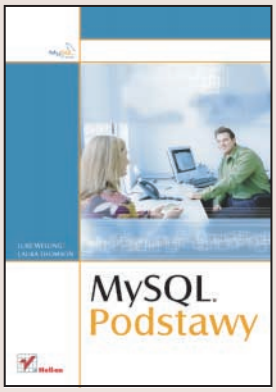
MySQL. Podstawy



Autorzy: Luke Welling, Laura Thomson
Tłumaczenie: Marek Koszykowski
ISBN: 83-7361-689-6
Tytuł oryginału: MySQL Tutorial
Format: B5, stron: 280

Poznaj ogromne możliwości MySQL-a

„MySQL. Podstawy” to wprowadzenie do pracy z MySQL. Opisuje system od strony użytkowej – przedstawia zasady tworzenia baz i tabel, definiowania indeksów i kluczy oraz stosowania języka SQL do wprowadzania danych i wybierania ich z tabel. Omawia również zagadnienia bardziej zaawansowane – mechanizmy transakcji. Jeśli chcesz dowiedzieć się czegoś o administrowaniu bazą MySQL, znajdziesz tu również informacje na ten temat – od sposobów definiowania uprawnień użytkowników aż po precyzyjne „strojenie” bazy za pomocą odpowiednich opcji konfiguracyjnych, wykonywania kopii bazy oraz zabezpieczanie zgromadzonych w niej danych przed atakami z zewnątrz.

- Instalacja MySQL
- Struktura katalogów i plików w bazie MySQL
- Tworzenie baz, tabel i indeksów
- Operacje na danych z wykorzystaniem języka SQL
- Złożone zapytania
- Typy tabel i transakcje
- Administrowanie kontami użytkowników
- Administrowanie bazą danych
- Tworzenie kopii zapasowych baz danych
- Zabezpieczanie danych
- Optymalizacja serwera i baz danych

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

Spis treści

Rozdział 5.

Wstawianie, usuwanie i aktualizacja danych

W tym rozdziale zobaczymy, w jaki sposób wstawiać i zmieniać dane w bazie danych MySQL przy użyciu instrukcji `INSERT`, `DELETE` oraz `UPDATE`.

Omówimy następujące zagadnienia:

- Używanie instrukcji `INSERT`
- Używanie instrukcji `DELETE`
- Używanie instrukcji `UPDATE`
- Wczytywanie danych za pomocą instrukcji `LOAD DATA INFILE`
- Korzystanie z instrukcji `REPLACE` i `TRUNCATE`

Przeszliśmy teraz do stosowania języka wybierania i manipulowania danymi (DML). Gdy nauczymy się wstawiać dane do baz danych, w następnych kilku rozdziałach poznamy różne sposoby pobierania danych z tych baz.

Instrukcja INSERT

Instrukcja `INSERT` jest używana do wstawiania wierszy do tabeli. Zacznijmy od przykładu. Jak już wcześniej mówiliśmy, instrukcje można wpisywać bezpośrednio w programie MySQL Monitor lub zapisywać je w pliku.

Przykłady instrukcji `insert` są pokazane w listingu 5.1.

Listing 5.1. *pracownik_dane.sql*

```
use pracownik;

delete from wydzial;
```

```
insert into wydzial values
(42, 'Finanse'),
(128, 'Badania i Rozwój'),
(NULL, 'Kadry'),
(NULL, 'Marketing');

delete from pracownik;
insert into pracownik values
(7513, 'Ewa Nowacka', 'programista', 128),
(9842, 'Bartosz Kowalski', 'administrator baz danych', 42),
(6651, 'Andrzej Plater', 'programista', 128),
(9006, 'Barbara Cetryk', ' administrator systemów', 128);

delete from umiejetnosciPracownika;
insert into umiejetnosciPracownika values
(7513, 'C'),
(7513, 'Perl'),
(7513, 'Java'),
(9842, 'DB2'),
(6651, 'VB'),
(6651, 'Java'),
(9006, 'NT'),
(9006, 'Linux');

delete from klient;
insert into klient values
(NULL, 'Telekom SA', 'ul. Nowa 1 Warszawa', 'Jan Nowak', '95551234'),
(NULL, 'Bank', 'ul. Brzozowa 100 Warszawa', 'Lech Turski', '95551234');

delete from przydzial;
insert into przydzial values
(1, 7513, '2003-01-20', 8.5);
```

Widzimy, że zanim wstawimy dane do każdej tabeli, używamy polecenia `DELETE`. Nie jest to konieczne, ale w ten sposób zostaną usunięte ewentualne dane próbne, które mogły już zostać umieszczone w tabeli. Do instrukcji `DELETE` przejdziemy w dalszym podrozdziale.

