Wprowadzenie do systemu MySQL

Spis treści

1 Czym jest MySQL?

MySQL jest szybkim i solidnym systemem zarządzania relacyjnymi bazami danych (*Relational Database Management System* - RDBMS). Bazy danych pozwalają na efektywne przechowywanie, wyszukiwanie, sortowanie oraz pozyskiwanie informacji. Serwer MySQL steruje dostępem do danych, udostępnia je równocześnie dla wielu użytkowników oraz zapewnia możliwość jak najszybszego z nich korzystania. Ponadto serwer steruje uwierzytelnianiem użytkowników, które pozwala operować na danych jedynie osobom upoważnionym. Jest to system wieloużytkownikowy i wielowątkowy korzystający ze standardu - języka SQL, dzięki któremu możliwe jest zadawanie zapytań do bazy danych. MySQL został oficjalnie opublikowany w roku 1996. Jest również dostępny na licencji Open Source, lecz jeśli jest to konieczne, można zakupić jego wersję komercyjną.

2 Niektóre zalety MySQL

Główni konkurenci MySQL to PostgreSQL, Microsoft SQL Server oraz Oracle. MySQL wyróżnia się przede wszystkim następującymi cechami:

- **Wydajność**. MySQL jest bardzo szybki, pod adresem `http://web.mysql.com/benchmark.html` dostępne są testy wydajności, z których łatwo wynika powyższy wniosek.
- **Niskie koszty użytkowania**. Podobnie jak PHP i inne produkty Open Source MySQL dostępny jest za darmo razem z kodem źródłowym.
- **Łatwość użycia**. MySQL jest łatwy w konfiguracji. A ponadto jeśli używaliśmy innego systemu RDBMS, przejście na MySQL nie sprawi nam żadnych problemów.
- **Wieloplatformowość i przenośność**. MySQL działa na różnych platformach uniksowych, a także pod kontrolą systemu Microsoft Windows.

3 Instalacja serwera MySQL

W pierwszej kolejności musimy pobrać wersję systemu na odpowiednią platformę z adresu `http://www.mysql.com`.

3.1 Instalacja na platformie MS Windows

Instalacja serwera MySQL. Po pobraniu archiwum MySQL rozpakowujemy go w dowolne miejsce na naszym dysku, następnie wchodzimy do katalogu gdzie rozpakowaliśmy pliki i uruchamiamy program `setup.exe`. Przywita nas program instalacyjny bazy danych MySQL oraz informacje o programie, klikamy "Next" dopóki program nie zapyta nas o ścieżkę w jakiej chcemy zainstalować MySQL. Domyślnie wpisana jest ścieżka `C:\mysql` lecz jeśli chcemy możemy zainstalować nasz serwer innym miejscu. Po wpisaniu ścieżki klikamy "Next" i program da nam do wyboru 3 możliwości instalacyjne:

- **Typical** - program zainstaluje się z najczęściej używany opcjami i tą opcję zalecamy wybrać;
- **Compact** - program zainstaluje minimalną ilość pakietów;
- **Custom** - mamy możliwość indywidualnego wyboru pakietów do zainstalowania.

Wybieramy "Next" i czekamy aż instalator przekopiuje wszystkie pliki. Teraz otwieramy katalog `C:\mysql\bin` i uruchamiamy program `winmysqladmin.exe`. Przy pierwszym uruchomieniu tego programu zapyta nas o wpisanie naszego loginu i hasła jakie będą chroniły naszą bazę przed niepowołanymi użytkownikami. Podajemy te dane i w pasku zadań powinna nam się pokazać ikona semafora z zielonym światłem, oznacza to że nasz serwer działa. Nazwa serwera MySQL dla Windows NT/2000 to mysql-nt. Chcąc zainstalować serwer jako usługę, należy wydać poniższe polecenie:

```
C:\mysql\bin\mysql-nt -install
```

Od tej chwili możemy uruchamiać i zatrzymywać tę usługę, korzystając z poleceń:

```
NET START mysql
NET STOP mysql
```

3.2 Instalacja na platformie Linux

Na początku rozpakowujemy źródła i przechodzimy do utworzonego w ten sposób katalogu:

```
#tar zxvf mysql-4.xx.xx.tar.gz
#cd mysql-4.xx.xx
```

Następnie konfigurujemy serwer MySQL. Służy do tego standardowy skrypt configure. Wszystkie dostępne opcje konfiguracji obejrzymy wpisując `./configure --help`. W naszym przypadku wystarczy następująca konfiguracja

```
#./configure --prefix=/usr/local/mysql
```

Następnie kompilujemy niezbędne binaria instrukcją `make`. Ostatnim krokiem jest instalacja w określonym katalogu, w tym celu wpisujemy `make install`. Teraz musimy już tylko stworzyć standardowe bazy danych systemy MySQL i uruchomić serwer:

```
#scripts/mysql_install_db
#cd /usr/local/mysql/bin
#./safe_mysql &
```

Pozostało jeszcze ustawić hasło administratora systemu MySQL:

```
#/usr/local/mysql/bin/mysqladmin -u root password 'nowe_hasło'
```

4 Relacyjne Bazy Danych. Pojęcia podstawowe.

Przez **bazę danych** rozumiemy, w pewnym uproszczeniu, uporządkowany zbiór danych, natomiast przez **system bazy danych** - bazę danych wraz z oprogramowaniem umożliwiającym operowanie na niej. Można powiedzieć, że baza danych jest abstrakcyjnym, informatycznym modelem wybranego fragmentu rzeczywistości.

Podstawową strukturą danych w tym przypadku jest relacja będąca podzbiorem iloczynu kartezjańskiego dwóch wybranych zbiorów reprezentujących dopuszczalne wartości. W bazach danych relacja przedstawiana jest w postaci tabeli. Relacja jest zbiorem krotek posiadających taką samą strukturę, lecz różne wartości. Każda krotka odpowiada jednemu wierszowi tablicy i posiada co najmniej jeden atrybut odpowiadający pojedynczej kolumnie tablicy. Każda relacja (tablica) posiada następujące własności:

- krotki (wiersze) są unikalne;
- atrybuty (kolumny) są również unikalne;
- kolejność atrybutów nie ma znaczenia;
- kolumny mają atomowe wartości.

Poniżej omówimy bardziej szczegółowo poszczególne aspekty relacyjnych systemów bazodanowych.

