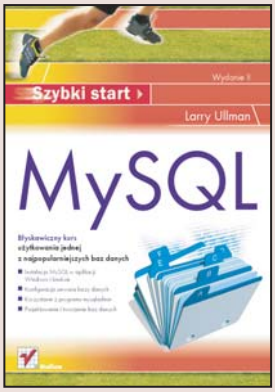
MySQL. Szybki start. Wydanie II



Autor: Larry Ullman
Tłumaczenie: Paweł Gonera na podstawie „MySQL. Szybki start" w tłumaczeniu Marka Pałczyńskiego
ISBN: 83-246-0665-3
Tytuł oryginału: MySQL, Second Edition: Visual QuickStart Guide (2nd Edition)
Format: B5, stron: 480

Błyskawiczny kurs użytkowania jednej z najpopularniejszych baz danych

MySQL to system zarządzania bazami danych, dostępny na licencji open-source. Swoimi możliwościami nie ustępuje w niczym potężnym komercyjnym systemom. Wykorzystywany jest zarówno jako zaplecze bazodanowe witryn WWW, jak i źródło danych dla rozbudowanych aplikacji korporacyjnych. MySQL dostępny jest niemal dla wszystkich systemów operacyjnych. Ogromną zaletą jest jego prosta obsługa. Dzięki temu nawet początkujący użytkownicy szybko opanują wszystkie możliwości i wykorzystają je w pracy. Zaawansowani z pewnością docenią wydajność, stabilność i funkcje znane z „kombajnów", takich jak Oracle lub MS SQL Server.

Książka „MySQL. Szybki start. Wydanie II" to kolejna edycja przewodnika po podstawach korzystania z tej bazy danych. W tej książce, zaktualizowanej zgodnie z najnowszą wersją programu, znajdziesz informacje dotyczące instalowania MySQL, uruchamiania go w różnych systemach operacyjnych oraz administrowania nim. Nauczysz się zakładać bazy i tabele, wykorzystywać język SQL do manipulowania danymi w bazie oraz łączyć się z bazą danych z poziomu programów napisanych w różnych językach. Każde zagadnienie jest przedstawione na praktycznym, bogato ilustrowanym przykładzie, co doskonale pomoże Ci w przyswojeniu wiedzy.

Książka porusza następujące tematy:

- Instalacja MySQL w aplikacji Windows i Linuksie
- Konfiguracja serwera bazy danych
- Korzystanie z programu mysqladmin
- Projektowanie i tworzenie baz danych
- Wprowadzanie, pobieranie i modyfikowanie danych za pomocą języka SQL
- Łączenie skryptów PHP i Perla z bazą danych MySQL
- Korzystanie z MySQL w programach napisanych w Javie
- Wyzwalacze i perspektywy
- Administrowanie serwerem MySQL

Poznaj ogrom możliwości systemu MySQL

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

Spis treści

Tworzenie bazy danych MySQL

4

W rozdziale 3., „Projektowanie bazy danych”, przedstawiłem operacje potrzebne do opracowania schematu bazy danych, które nazywane są *normalizacją*. Operacje te mogą być wykonywane w dowolnej relacyjnej bazie danych i nie są specyficzne dla MySQL. W tym rozdziale napiszę, w jaki sposób zaimplementować bazę danych w serwerze MySQL.

Proces ten rozpoczniemy od zapoznania się z dostępnymi typami danych oraz sposobami ich dalszego dostosowywania. Kolejno poznamy indeksy poprawiające wydajność bazy danych, zakończymy projektowanie każdej z tabel przez wybranie właściwych nazw i wybierzemy typy tabel. Na koniec wrócimy do pracy z narzędziami MySQL, tworząc i być może modyfikując bazę danych i tabele.

Typy danych MySQL

Po zdefiniowaniu wszystkich wymaganych tabel i kolumn konieczne jest określenie typu danych przechowywanych w każdym z pól. Podczas tworzenia bazy danych (co zostanie przedstawione w następnym rozdziale) będzie wymagane określenie rodzaju informacji, które będą przechowywane w każdym z pól. Niemal każda baza danych opiera się na trzech ich kategoriach:

- tekst;
- liczby;
- data i czas.

W każdej z wymienionych grup wyróżnia się kilka odmian typów danych, z których pewne są charakterystyczne jedynie dla MySQL. Właściwy wybór typu dla danej kolumny wpływa nie tylko na rodzaj informacji, jakie mogą być w niej gromadzone oraz na sposób ich przechowywania, ale również na całkowitą wydajność bazy danych. Wiele typów pozwala na określenie opcjonalnego atrybutu *Długość* (nawiasy kwadratowe — `[]` — oznaczają, że pomiędzy nimi można wstawić parametr opcjonalny, tymczasem nawiasy okrągłe odpowiadają argumentom obowiązkowym).

Tabela 4.1. *Znajduje się tu większość typów numerycznych, jakich można użyć w bazach danych MySQL. Dla typów FLAT, DOUBLE oraz DECIMAL argument Długość określa maksymalną liczbę cyfr, natomiast argument Pozycje określa liczbę cyfr po kropce dziesiętnej (od MySQL 5.0.3 rozmiar kolumny DECIMAL jest określany przez wyrażenie)*

Numeryczne typy danych MySQL		
Typ	**Rozmiar**	**Opis**
`TINYINT[Długość]`	1 bajt	Liczba z zakresu od –128 do 127 lub od 0 do 255, jeżeli jest typu `UNSIGNED`.
`SMALLINT[Długość]`	2 bajty	Liczba z zakresu od –32 768 do 32 767 lub od 0 do 65 535, jeżeli jest typu `UNSIGNED`.
`MEDIUMINT[Długość]`	3 bajty	Liczba z zakresu od –8 388 608 do 8 388 607 lub od 0 do 16 777 215, jeżeli jest typu `UNSIGNED`.
`INT[Długość]`	4 bajty	Liczba z zakresu od –2 147 483 648 do 2 147 483 647 lub od 0 do 4 294 967 295, jeżeli jest typu `UNSIGNED`.
`BIGINT[Długość]`	8 bajtów	Liczba z zakresu od –9 223 372 036 854 775 808 do 9 223 372 036 854 775 807 lub od 0 do 18 446 744 073 709 551 615, jeżeli jest typu `UNSIGNED`.
`FLOAT[Długość,Pozycje]`	4 bajty	Mała wartość zmiennoprzecinkowa.
`DOUBLE[Długość,Pozycje]`	8 bajtów	Duża wartość zmiennoprzecinkowa.
`DECIMAL[Długość,Pozycje]`	Długość + 1 lub Długość + 2 bajty	Wartość typu `DOUBLE` ze stałą liczbą cyfr po przecinku.

W tabeli 4.1 wymienione są typy numeryczne. Największa różnica występuje między typami całkowitymi i zmiennoprzecinkowymi (które zawierają kropkę dziesiętną). Następne różnice łatwo zauważyć w zakresie możliwych wartości (dla liczb całkowitych) lub poziomu dokładności (dla liczb zmiennoprzecinkowych).

W tabeli 4.2 wymienione są typy tekstowe. Większość z nich różni się rozmiarem, ale kilka pozwala na przechowywanie danych binarnych zamiast ciągów znaków. Dostępne są również dwa rozszerzenia dla typów tekstowych — ENUM oraz SET — które umożliwiają definiowanie serii akceptowanych wartości w czasie tworzenia tabeli. Pole typu ENUM pozwala na użycie jednej z kilku tysięcy wartości, natomiast typu SET — na wybranie kilku z maksymalnie 64 możliwych wartości. Z typami ENUM i SET są związane dwie pułapki — typy te nie są obsługiwane w innych bazach danych i ich zastosowanie podważa zasady normalizacji.

Tabela 4.2. *Znajdują się tu najczęściej używane typy pozwalające na przechowywanie tekstu w bazie danych MySQL*

Tekstowe typy danych MySQL		
Typ	**Rozmiar**	**Opis**
CHAR[Długość]	Liczba bajtów	Pole o stałej długości; długość od 0 do 255 znaków.
VARCHAR(Długość)	Długość ciągu + 1 bajt	Pole o stałej długości; długość od 0 do 255 znaków (od MySQL 5.0.3 65 535 znaków).
TINYTEXT	Długość ciągu + 1 bajt	Ciąg tekstowy o maksymalnej długości 255 znaków.
TEXT	Długość ciągu + 2 bajty	Ciąg tekstowy o maksymalnej długości 65 536 znaków.
MEDIUMTEXT	Długość ciągu + 3 bajty	Ciąg tekstowy o maksymalnej długości 16 777 215 znaków.
LONGTEXT	Długość ciągu + 4 bajty	Ciąg tekstowy o maksymalnej długości 4 294 967 295 znaków.
BINARY[Długość]	Długość bajtów	Podobny do CHAR, ale przechowuje dane binarne.
VARBINARY[Długość]	Długość danych + 1 bajt	Podobny do VARCHAR, ale przechowuje dane binarne.
TINYBLOB	Długość danych + 1 bajt	Przechowuje dane binarne o maksymalnej długości 255 bajtów.
BLOB	Długość danych + 2 bajty	Przechowuje dane binarne o maksymalnej długości 65 535 bajtów.
MEDIUMBLOB	Długość danych + 3 bajty	Przechowuje dane binarne o maksymalnej długości 16 777 215 bajtów.
LONGBLOB	Długość danych + 4 bajty	Przechowuje dane binarne o maksymalnej długości 4 294 697 295 bajtów.
ENUM	1 lub 2 bajty	Wyliczenie, które pozwala na to, by każda kolumna posiadała jedną z kilku możliwych wartości.
SET	1,2,3,4 lub 8 bajtów	Typ podobny do ENUM z tą różnicą, że może posiadać więcej niż jedną z dopuszczalnych wartości.

