Język SQL pozwala tworzyć, modyfikować i usuwać bazy danych oraz znajdujące się w nich obiekty, takie jak tabele, widoki, indeksy, procedury czy funkcje. Niestety, o ile składnia instrukcji SELECT, INSERT, DELETE oraz UPDATE jest w dużym stopniu ustandaryzowana, to każdy serwer bazodanowy zawiera specyficzne funkcjonalności, często wynikające ze sposobu jego działania. Instrukcje CREATE, ALTER i DROP wykorzystują unikalne cechy danego serwera i w konsekwencji ich składnia jest bardzo różna. W tym odcinku utworzysz wcześniej zaprojektowane tabele i poznasz składnie instrukcji CREATE TABLE, ALTER TABLE i DROP TABLE serwera MySQL. Dowiesz się również, do czego służą w bazach danych indeksy i jak je tworzyć.

Bazy danych

W MySQL-u przed rozpoczęciem tworzenia tabel należy utworzyć i skonfigurować samą bazę danych. Służy do tego instrukcja CREATE DATABASE. W najprostszej formie instrukcja ta wygląda następująco: CREATE DATABASE nazwa bazy.

Obiekty dowolnego typu, w tym bazy danych, możemy tworzyć instrukcją CREATE. Po czasowniku CREATE należy podać typ tworzonego obiektu i jego nazwę.

Aby utworzyć bazę danych firma, należy załogować się jako root i wykonać polecenie (listing 10.1).

Listing 10.1. Tworzymy bazę danych i łączymy się z nią

```
CREATE DATABASE firma;

Query OK, 1 row affected (0.08 sec)


USE firma;

Database changed
```

Tabele

W MySQL-u odpowiednikiem schematu jest baza danych (z reguły w ramach jednej bazy można tworzyć wiele różnych schematów, nazywanych czasem przestrzeniami nazw), tak więc po utworzeniu bazy wystarczy się z nią połączyć i jeżeli mamy nadane odpowiednie uprawnienia (ta kwestia została opisana w drugim odcinku kursu), możemy tworzyć tabele.

Tworzenie tabel

Do zdefiniowania nowej tabeli używamy instrukcji CREATE TABLE. Obowiązkowymi parametrami instrukcji są: nazwa tworzonej tabeli, nazwy wchodzących w skład tabeli kolumn oraz typ danych przechowywanych w poszczególnych tabelach. MySQL nakłada pewne ograniczenia na każdy z tych parametrów. Nazwa tabeli musi być zgodna z regułami nazewnictwa. W przypadku serwera MySQL nazwy tabel i kolumn:

1. Mogą zawierać litery, cyfry i znaki specjalne.
2. Mogą zawierać litery dowolnej wielkości, przy czym rozróżnianie dużych i małych liter zależy od konfiguracji systemu operacyjnego i samej bazy danych.
3. Muszą być unikalne — niedopuszczalne jest istnienie w bazie kilku tabel lub w pojedynczej tabeli kilku kolumn o tej samej nazwie.
4. Nie powinny być terminem zastrzeżonym dla języka, np. nazwą instrukcji, funkcji lub klauzuli — w takim przypadku posługując się nazwą, trzeba będzie umieszczać ją w apostrofach.
5. Nie powinny kończyć się znakiem spacji.
6. Nie mogą zawierać znaków ukośnika /, lewego ukośnika i kropki..

Typy kolumn

Dla każdej kolumny należy zdefiniować określony typ danych. Określony typ danych powinien w maksymalnym stopniu odpowiadać rodzajowi danych przechowywanych w konkretnej tabeli. Marnotrawstwem byłoby na przykład rezerwowanie 255 bajtów dla pola, które wykorzystuje tylko 2 bajty. Analogicznie rezerwowanie 5 bajtów dla numeru telefonu, który może mieć 10 cyfr, to zdecydowanie za mało. Na szczęście relacyjne bazy danych dostarczają bardzo bogatego zestawu typów danych (tabele 10.1 – 10.3).

Dobierając właściwy typ kolumny do przechowywanych w niej danych, poprawimy wydajność zapytań, zmniejszymy wielkość tabeli oraz w pewnym stopniu zabezpieczymy się przed zapisywaniem w kolumnie błędnych danych.