4.1 Tabele

Są podstawowym elementem relacyjnych baz danych, określane również jako relacje. Tabela jest niczym innym jak tabelarycznym zbiorem danych. Jeśli korzystaliśmy kiedykolwiek z arkusza kalkulacyjnego, to nie powinniśmy mieć problemów z wyobrażeniem sobie jak taka relacja wygląda.

Poniżej przedstawiony został prosty przykład, jest to tabela zawierająca imiona, nazwiska oraz adresy klientów pewnej księgarni. Nazwijmy tę tabelę Customers

CustomerID	**Name**	**Address**	**City**
1	Jan Rybak	Banacha 23	Łódź
2	Urszula Nowak	Sienkiewicza 234/45	Warszawa
3	Katarzyna Kowalska	Kossaka 113	Pabianice

4.2 Kolumny

Określane są również mianem atrybutów lub pól. Każdej kolumnie przyporządkowana jest niepowtarzalna nazwa oraz pewien fragment danych. Ponadto z każdą kolumną skojarzony jest określony typ danych. Przyglądając się powyższej tabeli zauważymy, że typem kolumny *CustomerID* są liczby całkowite (*integer*), natomiast pozostałe kolumny są typu łańcuchowego (*string*).

4.3 Wiersze

Każdy wiersz tabeli reprezentuje jednego klienta. Wszyscy klienci posiadają identyczne pola. Wiersze nazywane są często krotkami lub rekordami.

4.4 Wartości

Każdy wiersz zawiera cały zestaw indywidualnych wartości, które odpowiadają poszczególnym kolumnom tabeli. Typ danych określonej wartości musi być zgodny z typem danych kolumny, w której znajduje się ta wartość.

4.5 Klucze

Musimy dysponować sposobem. który umożliwiłby jednoznaczną identyfikację każdego klienta. Dane osobowe niezbyt dobrze się do tego celu nadają, ponieważ wielokrotnie spotykamy osoby o tych samych nazwiskach i imionach.

Określoną osobę możemy odróżnić od innych na kilka sposobów. Możemy raczej być pewni, że pod jednym adresem mieszka tylko jedna osoba o danym imieniu i nazwisku. Jednak określenie typu "Jan Rybak zamieszkały w Łodzi na ulicy Banacha 23" ze względu na swą długość i złożony charakter, nie jest dobrym rozwiązaniem. Ponadto taki sposób identyfikacji wymagałby posłużenia się danymi z kilku kolumn.

W naszym przypadku problem ten skutecznie rozwiązaliśmy dzięki przypisaniu każdemu klientowi niepowtarzalnego identyfikatora *CustomerID*. Takie numery jednoznacznie identyfikujące każdy wiersz tabeli, upraszczają proces zarówno przechowywania, jak i wyszukiwania danych w bazie.

Kolumna, która zawiera dane jednoznacznie identyfikujące wiersze tabeli, określana jest mianem klucza lub klucza głównego (*primary key*). Klucz może się składać z kilku kolumn. Gdybyśmy mimo wszystko zdecydowali się identyfikować klientów w następujący sposób: typu "Jan Rybak zamieszkały w Łodzi na ulicy Banacha 23", wówczas klucz musiałby składać się z kolumn Name, Address oraz City (co i tak nie daje gwarancji niepowtarzalności).

Zazwyczaj bazy danych składają się z kilku tabel, które powiązane są ze sobą właśnie za pomocą kluczy. Jeśli do powyższej tabeli dodamy kolejną przechowującą zamówienia złożone przez klientów Orders, jak na rysunku poniżej,

OrderID	**CustomerID**	**Amount**	**Date**
1	2	88	2004/05/22
2	1	33	2004/04/13
3	2	21	2004/08/12

to widzimy, jak użyteczne mogą być klucze. Każdy wiersz tabeli *Orders* reprezentuje jedno zamówienie, złożone przez jednego klienta. Wiemy kim jest dany klient, ponieważ w tabeli zamówień przechowujemy jego niepowtarzalny identyfikator *CustomerID*.

W świecie relacyjnych baz danych numer klienta, znajdujący się w tabeli innej niż tabela klientów, określany jest mianem klucza obcego (*foreign key*). Tak więc identyfikator *CustomerID* jest kluczem głównym w tabeli *Customers*, lecz ten sam identyfikator pojawiający się w innej tabeli - na przykład *Orders* - jest kluczem obcym.

Być może nasuwa nam się pytanie, po co w powyższym przykładzie zostały użyte aż dwie tabele, skoro cały adres klienta można by zapisać w tabeli z zamówieniami. Za chwilę wyjaśnimy, czym podyktowany jest taki a nie inny sposób postępowania.

4.6 Schematy

Pełny zestaw tabel składających się na kompletną bazę danych, określany jest mianem schematu (*schema*) bazy danych. W skład schematu powinny zatem wchodzić tabele, które z

kolei zbudowane są z kolumn charakteryzujących się określonymi typami danych. Jedna kolumna w każdej tabeli powinna reprezentować klucz główny tej tabeli, a ponadto mogą się tam znajdować kolumny kluczy obcych. Struktura taka może być przedstawiona w poniższej postaci:

Customers (CustomerID, Name, Address, City)

Orders (OrderID, CustomerID, Amount, Date)

Podkreślone linią ciągłą nazwy pól reprezentują klucze główne, natomiast nazwy podkreślone linią przerywaną są kluczami obcymi w danej tabeli.

4.7 Relacje

Klucze reprezentują powiązania, jakie występują pomiędzy danymi, znajdującymi się w dwóch tabelach. W teorii baz danych powiązania te są określane mianem relacji (*relationship*). Na przykład pomiędzy tabelami *Orders* i *Customers* istnieje powiązanie wyznaczone przez relację, jaka zachodzi pomiędzy wierszem tabeli *Orders* i *Customers*.

W relacyjnej bazie danych możemy wyróżnić trzy podstawowe typy relacji. Różnice pomiędzy nimi dotyczą ilości danych, które mogą znaleźć się po obu stronach relacji. Te trzy rodzaje relacji noszą nazwy **jeden-do-jednego**, **jeden-do-wielu**, **wiele-do-wielu**.