Różne typy dla daty i czasu (tabela 4.3) mają własne unikalne cechy, które są udokumentowane w podręczniku i opisywane w różnych fragmentach tej książki. Zazwyczaj z tych typów korzysta się bez modyfikacji, więc nie trzeba brać pod uwagę ich zawiłości.

Tabela 4.3. *Dostępne w MySQL typy daty i czasu*

Typy danych MySQL		
Typ	**Rozmiar**	**Opis**
DATE	3 bajty	Data w formacie: RRRR-MM-DD.
DATETIME	8 bajtów	Data i czas w formacie: RRRR-MM-DD GG:MM:SS.
TIMESTAMP	4 bajty	Znacznik czasowy w formacie: GG:MM:SS; zakres wartości kończy się w roku 2037.
TIME	3 bajty	Znacznik czasowy w formacie GG:MM:SS.
YEAR	1 bajt	Rok w formacie RRRR i zakresie od 1901 do 2155.

Wybierając typ danych:

1. Określ, czy kolumna będzie przechowywała dane tekstowe, liczbowe, czy też daty.

   Zazwyczaj jest to prosty i oczywisty etap. Do przechowywania wartości liczbowych należy użyć typu numerycznego. Jeżeli kolumna może zawierać wartości inne niż numeryczne, należy użyć kolumn tekstowych.

   W przypadku danych, takich jak kody pocztowe czy sumy pieniężne, które będą przechowywane wraz z dodatkowymi znakami (np. znak myślnika czy oznaczenie waluty), używa się pól tekstowych, choć zapisanie ich jako wartości liczbowych daje lepsze rezultaty. Problem formatowania będzie rozwiązywany w innymi miejscu.

2. Wybierz dla danej kolumny odpowiedni typ z danej kategorii.

   Mając na uwadze wysoką wydajność bazy danych, warto pamiętać, że:

   - pola o stałej długości (jak `CHAR`) są zazwyczaj szybciej przetwarzane niż pola o zmiennej długości (jak `VARCHAR`), choć z drugiej strony zajmują więcej przestrzeni dyskowej — więcej informacji na ten temat zamieszczono we wskazówce;
   - rozmiar każdego z pól powinien być ograniczony do najmniejszej możliwej wartości, którą można wyznaczyć, określając największą możliwą wartość wprowadzaną do danego pola; jeżeli przykładowo największa długość nazwy towaru będzie równa 20, to dla danej kolumny powinno się wybrać typ `VARCHAR(20)`;
   - należy pamiętać, że wprowadzenie pięcioznakowego ciągu tekstowego do pola typu `CHAR(2)` spowoduje obcięcie trzech ostatnich znaków, ta prawidłowość znajduje zastosowanie we wszystkich polach — jeżeli przekroczy się maksymalny zakres dla kolumny, cześć danych zostanie utracona.

Tabela 4.4. *Innym aspektem projektowania bazy danych jest dobór optymalnego typu danych dla każdego z pól*

Baza danych finansów		
Nazwa kolumny	**Tabela**	**Typ kolumny**
Numer faktury	Faktury	`SMALLINT(4)`
Klient ID	Faktury	`SMALLINT(3)`
Data wystawienia faktury	Faktury	`TIMESTAMP`
Wartość faktury	Faktury	`DECIMAL(10,2)`
Opis faktury	Faktury	`TINYTEXT`
Termin płatności	Faktury	`DATE`
Klient ID	Klienci	`SMALLINT(3)`
Nazwa klienta	Klienci	`VARCHAR(40)`
Ulica klienta	Klienci	`VARCHAR(80)`
Miasto klienta	Klienci	`VARCHAR(30)`
Stan klienta	Klienci	`CHAR(2)`
Kod pocztowy klienta	Klienci	`MEDIUMINT(5)`
Telefon klienta	Klienci	`VARCHAR(14)`
Osoba kontaktowa	Klienci	`VARCHAR(40)`
Adres e-mail kontaktowy	Klienci	`VARCHAR(60)`
Wydatek ID	Wydatki	`SMALLINT(4)`
Kategoria wydatku ID	Wydatki	`TINYINT(3)`
Wartość wydatku	Wydatki	`DECIMAL(10,2)`
Opis wydatku	Wydatki	`TINYTEXT`
Data zapłaty	Wydatki	`DATE`
Kategoria wydatku ID	Kategorie wydatków	`TINYINT(3)`
Kategoria wydatku	Kategorie wydatków	`VARCHAR(30)`

W przypadku liczb należy się zdecydować, czy trzeba przechowywać część ułamkową. Decyzja ta dzieli nasz obszar zainteresowania na liczby całkowite i rzeczywiste. Jeżeli ważna jest dokładność matematyczna, należy użyć typu `DECIMAL`, który jest znacznie dokładniejszy niż `FLOAT` lub `DOUBLE`.

3. Ustal maksymalną długość kolumn tekstowych lub liczbowych (tabela 4.4).

   Zamiast rozpisywania się o sposobach i przyczynach takiego, a nie innego zdefiniowania wszystkich 22 przykładowych kolumn, wszystkie ich własności zestawiono w tabeli 4.4. Niektórzy programiści mogą mieć odmienne propozycje. Najistotniejsze jest jednak, aby dostosować każdy typ do rozmiarów przechowywanych informacji, zamiast korzystać zawsze z podstawowych (nieefektywnych) typów `TEXT` i `INT`.

Wskazówki

- Wiele z nazw typów posiada synonimy, np. `INT` — `INTEGER`, `DEC` — `DECIMAL` itd.
- Pole typu `TIMESTAMP` jest uaktualniane automatycznie podczas wykonywania polecenia `INSERT` czy `UPDATE`, nawet jeżeli dla danego pola nie określono żadnej wartości. W zależności od wersji MySQL, pola typu `TIMESTAMP` mają różne właściwości.
- Dostępny jest również typ `BLOB`, będący odmianą typu `TEXT`, który pozwala na przechowywanie w tabeli plików binarnych. Przykład użycia zostanie zaprezentowany w rozdziale 12. „Techniki programowania".
- W polach daty i czasu MySQL dodatkowo sprawdza poprawność podanych danych w czasie ich wstawiania. Do wersji 5.0.2 dostępna była tylko podstawowa kontrola, czy miesiąc nie jest większy niż 12 i dzień większy niż 31. Od wersji 5.0.2 dodatkowo podany dzień musi istnieć w kalendarzu, np. jeżeli zostanie podana data 2006-02-31, zwrócony zostanie błąd daty.
- Rozmiar wymagany do zapamiętania ciągu znaków o zmiennej długości zależy również od wykorzystywanego zestawu znaków, np. znaki spoza alfabetu angielskiego mogą potrzebować więcej miejsca.

CHAR a VARCHAR

Co do przewagi któregokolwiek z tych dwóch podobnych do siebie typów wciąż trwają dyskusje. Oba przechowują ciągi tekstowe i mogą być definiowane z podaniem maksymalnej jego długości. Podstawowa różnica polega na tym, że jakiekolwiek dane zapisane jako `CHAR` zawsze będą zapisywane jako ciąg tekstowy o długości określonej dla danej kolumny (wypełnienie znakami spacji). Z kolei długość ciągów tekstowych typu `VARCHAR` jest równa długości przechowywanego ciągu danych.

Wynika z tego, że:

- kolumny `VARCHAR` zajmują mniej miejsca na dysku;
- kolumny `CHAR` są przetwarzane szybciej niż `VARCHAR`, o ile nie są stosowane typy tabel InnoDB (więcej informacji na ten temat zamieszczono w podrozdziale „Wybór maszyny zapisu" w dalszej części tego rozdziału).

Trzeba przyznać, że w większości przypadków różnica w wielkości zajmowanego miejsca na dysku oraz w szybkości pomiędzy oboma typami jest niezauważalna, przez co rozważanie tego problemu nie ma szczególnego znaczenia. Co więcej, można czasami zauważyć, że pomimo zdefiniowania kolumny z użyciem jednego typu, MySQL skorzystał z drugiego.

Istnieje jeszcze jedna, mniej istotna różnica pomiędzy omawianymi typami danych — przed wersją 5.0.3 MySQL usuwał nadmiarowe znaki spacji z kolumn `CHAR` podczas pobierania danych, a z kolumn `VARCHAR` podczas ich wstawiania. Od wersji 5.0.3 kolumny VARCHAR nie mają tej właściwości i wszystkie dodatkowe spacje są przechowywane.

Jako zasadę można przyjąć korzystanie z typu VARCHAR, chyba że dana zawsze lub niemal zawsze ma taką samą długość, co czasami zdarza się w przypadku identyfikatorów produktów (SD123, PA456 itp.).

Dodatkowe charakterystyki kolumn

Deklarując typ dla kolumny, na początku wybieramy szeroki typ — liczba, tekst lub data — a następnie wskazujemy dokładniejszy typ w danej grupie. Później kolumnie można nadać dodatkową charakterystykę. Specjalny atrybut, `AUTO_INCREMENT`, jest przedstawiony we wskazówce, ale można również korzystać z innych, takich jak `UNSIGNED`, `ZEROFILL`, `NOT NULL` oraz `DEFAULT`.

Typy numeryczne mogą być określane jako `UNSIGNED`. Oznacza to, że w kolumnie takiej mogą być zapisywane tylko liczby nieujemne. W przypadku liczb całkowitych dodatkowym efektem ubocznym jest podwojenie możliwego zakresu wartości (dla liczb rzeczywistych tak nie jest). Typy numeryczne mogą być również deklarowane jako `ZEROFILL`, co oznacza, że dodatkowe miejsce jest wypełniane od lewej strony znakami zera (`ZEROFILL` automatycznie powoduje zastosowanie atrybutu `UNSIGNED`).