Tabela 10.1. Numeryczne typy danych serwera MySQL

Typ

Rozmiar w bajtach

Wartość minimalna

Wartość maksymalna

TINYINT (ze znakiem)

1

–128

127

TINYINT (bez znaku)

1

0

255

SMALLINT (ze znakiem)

2

–32 768

32 767

SMALLINT (bez znaku)

2

0

65 535

MEDIUMINT (ze znakiem)

3

–8 388 608

8 388 607

MEDIUMINT (bez znaku)

3

0

16 777 215

INT (ze znakiem)

4

–2 147 483 648

2 147 483 647

INT (bez znaku)

4

0

4 294 967 295

BIGINT (ze znakiem)

8

–9 223 372 036 854 775 808

9 223 372 036 854 775 807

BIGINT (bez znaku)

8

0

18 446 744 073 709 551 615

BIT (x)

około (x+7)/8

wyzerowane wszystkie bity

ustawione wszystkie bity

DECIMAL(s,p), NUMERIC(s,p)

–10 ^{38} –1

10 ^{38} –1

FLOAT

4

1,79E+308

1,79E+308

DOUBLE [PRECISION], REAL

8

–3,40E+38

3,40E+38

Tabela 10.2. Typy daty i czasu

Typ

Rozmiar w bajtach

DATE

3

DATETIME

8

TIMESTAMP

4

TIME

3

YEAR

1

Tabela 10.3. Typy znakowe

Typ

Rozmiar w bajtach

CHAR(x)

x gdzie 0 <= x <= 255

VARCHAR(x)

x+1 dla x <= 255 i x+2 dla 256 <= x <= 65 535

BINARY(x)

x gdzie 0 <= x <= 255

VARBINARY(x)

x+1 dla x <= 255 i x+2 dla 256 <= x <= 65 535

TINYBLOB, TINYTEXT

x+1 gdzie x < 2 8

BLOB, TEXT

x+2 gdzie x < 2 16

MEDIUMBLOB, MEDIUMTEXT

x+3 gdzie x < 2 24

LONGBLOB, LONGTEXT

x+4 gdzie x < 2 [32]

Instrukcja CREATE TABLE

Wykonanie instrukcji spowoduje dodanie nowej tabeli do bazy danych. Składnię instrukcji poznamy, kolejno tworząc tabele bazy firma.

W pierwszej kolejności należy utworzyć tabele słownikowe — tabele, które nie odwołują się do żadnych innych (listingi 10.2, 10.3).

Listing 10.2. Tworzymy pierwszą tabelę bazy firma. Tabela to po prostu zbiór kolumn, które mają jakąś nazwę i są określonego typu

```
CREATE TABLE seria_wydawnicza

(id_serii int AUTO_INCREMENT PRIMARY KEY,

seria varchar (50) NOT NULL,

uwagi varchar (200))

ENGINE = InnoDB;

Query OK, 0 rows affected (0.11 sec)
```

Listing 10.3. Tworzymy kolejną tabelę słownikową. Tym razem definiujemy ograniczenie typu DEFAULT

```
CREATE TABLE nosnik

(id_nosnika int AUTO_INCREMENT PRIMARY KEY,

nosnik varchar (50) NOT NULL DEFAULT 'Twarda oprawa',

uwagi varchar (250))

ENGINE = InnoDB;

Query OK, 0 rows affected (0.04 sec)
```

Kolumna id_serii została zdefiniowana jako klucz podstawowy tabeli. Dodatkowo wartości w tej kolumnie będą automatycznie generowane przez MYSQL-a. W kolumnie seria nie można przechowywać wartości Null, natomiast w kolumnie uwagi — tak. Ostatni parametr określa typ tworzonej tabeli i w ten sposób determinuje jej funkcjonalność. W ramach kursu będziemy używać tylko najbardziej zgodnych ze standardami ANSI i ISO tabel typu InnoDB.

Po utworzeniu tych dwóch tabel możemy już utworzyć tabelę nadrzędną ksiazka (listing 10.4).

