 1214 (HY000): The used table type doesn't support FULLTEXT indexes
mysql>
```

Rysunek 6.28. Indeksów FULLTEXT nie można stosować dla wszystkich typów tabel. Jeśli napotkasz ten komunikat o błędzie, to rozwiązanie znajdziesz w tym rozdziale w części „Stosowanie różnych typów tabel”

3. Dodaj indeksy do tabeli users (patrz rysunek 6.29).

```
ALTER TABLE users
ADD UNIQUE (username),
ADD INDEX (pass, username),
ADD UNIQUE (email);
```

Tabela users będzie mieć dwa indeksy UNIQUE oraz jeden indeks założony na dwóch kolumnach. Indeksy UNIQUE zastosowałem w tym przypadku, ponieważ chcę zapobiec możliwości zarejestrowania dwóch użytkowników o tej samej nazwie lub wielokrotnego zarejestrowania użytkownika o tym samym adresie e-mail.



```
C:\WINDOWS\System32\cmd.exe - mysql...
mysql> ALTER TABLE users
-> ADD UNIQUE (username),
-> ADD INDEX (pass, username),
-> ADD UNIQUE (email);
Query OK, 4 rows affected (0.48 sec)
Records: 4 Duplicates: 0 Warnings: 0
mysql>
```

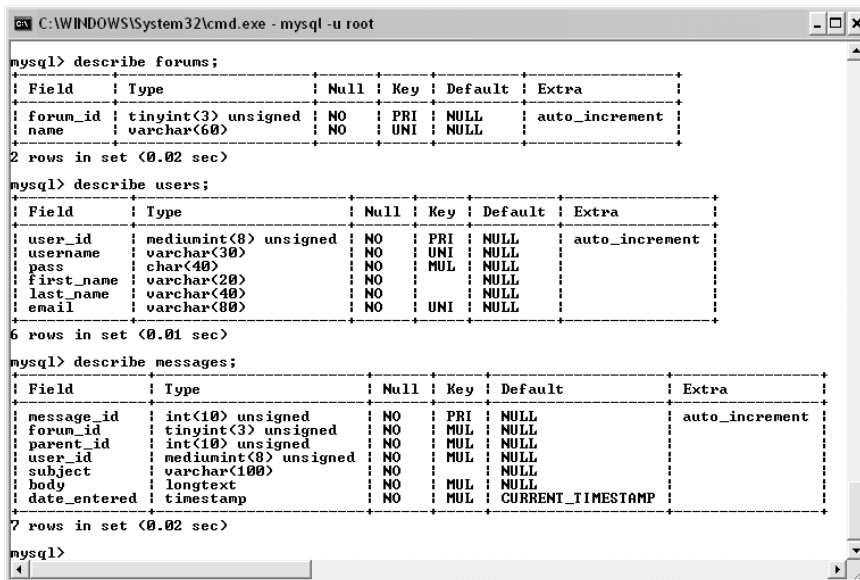
Rysunek 6.29. Indeksy zostały również dodane do trzeciej tabeli

Indeks założony na kolumnach pass i username ma za zadanie poprawić efektywność zapytań wykonywanych podczas uwierzytelniania użytkownika, gdy kombinacja tych dwóch kolumn będzie używana w wyrażeniu warunkowym WHERE.

4. Obejrzyj bieżącą strukturę wszystkich tabel (patrz rysunek 6.30).

```
DESCRIBE forums;
DESCRIBE messages;
DESCRIBE users;
```

Polecenie DESCRIBE języka SQL zwraca informacje o nazwach kolumn występujących w tabeli i w ich typach, a także o kolejności, w jakiej one występują i o pozakładanych indeksach. Podaje ono także, czy dane pole może przyjmować wartość NULL, czy określono dla niego jakąś wartość domyślną itd.



```
C:\WINDOWS\System32\cmd.exe - mysql -u root
mysql> describe forums;
+----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+----+-----+-----+-----+-----+-----+
| forum_id | tinyint(3) unsigned | NO | PRI | NULL | auto_increment |
| name | varchar(60) | NO | UNI | NULL | |
+----+-----+-----+-----+-----+-----+
2 rows in set (0.02 sec)

mysql> describe users;
+----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+----+-----+-----+-----+-----+-----+
| user_id | mediumint(8) unsigned | NO | PRI | NULL | auto_increment |
| username | varchar(30) | NO | UNI | NULL | |
| pass | char(40) | NO | MUL | NULL | |
| first_name | varchar(20) | NO | | NULL | |
| last_name | varchar(40) | NO | | NULL | |
| email | varchar(80) | NO | UNI | NULL | |
+----+-----+-----+-----+-----+-----+
6 rows in set (0.01 sec)

mysql> describe messages;
+----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+----+-----+-----+-----+-----+-----+
| message_id | int(10) unsigned | NO | PRI | NULL | auto_increment |
| forum_id | tinyint(3) unsigned | NO | MUL | NULL | |
| parent_id | int(10) unsigned | NO | MUL | NULL | |
| user_id | mediumint(8) unsigned | NO | MUL | NULL | |
| subject | varchar(100) | NO | | NULL | |
| body | longtext | NO | MUL | NULL | |
| date_entered | timestamp | NO | MUL | CURRENT_TIMESTAMP | |
+----+-----+-----+-----+-----+-----+
7 rows in set (0.02 sec)

mysql>
```

Rysunek 6.30. Aby zobaczyć szczegółowe informacje na temat struktury tabeli, użyj polecenia DESCRIBE. Kolumna Key informuje o indeksach

Wskazówki

- Próba założenia indeksu UNIQUE na kolumnie zawierającej duplikaty spowoduje błąd, a indeks nie zostanie utworzony.
- Tworząc indeks, możesz nadać mu nazwę.