Każda z kolumn może być zadeklarowana jako `NOT NULL`. W przypadku tworzenia bazy danych użycie `NULL` jest jednoznaczne z poinformowaniem, że dane pole nie przechowuje żadnej wartości (może się to różnić od interpretacji wartości `NULL` w innych kontekstach). Rozwiązaniem idealnym byłoby oczywiście przypisanie każdemu rekordowi bazy danych pewnej konkretnej wartości. W rzeczywistości jednak takie sytuacje zdarzają się rzadko. Dołączając do deklaracji typu ciąg `NOT NULL` możliwe jest wymuszenie takiego ograniczenia na danym polu.

Oznaczenie AUTO_INCREMENT

Jednym z atrybutów kolumny numerycznej jest `AUTO_INCREMENT`. Jeżeli zdefiniujemy pole posiadające taką właściwość, powoduje to nadawanie przez MySQL następnej logicznej wartości z serii. Atrybut ten jest zwykle nadawany kolumnom klucza głównego, takim jak *Numer faktury* lub *Identyfikator klienta*.

Jeżeli zdefiniujemy taką kolumnę, a podczas wstawiania wartości do tabeli nie zostanie podana w tej kolumnie wartość, program wstawi do niej następną logiczną wartość. Dzięki temu pierwszy numer faktury będzie miał wartość 1, drugi 2, trzeci 3 itd. MySQL będzie nadawał te wartości automatycznie.

Niektórzy mogą zastanawiać się, co się stanie, gdy później zdecydujemy się skasować fakturę numer 3. W numeracji pojawi się „dziura", ale taka sytuacja jest całkowicie prawidłowa. Nie ma żadnego problemu z powodu tego, że kolejne numery faktur będą miały wartości 1, 2, 4, 5, 8… W rzeczywistości problemy mogłyby się pojawić, jeżeli system próbowałby „poprawić" tę sytuację.

Tworząc tabelę, można również określić wartość domyślną dla kolumny (poza typami TEXT i BLOB). Wtedy gdy duża część rekordów będzie miała taką samą zawartość, wartość domyślna pozwala uniknąć konieczności wpisywania wszystkich wartości w czasie wstawiania nowych wierszy, pod warunkiem, że wartości te są równe wartości domyślnej. Przykładem takiej deklaracji kolumny może być:

```
plec ENUM('M', 'K') DEFAULT 'K'.
```

W tabeli 4.5 wymienione są wszystkie kolumny z bazy *finanse*, wraz z ich pełną definicją.

Tabela 4.5. *Do każdej z kolumn można w razie potrzeby dodać dodatkowe atrybuty*

Baza danych finanse, zmodyfikowana		
Nazwa kolumny	**Tabela**	**Typ kolumny**
Numer faktury	Faktury	SMALLINT(4) UNSIGNED NOT NULL AUTO_INCREMENT
Klient ID	Faktury	SMALLINT(3) UNSIGNED NOT NULL
Data wystawienia faktury	Faktury	TIMESTAMP NOT NULL
Wartość faktury	Faktury	DECIMAL(10,2) UNSIGNED NOT NULL
Opis faktury	Faktury	TINYTEXT NOT NULL
Termin płatności	Faktury	DATE
Klient ID	Klienci	SMALLINT(3) UNSIGNED NOT NULL AUTO_INCREMENT
Nazwa klienta	Klienci	VARCHAR(40) NOT NULL
Ulica klienta	Klienci	VARCHAR(80) NOT NULL
Miasto klienta	Klienci	VARCHAR(30) NOT NULL
Stan klienta	Klienci	CHAR(2) NOT NULL
Kod pocztowy klienta	Klienci	MEDIUMINT(5) UNSIGNED ZEROFILL NOT NULL
Telefon klienta	Klienci	VARCHAR(14)
Osoba kontaktowa	Klienci	VARCHAR(40)
Adres e-mail	Klienci	VARCHAR(60)
Wydatek ID	Wydatki	SMALLINT(4) UNSIGNED NOT NULL AUTO_INCREMENT
Kategoria wydatku ID	Wydatki	TINYINT(3) UNSIGNED NOT NULL
Wartość wydatku	Wydatki	DECIMAL(10,2) UNSIGNED NOT NULL
Opis wydatku	Wydatki	TINYTEXT NOT NULL
Data zapłaty	Wydatki	TIMESTAMP NOT NULL
Kategoria wydatku ID	Kategorie wydatków	TIMYINT(3) UNSIGNED NOT NULL AUTO_INCREMENT
Kategoria wydatku	Kategorie wydatków	VARCHAR(30) NOT NULL

Gdy chcesz dostosować kolumny:

1. Wyszukaj te, które nie mogą zawierać wartości NULL.

   Jest to najważniejszy z dodatkowych atrybutów. Każda kolumna oznaczona jako NOT NULL zawsze musi mieć podaną wartość. Jak się okaże przy okazji dodawania rekordów, brak podanej wartości w kolumnie NOT NULL powoduje wygenerowanie błędu.

   Jako zasadę należy przyjąć definiowanie kolumny jako NOT NULL wszędzie tam, gdzie jest to możliwe.

2. Wyszukaj kolumny numeryczne, które powinny być oznaczone jako UNSIGNED.

   Jest to bardzo łatwa operacja. Jeżeli liczba, która znajdzie się w kolumnie, musi być dodatnia, tak jak cena lub ilość, kolumna powinna być oznaczona jako UNSIGNED. Jeżeli liczba może być ujemna, jak np. temperatura lub stan konta (niestety!), nie należy kolumny tak oznaczać.

3. Wyszukaj kolumny numeryczne, które powinny być oznaczone jako ZEROFILL.

   Atrybut ZEROFILL jest znacznie rzadziej wykorzystywany niż UNSIGNED, ale w niektórych przypadkach jest niezbędny. Niektóre kody zaczynają się od 0, tak jak np. 02101. Jeżeli chcielibyśmy zapisać taki kod w kolumnie całkowitej, wartość ta zostałaby zapisana jako 2101 (ponieważ początkowe zera nie mają znaczenia w liczbach całkowitych). Jeżeli zdefiniujemy kolumnę jako MEDIUMINT(5) UNSIGNED ZEROFILL, zapisany kod zachowa początkowe zero.

4. Wyszukaj kolumny, które powinny mieć wartość domyślną.

   Krok ten jest w zasadzie zależny od osobistych upodobań.

Wskazówki

- Zgodnie ze sztuką projektowania bazy danych oraz zasadami funkcjonowania MySQL klucze główne nie mogą zawierać wartości NULL.
- Jeżeli kolumna ENUM zostanie określona jako NOT NULL, wówczas pierwsza dopuszczalna wartość stanie się wartością domyślną.
- Istotnym jest, aby mieć świadomość, że NULL nie jest wartością równoznaczną z zerem, pustym ciągiem ("") czy znakiem spacji (" "), które to są znanymi *wartościami*.
- Trzeba wiedzieć, że MySQL w dziwny sposób obsługuje liczby UNSIGNED. Jeżeli wykonamy odejmowanie z co najmniej jedną liczbą UNSIGNED, wynik zawsze będzie UNSIGNED. Przez to odjęcie od wartości 2 z kolumny UNSIGNED wartości 10 z kolumny SIGNED nie da w wyniku –8.

Wprowadzenie do indeksów

Indeksy składają się na szczególny system, wykorzystywany do poprawienia całościowej wydajności bazy danych. Ustalając indeksy w ramach tabeli, wskazuje się kolumny, które są w danej tabeli ważniejsze od innych kolumn tej samej tabeli (definicja dla laików).

MySQL pozwala na utworzenie maksymalnie od 16 do 64 indeksów dla jednej tabeli, w zależności od wykorzystywanej maszyny zapisu, a każdy z nich może obejmować do 15 kolumn. Wykorzystanie indeksów wielokolumnowych nie musi się wydawać takie oczywiste, jednak stają się użyteczne w przypadku częstego przeszukiwania grupy tych samych kolumn (np. zawierających dane na temat imienia, nazwiska, miasta i województwa).

W stosowaniu indeksów wskazany jest umiar. Zwiększają one, co prawda, szybkość odczytu danych z bazy, ale spowalniają proces ich modyfikacji (z uwagi na fakt, że zmiany muszą być odwzorowane także w indeksach). Z drugiej strony, zwykle znacznie częściej odczytuje się dane z tabeli, niż wstawia nowe rekordy i zmienia istniejące.

Najlepszym zastosowaniem dla indeksów jest użycie ich w kolumnach, które:

- są często wykorzystywane w części `WHERE` zapytań;
- są często wykorzystywane w części `ORDER BY` zapytań;
- cechują się różnorodnością wartości (kolumny, w których wartości powtarzają się wielokrotnie nie powinny być indeksowane).
- są często stosowane w wyrażeniach `JOIN`.

Tabela 4.6. *W celu zwiększenia wydajności baza danych została wzbogacona o kilka (choć nie jest ich zbyt wiele) indeksów, które pozwolą jej na efektywniejsze pobieranie przechowywanych informacji*

Indeksy bazy danych finansów		
Kolumna	**Tabela**	**Typ indeksu**
Numer faktury	Faktury	`PRIMARY KEY`
Klient ID	Faktury	`INDEX`
Data wystawienia faktury	Faktury	`INDEX`
Wartość faktury	Faktury	`INDEX`
Termin płatności	Faktury	`INDEX`
Klient ID	Klienci	`PRIMARY KEY`
Nazwa klienta	Klienci	`INDEX` (lub `UNIQUE`)
Wydatek ID	Wydatki	`PRIMARY KEY`
Kategoria wydatku ID	Wydatki	`INDEX`
Wartość wydatku	Wydatki	`INDEX`
Data zapłaty	Wydatki	`INDEX`
Kategoria wydatku ID	Kategorie wydatków	`PRIMARY KEY`
Kategoria wydatku	Kategorie wydatków	`UNIQUE`

W MySQL wyróżnia się kilka typów indeksów: `INDEX`, `UNIQUE` (narzucający konieczność wprowadzania unikatowej wartości w każdym wierszu) oraz `PRIMARY KEY` (będący szczególną postacią indeksu `UNIQUE`). Dostępny jest również indeks `FULLTEXT`, który jest opisany w rozdziale 10., „SQL i MySQL dla zaawansowanych”. Propozycje indeksów dla bazy danych *finanse* zestawiono w tabeli 4.6.

