10.1 10.1 Teoria

Step 1

Baza danych to uporządkowany zbiór rekordów lub danych przechowywanych w systemie komputerowym i zorganizowany w taki sposób, aby można było je szybko przeszukiwać, a informacje można było szybko odzyskać. `SQL` oznacza **Structured Query Language**. Ten język jest luźno oparty na języku angielskim i jest również używany w innych bazach danych, takich jak `Oracle` i `Microsoft SQL Server`. Został zaprojektowany, aby umożliwić proste żądania z bazy danych za pomocą poleceń, takich jak:

```
SELECT title FROM publications WHERE author = 'Charles Dickens';
```

Do najpopularniejszego systemu bazodanowego dla serwerów WWW należy system **MySQL** (https://www.mysql.com/ (https://www.mysql.com/)). Opracowana w połowie lat 90. XX wieku jest obecnie dojrzałą technologią, która obsługuje wiele z najczęściej odwiedzanych obecnie miejsc w Internecie. Jednym z powodów jego sukcesu (tak jak w przypadku PHP) jest fakt, że jest on darmowy. Ale jest też niezwykle potężny i wyjątkowo szybki - może działać nawet na najbardziej podstawowym sprzęcie i prawie nie wpływa na zasoby systemowe. MySQL jest również wysoce skalowalny, co oznacza, że może rosnąć wraz z witryną.

Logo MySQL

Baza danych MySQL zawiera jedną lub więcej tabel, z których każda zawiera rekordy lub wiersze. W tych wierszach znajdują się różne kolumny lub pola, które zawierają same dane. Poniżej znajduje się przykład bazy danych pięciu publikacji z wyszczególnieniem autora, tytułu, rodzaju i roku publikacji.

Author	Title	Type	Year
Mark Twain	The Adventures of Tom Sawyer	Fiction	1876
Jane Austen	Pride and Prejudice	Fiction	1811
William Shakespeare	Romeo and Juliet	Play	1594

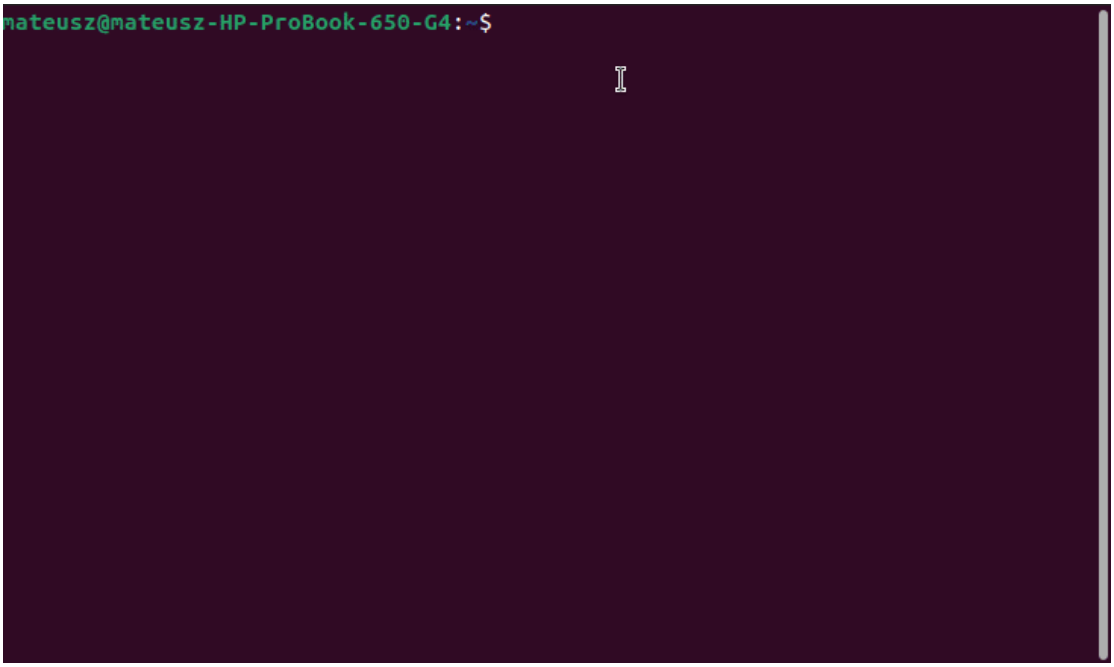
Każdy wiersz w tabeli jest taki sam jak wiersz w tabeli MySQL, kolumna w tabeli odpowiada kolumnie w MySQL, a każdy element w wierszu jest taki sam jak pole MySQL. Do głównych terminów związanych z obsługą baz danych (w kontekście MySQL) należą:

- **Baza danych** - Ogólny kontener dla kolekcji danych MySQL
- **Tabela** - Kontener podrzędny w bazie danych, który przechowuje rzeczywiste dane
- **Krotka (wiersz)** - Pojedynczy rekord w tabeli, który może zawierać kilka pól
- **Kolumna** - Nazwa pola w wierszu

Instalacja serwera MySQL w systemie Ubuntu wygląda następująco:

1. Najpierw należy zaktualizować pakiety, korzystając z polecenia `sudo apt update`;
2. Następnie dokonujemy instalacji serwera za pomocą polecenia `sudo apt install mysql-server-8.0`.

Spowoduje to zainstalowanie MySQL w wersji 8 i wyżej, ale nie wyświetli monitu o ustawienie hasła ani żadnych innych zmian w konfiguracji.



W przypadku nowych instalacji warto uruchomić skrypt bezpieczeństwa (ang. security script). Spowoduje to zmianę niektórych mniej bezpiecznych opcji domyślnych, takich jak zdalne logowanie do roota i przykładowi użytkownicy. Aby uruchomić skrypt bezpieczeństwa należy skorzystać z polecenia:

```
sudo mysql_secure_installation
```

Powyższe polecenie poprowadzi przez serię monitów, w których możemy wprowadzić pewne zmiany w opcjach bezpieczeństwa instalacji MySQL. Pierwszy monit zapyta, czy chcemy skonfigurować wtyczkę `Validate Password`, której można użyć do przetestowania siły hasła MySQL. Niezależnie od wyboru następnym pytaniem będzie ustawienie hasła dla użytkownika `root` MySQL. Następnie można nacisnąć klawisz ENTER, aby zaakceptować wartości domyślne dla wszystkich kolejnych pytań. Spowoduje to usunięcie anonimowych użytkowników i testową bazę danych, wyłączenie zdalnego logowania roota i załadowanie tych nowych reguł, aby MySQL natychmiast uwzględnił wprowadzone zmiany.