Zauważmy również, że wstawiliśmy te same dane, których używaliśmy w przykładach z rozdziału 3., „Błyskawiczny kurs tworzenia bazy danych". Dodaliśmy też nowe wiersze.

Wszystkie instrukcje `INSERT` są bardzo podobne. Spójrzmy na pierwszą, aby zobaczyć, jak działa.

```
insert into wydzial values
(42, 'Finanse'),
(128, 'Badania i Rozwój'),
(NULL, 'Kadry'),
(NULL, 'Marketing');
```

Tabelę, do której chcemy wstawić dane, określamy w pierwszym wierszu — w tym przykładzie jest to tabela `wydzial`. Umieszczamy w niej cztery wiersze. Jak pamiętamy, tabela `wydzial` miała dwie kolumny — `IDwydzialu` i `nazwa` (można sprawdzić to samodzielnie, używając polecenia `describe wydzial`).

W pierwszych dwóch wierszach określiliśmy wartość identyfikatora wydziału, której chcieliśmy użyć. Spójrzmy jeszcze raz na definicję `IDwydzialu`. Przypomnijmy, że w ostatnim rozdziale deklarowaliśmy identyfikator wydziału jako:

```
IDwydzialu int not null auto_increment primary key
```

Ponieważ jest to kolumna `auto_increment`, możemy ustalić wartość lub pozwolić, aby MySQL wybrał ją za nas (zwykle w takich kolumnach nie podaje się liczby samemu, ale mogą istnieć sytuacje takie jak ta, w której mamy już istniejącą wartość do zastosowania).

W wierszach z wydziałami `Kadry` oraz `Badania i Rozwój` pozostawiliśmy kolumnę `ID➥wydzialu` pustą (ma wartość `NULL`). Zadziała wówczas opcja `auto_increment`, powodując przydzielenie odpowiednich wartości. Zobaczmy, jaki będzie wynik działania instrukcji `INSERT`.

Jeżeli przejrzymy różne instrukcje `INSERT` z przykładu, zobaczymy, że dane typu łańcucha znakowego i daty ujęliśmy w apostrofy, na przykład `'Badania i Rozwój'`. Z kolei danych typu liczbowego nie należy ujmować w apostrofy.

A co powinniśmy zrobić, gdy dane, które znajdują się między apostrofami, same zawierają apostrofy? Odpowiedzią jest oznaczenie apostrofów *znakami sterującymi*. Mówiąc prosto, powinniśmy wpisać lewy ukośnik (`\`) przed znakiem apostrofu, na przykład `'O\'Leary'`.

Oczywiście rodzi się następne pytanie — Co zrobić, jeżeli chcemy wstawić lewy ukośnik jako zwykły znak, a nie znak o specjalnym znaczeniu? Musimy uciec od niego w ten sam sposób — zamiast jednego ukośnika, powinniśmy wpisać dwa (`\\`).

Dane z bazy danych pobieramy przy użyciu instrukcji `SELECT`. Tę instrukcję omówimy obszernie w kolejnych kilku rozdziałach. Teraz wystarczy, abyśmy wiedzieli, że wpisanie

```
select * from nazwatab;
```

zwróci wszystkie dane obecnie przechowywane w tabeli.

Jeżeli wpiszemy:

```
select * from wydzial;
```

powinien pojawić się następujący wynik:

```
+------------+------------------+
| IDwydzialu | nazwa            |
+------------+------------------+
|         42 | Finanse          |
|        128 | Badania i Rozwój |
|        129 | Kadry            |
|        130 | Marketing        |
+------------+------------------+
4 rows in set (0.01 sec)
```

Możemy zauważyć, że wynikiem działania opcji `auto_increment` jest wartość o jeden większa niż największa wartość w kolumnie.

Ogólny wzorzec instrukcji INSERT według podręcznika MySQL jest następujący:

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] nazwa_tab [(nazwa_kol,…)]
    VALUES ((wyrazenie | DEFAULT),…),(…),…
    [ON DUPLICATE KEY UPDATE nazwa_kol=wyrazenie, …]

lub  INSERT [[LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] nazwa_tab [(nazwa_kol,…)]
    SELECT …

lub  INSERT [[LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] nazwa_tab
    SET nazwa_kol=(wyrazenie | DEFAULT), …
    [ON DUPLICATE KEY UPDATE nazwa_kol=wyrazenie, …]
```

Wszystkie przykłady, które widzieliśmy do tej pory, są zgodne z pierwszym wzorcem. Zauważmy, że słowo kluczowe INTO jest opcjonalne. Moglibyśmy je opuścić i zacząć instrukcję od `insert pracownik values`, ale uważamy, że dzięki słowu INTO powstała składnia jest nieco łatwiejsza do zrozumienia dla osób znających angielski.

Korzystając z pierwszego wzorca, musimy podać wszystkie wartości dla każdej kolumny w każdym wierszu w tej samej kolejności, w jakiej kolumny są umieszczone w tabeli. Musimy, na przykład, najpierw podać IDwydzialu, a potem nazwę, ponieważ taka jest struktura tabeli wydzial. Jak pokazano, ten wzorzec pozwala wstawić wiele wierszy do tabeli za pomocą pojedynczej instrukcji INSERT.

Drugi wzorzec kończy się instrukcją SELECT. Zamiast wstawiania wartości ręcznie, umożliwia nam pobranie danych z innej tabeli lub tabel bazy danych i zachowanie tych wartości w tabeli.

Trzeci wzorzec pozwala sprecyzować, do których kolumn mają być wstawiane dane. Przykład użycia instrukcji INSERT, zgodnej z tym wzorcem, jest następujący:

```
insert into wydzial
set nazwa='Zarządzanie aktywami';
```

W instrukcji zgodnej z tym wzorcem można wstawić za jednym razem tylko jeden wiersz, ale nie jest konieczne podawanie wartości dla wszystkich kolumn. W naszym przykładzie sprecyzowaliśmy tylko wartość dla pola nazwa. Wszystkie niepodane wartości przyjmą albo wartość domyślną, jeżeli została określona, albo wartość NULL. W tym przykładzie IDwydzialu przyjmie wartość NULL, co sprawi, że zadziała magia opcji auto_increment i wygenerowana zostanie wartość dla identyfikatora wydziału (można to sprawdzić, wpisując jeszcze raz `select * from wydzial`).

W instrukcji INSERT występuje wiele klauzul opcjonalnych. Przejrzyjmy je krótko, aby zorientować się, do czego służą:

- Możemy zastosować opcje LOW PRIORITY lub DELAYED. Powodują one, że wstawianie będzie opóźnione do momentu, aż żaden klient nie będzie odczytywał tabeli. Różnica między nimi jest taka, że jeżeli użyjemy opcji LOW PRIORITY, możemy być zmuszeni odczekać pewien czas, zanim będziemy mogli wprowadzić następne zapytanie. Przy opcji DELAYED otrzymamy komunikat OK

i będziemy mogli kontynuować wprowadzanie zapytań. Jednak powinniśmy pamiętać, że wstawienie zostanie wykonane, dopiero gdy tabela przestanie być używana.

- Opcja `IGNORE` jest przydatna zwłaszcza przy wstawianiu wielu wierszy. Zazwyczaj gdy jeden z wierszy, który próbujemy dodać, koliduje z istniejącym kluczem podstawowym lub wartością unikalną, wystąpi błąd i wstawianie zostanie anulowane. Jeżeli użyjemy opcji `IGNORE`, błąd zostanie zignorowany, a wstawianie będzie kontynuowane i nastąpi próba dodania kolejnego wiersza.
- Możemy sprawić, aby kolumnie przypisana była wartość domyślna, używając opcji `DEFAULT` do określenia tej wartości.
- Opcja `ON DUPLICATE KEY UPDATE` pozwala poradzić sobie z kolizjami kluczy podstawowych i wartości unikalnych. Po tym wyrażeniu wpisujemy instrukcję `update`, której możemy użyć do takiej zmiany wartości unikalnej (lub klucza podstawowego) obecnej w tabeli, aby nie kolidowała ona już z wstawianym wierszem.

  Poniższy krótki przykład ilustruje typowe użycie opcji `ON DUPLICATE KEY UPDATE`:

```
create table ostrzezenie
(IDpracownika int primary key not null references pracownik(IDpracownika),
licznik int default 1
) type =InnoDB;