W celu dodania indeksu:

1. Wyszukaj wszystkie kolumny, które powinny być oznaczone jako `PRIMARY KEY`.

   Powinno być to oczywiste szczególnie wtedy, jeżeli wykonałeś operacje normalizacji opisaną w poprzednim rozdziale. Należy pamiętać, że może istnieć tylko jeden klucz główny w tabeli (choć możliwe jest tworzenie złożonego klucza głównego składającego się z wielu kolumn).

2. Wyszukaj wszystkie kolumny, których wartości muszą być zawsze unikalne.

   Również indeks `UNIQUE` nie jest wykorzystywany zbyt często. Większość wartości — daty, liczby, nazwy, miasta, kody pocztowe — mogą się powtarzać, szczególnie w tabelach zawierających tysiące wierszy. Jednak czasami okazuje się, że kolumna musi zawierać unikalne wartości, np. adres e-mail, nazwa użytkownika (w przypadku systemów do rejestracji lub logowania) lub też pole *Kategoria wydatku* w tabeli *Kategorie wydatków*.

   Nie ma potrzeby definiowania kolumn `PRIMARY KEY` jako `UNIQUE`, ponieważ oznaczenie `PRIMARY KEY` również powoduje unikalność.

3. Wyszukaj wszystkie pozostałe kolumny, które powinny skorzystać z indeksu.

   Skorzystaj z wcześniejszych zaleceń na temat miejsc, w których warto tworzyć indeksy i pomyśl jakie dane będą odczytywane. Jeżeli potrzebna będzie lista faktur w zakresie dat lub całkowita wartość zamówienia, utwórz indeksy w logiczny sposób. Jeżeli tabela rejestracji i logowania będzie przeszukiwana z użyciem połączenia nazwy użytkownika i hasła, w taki sam sposób powinna być poindeksowana. Powinno się również indeksować klucze obce.

Wskazówka

- Indeksy stosowane w kolumnach o zmiennej długości cechuje mniejsza wydajność. Ogólnie, stosowanie pól, których długość nie jest stała, spowalnia pracę MySQL. Można to skompensować indeksując tylko część ciągu o zmiennej długości, np. pierwsze pięć lub dziesięć znaków.

Końcowy etap projektu

Ostatnim etapem projektowania bazy danych jest zastosowanie odpowiedniej konwencji nazewnictwa. Co prawda, MySQL nie narzuca zasad nazywania baz danych, tabel czy kolumn, istnieją jednak pewne sprawdzone reguły, których należy przestrzegać (zasady wymagane są zaznaczone pogrubieniem):

- **używanie znaków alfanumerycznych**;
- **nie korzystanie ze spacji**;
- **ograniczenie maksymalnej długości nazw do 64 znaków**;
- nazwy pól powinny mieć charakter opisowy;
- nazwy pól, z wyjątkiem kluczy, powinny być unikatowe w obrębie wszystkich tabel;
- nie należy korzystać ze słów kluczowych MySQL;
- używanie znaków podkreślenia (_) w celu rozdzielania wyrazów;
- korzystanie tylko z małych liter (choć nie jest to rzecz obowiązkowa);
- używanie liczby mnogiej w oznaczaniu tabel i pojedynczej w definiowaniu kolumn;
- dołączanie `id` (lub `ID`) do nazw kolumn kluczy głównych i obcych;
- umieszczanie kluczy głównych w początkowej części tabeli, a w dalszej kolejności kluczy obcych.

Zamieszczone powyżej reguły mają jedynie charakter zalecenia, ich przestrzeganie, poza koniecznością posługiwania się znakami alfanumerycznymi bez znaków spacji, nie jest zatem obowiązkowe. Część programistów preferuje używanie wielkich liter do rozdzielania wyrazów (zamiast znaku podkreślenia). Inni z kolei uwzględniają w nazwie kolumny jej typ. Najistotniejsze jest jednak to, by przestrzegać ustalonej konwencji.

Ostateczny projekt bazy danych przedstawiono w tabeli 4.7, który otrzymamy po wykonaniu kolejnych kroków.

W celu dokończenia projektowania bazy danych:

1. Określ nazwę całej bazy danych.

   Powinna być ona łatwa do zapamiętania i opisowa. Nazwa bazy danych również musi być unikalna, ponieważ na jednym serwerze MySQL nie mogą istnieć bazy danych o takich samych nazwach.

   Przykładowo w tym i poprzednim rozdziale korzystaliśmy z bazy *finanse*. Można ją również nazwać *Finanse*, ale osobiście preferuję nazwy zawierające tylko małe litery.

Tabela 4.7. *Ostatni etap projektu polega na zastosowaniu odpowiedniej konwencji nazewnictwa oraz uporządkowania kolumn w tabelach*

Baza danych finanse		
Nazwa kolumny	**Tabela**	**Typ kolumny**
faktura_id	faktury	SMALLINT(4) UNSIGNED NOT NULL AUTO_INCREMENT
klient_id	faktury	SMALLINT(3) UNSIGNED NOT NULL
data_faktury	faktury	TIMESTAMP NOT NULL
wartosc_faktury	faktury	DECIMAL(10,2) UNSIGNED NOT NULL
opis_faktury	faktury	TINYTEXT NOT NULL
data_platnosci	faktury	DATE
klient_id	klienci	SMALLINT(3) UNSIGNED NOT NULL AUTO_INCREMENT
nazwa_klienta	klienci	VARCHAR(40) NOT NULL
ulica_klienta	klienci	VARCHAR(80) NOT NULL
miasto_klienta	klienci	VARCHAR(30) NOT NULL
stan_klienta	klienci	CHAR(2) NOT NULL
kod_pocztowy_klienta	klienci	MEDIUMINT(5) UNSIGNED ZEROFILL NOT NULL
telefon_klienta	klienci	VARCHAR(14)
osoba_kontaktowa	klienci	VARCHAR(40)
email_kontaktowy	klienci	VARCHAR(60)
wydatek_id	wydatki	SMALLINT(4) UNSIGNED NOT NULL AUTO_INCREMENT
kategoria_wydatku_id	wydatki	TINYINT(3) UNSIGNED NOT NULL
wartosc_wydatku	wydatki	DECIMAL(10,2) UNSIGNED NOT NULL
opis_wydatku	wydatki	TINYTEXT NOT NULL
data_zaplaty	wydatki	TIMESTAMP NOT NULL
kategoria_wydatku_id	wydatki_kategorie	TINYINT(3) UNSIGNED NOT NULL AUTO_INCREMENT
kategoria_wydatku	wydatki_kategorie	VARCHAR(30) NOT NULL

2. Określ nazwę każdej tabeli.

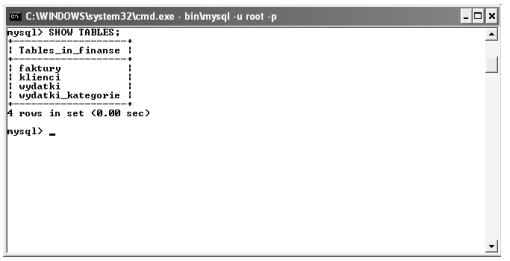
   Należy pamiętać, że powinny być one łatwe do zapamiętania i opisowe. Dodatkowo, w jednej bazie danych nie mogą istnieć dwie tabele o takich samych nazwach (tabele w różnych bazach mogą mieć te same nazwy). Zdecydowałem się na tabele `faktury`, `klienci`, `wydatki` i `wydatki _kategorie`.

3. Nadaj nazwę każdej kolumnie w każdej tabeli.

   W tym przypadku można spotkać się z wieloma wariantami, ponieważ każdy ma swój styl. Jak wspomniałem, do kolumn kluczy głównych i obcych dodałem frazę *_id*. Jeżeli w tabeli znajduje się pole z datą, w jej nazwie umieszczam frazę *data*.

4. Uporządkuj kolumny w każdej z tabel.

   Wyniki tego kroku zależą bardziej od własnej organizacji niż innych względów. Kolejność kolumn nie ma żadnego wpływu na działanie tabeli lub bazy danych. Osobiście umieszczam kolumny klucza głównego na początku, a następnie układam klucze obce.



Rysunek 4.1. *Tabele są wyświetlane w porządku alfabetycznym; można z tego skorzystać, nadając związanym tabelom podobne nazwy*

Wskazówki

- Jeżeli związanym tabelom zostaną nadane nazwy rozpoczynające się tak samo, to gdy wyświetlimy listę tabel, będą one wyświetlane razem. Przykładem są tabele `wydatki` i `wydatki_kategorie`, które na rysunku 4.1 są wyświetlane razem).
- W nazwach baz danych i tabel są rozpoznawane wielkie i małe litery w systemach typu Unix, a w Windows nie. W nazwach kolumn wielkie i małe litery są zawsze ignorowane.
- Ściśle stosując się do zasad projektowania bazy danych, minimalizujemy liczbę błędów, jakie mogą powstać przy programowaniu interfejsu bazy danych.
- Technicznie rzecz biorąc, można korzystać z istniejących słów kluczowych w nazwach tabel i kolumn. Jednak w celu odwołania się do nich zawsze należy umieszczać te nazwy we wstecznych apostrofach:

  ```
  SELECT * FROM `table`
  ```

  Jednak najlepiej nie korzystać z istniejących słów kluczowych.