Aby zainicjować katalog danych MySQL, należy użyć `mysql_install_db` dla wersji wcześniejszych niż 5.7.6 i `mysqld --initialize` dla wersji 5.7.6 i nowszych. Jeśli jednak zainstalowany został MySQL tak jak powyżej, katalog danych został zainicjowany automatycznie; Jeśli mimo wszystko spróbujemy uruchomić polecenie, zobaczymy następujący błąd:

```
mysqld: Can't create directory '/var/lib/mysql/' (OS errno 17 - File exists)
2021-05-13T09:38:14.757885Z 0 [System] [MY-013169] [Server] /usr/sbin/mysqld (mysqld 8.0.23-
0ubuntu0.20.10.1) initializing of server in progress as process 273800
2021-05-13T09:38:14.758794Z 0 [ERROR] [MY-010187] [Server] Could not open file '/var/log/mysql/error.log'
for error logging: Permission denied
2021-05-13T09:38:14.758815Z 0 [ERROR] [MY-013236] [Server] The designated data directory /var/lib/mysql/ is
unusable. You can remove all files that the server added to it.
2021-05-13T09:38:14.758821Z 0 [ERROR] [MY-010119] [Server] Aborting
2021-05-13T09:38:14.758928Z 0 [System] [MY-010910] [Server] /usr/sbin/mysqld: Shutdown complete (mysqld
8.0.23-0ubuntu0.20.10.1)  (Ubuntu).
```

Mimo to, że zostało ustawione hasło dla użytkownika root MySQL, użytkownik ten nie jest skonfigurowany do uwierzytelniania za pomocą hasła podczas łączenia się z powłoką MySQL. W systemach Ubuntu z MySQL 5.7 (i nowszymi wersjami) `root` użytkownika MySQL jest domyślnie ustawiony na uwierzytelnianie za pomocą wtyczki `auth_socket`, a nie za pomocą hasła. Pozwala to w wielu przypadkach na większe bezpieczeństwo i użyteczność, ale może również skomplikować sytuację, gdy trzeba zezwolić zewnętrznemu programowi (np. PhpMyAdmin) na dostęp do użytkownika. Aby użyć hasła do połączenia się z MySQL jako root, musimy zmienić jego metodę uwierzytelniania z `auth_socket` na `mysql_native_password`. Aby to zrobić, należy otworzyć monit MySQL w terminalu:

```
sudo mysql
```

Następnie należy sprawdzić, która metody uwierzytelniania używa każde z kont użytkowników MySQL, używając następującego polecenia:

```
SELECT user,authentication_string,plugin,host FROM mysql.user;
```

Wynik prawdopodobnie będzie wyglądał następująco:

```
+------------------+-------------------------------------------+-----------------------+-----------+
| user             | authentication_string                     | plugin                | host      |
+------------------+-------------------------------------------+-----------------------+-----------+
| root             |                                           | auth_socket           | localhost |
| mysql.session    | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE | mysql_native_password | localhost |
| mysql.sys        | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE | mysql_native_password | localhost |
| debian-sys-maint | *CC744277A401A7D25BE1CA89AFF17BF607F876FF | mysql_native_password | localhost |
+------------------+-------------------------------------------+-----------------------+-----------+
4 rows in set (0.00 sec)
```

W tym przykładzie widać, że użytkownik root faktycznie uwierzytelnia się za pomocą wtyczki `auth_socket`. Aby skonfigurować konto `root` do uwierzytelniania za pomocą hasła, należy uruchomić następujące polecenie `ALTER USER`. Należy pamiętać że hasło powinno być silne oraz, że to polecenie zmieni hasło roota ustawione podczas użycia polecenia `mysql_secure_installation`.

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'silneHaslo123@';
```

Następnie należy uruchomić polecenie `FLUSH PRIVILEGES`, które nakazują serwerowi ponowne załadowanie tabel grantów i wprowadzenie nowych zmian w życie:

```
FLUSH PRIVILEGES;
```

Możemy sprawdzić ponownie metody uwierzytelniania stosowane przez każdego z użytkowników, aby upewnić się, że `root` nie jest już uwierzytelniany za pomocą wtyczki `auth_socket`:

```
+------------------+-------------------------------------------+-----------------------+-----------+
| user             | authentication_string                     | plugin                | host      |
+------------------+-------------------------------------------+-----------------------+-----------+
| root             | *3636DACC8616D997782ADD0839F92C1571D6D78F | mysql_native_password | localhost |
| mysql.session    | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE | mysql_native_password | localhost |
| mysql.sys        | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE | mysql_native_password | localhost |
| debian-sys-maint | *CC744277A401A7D25BE1CA89AFF17BF607F876FF | mysql_native_password | localhost |
+------------------+-------------------------------------------+-----------------------+-----------+
4 rows in set (0.00 sec)
```

Po wyjściu z konsoli (po dwukrotnym poleceniu `exit`), nie będziemy mogli już wejść do `mysql` za pomocą polecenia `sudo mysql`, ale za pomocą polecenia (dla konta `root`):

```
mysql -u root -p
```

Step 2

Niezależnie od tego, jak system MySQL został zainstalowany, powinien zacząć działać automatycznie. Aby to przetestować, możemy sprawdzić jego stan za pomocą polecenia:

```
systemctl status mysql.service
```

Wynik polecenia będzie mniej więcej wyglądał następująco:

```
 mysql.service - MySQL Community Server
     Loaded: loaded (/lib/systemd/system/mysql.service; enabled; vendor preset:>
     Active: active (running) since Thu 2021-05-13 11:28:40 CEST; 31min ago
   Main PID: 268387 (mysqld)
     Status: "Server is operational"
      Tasks: 39 (limit: 18955)
     Memory: 337.2M
     CGroup: /system.slice/mysql.service
             └─268387 /usr/sbin/mysqld

maj 13 11:28:40 mateusz-HP-ProBook-650-G4 systemd[1]: Starting MySQL Community >
maj 13 11:28:40 mateusz-HP-ProBook-650-G4 systemd[1]: Started MySQL Community S
```

Jeśli MySQL nie działa, możemy go uruchomić za pomocą polecenia:

```
sudo systemctl start mysql
```

W celu dodatkowego sprawdzenia możemy spróbować połączyć się z bazą danych za pomocą narzędzia `mysqladmin`, które jest klientem umożliwiającym uruchamianie poleceń administracyjnych. Na przykład to polecenie mówi, aby połączyć się z MySQL jako root (-u root), poprosić o hasło (-p) i zwrócić wersję:

```
sudo mysqladmin -p -u root version
```

Aby wyświetlić bazy danych w MySQL, należy z poziomu konsoli wpisać polecenie:

```
SHOW DATABASES;
```

Po świeżej instalacji wynik wygląda następująco:

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.01 sec)
```

Znak średnika jest używany przez MySQL do oddzielania lub kończenia poleceń. Jeśli zapomnimy go wprowadzić, MySQL wyświetli monit i zaczeka, aż użytkownik zakończy polecenie. Wymagany średnik został wprowadzony do składni, aby umożliwić wprowadzanie poleceń wielowierszowych, co może być wygodne, ponieważ niektóre polecenia są dość długie. Pozwala również na wydanie więcej niż jednego polecenia naraz, umieszczając średnik po każdym z nich. Interpreter pobiera je wszystkie w pakiecie po naciśnięciu klawisza Enter (lub Return) i wykonuje je po kolei. Bardzo często pojawia się w tej sytuacji znak zachęty MySQL w celu oczekiwania wpisania znaku średnika. Istnieje sześć różnych podpowiedzi, które może wyświetlić MySQL, które zobrazuje poniższa tabela:

Znak zachęty MySQL	Znaczenie
`mysql>`	Gotowy i czekający na polecenie
`->`	Oczekiwanie na następny wiersz polecenia
`'>`	Oczekiwanie na następną linię ciągu zaczynającą się od pojedynczego cudzysłowu
`">`	Oczekiwanie na następną linię ciągu zaczynającą się od podwójnego cudzysłowu

Znak zachęty MySQL	Znaczenie
`` `> ``	Oczekiwanie na następną linię łańcucha rozpoczęte od `
`/*>`	Oczekiwanie na następny wiersz komentarza zaczynający się od / *

Jeśli jesteśmy w trakcie wprowadzania polecenia i zdecydujemy się na nie wykonywanie go, nie należy naciskać kombinacji `Ctrl-C`! To zamknie program. Zamiast tego możemy wpisać `\c` i nacisnąć klawisz ENTER. Kiedy wpiszemy znak `\c`, to MySQL zignoruje wszystko, co zostało wpisane i wyświetli nowy znak zachęty. Bez `\c` wyświetliłby się komunikat o błędzie. Jeśli jednak został otwarty napis lub komentarz, należy zamknąć go przed użyciem `\c`; inaczej MySQL pomyśli, że `\c` jest tylko częścią ciągu.

Step 3

Do najpopularniejszych poleceń używanych w konsoli `mysql` należą:

- `ALTER` - Dokonuje zmian w tabeli lub bazie danych
- `BACKUP` - Tworzy zapasową kopie bazy danych
- `\c` - Anuluj wprowadzenie polecenia
- `CREATE` - Tworzy bazę danych
- `DELETE` - Usuwa wiersz z tabeli
- `DESCRIBE` - Opisuje kolumnę tabeli
- `DROP` - Usuń bazę danych lub tabelę
- `EXIT` (CTRL-C) - Wyjście
- `GRANT` - Zmienia uprawnienia użytkownika
- `HELP` (`\h`, `\?`) - Wyświetla pomoc
- `INSERT` - Wstawia dane do tabeli
- `LOCK` - Blokada tabeli
- `QUIT` (`\q`) - Ma takie samo działanie jak `EXIT`
- `RENAME` - Zmień nazwę tabeli
- `SHOW` - Wyświetla szczegóły dotyczące obiektu
- `SOURCE` - Wykonuje plik z poleceniami
- `STATUS` (`\s`) - Wyświetla aktualny stan
- `TRUNCATE` - Opróżnij tabelę
- `UNLOCK` - Odblokuj tabelę
- `UPDATE` - Zaktualizuj istniejące rekord
- `USE` - Użyj bazy danych

W poleceniach i słowach kluczowych SQL nie jest rozróżniana wielkość liter. `CREATE`, `create`, oraz `CrEaTe` oznacza to samo. Jednak ze względu na przejrzystość możemy preferować używanie wielkich liter. W nazwach tabel jest rozróżniana wielkość liter w systemie Linux i macOS, ale w systemie Windows nie jest rozróżniana wielkość liter. Dlatego ze względu na przenośność należy zawsze wybierać etui i się go trzymać. Zalecanym stylem jest używanie małych liter w nazwach tabel.

Aby utworzyć nową bazę danych o nazwie `publications` należy w konsoli `mysql` skorzystać z następującego polecenia:

```
CREATE DATABASE publications;
```

Powodzenie powyższego polecenie zwróci komunikat, który jeszcze nie znaczy wiele - `Query OK, 1 row affected (0.02 sec)` .Po utworzeniu bazy danych jeżeli chcemy z nią pracować, należy wydać następujące polecenie:

```
USE publications;
```

Teraz należy przyjrzeć się, jak tworzyć użytkowników, ponieważ prawdopodobnie nie będziemy chcieli przyznawać skryptom PHP dostępu `root` do MySQL. Aby utworzyć użytkownika, należy skorzystać z polecenia `CREATE USER`, która przyjmuje następującą postać:

```
CREATE USER 'username'@'hostname' IDENTIFIED BY 'password';
GRANT PRIVILEGES ON database.object TO 'username'@'hostname'
```

Wszystko to powinno wyglądać całkiem prosto, z możliwym wyjątkiem części `database.object`, która odnosi się do samej bazy danych i zawartych w niej obiektów. W przypadku użycia polecenia GRANT można skorzystać z następujących parametrów:

Argument	Znaczenie
`*.*`	Wszystkie bazy danych oraz wszystkie obiektu
`database.*`	Tylko baza danych o nazwie `database` i wszystkie zawarte w niej obiekty
`database.object`	Tylko baza danych o nazwie `database` i obiekt o nazwie `object`