```
ALTER TABLE tabela
  ADD INDEX nazwa_indeksu (kolumna);
```

Jeżeli tego nie zrobisz, otrzyma on nazwę kolumny, na której go zakładasz.

- Słowo COLUMN w większości poleceń ALTER jest opcjonalne.
- Załóżmy, że stworzyłeś indeks dla kilku kolumn:

```
ALTER TABLE tabela
  ADD INDEX (ko11, ko12, ko13)
```

Powstały indeks umożliwi efektywne przeszukiwanie kolumny *ko11*, kombinacji kolumn *ko11* i *ko12* oraz kombinacji wszystkich trzech podanych kolumn. Nie zwiększa efektywności przeszukiwania kolumny *ko12* i *ko13* lub ich kombinacji.

Stosowanie różnych typów tabeli

MySQL obsługuje kilka różnych typów tabeli (typ tabeli nazywany bywa również *silnikiem przechowywania*). Każdy z tych typów ma inne właściwości, odrębne ograniczenia (dotyczące ilości przechowywanych danych) i charakteryzuje się inną efektywnością działania w określonych sytuacjach. Jednak interakcje użytkownika (w sensie wykonywania zapytań) z różnymi typami tabel są spójne.

Najważniejszym typem tabeli jest MyISAM. Jest on domyślnym typem tabeli na wszystkich platformach oprócz Windows. Tabele tego typu są najodpowiedniejsze dla większości aplikacji i umożliwiają szybkie wykonywanie poleceń SELECT oraz INSERT. Ich podstawową wadą jest to, że nie obsługują transakcji.

Kolejnym typem tabeli pod względem popularności zastosowań jest InnoDB, który jest domyślnym typem tabeli MySQL-a w systemie Windows. Tabele InnoDB obsługują transakcje i umożliwiają szybkie wykonywanie poleceń UPDATE. Jednak silnik InnoDB jest w ogólnym przypadku wolniejszy niż MyISAM i wymaga większej przestrzeni dyskowej serwera. Tabele typu InnoDB nie obsługują również indeksów typu FULLTEXT (i dlatego mogłeś napotkać błąd przedstawiony na rysunku 6.28, jeśli używasz Windows).

Aby określić, którego silnika chcemy używać, na końcu polecenia CREATE dodajemy specjalną klauzulę:

```
CREATE TABLE tabela (
  kolumna1 TYPKOLUMNY,
  kolumna2 TYPKOLUMNY...
) ENGINE = INNODB
```

Jeśli tworząc tabele, nie podamy ich typu, to MySQL użyje typu domyślnego.

Typ istniejącej tabeli możemy zmienić za pomocą polecenia ALTER:

```
ALTER TABLE tabela ENGINE = MYISAM
```

Ponieważ następny przykład w tym rozdziale będzie wymagał tabeli typu MyISAM, to pokażę najpierw kroki konieczne do skonfigurowania odpowiedniego typu tabeli na przykładzie tabeli messages. W kilku pierwszych krokach dowiesz się, jak sprawdzić wykorzystywany typ silnika, (ponieważ zmiana typu tabeli może nie być konieczna).

Aby zmienić typ tabeli:

1. Obejrzyj aktualną informację o tabeli (patrz rysunek 6.31).

```
SHOW TABLE STATUS;
```

Polecenie `SHOW TABLE STATUS` zwraca wiele przydatnych informacji o tabelach bazy danych. Informacje te odczytuje się niewygodnie, ponieważ mają one postać szerokiej tabeli zajmującej wiele wierszy. W każdym wierszu znajduje się najpierw nazwa tabeli, a następnie jej typ. Zwykle jest nim MyISAM lub InnoDB.

2. Jeśli to konieczne, zmień typ tabeli `messages` na MyISAM (rysunek 6.32).

```
ALTER TABLE messages ENGINE=MYISAM;
```

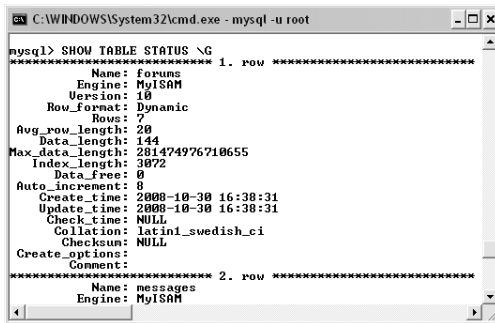
Jeśli w punkcie 1. okazało się, że typ tabeli jest inny niż MyISAM, to zmień go za pomocą powyższego polecenia (nie musisz zwracać uwagi na małe i duże litery w typie tabeli). Jeśli wykonałeś domyślną instalację i konfigurację MySQL-a, to zmiana typu tabeli nie będzie potrzebna w systemie Mac OS X, ale musisz jej dokonać w systemie Windows.

```
C:\WINDOWS\System32\cmd.exe - mysql -u root
mysql> SHOW TABLE STATUS;
+-----+-----+-----+-----+-----+-----+
| Name      | Engine | Version | Row_format | Rows | Avg_row_length |
+-----+-----+-----+-----+-----+-----+
| forums    | MyISAM | 10      | Dynamic    | 7    | 20              |
| messages  | MyISAM | 10      | Dynamic    | 5    | 109             |
| users     | MyISAM | 10      | Dynamic    | 4    | 71              |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.02 sec)
mysql>
```

Rysunek 6.31. Zanim zmienisz typ tabeli, sprawdź go za pomocą polecenia `SHOW TABLE STATUS`

```
C:\WINDOWS\System32\cmd.exe - mysql -u root
mysql> ALTER TABLE messages ENGINE = MYISAM;
Query OK, 5 rows affected (0.50 sec)
Records: 5 Duplicates: 0 Warnings: 0
mysql>
```

Rysunek 6.32. Zmieniłem typ tabeli (lub silnik przechowywania) za pomocą polecenia `ALTER`