Listing 10.4. Pora utworzyć pierwszą tabelę powiązaną z innymi tabelami

```
CREATE TABLE ksiazka
```

```
(id_ksiazki int AUTO_INCREMENT PRIMARY KEY,

tytul varchar (50) NOT NULL,

isbn char (12) NOT NULL UNIQUE,

liczba_stron smallint NOT NULL CHECK  (liczba_stron BETWEEN 50 AND 3000),

rok_wydania date  NOT NULL ,

isbn_oryginalu char(12) UNIQUE,

id_serii int REFERENCES seria_wydawnicza(id_serii)

ON UPDATE CASCADE ON DELETE NO ACTION,

id_nosnika int REFERENCES nosnik(id_nosnika)

ON UPDATE CASCADE ON DELETE NO ACTION,

uwagi varchar (250),

notka_redakcyjna text)

ENGINE = InnoDB;

Query OK, 0 rows affected (0.03 sec)
```

Przeanalizujmy definicję niektórych kolumn i ograniczeń.

1. Kolumna id_ksiazki int AUTO_INCREMENT PRIMARY KEY — definiujemy klucz podstawowy (a więc w kolumnie id_ksiazki nie będzie można przechowywać wartości NULL i powtarzających się wartości), dodatkowo wymuszając automatyczne generowanie wartości klucza.
2. Kolumna tytul varchar (50) NOT NULL — tytuł będzie wymagany, a jego maksymalny rozmiar to 50 znaków.
3. Kolumna isbn char (12) NOT NULL UNIQUE — ISBN będzie nie tylko wymagany, ale również unikatowy. Każda wartość w tej kolumnie będzie liczyła dokładnie 12 znaków.
4. Kolumna liczba_stron smallint NOT NULL CHECK (liczba_stron BETWEEN 50 AND 3000) — obowiązkowa liczba stron książki musi mieścić się w zakresie od 50 do 3000.
5. Kolumna id_serii int REFERENCES seria_wydawnicza(id_serii) ON UPDATE CASCADE ON DELETE NO ACTION — wartości id_serii muszą dopowiadać jednemu z identyfikatorów serii zapisanych w kolumnie id_serii tabeli seria_wydawnicza. Dodatkowo zmiana identyfikatora serii w jednej tabeli spowoduje automatyczne uaktualnienie tej wartości w powiązanej tabeli. Natomiast usunięcie serii wydawniczej, w której wydano przynajmniej jedną książkę, będzie niemożliwe — najpierw trzeba będzie albo usunąć, albo przypisać do innych serii wszystkie wydane w niej książki.
6. Kolumna id_nosnika int REFERENCES nosnik(id_nosnika) ON UPDATE CASCADE ON DELETE NO ACTION jest podobnie zdefiniowanym kluczem obcym, tym razem wskazującym na typ nośnika.

Utworzenie kolejnej tabeli nie powinno nikomu sprawić kłopotu (listing 10.5).

Listing 10.5. Tworzymy prostą, złożoną z czterech kolumn tabelę miasto

```
CREATE TABLE miasto

(id_miasta int AUTO_INCREMENT PRIMARY KEY,
```

```
miasto varchar (30),

mieszkancow int,

uwagi varchar (250))

ENGINE = InnoDB;

Query OK, 0 rows affected (0.33 sec)
```

Po utworzeniu tabeli miasto możemy już utworzyć odwołującą się do niej tabelę autor (listing 10.6).

Listing 10.6. Utworzenie tabeli autor możemy potraktować jako powtórzenie dotychczas omówionego materiału

```
CREATE TABLE autor

(id_autora int AUTO_INCREMENT PRIMARY KEY,

imie varchar (20) NOT NULL,

nazwisko varchar (25) NOT NULL,

kod char (6),

ulica_nr_domu varchar (50),

telefon varchar (15),

e_mail varchar (40) CHECK (LENGTH(e_mail)-LENGTH(REPLACE(e_mail,'@',''))>0),

bank varchar (40),

konto varchar (30),

id_miasta int REFERENCES miasto(id_miasta)

ON UPDATE CASCADE ON DELETE NO ACTION,

uwagi varchar(250))

ENGINE = InnoDB;

Query OK, 0 rows affected (0.03 sec)
```

Spróbuj samodzielnie utworzyć tabelę wydawnictwo — zacznij od utworzenia dwóch tabel słownikowych osoba i kraj (listing 10.7).

