helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

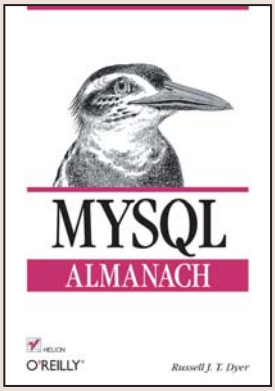
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

MySQL. Almanach



Autor: Russell J. T. Dyer
Tłumaczenie: Rafał Jońca
ISBN: 83-246-0130-9
Tytuł oryginału: MySQL in a Nutshell
Format: B5, stron: 294

Przewodnik po najpopularniejszej dostępnej nieodpłatnie bazie danych

- Przegląd instrukcji języka SQL
- Polecenia klienta i serwera MySQL
- Funkcje interfejsów programistycznych

MySQL to stabilny, wydajny i szybki system zarządzania bazami danych dostępny nieodpłatnie, na licencji open source. Najczęściej stosowany jest jako zaplecze bazodanowe witryn WWW, ale coraz częściej sięgają po niego twórcy rozbudowanych aplikacji, którzy do niedawna wykorzystywali drogie, komercyjne bazy danych. MySQL posiada spore możliwości, a administracja nim nie nastręcza większych problemów, dzięki wielu narzędziom tekstowym i graficznym ułatwiającym pracę z tym systemem. Dostępność wielu interfejsów programistycznych (API) bardzo ułatwia tworzenie aplikacji opartych na MySQL.

Książka „MySQL. Almanach" to podręcznik dla użytkowników, administratorów i programistów korzystających z bazy danych MySQL. Zawiera opisy instrukcji i funkcji MySQL, narzędzi administracyjnych i najpopularniejszych interfejsów programistycznych. Przedstawia proces instalacji bazy i tworzenia nowych tabel, sposoby konstruowania efektywnych zapytań oraz składnię i parametry poleceń stosowanych w pracy z tekstowymi narzędziami klienckimi i administracyjnymi.

- Instalacja MySQL w różnych systemach operacyjnych
- Tworzenie baz i tabel
- Wprowadzanie danych i import z plików tekstowych
- Wybieranie danych
- Instrukcje i funkcje języka SQL
- Operacje na liczbach, tekstach i datach
- Narzędzia dostępne z wiersza poleceń
- Funkcje API dla Perla, PHP i C

Dzięki wiadomościom zawartym w tej książce praca z MySQL stanie się bardziej wydajna.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

ROZDZIAŁ 3.

Podstawy MySQL

Choć niniejsza książka zawiera materiał podręcznikowy, który można czytać małymi fragmentami w razie potrzeby, w niniejszym rozdziale znajduje się proste ćwiczenie uczące podstaw MySQL. Informuje, w jaki sposób zalogować się do serwera, utworzyć bazę danych, a także wpisać i dokonać edycji znajdujących się w niej danych. Trzeba jednak zdawać sobie sprawę z tego, iż ćwiczenie nie obejmuje wszystkich zagadnień. Stanowi raczej wskazówkę, w jaki sposób należy wykonywać pewne rodzaje zadań w MySQL.

Klient mysql

Istnieje wiele sposobów interakcji z serwerem MySQL, a tym samym tworzenia i używania bazy danych. Najprostszym interfejsem jest klient *mysql*. Dzięki niemu możliwa jest interakcja przy użyciu wiersza poleceń. Program często nazywany jest monitorem MySQL.

Jeśli serwer MySQL został poprawnie zainstalowany i uruchomiony, *mysql* powinien włączyć się bez przeszkód. Jeżeli tak się nie stanie, należy przeprowadzić instalację zgodnie z krokami opisanymi w rozdziale 2. Jeżeli instalacja została przeprowadzona w sposób domyślny, program *mysql* znajduje się w katalogu */usr/local/mysql/bin/*. Aby upewnić się, iż znajdzie się on w ścieżce wyszukiwania, wystarczy wykonać poniższe wiersze:

```
PATH=$PATH:/usr/local/mysql/bin
export PATH
```

Zakładając, iż wszystko działa poprawnie, potrzebna jest jeszcze nazwa użytkownika i hasło. Jeśli nie jest się administratorem, należy uzyskać obie informacje od osoby zarządzającej serwerem. Jeżeli serwer MySQL został zainstalowany dopiero przed chwilą, użytkownik *root* posiada puste hasło. Sposób ustawiania haseł i tworzenia użytkowników z różnymi prawami został opisany w rozdziale 2.

Logowanie się do serwera MySQL z poziomu powłoki wygląda następująco.

```
mysql -h host -u użytkownik -p
```

Jeśli dokonuje się logowania do lokalnego serwera (mieszczącego się na tym samym komputerze fizycznie lub logicznie, na przykład dzięki połączeniu Telnet lub SSH), można pominąć argument `-h host`. Klient domyślnie przyjmuje, iż logowanie dotyczy hosta *localhost*, który odnosi się do aktualnego systemu. W przypadku chęci dołączenia do serwera istniejącego na innym komputerze, trzeba podać jego nazwę, którą można przełożyć na adres IP, lub bezpośrednio wpisać adres IP.

Argument *użytkownik* należy zastąpić właściwą nazwą użytkownika. Opcja `-p` instruuje *mysql*, aby poprosił o podanie hasła. Możliwe jest przekazanie hasła na końcu opcji `-p` (wpisz `-prower`, jeśli hasłem jest `rower`); między opcją a hasłem nie występuje znak spacji. Wpisywanie hasła w wierszu poleceń nie jest dobrym rozwiązaniem ze względów bezpieczeństwa, ponieważ jest wówczas przesyłane przez sieć jako niezakodowany tekst, a w dodatku ktoś może podejrzeć listę procesów uruchomionych na danym komputerze.

Aby zakończyć pracę z *mysql*, wpisz **`quit`** lub **`exit`** i naciśnij klawisz *Enter*.