Po obu stronach relacji **jeden-do-jednego** może wystąpić tylko jeden rekord. Gdybyśmy na przykład umieścili adresy klientów w odrębnej tabeli, to pomiędzy tabelą klientów, a tabelą adresów zachodziłaby relacja **jeden-do-jednego** (jednemu rekordowi z tabeli klientów mógłby być przyporządkowany wyłącznie jeden rekord z tabeli adresów). Musielibyśmy wówczas dysponować kluczem obcym, znajdującym się bądź to w tabeli adresów bądź klientów, który jednoznacznie identyfikowałby wiersz z drugiej tabeli (przy takim typie relacji nie jest wymagane istnienie kluczy obcych w obu łączonych tabelach).

W relacji **jeden-do-wielu**, pojedynczy wiersz pierwszej tabeli może być powiązany z wieloma wierszami drugiej tabeli. Taki typ relacji występuje w naszym przykładzie tabel klientów i zamówień. Jeden klient może złożyć wiele zamówień. W tabeli, która występuje po stronie "wiele" tej relacji, musi znajdować się klucz obcy, który jednoznacznie zidentyfikuje wiersz umieszczony w tabeli występującej po stronie "jeden". W powyższym przykładzie tym kluczem obcym w tabeli *Orders* jest kolumna o nazwie *CustomerID*.

W relacji **wiele-do-wielu**, wiele wierszy jednej tabeli powiązanych jest z wieloma wierszami drugiej. Gdybyśmy mieli dwie tabele o nazwach *Books* i *Authors*, to normalną byłaby sytuacja, w której jedna książka stanowiłaby dzieło kilku współautorów, ale też każdy z nich mógłby napisać jeszcze inne książki (zarówno samodzielnie, jak też wspólnie z innymi autorami). W przypadku występowania tego typu relacji zazwyczaj stosuje się dodatkową tabelę, rozbijającą relację **wiele-do-wielu** na dwie relacje **jeden-do-wielu**. Obok tabel *Books* i *Au-*

thors moglibyśmy utworzyć tabelę *Books_ Authors*, która zawierałaby jedynie występujące w parach klucze obce, identyfikujące rekordy w pozostałych dwóch tabelach. Takie rozwiązanie pozwalałoby przyporządkować jednemu autorowi wszystkie napisane przez niego książki, a także każdej książce wszystkich jej autorów.

5 Praca z systemem MySQL

5.1 Logowanie do systemu

Na początku musimy się zalogować do systemy MySQL (zakładamy, że masz już założone konto, jeśli nie patrz dalej jak to zrobić) Ogólna składnia polecenia `mysql` jest następująca:

```
#mysql [-h adres_hosta] [-u nazwa_użytkownika] [-ptwoje_haslo]
```

Wszystkie argumenty są opcjonalne i po kolei oznaczają:

`adres_hosta` - serwer, na którym zainstalowany jest system, domyślna wartość to `localhost`

`nazwa_użytkownika` - login użytkownika w systemie MySQL,

`twoje_haslo` - hasło w systemie MySQL, nie musimy go podawać, lepiej zostawić to pole puste, a system sam zapyta nas o hasło.

Warto pamiętać o poleceniu `mysqladmin`, które pomaga w zarządzaniu systemem MySQL. Jeśli chcemy poznać ustawienia naszego systemu musimy wywołać polecenie:

```
#mysqladmin variables
```

lub

```
#mysqladmin -u root -p variables [>plik.txt]
```

jeśli mamy w systemie ograniczony dostęp (dane możemy przekierować do pliku `plik.txt` - co ułatwi ich przeglądanie).

Jeśli chcemy zmienić hasło, to wywołamy

```
#mysqladmin -u username [-p] password new_password
```

Przy czym jeśli użytkownik `username` nie używał wcześniej hasła nie piszemy `-p`, a `new_password` oznacza ciąg znaków, który jest nowym hasłem. Używając tego polecenia możemy stworzyć nowego użytkownika:

```
#mysqladmin -u new_user password new_password
```

bez żadnych uprawnień w systemie. Hasło można zmienić również po zalogowaniu do systemu poleceniem:

```
set password for username=password('new_password');
```

Polecenie to zmieni hasło dla użytkownika `username` (o ile mamy uprawnienie), na `new_password` i zapisze je w odpowiedniej tabeli po zakodowaniu funkcją `password()`. Więcej możliwości polecenia `mysqladmin` możemy zobaczyć po wpisaniu:

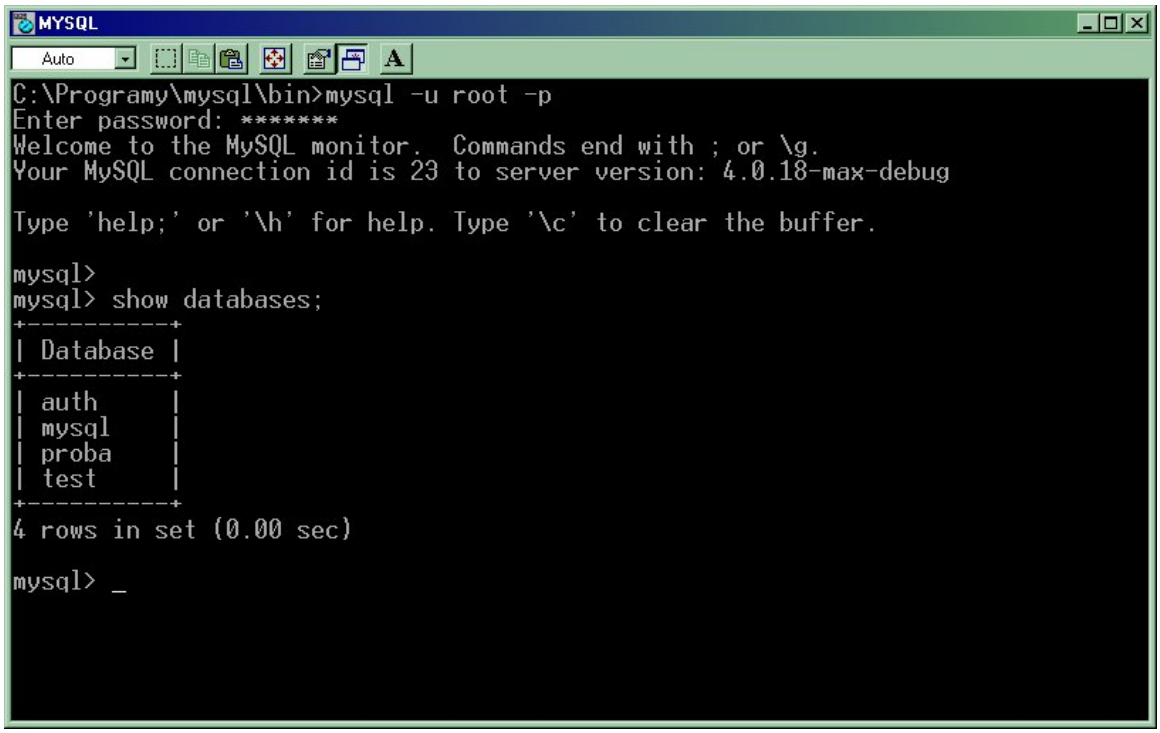
```
#mysqladmin -help
```

5.2 Wyświetlanie istniejących baz danych

Po zalogowaniu możemy wyświetlić istniejące w systemie bazy danych:

```
show databases;
```

pamiętajmy o średniku na końcu każdego polecenia i o tym, że zawsze możemy wyjść z systemu używając polecenia `quit`.