Stwórzmy więc użytkownika, który będzie miał dostęp tylko do nowej bazy danych publikacji i wszystkich jej obiektów, wprowadzając następujące polecenia:

```
CREATE USER 'jim'@'localhost' IDENTIFIED BY 'password'; GRANT ALL ON publications.* TO 'jim'@'localhost';
```

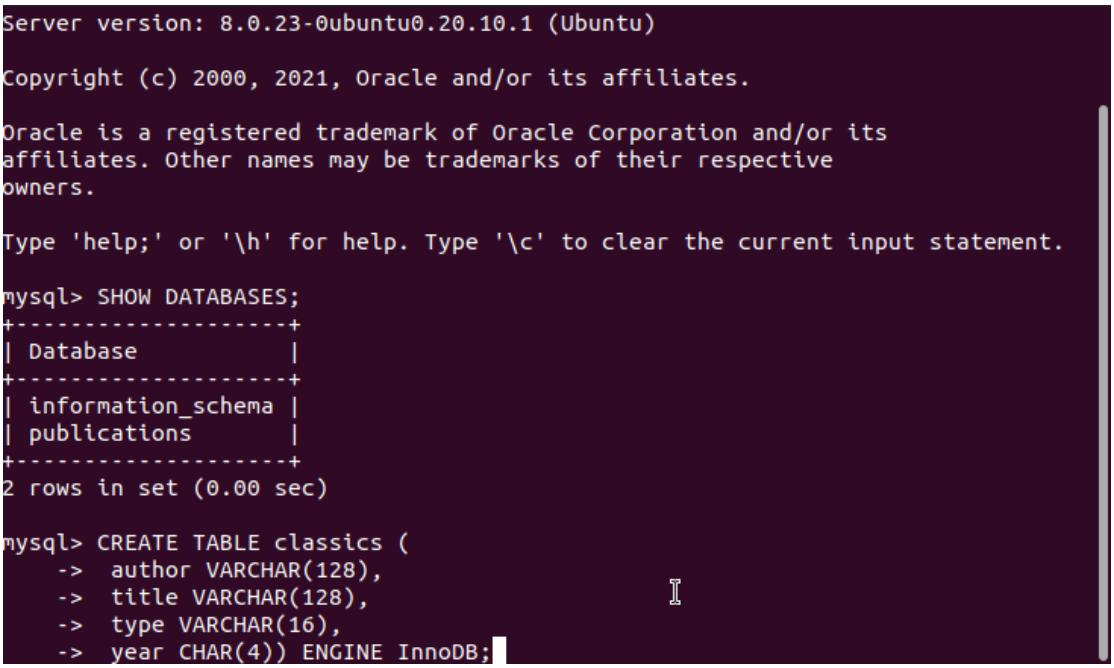
Umożliwia to użytkownikowi `jim@localhost` pełny dostęp do bazy danych publikacji przy użyciu hasła `silneHaslo123@`. Możemy teraz zalogować się na użytkownik jim korzystając z polecenia:

```
mysql -u jim -p
```

Aby utworzyć tabelę w bazie publications skorzystamy z prostego zapytania SQL:

```
CREATE TABLE classics (
 author VARCHAR(128),
 title VARCHAR(128),
 type VARCHAR(16),
 year CHAR(4)) ENGINE InnoDB;
```

MySQL powinien następnie wydać odpowiedź `Query OK,` wraz z informacją o tym, ile czasu zajęło wykonanie polecenia. Jeśli zamiast tego pojawi się komunikat o błędzie, należy dokładnie sprawdzić składnię. Liczy się każdy nawias i przecinek, a błędy wpisywania są łatwe do popełnienia.



Ostatnie dwa słowa powyższego polecenia wymagają małego wyjaśnienia. MySQL może wewnętrznie przetwarzać zapytania na wiele różnych sposobów, a te różne sposoby są obsługiwane przez różne silniki. Począwszy od wersji 5.6 `InnoDB` jest domyślnym mechanizmem przechowywania danych dla MySQL i jest on tutaj używany, ponieważ obsługuje wyszukiwania `FULLTEXT`. Dopóki mamy stosunkowo aktualną wersję MySQL, możemy pominąć sekcję ENGINE InnoDB polecenia podczas tworzenia tabeli. Jeśli używamy wersji MySQL wcześniejszej niż 5.6, silnik InnoDB nie będzie obsługiwał indeksów `FULLTEXT`, więc będzie trzeba zamienić InnoDB w poleceniu na `MyISAM`, aby wskazać, że chcemy używać tego silnika. InnoDB jest ogólnie bardziej wydajnym rozwiązaniem i jest zalecaną opcją.

Aby sprawdzić, czy nowa tabela została utworzona, należy wpisać:

```
DESCRIBE classics;
```