insert into ostrzezenie (IDpracownika)
values (6651)
on duplicate key update licznik=licznik+1;
```

  Opcja ta jest przydatna nie tylko w sytuacjach związanych z rekordami unikalnymi, ale również gdy chcemy wykonać jakieś działanie, takie jak zwiększenie licznika dla nieunikalnych zdarzeń. Dobrym przykładem przydatności tej opcji jest dowolny rodzaj logowania. Jednak aby pozostać przy przykładzie bazy danych `pracownik`, w tabeli `ostrzezenie` będziemy przechowywać identyfikatory pracowników, którzy otrzymali ostrzeżenie.

  Aby nanosić otrzymane ostrzeżenia, uruchamiamy powyższą instrukcję `insert`. Ponieważ wartość domyślna pola `licznik` jest równa `1`, a przy wstawianiu nie podajemy innej jego wartości, dlatego dla każdego pracownika wartość tego pola przy pierwszym wstawieniu będzie równa `1`. Kolejne uruchomienia instrukcji `insert` dla tego samego `IDpracownika` spowodują włączanie opcji `ON DUPLICATE KEY UPDATE`, a przez to zwiększanie licznika.

Instrukcja REPLACE

Instrukcja `REPLACE` działa podobnie jak instrukcja `INSERT`. Różni się od niej wyłącznie tym, że gdy przy wstawianiu wartości pojawi się kolizja klucza, nowy wiersz zastąpi istniejący.

Ogólny wzorzec instrukcji INSERT według podręcznika MySQL jest następujący:

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] nazwa_tab [(nazwa_kol,…)]
    VALUES ((wyrazenie | DEFAULT),…),(…),…

lub  REPLACE [[LOW_PRIORITY | DELAYED]
    [INTO] nazwa_tab [(nazwa_kol,…)]
    SELECT …

lub  REPLACE [[LOW_PRIORITY | DELAYED]
    [INTO] nazwa_tab
    SET nazwa_kol=(wyrazenie | DEFAULT), …
```

Podobieństwo do instrukcji `INSERT` powinno być oczywiste.

Instrukcja DELETE

Instrukcja SQL `DELETE` pozwala usunąć wiersze z tabeli. W listingu 5.1 na przykład występuje taka instrukcja:

```
delete from wydzial;
```

W tym przykładzie instrukcja `delete` spowoduje usunięcie wszystkich wierszy z tabeli `wydzial`. Możemy też podać, które wiersze mają zostać usunięte, używając opcji `WHERE`, na przykład:

```
delete from wydzial where nazwa='Zarządzanie aktywami';
```

Zostaną wówczas usunięte tylko te wiersze, które spełniają kryteria zawarte w klauzuli `WHERE`. W tym przykładzie zostaną usunięte tylko te wiersze, dla których nazwa wydziału to `'Zarządzanie aktywami'`.

Rzadko są usuwane wszystkie wiersze tabeli. Jednak ponieważ jest to najkrótsza forma instrukcji `delete`, może się czasem zdarzyć, że przez przypadek wpiszemy instrukcję `delete` bez klauzuli `WHERE`. Możemy zaoszczędzić sobie takich problemów, włączając opcję `--safe-updates` lub `--i-am-a-dummy` w wierszu polecenia klienta *mysql* (jak to zostało omówione w rozdziale 2., „Szybkie wprowadzenie”.) Te opcje zapobiegają usunięciu (lub aktualizacji) wierszy bez podania ograniczenia klucza w warunku `WHERE`. Oznacza to, że należy uściślić zamiar usunięcia wierszy przez podanie określonej wartości klucza.

W podręczniku MySQL ogólny wzorzec instrukcji `DELETE` jest następujący:

```
DELETE [LOW_PRIORITY] [QUICK] FROM nazwa_tabeli
       [WHERE definicja_where]
       [ORDER BY …]
       [LIMIT wiersze]

lub
```

```
DELETE [LOW_PRIORITY] [QUICK] nazwa_tabeli[.*] [, nazwa_tabeli[.*] …]
       FROM odwolanie-do-tabeli
       [WHERE definicja_where]

lub

DELETE [LOW_PRIORITY] [QUICK]
       FROM nazwa_tabeli[.*] [, nazwa_tabeli[.*] …]
       USING odwolanie-do-tabeli
       [WHERE definicja_where]
```

Do tej pory używaliśmy pierwszego wzorca instrukcji `DELETE`.

Pozostałe dwa wzorce są zaprojektowane w celu umożliwienia usunięcia wierszy z jednej lub większej ilości tabel przy wykorzystaniu odwołania do innych tabel, na przykład:

```
delete pracownik, umiejetnosciPracownika
from pracownik, umiejetnosciPracownika, wydzial
where pracownik.IDpracownika = umiejetnosciPracownika.IDpracownika
and pracownik.IDwydzialu = wydzial.IDwydzialu
and wydzial.nazwa = 'Finanse';
```

W tym przykładzie usunięto wszystkich pracowników, którzy pracują w wydziale `Fin➥anse` i usunięto wszystkie rekordy zawierające umiejętności tych pracowników. Zauważmy, że wiersze są usuwane z tabel `pracownik` i `umiejetnosciPracownika` (tabele wymienione w początkowej części klauzuli `delete`), ale nie `wydzial` (ponieważ ta tabela jest wymieniona tylko w klauzuli `from`).