Wybór maszyny zapisu

Serwer bazy danych MySQL obsługuje kilka różnych typów tabel (typ tabeli określa również typ *maszyny zapisu*). Choć każdy z typów obsługuje różny zbiór funkcji, sposób ich pracy — mówimy tu o wykonywaniu zapytań — jest właściwie spójny. W tym miejscu krótko omówię trzy główne typy tabel; wraz z postępami poznawania MySQL będziesz mógł zapoznać się ze wszystkimi szczegółami opisanymi w podręczniku.

Najważniejszym typem tabel jest *MyISAM*. Tabele tego typu doskonale nadają się dla większości aplikacji, ponieważ bardzo szybko wykonują operacje `SELECT` i `INSERT`. Jednak maszyna zapisu MyISAM nie obsługuje transakcji, o których będzie mowa w rozdziale 10.

Kolejnymi popularnymi typami tabel są *InnoDB* oraz *MEMORY* (czasami nazywany *HEAP*). Od wersji 4.0 tabele InnoDB wchodzą w skład domyślnej instalacji MySQL (jeżeli korzystasz z wcześniejszej wersji serwera, musisz włączyć obsługę InnoDB; więcej informacji na ten temat można znaleźć w podręczniku). Tabele InnoDB mogą być wykorzystywane w transakcjach i elegancko obsługują operacje `UPDATE`. Jednak maszyna InnoDB jest wolniejsza niż MyISAM i wymaga większej ilości miejsca na dysku.

Typ tabel MEMORY pozwala na najszybsze wykonywanie operacji, ponieważ takie tabele przechowują dane w pamięci, a nie na dysku. Jest to jednak okupione ograniczeniami, ponieważ tabele MEMORY obsługują tylko kolumny o stałej szerokości, nie można korzystać z atrybutu `AUTO_INCREMENT` i w czasie awarii traci się wszystkie dane.

W celu wybrania maszyny zapisu:

1. Zaloguj się do klienta *mysql*.

Aby wybrać maszynę zapisu, należy najpierw sprawdzić, jakie opcje są dostępne. W tym celu należy odpytać serwer MySQL, do którego podłączył się program *mysql* (instrukcje specyficzne dla platformy można znaleźć w rozdziale 2., „Uruchamianie MySQL").

2. Sprawdź, jakie maszyny zapisu są obsługiwane przez dany serwer MySQL, uruchamiając następujące zapytanie (rysunek 4.2):

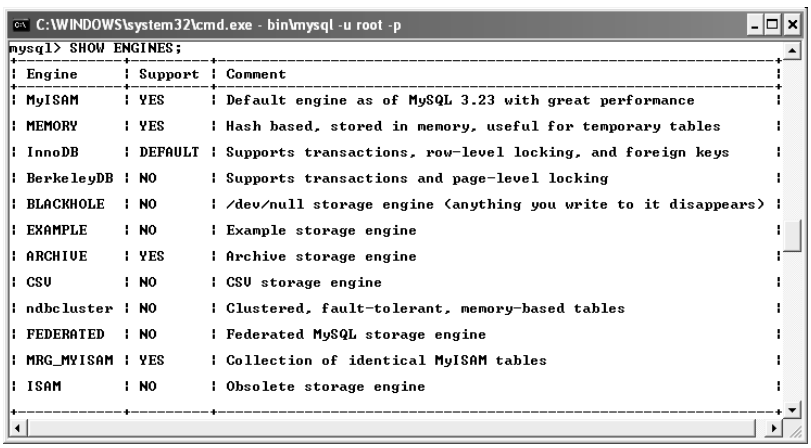
```
SHOW ENGINES;
```

Wyniki będą inne w różnych instalacjach MySQL. Krok ten warto wykonać, ponieważ nie ma sensu próbować użyć niedostępnej maszyny zapisu.

3. Określ, czy potrzebujesz transakcji.

Transakcje są bezpieczniejsze, ponieważ pozwalają wycofać zmiany i chronią dane w przypadku awarii. Jednak tabele nietransakcyjne działają szybciej oraz wymagają mniej pamięci operacyjnej i dyskowej.

Jako zasadę należy przyjąć, że jeżeli transakcje są potrzebne, należy skorzystać z InnoDB. Jeżeli nie, najlepiej wybrać MyISAM.



Rysunek 4.2. *Lista dostępnych maszyn zapisu w jednej z instalacji MySQL*

4. Określ, czy możesz poświęcić wydajność na rzecz trwałości.

   Jeżeli wybrana tabela musi być obsługiwana naprawdę szybko, to prawdopodobnie najlepszym typem będzie MEMORY. Interakcja z taką tabelą jest naprawdę bardzo szybka, ale w przypadku awarii tracimy wszystkie dane!

Wskazówki

- Ta sama baza danych może zawierać tabele różnych typów. Baza danych do obsługi handlu elektronicznego może korzystać z tabel MyISAM do przechowywania danych klientów i produktów, ale do zamówień może używać tabel InnoDB (w celu wykorzystania transakcji).
- Dwoma innymi popularnymi typami tabel są MERGE oraz BDB (*Berkeley Database*). Pierwszy z typów pozwala na traktowanie wielu tabel MyISAM jak jednej. Drugi jest alternatywą dla InnoDB i również obsługuje transakcje.
- Maszyna zapisu InnoDB została zakupiona przez firmę Oracle, konkurencyjną firmę z rynku baz danych. Oracle kupił również firmę Sleepycat Software, w której powstała maszyna zapisu BDB. Jeszcze nie wiadomo, jak to wpłynie na MySQL, ale możliwe jest, że w przyszłych wersjach obie maszyny znikną. Jest to jeden z powodów, dla którego warto sprawdzać, jakie maszyny zapisu są dostępne, wykonując zapytanie `SHOW ENGINES` (dostępne od MySQL 4.0).
- Każda maszyna zapisu posiada inne właściwości, takie jak maksymalna liczba obsługiwanych indeksów, typy kolumn, dla których można zakładać indeksy, maksymalna wielkość tabel (określana jako wielkość na serwerze) itd. Jeżeli rozpoczniesz korzystanie z MySQL na tak wysokim poziomie, konieczne będzie zapoznanie się z podręcznikiem.
- Jeżeli będziesz próbował utworzyć tabelę, korzystając z maszyny zapisu niedostępnej w danej wersji MySQL, zostanie zastosowana domyślna maszyna zapisu dla tego serwera.

Zestawy znaków i sposoby sortowania

W czasie tworzenia tabel, oprócz wyboru maszyny zapisu, należy określić zestaw znaków i sposób sortowania. Oba te parametry wpływają na obsługę tekstu przez MySQL. Są to dosyć nowe możliwości, ponieważ wyraźnie zaistniały w wersji MySQL 4.1 (były we wcześniejszych wersjach MySQL, ale nie w tak formalnej postaci).

Zestaw znaków określa sposób zapisu liter, cyfr i symboli w kolumnach tekstowych (odnosi się tylko do typów tekstowych). Aby określić zestaw znaków dla określonej kolumny, należy do definicji tej kolumny dodać frazę `CHARACTER SET nazwa_zestawu`, gdzie `nazwa_zestawu` określa wybrany zestaw znaków. W celu sprawdzenia dostępnych zestawów znaków można użyć polecenia `SHOW CHARACTER SET` w kliencie *mysql*. Domyślnym zestawem znaków jest `latin1`, który obejmuje język angielski i inne języki zachodnioeuropejskie, ale dostępne są również zestawy dla alfabetów środkowoeuropejskich, greki, cyrylicy, języka chińskiego, koreańskiego i innych. W czasie pisania tej książki MySQL obsługiwał ponad 70 zestawów znaków!

Pokrewnym pojęciem jest *sposób sortowania*, który określa zasady porównywania znaków. Przykładowo sposób sortowania bez rozróżniania wielkości znaków identycznie traktuje małe i wielkie litery. Inny może określać sposób sortowania znaków z ogonkami i kropkami w języku polskim. Każdy zestaw znaków posiada zbiór skojarzonych sposobów sortowania. Dla zestawu znaków `latin1` domyślnym sposobem sortowania jest `latin1_swedish_ci`. Choć może się to wydawać dziwne, jest to doskonałe połączenie do obsługi języka angielskiego i większości języków zachodnich.

W celu wybrania zestawu znaków i sortowania:

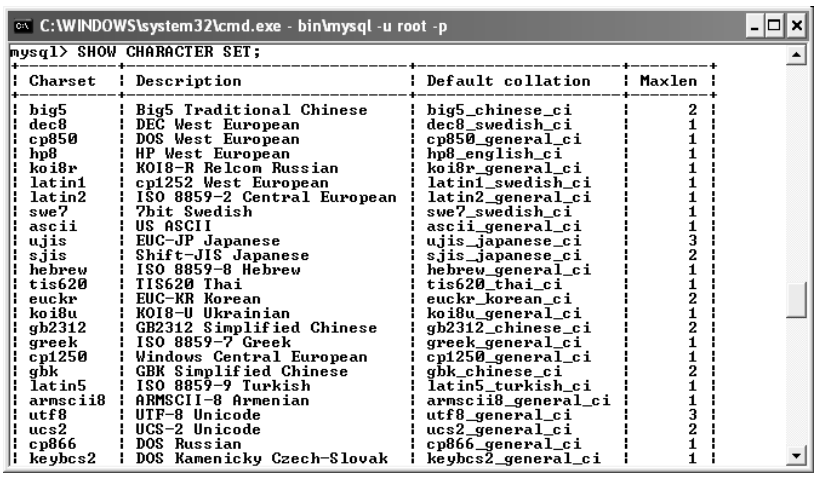
1. Zaloguj się do klienta *mysql*.

   Podobnie jak w przypadku maszyn zapisu, na początek należy sprawdzić, jakie mamy dostępne opcje. W tym celu należy odpytać serwer MySQL, do którego podłączył się program *mysql* (instrukcje specyficzne dla platformy można znaleźć w rozdziale 2.).