Listing 10.7. Odpowiedź do samodzielnie wykonanego ćwiczenia

```
CREATE TABLE kraj

(id_kraju int AUTO_INCREMENT PRIMARY KEY,

kraj varchar (20) NOT NULL UNIQUE,
```

```
uwagi varchar(250))

ENGINE = InnoDB;

Query OK, 0 rows affected (0.36 sec)


CREATE TABLE osoba

(id_osoby int AUTO_INCREMENT PRIMARY KEY,

imie varchar (20) NOT NULL,

nazwisko varchar (25) NOT NULL,

telefon varchar (15),

e_mail varchar (40) CHECK  (LENGTH(e_mail)-LENGTH(REPLACE(e_mail,'@',''))>0),

prezes bit DEFAULT 0,

uwagi varchar(250))

ENGINE = InnoDB;

Query OK, 0 rows affected (0.11 sec)


CREATE TABLE wydawnictwo

(id_wydawnictwa int AUTO_INCREMENT PRIMARY  KEY,

nazwa varchar (40) NOT NULL,

kod char (6),

ulica_nr_domu varchar (50),

telefon varchar (15),

e_mail varchar (40) CHECK  (LEN(e_mail)-LEN(REPLACE(e_mail,'@',''))>0),

bank varchar (40),

konto varchar (30),

id_miasta int REFERENCES miasto(id_miasta)

ON UPDATE CASCADE ON DELETE NO ACTION,

id_kraju int REFERENCES kraj(id_kraju)

ON UPDATE CASCADE ON DELETE NO ACTION,
```

id_osoby int REFERENCES osoba(id_osoby)

ON UPDATE NO ACTION ON DELETE NO ACTION,

id_prezesa int REFERENCES osoba(id_osoby)

ON UPDATE NO ACTION ON DELETE NO ACTION,

uwagi varchar(250))

ENGINE = InnoDB;

Query OK, 0 rows affected (0.05 sec)

Jedyny typ tabel, których do tej pory nie utworzyliśmy w bazie firma, to tabele łącznikowe. Listing 10.8 pokazuje schemat tworzenia tego typu tabel.

Listing 10.8. Tabela implementująca związek typu „wiele do wielu” zachodzący pomiędzy autorem a książką

CREATE TABLE autor_ksiazka

(id_autora int REFERENCES autor(id_autora)

ON UPDATE CASCADE ON DELETE CASCADE,

id_ksiazki int REFERENCES ksiazka(id_ksiazki)

ON UPDATE CASCADE ON DELETE CASCADE)

ENGINE = InnoDB;

Query OK, 0 rows affected (0.24 sec)

Zwróć uwagę, że w przypadku tabel łącznikowych klucz główny czasem nie jest potrzebny — tabela autor_ksiazka nie przechowuje informacji o żadnych obiektach, których nie ma już w innych tabelach bazy firma. Z tego samego powodu dla tabel słownikowych z reguły definiuje się kaskadowe usuwanie powiązanych wierszy — przecież po skasowaniu danych autora informacja o tym, że napisał on jakieś książki, jest bezużyteczna. **Ograniczenia**

Wiesz już, że dla tabel można zdefiniować ograniczenia — warunki, które określają, jakie dane można w nich zapisywać. Ograniczenia są sprawdzane, zanim dane zostaną wstawione, zmodyfikowane lub usunięte, dzięki czemu serwery bazodanowe nie muszą wycofywać transakcji naruszających spójność danych[1].

Ograniczenia mogą być definiowane podczas tworzenia i modyfikowania tabel. W obu przypadkach definicja ograniczenia może być umieszczona bezpośrednio po kolumnie, z którą ma ono być powiązane, lub po wymienieniu wszystkich kolumn, a więc pod koniec instrukcji CREATE lub ALTER.

[1] Wycofanie transakcji jest zawsze znacznie kosztowniejsze od jej zatwierdzenia. **NOT NULL**

NULL jako symbol reprezentujący brakujące, nieistotne lub nieznane wartości nie powinien być wstawiany do większości kolumn. Na przykład służąca do identyfikacji wierszy tabeli kolumna klucza podstawowego nie powinna zawierać wartości NULL — ponieważ niemożliwe jest odróżnienie jednej wartości NULL od drugiej,

nieokreślony klucz podstawowy uniemożliwiałby jednoznaczne identyfikowanie wierszy.