5.3 Tworzenie nowej bazy danych

Zanim stworzymy nową bazę danych, musimy wiedzieć, że w systemie MySQL są 4 rodzaje identyfikatorów, odpowiednio dla:

- baz danych,

- tabel,
- kolumn,
- aliasów.

Fizycznie bazy danych w MySQL reprezentowane są przez katalogi, a tabele przez pliki. Ich lokalizacja znajduje się w zmiennej systemowej `datadir` (na ogół jest to `/var/lib/mysql`). Poszczególne identyfikatory powinny spełniać następującą składnię:

Typ	**Maks. długość**	**Rozróżnianie wielkości znaków**	**Dopuszczalne znaki**
bazy danych	64	tak samo jak w systemie plików	wszystkie znaki, które system dopuszcza w nazwie katalogu, za wyjątkiem znaku `/`
tabela	64	jak wyżej	jak wyżej, za wyjątkiem znaków `/` i `.`
kolumna	64	nie	wszystkie znaki
alias	255	nie	wszystkie znaki

Teraz możemy już stworzyć bazę danych:

```
creat database nazwa_nowej_bazy;
```

5.4 Usuwanie baz danych

Aby usunąć bazę danych używamy polecenia:

```
drop database nazwa_bazy_do_usuniecia;
```

5.5 Praca z wybraną bazą danych

Aby możliwe było zarządzanie tabelami w ramach konkretnej bazy, musimy poinformować system MySQL, o którą bazę danych nam chodzi. Możemy to zrobić na dwa sposoby:

- podając ścieżkę do tabeli w notacji `<nazwa_bazy>.<nazwa_tabeli>`,
- wywołując komendę `use <nazwa_bazy>`. Wtedy nie musimy już podawać przed nazwą tabeli nazwy bazy, ponieważ domyślnie to w niej MySQL szuka podanej tabeli.

Aby wybrać bazę `proba` wywołamy polecenie:

```
use proba;
```

5.6 Tworzenie tabel

Aby stworzyć nową tabelę używamy następującej składni:

```
create table nazwa_tabeli
(
nazwa_kolumny_1 opis_kolumny_1,
nazwa_kolumny_2 opis_kolumny_2,
...
nazwa_kolumny_n opis_kolumny_n,
primary key(nazwa_kolumny_s,nazwa_kolumny_r))
```

W opisie kolumn obok ich typów (które omówimy za chwilę) mogą pojawić się słowa kluczowe:

- `primary key` - jeśli kluczem jest jedna kolumna, to wskazujemy ją umieszczając te słowa w opisie (wtedy nie musimy używać tych słów na końcu definicji tablicy). Klucz główny jest sposobem umożliwiającym jednoznaczną identyfikację każdego rekordu, dlatego dane w kolumnach (lub kolumnie) będących kluczem głównym muszą być niepowtarzalne. System MySQL automatycznie indeksuje kolumny klucza głównego. Indeks zapewnia bezpośredni dostęp do wierszy w tabeli w celu zredukowania czasu wykonywania operacji. Indeks zawiera informację o każdej wartości, która jest zapisana w indeksowanej kolumnie. Indeks może tworzyć (jeśli nie jest on tworzony automatycznie) właściciel tabeli, użytkownik posiadający uprawnienie `INDEX` dla danej tabeli lub administrator systemu.

- `not null` - wszystkie rekordy w tabeli muszą nieć w danym polu jakąś wartość.

- `auto_increment` - używane w odniesieniu do kolumn liczbowych. Jeśli podczas wypełniania danych, pominiemy wartość pola z tymi słowami, to system MySQL automatycznie umieści w tym polu niepowtarzalną wartość liczbową. Będzie ona o jeden większa od maksymalnej wartości jaka znajduje się w tej kolumnie. W każdej tabeli może być tylko jedna kolumna o tej właściwości. Ponadto kolumny z tymi słowami kluczowymi są indeksowane.

- `default` - umożliwia nadanie wartości domyślnej, będzie ona użyta w sytuacji, gdy zostawimy to pole puste.

Poniżej zestawiamy typy danych dostępne w systemie MySQL:

Całkowite typy danych:

Typ	**Zakres**	**Zajmowana pamięć (w bajtach)**
smallint	$-32768...32768$	2
unsigned	$0...65535$	
int integer	$-2^{31}...2^{31}-1$	4
unsigned	$0...2^{32}-1$	
bigint	$-2^{63}...2^{63}$	8
unsigned	$0...2^{64}$	

Typy zmiennoprzecinkowe:

Typ	**Zakres**	**Pamięć (w bajtach)**	**Opis**
float[(M,D)]	$\pm 1.175E-38$ $\pm 3.402E+38$	4	Liczby zmiennoprzecinkowe pojedynczej precyzji. Możliwość określenia ilości znaków M i miejsc po przecinku D.
double[(M,D)] real[(M,D)] precision[(M,D)]	$\pm 1.179E-308$ $\pm 2.225E+308$	8	Liczby zmiennoprzecinkowe podwójnej precyzji. Możliwość określenia ilości znaków M i miejsc po przecinku D.

Typy przechowujące daty i godziny:

Typ	**Zakres**	**Opis**
date	1000-01-01 9999-12-31	Data w postaci YYYY-MM-DD.
time	-838:59:59 838:59:59	Godzina w formacie HH:MM:SS
datetime	1000-01-01 00:00:00 9999-12-31 23:59:59	Data i godzina w formacie YYYY-MM-DD HH:MM:SS
timestamp	1970-01-01 00:00:00 do roku 2037	Znacznik czasowy przydatny przy przetwarzaniu transakcyjnym

Typy łańcuchowe:

Typ	**Zakres**	**Opis**
char[(M)]	od 1 do 255 znaków	Łańcuchy o ustalonej długości M. Dane są uzupełniane spacjami (o ile to potrzebne) do podanej długości M. Poprawia to wydajność kosztem zajmowanego miejsca.
varchar[(M)]	od 1 do 255 znaków	Łańcuchy i zmiennej długości, nie są uzupełniane spacjami

Pola typu `blob` i `text`.

