Do tej pory jedynie odczytywaliśmy zapisane w bazach informacje. W tym odcinku nauczysz się wstawiać, usuwać i modyfikować dane. Dowiesz się również, czym jest transakcja, blokada i na czym polega transakcyjne przetwarzanie danych.

Transakcje

W poprzednich odcinkach instrukcje języka SQL traktowane były jak indywidualne operacje (transakcje) — zapytania, które zwracały pewne dane. Teraz pora spojrzeć na nie inaczej — jako na części pewnych operacji (transakcji). Inaczej rzecz ujmując, transakcje są grupami instrukcji języka SQL, traktowanymi z punktu widzenia ich przetwarzania jako jedna całość.

Transakcje gwarantują spójność modyfikowanych informacji. Typowym przykładem transakcyjnego przetwarzania danych jest przeniesienie pieniędzy z jednego konta na drugie. Taka operacja przebiega w dwóch etapach:

1. zmniejszenie o pewną sumę stanu konta X,
2. dodanie tej sumy do stanu konta Y.

Gdyby po wykonaniu pierwszej operacji wystąpił błąd uniemożliwiający wykonanie drugiej, z systemu zniknęłaby pewna suma pieniędzy. Równie nieprzyjemnym zaskoczeniem dla właściciela byłoby sprawdzenie przez niego stanu obu jego kont już po odjęciu danej sumy z pierwszego konta, ale przed jej dodaniem do drugiego konta.

Baza danych jest przydatna tak długo, jak długo znajdują się w niej wyłącznie prawdziwe informacje. Im częściej zmieniają się opisywane obiekty, tym częściej musimy aktualizować zawartość bazy danych. Sytuację komplikuje fakt, że bazy danych w większości są aplikacjami sieciowymi i dostęp do danych oraz możliwość ich modyfikacji muszą być zapewnione jednocześnie wielu użytkownikom.

Podstawowym sposobem zachowania integralności (spójności) danych jest przetwarzanie ich w operacjach noszących nazwę transakcji. Transakcja to seria zmian wprowadzonych do bazy danych i traktowanych przez serwer baz danych jako pojedyncza zmiana. Transakcja może składać się z pojedynczej instrukcji, jeżeli wynika to ze specyfiki realizowanego zadania, lub z wielu instrukcji. Istotną cechą transakcji jest to, że zmiany wprowadzane przez nie do bazy danych są trwale zapisywane tylko wtedy, gdy wykonane zostaną wszystkie wchodzące w skład transakcji instrukcje.

Działanie takie ma szczególne znaczenie, ponieważ pozwala zachować integralność bazy danych w sytuacji, gdy w trakcie wykonywania transakcji nastąpi awaria systemu, zerwanie połączenia itp. Przypuśćmy, że modyfikujemy dane o pracownikach naszej firmy. Postanowiliśmy podnieść pensje wszystkich osób o 10%. W tym celu odczytujemy aktualną wysokość pensji, dodajemy do niej 10% i tak zmienioną wartość zapisujemy z powrotem do bazy. Załóżmy dodatkowo, że w bazie przechowywana jest wyłącznie informacja o aktualnej wysokości pensji każdego z pracowników. W trakcie wykonywania instrukcji serwer bazodanowy został nagle wyłączony. Po jego ponownym uruchomieniu sprawdzamy wysokość pensji współpracowników i odkrywamy, że część danych się zmieniła. W przypadku tabeli zawierającej kilkaset wierszy możemy mieć poważne problemy z określeniem, którym pracownikom podwyższyliśmy już pensje, a którym dopiero zamierzamy je zwiększyć. Aby zapobiec takim problemom, powinniśmy zdefiniować instrukcję zmieniającą wypłatę wszystkim osobom jako jedną transakcję — wtedy MySQL albo zmodyfikowałby wszystkie rekordy, albo w przypadku wystąpienia niespodziewanej awarii — zostawiłby wszystkie rekordy niezmienione.