Również kolumny przechowujące podstawowe atrybuty obiektu nie powinny zawierać wartości NULL. W innym przypadku informacje zapisane w bazie danych będą niekompletne i mało przydatne, na przykład niektóre dane towarów bez nazw i cen nie pozwolą przygotować oferty czy zestawienia rocznej sprzedaży. **Klucz podstawowy**

Tabela może mieć tylko jeden klucz podstawowy (PK — skrót od ang. Primary Key). Ponieważ jest on wykorzystywany do identyfikowania wierszy tej tabeli, wartości klucza podstawowego nie mogą być nieokreślone i muszą być niepowtarzalne. Poszczególne serwery bazodanowe mają własne mechanizmy generowania wartości — serwer MySQL używa do tego wewnętrznych liczników i słowa kluczowego AUTO_INCREMENT.
Niepowtarzalność

W przeciwieństwie do klucza podstawowego, który może być tylko jeden, ograniczenie niepowtarzalności (ang. Unique) możemy zdefiniować dla dowolnej liczby kolumn tabel. Większość serwerów bazodanowych automatycznie indeksuje kolumny, w których nie można zapisywać duplikatów danych.

Ograniczenie niepowtarzalności w odróżnieniu od ograniczenia klucza podstawowego nie uniemożliwia zapisywania wartości NULL. Serwery jednak w różny sposób traktują ją w ograniczeniu niepowtarzalności, na przykład serwer MySQL uznaje wartości NULL za różne i w konsekwencji w kolumnie z nałożonym ograniczeniem niepowtarzalności wartość NULL może się powtórzyć (listing 10.9).

Listing 10.9. Pseudowartości NULL nie są sobie równe

```
CREATE TABLE t1

(a INT AUTO_INCREMENT PRIMARY KEY,

b INT UNIQUE);

Query OK, 0 rows affected (0.07 sec)



INSERT INTO t1 (b)

VALUES (NULL),(NULL);

Query OK, 2 rows affected (0.01 sec)

Records: 2  Duplicates: 0  Warnings: 0



SELECT *

FROM t1;

+---+------+

| a | b    |

+---+------+
```

| 1 | NULL |

| 2 | NULL |

+---+------+ **Wartość domyślna**

Każdej kolumnie tabeli można nadać jedno ograniczenie wartości domyślnej (ang. Default). Tego typu ograniczenia sprawdzane są tylko podczas wstawiania wierszy. Jeżeli użytkownik nie wstawi danych do kolumny z ograniczeniem wartości domyślnej, serwer bazodanowy zrobi to za niego. **Warunek logiczny**

Najbardziej uniwersalnym zawężeniem jest warunek logiczny (ang. Check), który musi być prawdziwy, żeby operacja wstawiania, modyfikowania czy usuwania danych zakończyła się powodzeniem. Dla każdej kolumny tabeli można zdefiniować wiele warunków, można też tworzyć złożone warunki za pomocą operatorów algebry Boole'a: NOT, AND oraz OR.

W ramach warunku logicznego niemożliwe jest odwoływanie się do tabel innych niż ta, dla której warunek został zdefiniowany, oraz używanie podzapytań. **Klucz obcy**

Ograniczenie klucza obcego (ang. Foreign Key) pozwala na automatyczne sprawdzanie spójności danych przechowywanych w powiązanych ze sobą tabelach. Jeżeli tabele powiązane są związkiem typu „jeden do wielu" (np. jeden towar składa się z wielu części, ale ta sama część może być użyta tylko w jednym produkcie), do tabeli podrzędnej (w tym przypadku do tabeli Produkcja.Części) należy dodać kolumnę, w której zapisane zostaną identyfikatory towarów. Żeby dane były spójne, wartości klucza obcego:

1.

muszą odpowiadać jednej z wartości powiązanego z nim klucza podstawowego (przy każdej części musi być zapisany identyfikator istniejącego towaru, w którym ta część jest użyta)

2.

lub być nieokreślone (oznacza to, że dana część nie jest używana w żadnym z towarów).

Tworzenie tabel poprzez zapytanie

W języku SQL można w łatwy sposób utworzyć tabele na podstawie innej tabeli, istniejącej już w bazie. Wynik zapytania (instrukcji SELECT) może być zapisany w bazie jako nowa tabela. Składnia tak zmodyfikowanej instrukcji CREATE TABLE wygląda następująco:

CREATE TABLA Nazwa tabeli

[(nazwa kolumny [NULL | NOT NULL],...)]

AS SELECT zapytanie

Lista kolumn tworzonej tabeli może być pominięta — w takim wypadku zostaną wykorzystane nazwy kolumn zwrócone przez instrukcję SELECT. W przeciwnym razie liczba kolumn zwróconych przez instrukcję SELECT musi być równa liczbie kolumn podanej w klauzuli CREATE TABLE.