Typ	**Zakres**
tinyblob tinytext	255
blob text	$2^{16} - 1 = 65535$
longblob longtext	$2^{32} - 1$

W polach typu `blob` (*Binary Large Object*) można przechowywać dowolne dane, na przykład grafikę lub dźwięki. W praktyce pola `blob` i `text` są takie same, z tym wyjątkiem, że w drugim z owych typów jest rozróżniana wielkość znaków, a w pierwszym nie.

Typy `enum` i `set`

Typ	**Zakres**	**Opis**
enum('value1','value2',...)	65535	w kolumnach tego typu może znajdować się wyłącznie jedna z podanych wartości lub wartość `null`
set('value1','value2',...)	64	w kolumnach tego typu może znajdować się zestaw podanych wartości lub wartość `null`

Na przykład, następującej tabeli możemy używać do przechowywania adresów:

```
create table adresy (
   id int unsigned not null auto_increment primary key,
   imie char(15) default '<podaj imie>',
   nazwisko char(25) default '<podaj nazwisko>',
   ulica char(40) not null,
   miasto char(20) not null,
   );
```

5.7 Przeglądanie struktury bazy danych

Po wybraniu bazy danych, możemy wyświetlić listę dostępnych tabel. Służy do tego polecenie:

```
show tables;
```

Serwer MySQL wyświetli nam listę tabel w aktualnie wybranej bazie danych:

```
+------------------+
|Tables in proba   |
+------------------+
|adresy            |
+------------------+
1 row in set (0.06 sec)
```

Aby obejrzeć strukturę określonej tabeli, możemy użyć polecenia `describe`. Oto jego wynik dla tabeli `adresy`:

```
+---------+------------+----------+----------+----------------+----------+
|Field    |Type        |Null      |Key       |Default         |Extra     |
+---------+------------+----------+----------+----------------+----------+
|id       |int unsigned|          |PRI       |                |          |
|imie     |char(15)    |YES       |          |<podaj imie>    |          |
|nazwisko |char(25)    |YES       |          |<podaj nazwisko>|          |
|ulica    |char(40)    |          |          |                |          |
|miasto   |char(20)    |          |          |                |          |
|telefon  |char(10)    |          |          |NULL            |          |
+---------+------------+----------+----------+----------------+----------+
6 rows in set (0.07 sec)
```

5.8 Wstawianie danych do tabel

W dalszym ciągu korzystać będziemy z następujących tabel:

```
create table customers
( customerid int unsigned not null auto_increment primary key,
  name char(30) not null,
  address char(40) not null,
  city char(20) not null);

create table orders
( orderid int unsigned not null auto_increment primary key,
  customerid int unsigned not null,
  amount float(6,2),
  date date not null);
```

```
create table books
(  isbn char(13) not null primary key,
   author char(30),
   title char(60),
   price float(4,2));

create table order_items
( orderid int unsigned not null,
  isbn char(13) not null,
  quantity tinyint unsigned,
  primary key (orderid, isbn));

create table book_reviews
( isbn char(13) not null primary key,
  review text);
```

Ten układ tabel może służyć do prowadzenia internetowej księgarni.

Chcąc wprowadzić rekordy do bazy danych, używamy instrukcji `INSERT`. Jej typowa składnia ma postać:

```
insert [into] table [(column1,column2,...,columnk)] values
(value1,value2,...,valuek);
```

Aby wprowadzić klienta do tablicy `customers`, możemy użyć składni:

```
insert into customers values
  (NULL, "Julie Smith", "25 Oak Street", "Airport West");
```

Wartości przekazane w instrukcji zostaną wprowadzone do bazy w kolejności ich podania. W sytuacji gdybyśmy chcieli wypełnić tylko niektóre kolumny, lub jeślibyśmy chcieli wprowadzić dane w innej kolejności, wówczas musimy podać opcjonalną część instrukcji podając nazwy kolumn, do których mają trafić dane. Spójrzmy na przykład:

```
insert into customers (name, city) values
("Jan Kowalski", "Koluszki");
```

Taki sam efekt osiągnęlibyśmy używając składni:

```
insert into customers
set name="Jan Kowalski",
city="Koluszki";
```

Warto zwrócić uwagę na jeszcze jedną rzecz. Przekazując dane o Julie Smith, do kolumny `customerid` wprowadziliśmy wartość `NULL`, natomiast w przypadku Jana Kowalskiego, pole to pominęliśmy. Kolumna ta tworzy klucz główny, więc dane zapisywane w niej nie mogą być puste. W definicji tej kolumny użyliśmy jednak klauzuli `auto_increment`. Oznacza ona, że jeśli wstawiamy do tej tabeli rekord, który dla tego pola będzie pusty lub będzie zawierał wartość `NULL`, to system MySQL automatycznie umieści w nim wartość o jeden większą od największej wartości jaka do tej pory znajduje się w tej kolumnie.

Przy pomocy instrukcji `insert` można wstawiać jednocześnie wiele rekordów. W takiej sytuacji każdy powinien być umieszczony w odrębnej parze nawiasów okrągłych i oddzielony od innych przecinkiem.