2. Sprawdź, jakie zestawy znaków są obsługiwane przez daną wersję MySQL, uruchamiając zapytanie (rysunek 4.3):

   ```
   SHOW CHARACTER SET;
   ```

   Wyniki będą różniły się w różnych instalacjach MySQL. W mojej instalacji MySQL 5.0.18 w systemie Windows dostępne jest 36 zestawów znaków.



```
C:\WINDOWS\system32\cmd.exe - bin\mysql -u root -p
mysql> SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |      2 |
| dec8     | DEC West European           | dec8_swedish_ci     |      1 |
| cp850    | DOS West European           | cp850_general_ci    |      1 |
| hp8      | HP West European            | hp8_english_ci      |      1 |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |      1 |
| latin1   | cp1252 West European        | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |      1 |
| swe7     | 7bit Swedish                | swe7_swedish_ci     |      1 |
| ascii    | US ASCII                    | ascii_general_ci    |      1 |
| ujis     | EUC-JP Japanese             | ujis_japanese_ci    |      3 |
| sjis     | Shift-JIS Japanese          | sjis_japanese_ci    |      2 |
| hebrew   | ISO 8859-8 Hebrew           | hebrew_general_ci   |      1 |
| tis620   | TIS620 Thai                 | tis620_thai_ci      |      1 |
| euckr    | EUC-KR Korean               | euckr_korean_ci     |      2 |
| koi8u    | KOI8-U Ukrainian            | koi8u_general_ci    |      1 |
| gb2312   | GB2312 Simplified Chinese   | gb2312_chinese_ci   |      2 |
| greek    | ISO 8859-7 Greek            | greek_general_ci    |      1 |
| cp1250   | Windows Central European    | cp1250_general_ci   |      1 |
| gbk      | GBK Simplified Chinese      | gbk_chinese_ci      |      2 |
| latin5   | ISO 8859-9 Turkish          | latin5_turkish_ci   |      1 |
| armscii8 | ARMSCII-8 Armenian          | armscii8_general_ci |      1 |
| utf8     | UTF-8 Unicode               | utf8_general_ci     |      3 |
| ucs2     | UCS-2 Unicode               | ucs2_general_ci     |      2 |
| cp866    | DOS Russian                 | cp866_general_ci    |      1 |
| keybcs2  | DOS Kamenicky Czech-Slovak  | keybcs2_general_ci  |      1 |
```

Rysunek 4.3. *Część listy dostępnych zestawów znaków*

3. Wybierz zestaw znaków do wykorzystania.

Zestaw znaków powinien opowiadać językom i typom znaków, jakie mają być zapisywane w bazie danych. Można również wybrać inny zestaw znaków dla każdej kolumny, jeżeli aplikacja wymaga, aby w jednej kolumnie tekst był zapisywany po polsku, a w drugiej po tajsku.

4. Sprawdź, jakie sposoby sortowania są dostępne dla wybranego zestawu znaków przez uruchomienie następującego zapytania (rysunek 4.4):

```
SHOW COLLATION LIKE 'latin1%'
```

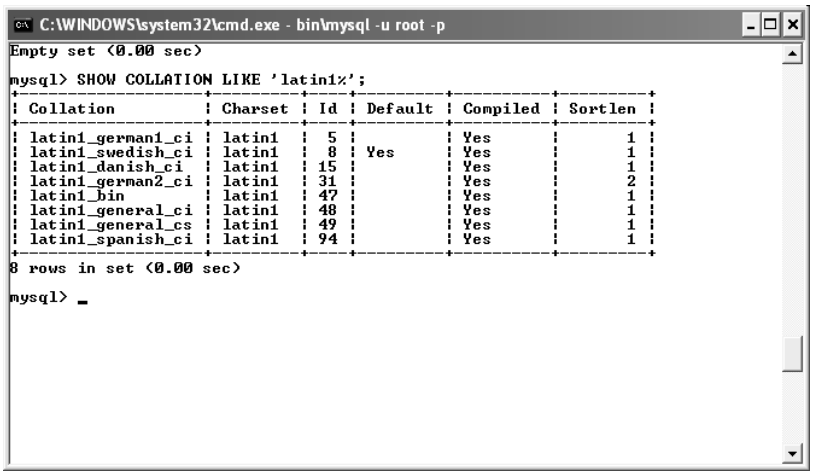
Aby znaleźć możliwe sposoby sortowania dla wybranego zestawu znaków, należy uruchomić to zapytanie. Trzeba zmienić `latin1` na używany zestaw znaków.

5. Wybierz sposób sortowania.

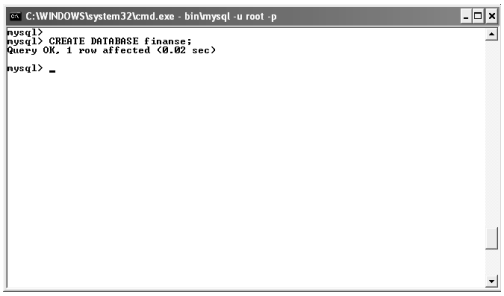
W liście wyświetlanej przez MySQL (rysunek 4.4) należy wyszukać domyślny sposób sortowania oraz sprawdzić, które sposoby są wkompilowane (czyli inaczej mówiąc, obsługiwane). Następnie, jeżeli stwierdzisz, że potrzebujesz innego sposobu niż domyślny, przeczytaj w podręczniku, jak zmienić sposób sortowania i wykonaj odpowiednią operację.

Wskazówki

- Zarówno zestaw znaków, jak i sposób sortowania mogą być ustalone na poziomie serwera, bazy danych, tabeli lub kolumny. Jeżeli nie zostanie określony zestaw znaków lub sposób sortowania dla kolumny, zostaną użyte parametry dla tabeli. To samo odnosi się do tabel (które korzystają z wartości domyślnych bazy danych) oraz baz danych (korzystających z ustawień serwera).
- Można również ustawić zestaw znaków dla sesji interakcyjnej dla połączenia z serwerem MySQL (np. gdy korzystamy z klienta *mysql*). Informacje na ten temat można znaleźć w podręczniku MySQL.
- Sposób sortowania można zmienić w zapytaniu. Ma to wpływ na porządkowanie i grupowanie wyników działania tylko tego jednego zapytania.



Rysunek 4.4. *Lista dostępnych sposobów sortowania dla jednego wybranego zestawu znaków*



Rysunek 4.5. *Tworzenie nowej bazy danych*

Tworzenie baz danych

Gdy przejdziemy przez wszystkie kroki projektowania i ostatecznego budowania projektu bazy danych (wymaga to zaskakująco dużo pracy), przyjdzie czas na utworzenie bazy danych w MySQL. W celu wykreowania bazy danych oraz tabel skorzystamy z klienta `mysql` oraz prostych instrukcji SQL.

W rozdziale 2., „Uruchamianie MySQL", szybko zdefiniowałem dwie bazy danych na potrzeby zademonstrowania, jak dodajemy użytkowników. Składnia polecenia tworzącego nową bazę danych wygląda zatem następująco:

```
CREATE DATABASE nazwa_bazy_danych
```

Aby określić zbiór znaków oraz sposób sortowania dla całej bazy danych, należy na końcu instrukcji `CREATE` dodać dodatkową klauzulę:

```
CREATE DATABASE nazwa_bazy_danych CHARACTER
↳SET nazwa_zestawu COLLATE
↳nazwa_sposobu_sort
```

W celu zademonstrowania instrukcji `CREATE` utworzymy teraz bazę danych *finanse*.

Aby utworzyć bazę danych:

1. Zaloguj się do klienta *mysql*.

   Jeżeli jeszcze nie wiesz, jak to zrobić, potrzebne wskazówki możesz znaleźć w rozdziale 2., „Uruchamianie MySQL". Należy się zalogować jako użytkownik mający uprawnienia do tworzenia nowych baz danych.

2. Utwórz i wybierz nową bazę danych (rysunek 4.5).

   ```
   CREATE DATABASE finanse;
   ```

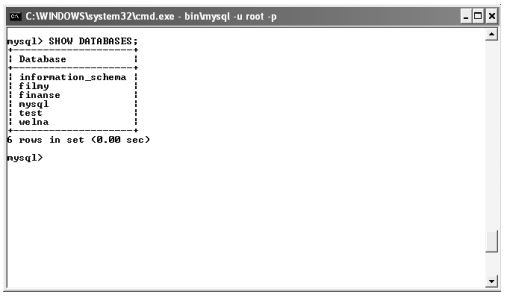
   Jak już wcześniej widzieliśmy, ten jeden wiersz kodu powoduje utworzenie bazy danych (zakładamy, że jesteśmy zalogowani do *mysql* jako użytkownik z uprawnieniami do tworzenia baz danych).

Dodatkowo, choć SQL nie rozróżnia wielkości liter, przyzwyczaiłem się do pisania słów kluczowych SQL wielkimi literami, co pomaga oddzielić je od nazw baz danych, tabel i kolumn. Jeżeli nie chcesz pisać tych słów wielkimi literami, nie jest to obowiązkowe.

3. Potwierdź istnienie bazy danych (rysunek 4.6):

```
SHOW DATABASES;
```

Polecenie SHOW, które nie musi być szczegółowo omawiane, wyświetla listę baz danych, do których ma dostęp zalogowany użytkownik.