Nazwę „transakcja” zapożyczono z bankowości, gdzie oznaczała pojedynczą operację przelania pieniędzy z konta na konto. Taka operacja przebiega co najmniej w dwóch etapach: najpierw zmniejsza się stan konta A, następnie zwiększa stan konta B. Oczywiście z założenia niedopuszczalna jest sytuacja, w której zmodyfikowano by stan tylko jednego konta (gdyż np. po zmniejszeniu stanu konta A wystąpiła jakaś awaria).

Kolejną zaletą metody transakcyjnej jest zachowanie spójności danych modyfikowanych jednocześnie przez kilku użytkowników. W takich wypadkach serwery MySQL utrzymują spójność danych poprzez nakładanie blokad na modyfikowane przez użytkowników obiekty. To, jaki obiekt i w jaki sposób zostanie zablokowany, zależy od ustawień serwera i aplikacji klienckiej. Przypuśćmy, że uruchomiliśmy skrypt obniżający cenę pewnych towarów o 5%. W tym samym czasie kolega na swojej stacji roboczej również wykonał tę instrukcję. W takiej sytuacji MySQL wykona obie instrukcje, w wyniku czego cena wybranych towarów będzie obniżona o 10%. Błędu tego uniknęlibyśmy, zakładając blokadę na ceny odczytywane przez nasz skrypt i umożliwiając modyfikację tych wierszy innym użytkownikom dopiero po zatwierdzeniu wprowadzanych przez nas zmian.

Transakcją nazywamy operację zmiany stanu bazy danych, najczęściej składającą się z wielu operacji aktualizacji wierszy w tabelach. W przypadku przerwania operacji już w trakcie jej trwania baza „wróci" do stanu sprzed jej rozpoczęcia. Jedynie pomyślnie zakończone i zatwierdzone zmiany będą zapisane w bazie.

Zgodnie z zasadą ACID (ang. Atomicity, Consistency, Isolation, and Durability) transakcje muszą być:

1. Niepodzielne (ang. Atomicity). Niepodzielność oznacza, że zatwierdzane są wszystkie wchodzące w skład transakcji instrukcje albo nie jest zatwierdzana żadna z nich. Innymi słowy, wszystkie wchodzące w skład transakcji instrukcje muszą być wykonane poprawnie — jeżeli choć jedna z nich zgłosi błąd, wszystkie przeprowadzone w ramach transakcji zmiany zostaną wycofane.
2. Spójne (ang. Consistency). Ta cecha transakcji gwarantuje, że ich wykonanie nie doprowadzi, nawet w przypadku awarii serwera, do utraty spójności danych. Ponieważ wszystkie zmiany danych wykonywane są w ramach transakcji, przechowywane w bazach informacje zawsze będą spójne.
3. Izolowane (ang. Isolation). Izolowanie transakcji wymaga albo zablokowania danych modyfikowanych w ramach jednej z nich, albo utworzenia dodatkowej wersji danych. W zależności od obowiązującego w ramach serwera lub sesji klienckiej poziomu izolowania transakcji może dojść do następujących sytuacji:

1. Utrata aktualizacji (ang. Lost Update) ma miejsce, gdy dwa procesy modyfikują jednocześnie te same dane. Przykładowo jeden użytkownik zmienia cenę towaru na 100 zł, a drugi — na 200. W takim przypadku jedna ze zmian zostanie utracona (zastąpiona drugą modyfikacją). Domyślnie skonfigurowane serwery bazodanowe nie dopuszczają do utraty aktualizacji.
2. Brudne odczyty (ang. Dirty Read) — do takiej sytuacji dochodzi, gdy możliwe jest odczytanie zmian niezatwierdzonych jeszcze przez inny proces. Jeżeli proces odczytujący nie zażąda założenia blokady na odczytywanych danych, uzyska do nich dostęp nawet wtedy, gdy właśnie będą modyfikowane. Gdyby proces modyfikujący wycofał wprowadzone zmiany, odczytane dane okazałyby się niespójne. Domyślnie skonfigurowane serwery bazodanowe nie dopuszczają brudnych odczytów.
3. Niepowtarzalne odczyty (ang. Non-Repeatable Reads) mają miejsce, gdy powtórzenie w ramach transakcji tego samego odczytu daje inny wynik. Różnice w wynikach są spowodowane tym, że natychmiast po zakończeniu odczytu (a nie po zakończeniu całej transakcji) proces odczytujący zdejmuje blokady założone na odczytywane dane. Niezablokowane dane mogą być zmienione przez inny proces, a więc ich powtórne odczytanie da inny (niespójny) wynik. Domyślnie skonfigurowane serwery bazodanowe dopuszczają niepowtarzalne odczyty.
4. Odczyty widma (ang. Phantom Reads) — sytuacja taka ma miejsce, jeżeli pomiędzy dwoma wykonanymi w ramach transakcji odczytami zmieni się liczba odczytywanych wierszy. Jeżeli np. podczas pierwszego odczytu w tabeli Produkty znajdowało się 100 produktów o cenach niższych niż 10 zł, instrukcja SELECT * FROM Produkty WHERE Cena <10 zwróciłaby 100 wierszy. W trakcie trwania transakcji możliwa jest jednak zmiana pozostałych wierszy tabeli, w tym zmniejszenie ceny jakiegoś produktu poniżej 10 zł. Możliwe jest również wstawienie do tej tabeli nowego produktu o cenie np. 7 zł. Z tego powodu drugie wykonanie tego samego zapytania zwróciłoby już 102 wiersze. Domyślnie skonfigurowane serwery bazodanowe dopuszczają odczyty widma.

5. Trwałe (ang. Durability). Trwałość transakcji gwarantuje, że efekty zatwierdzonych transakcji będą zapisane w bazie, nawet w przypadku awarii serwera SQL 2005. Do przywrócenia spójności danych serwery bazodanowe z reguły używają jakiejś formy dziennika transakcyjnego.

Przetwarzanie transakcyjne

Przetwarzanie transakcji przebiega w dwóch fazach. Pierwsza z nich to inicjalizacja — rozpoczęcie transakcji. Po

zainicjalizowaniu transakcji wszystkie kolejne wyrażenia SQL traktowane są jako część transakcji, aż do momentu jej zakończenia. Transakcję rozpoczyna wykonanie pierwszej instrukcji języka SQL.

Drugą fazą jest zatwierdzenie lub wycofanie transakcji. Jest ono równoznaczne z jej zakończeniem i sprawia, że wszelkie wprowadzone przez nią modyfikacje są na stałe zapisywane lub anulowane w bazie danych. Do momentu, kiedy transakcja nie zostanie zatwierdzona, wprowadzone przez nią zmiany nie są widoczne dla innych użytkowników korzystających z tej samej bazy danych. Transakcja może być zakończona poprzez:

1. Jawne jej zatwierdzenie instrukcją COMMIT; zmiany są nieodwracalnie wprowadzone do bazy.
2. Jawne jej wycofanie instrukcją ROOLBACK; zmiany są odrzucone, a baza wraca do stanu sprzed rozpoczęcia transakcji.
3. Zakończenie sesji, w ramach której rozpoczęliśmy transakcję; zmiany powinny być wycofane.
4. Przerwanie sesji (np. w wyniku awarii zasilania lub sieci); zmiany są odrzucone, a baza wraca do stanu sprzed rozpoczęcia transakcji.

Automatyczne zatwierdzanie transakcji

MySQL posiada domyślnie włączony specjalny tryb pracy (ang. Autocommit mode), w którym każde wykonanie zapytania powoduje automatyczne zatwierdzenie transakcji. Można powiedzieć, że przed wykonaniem każdej instrukcji serwer niejawnie rozpoczyna, a natychmiast po jej zakończeniu zatwierdza lub wycofuje transakcję (jeśli nie udało się wykonać tej instrukcji). Aby wyłączyć ten mechanizm na poziomie sesji klienckiej, należy wykonać instrukcję SET AUTOCOMMIT=0;.