Tworzenie bazy danych i tabel

Zakładając, iż ma się wszystkie prawa wymagane do tworzenia i modyfikowania bazy danych na serwerze, można przystąpić do tworzenia nowej bazy danych i tabel. W niniejszym rozdziale wykonamy bazę danych dla fikcyjnej księgarni.

```
CREATE DATABASE ksiegarnia;
```

To krótkie polecenie tworzy bazę danych o nazwie *ksiegarnia*. W niniejszej książce polecenia i zarezerwowane słowa będą pisane wielkimi literami. Nie jest to jednak wymagane — MySQL nie rozróżnia wielkości liter w słowach kluczowych i klauzulach. Nazwy baz danych i tabel są czułe na wielkość liter w systemach operacyjnych czułych na wielkość liter, na przykład systemach uniksowych, ale nie są czułe w systemach, które nie zwracają uwagi na wielkość liter, na przykład systemach Windows. Przyjęło się jednak, by słowa kluczowe w dokumentacjach SQL pisać wielkimi literami, a nazwy tabel, baz danych i kolumn małymi literami.

Polecenia SQL są zakończone znakiem średnika. Polecenie SQL może rozciągać się na więcej niż jeden wiersz. Jego wysłanie do serwera i przetworzenie rozpoczyna się dopiero po wykryciu znaku średnika. Aby zatrzymać uruchomione polecenie SQL, zamiast średnika należy wpisać **`\c`**.

Skoro baza danych jest już założona, warto w aktualnej sesji określić domyślną bazę danych, stosując poniższe polecenie:

```
USE ksiegarnia;
```

Kolejny krok to utworzenie pierwszej tabeli, w której później znajdą się dane. Pierwsza tabela będzie przechowywała podstawowe informacje o książce, ponieważ jest to najważniejszy element książkowego biznesu.

```
CREATE TABLE ksiazki (
  id_rek INT,
  tytul VARCHAR(50),
  autor VARCHAR(50)
);
```

Polecenie tworzy tabelę o nazwie *ksiazki* z trzema kolumnami. Pierwsza kolumna to po prostu numer identyfikacyjny każdego z rekordów. Jest typu całkowitoliczbowego. Warto pamiętać, iż w MySQL pola nazywane są **kolumnami**, a rekordy **wierszami**. Typem danych dla drugiej i trzeciej kolumny są pola tekstowe o zmiennej długości — mogą pomieścić maksymalnie 50 znaków. Lista kolumn znajduje się w nawiasach.

Aby poznać opis właśnie utworzonej tabeli, wpisz instrukcję `DESCRIBE`, która wyświetli poniższą tabelę:

```
DESCRIBE ksiazki;
```

```
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id_rek | int(11)     | YES  |     | NULL    |       |
| tytul  | varchar(50) | YES  |     | NULL    |       |
| autor  | varchar(50) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
```

Przyglądając się tabeli, nietrudno się domyślić, iż potrzebne byłyby dodatkowe kolumny z wydawcą, rokiem publikacji, numerem ISBN, gatunkiem, opisem książki itp. Co więcej, przydałoby się, by MySQL automatycznie przypisał wartość kolumnie `id_rek`, aby nie trzeba było samemu martwić się wymyślaniem wartości i sprawdzaniem, czy się nie powielają. Dodatkowo, decydujemy się zamienić kolumnę `autor` z wersji zawierającej nazwisko autora na numer identyfikacyjny autora, który będzie pochodził z osobnej tabeli zawierającej listę autorów. Zredukuje to ilość tekstu do wpisywania, ułatwi sortowanie i wyszukiwanie, gdyż dane będą jednorodne. Aby dokonać zmian w już istniejącej tabeli, zastosuj następującą instrukcję SQL:

```
ALTER TABLE ksiazki
CHANGE COLUMN id_rek id_rek INT AUTO_INCREMENT PRIMARY KEY,
CHANGE COLUMN autor id_autora INT,
ADD COLUMN opis BLOB,
ADD COLUMN gatunek ENUM('powieść','poezja','dramat'),
ADD COLUMN id_wydawcy INT,
ADD COLUMN rok_pub VARCHAR(4),
ADD COLUMN isbn VARCHAR(20);
```

Każdy wiersz, w którym znajdują się informacje o zmienianych lub dodawanych kolumnach, poza pierwszym, musi być oddzielony przecinkiem. W drugim wierszu następuje zmiana kolumny *id_rek*. Choć nazwa kolumny i jej typ pozostaje bez zmian, trzeba podać je raz jeszcze. Znacznik `AUTO_INCREMENT` służy do zapewnienia działania opisanego we wcześniejszym akapicie, czyli przypisania każdej nowej książce unikatowej wartości. Dodatkowo wiersz zostaje oznaczony jako `PRIMARY KEY`, aby przyspieszyć pobieranie danych.

Trzeci wiersz dokonuje zmian w kolumnie *autor* w taki sposób, aby dostosować jej tytuł i typ do tabeli *autorzy*, która wkrótce powstanie. Tabela *autorzy* będzie zawierała kolumnę klucza głównego, po której dokona się złączenia tabel. Ponieważ kolumna ta będzie typu całkowitoliczbowego, taki typ trzeba zastosować w zmienianej kolumnie autora.

Czwarty wiersz dodaje kolumnę z opisem książki. Zastosowany typ danych to `BLOB`, co jest skrótem od *binary large object* (duży obiekt binarny). Ten typ danych ma zmienną długość i potrafi przechowywać do 64 kilobajtów danych. Istnieją typy danych potrafiące przechowywać jeszcze więcej informacji. Ich pełna lista wraz z limitami znajduje się w dodatku A.

W kolumnie *gatunek* wymienia się możliwe wartości, aby zapewnić jednorodność. Dostępne są również wartości: pusta i NULL, choć nie zostały jawnie wymienione w poleceniu.

Przed rozpoczęciem dodawania danych do tabeli *ksiazki*, warto jeszcze utworzyć tabelę *autorzy*. Tabela *autorzy* będzie tak zwaną tabelą referencyjną (nazywaną też czasem słownikiem). Trzeba ją ustawić jako pierwszą, ponieważ w trakcie wpisywania danych do tabeli *ksiazki* potrzebny będzie numer identyfikacyjny autora.

```
CREATE TABLE autorzy
(id_rek INT AUTO_INCREMENT PRIMARY KEY,
nazwisko VARCHAR(50),
imie VARCHAR(50),
kraj VARCHAR(50));
```

Ta tabela nie wymaga znaczącej liczby kolumn, choć w rzeczywistej księgarni zapewne zostałyby zastosowane dodatkowe kolumny opisowe. W przyszłości dojdzie do złączenia tabel *ksiazki* i *autorzy* za pomocą wartości z kolumny *id_autora* tabeli *ksiazki* i kolumny *id_rek* powyższej tabeli. Osobiście zawsze w ten sam sposób nazywam kolumnę klucza głównego każdej tabeli (*id_rek*), aby przy wpisywaniu zapytań nie zastanawiać się nad nazwą i nie musieć sprawdzać jej poleceniem `DESCRIBE`.