Rysunek 4.6. *Nowo utworzona baza finanse jest wyświetlana w liście baz danych*

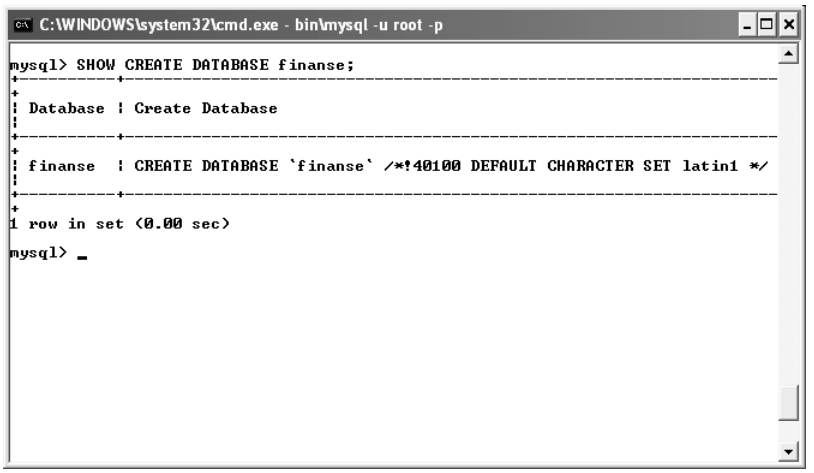
Wskazówki

- Bazy danych można również tworzyć, korzystając z aplikacji *mysqladmin*. Za jej pomocą nie można jednak kreować tabel.

```
mysqladmin -u root -p create
↳nazwa_bazy_danych
```

- Przy użyciu polecenia SHOW CREATE można zawsze sprawdzić, w jaki sposób została utworzona istniejąca baza danych (rysunek 4.7):

```
SHOW CREATE DATABASE nazwa_bazy_danych
```



Rysunek 4.7. *Zapytanie SHOW CREATE pokazuje sposób, w jaki została utworzona baza danych lub tabela. Zwróćmy uwagę, że w tym przykładzie wyświetlany jest komentarz dla wersji 4.01.00 (i wyższych) odnoszący się do domyślnego zestawu znaków*

Tworzenie tabel

Gdy mamy już bazę danych, można rozpocząć tworzenie tabel za pomocą polecenia CREATE:

```
CREATE TABLE nazwa_tabeli (
nazwa_kolumny1 opis,
nazwa_kolumny2 opis…
)
```

Łatwo zauważyć, że po podaniu nazwy tabeli należy w nawiasie kolejno zdefiniować kolumny. Każda kolumna i opis powinny być oddzielone przecinkiem. Jeżeli chcemy od razu utworzyć indeksy, na końcu instrukcji tworzenia można dodać odpowiednią klauzulę (można również utworzyć indeksy później).

```
CREATE TABLE nazwa_tabeli (
nazwa_kolumny1 opis,
nazwa_kolumny2 opis,
typ_indeksu(kolumny)
)
```

Jeżeli chcemy nazwać tworzone indeksy, należy zmienić tę część zapytania na:

```
nazwa_indeksu (kolumny)
```

Aby w czasie tworzenia tabeli określić maszynę zapisu, należy na końcu instrukcji tworzącej tabelę dodać klauzulę:

```
CREATE TABLE nazwa_tabeli (
nazwa_kolumny1 opis,
nazwa_kolumny2 opis…
) ENGINE = INNODB
```

W wersjach wcześniejszych niż 4.0.18 należało korzystać ze słowa kluczowego TYPE zamiast ENGINE. Jeżeli w czasie tworzenia tabel nie zostanie podana maszyna zapisu, MySQL użyje domyślnego typu tabel (InnoDB w Windows, w innych systemach MyISAM).

Aby określić zestaw znaków lub sposób sortowania dla całej tabeli, należy na końcu zapytania CREATE dodać odpowiednią klauzulę:

```
CREATE TABLE nazwa_tabeli (
nazwa_kolumny1 opis,
nazwa_kolumny2 opis…
) ENGINE = MyISAM CHARACTER SET
↳nazwa_zestawu COLLATE
nazwa_sposobu_sortowania
```

Teraz utwórzmy cztery tabele składające się na bazę danych *finanse*.

W celu utworzenia tabel:

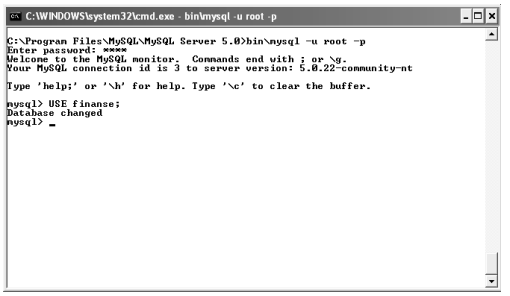
1. Otwórz klienta *mysql* i wybierz bazę finanse (rysunek 4.8).

   ```
   USE finanse;
   ```

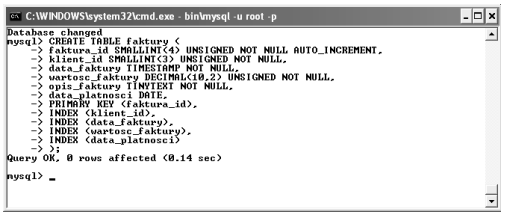
   Tworzenie tabel będzie łatwiejsze, jeżeli wcześniej zostanie wybrana baza danych. Trzeba zalogować się jako użytkownik z uprawnieniami do tworzenia tabel w tej bazie danych.

2. Utwórz tabelę faktury (rysunek 4.9).

   ```
   CREATE TABLE faktury (
   faktura_id SMALLINT(4) UNSIGNED NOT
   ↳NULL AUTO_INCREMENT,
   klient_id SMALLINT(3) UNSIGNED NOT NULL,
   data_faktury TIMESTAMP NOT NULL,
   wartosc_faktury DECIMAL(10,2) UNSIGNED
   ↳NOT NULL,
   opis_faktury TINYTEXT NOT NULL,
   data_platnosci DATE,
   PRIMARY KEY (faktura_id),
   INDEX (klient_id),
   INDEX (data_faktury),
   INDEX (wartosc_faktury),
   INDEX (data_platnosci)
   );
   ```



Rysunek 4.8. *Pierwszymi wykonanymi operacjami jest zalogowanie się do mysql i wybranie bazy finanse*



Rysunek 4.9. *Klient mysql pozwala na wprowadzanie poleceń zajmujących kilka linii, co sprawia, że dłuższe zapytania SQL są czytelniejsze*

Rysunek 4.10. *MySQL informuje o poprawnym wykonaniu polecenia za pomocą komunikatu Query OK*

Przedstawione polecenie CREATE wykorzystuje dane dotyczące tabeli *faktury*, które zostały opracowane we wcześniejszej części rozdziału. Kolejność wprowadzania kolumn determinuje sposób ich rozmieszczenia w tabeli. Po utworzeniu samych kolumn określono również związane z nimi indeksy, dzięki temu będą one istniały natychmiast po utworzeniu tabeli.

Z uwagi na fakt, że monitor *mysql* nie prześle zapytania, zanim nie napotka znaku średnika, wprowadzane polecenie może zajmować kilka linii, podobnie jak ma to miejsce w sytuacji zaprezentowanej na rysunku 4.9.

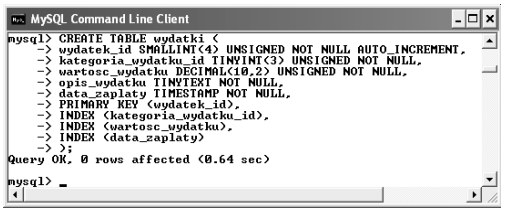
3. Utwórz tabelę klienci (rysunek 4.10).

```
CREATE TABLE klienci (
klient_id SMALLINT(3) UNSIGNED NOT
↳NULL AUTO_INCREMENT,
nazwa_klienta VARCHAR(40) NOT NULL,
ulica_klienta VARCHAR(80) NOT NULL,
miasto_klienta VARCHAR(30) NOT NULL,
stan_klienta VARCHAR(2) NOT NULL,
kod_pocztowy_klienta MEDIUMINT(5)
↳UNSIGNED ZEROFILL NOT NULL,
telefon_klienta VARCHAR(14),
osoba_kontaktowa VARCHAR(40),
email_kontaktowy VARCHAR(60),
PRIMARY KEY (klient_id),
INDEX (nazwa_klienta)
);
```

Tabela ta posiada więcej kolumn, ale tylko dwa indeksy.

4. Utwórz tabelę wydatki (rysunek 4.11).

```
CREATE TABLE wydatki (
wydatek_id SMALLINT(4) UNSIGNED NOT
↳NULL AUTO_INCREMENT,
kategoria_wydatku_id TINYINT(3)
↳UNSIGNED NOT NULL,
wartosc_wydatku DECIMAL(10,2)
↳UNSIGNED NOT NULL,
opis_wydatku TINYTEXT NOT NULL,
data_zaplaty TIMESTAMP NOT NULL,
PRIMARY KEY (wydatek_id),
INDEX (kategoria_wydatku_id),
INDEX (wartosc_wydatku),
INDEX (data_zaplaty)
);
```



Rysunek 4.11. *Tworzenie trzeciej tabeli*

5. Na koniec utwórz tabelę wydatki_kategorie (rysunek 4.12).

```
CREATE TABLE wydatki_kategorie (
kategoria_wydatku_id TINYINT(3)
↳UNSIGNED NOT NULL AUTO_INCREMENT,
kategoria_wydatku VARCHAR(30) NOT NULL,
PRIMARY KEY (kategoria_wydatku_id),
UNIQUE (kategoria_wydatku)
);
```

To najprostsza z przedstawionych czterech tabel, ponieważ posiada tylko dwie kolumny i dwa indeksy.

Rysunek 4.12. *Tworzenie czwartej i ostatniej tabeli*

6. Sprawdź, czy tabele zostały utworzone (rysunek 4.13).

```
SHOW TABLES;
SHOW COLUMNS FROM faktury;
```

Polecenie `SHOW` (ang. *pokaż*) powoduje wyświetlenie tabel bazy danych lub nazw kolumn tabeli i ich typów.