Powinna pojawić się następująca odpowiedź:

```
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| author | varchar(128) | YES  |     | NULL    |       |
| title  | varchar(128) | YES  |     | NULL    |       |
| type   | varchar(16)  | YES  |     | NULL    |       |
| year   | char(4)      | YES  |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Polecenie `DESCRIBE` jest nieocenioną pomocą przy debugowaniu, gdy musimy upewnić się, że poprawnie zostały utworzone tabelę MySQL. Możemy go również użyć do przypomnienia sobie o nazwach pól lub kolumn tabeli i typach danych w każdej z nich. Polecenie to wyświetla tabelę, korzystając z następujących nagłówków:

- `Field` - Nazwa każdego pola lub kolumny w tabeli;
- `Type` - Typ danych przechowywanych w polu;
- `Null` - Czy pole może zawierać wartość `NULL`;
- `Key` - Typ klucza, jeśli został zastosowany;
- `Default` - Wartość domyślna, która zostanie przypisana do pola, jeśli żadna wartość nie zostanie określona podczas tworzenia;
- `Extra` - Dodatkowe informacje, na przykład czy pole jest ustawione na automatyczne zwiększanie wartości.

Step 4

W poprzednim przykładzie można było zauważyć, że trzem polom tabeli nadano typ danych `VARCHAR`, a jednemu nadano typ `CHAR`. Termin `VARCHAR` oznacza ciąg znaków o zmiennej długości, a polecenie przyjmuje wartość liczbową, która mówi MySQL o maksymalnej dozwolonej długości ciągu przechowywanego w tym polu. Zarówno `CHAR`, jak i `VARCHAR` akceptują ciągi tekstowe i nakładają ograniczenie na rozmiar pola. Różnica polega na tym, że każdy łańcuch w polu `CHAR` ma określony rozmiar. Jeśli umieścisz

mniejszy tekst, zostanie on wypełniony spacjami. Pole `VARCHAR` nie wypełnia tekstu; pozwala zmieniać rozmiar pola, aby dopasować go do wstawianego tekstu. Ale `VARCHAR` wymaga niewielkiej ilości narzutów, aby śledzić rozmiar każdej wartości. Tak więc `CHAR` jest nieco bardziej wydajny, jeśli rozmiary są podobne we wszystkich rekordach, podczas gdy `VARCHAR` jest bardziej wydajny, jeśli rozmiary mogą się znacznie różnić i stać się duże. Ponadto narzut powoduje, że dostęp do danych `VARCHAR` jest nieco wolniejszy niż do danych `CHAR`. Kolejną cechą kolumn znakowych i tekstowych, ważną dla dzisiejszego globalnego zasięgu w sieci, są zestawy znaków. Przypisują one określone wartości binarne do określonych znaków. Zestaw znaków, którego używasz w języku angielskim, jest oczywiście inny niż w przypadku języka polskiego. Podczas tworzenia zestawu znaków można przypisać znak lub kolumnę tekstową. `VARCHAR` jest przydatny w naszym przykładzie, ponieważ może pomieścić nazwiska autorów i tytuły o różnej długości, jednocześnie pomagając MySQL w planowaniu rozmiaru bazy danych i łatwiejszym wykonywaniu wyszukiwań i wyszukiwań. Pamiętaj tylko, że jeśli kiedykolwiek spróbujesz przypisać wartość ciągu dłuższą niż dozwolona długość, zostanie ona obcięta do maksymalnej długości zadeklarowanej w definicji tabeli. Pole roku ma jednak przewidywalne wartości, dlatego zamiast `VARCHAR` używamy bardziej wydajnego typu danych `CHAR(4)`. Parametr 4 pozwala na 4 bajty danych, obsługując wszystkie lata od –999 do 9999; bajt składa się z 8 bitów i może mieć wartości od 00000000 do 11111111, które są dziesiętne od 0 do 255. Możemy oczywiście po prostu zapisać dwucyfrowe wartości dla roku, ale jeśli dane będą nadal potrzebne w następnym stuleciu lub w inny sposób mogą się zawijać, najpierw będą musiały zostać oczyszczone - pomyśl o „Milenijnym błędzie”, który spowodował, że daty rozpoczynające się 1 stycznia 2000 r. Były traktowane jako 1900 w wielu największych instalacjach komputerowych na świecie. Również nie został użyty typ `YEAR`, ponieważ obsługuje tylko lata od 0000 i 1901 do 2155. Dzieje się tak, ponieważ MySQL przechowuje rok w jednym bajcie ze względu na wydajność, ale oznacza to, że dostępnych jest tylko 256 lat, a lata publikacji tytułów w tabeli klasyków są na długo przed 1901 rokiem.

Poniżej znajduje się tabela opisująca listę typów danych `CHAR`. Oba typy oferują parametr, który ustawia maksymalną (lub dokładną) długość ciągu dozwoloną w polu. Jak pokazuje tabela, każdy typ ma wbudowaną maksymalną liczbę bajtów, które może zajmować.

Typ	Ilość bajtów	Przykład
`CHAR(n)`	Dokładnie n (<= 255)	`CHAR(5) "Hello"` używa 5 bajtów; `CHAR(57) "Goodbye"` używa 57 bajtów
`VARCHAR(n)`	Co najwyżej n (<= 65535)	`VARCHAR(7) "Hello"` używa 5 bajtów; `VARCHAR(100) "Goodbye"` używa 7 bajtów

Typ `BINARY` przechowuje ciągi bajtów, które nie mają skojarzonego zestawu znaków. Na przykład możemy użyć typu danych `BINARY` do przechowywania obrazu GIF.

Typ	Ilość bajtów	Przykład
`BINARY(n)`	Dokładnie n (<=255)	Jako `CHAR` ale w postaci binarnej
`VARBINARY(n)`	Co najwyżej n (<=65535)	Jako `VARCHAR` ale w postaci binarnej

Dane tekstowe mogą być również przechowywane w jednym z zestawów pól `TEXT`. Różnice między tymi polami a polami `VARCHAR` są niewielkie: Przed wersją 5.0.3 MySQL usuwał początkowe i końcowe spacje z pól `VARCHAR`. Pola `TEXT` nie mogą mieć wartości domyślnych. MySQL indeksuje tylko pierwsze `n` znaków kolumny `TEXT` (`n` jest określane podczas tworzenia indeksu). Oznacza to, że `VARCHAR` jest lepszym i szybszym typem danych do wykorzystania, jeśli chcesz przeszukać całą zawartość pola. Jeśli nigdy nie będziemy przeszukiwać więcej niż określoną liczbę początkowych znaków w polu, prawdopodobnie powinniśmy użyć typu danych `TEXT`.

Typ	Ilość bajtów	Przykład
`TINYTEXT(n)`	Co najwyżej n (<=255)	Traktowane jako ciąg z zestawem znaków
`TEXT(n)`	Co najwyżej n (<=65535)	Traktowane jako ciąg z zestawem znaków
`MEDIUMTEXT(n)`	Co najwyżej n (<=1.67e+7)	Traktowane jako ciąg z zestawem znaków
`LONGTEXT(n)`	Co najwyżej n (<=4.29e+9)	Traktowane jako ciąg z zestawem znaków

Termin `BLOB` oznacza **Binar Large OBject** i dlatego, jak można by pomyśleć, typ danych `BLOB` jest najbardziej przydatny w przypadku danych binarnych o rozmiarze przekraczającym 65 536 bajtów. Główną różnicą między typami danych `BLOB` i `BINARY` jest to, że obiekty `BLOB` nie mogą mieć wartości domyślnych.

Typ	Ilość bajtów	Przykład
`TINYBLOB(n)`	Co najwyżej n (<=255)	Traktowane jako dane binarne - bez zestawu znaków
`BLOB(n)`	Co najwyżej n (<=65535)	Traktowane jako dane binarne - bez zestawu znaków
`MEDIUMBLOB(n)`	Co najwyżej n (<=1.67e+7)	Traktowane jako dane binarne - bez zestawu znaków
`LONGBLOB(n)`	Co najwyżej n (<=4.29e+9)	Traktowane jako dane binarne - bez zestawu znaków

MySQL obsługuje różne numeryczne typy danych, od pojedynczych bajtów po liczby zmiennoprzecinkowe o podwójnej precyzji. Chociaż najwięcej pamięci, jaką może wykorzystać pole numeryczne, wynosi 8 bajtów, zaleca się wybranie najmniejszego typu danych, który będzie odpowiednio obsługiwał największą oczekiwaną wartość. Poniżej znajduje się lista numerycznych typów danych obsługiwanych przez MySQL oraz zakresy wartości, które mogą zawierać. Jeśli nie jesteś zaznajomiony z terminami, liczba ze znakiem to liczba z możliwym zakresem od wartości ujemnej, przez 0, do dodatniej; a liczba bez znaku ma wartość od 0 do dodatniej. Oba mogą zawierać tę samą liczbę wartości.

Typ	Ilość bajtów	Min wartość signed	Min wartość unsigned	Max wartość signed	Max wartość unsigned
`TINYINT`	1	-128	0	127	255
`SMALLINT`	2	-32768	0	32767	65535
`MEDIUMINT`	3	−8.38e+6	0	8.38e+6	1.67e+7
`INT/INTEGER`	4	−2.15e+9	0	2.15e+9	4.29e+9
`BIGINT`	8	−9.22e+18	0	9.22e+18	1.84e+19
`FLOAT`	4	−3.40e+38	n/a	3.40e+38	n/a
`DOUBLE/REAL`	8	−1.80e+308	n/a	1.80e+308	n/a

Aby określić, czy typ danych jest bez znaku, należy użyć kwalifikatora `UNSIGNED`.