Z tabel wymienionych w początkowej klauzuli `delete` zostaną usunięte wiersze. Natomiast tabele wymienione w wyrażeniu `from` zostaną użyte do wyszukania danych i nie zostaną z nich usunięte wiersze, jeżeli nie wymieniono ich w klauzuli `delete`.

Zauważmy, że przykład ten jest dosyć złożony, ponieważ wykorzystane są w nim trzy tabele! Tylu tabel jednak potrzebowaliśmy do zilustrowania działania instrukcji `delete`. Zalecamy ponowne przejrzenie klauzuli `WHERE` po przeczytaniu informacji na temat złączeń w rozdziale 7., „Zapytania zaawansowane”.

Użyliśmy kilku nowych elementów w warunku `WHERE` — operatora `and` oraz zapisu *`tabela.kolumna`*. Zastosowaliśmy operator `and`, aby połączyć nasze warunki. `And` jest to prosty operator logiczny *I*. Użyliśmy również zapisu `pracownik.IDpracownika`. Do obu tych zagadnień powrócimy dokładniej w następnych dwóch rozdziałach.

Trzeci wzorzec `DELETE` jest podobny do drugiego. Różni się od niego tym, że usuwamy tylko tabele wymienione w wyrażeniu `FROM`, odwołując się do tabel z wyrażenia `USING`, na przykład:

```
delete from pracownik, umiejetnosciPracownika
using pracownik, umiejetnosciPracownika, wydzial
where pracownik.IDpracownika = umiejetnosciPracownika.IDpracownika
and pracownik.IDwydzialu = wydzial.IDwydzialu
and wydzial.nazwa = 'Finanse';
```

Ta instrukcja oznacza to samo co poprzednia, inna jest tylko jej składnia.

W ogólnym wzorcu instrukcji DELETE występują też inne klauzule opcjonalne:

- Klauzula LOW_PRIORITY działa w taki sam sposób, jak w instrukcji INSERT.
- Podanie opcji QUICK może przyspieszyć działanie instrukcji DELETE, ponieważ wówczas nie będzie wykonywana standardowa obsługa indeksów w czasie usuwania danych z tabeli.
- Klauzula ORDER BY ustala kolejność usuwania wierszy. Jest ona najbardziej przydatna w połączeniu z klauzulą LIMIT — na przykład, możemy chcieć usunąć z tabeli *n* najstarszych wierszy.
- Klauzula LIMIT pozwala nam podać maksymalną liczbę wierszy tabeli, które mogą zostać usunięte przez instrukcję DELETE. Klauzula ta jest najbardziej funkcjonalna w połączeniu z wyrażeniem ORDER BY; chroni także przed usunięciem zbyt wielu wierszy.

Instrukcja TRUNCATE

Instrukcja TRUNCATE umożliwia nam usunięcie wszystkich wierszy w tabeli, na przykład:

```
TRUNCATE TABLE pracownik;
```

To zapytanie usunęłoby wszystkich pracowników z tabeli pracownik. Jest ono szybsze niż instrukcja DELETE, ponieważ powoduje usunięcie tabeli i utworzenie nowej — pustej. Należy pamiętać o tym, że TRUNCATE nie zapewnia bezpieczeństwa właściwego dla transakcji.

Instrukcja UPDATE

Możemy użyć instrukcji SQL UPDATE, aby zmienić wiersze przechowywane w bazie danych. Na przykład, wyobraźmy sobie, że jeden z naszych pracowników zmienia stanowisko:

```
update pracownik
set stanowisko='administrator baz danych'
where IDpracownika='6651';
```

To wyrażenie zmienia wartość kolumny stanowisko dla pracownika o identyfikatorze 6651.

Ogólny wzorzec instrukcji UPDATE według podręcznika MySQL jest następujący:

```
UPDATE [LOW_PRIORITY] [IGNORE] nazwa_tab
SET nazwa_kol1=wyrazenie1 [, nazwa_kol2=wyrazenie2 …]
[WHERE definicja_where]
[ORDER BY …]
[LIMIT wiersze]
```

```
lub

UPDATE [LOW_PRIORITY] [IGNORE] nazwa_tab [, nazwa_tab …]
SET nazwa_kol1=wyrazenie1 [, nazwa_kol2=wyrazenie2 …]
[WHERE definicja_where]
```

Instrukcja `UPDATE` w wielu względach jest podobna do instrukcji `DELETE`.