```

C:\WINDOWS\system32\cmd.exe - mysql -u root
mysql> SHOW TABLE STATUS \G
***** 1. row *****
      Name: forums
      Engine: MyISAM
      Version: 10
      Row_format: Dynamic
      Rows: 7
      Avg_row_length: 20
      Data_length: 144
      Max_data_length: 281474976710655
      Index_length: 3072
      Data_free: 0
      Auto_increment: 0
      Create_time: 2008-10-30 16:38:31
      Update_time: 2008-10-30 16:38:31
      Check_time: NULL
      Collation: latin1_swedish_ci
      Checksum: NULL
      Create_options:
      Comment:
***** 2. row *****
      Name: messages
      Engine: MyISAM

```

Rysunek 6.33. Aby uzyskać bardziej czytelną postać wyników zapytania, użyłem znacznika \G

- Możesz sprawdzić zmianę typu tabeli, wykonując ponownie polecenie SHOW TABLE STATUS.

Wskazówki

- Aby ułatwić odczytanie wyników dowolnego zapytania, możesz dodać w kliencie mySQL parametr \G na końcu zapytania (rysunek 6.33):

```
SHOW TABLE STATUS \G
```

Znacznik ten informuje, że tabela wyników powinna zostać wyświetlona pionowo zamiast poziomo. Zwróć uwagę, że nie musisz w tym przypadku użyć średnika kończącego polecenie SQL, ponieważ funkcję tę spełnia znacznik \G.

- W tej samej bazie danych mogą występować różne typy tabel. W zależności od domyślnego typu tabel w Twojej instalacji MySQL-a, stwierdzenie to może już być prawdziwe dla bazy danych *forum*. To samo możesz zaobserwować w przypadku bazy danych *ecommerce*, w której tabele klientów i produktów są typu MyISAM, ale tabela zamówień jest typu InnoDB (aby umożliwić stosowanie transakcji).

Wyszukiwanie FULLTEXT

W rozdziale 5. przedstawiłem słowo kluczowe LIKE umożliwiające proste porównania łańcuchów, na przykład:

```
SELECT * FROM users
WHERE last_name LIKE 'Kowalsk%';
```

W ten sposób nie wykonamy jednak wyszukiwań z zastosowaniem wielu słów. Aby stało się to możliwe, wprowadzono wyszukiwania FULLTEXT.

Wyszukiwania FULLTEXT wymagają obecności indeksu FULLTEXT, który można założyć jedynie na tabelach typu MyISAM. W następnych przykładach będę używać tabeli messages bazy forum. Jeśli utworzona przez Ciebie tabela messages nie jest typu MyISAM i (lub) nie ma indeksu FULLTEXT założonego na kolumnach body i subject, wykonaj kroki podane na kilku poprzednich stronach, aby to zmienić.

Wskazówki

- Wstawianie rekordów do tabel, na których założono indeks FULLTEXT, może być znacznie wolniejsze ze względu na skomplikowaną naturę tego indeksu.
- Indeks FULLTEXT możemy zakładać na wielu kolumnach, jeśli wszystkie mają być przeszukiwane.
- Wyszukiwania FULLTEXT mogą służyć do implementacji prostych usług wyszukiwania. Ponieważ jednak indeks FULLTEXT stosuje się do jednej tabeli, to usługi wyszukiwania informacji w wielu tabelach wymagają zastosowania bardziej zaawansowanych rozwiązań.

Realizacja podstawowego wyszukiwania FULLTEXT

Po założeniu indeksu FULLTEXT możesz wydawać zapytania przy użyciu funkcji MATCH i AGAINST w wyrażeniach warunkowych WHERE:

```
SELECT * FROM tabela
WHERE MATCH (kolumna)
AGAINST ('poszukiwane_słowa');
```

MySQL zwróci pasujące rekordy począwszy od spełniających warunek wyszukiwania w największym stopniu. Podczas wyszukiwania stosowane są następujące reguły:

- ◆ Łańcuchy rozbijane są na poszczególne słowa.
- ◆ Słowa krótsze niż 4-znakowe są ignorowane.
- ◆ Słowa często używane są ignorowane.
- ◆ Jeśli ponad połowa rekordów spełnia warunek wyszukiwania, to nie jest zwracany żaden rekord.

Zwłaszcza ostatnia reguła jest zaskakująca dla wielu użytkowników, którzy rozpoczynają dopiero przygodę z wyszukiwaniami FULLTEXT i dziwią się, dlaczego nie zwracają one żadnych wyników. Jeśli wypełnisz tabelę zbyt małą liczbą rekordów, to MySQL nie zwróci właściwych wyników.

Aby wykonać wyszukiwanie FULLTEXT:

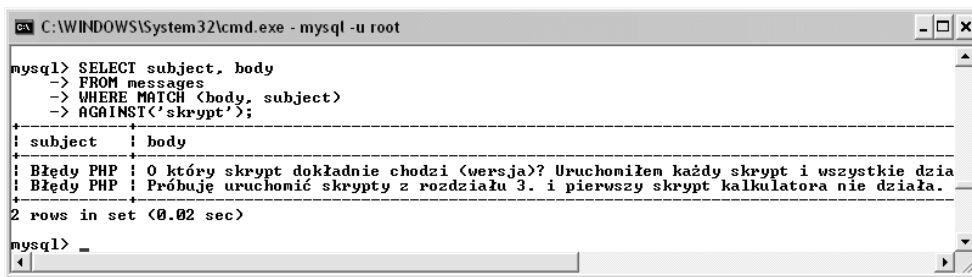
1. Wypełnij tabelę messages dużą ilością danych — zwłaszcza w polu body.

Możesz użyć w tym celu poleceń INSERT ściągniętych z serwera ftp.

2. Wykonaj proste wyszukiwanie FULLTEXT dla słowa *skrypt* (patrz rysunek 6.34).

```
SELECT subject, body FROM messages
WHERE MATCH (body, subject)
AGAINST ('skrypt');
```

Ten prosty przykład zwróci wynik pod warunkiem, że co najmniej jeden rekord i mniej niż połowa rekordów tabeli messages zawiera słowo *tabela* w polu body lub subject. Zwróc uwagę, że kolumny, do których odwołuje się MATCH, muszą być tymi samymi kolumnami, jakie podałeś, tworząc indeks FULLTEXT. W tym przykładzie mogłeś zatem użyć albo body, subject, albo subject, body, ale nie samych body lub subject (patrz rysunek 6.35).