W powyższej tabeli imię i nazwisko autora zostało rozbite na dwie kolumny, aby ułatwić sortowanie i wyszukiwanie po nazwisku. Dodatkowo pojawiła się kolumna kraju pochodzenia autora, co umożliwi wyszukiwanie książek napisanych przez autorów pochodzących z kraju podanego przez użytkownika.

Pokaż mi

Warto zatrzymać się na chwilę i popodziwiać wykonaną do tej pory pracę. Aby uzyskać listę baz danych, należy wykonać polecenie `SHOW DATABASES`.

```
SHOW DATABASES;
```

```
+------------+
| Database   |
+------------+
| ksiegarnia |
| mysql      |
| test       |
+------------+
```

Wynik wykonania polecenia ujawnia istnienie dwóch dodatkowych baz danych poza właśnie wykonaną bazą. Jedną z tych dodatkowych baz jest *mysql*, która zawiera dane dotyczące przywilejów użytkowników. Została ona pokrótce omówiona w poprzednim rozdziale. Trzecia wymieniona baza danych (*test*) jest tworzona automatycznie w trakcie instalacji MySQL. Tradycyjnie używa się jej do dodawania tabel, dla których chce się przeprowadzić testowanie poleceń SQL.

Aby wyświetlić listę tabel bazy danych *ksiegarnia* (po wcześniejszym wybraniu tej bazy danych poleceniem `USE`), wpisz poniższe polecenie:

```
SHOW TABLES;
```

```
+----------------------+
| Tables_in_ksiegarnia |
+----------------------+
| autorzy              |
| ksiazki              |
+----------------------+
```

Wynik działania polecenia SHOW TABLES przedstawia listę zawierającą, zgodnie z oczekiwaniami, dwie tabele. Jeśli chce się wyświetlić listę tabel innej bazy danych, korzystając nadal z aktywnej bazy danych *ksiegarnia*, należy do wcześniejszego polecenia dodać klauzulę FROM.

```
USE ksiegarnia;
SHOW TABLES FROM mysql;
```

Spowoduje to wyświetlenie tabel bazy danych *mysql*, choć klient nadal będzie ściśle związany z bazą danych *ksiegarnia*.

Wstawianie danych

Po wykonaniu dwóch pierwszych tabel warto wstawić do nich dane. Najprostszy sposób wykonania tego zadania to użycie polecenia INSERT. Dzięki temu poleceniu można za jednym podejściem dodać jeden lub więcej rekordów. Przed dodaniem informacji o książce do tabeli *ksiazki* trzeba wypełnić pole tabeli *autorzy*, aby móc odnieść się od identyfikatora autora. Wykonaj poniższe polecenie, korzystając z klienta *mysql*:

```
INSERT INTO autorzy
(nazwisko, imie, kraj)
VALUES('Grochola','Katarzyna','Polska');
```

Po dodaniu informacji o autorze, można wstawić napisaną przez niego książkę.

```
INSERT INTO ksiazki
(tytul, id_autora, isbn, gatunek, rok_pub)
VALUES('Nigdy w życiu', LAST_INSERT_ID(), '83-88221-55-8','powieść','2001');
```

Pierwsze polecenie spowodowało dodanie rekordu dla Katarzyny Grocholi, autorki książki *Nigdy w życiu*. Standardowa składnia polecenia INSERT najpierw określa kolumny, w których mają zostać umieszczone dane. Jeśli chce się wstawiać dane do wszystkich kolumn w kolejności określonej w bazie danych, nie trzeba podawać nazw kolumn. W drugim poleceniu SQL lista kolumn ma inną kolejność niż oryginalna lista kolumn w tabeli. Takie rozwiązanie jest w pełni akceptowane przez MySQL — trzeba jednak zapewnić taką samą kolejność przekazywanych danych. Identyfikator autora dla kolumny *id_autora* pobieramy z poprzedniego polecenia, korzystając z funkcji LAST_INSERT_ID().

Pobieranie danych

Skoro w obu tabelach znajdują się dane, warto wykonać pewne zapytania. Do pobrania danych służy polecenie SELECT. Aby pobrać wszystkie kolumny i wiersze tabeli *ksiazki*, wpisz poniższe polecenie:

```
SELECT * FROM ksiazki;
```

Znak gwiazdki jest w tym przypadku znakiem wieloznaczności powodującym pobranie wszystkich kolumn. Ponieważ nie zostały określone żadne kryteria, dzięki którym miałyby zostać pobrane jedynie niektóre wiersze, polecenie spowoduje pobranie wszystkich wierszy tabeli *ksiazki*. Aby pobrać konkretne kolumny i wiersze, należy podać nazwy kolumn i zastosować klauzulę WHERE z warunkami stawianymi poszczególnym wierszom.

```
SELECT id_rek, tytul, opis
FROM ksiazki
WHERE gatunek = 'nowela';
```

Polecenie SQL wyświetla jedynie numer identyfikacyjny, tytuł i opis wszystkich książek z tabeli *ksiazki*, które należą do gatunku nowela. Oczywiście wyniki byłyby bardziej imponujące, gdyby w tabeli znajdowało się więcej książek. Załóżmy, iż wpisaliśmy do bazy danych kilkadziesiąt książek i kontynuujmy pracę.