Możemy podać opcjonalną klauzulę `WHERE`, aby aktualizować określone wiersze, lub nie podawać jej, aby uaktualnić wszystkie wiersze. Tutaj również zapomnienie dodania klauzuli `WHERE` może mieć nieprzyjemne skutki — pamiętam jeden projekt, w którym mój nierozsądny kolega wpisał między wierszami:

```
update user
set password='test';
```

Dlatego podkreślamy użyteczność opcji `--i-am-a-dummy`, szczególnie gdy musimy pracować z nieuważnymi osobami.

Druga z podanych wersji instrukcji `UPDATE` służy do aktualizacji wielu tabel. Działa podobnie do omawianej wcześniej instrukcji `DELETE`, usuwającej wiersze z wielu tabel. Zauważmy, że tylko kolumny wymienione w klauzuli `SET` będą aktualizowane.

Wszystkie pozostałe klauzule instrukcji `UPDATE` analizowaliśmy wcześniej. Opcje `LOW_PRIORITY` oraz `IGNORE` działają w taki sam sposób, jak w instrukcji `INSERT`. Klauzule `ORDER BY` oraz `LIMIT` działają tak, jak w instrukcji `DELETE`.

Wczytywanie danych za pomocą instrukcji LOAD DATA INFILE

Polecenie `LOAD DATA INFILE` pozwala wprowadzić dane z pliku tekstowego do pojedynczej tabeli bez konieczności pisania instrukcji `INSERT`. W listingu 5.2 pokazano zawartość pliku, w którym znajdują się informacje na temat wydziałów.

Listing 5.2. *wydzial_infile.txt*

```
42      Finanse
128     Badania i rozwoj
NULL    Kadry
NULL    Marketing
```

Plik ma domyślny format dla instrukcji `LOAD DATA INFILE` — każdy rekord znajduje się w oddzielnym wierszu, a wartości kolumn są oddzielone od siebie tabulatorem (format wczytywania danych można konfigurować; za chwilę zobaczymy, jak się to robi).

Możemy pobrać dane do tabeli wydzial za pomocą następującej instrukcji `LOAD DATA INFILE`:

```
load data local infile 'wydzial_infile.txt'
into table wydzial;
```

Ta instrukcja jest szczególnie przydatna przy konwertowaniu danych z innego formatu bazy danych, arkusza kalkulacyjnego lub pliku CSV (ang. *comma-separated values*).

Instrukcja `LOAD DATA INFILE` wymaga uprawnienia FILE; więcej informacji na temat uprawnień znajduje się w rozdziale 11., „Zarządzanie uprawnieniami użytkowników” — w razie pojawienia się problemów podczas wykonania tego polecenia należy zajrzeć do tego rozdziału. Z istotnych względów bezpieczeństwa uprawnienie to jest często ograniczane, aby nie pozwolić użytkownikom na wczytywanie na przykład pliku */etc/passwd*.

Ogólny wzorzec instrukcji `LOAD DATA INFILE` jest następujący:

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'nazwaPliku.txt'
    [REPLACE | IGNORE]
    INTO TABLE nazwa_tab
    [FIELDS
        [TERMINATED BY '\t']
        [[OPTIONALLY] ENCLOSED BY '']
        ESCAPED BY '\\']
    ]
    [LINES TERMINATED BY '\n']
    [IGNORE liczba LINES]
    [(nazwa_kol,…)]
```

Klauzule opcjonalne to:

- Opcja `LOW PRIORITY` działa w ten sam sposób, jak w instrukcji `INSERT`, czyli jej działanie jest wstrzymywane do momentu, w którym inni klienci nie zakończą odczytywania tabeli. Z kolei opcja `CONCURRENT` pozwala innym klientom odczytywać tabelę w czasie wstawiania danych.
- W przykładzie podaliśmy opcjonalne słowo kluczowe `LOCAL`, oznaczające, że plik z danymi jest na komputerze klienta. Jeżeli nie zostanie ono dodane, plik będzie szukany przez MySQL na serwerze.
- Jeżeli klucze kolidują w czasie wstawiania danych, opcje `REPLACE` i `IGNORE` dostarczają dwóch metod do ich obsługi. Wstawienie `REPLACE` oznacza, że istniejący wiersz zostanie zamieniony na nowy, `IGNORE` — że istniejący wiersz pozostanie w tabeli.
- Klauzule `FIELDS` i `LINES` precyzują, w jaki sposób ułożone są dane w pliku wczytywanym. Domyślnie dla instrukcji `LOAD DATA INFILE` jest przyjęte, że każdy rekord znajduje się w nowym wierszu, natomiast kolumny są rozdzielone tabulatorami. Możemy także ująć wartości kolumn w apostrofy oraz zastosować znak lewego ukośnika, aby uniknąć ewentualnych problemów, które mogłyby zostać wywołane przez znaki specjalne (takie jak apostrofy).
- Klauzula `IGNORE liczba LINES` stanowi informację, że należy ignorować określoną liczbę (*liczba*) pierwszych wierszy w pliku wczytywanym.
- Ostatnia klauzula pozwala wczytać dane tylko do niektórych kolumn tabeli.

Popularny format pobierania danych to CSV lub inaczej format pliku z wartościami oddzielonymi przecinkami. Wiele programów potrafi zapisywać i odczytywać pliki tego typu, jednym z przykładów jest Microsoft Excel[1]. W listingu 5.3 pokazano plik CSV zapisany w arkuszu Excel.

Listing 5.3. *nowi_programisci.csv*