```
C:\WINDOWS\System32\cmd.exe - mysql -u root

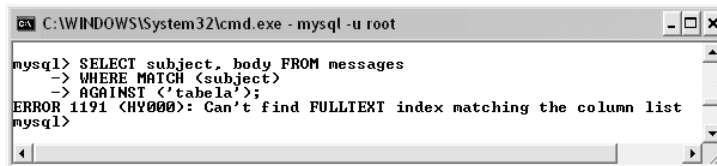
mysql> SELECT subject, body
-> FROM messages
-> WHERE MATCH (body, subject)
-> AGAINST ('skrypt');
```

subject	body
Błędy PHP	0 który skrypt dokładnie chodzi (wersja)? Uruchoniłem każdy skrypt i wszystkie dzia
Błędy PHP	Próbuję uruchomić skrypty z rozdziału 3. i pierwszy skrypt kalkulatora nie działa.

```
2 rows in set (0.02 sec)

mysql> _
```

Rysunek 6.34. Podstawowe wyszukiwanie FULLTEXT



```
C:\WINDOWS\System32\cmd.exe - mysql -u root

mysql> SELECT subject, body FROM messages
-> WHERE MATCH (subject)
-> AGAINST ('tabela');
```

```
ERROR 1191 (HY000): Can't find FULLTEXT index matching the column list
mysql>
```

Rysunek 6.35. Zapytanie FULLTEXT możesz wykonać jedynie na tej samej kolumnie lub kombinacji kolumn, dla której utworzyłeś indeks FULLTEXT. W tym przypadku zapytanie zakończy się błędem, mimo że kombinacja kolumn body i subject ma indeks FULLTEXT

3. Wykonaj to samo wyszukiwanie FULLTEXT, pokazując dodatkowo stopień spełnienia warunku przez poszczególne rekordy (patrz rysunek 6.36).

```
SELECT subject, body,
       MATCH (body, subject)
AGAINST ('skrypt') AS R
FROM messages
WHERE MATCH (body, subject)
AGAINST ('skrypt');
```

Jeśli użyjesz tego samego wyrażenia MATCH...AGAINST jako wybieranej wartości, to pokazany zostanie również stopień spełnienia warunku przez rekordy.

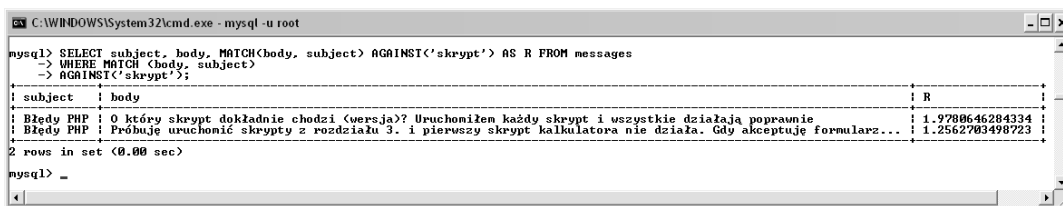
4. Wykonaj wyszukiwanie FULLTEXT stosując kilka słów (rysunek 6.37).

```
SELECT subject, body
FROM messages
WHERE MATCH (body, subject)
AGAINST ('projekt formularz');
```

W tym przypadku rekord spełni warunek wyszukiwania, jeśli pole subject lub body będzie zawierać jedno z podanych słów. Rekord zawierający oba słowa uzyska wyższy stopień spełnienia warunku.

Wskazówki

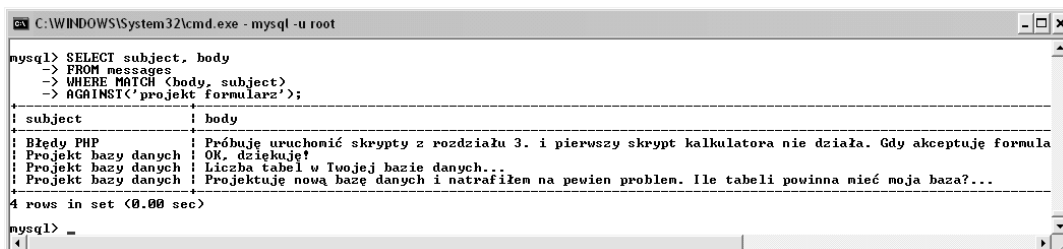
- Pamiętaj, że jeśli wyszukiwanie FULLTEXT nie zwróci żadnych rekordów, to albo żaden rekord nie spełnia warunku wyszukiwania, albo spełnia go ponad połowa rekordów.
- Dla uproszczenia wszystkie zapytania przedstawione w tym podrozdziale zapisałem jako najprostsze polecenia SELECT. Wyszukiwania FULLTEXT można też używać w bardziej skomplikowanych zapytaniach czy złączeniach.
- MySQL zawiera w kodzie źródłowym listę kilkuset słów pospolitych, które są ignorowane w wyszukiwaniach FULLTEXT.
- Minimalną długość wyszukiwanego słowa (wynoszącą domyślnie 4 znaki) można konfigurować.
- Domyślnie wyszukiwania FULLTEXT nie rozróżniają małych i wielkich liter.