Wprowadźmy do naszej bazy danych następujące dane:

```
use proba;

insert into customers values
  (NULL, "Julie Smith", "25 Oak Street", "Airport West"),
  (NULL, "Alan Wong", "1/47 Haines Avenue", "Box Hill"),
  (NULL, "Michelle Arthur", "357 North Road", "Yarraville"),
  (NULL, "Jan Kowalski", "Narutowicza 54", "Łódź"),
  (NULL, "Zbigniew Barszcz","Robotnicza 167", "Starachowice");

insert into orders values
  (NULL, 3, 69.98, "02-Apr-2000"),
  (NULL, 1, 49.99, "15-Apr-2000"),
  (NULL, 2, 74.98, "19-Apr-2000"),
  (NULL, 3, 24.99, "01-May-2000");

insert into books values
  ("0-672-31697-8", "Michael Morgan",
  "Java 2 for Professional Developers", 34.99),
  ("0-672-31745-1", "Thomas Down",
  "Installing Debian GNU/Linux", 24.99),
  ("0-672-31509-2", "Pruitt, et al.",
  "Teach Yourself GIMP in 24 Hours", 24.99),
  ("0-672-31769-9", "Thomas Schenk",
  "Caldera OpenLinux System Administration Unleashed", 49.99),
  ("83-87150-42-8", "Luke Welling, Laura Thompson",
```

```
  "PHP i MySQL Programowanie sieci Web", 97.99),
  ("83-87216-99-2","Ellen Siever",
  "Linux Podręcznik użytkownika",72.40);

insert into order_items values
  (1, "0-672-31697-8", 2),
  (2, "0-672-31769-9", 1),
  (3, "0-672-31769-9", 1),
  (3, "0-672-31509-2", 1),
  (4, "0-672-31745-1", 3);

insert into book_reviews values
  ("0-672-31697-8", "Morgan's book is clearly written and goes well beyond
   most of the basic Java books out there.");
```

Aby te dane wprowadzić do systemy MySQL, możemy zapisać je w postaci skryptu SQL-owego `insert.sql` i wykonać następująco:

```
>mysql -u user -p <insert.sql
```

5.9 Pobieranie danych z baz danych

Aby pobrać dane z bazy, używamy instrukcji `select`. Oto podstawowa jej składnia:

```
select items
from tables
[where condition]
[group by group_type]
[having where_definition]
[order by order_type]
[limit limit_criteria];
```

Omówimy po kolei różne składnie polecenia `select`. Zaczniemy od pobrania nazw klientów i ich miast.

```
select name, city
form customers;
```

Jeśli chcemy pobrać wszystkie kolumny z tabeli `order_items`, to możemy użyć składni:

```
select * from order_items;
```

5.9.1 Pobieranie danych spełniających dodatkowe warunki

Aby pobrać jedynie pewien podzbiór rekordów, musimy określić kryteria selekcji. Służy do tego klauzula `where`. Spójrzmy na przykład:

```
select * form orders
where customerid=3;
```

Poniżej zestawiamy listę najczęściej używanych operatorów w klauzuli `where`:

Operator	Przykład	Opis
=	customerid = 3	Testuje, czy wartości są równe
>	amount > 60.00	Testuje, czy wartość jest większa
<	amount < 60.00	Testuje, czy wartość jest mniejsza
>=	amount >= 60.00	Testuje, czy wartość jest większa lub równa
<=	amount <= 60.00	Testuje, czy wartość jest mniejsza lub równa
!= lub <>	quantity != 0	Testuje, czy wartość jest różna
is not null	address is not null	Testuje, czy pole zawiera jakąś wartość
is null	address is null	Testuje, czy pole jest puste
between	amount between 0 and 60.00	Testuje, czy wartość jest większa lub równa od pierwszej liczby i większa lub równa od drugiej
in	city in ('London','Moscow')	Testuje, czy wartość jest w określonym zestawie
not in	city not in ('London','Moscow')	Testuje, czy nie znajduje się w określonym zestawie
like	name like "Fred%"	Testuje, czy wartość w polu pasuje do wzorca SQL
not like	name like "Fred%"	Testuje, czy wartość w polu nie pasuje do wzorca SQL
regexp	name REGEXP "^fo$";	Testuje, czy wartość jest zgodna z POSIX-owym wyrażeniem regularnym

Operator `like` wymaga podania wzorca SQL, na który składa się prosty tekst lub znaki wieloznaczne:

- `%` (procent) - oznaczający dowolną ilość dowolnych znaków,
- `_` (podkreślenie) - zastępujący jeden dowolny znak.

Warto pamiętać, że MySQL nie rozróżnia wielkości znaków we wzorcach.

Słowo kluczowe `regexp` umożliwia używanie do porównań POSIX-owych wyrażeń regularnych. Więcej na ten temat możemy znaleźć w poradnikach "Kurs języka JavaScript" i "Kurs PHP".

W zapytaniu `select` z klauzulą `where` możemy również używać operatorów logicznych `or` i `and`, co dodatkowo poszerza jego możliwości:

```
select * form orders
where customerid = 3 or customerid = 4;
```

5.9.2 Pobieranie danych z kilku tabel

Często się zdarza, że dane, które chcemy pobrać jednocześnie, znajdują się w kilku tabelach. W takiej sytuacji musimy złączyć te tabele w oparciu o relacje między nimi zachodzące.

Proste złączenie dwóch tabel. Zamówienia złożone przez Julie Smith:

```
select orders.orderid, orders.amount, orders.date
from customers, orders
where customers.name='Julie Smith'
and customers.customerid=orders.customerid;
```

Wynik tego polecenia to:

```
+---------+--------+------------+
| orderid | amount | date       |
+---------+--------+------------+
|       2 |  49.99 | 0000-00-00 |
+---------+--------+------------+
1 row in set (0.00 sec)
```

Po pierwsze określiliśmy nazwy tabel, które łączymy, w tym wypadku są to: `customers, orders`. Po drugie podaliśmy typ złączenia umieszczając między nazwami tabel przecinek. Jest on równoważny zapisowi `inner join` lub `cross join`. Jest to tzw. złączenie pełne (ang. *full join*) i można je opisać następująco: "weź wszystkie tabele i połącz je w jedną tabelę. Rekordy tej dużej tabeli powinny reprezentować każdą możliwą kombinację rekordów pochodzących z każdej tabeli składowej. Raczej nie ma sensu wyświetlanie całej takiej złączonej tabeli, dlatego w naszym przypadku dodaliśmy klauzulę `where`. Tym samym przekształciliśmy złączenie pełne na równo-złączenie (ang. *equi-join*). Warto zwrócić uwagę na notację kropkową, dzięki której system jest w stanie odróżnić, z której tabeli pochodzą określone kolumny.

Łączenie więcej niż dwu tabel. Jak odszukać klientów, którzy złożyli przynajmniej jedno zamówienie dotyczące książki o Javie? Oto konstrukcja zapytania:

```
select customers.name
 from customers, orders, order_items, books
 where customers.customerid=orders.customerid
 and orders.orderid=order_items.orderid
 and order_items.isbn=books.isbn
 and books.title like "%Java%";
```

A oto wynik tego zapytania:

```
 +-----------------+
 | name            |
 +-----------------+
 | Michelle Arthur |
 +-----------------+
 1 row in set (0.01 sec)
```

Złączenie lewostronne (left join) Jeżeli podczas takiego złączenia okaże się, ze w tabeli występującej po prawej stronie złączenia nie ma rekordów odpowiadających tabeli umieszczonej po stronie lewej, wówczas takie rekordy z tabeli zostaną uwzględnione w wyniku, z tym, ze w odpowiednich kolumnach tabeli wynikowej wystąpią wartości `null`.