```
Nazwisko;Stanowisko;IDwydzialu

Julia Lenin;programista;128
Dariusz Nowak;programista;128
Tim O'Leary;programista;128
```

Możemy wczytać te dane do tabeli `pracownik` za pomocą następującego zapytania:

```
load data infile 'e:\\nowi_programisci.csv'
into table pracownik
fields terminated by ';'
lines terminated by '\n'
ignore 2 lines
(nazwisko, stanowisko, IDwydzialu);
```

Jak widać, użyliśmy więcej opcji niż w przykładzie z danymi w formacie domyślnym. Warto omówić kilka z tych elementów:

- Ponieważ zastosowaliśmy ścieżkę używaną w systemie Windows (DOS), która zawiera lewy ukośnik, powinniśmy sprawić, aby nie został on źle zinterpretowany. Dlatego teraz nasza ścieżka ma postać: `'e:\\nowi_programisci.csv'`.
- Prawdopodobnie zapytanie zadziała bez podania znaku oddzielającego kolumny w pliku CSV, jednak lepiej jest go podać.
- W tym przykładzie nie musimy podawać znaku oddzielającego rekordy, mimo to wpisaliśmy go (znak nowego wiersza).
- Plik ma nagłówek; pierwsze dwa wiersze nie zawierają danych, dlatego powinny być zignorowane.
- Dane w pliku nie zawierają `IDpracownika`, dlatego aby przypisać trzy kolumny do czterech kolumn w bazie danych, musieliśmy sprecyzować, do których kolumn (w kolejności) dane zostaną przyporządkowane. W tym przykładzie ustaliliśmy kolumny, wpisując `(nazwisko, stanowisko, IDwydzialu)`.

Podsumowanie

W tym rozdziale poznaliśmy sposoby wstawiania, usuwania i aktualizacji danych tabeli bazy danych.

[1] W polskiej wersji językowej Excel zapisuje plik CSV jako plik wartości rozdzielonych średnikami (chociaż nazywa go plikiem wartości rozdzielonych przecinkami) — *przyp. red.*

Wstawianie danych

- Wartości łańcuchów znakowych powinny być umieszczane w apostrofach. Przed apostrofami i lewymi ukośnikami należy wstawiać lewy ukośnik.
- Dodawanie danych do tabel za pomocą instrukcji `INSERT`:

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] nazwa_tab [(nazwa_kol,…)]
    VALUES ((wyrazenie | DEFAULT),…),(…),…
    [ON DUPLICATE KEY UPDATE nazwa_kol=wyrazenie, …]

lub  INSERT [[LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] nazwa_tab [(nazwa_kol,…)]
    SELECT …

lub  INSERT [[LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] nazwa_tab
    SET nazwa_kol=(wyrazenie | DEFAULT), …
    [ON DUPLICATE KEY UPDATE nazwa_kol=wyrazenie, …]
```

- Instrukcja `REPLACE` działa w taki sam sposób, jak instrukcja `INSERT`, tylko powoduje nadpisanie istniejących wierszy, gdy pojawi się kolizja kluczy.

Usuwanie danych

- Unikanie błędów za pomocą opcji `--i-am-a-dummy`.
- Usuwanie danych z tabeli instrukcją `DELETE`:

```
DELETE [LOW_PRIORITY] [QUICK] FROM nazwa_tabeli
       [WHERE definicja_where]
       [ORDER by …]
       [LIMIT wiersze]

lub

DELETE [LOW_PRIORITY] [QUICK] nazwa_tabeli[.*] [, nazwa_tabeli[.*] …]
       FROM odwolanie-do-tabeli
       [WHERE definicja_where]

lub

DELETE [LOW_PRIORITY] [QUICK]
       FROM nazwa_tabeli[.*] [, nazwa_tabeli[.*] …]
       USING odwolanie-do-tabeli
       [WHERE definicja_where]
```

- Instrukcja `TRUNCATE TABLE` powoduje usunięcie wszystkich wierszy z tabeli.