```
C:\WINDOWS\System32\cmd.exe - mysql -u root
mysql> SELECT subject, body, MATCH (body, subject) AGAINST ('skrypt') AS R FROM messages
-> WHERE MATCH (body, subject)
-> AGAINST ('skrypt');
+-----+-----+-----+
| subject | body | R |
+-----+-----+-----+
| Błędy PHP | 0 który skrypt dokładnie chodzi (wersja)? Uruchomiłem każdy skrypt i wszystkie działają poprawnie | 1.9780646284334 |
| Błędy PHP | Próbuje uruchomić skrypty z rozdziału 3. i pierwszy skrypt kalkulatora nie działa. Gdy akceptuję formularz... | 1.2562703498723 |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> _
```

Rysunek 6.36. Możesz również zobaczyć stopień spełniania warunku wyszukiwania przez poszczególne rekordy



```
C:\WINDOWS\System32\cmd.exe - mysql -u root
mysql> SELECT subject, body
-> FROM messages
-> WHERE MATCH (body, subject)
-> AGAINST ('projekt formularz');
+-----+-----+
| subject | body |
+-----+-----+
| Błędy PHP | Próbuje uruchomić skrypty z rozdziału 3. i pierwszy skrypt kalkulatora nie działa. Gdy akceptuję formuła |
| Projekt bazy danych | OK, dziękuję! |
| Projekt bazy danych | Liczba tabel w Twojej bazie danych... |
| Projekt bazy danych | Projektuję nową bazę danych i natrafiłem na pewien problem. Ile tabeli powinna mieć moja baza?... |
+-----+-----+
4 rows in set (0.00 sec)

mysql> _
```

Rysunek 6.37. Wyszukiwanie FULLTEXT można także stosować dla wielu słów

Wyszukiwania FULLTEXT w trybie Boolean

Bardziej zaawansowane wyszukiwania FULLTEXT można wykonywać, korzystając z trybu Boolean. W trybie tym używamy wyrażenia IN BOOLEAN MODE wewnątrz klauzuli AGAINST:

```
SELECT * FROM tabela WHERE
  MATCH(kolumny)
  AGAINST('poszukiwane_słowa'
  IN BOOLEAN MODE)
```

W trybie Boolean możemy stosować szereg operatorów (tabela 6.10) określających sposób traktowania każdego wyszukiwanego słowa:

```
SELECT * FROM tabela WHERE
  MATCH(kolumny)
  AGAINST('+baza -mysql' IN BOOLEAN MODE)
```

W tym przykładzie rekord zostanie wybrany, jeśli zawiera słowo *baza* i nie zawiera słowa *mysql*. Jako słabszą formę operatora reprezentowanego przez znak minus (-) możemy stosować tyldę (~). Oznacza ona, że rekord może zawierać dane słowo, ale wtedy w mniejszym stopniu spełnia warunek wyszukiwania.

Tabela 6.10. Operatory umożliwiające precyzyjniejsze wyszukiwanie FULLTEXT

Operatory dostępne w trybie Boolean	
Operator	Znaczenie
+	Rekord musi zawierać słowo poprzedzone tym operatorem.
-	Rekord nie może zawierać słowa poprzedzonego tym operatorem.
~	Słowo poprzedzone tym operatorem obniża wartość rekordu jako wyniku wyszukiwania.
*	Maska wyszukiwania.
<	Zmniejsza znaczenie słowa.
>	Zwiększa znaczenie słowa.
" "	Rekord musi zawierać frazę umieszczoną w znakach cudzysłowu.
()	Tworzy podwyrażenie.

Znak maski (*) umożliwia wyszukiwanie różnych wariantów słowa. Na przykład zastosowanie go w słowie *bezpiecz** spowoduje wyszukanie rekordów zawierających słowa *bezpieczny*, *bezpieczeństwo* i tym podobnych. Dwa kolejne operatory umożliwiają określenie, że dane słowo jest mniej (<) lub bardziej (>) ważne. Znaki cudzysłowu pozwalają wyszukiwać całe frazy, a nawiasy — tworzyć podwyrażenia.

Poniższe zapytanie wyszuka rekordy zawierające frazę *serwer WWW* oraz słowo *html*. Obecność słowa *JavaScript* obniży wartość rekordu jako wyniku wyszukiwania.

```
SELECT * FROM tabela WHERE
  MATCH (kolumny) AGAINST('>"serwer WWW" +html
  ~JavaScript' IN BOOLEAN MODE)
```

W trybie Boolean pojawiają się następujące różnice w sposobie działania wyszukiwania FULLTEXT:

- ◆ Jeśli słowo nie jest poprzedzone żadnym operatorem, to jest opcjonalne, ale podnosi wartość rekordu, który je zawiera.
- ◆ Wyniki są zwracane nawet wtedy, gdy ponad połowa rekordów spełnia warunek wyszukiwania.
- ◆ Wyniki nie są automatycznie sortowane na podstawie stopnia, w jakim spełniają warunek wyszukiwania.

Z uwagi na tę ostatnią właściwość można samodzielnie posortować rekordy według stopnia w jakim spełniają warunek wyszukiwania, co zaprezentuję w kilku następnych krokach. W przypadku wyszukiwań w trybie Boolean nadal obowiązuje zasada, że słowa krótsze niż 4-znakowe są ignorowane. Nie zmienia tego użycie operatora +, na przykład słowo *+php* zostanie zignorowane.

Aby wykonać wyszukiwanie FULLTEXT w trybie Boolean:

1. Wykonaj proste wyszukiwanie FULLTEXT różnych przypadków słowa *baza* (*bazy*, *bazie* itp.) (patrz rysunek 6.38).

```
SELECT subject, body FROM
messages WHERE MATCH(body, subject)
AGAINST('baz*' IN BOOLEAN MODE) \G
```

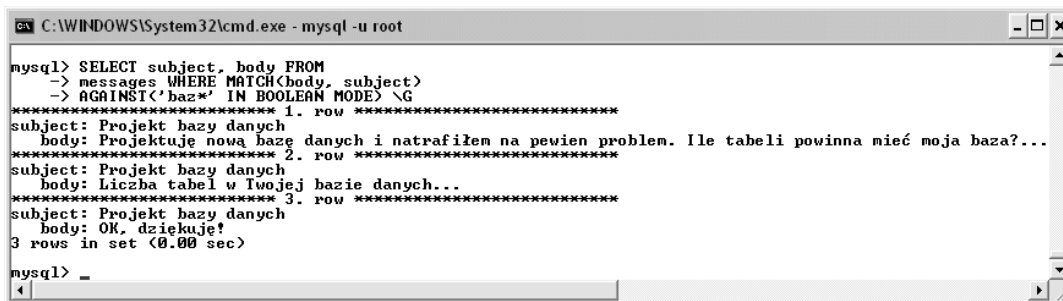
Słowo *baza* może pojawić się w wiadomościach forum w wielu różnych przypadkach takich jak *bazy* lub *bazie*. Powyższe zapytanie wyszuka rekordy zawierające te przypadki *dzięki zastosowaniu znaku maski* (*).

Dodatkowo zastosowałem znacznik \G, aby łatwiej przeanalizować wyniki.

2. Wykonaj zapytanie wyszukujące wiadomości dotyczące tabel ze szczególnym uwzględnieniem zawierających termin normalizacja (patrz rysunek 6.39).