Rozpoczynanie transakcji

Aby jawnie rozpocząć nową transakcję, należy wykonać instrukcję START TRANSACTION lub BEGIN [WORK]. W tym momencie MySQL rozpocznie transakcję, w której skład będą wchodziły wszystkie wpisane przez nas instrukcje SQL.

Zatwierdzanie transakcji

Aby jawnie zatwierdzić transakcję, należy wykonać instrukcję COMMIT [WORK]. Jej wykonanie spowoduje:

1. Zakończenie aktywnej transakcji.
2. Zatwierdzenie przeprowadzonych w ramach transakcji zmian w bazie danych.
3. Usunięcie wszystkich założonych blokad.
4. Udostępnienie uaktualnionych danych innym użytkownikom bazy.

Wycofywanie transakcji

Aby jawnie wycofać rozpoczętą transakcję (transakcja może zostać niejawnie wycofana z powodu szeroko rozumianej awarii systemu lub błędu wykonania jednej z instrukcji wchodzących w skład transakcji), należy wykonać instrukcję ROLLBACK [WORK]. W rezultacie MySQL:

1. Zakończy transakcję.
2. Wycofa wszystkie zmiany dokonane przez użytkownika od rozpoczęcia transakcji.
3. Usunie wszystkie założone blokady.
4. Udostępni oryginalne dane innym użytkownikom bazy.

Transakcyjne przetwarzanie danych przedstawiają listingi 8.1 – 8.4.

Przed wykonaniem ćwiczeń wykonaj kopię bazy test. Niektóre z poniższych ćwiczeń wymagają jednoczesnego nawiązania kilku sesji z serwerem MySQL. Najlepiej zrób to, wykonując jedną część listingów w programie MySQL Query Browser, a drugą — w programie mysql. Dokładny opis wykorzystywanej instrukcji UPDATE znajdziesz w dalszej części odcinka — na razie wystarczy wiedzieć, że modyfikuje ona zapisane w tabeli dane.

Listing 8.1. Sesja 1. — przykład transakcyjnego przetwarzania danych. W jednej sesji jawnie rozpoczynamy transakcję, zmieniamy cenę zakupu wskazanego towaru i odczytujemy w ramach tej samej sesji dane tego

towaru

```
BEGIN WORK;

Query OK, 0 rows affected (0.05 sec)


UPDATE item

SET cost_price = 750

WHERE item_id = 12;

Query OK, 1 row affected (7.61 sec)

Rows matched: 1  Changed: 1  Warnings: 0


SELECT *

FROM item

WHERE item_id = 12;

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

| item_id | description     | cost_price |  sell_price |

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

| 12      | SQL Server 2005 | 550.00     |  NULL       |

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

Listing 8.2. Sesja 2. — próba odczytania danych o tym samym towarze w ramach innej sesji. Zwróć uwagę, że zwrócone zostały dane sprzed rozpoczęcia transakcji

```
SELECT *

FROM item

WHERE item_id = 12;

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

| item_id | description     | cost_price |  sell_price |

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

| 12      | SQL Server 2005 | NULL       |  NULL       |

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

Listing 8.3. Sesja 1. — wycofujemy transakcję i raz jeszcze odczytujemy dane towaru

```
ROLLBACK WORK;

Query OK, 0 rows affected (0.11 sec)


SELECT *

FROM item

WHERE item_id = 12;

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

| item_id | description     | cost_price |  sell_price |

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

| 12      | SQL Server 2005 | NULL       |  NULL       |

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

Listing 8.4. Sesja 2. — sprawdzamy, czy dane towaru pozostały w czasie transakcyjnego przetwarzania spójne dla innych użytkowników bazy

```
SELECT *

FROM item

WHERE item_id = 12;

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

| item_id | description     | cost_price |  sell_price |

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

| 12      | SQL Server 2005 | NULL       |  NULL       |

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

Punkty zachowania

Transakcje składające się z dużej liczby poleceń lub modyfikujące dużą liczbę wierszy warto podzielić na kilka mniejszych części. W ramach transakcji MySQL pozwala na definiowanie punktów kontrolnych lub punktów zachowania. Aby określić punkt kontrolny, należy wykonać instrukcję: SAVEPOINT Nazwa punktu.