Aktualizacja danych

Aktualizacja danych w tabeli instrukcją `UPDATE TABLE`:

```
UPDATE [LOW_PRIORITY] [IGNORE] nazwa_tab
SET nazwa_kol1=wyrazenie1 [, nazwa_kol2=wyrazenie2 …]
[WHERE definicja_where]
[ORDER BY …]
[LIMIT wiersze]

lub

UPDATE [LOW_PRIORITY] [IGNORE] nazwa_tab [, nazwa_tab …]
SET nazwa_kol1=wyrazenie1 [, nazwa_kol2=wyrazenie2 …]
[WHERE definicja_where]
```

LOAD DATA INFILE

Używamy instrukcji `LOAD DATA INFILE`, aby wczytać zawartość pliku tekstowego do tabeli:

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'nazwaPliku.txt'
    [REPLACE | IGNORE]
    INTO TABLE nazwa_tab
    [FIELDS
        [TERMINATED BY '\t']
        [[OPTIONALLY] ENCLOSED BY '']
        ESCAPED BY '\\']
    ]
    [LINES TERMINATED BY '\n']
    [IGNORE liczba LINES]
    [(nazwa_kol,…)]
```

Quiz

1. Która z poniższych instrukcji spowoduje prawidłowe wstawienie wiersza do tabeli pracownik?

   a)

   ```
   insert into pracownik values
   set IDpracownika=NULL, nazwisko='Laura Tomaszewska',
   stanowisko='programista', IDwydzialu=128;
   ```

   b)

   ```
   insert pracownik values
   (NULL, 'Laura Tomaszewska', 'programista', 128);
   ```

   c)

   ```
   insert into pracownik values
   (NULL, Laura Tomaszewska, programista, 128);
   ```

   d)

   ```
   insert pracownik values
   (NULL, 'Laura O'Leary', 'programista', 128);
   ```

2. Instrukcja `REPLACE`:
   a) działa tak samo, jak `INSERT`, lecz gdy pojawi się kolizja klucza, istniejący wiersz zostanie zastąpiony nowym wierszem;
   b) działa tak samo, jak `INSERT`, lecz gdy pojawi się kolizja klucza, pozostawiony zostanie istniejący wiersz, a zignorowany nowy;
   c) działa tak samo, jak `UPDATE`, lecz gdy pojawi się kolizja klucza, istniejący wiersz zostanie zastąpiony nowym;
   d) działa tak samo, jak `UPDATE`, lecz gdy pojawi się kolizja klucza, pozostawiony zostanie istniejący wiersz, a zignorowany nowy.
3. Opcja uruchomieniowa klienta *mysql* `--i-am-a-dummy`:
   a) zapobiega wstawianiu wszelkich danych;
   b) zapobiega aktualizacji danych, jeżeli nie jest sprecyzowane ograniczenie dotyczące klucza;
   c) zapobiega usunięciu danych, jeżeli nie jest sprecyzowane ograniczenie dotyczące klucza;
   d) oba b) i c).
4. Domyślnie pola w plikach danych, wczytywanych instrukcją `load data infile`, są oddzielone:
   a) przecinkami,
   b) spacjami,
   c) tabulatorami,
   d) znakami „|”.
5. Opcja `LOCAL` w instrukcji `LOAD DATA INFILE` mówi, że:
   a) klient i serwer działają na tym samym komputerze,
   b) plik danych jest na serwerze,
   c) plik danych jest na kliencie,
   d) serwer działa na lokalnym hoście.

Ćwiczenia

1. Utwórz zestaw instrukcji `INSERT`, aby wstawić dane do każdej tabeli bazy danych zamowienia.
2. Usuń dane z tabel.
3. Zapisz plik zawierający te same dane, które były wstawiane w punkcie pierwszym, a następnie wczytaj je do bazy danych, używając instrukcji `LOAD DATA INFILE`.

Odpowiedzi

Quiz

1. b
2. a
3. d
4. c
5. c

Ćwiczenia

Nie ma jednej *prawidłowej* odpowiedzi w ćwiczeniach z tego rozdziału. Po prostu upewnij się, że potrafisz zrobić wszystkie ćwiczenia.

Następnie

W rozdziale 6., „Zapytania w MySQL”, zaczniemy analizować podstawowe narzędzie SQL — instrukcję `SELECT` w jej wielu odmianach.