```
SELECT subject, body FROM messages
WHERE MATCH(body, subject)
AGAINST ('>formularz* +skrypt*'
IN BOOLEAN MODE)\G
```

Zapytanie to najpierw wyszukuje wszystkie rekordy zawierające słowo *skrypt** (*skryptu*, *skrypty*, ...) i *formularz** (*formularz*, *formularza*, *formularzy*, ...). Obecność terminu *skrypt** jest wymagana (na co wskazuje operator +), natomiast wyróżnione zostają rekordy zawierające termin *formularz** (na co wskazuje operator >).

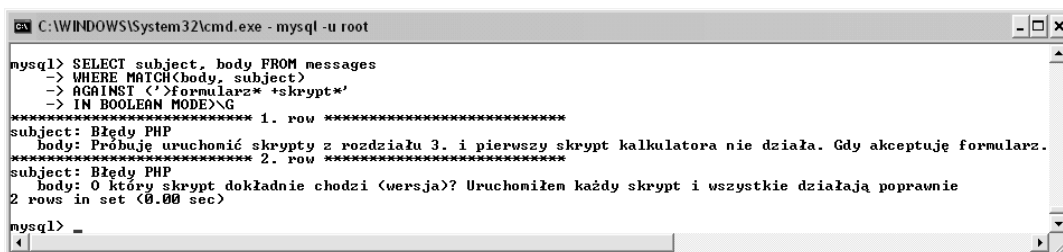


```
C:\WINDOWS\System32\cmd.exe - mysql -u root

mysql> SELECT subject, body FROM
-> messages WHERE MATCH(body, subject)
-> AGAINST('baz*' IN BOOLEAN MODE) \G
***** 1. row *****
subject: Projekt bazy danych
body: Projektuję nową bazę danych i natrafiłem na pewien problem. Ile tabeli powinna mieć moja baza?...
***** 2. row *****
subject: Projekt bazy danych
body: Liczba tabel w Twojej bazie danych...
***** 3. row *****
subject: Projekt bazy danych
body: OK, dziękuję!
3 rows in set (0.00 sec)

mysql> _
```

Rysunek 6.38. Proste wyszukiwanie FULLTEXT w trybie BOOLEAN



```
C:\WINDOWS\System32\cmd.exe - mysql -u root

mysql> SELECT subject, body FROM messages
-> WHERE MATCH(body, subject)
-> AGAINST ('>formularz* +skrypt*'
-> IN BOOLEAN MODE)\G
***** 1. row *****
subject: Błędy PHP
body: Próbuje uruchomić skrypty z rozdziału 3. i pierwszy skrypt kalkulatora nie działa. Gdy akceptuję formularz.
***** 2. row *****
subject: Błędy PHP
body: 0 który skrypt dokładnie chodzi (wersja)? Uruchoniłem każdy skrypt i wszystkie działają poprawnie
2 rows in set (0.00 sec)

mysql> _
```

Rysunek 6.39. To wyszukiwanie dotyczy wariantów dwóch różnych słów, z których jedno ma wyższy priorytet



Optymalizacja bazy danych

Efektywność działania bazy danych zależy głównie od jej struktury i indeksów. Tworząc bazę, powinieneś spróbować:

- ◆ Wybrać najlepszy silnik przechowywania.
- ◆ Używać jak najmniej pojemnego typu danych dla każdej kolumny.
- ◆ Definiować kolumny jako NOT NULL tam, gdzie to możliwe.
- ◆ Używać wartości całkowitych jako kluczy głównych.
- ◆ Uważnie definiować indeksy, wybierając odpowiedni typ oraz stosując je dla właściwych kolumn.
- ◆ Ograniczać indeksy do pewnej liczby znaków, jeśli to możliwe.

Oprócz tych wskazówek warto zastosować dwie proste techniki optymalizacji baz danych. Jednym ze sposobów poprawy efektywności działania MySQL-a jest wykonanie polecenia `OPTIMIZE` dla tabel. Polecenie to usuwa wolne przestrzenie i przywraca wysoką efektywność działania tabel.