Aby na przykład utworzyć nową tabelę zawierającą jedynie imiona i nazwiska osób, należy wykonać instrukcję (listing 10.10):

Listing 10.10. Tworzenie tabeli na podstawie wyniku zapytania

CREATE TABLE publiczne_dane

SELECT imie, nazwisko

FROM osoba;

Query OK, 0 rows affected (0.03 sec)

Records: 0 Duplicates: 0 Warnings: 0

Tak utworzona kopia danych nie jest automatycznie synchronizowana z oryginałem. Jeżeli dopiszemy do tabeli osoba, usuniemy z niej lub zmienimy w niej jakiś dane, to informacje w tabeli publiczne_dane pozostaną niezmienione. Żeby uzyskać efekt automatycznej synchronizacji, należy utworzyć opisywane w następnym odcinku widoki (perspektywy).

Modyfikowanie tabel

Raz utworzone tabele mogą być zmieniane — zawsze możemy dodać lub usunąć kolumnę czy zawężenie, możemy również zmodyfikować istniejące kolumny i zawężania, wyłączyć niektóre zawężenia albo zmienić nazwę tabeli lub poszczególnych kolumn. Wszystkie te zmiany możemy przeprowadzić, wykonując instrukcję ALTER TABLE. Wykonując kod z listingu 10.11, zmodyfikujemy przykładową tabelę.

Listing 10.11. Najpierw usuwamy jedną kolumnę, potem zmieniamy typ i nazwę pozostałej kolumny, a na końcu nazwę całej tabeli

ALTER TABLE publiczne_dane

DROP COLUMN imie;

Query OK, 0 rows affected (0.08 sec)

Records: 0 Duplicates: 0 Warnings: 0

ALTER TABLE publiczne_dane

CHANGE nazwisko login CHAR(20);

Query OK, 0 rows affected (0.38 sec)

Records: 0 Duplicates: 0 Warnings: 0

ALTER TABLE publiczne_dane

RENAME TO loginy;

Query OK, 0 rows affected (0.02 sec)

Usuwanie tabel

Niepotrzebne tabele powinny być usunięte. Aby usunąć tabelę, należy wykonać instrukcję DROP TABLE (listing 10.12).

Listing 10.12. Usunięcie tabeli automatycznie kasuje wszystkie zapisane w niej dane

DROP TABLE loginy;

Query OK, 0 rows affected (0.02 sec)

Indeksy

Jedynym powodem tworzenia indeksów jest poprawa wydajności bazy danych. Indeksy, tak jak statystyki, nie wpływają na wynik instrukcji języka SQL, a jedynie na plan i czas ich wykonania. Jeżeli nie istnieją indeksy, wyszukanie pojedynczej wartości wiąże się z koniecznością odczytania całej tabeli. Wynika z tego, że brak indeksu może oznaczać odczytanie megabajtów danych tylko po to, aby np. znaleźć numer czyjegoś telefonu.

Indeksy organizują dane w sposób umożliwiający ich wydajne odczytywanie i modyfikowanie. Chociaż bez indeksów serwery bazodanowe też są w stanie odczytać wszystkie dane wymagane do wykonania instrukcji SQL, to indeksy wielokrotnie skracają czas takich operacji.

Jeżeli natomiast istnieją powiązane z tabelą indeksy, znalezienie żądanych danych sprowadza się do znalezienia w indeksie (który z reguły jest obiektem wielokrotnie mniejszym niż tabela) kluczy spełniających podane kryteria i odczytania wyłącznie tych wierszy tabeli, na które wskazują znalezione klucze indeksu.

Indeksy niezgrupowane pełnią zbliżoną funkcję do skorowidzów w książce. Zamiast szukać jakiegoś terminu w całej książce, znacznie szybciej i łatwiej jest znaleźć go w ułożonym alfabetycznie skorowidzu. Po znalezieniu terminu wystarczy odczytać numer strony, na której został on opisany, i otworzyć książkę na tej stronie. Indeksy zgrupowane przypominają książkę telefoniczną, w której dane są uporządkowane według wybranego klucza (np. nazwiska). Dzięki temu, gdy chcemy znaleźć numer telefonu, wystarczy, że serwer bazodanowy odczyta stronę (lub strony), na której znajdują się dane osób o szukanym nazwisku.