```
CREATE TABLE tablename (fieldname INT UNSIGNED);
```

Tworząc pole liczbowe, możemy również przekazać opcjonalną liczbę jako parametr, na przykład:

```
CREATE TABLE tablename (fieldname INT(4));
```

Należy jednak pamiętać, że w przeciwieństwie do typów danych `BINARY` i `CHAR`, parametr ten nie wskazuje liczby bajtów pamięci do wykorzystania. Może się to wydawać sprzeczne z intuicją, ale to, co faktycznie reprezentuje liczba, to szerokość wyświetlania danych w polu podczas ich pobierania. Jest powszechnie używany z kwalifikatorem `ZEROFILL`, na przykład:

```
CREATE TABLE tablename (fieldname INT(4) ZEROFILL);
```

Powoduje to, że wszystkie liczby o szerokości mniejszej niż cztery znaki są uzupełniane jednym lub większą liczbą zer, co wystarcza, aby szerokość wyświetlania pola była długa na cztery znaki. Gdy pole ma już określoną szerokość lub większą, dopełnienie nie jest stosowane.

Główne pozostałe typy danych obsługiwane przez MySQL odnoszą się do daty i czasu i można je zobaczyć w poniższej tabeli:

Typ danych	Format
`DATETIME`	`0000-00-00 00:00:00`
`DATE`	`0000-00-00`
`TIMESTAMP`	`0000-00-00 00:00:00`

Typ danych	Format
`TIME`	`00:00:00`
`YEAR`	`0000` (Tylko 0000 oraz 1901-2155)

Typy danych `DATETIME` i `TIMESTAMP` są wyświetlane w ten sam sposób. Główna różnica polega na tym, że `TIMESTAMP` ma bardzo wąski zakres (od lat 1970 do 2037), podczas gdy `DATETIME` będzie zawierać prawie każdą datę, którą prawdopodobnie określimy, chyba że interesujemy się historią starożytną lub science fiction. `TIMESTAMP` jest jednak przydatny, ponieważ możesz pozwolić MySQL ustawić wartość za nas. Jeśli nie określimy wartości podczas dodawania wiersza, bieżący czas zostanie wstawiony automatycznie.

Step 5

Czasami trzeba się upewnić, że każdy wiersz w bazie danych jest niepowtarzalny. Możemy to zrobić w swoim programie, dokładnie sprawdzając wprowadzone dane i upewniając się, że istnieje co najmniej jedna wartość, która różni się w dowolnych dwóch wierszach, ale takie podejście jest podatne na błędy i działa tylko w określonych okolicznościach. Na przykład w tabeli `classics` autor może pojawiać się wielokrotnie. Podobnie rok publikacji będzie często powielany i tak dalej. Trudno byłoby zagwarantować, że nie będzie zduplikowanych wierszy. Ogólnym rozwiązaniem jest użycie dodatkowej kolumny tylko do tego celu z ustawionym atrybutem `AUTO_INCREMENT`. Jak sama nazwa wskazuje, kolumna o danym typie danych ustawi wartość swojej zawartości na wartość wpisu kolumny we wcześniej wstawionym wierszu, plus 1.

```
ALTER TABLE classics ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT KEY;
```

Polecenie `ALTER` działa na istniejącej tabeli i może dodawać, zmieniać lub usuwać kolumny. Powyższy przykład dodaje kolumnę o nazwie `id` z następującymi cechami:

- `INT UNSIGNED` - Sprawia, że kolumna przyjmuje liczbę całkowitą wystarczająco dużą, abyśmy mogli przechowywać w tabeli ponad 4 miliardy rekordów.
- `NOT NULL` - Zapewnia, że kolumna ma wartość. Wielu programistów używa wartości NULL w polu, aby wskazać, że nie ma ono żadnej wartości. Dopuściłoby to jednak duplikaty, co naruszyłoby cały powód istnienia tej kolumny, dlatego nie zezwalamy na wartości NULL.
- `AUTO_INCREMENT` - Powoduje, że MySQL ustawia unikalną wartość dla tej kolumny w każdym wierszu, jak opisano wcześniej. Tak naprawdę nie mamy kontroli nad wartością, jaką ta kolumna przyjmie w każdym wierszu, ale nie obchodzi nas to: zależy nam tylko na tym, aby zagwarantować nam niepowtarzalną wartość.
- `KEY` - Kolumna z automatycznym zwiększaniem jest przydatna jako klucz, ponieważ będziemy wyszukiwać wiersze na podstawie tej kolumny.

Każdy wpis w identyfikatorze kolumny będzie miał teraz unikalny numer, przy czym pierwszy zaczyna się od 1, a pozostałe liczą się w górę. Za każdym razem, gdy zostanie wstawiony nowy wiersz, kolumna `id` automatycznie otrzyma kolejną liczbę w sekwencji. Zamiast stosować kolumnę z mocą wsteczną, można było ją uwzględnić, wydając polecenie `CREATE` w nieco innym formacie.