Aby z bazy danych pobrać wszystkie książki napisane przez wybranego autora, trzeba dokonać połączenia tabeli *ksiazki* z tabelą *autorzy*. Poniżej został przedstawiony przykład takiego połączenia.

```
SELECT ksiazki.id_rek, tytul, rok_pub,
   CONCAT(imie, ' ' , nazwisko) AS autor
FROM ksiazki, autorzy
WHERE nazwisko = 'Grochola' AND id_autora = autorzy.id_rek;
```

Obie tabele posiadają kolumnę o nazwie *id_rek*, więc poza nazwą kolumny trzeba określić nazwę tabeli, by poprawnie dokonać złączenia. W tym celu przed nazwą kolumny umieszcza się nazwę tabeli oraz znak kropki jako separator. Przykład takiego rozwiązania znajduje się w pierwszym wierszu, w którym dochodzi do pobrania numeru identyfikacyjnego rekordu. Drugi wiersz korzysta z funkcji tekstów o nazwie `CONCAT()`. Dzięki tej funkcji można połączyć ze sobą kilka fragmentów danych w jeden tekst, aby uzyskać lepiej wyglądający wynik. W tym przypadku funkcja łączy imię, znak spacji (w apostrofach) oraz nazwisko autora. Wynik połączenia będzie widoczny jako jedna kolumna o nazwie *autor*, ponieważ taka nazwa została podana jako alias dzięki słowu kluczowemu `AS`. Klauzula `FROM` wymienia obie tabele oddzielone przecinkiem. Jeśli istniałoby więcej tabel, wystarczyłoby je podać w tej klauzuli w dowolnej kolejności, oddzielając je przecinkami. W klauzuli `WHERE` informujemy serwer, iż jesteśmy zainteresowani książkami napisanymi przez autora o nazwisku „Grochola". Dodatkowo klauzula ta zawiera warunek złączenia obu tabel (ostatni wiersz). Złączenie dotyczy kolumny *id_autora* tabeli *ksiazki* i kolumny *id_rek* tabeli *autorzy*. Jeśli w tabeli nie znajduje się żadna książka napisana przez takiego autora, nie zostaną zwrócone żadne wiersze. Jeżeli książka takiego autora została wpisana, ale jego dane nie znajdują się w tabeli *autorzy*, również nie zostaną wyświetlone żadne wiersze. Oto przykładowy wynik wykonania poprzedniego polecenia SQL.

```
+--------+-------------------+---------+--------------------+
| id_rek | tytul             | rok_pub | autor              |
+--------+-------------------+---------+--------------------+
|      1 | Nigdy w życiu     | 2001    | Katarzyna Grochola |
|      2 | Podanie o miłość  | 2002    | Katarzyna Grochola |
+--------+-------------------+---------+--------------------+
```

Nietrudno zauważyć, iż zostały odnalezione dwie książki Katarzyny Grocholi. Tytuł ostatniej kolumny został określony w zapytaniu za pomocą słowa kluczowego `AS`. Podobne zmiany można wymusić dla innych kolumn, korzystając z tego samego słowa kluczowego. Alias *autor* może zostać użyty w innym miejscu polecenia `SELECT`, ale niestety nie w klauzuli `WHERE`. Więcej informacji na temat `AS` znajduje się w rozdziale 4.

Kolejność, limitowanie wyników i grupowanie

Gdy pobiera się duży zbiór danych, warto posortować znajdujące się w nim informacje według konkretnego klucza. W tym celu stosuje się klauzulę `ORDER BY`. Przypuśćmy, iż potrzebujemy pobrać z bazy danych wszystkie sztuki napisane przez Williama Shakespeare'a. Poniższe zapytanie SQL pobierze odpowiednią listę i posortuje ją według tytułu sztuki:

```
SELECT ksiazki.id_rek, tytul, wydawca
FROM ksiazki, autorzy, wydawcy
WHERE nazwisko = 'Shakespeare'
   AND gatunek = 'sztuka'
   AND id_autora = autorzy.id_rek
   AND id_wydawcy = wydawcy.id_rek
ORDER BY tytul, rok_pub;
```

Klauzula `ORDER BY` znajduje się na końcu, po klauzuli `WHERE`. Najpierw sortowanie odbywa się po tytule (kolumnie *tytul*), a w ramach tego samego tytułu po roku publikacji (kolumnie *rok_pub*). Domyślnie dane porządkowane są w porządku alfabetycznym. Jeśli chce się posortować tytuły w odwrotnym porządku alfabetycznym, zaraz po nazwie kolumny *title* w klauzuli `ORDER BY`, ale przed przecinkiem rozpoczynającym *rok_pub*, trzeba zastosować opcję `DESC`.

Duża księgarnia może posiada wiele wydań sztuk Shakespeare'a, być może nawet kilka wydań tej samej sztuki. Aby ograniczyć liczbę wyświetlanych rekordów, stosuje się klauzulę `LIMIT` na końcu polecenia SQL.

```
SELECT ksiazki.id_rek, tytul
FROM ksiazki, autorzy, wydawcy
WHERE nazwisko = 'Shakespeare'
   AND gatunek = 'sztuka'
   AND id_autora = autorzy.id_rek
   AND id_wydawcy = wydawcy.id_rek
ORDER BY tytul, rok_pub
LIMIT 20;
```

Dodatkowa klauzula ograniczy liczbę zwróconych wierszy do pierwszych 20. Liczenie rozpoczyna się od pierwszego wiersza zbioru wyników zaraz po posortowaniu danych zgodnie z wymaganiami zawartymi w klauzuli `ORDER BY`. Jeśli chce się pobrać kolejnych 10 pozycji, trzeba w klauzuli `LIMIT` najpierw podać liczbę wierszy do pominięcia, a następnie po przecinku liczbę wierszy do pobrania. Jeżeli zechcemy pominąć pierwszych 20 wyników i wyświetlić jedynie kolejnych 10, trzeba zastąpić wcześniejszą klauzulę `LIMIT` następującą wersją:

```
...
LIMIT 20, 10;
```

W dwuargumentowej wersji klauzuli pierwsza wartość określa liczbę wierszy do pominięcia (w przykładzie 20), a druga maksymalną liczbę wierszy do pobrania (w przykładzie 10).