Typ możliwych do utworzenia indeksów zależy od typu tabeli. Dla tabel typu InnoDB można tworzyć indeksy typu B-tree, czyli indeksy zapisane jako zrównoważone drzewa binarne. Tego typu indeksy doskonale nadają się do wyszukiwania określonych wartości (koszt przeszukiwania drzewa binarnego jest równy logarytmowi jego elementów, a więc w celu znalezienia jednego z 10 000 elementów wystarczy wykonać $\log_2 10000$, czyli średnio 13 operacji).

Główną zaletą indeksów jest ograniczenie danych odczytywanych z dysków. Różnice w czasie wykonania tej samej instrukcji bez właściwych indeksów i z nimi mogą być ogromne — w pierwszym przypadku konieczne jest odczytanie wszystkich stron tabeli nawet wtedy, gdy interesuje nas jeden z milionów wierszy, w drugim przypadku serwer bazodanowy znajdzie potrzebne dane w indeksie.

Jeżeli jednak indeks nie zawiera wszystkich danych potrzebnych do wykonania instrukcji, serwer bazodanowy będzie musiał je odczytać z tabeli. Ta operacja sięgania po dane (ang. Lookup) może okazać się bardzo kosztowna. Dlatego serwery bazodanowe używają indeksów niezawierających zapytania tylko wtedy, gdy są one wystarczająco selektywne, czyli zwracają mniej niż kilka procent ze wszystkich wierszy tabeli[2].

Drugą zaletą indeksów jest to, że odczytując je, serwer otrzymuje posortowane dane. Sortowanie ogromnych zbiorów danych jest czasochłonną i wymagającą dużych ilości pamięci operacją, która jest wykonywana nie tylko na potrzeby klauzuli ORDER BY, ale również w niektórych typach łączenia i grupowania danych.

Indeksy mają też wady — muszą być na bieżąco aktualizowane i zajmują dodatkowe miejsce. Każde wstawienie, usunięcie czy aktualizacja danych w poindeksowanej tabeli wiąże się z aktualizacją wszystkich zdefiniowanych dla tej tabeli indeksów.

Składnia instrukcji tworzenia indeksu wygląda następująco:

CREATE [UNIQUE] INDEX nazwa indeksu ON tablica (kolumna)

Aby na przykład utworzyć indeks dla kolumny imie tabeli osoba, napiszemy (listing 10.13):

Listing 10.13. Tworzymy indeks zawierający posortowaną listę imion

```
CREATE INDEX Indeks_Imie ON osoba (imie);

Query OK, 0 rows affected (0.35 sec)

Records: 0  Duplicates: 0  Warnings: 0
```

MySQL umożliwia również tworzenie indeksów podczas tworzenia tabel. W takim przypadku po podaniu typu kolumny należy użyć słowa kluczowego INDEX, ewentualnie określić typ indeksu i wskazać indeksowaną kolumnę (listing 10.14).

Listing 10.14. Tymczasowa tabela utworzona razem z indeksem na jednej kolumnie

```
CREATE TEMPORARY TABLE lookup

(id int, INDEX USING BTREE (id))

ENGINE = MEMORY;

Query OK, 0 rows affected (0.03 sec)
```

[2] Dla serwera SQL 2008 próg selektywności wynosi około 1%.

Indeksy unikatowe

Jeżeli dla jakiejś kolumny zdefiniowaliśmy zawężenie UNIQUE, nie należy tworzyć dla niej dodatkowego indeksu — MySQL automatycznie utworzył odpowiedni indeks. Możemy utworzyć unikatowy indeks, ale z reguły bardziej eleganckim rozwiązaniem będzie nałożenie odpowiedniego zawężania. Indeksy unikatowe powinny być tworzenie jedynie jako indeksy kompozytowe.

Indeksy kompozytowe

Indeksy mogą być tworzone dla jednej lub kilku kolumn. Kolumny przechowujące wartości, do których odwołujemy się najczęściej jednocześnie (np. imię i nazwisko), powinny być powiązane z jednym indeksem kompozytowym. W ten sposób utworzymy indeks zawierający zapytanie, co jest optymalną metodą poprawiania wydajności bazy danych. W takim przypadku MySQL w celu wykonania zapytania w ogóle nie będzie musiał odczytywać tabeli — wszystkie potrzebne dane są przecież zapisane w indeksie.