```
CREATE TABLE classics (
 author VARCHAR(128),
 title VARCHAR(128),
 type VARCHAR(16),
 year CHAR(4),
 id INT UNSIGNED NOT NULL AUTO_INCREMENT KEY) ENGINE InnoDB;
```

Po wywołaniu `DESCRIBE classics` powinniśmy otrzymać następujący rezultat:

```
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| author | varchar(128) | YES  |     | NULL    |                |
| title  | varchar(128) | YES  |     | NULL    |                |
| type   | varchar(16)  | YES  |     | NULL    |                |
| year   | char(4)      | YES  |     | NULL    |                |
| id     | int unsigned | NO   | PRI | NULL    | auto_increment |
+--------+--------------+------+-----+---------+----------------+
```

Gdyby się okazało, że kolumna `id` nie jest już potrzebna, to możemy ją usunąć kolumnę za pomocą polecenia:

```
ALTER TABLE classics DROP id;
```

Aby dodać dane do tabeli, należy użyć polecenia `INSERT INTO`. Na przykład dla tabeli `classics` wstawianie danych może wyglądać następująco:

```
INSERT INTO classics(author, title, type, year)
 VALUES('Mark Twain','The Adventures of Tom Sawyer','Fiction','1876');
INSERT INTO classics(author, title, type, year)
 VALUES('Jane Austen','Pride and Prejudice','Fiction','1811');
INSERT INTO classics(author, title, type, year)
 VALUES('Charles Darwin','The Origin of Species','Non-Fiction','1856');
INSERT INTO classics(author, title, type, year)
 VALUES('Charles Dickens','The Old Curiosity Shop','Fiction','1841');
INSERT INTO classics(author, title, type, year)
 VALUES('William Shakespeare','Romeo and Juliet','Play','1594');
```

Po każdej drugiej linii powinien pojawić się komunikat `Query OK`. Po wprowadzeniu wszystkich wierszy wyświetlenie ich można wywołać poleceniem:

```
SELECT * FROM classics;
```

Rezultat powinien wyglądać następująco:

```
+--------------------+-----------------------------+-------------+------+
| author             | title                       | type        | year |
+--------------------+-----------------------------+-------------+------+
| Mark Twain         | The Adventures of Tom Sawyer | Fiction     | 1876 |
| Jane Austen        | Pride and Prejudice         | Fiction     | 1811 |
| Charles Darwin     | The Origin of Species       | Non-Fiction | 1856 |
| Charles Dickens    | The Old Curiosity Shop      | Fiction     | 1841 |
| William Shakespeare | Romeo and Juliet           | Play        | 1594 |
+--------------------+-----------------------------+-------------+------+
5 rows in set (0.00 sec)
```

Może się zdarzyć, że zwrócone wyniki w poleceniu `SELECT` są w innej kolejności, ponieważ kolejność nie jest w tym momencie określona. Pierwsza część, `INSERT INTO classics`, mówi MySQL, gdzie wstawić następujące dane. Następnie w nawiasach wymienione są cztery nazwy kolumn - `author`, `title`, `type` i `year` - wszystkie oddzielone przecinkami. To mówi MySQL, że są to pola, do których mają zostać wstawione dane. Drugi wiersz każdego polecenia `INSERT` zawiera słowo kluczowe `VALUES`, po którym następują cztery ciągi w nawiasach, oddzielone przecinkami. Zapewnia to MySQL cztery wartości, które mają zostać wstawione do czterech wcześniej określonych kolumn. Każda pozycja danych zostanie wstawiona do odpowiedniej kolumny w korespondencji jeden do jednego. Jeśli przypadkowo umieścimy kolumny w innej kolejności niż dane, dane zostaną umieszczone w niewłaściwych kolumnach. Ponadto liczba kolumn musi odpowiadać liczbie elementów danych.

Zmiana nazwy tabeli, podobnie jak każda inna zmiana struktury lub metainformacji tabeli, odbywa się za pomocą polecenia `ALTER`. Na przykład, aby zmienić nazwę `classics` na pre1900, należy użyć następującego polecenia:

```
ALTER TABLE classics RENAME pre1900;
```

Zmiana typu danych w kolumnie również wykorzystuje polecenie `ALTER`, tym razem w połączeniu ze słowem kluczowym `MODIFY`. Aby zmienić typ danych roku kolumny z `CHAR(4)` na `SMALLINT` (który wymaga tylko 2 bajtów pamięci, a więc oszczędza miejsce na dysku), należy użyć polecenia:

```
ALTER TABLE classics MODIFY year SMALLINT;
```

Gdy to zrobisz, jeśli konwersja typu danych ma sens do MySQL, automatycznie zmieni dane, zachowując znaczenie. W takim przypadku zmieni każdy ciąg na porównywalną liczbę całkowitą, o ile ciąg jest rozpoznawalny jako odnoszący się do liczby całkowitej. Załóżmy, że utworzyliśmy tabelę i zapełniliśmy ją dużą ilością danych, tylko po to, aby odkryć, że potrzebujemy dodatkowej kolumny. Aby dodać nową kolumnę, określającą ilość stron należy użyć polecenia:

```
ALTER TABLE classics ADD pages SMALLINT UNSIGNED;
```

Spowoduje to dodanie nowej kolumny ze stronami z nazwami korzystającymi z typu danych `UNSIGNED` `SMALLINT`, wystarczającym do przechowywania wartości do 65 535. Wywołując polecenie `DESCRIBE classics` otrzymamy następujący rezultat:

```
+--------+------------------+------+-----+---------+-------+
| Field  | Type             | Null | Key | Default | Extra |
+--------+------------------+------+-----+---------+-------+
| author | varchar(128)     | YES  |     | NULL    |       |
| title  | varchar(128)     | YES  |     | NULL    |       |
| type   | varchar(16)      | YES  |     | NULL    |       |
| year   | char(4)          | YES  |     | NULL    |       |
| pages  | smallint unsigned | YES  |     | NULL    |       |
+--------+------------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Patrząc ponownie powyższą tabelę, łatwo stwierdzić, że kolumna o nazwie `type` jest myląca, ponieważ jest to nazwa używana przez MySQL do identyfikowania typów danych. Zmienimy jego nazwę na `category`:

```
ALTER TABLE classics CHANGE type category VARCHAR(16);
```

```
+----------+------------------+------+-----+---------+-------+
| Field    | Type             | Null | Key | Default | Extra |
+----------+------------------+------+-----+---------+-------+
| author   | varchar(128)     | YES  |     | NULL    |       |
| title    | varchar(128)     | YES  |     | NULL    |       |
| category | varchar(16)      | YES  |     | NULL    |       |
| year     | char(4)          | YES  |     | NULL    |       |
| pages    | smallint unsigned | YES  |     | NULL    |       |
+----------+------------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Zwróć uwagę na dodanie `VARCHAR(16)` na końcu tego polecenia. Dzieje się tak, ponieważ słowo kluczowe `CHANGE` wymaga określenia typu danych, nawet jeśli nie zamierzamy go zmieniać, a `VARCHAR(16)` był typem danych określonym podczas początkowego tworzenia tej kolumny jako typ. Właściwie po zastanowieniu możesz zdecydować, że strony z kolumnami z liczbą stron nie są w rzeczywistości przydatne w tej konkretnej bazie danych, więc oto jak usunąć tę kolumnę za pomocą słowa kluczowego `DROP`:

```
ALTER TABLE classics DROP pages;
```

```
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| author   | varchar(128) | YES  |     | NULL    |       |
| title    | varchar(128) | YES  |     | NULL    |       |
| category | varchar(16)  | YES  |     | NULL    |       |
| year     | char(4)      | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Należy pamiętać, że polecenie `DROP` jest nieodwracalne. Powinno zawsze używać się go ostrożnie, ponieważ możemy nieumyślnie usunąć całe tabele (a nawet bazy danych)!

Usunięcie tabeli jest rzeczywiście bardzo łatwe. Dla przykładu utwórzmy bardzo prostą tabelę o nazwie trash:

```
CREATE TABLE trash(id INT);
```

Wynik polecenia `SHOW TABLES` wygląda następująco:

```
+------------------------+
| Tables_in_publications |
+------------------------+
| classics               |
| trash                  |
+------------------------+
2 rows in set (0.01 sec)
```

Aby usunąć tabelę, korzystamy tak samo jak w przypadku kolumny z polecenia DROP:

```
DROP TABLE trash;
```

```
+------------------------+
| Tables_in_publications |
+------------------------+
| classics               |
+------------------------+
1 row in set (0.01 sec)
```

Step 6

Do tej pory stworzyliśmy tabelę `classics`. Wszystkie publikacje w tabeli można przeszukiwać, ale nie ma jednego unikalnego klucza dla każdej publikacji, aby umożliwić natychmiastowy dostęp do wiersza. Znaczenie posiadania klucza z unikalną wartością dla każdego wiersza pojawi się, gdy zaczniemy łączyć dane z różnych tabel. Atrybut `AUTO_INCREMENT` przedstawiał ideę klucza podstawowego podczas tworzenia identyfikatora kolumny z automatycznym zwiększaniem wartości, który mógłby zostać użyty jako klucz podstawowy dla tej tabeli. Wykorzystamy to dodając do każdej publikacji unikalny numer ISBN. Numery ISBN mają 13 znaków, więc zacznijmy od dodania odpowiedniej kolumny do naszej tabeli:

```
ALTER TABLE classics ADD isbn CHAR(13) PRIMARY KEY;
```

Powyższe polecenie zakończy się niepowodzeniem. Otrzymamy błąd `Duplicate entry '' for key 'classics.PRIMARY'`. Przyczyną jest to, że tabela jest już wypełniona niektórymi danymi i to polecenie próbuje dodać kolumnę o wartości NULL do każdego wiersza, co jest niedozwolone , ponieważ wszystkie wartości muszą być unikalne w każdej kolumnie mającej indeks klucza podstawowego. Jeśli jednak w tabeli nie ma żadnych danych, to polecenie działałoby dobrze. W tej obecnej sytuacji musimy stworzyć nową kolumnę bez indeksu, wypełnić ją danymi, a następnie dodać indeks klucza głównego. Na szczęście każdy rok jest unikalny w bieżącym zestawie danych, więc możemy użyć kolumny roku do zidentyfikowania każdego wiersza do aktualizacji.

```
ALTER TABLE classics ADD isbn CHAR(13);
UPDATE classics SET isbn='9781598184891' WHERE year='1876';
UPDATE classics SET isbn='9780582506206' WHERE year='1811';
UPDATE classics SET isbn='9780517123201' WHERE year='1856';
UPDATE classics SET isbn='9780099533474' WHERE year='1841';
UPDATE classics SET isbn='9780192814968' WHERE year='1594';
ALTER TABLE classics ADD PRIMARY KEY(isbn);
DESCRIBE classics;
```

```
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| author   | varchar(128) | YES  |     | NULL    |       |
| title    | varchar(128) | YES  |     | NULL    |       |
| category | varchar(16)  | YES  |     | NULL    |       |
| year     | char(4)      | YES  |     | NULL    |       |
| isbn     | char(13)     | NO   | PRI | NULL    |       |
+----------+--------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Aby utworzyć klucz podstawowy podczas tworzenia klasycznych tabel, można użyć następującego polecenia:

```
CREATE TABLE classics (
 author VARCHAR(128),
 title VARCHAR(128),
 category VARCHAR(16),
 year SMALLINT,
 isbn CHAR(13),
 INDEX(author(20)),
 INDEX(title(20)),
 INDEX(category(4)),
 INDEX(year),
 PRIMARY KEY (isbn)) ENGINE InnoDB;
```