```
OPTIMIZE TABLE tabela;
```

Wykonanie tego polecenia jest szczególnie zalecane po modyfikacji tabeli za pomocą polecenia `ALTER`.

Aby zwiększyć wydajność wykonywanego zapytania, warto zrozumieć, w jaki sposób serwer MySQL-a je przetwarza. W tym celu należy posłużyć się słowem kluczowym języka SQL o nazwie `EXPLAIN`. Informacje na temat działania zapytań znajdziesz w podręczniku MySQL-a.

Wskazówki

- W MySQL 5.1.7 wprowadzono jeszcze jeden tryb wyszukiwań `FULLTEXT`: tryb języka naturalnego. Jeśli nie wybierzesz innego trybu (np. `Boolean`), to jest on trybem domyślnym.
- Modyfikator `WITH QUERY EXPANSION` pozwala zwiększyć liczbę zwracanych wyników. Zapytania z tym modyfikatorem wykonują w rzeczywistości dwa wyszukiwania, ale zwracają jeden zbiór wyników. Drugie wyszukiwanie bazuje na terminach wyszukanych dodatkowo w najlepszych wynikach pierwszego wyszukiwania. Pozwala to znaleźć dodatkowe rekordy, ale nie zawsze odpowiadają one początkowym kryteriom wyszukiwania.

Wykonywanie transakcji

Transakcja bazodanowa to seria zapytań wykonywanych podczas jednej sesji. Przypuśćmy, że wstawiamy rekord do jednej tabeli, inny rekord do drugiej tabeli, a potem wykonujemy aktualizację. Bez transakcji każda operacja jest realizowana natychmiast i nie można jej cofnąć. Jeśli użyjemy transakcji, będziemy mogli ustawić punkt początkowy i końcowy, a następnie zatwierdzić lub wycofać wszystkie zapytania (jeśli, na przykład, zawiedzie jedno zapytanie, będzie można wycofać wszystkie).

Transakcje są często potrzebne w interakcjach finansowych, nawet tak podstawowych jak przelew 100 złotych z jednego konta na drugie. Proces ten wydaje się prosty, ale w rzeczywistości składa się z kilku etapów:

- ◆ Potwierdzenia, że na pierwszym koncie znajduje się przynajmniej 100 złotych.
- ◆ Zmniejszenia stanu pierwszego konta o 100 złotych.
- ◆ Zwiększenia stanu drugiego konta o 100 złotych.
- ◆ Sprawdzenia, czy udało się zwiększyć stan drugiego konta.

Jeśli którakolwiek z tych operacji się nie powiedzie, należy cofnąć je wszystkie. Jeśli na przykład nie uda się umieścić pieniędzy na drugim koncie, powinny one wrócić na pierwsze. Odbywa się to do momentu, w którym uda się przeprowadzić całą transakcję.

Aby korzystać z transakcji w MySQL-u, trzeba posługiwać się tabelami InnoDB (lub silnikiem tego typu). W celu rozpoczęcia nowej transakcji w kliencie mysql należy wpisać:

```
START TRANSACTION;
```

Na zakończenie transakcji należy wydać instrukcję COMMIT, aby zatwierdzić wszystkie zapytania, albo ROLLBACK, aby je wycofać.

Po zatwierdzeniu lub wycofaniu zapytań transakcja jest uznawana za zakończoną i MySQL wraca do trybu *automatycznego zatwierdzania* (ang. *autocommit*). Oznacza to, że wszystkie zapytania są natychmiast realizowane. Aby rozpocząć następną transakcję, wystarczy ponownie wpisać START TRANSACTION.

Trzeba wiedzieć, że niektórych typów transakcji nie da się wycofać; dotyczy to w szczególności zapytań, które tworzą, modyfikują, opróżniają lub usuwają tabele albo tworzą lub usuwają bazy danych. Co więcej, wykonanie takiego zapytania skutkuje zatwierdzeniem i zakończeniem bieżącej transakcji.

Powinieneś też pamiętać, że transakcje są specyficzne dla każdego połączenia. Zatem jeden użytkownik klienta mysql dokonuje innej transakcji niż użytkownik drugiego klienta, a obie te transakcje są niezależne od realizowanych przez skrypt PHP.

To powiedziawszy, zaprezentuję bardzo prosty przykład użycia transakcji w kliencie mysql. W rozdziale 17. pokażę, jak realizować transakcje z wykorzystaniem skryptów PHP.

**Aby wykonać transakcję,
należy wykonać poniższe czynności:**

1. Uruchomić klienta mysql i wybrać bazę danych *test*.

Ponieważ jest to tylko przykład, wykorzystam hipotetyczną bazę danych *test*.
2. Utworzyć nową tabelę *accounts* (patrz rysunek 6.40).

```
CREATE TABLE accounts (
  id INT UNSIGNED NOT NULL
  ↳AUTO_INCREMENT,
  name VARCHAR(40) NOT NULL,
  balance DECIMAL(10,2) NOT NULL
  ↳DEFAULT 0.0,
  PRIMARY KEY (id))
ENGINE=InnoDB;
```

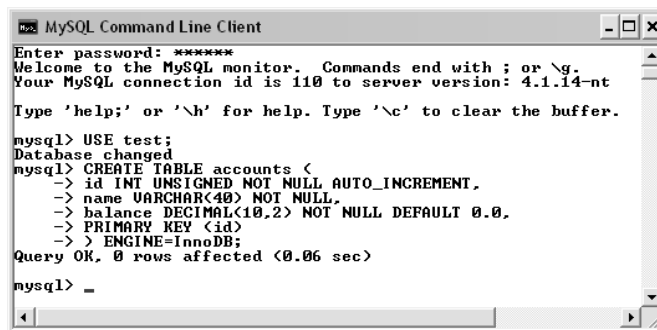
Oczywiście trudno uznać to za kompletny projekt tabeli czy bazy danych. Zasady normalizacji nakazywałyby rozdzielić imię i nazwisko użytkownika na dwie kolumny, a nawet umieścić je w innej tabeli. Jednakże tabela ta w zupełności wystarczy do naszych celów.

Najważniejszym aspektem definicji tabeli jest określenie mechanizmu InnoDB, który obsługuje transakcje.

3. Wypełnić tabelę danymi.

```
INSERT INTO accounts (name, balance)
VALUES ('Beata Piechota', 5460.30),
('Dawid Wójcik', 909325.24),
('Ilona Machnik', 892.00);
```

Można użyć dowolnie wybranych nazwisk i wartości. Należy zauważyć, że MySQL automatycznie zatwierdzi to zapytanie, ponieważ nie została jeszcze rozpoczęta żadna transakcja.



```
MySQL Command Line Client
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 110 to server version: 4.1.14-nt
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> USE test;
Database changed
mysql> CREATE TABLE accounts (
-> id INT UNSIGNED NOT NULL AUTO_INCREMENT,
-> name VARCHAR(40) NOT NULL,
-> balance DECIMAL(10,2) NOT NULL DEFAULT 0.0,
-> PRIMARY KEY (id)
-> ) ENGINE=InnoDB;
Query OK, 0 rows affected (0.06 sec)

mysql> _
```

Rysunek 6.40. Nowa tabela utworzona w bazie danych *test* posłuży do zademonstrowania transakcji

4. Rozpocząć transakcję i pobrać bieżącą zawartość tabeli (patrz rysunek 6.41).

```
START TRANSACTION;
SELECT * FROM accounts;
```

5. Podjąć 100 złotych z konta Dawida Wójcika (albo dowolnego innego użytkownika).