Kto nie złożył żadnych zamówień?

```
select customers.customerid, customers.name, orders.orderid
 from customers left join orders
 on customers.customerid=orders.customerid;
```

I wynik polecenia:

```
 +------------+------------------+---------+
 | customerid | name             | orderid |
 +------------+------------------+---------+
 |          1 | Julie Smith      |       2 |
 |          2 | Alan Wong        |       3 |
 |          3 | Michelle Arthur  |       1 |
 |          3 | Michelle Arthur  |       4 |
 |          4 | Jan Kowalski     |    NULL |
 |          5 | Zbigniew Barszcz |    NULL |
 +------------+------------------+---------+
 6 rows in set (0.00 sec)
```

Albo inaczej:

```
select customers.customerid, customers.name
 from customers left join orders
 using (customerid)
 where orders.orderid is null;
```

I wynik:

```
 +------------+------------------+
 | customerid | name             |
 +------------+------------------+
 |          4 | Jan Kowalski     |
 |          5 | Zbigniew Barszcz |
 +------------+------------------+
 2 rows in set (0.00 sec)
```

5.9.3 Korzystanie z aliasów nazw tabel

Często dla wygody (np. aby skrócić zapytanie) lub z konieczności używamy zastępczych nazw tabel, tzn. *aliasów*. Rozważmy wcześniejsze zapytanie, zapisane z użyciem nazw zastępczych:

```
select c.name
 from customers as c, orders as o, order_items as oi, books as b
 where c.customerid=o.customerid
 and o.orderid=oi.orderid
 and oi.isbn=b.isbn
 and b.title like "%Java%";
```

Aliasy musimy stosować w przypadku, gdy tabelę łączmy z sobą. Gdybyśmy chcieli odszukać klientów mieszkających w tym samym mieście, wówczas musiellibyśmy nadać tabeli `customers` dwa różne aliasy:

```
select c1.name, c2.name, c1.city
 from customers as c1, customers as c2
 where c1.city=c2.city
 and c1.name != c2.name;
```

5.9.4 Pobieranie danych w określonym porządku

Chcąc wyświetlić dane pobrane przez zapytanie w określonym porządku, musimy skorzystać z klauzuli `order by` polecenia `select`.

```
select name, address
 from customers
 order by name;
```

To samo zwróci:

```
select name, address
 from customers
 order by name asc;
```

Aby wyświetlić dane w odwrotnym porządku (posortowane malejąco), należy użyć słowa kluczowego `desc`:

```
select name, address
 from customers
 order by name desc;
```

5.9.5 Agregowanie danych z zapytań

Często chcemy poznać średnią wartość w jakiejś kolumnie lub wartości ekstremalne. Do tego służą funkcje agregujące. Należy pamiętać, że używamy ich zawsze do grupy rekordów.

Oto lista najczęściej używanych funkcji agregujących systemu MySQL:

Nazwa	**Opis**
`avg(column)`	Wylicza wartość średnią w kolumnie
`count(items)`	Jeśli podamy nazwę nazwę kolumny, to funkcja zwróci ilość różnych od `NULL` wartości, jakie znajdują się w tej kolumnie. Jeśli przed nazwą kolumny umieścimy słowo `distinc`, to dostaniemy ilość różniących się od siebie wartości występujących w kolumnie. Wywołanie `count(*)` wyświetli ilość rekordów w tabeli, niezależnie od tego czy pola tych rekordów mają wartości `NULL` czy też nie.
`min(column)`	Podaje minimalną wartość w kolumnie
`max(column)`	Podaje maksymalną wartość w kolumnie
`std(column)`	Wylicza standardowe odchylenie w podanej kolumnie
`sum(column)`	Liczy sumę wartości w kolumnie

5.9.6 Porcjowanie zwracanych danych

Dzięki klauzuli `limit` możemy określić, ile wierszy ma być zwróconych w zapytaniu. Użycie tej dyrektywy wymaga podania dwóch argumentów: początkowego rekordu i ilości zwracanych rekordów.

```
select name
from customers
limit 0,3
```

A oto wynik zapytania:

```
+-----------------+
| name            |
+-----------------+
| Julie Smith     |
| Alan Wong       |
| Michelle Arthur |
+-----------------+
3 rows in set (0.00 sec)
```

Zapytanie to należy rozumieć następująco: "Wybierz pole `name` z tabeli `customers`, a następnie zwróć 3 rekordy począwszy od pierwszego". Należy pamiętać, że numeracja w systemie MySQL zaczyna się od zera, stąd rekord pierwszy ma numer 0.

5.10 Aktualizowanie rekordów w bazie danych

Zadanie to możemy wykonać za pomocą zapytania `update`. A oto najczęściej stosowana postać tego polecenia:

```
update tablename
set column1=expression1,column2=expression2,...
[where condiotion]
[limit number]
```

Podobnie jak poprzednio, dyrektywy `where` i `limit` pozwalają zawęzić działanie polecenia do określonych rekordów, np. `limit` umożliwia podanie konkretnej ilości zmienianych rekordów.

Jeśli chcemy zwiększyć cenę książek o 10% to użyjemy składni:

```
update books
set price=price*1.1;
```

A jeśli chcemy zmienić adres jednej osoby w tabeli `customers`, to możemy to zrobić następująco:

```
update customers
set address = "345 West Road"
where customerid = 4;
```

5.11 Usuwanie rekordów z bazy danych

Aby skasować wiersze z tabeli możemy użyć instrukcji `delete`. Najczęściej używana składnia polecenia ma postać:

```
delete from table
[where condition]
[limit number]
```

W przypadku, gdy użyjemy składni:

```
delete from table;
```

to skasujemy wszystkie rekordy z tabeli `table`! Aby usunąć klienta, użyjemy składni:

```
delete form customers
where customerid = 5;
```

W tym wypadku dyrektywa `limit` działa tak samo jak w przypadku modyfikacji rekordów.