Wskazówki

- Wymieniane w innych publikacjach polecenie `DESCRIBE nazwa_tabeli` jest tożsame z instrukcją `SHOW COLUMNS FROM nazwa_tabeli`.
- Można również wykonać zapytanie `SHOW CREATE TABLE nazwa_tabeli`, aby sprawdzić, za pomocą jakiego polecenia została utworzona tabela.
- Jeżeli wykonamy zapytanie `SHOW CREATE TABLE nazwa_tabeli` po utworzeniu tabeli, możemy zweryfikować sposób, w jaki MySQL implementuje nasze wyrażenie. Operacja ta pozwala sprawdzić, jak MySQL zmienia typy kolumn ze względów wydajnościowych.


```
C:\WINDOWS\system32\cmd.exe - bin\mysql -u root -p
6 rows in set (0.03 sec)

mysql> SHOW TABLES;
+--------------------+
| Tables_in_finanse  |
+--------------------+
| faktury            |
| klienci            |
| wydatki            |
| wydatki_kategorie  |
+--------------------+
4 rows in set (0.03 sec)

mysql> SHOW COLUMNS FROM faktury;
+-----------------+--------------------------+------+-----+-------------------+----------------+
| Field           | Type                     | Null | Key | Default           | Extra          |
+-----------------+--------------------------+------+-----+-------------------+----------------+
| faktura_id      | smallint(4) unsigned     | NO   | PRI | NULL              | auto_increment |
| klient_id       | smallint(3) unsigned     | NO   | MUL | NULL              |                |
| data_faktury    | timestamp                | YES  | MUL | CURRENT_TIMESTAMP |                |
| wartosc_faktury | decimal(10,2) unsigned   | NO   | MUL | NULL              |                |
| opis_faktury    | tinytext                 | NO   |     | NULL              |                |
| data_platnosci  | date                     | YES  | MUL | NULL              |                |
+-----------------+--------------------------+------+-----+-------------------+----------------+
6 rows in set (0.09 sec)

mysql> _
```

Rysunek 4.13. *Gdy chcesz sprawdzić istnienie i strukturę bazy danych, należy posłużyć się poleceniem SHOW. Nie daj się zaskoczyć przez wyniki działania tego polecenia. MySQL posiada własny sposób opisu tabeli, który może różnić się od tego, w jaki sposób została ona utworzona*

Modyfikacja tabel

Ostatnim tematem, jaki przedstawimy w tym rozdziale, jest modyfikacja istniejących tabel. Można to wykonywać z dowolnych powodów, ale przed wprowadzeniem zmian należy pomyśleć o wszystkich zasadach normalizacji, indeksach, konwencjach nazewnictwa i tego typu elementach. Bardzo łatwo zaprzepaścić całą pracę włożoną w planowanie przez wprowadzenie do bazy danych „niewielkiej poprawki".

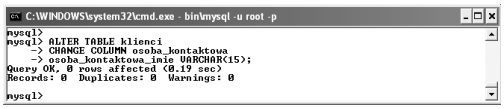
Podstawową instrukcją SQL, służącą do modyfikowania struktury tabel w bazie danych, jest `ALTER`. Zazwyczaj wiąże się ona z procesem dodawania, usuwania lub przekształcania kolumn, choć pozwala również na zmianę nazwy tabeli oraz zmianę kluczy i indeksów. Umożliwia ona również zmianę nazwy całej tabeli i modyfikowanie indeksów. Podstawową składnią `ALTER` jest:

```
ALTER TABLE nazwa_tabeli KLAUZULE
```

Ze względu na mnogość dostępnych klauzul zostały one zestawione w tabeli 4.8. Pełna ich lista znajduje się w dodatku B, „Przegląd SQL i MySQL".

Tabela 4.8. *Polecenie SQL ALTER można wykorzystywać do modyfikowania tabeli na wiele sposobów*

Klauzule ALTER TABLE		
Klauzula	**Użycie**	**Znaczenie**
`ADD COLUMN`	`ALTER TABLE nazwa_tabeli ADD COLUMN nazwa_kolumny typ_kolumny`	Umieszcza na końcu tabeli nową kolumnę.
`CHANGE COLUMN`	`ALTER TABLE nazwa_tabeli CHANGE COLUMN nazwa_kolumny nowa_nazwa_kolumny nowy_typ_kolumny`	Pozwala na zmianę typu i właściwości kolumny.
`DROP COLUMN`	`ALTER TABLE nazwa_tabeli DROP COLUMN nazwa_kolumny`	Usuwa kolumnę z tabeli wraz z jej danymi.
`ADD INDEX`	`ALTER TABLE nazwa_tabeli ADD INDEX nazwa_indeksu (nazwa_kolumny)`	Dodaje nowy indeks dla danej kolumny.
`DROP INDEX`	`ALTER TABLE nazwa_tabeli DROP INDEX nazwa_indeksu`	Usuwa istniejący indeks.
`RENAME AS`	`ALTER TABLE nazwa_tabeli RENAME AS nowa_nazwa_tabeli`	Zmienia nazwę tabeli.



Rysunek 4.14. *Do zmiany nazwy lub typu kolumny służy polecenie ALTER TABLE nazwa_kolumny CHANGE COLUMN*

W celu zademonstrowania zasad posługiwania się poleceniem `ALTER` pole *osoba_kontaktowa* tabeli *klienci* zostanie rozdzielone na dwie kolumny (zgodne z założeniami normalizacji) *osoba_kontaktowa_imie* i *osoba_kontaktowa_nazwisko*. W tym przykładzie zakładam, że w tabeli nie ma jeszcze żadnych danych. Jeżeli w tabeli znajdowałyby się dane, trzeba by wziąć to pod uwagę (dodać dwie kolumny, przenieść do nich istniejące dane i skasować kolumnę źródłową). Ponieważ polecenie `ALTER` powoduje w tabeli znaczne zmiany, powinno się zawsze wykonać kopię tabeli przed rozpoczęciem jej modyfikacji (rozdział 13.).

Gdy chcesz zmienić strukturę tabeli:

1. Otwórz klienta *mysql* i wybierz bazę *finanse*, o ile wcześniej tego nie zrobiłeś.

```
USE finanse;
```

2. Zmień nazwę pola *osoba_kontaktowa* (rysunek 4.14).

```
ALTER TABLE klienci CHANGE COLUMN
↳osoba_kontaktowa osoba_kontaktowa_imie
↳VARCHAR(15);
```

   Powyższe polecenie zmienia jedynie nazwę i typ danych kolumny *osoba_kontaktowa*. Od tej chwili nazwą pola jest *osoba_kontaktowa_imię* a typem danych w nim przechowywanych — `VARCHAR(15)`. Wszystkie dane przechowywane w kolumnie pozostały w niej, lecz zostały skrócone do 15 znaków.

Kasowanie tabel i baz danych

Aby usunąć tabelę lub bazę danych, należy skorzystać z polecenia `DROP`. Użycie tego polecenia jest bardzo proste:

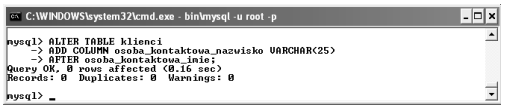
```
DROP DATABASE nazwa_bazy_danych;
DROP TABLE nazwa_tabeli;
```

Oczywiście, jeżeli skasujemy tabelę, wszystkie zapisane w niej dane zostaną utracone. Gdy skasujemy bazę danych, wszystkie tabele przejdą do historii.

3. Utwórz nową kolumnę *osoba_kontaktowa_nazwisko* (rysunek 4.15).

```
ALTER TABLE klienci ADD COLUMN
↳osoba_kontaktowa_nazwisko VARCHAR(25)
↳AFTER osoba_kontaktowa_imie;
```

Od tej chwili tabela posiada jeszcze jedną kolumnę, w której nie ma aktualnie żadnych wartości. Dodając kolumnę do tabeli, można skorzystać z klauzuli `AFTER nazwa_kolumny`, aby wskazać, w którym miejscu tabeli powinna być umieszczona nowa kolumna.



Rysunek 4.15. *ALTER TABLE nazwa_tabeli ADD COLUMN dodaje nową kolumnę do tabeli*

4. Potwierdź wprowadzenie zmian w strukturze tabeli (rysunek 4.16).

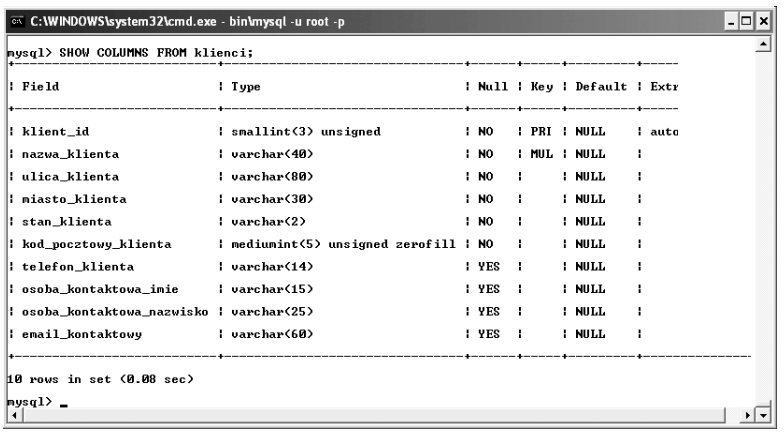
```
SHOW COLUMNS FROM klienci;
```

Wskazówki

- Aby zmienić typ istniejącej tabeli — jest to oczywiście dozwolone — należy skorzystać z następującego polecenia `ALTER`:

```
ALTER TABLE nazwa_tabeli ENGINE = MYISAM
```

- Można również potwierdzić wprowadzenie zmian w strukturze tabeli przez użycie polecenia `SHOW CREATE TABLE nazwa_tabeli`. Jak zauważymy po uruchomieniu tego zapytania, nie będzie ono pokazywało oryginalnego polecenia `CREATE`, ale raczej polecenie `CREATE`, jakie spowoduje utworzenie tabeli w takiej postaci jak obecnie.



Rysunek 4.16. *Potwierdzenie zmian w strukturze tabeli przez wykonanie polecenia SHOW COLUMNS*