```
UPDATE accounts
SET balance=(balance-100)
WHERE id=2;
```

Instrukcja UPDATE, odrobina matematyki i klauzula WHERE pozwalają zmniejszyć stan konta o 100 złotych. Choć MySQL informuje, że zmodyfikowany został jeden wiersz, efekt nie będzie trwały, dopóki transakcja nie zostanie zatwierdzona.

6. Wpłacić 100 złotych na konto Beaty Piechoty:

```
UPDATE accounts
SET balance=(balance+100)
WHERE id=1;
```

Jest to przeciwieństwo operacji wykonanej w etapie 5., co odzwierciedla przelewanie 100 złotych z jednego konta na drugie.

7. Sprawdzić wyniki (patrz rysunek 6.42).

```
SELECT * FROM accounts;
```

Jak widać na rysunku, na jednym koncie jest teraz o 100 złotych więcej, a na drugim o 100 złotych mniej niż na początku (patrz rysunek 6.41).

```
MySQL Command Line Client
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM accounts;
+----+-----+-----+
| id | name  | balance |
+----+-----+-----+
| 1  | Beata Piechota | 5460.30 |
| 2  | Dawid Wójcik  | 909325.24 |
| 3  | Ilona Machnik  | 892.00  |
+----+-----+-----+
3 rows in set (0.00 sec)

mysql> _
```

Rysunek 6.41. Rozpoczęto transakcję i wyświetlono wszystkie rekordy znajdujące się w tabeli

```
MySQL Command Line Client
mysql> UPDATE accounts SET
-> balance=(balance-100)
-> WHERE id=2;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> UPDATE accounts SET
-> balance=(balance+100)
-> WHERE id=1;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM accounts;
+----+-----+-----+
| id | name  | balance |
+----+-----+-----+
| 1  | Beata Piechota | 5560.30 |
| 2  | Dawid Wójcik  | 909225.24 |
| 3  | Ilona Machnik  | 892.00  |
+----+-----+-----+
3 rows in set (0.00 sec)

mysql> _
```

Rysunek 6.42. Wykonano dwa zapytania i wyświetlono wyniki

```

mysql> SELECT * FROM accounts;
+----+-----+-----+
| id | name  | balance |
+----+-----+-----+
| 1  | Beata Piechota | 5560.30 |
| 2  | David Wójcik   | 909225.24 |
| 3  | Ilona Machnik  | 892.00  |
+----+-----+-----+
3 rows in set (0.00 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0.03 sec)

mysql> SELECT * FROM accounts;
+----+-----+-----+
| id | name  | balance |
+----+-----+-----+
| 1  | Beata Piechota | 5560.30 |
| 2  | David Wójcik   | 909225.24 |
| 3  | Ilona Machnik  | 892.00  |
+----+-----+-----+
3 rows in set (0.00 sec)

mysql> _

```

Rysunek 6.43. Ponieważ użyto instrukcji `ROLLBACK`, wykonane wcześniej zapytania zostały wycofane

```

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE accounts SET
-> balance=balance+100)
-> WHERE id=2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> UPDATE accounts SET
-> balance=balance-100)
-> WHERE id=1;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.03 sec)

mysql> SELECT * FROM accounts;
+----+-----+-----+
| id | name  | balance |
+----+-----+-----+
| 1  | Beata Piechota | 5560.30 |
| 2  | David Wójcik   | 909225.24 |
| 3  | Ilona Machnik  | 892.00  |
+----+-----+-----+
3 rows in set (0.02 sec)

mysql> _

```

Rysunek 6.44. Instrukcja `COMMIT` utrwała rezultaty transakcji

8. Wycofać transakcję.

```
ROLLBACK;
```

Aby zademonstrować wycofywanie transakcji, unieważniam wykonane wcześniej zapytania. Instrukcja `ROLLBACK` przywraca bazę danych do stanu sprzed rozpoczęcia transakcji. Instrukcja ta kończy jednocześnie transakcję, przełączając MySQL-a z powrotem do trybu automatycznego zatwierdzania.

9. Sprawdzić wyniki (patrz rysunek 6.43).

```
SELECT * FROM accounts;
```

Zapytanie powinno zwrócić zawartość tabeli sprzed rozpoczęcia transakcji.

10. Powtórzyć etapy 4. – 6.

Aby sprawdzić, co się stanie w razie zatwierdzenia transakcji, należy ponownie wykonać dwa zapytania `UPDATE`. Należy pamiętać o rozpoczęciu nowej transakcji, ponieważ w przeciwnym razie zapytania zostaną automatycznie zatwierdzone!

II. Zatwierdzić transakcję i sprawdzić wyniki (patrz rysunek 6.44).

```
COMMIT;
SELECT * FROM accounts;
```

Po wydaniu instrukcji `COMMIT` transakcja zostanie zatwierdzona, co oznacza, że wszystkie zmiany staną się trwałe. Instrukcja ta jednocześnie kończy transakcję, przełączając MySQL-a z powrotem do trybu automatycznego zatwierdzania.

Wskazówki

- Jedną z największych zalet transakcji jest to, że zapewniają one ochronę przed zdarzeniami losowymi, takimi jak awaria serwera. Transakcja jest albo w całości zatwierdzana, albo ignorowana.
- Aby wyłączyć tryb automatycznego zatwierdzania w serwerze MySQL-a, należy wpisać:

```
SET AUTOCOMMIT=0;
```

Nie trzeba będzie wówczas wydawać instrukcji `START TRANSACTION`, a wszystkie wykonane zapytania zostaną zrealizowane dopiero po wpisaniu instrukcji `COMMIT` (albo użyciu zapytania `CREATE`, `ALTER` itp.).

- W transakcjach można tworzyć *punkty zapisu*:

```
SAVEPOINT nazwa_punktu_zapisu;
```

Funkcja ta umożliwia wycofanie transakcji do określonego punktu zapisu:

```
ROLLBACK TO SAVEPOINT nazwa_punktu_zapisu;
```