5.12 Modyfikacja struktury tabeli

Aby zmodyfikować strukturę tabeli musimy się posłużyć instrukcją `alter`. Podstawowa jej składnia wygląda następująco:

```
alter table tabename alternation1 [, alternation2 ...]
```

Poniżej zestawiamy listę możliwych zmian przy pomocy polecenia `alter`:

Składnia	**Opis**
`add [column] column_description [first | after column]`	Dodaje nową kolumnę w określonej lokalizacji (jeśli jej nie podamy, to nowa kolumna będzie dodana na końcu). Symbol `column_description` reprezentuje pełen opis nowej kolumny (łącznie z nazwą).
`add [column] (column_description1 column_description2,...)`	Dodaje jedną lub więcej nowych kolumn, umieszczając je za ostatnią istniejącą kolumną tabeli.

Składnia	Opis
`add index [index_name] (column,...)`	Dodaje do tabeli indeks, zbudowany w oparciu o podane kolumny lub kolumnę.
`add primary key (column,...)`	Tworzy z określonej kolumny lub grupy kolumn klucz główny.
`alter [column] col_name {set default literal | drop default}`	Dodaj lub usuwa wartość domyślną dla określonej kolumny.
`change [column] old_col_name create_definition`	Zmienia kolumne reprezentowaną przez `old_col_name` przypisując jej nowy opis. W ten sposób można zmienić również nazwę kolumny.
`modify [column] create_definition`	Działanie podobne do `change`. Nie można jednak w ten sposób zmienić nazwy kolumny.
`drop [column] col_name`	Usuwa kolumnę o podanej nazwie.
`drop primary key`	Usuwa indeks główny.
`drop index index_name`	Usuwa indeks o podanej nazwie.
`rename [to] new_tbl_name`	Zmienia nazwę tabeli.

Jeśli to poprzednio utworzonej tabeli `adresy` chcielibyśmy dodać pole `adres_email`, to użyjemy składni:

```
alter table adresy
add adres_email varchar(60);
```

Aby zmienić nazwę pola `adres_email` na `email` wywołamy polecenie:

```
alter table adresy
change adres_email email varchar(60);
```

Zmieńmy maksymalną długość pola `email` z 60 znaków na 80:

```
alter table adresy
modify email varchar(80);
```

Usuńmy teraz pole `nazwisko`:

```
alter table adresy
drop column nazwisko;
```

W końcu dodajmy pole `telefon`, które ma występować bezpośrednio po polu `ulica`:

```
alter table adresy
add telefon varchar(15) after ulica;
```

5.13 Usuwanie tabel

Aby usunąć tabelę wywołujemy następującą komendę:

```
drop table nazwa_tabeli;
```

5.14 Ustawianie i usuwanie uprawnień użytkowników

W systemie MySQL może istnieć wiele kont użytkowników. Ze względów bezpieczeństwa użytkownik `root` powinien być używany wyłącznie do celów administracyjnych. Nie należy mylić użytkowników baz danych MySQL z kontami w systemie Unix, GNU/Linux, czy MS Windows.

Każde konto ma określone uprawnienia, które pozwalają użytkownikowi na wykonywanie określonych zadań w systemie MySQL. Uprawnienia te podzielone są na trzy grupy: uprawnienia użytkowników, uprawnienia administracyjne i uprawnienia specjalne. Poniżej zestawione zostały tabele tych uprawnień.

Uprawnienia użytkowników:

Uprawnienie	**Dotyczy**
`ALTER`	tabele
`DELETE`	tabele
`INDEX`	tabele
`INSERT`	tabele, kolumny
`SELECT`	tabele, kolumny
`UPDATE`	tabele, kolumny
`CREATE`	bazy danych, tabele, kolumny
`DROP`	bazy danych, tabele, kolumny

Uprawnienia administracyjne:

Uprawnienie	**Opis**
`FILE`	Pozwala na używanie składni `SELECT ... INTO OUTFILE` i `LOAD DATA INFILE`
`PROCESS`	Pozwala użyć: `SHOW FULL PROCESSLIST`
`RELOAD`	Pozwala na użycie polecenia `FLUSH`, które umożliwia ponowne wczytanie tabeli uprawnień systemu
`SHUTDOWN`	Pozwala wykonać: `mysqladmin shutdown`, tzn. zatrzymać serwer MySQL

Uprawnienia specjalne:

Uprawnienie	**Opis**
`ALL [PRIVILEGES]`	Nadaje wszystkie uprawnienia bez `WITH GRANT OPTION`
`USAGE`	Synonim dla `no privileges`

5.14.1 Przydzielanie uprawnień: polecenie GRANT

Ogólna składnia polecenia wygląda następująco:

```
grant priv_type [(column_list)] [, priv_type [(column_list)] ...]
on {tbl_name | * | *.* | db_name.*}
to user_name [identified by [password] 'password']
[with [grant option]
```

Oto kilka przykładów polecenia:

```
grant all
on *
to ja identified by "trudne_haslo"
with grant option;

grant usage
on books.*
to maria identified by 'mniam';

grant selcet,insert,delete,update
on books.*
to ktos identified by '12tu4tu34';
```

5.14.2 Usuwanie uprawnień: polecenie REVOKE

Składnia polecenia wygląda następująco:

```
revoke priv_type [(column_list)] [, priv_type [(column_list)] ...]
on {tbl_name | * | *.* | db_name.*}
from user_name [, user_name ...]
```

Na koniec kilka przykładów użycia polecenia:

```
revoke all
on *
from ja;

revoke delete
on books.*
from ktos;
```

6 Zasoby sieciowe związane z systemem MySQL

- http://www.mysql.com - oficjalna witryna MySQL, udostępnia pakiety MySQL na większość platform, kod źródłowy oraz doskonałą dokumentację.
- http://www.w3schools.com/sql/default.asp - SQL Tutorial - doskonały kurs SQL z przykładami i ćwiczeniami.
- http://sqlcourse.com - SQL Course - kolejny kurs SQL dla początkujących.
- http://searchdatabase.techtarget.com/ - portal z wieloma cennymi informacjami na temat baz danych - podręczniki, dokumentacja, FAQ, itp.
- http://www.webhelp.pl/ - Webhelp - polski portal zawierający wiele materiałów dotyczących m. in. php, mysql, html, css, flash, javascript, asp, cgi.
- http://webcity.pl/ - WebCity - strona po Polsku, zawiera kursy PHP i MySQL.
- http://4programmers.net/ - polska witryna o programowaniu.