Wszystkie kolumny wchodzące w skład indeksu kompozytowego muszą należeć do tej samej tabeli.

Kolejność, w jakiej wymienione są poszczególne kolumny indeksu kompozytowego, ma kluczowe znaczenie dla wykorzystania indeksu przez MySQL (z reguły jako pierwszą należy określić kolumnę przechowującą bardziej różnorodne wartości) — tak jak książka telefoniczna jest dla nas bezużyteczna, jeżeli znamy jedynie imię, a nie nazwisko osoby, tak optymalizator nie jest w stanie skorzystać z indeksu kompozytowego uporządkowanego według wartości kolumny indeksu innej niż pierwsza (listing 10.15).

Listing 10.15. Indeks, który wielokrotnie skróci czas wyszukiwania tytułu książki o znanym numerze ISBN

```
CREATE UNIQUE INDEX Ix_isbn_tyt

ON ksiazka(isbn, tytul);

Query OK, 0 rows affected (0.12 sec)
```

Records: 0 Duplicates: 0 Warnings: 0

Dla których kolumn tworzyć indeksy?

Indeksy skracają czas potrzebny na wyszukanie i odczytanie żądanych danych. Jednak mogą wydłużyć czas operacji wstawiania i modyfikowania danych. Zmiana lub dodanie danych do poindeksowanych kolumn powoduje automatyczną aktualizację indeksu, z drugiej strony jednak duża część operacji aktualizacji danych wymaga również ich odczytania (np. instrukcja UPDATE z klauzulą WHERE), a koszt znalezienia poindeksowanych danych i aktualizacji zarówno danych, jak i indeksu może okazać się niższy niż koszt wyszukania danych na stronach tabeli.

W typowych bazach nie należy tworzyć więcej niż 8 – 10 indeksów dla jednej tabeli.

Po wybraniu kolumny (lub kolumn), dla której zostanie utworzony indeks grupujący, należy określić kolumny powiązane z indeksami niegrupującymi. Odpowiednie do tego celu są kolumny, które:

1.

Przechowują wartości częściej odczytywane niż modyfikowane.

2.

Wykorzystywane są do wyszukiwania danych. Im więcej kolumn indeksowanych pojawia się w klauzuli WHERE zapytania, tym lepsza jest jego wydajność. W przypadku pól, których wartości nigdy nie są sprawdzane w kryteriach zapytań, tworzenie indeksu z reguły nie ma sensu.

3.

Są regularnie wykorzystywane do łączenia tabel. Łączenie tabel poprzez kolumny indeksowane znacznie przyspiesza tę operację. Wartości w polach indeksowanych są uporządkowane, więc można znacznie szybciej dopasować poszczególne wartości pochodzące z kolumn poindeksowanych.

4.

Przechowują różnorodne wartości, dzięki czemu ta sama wartość indeksu nie wskazuje na dużą liczbę wierszy tabeli. Jeśli w kolumnie tabeli znajdują się tylko dwie różne wartości (jak na przykład w kolumnie płeć), to utworzenie indeksu dla tej kolumny prawie na pewno jest błędem — MySQL i tak z niego nie skorzysta w zapytaniach, za to indeks będzie zajmował miejsce na dysku, a każda zmiana danych w tabeli będzie związana ze zmianą indeksu.

W praktyce indeksy tworzy się dla:

1.

kolumn, na które nałożono ograniczenie typu PRIMARY KEY,

2.

kolumn, na które nałożono ograniczenie typu FOREIGN KEY, oraz kolumn wykorzystywanych przy łączeniu tabel.

3.

kolumn przechowujących dane wykorzystywane jako argumenty wyszukiwania,

4.

kolumn przechowujących często sortowane dane.

Natomiast nie zaleca się zakładania indeksów dla tabel, które:

1.

często są modyfikowane,
2.

rzadko biorą udział w zapytaniach,
3.

przechowują niezbyt różnorodne dane lub wiele duplikatów danych.

Usuwanie indeksów

Niepotrzebne indeksy powinny być usuwane. Służy do tego instrukcja DROP INDEX (listing 10.16).

Listing 10.16. Usuwamy wcześniej utworzony indeks

```
DROP INDEX ix_isbn_tyt ON ksiazka;

Query OK, 0 rows affected (0.34 sec)

Records: 0  Duplicates: 0  Warnings: 0
```