Jeżeli chcemy pobrać jedynie listę sztuk Shakespeare'a i nie jesteśmy zainteresowani datą publikacji ani wydawcą — innymi słowy, jesteśmy zainteresowani jedynie pierwszym znalezionym wierszem dla każdego tytułu — możemy zastosować klauzulę `GROUP BY`.

```
SELECT ksiazki.id_rek, tytul
FROM ksiazki, autorzy
WHERE nazwisko = 'Shakespeare'
   AND id_autora = autorzy.id_rek
GROUP BY tytul;
```

Wynikiem działania powyższego polecenia SQL jest lista wszystkich tytułów sztuk Shakespeare'a istniejących w bazie danych. Numer identyfikacyjny będzie dotyczył pierwszego znalezionego wiersza dla każdego tytułu. Co ciekawe, `GROUP BY` zwróci te same dane co `ORDER BY` zastosowane dla tej samej kolumny.

Analiza i manipulacja danymi

MySQL umożliwia nie tylko pobieranie surowych danych, ale również ich analizę i formatowanie. Przypuśćmy, iż chcemy się dowiedzieć, ile powieści Tołstoja posiadamy. W tym celu trzeba w poleceniu SQL zastosować funkcję `COUNT()`.

```
SELECT COUNT(*)
FROM ksiazki, autorzy
WHERE nazwisko = 'Tołstoj'
   AND id_autora = autorzy.id_rek;

+----------+
| COUNT(*) |
+----------+
|       12 |
+----------+
```

Załóżmy, że po ustawieniu i uruchomieniu bazy danych zawiera ona tabelę *zamowienia* z informacjami na temat zamówień użytkowników. Możemy wykorzystać tę tabelę, aby sprawdzić sprzedaż wybranej książki. Aby sprawdzić kwotę uzyskaną ze sprzedaży książki *Armadillo* autorstwa Williama Boyda, wystarczy wpisać poniższe polecenie SQL w kliencie *mysql*:

```
SELECT SUM(kwota_sprzedazy) AS 'Sprzedaż Armadillo'
FROM zamowienia, ksiazki, autorzy
WHERE tytul = 'Armadillo'
   AND nazwisko = 'Boyd'
   AND id_ksiazki = ksiazki.id_rek;
   AND id_autora = autorzy.id_rek;

+--------------------+
| Sprzedaż Armadillo |
+--------------------+
|             250.25 |
+--------------------+
```

Aby uzyskać odpowiednie informacje, łączymy trzy tabele. MySQL pobiera wartość kolumny *kwota_sprzedazy* dla każdego wiersza tabeli *zamowienia* spełniającego kryteria zawarte w klauzuli `WHERE`. Następnie sumuje znajdujące się tam wartości i wyświetla je w kolumnie o podanej nazwie. Większość nazw kolumn występuje tylko w jednej tabeli, więc MySQL nie ma problemów ze stwierdzeniem, czego dotyczą. Niemniej dla kilku kolumn trzeba zastosować wersję *`tabela.kolumna`*.

Korzystając z wielu różnorodnych funkcji, można sterować sposobem formatowania kolumn zawierających datę lub czas. Załóżmy, iż chcemy wydobyć z tabeli *zamowienia* datę złożenia zamówienia na podstawie posiadanego numeru rachunku (na przykład 1250), który tak naprawdę jest numerem identyfikującym rekord (*id_rek*). Zastosowanie poniższego polecenia SQL spowoduje zwrócenie daty w domyślnym formacie.

```
SELECT data_zakupu AS 'Data zakupu'
FROM zamowienia
WHERE id_rek = '125O';

+---------------+
| Data zakupu   |
+---------------+
| 2004-03-01    |
+---------------+
```

Zastosowany format (rok-miesiąc-dzień) jest w pełni zrozumiały. Jeżeli jednak chce się wyświetlić nazwę miesiąca jako tekst, a nie liczbę, można skorzystać z odpowiednich funkcji daty.

```
SELECT CONCAT(DAYOFMONTH(data_zakupu), ' ',
       MONTHNAME(data_zakupu), ' ',
       YEAR(data_zakupu)) AS 'Data zakupu'
FROM orders
WHERE rec_id = '125O';

+---------------+
| Data zakupu   |
+---------------+
| 1 March 2004  |
+---------------+
```

Aby przedstawić datę w formacie często stosowanym w Polsce, korzystamy z funkcji `CONCAT()` i kilku funkcji daty. Początkowo zastosowany kod może wydawać się niezrozumiały z powodu wstawiania dodatkowych znaków spacji między poszczególne elementy daty. Pierwsza z funkcji pobiera z daty liczbę reprezentującą dzień i po prostu ją wyświetla. Kolejna funkcja wydobywa z kolumny `data_zakupu` miesiąc i zwraca go jako nazwę w języku angielskim. Trzecia funkcja, znajdująca się w trzecim wierszu, wydobywa rok. Przyglądając się wynikowi, łatwo stwierdzić, iż ten złożony kod działa prawidłowo. Nie jest to jednak najbardziej wygodny sposób formatowania daty. Lepsze rozwiązanie polega na zastosowaniu funkcji `DATE_FORMAT()`.

```
SELECT DATE_FORMAT(data_zakupu, "%d %M %Y")
       AS 'Data zakupu'
FROM orders
WHERE rec_id = '125O';
```

To rozwiązanie jest znacznie prostsze i krótsze, a co najważniejsze, daje identyczne wyniki. Aby poprawnie skorzystać z powyższej funkcji, trzeba znać kody formatujące. Zostały one wymienione w rozdziale 6.

Modyfikacja danych

Do modyfikacji danych w bazie danych służy kilka różnych poleceń. Najbardziej podstawową i chyba najpopularniejszą instrukcją jest `UPDATE`. Dzięki niej można zmienić dane we wskazanych kolumnach wszystkich wierszy spełniających klauzulę `WHERE`. Przyglądając się wynikom jednego z wcześniejszych zapytań można zauważyć, iż data wydania dla książki Katarzyny Grocholi *Podanie o miłość* to rok 2002. Nie jest to poprawna wartość, gdyż książka została wydana w roku 2001. Aby uaktualnić tę informację, wpisz poniższe polecenie SQL:

```
UPDATE ksiazki
SET rok_pub = 2001
WHERE id_rek = '2';

Query OK, 1 rows affected (0.22 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Najpierw podaje się nazwę aktualizowanej tabeli. Następnie po słowie kluczowym `SET` pojawiają się nazwy kolumn i przypisywane im nowe wartości. Jeżeli zmienia się wartość więcej niż jednej kolumny, poszczególne przypisania należy rozdzielić przecinkami. Słowo kluczowe `SET` stosuje się tylko raz.

Powyższe polecenie posiada klauzulę `WHERE`, co powoduje ograniczenie liczby modyfikowanych wierszy wyłącznie do wierszy spełniających podane warunki. W przedstawionym przykładzie warunek dotyczy kolumny z unikatowymi wartościami, więc zmieniony zostanie tylko jeden wiersz. Wynik wykonania polecenia wskazuje na wpłynięcie na jeden z wierszy, dopasowanie się do jednego wiersza, zmienienie jednego z wierszy. W trakcie wykonywania nie pojawiły się żadne błędy powodujące zgłoszenie ostrzeżeń.

Czasem wstawienie danych do tabeli spowoduje powstanie duplikatu wiersza, gdy dane wiersza istniały już wcześniej. Przypuśćmy, iż wykonaliśmy polecenie SQL wstawiające do tabeli kilka książek, ale jedna z tych książek już była zawarta w bazie danych. Jeżeli zastosuje się polecenie `INSERT`, powstanie duplikat. Aby temu zapobiec, można użyć polecenia `REPLACE`, które wstawia nowy wiersz lub zastępuje istniejący wiersz nowymi danymi. Z perspektywy MySQL duplikacja występuje tylko wtedy, gdy unikatowe kolumny miałyby zawierać tę samą wartość. Ponieważ wartość kolumny *id_rek* jest przypisywana automatycznie, jej duplikacja jest mało prawdopodobna na etapie dodawania nowych rekordów. Unikatowym elementem każdej książki jest numer ISBN — dzięki niemu można jednoznacznie określić książkę. Aby zapewnić, iż nie pojawią się wiersze z tym samym numerem ISBN, warto ponownie zmodyfikować tabelę *ksiazki* i wymusić unikatowość wartości znajdujących się w kolumnie *isbn*. W ten sposób zapobiegnie się dwukrotnemu wpisaniu danych związanych z tą samą książką.

```
ALTER TABLE ksiazki
CHANGE COLUMN isbn isbn VARCHAR(20) UNIQUE;
```

Od teraz można wstawiać dane kolejnych książek nie martwiąc się o zduplikowane wiersze o tym samym numerze ISBN. Poniżej znajduje się przykładowy kod próbujący dodać dwie książki autorstwa Katarzyny Grocholi, z których jedna już znajduje się w tabeli:

```
REPLACE INTO ksiazki
(tytul, id_autora, isbn, gatunek, rok_pub)
VALUES('Nigdy w życiu', '1000', '83-88221-55-8','powieść','2001'),
      ('Ja wam pokażę', '1000', '83-89291-84-3','powieść','2004'),
```

Składnia polecenie `REPLACE` jest taka sama jak polecenia `INSERT`. Warto zauważyć dodanie dwóch wierszy w jednym poleceniu. Dokładnie tę samą składnię stosuje się, aby dodać wiele wierszy poleceniem INSERT. Dane każdego wiersza umieszcza się w nawiasach, a poszczególne grupy nawiasów oddziela przecinkami. Ponieważ w przedstawionym przykładzie w bazie danych istnieje wiersz dla książki o numerze ISBN 83-88221-55-8 (*Nigdy w życiu*), zostanie on zastąpiony, a nie dodany. Ponieważ drugiej książki nie ma jeszcze w tabeli, zostanie do niej dodana.

Usuwanie danych

Do usuwania konkretnych wierszy danych służy polecenie `DELETE`. Jeśli na przykład chce się usunąć z tabeli *ksiazki* wszystkie wiersze dotyczące książek autorki J. K. Rowling, ponieważ podjęło się decyzję o niesprzedawaniu książek *Harry Potter* (nie chce się prowadzić tego rodzaju biznesu), można użyć poniższego polecenia:

```
DELETE FROM ksiazki
WHERE id_autora =
   (SELECT autorzy.id_rek FROM autorzy
    WHERE nazwisko = 'Rowling'
       AND imie = 'J.K.');

DELETE FROM autorzy
WHERE nazwisko = 'Rowling'
   AND imie = 'J.K.';
```

Powyższy kod usuwa z tabeli *ksiazki* wszystkie książki, których identyfikator autora zawiera wartość zwróconą przez podzapytanie. Zapytanie zwraca identyfikator z tabeli *autorzy* dotyczący autora o wskazanym imieniu i nazwisku. Innymi słowy, kolumna *id_autora* musi zawierać wartość zwróconą przez polecenie `SELECT` (podzapytanie w nawiasie). Ponieważ kod korzysta z podzapytań, do poprawnego działania wymaga MySQL w wersji 4.1 lub nowszej. Aby wykonać to samo zadanie w jednej z wcześniejszych wersji MySQL, trzeba by wykonać polecenie `SELECT`, zapamiętać zwrócony identyfikator autora, a następnie wykonać polecenie `DELETE`, ręcznie wpisując numer identyfikacyjny.

Alternatywne rozwiązanie dla poprzedniej konstrukcji polega na zastosowaniu zmiennych zdefiniowanych przez użytkownika. Oto przykład korzystający ze zmiennych:

```
SET @potter =
   (SELECT id_rek FROM autorzy
     WHERE nazwisko = 'Rowling'
        AND imie = 'J.K.');

DELETE FROM ksiazki
WHERE id_autora = @potter;

DELETE FROM autorzy
WHERE id_rek = @potter;
```

W pierwszym fragmencie polecenie `SET` służy do utworzenia zmiennej o nazwie `@potter`, która będzie zawierać wynik działania polecenia `SELECT` umieszczonego w nawiasach (podzapytanie). Choć MySQL w wersjach wcześniejszych niż 4.1 nie obsługuje podzapytań, przedstawiony kod zadziała, gdyż dotyczy zmiennych definiowanych przez użytkownika. Drugie polecenie SQL usuwa z tabeli *ksiazki* wszystkie książki, które jako identyfikator autora posiadają wartość znajdującą się w zmiennej tymczasowej. Ostatnie polecenie usuwa dane z tabeli *autorzy*, ponownie korzystając ze zmiennej. Zmienne zdefiniowane przez użytkownika istnieją do momentu ich zresetowania lub zamknięcia sesji z serwerem MySQL.

Wyszukiwanie danych

Gdy baza danych zawiera ogromną liczbę informacji, znajdowanie danych przy użyciu ręcznego przeszukiwania wyników polecenia `SELECT` jest nie do zaakceptowania. Co więcej, czasem nie zna się dokładnego lub pełnego tekstu, który miałby istnieć w wybranej kolumnie. W takich sytuacjach korzysta się z operatora `LIKE`. Załóżmy, iż tabela *ksiazki* zawiera tysiące wpisów. Klient chce odnaleźć książkę, ale nie pamięta jej autora. Wie jedynie, iż w tytule książki występowały wyrazy **zimową** i **podróżny**. Można użyć tego strzępka informacji do przeszukania zawartości bazy danych, używając poniższego polecenia:

```
SELECT ksiazki.id_rek, tytul,
       CONCAT(imie, ' ', nazwisko) AS autor
FROM ksiazki, autorzy
WHERE tytul LIKE '%podróżny%'
   AND tytul LIKE '%zimową%'
   AND id_autora = autorzy.id_rek;

+--------+------------------------------+---------------+
| id_rek | tytul                        | autor         |
+--------+------------------------------+---------------+
|   1400 | Jeśli zimową nocą podróżny   | Italo Calvino |
+--------+------------------------------+---------------+
```

Poza operatorem `LIKE` dwukrotnie został użyty znak wieloznaczny procenta, aby wskazać, iż poszukuje się wszystkich wierszy, w których kolumna *tytul* zaczyna się od zera lub więcej znaków przed wzorcem **podróżny**, a po wzorcu może się pojawić 0 lub więcej innych znaków. Innymi słowy, wyraz **podróżny** musi się znaleźć w dowolnym miejscu w danych wybranej kolumny. Podobna sytuacja dotyczy wyrazu **zimową**. Warto pamiętać, iż słowo `LIKE` to operator. Więcej informacji na temat operatorów znajduje się w dodatku B.

Jeżeli inny klient poprosi o wyszukanie w bazie danych tytułu książki zawierającej wyraz **Ford** lub **Chevrolet**, należy zastosować w klauzuli `WHERE` operator `OR`.

```
SELECT ksiazki.id_rek, tytul,
       CONCAT(imie, ' ', nazwisko) AS autor
FROM ksiazki, autorzy
WHERE tytul LIKE '%Ford%' AND id_autora = autorzy.id_rek
OR tytul LIKE '%Chevrolet%' AND id_autora = autorzy.id_rek;
```

Więcej przykładów i możliwości wyszukiwania danych znajduje się w rozdziale 4.

Hurtowy import danych

Choć polecenia `INSERT` i `REPLACE` są bardzo użyteczne, potrafią być czasochłonne w przypadku wprowadzania dużej ilości danych, gdyż wymuszają ich ręczne wpisywanie. Często w momencie tworzenia nowej bazy danych trzeba dokonać przeniesienia danych ze starej bazy danych. Załóżmy, iż wydawca wysłał nam dysk z listą wszystkich swoich książek znajdującą się w zwykłym pliku tekstowym. Każdy rekord dotyczący poszczególnej książki znajduje się w osobnym wierszu, a każde pole zostało oddzielone znakiem pionowej kreski. Oto w jaki sposób mogą wyglądać fikcyjne dane od wydawcy.

```
ISBN|TYTUL|NAZWISKO|IMIE|DATA WYDANIA
0-907587-68-2|Notatki z podziemia|Dostojewski|Fiodor|kwiecień 1992|
...
```

Oczywiście rzeczywisty plik od wydawcy zawierałby znacznie więcej pól i rekordów niż zostało tutaj przedstawione, ale tutaj ograniczymy się jedynie do przykładu. Pierwszy wiersz zawiera opis pól rekordów. Nie należy pobierać pierwszego wiersza. Zawiera on po prostu instrukcje dla osoby edytującej plik. Poinformujemy MySQL, by zignorował pierwszy wiersz. Jeśli chodzi o dane, trzeba zająć się kilkoma problemami. Pola nie znajdują się w takiej samej kolejności, w jakiej są zapisane w bazie danych. Trzeba poinformować MySQL o zmianie kolejności przy zapisie. Inny problem polega na tym, iż plik tekstowy zawiera dane dla tabel *ksiazki* i *autorzy*. Obejście tego problemu nie jest łatwe, ale wykonalne. W pierwszym podejściu wydobędziemy jedynie informacje o autorze. Osobne polecenie SQL posłuży do wydobycia informacji na temat książki. Na początek należy przenieść plik od wydawcy (*ksiazki.txt*) do katalogu */tmp*, a następnie wykonać polecenie `LOAD DATA INFILE` w kliencie *mysql*.

```
LOAD DATA INFILE '/tmp/ksiazki.txt' REPLACE INTO TABLE autorzy
FIELDS TERMINATED BY '|' LINES TERMINATED BY '\r\n'
TEXT_FIELDS(kol1, kol2, kol3, kol4, kol5)
SET nazwisko = kol3, imie = kol4
IGNORE kol1, kol2, kol5, 1 LINES;
```

Klauzule `TEXT_FIELDS` i `IGNORE` dla kolumn nie są dostępne w wersjach MySQL starszych niż 4.1. Klauzula `IGNORE` *n* `LINES` jest dostępna w MySQL już od dłuższego czasu. Treść `IGNORE 1 LINES` spowoduje pominięcie pierwszego wiersza. Wracając do pierwszego wiersza polecenia, występuje w nim nazwa wczytywanego pliku i tabela, w której mają zostać umieszczone dane. Znacznik `REPLACE` daje taki sam efekt jak opisywane wcześniej polecenie `REPLACE`.

Drugi wiersz informuje o tym, iż pola oddzielane są znakiem pionowej kreski, a wiersze oddzielane znakami powrotu karetki (`\r`) i przejścia do nowego wiersza (`\n`). Jest to format typowego pliku MS-DOS. Pliki systemu Unix korzystają jedynie ze znaku przejścia do nowego wiersza. Trzeci wiersz tworzy aliasy dla poszczególnych kolumn. Czwarty wiersz zawiera nazwy kolumn tabeli i przypisuje im odpowiednie dane, korzystając z aliasów określonych w poprzednim wierszu. Ostatni wiersz informuje MySQL, aby zignorował niechciane kolumny, a także pominął pierwszy wiersz pliku tekstowego, gdyż nie zawiera on danych.

Jeśli korzysta się ze starszej wersji serwera MySQL, który nie zawiera nowej funkcji pozwalającej zignorować niechciane kolumny, trzeba wykonać kilka dodatkowych kroków. Istnieje kilka możliwych sposobów na wykonanie tego zadania. Jednym z prostszych sposobów (jeśli wczytywanych danych nie jest zbyt dużo) jest dodanie tymczasowych kolumn do tabeli autorzy. Kolumny te pomieszczą nadmiarowe dane z pliku tekstowego. Później będzie można je usunąć. Oto skrypt wykonujący całe zadanie:

```
ALTER TABLE autorzy
ADD COLUMN kol1 VARCHAR(50),
ADD COLUMN kol2 VARCHAR(50),
ADD COLUMN kol5 VARCHAR(50);

LOAD DATA INFILE '/tmp/ksiazki.txt' REPLACE INTO TABLE autorzy
FIELDS TERMINATED BY '|' LINES TERMINATED BY '\r\n'
IGNORE 1 LINES
(kol1, kol2, nazwisko, imie, kol5);

ALTER TABLE autorzy
DROP COLUMN kol1,
DROP COLUMN kol2,
DROP COLUMN kol5;
```

Przedstawione rozwiązanie działa poprawnie, choć nie jest tak przyjemne i proste jak wcześniejsze polecenie. Zauważ, iż w drugim poleceniu SQL klauzula `IGNORE` wymusza pominięcie jednego wiersza. Ostatni wiersz tego polecenia wymienia kolumny tabeli *autorzy*, w których mają zostać umieszczone importowane dane. Trzecie polecenie usuwa kolumny tymczasowe po zakończeniu wczytywania danych z pliku tekstowego. W tym celu stosuje instrukcję `DROP`. Na ogół nie można cofnąć wyników działania tej instrukcji, więc warto uważać.

Po umieszczeniu danych autorów z pliku tekstowego w tabeli *autorzy*, można przystąpić do wczytania danych książek i odszukania poprawnych wartości kolumny *id_autora* dla każdej książki. Zadanie to wykona poniższe polecenie SQL:

```
LOAD DATA INFILE '/tmp/ksiazki.txt' IGNORE INTO TABLE ksiazki
FIELDS TERMINATED BY '|' LINES TERMINATED BY '\r\n'
TEXT_FIELDS(kol1, kol2, kol3, kol4, kol5)
SET isbn = kol1, tytul = kol2,
    rok_pub = RIGHT(kol5, 4),
    id_autora =
      (SELECT autorzy.id_rek
       WHERE nazwisko = kol3
       AND imie = kol4)
IGNORE kol3, kol4, 1 LINES;
```

Polecenie zawiera kilka sztuczek, umożliwiających poprawne wykonanie całego zadania. Piąty wiersz polecenia wydobywa rok wydania z pola daty wydania (oryginalnie data wydania zawiera miesiąc i rok wydania), stosując funkcję `RIGHT()`, która pobiera cztery ostatnie znaki `kol5`. Od 6. wiersza występuje podzapytanie, które określa wartość kolumny *id_autora* na podstawie imienia i nazwiska autora. Wynik uzyskany w nawiasach zostanie przypisany do kolumny *id_autora*. Na końcu dochodzi do zignorowania kolumn `kol3`, `kol4` i pierwszego wiersza pliku tekstowego. Wykonanie tego samego zadania we wcześniejszych wersjach MySQL wymagałoby użycia kolumn tymczasowych lub dodatkowej tabeli. Znacznik IGNORE z pierwszego wiersza instruuje MySQL, aby ignorował wszystkie komunikaty błędów, nie zastępował żadnych duplikatów i kontynuował wstawianie kolejnych rekordów.

Interfejs wiersza poleceń

Aby wysłać zapytanie SQL do serwera MySQL, nie trzeba otwierać monitora MySQL. Czasem zachodzi potrzeba szybkiego wykonania polecenia SQL z samej powłoki lub wiersza poleceń. Przypuśćmy, iż mamy tabelę o nazwie *dostawcy* i chcemy szybko uzyskać listę dostawców z województwa śląskiego wraz z numerami telefonów. Aby ją uzyskać, wystarczy wpisać poniższe polecenie w powłoce systemu Linux (lub innego równoważnego systemu).

```
mysql --user='tina' --password='muller' \
-e "SELECT dostawca, telefon FROM dostawca \
    WHERE woj='śląskie'" ksiegarnia
```

Narzędzie *mysql* zostaje wywołane, ale nie wchodzi w tryb monitora. Pierwsze argumenty przekazują nazwę użytkownika i hasło. Pierwszy wiersz kończy się znakiem lewego ukośnika, aby poinformować powłokę, iż będą jeszcze następne argumenty. W przeciwnym przypadku trzeba by wszystko umieścić w jednym wierszu. Drugi wiersz zawiera argument `-e` wskazujący, iż tekst umieszczony za nim w cudzysłowach powinien zostać wykonany przez klienta *mysql*. Zawarte w cudzysłowach polecenie SQL ma dokładnie taką samą postać, jaka byłaby użyta w trybie monitora. Na końcu pojawia się nazwa bazy danych.

Istnieją inne opcje i narzędzia wiersza poleceń. Niektóre z nich służą da tworzenia kopii bezpieczeństwa lub wykonywania konserwacji serwera. Ich pełne omówienie znajduje się w rozdziale 11.

Podsumowanie

Oczywiście to nie wszystkie zadania, które można wykonywać przy użyciu serwera MySQL. Niniejsze ćwiczenie miało jedynie na celu przybliżenie podstaw tworzenia i zarządzania bazą danych. Kolejne rozdziały książki zawierają szczegółowe omówienia wszystkich instrukcji, klauzul, argumentów, opcji i funkcji serwera MySQL. Jeśli dopiero rozpoczyna się swoją przygodę z MySQL, warto zacząć od instrukcji i klauzul wymienionych w niniejszym rozdziale, a następnie skorzystać z kolejnych rozdziałów, by rozszerzyć swą wiedzę o dostępnych opcjach i funkcjach.