Wprowadzenie punktu zachowania umożliwia częściowe wycofanie rozpoczętej transakcji. Dzięki temu zmiany wprowadzone przed punktem kontrolnym nie zostają utracone.

O ile zdefiniowaliśmy punkt kontrolny, możemy wycofać część zmian wprowadzonych w ramach transakcji. W tym celu należy wykonać instrukcję: ROLLBACK [WORK] TO SAVEPOINT Nazwa punktu.

Niepotrzebne dłużej punkty kontrolne należy usunąć, wykonując instrukcję RELEASE SAVEPOINT Nazwa

punktu. **Poziomy izolowania transakcji**

Żeby transakcyjnie przetwarzać dane, MySQL musi blokować na ten czas dane w bazie. Generalnie serwery baz danych używają dwóch typów blokad.

1. Założenie blokady współdzielonej uniemożliwia zablokowanie tych samych danych w trybie wyłącznym, ale nie uniemożliwia zakładania innych blokad współdzielonych. Domyślnie taka blokada zakładana jest podczas wykonywania instrukcji SELECT, czyli wielu użytkowników może jednocześnie odczytywać te same dane.

2. Założenie blokady wyłącznej uniemożliwia założenie jakiejkolwiek innej blokady na tych samych danych. Domyślnie taka blokada zakładana jest podczas modyfikowania danych, czyli tylko jeden użytkownik może w tym samym czasie zmieniać te same dane.

MySQL, tak jak większość serwerów baz danych, umożliwia pracę w jednym z czterech poziomów izolowania transakcji (poziom izolowania transakcji można również ustalić na poziomie serwera, modyfikując parametry uruchomieniowe MySQL-a):
1. Tryb odczytu zatwierdzonego (ang. Read Committed) jest domyślnym poziomem izolowania transakcji. Na tym poziomie odczyt danych wymaga założenia na nich blokady współdzielonej. Ponieważ zakładana na czas zmiany blokada wyłączna jest niekompatybilna z innymi blokadami, w tym z blokadą współdzieloną, eliminuje to brudne odczyty. Jednak na tym poziomie nadal występują niepowtarzalne odczyty i odczyty widma. W efekcie pozostali użytkownicy nie mają dostępu do zmienianych danych i dopiero po zatwierdzeniu transakcji zobaczą wprowadzone zmiany.
2. W trybie niezatwierdzonego odczytu (ang. Read Uncommitted) odczyt danych nie powoduje założenia blokady współdzielonej. Na tym poziomie występują brudne odczyty, niepowtarzalne odczyty i odczyty widma (jedynym niekorzystnym zjawiskiem niewystępującym na tym poziomie jest utrata aktualizacji) (listingi 8.5 – 8.8).

Listing 8.5. Sesja 1. — raz jeszcze rozpoczynamy transakcję, w ramach której zmieniamy cenę zakupu towaru

```
BEGIN WORK;

Query OK, 0 rows affected (0.06 sec)


SELECT *

FROM item;

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

| item_id | description     | cost_price |  sell_price |

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

| 1       | Wood Puzzle     | 15.23      |  21.95      |

| 2       | Rubik Cube      | 7.45       |  11.49      |

| 3       | Linux CD        | 1.99       |  2.49       |

| 4       | Tissues         | 2.11       |  3.99       |

| 5       | Picture Frame   | 7.54       |  9.95       |
```

```
| 6       | Fan Small       | 9.23       |  15.75      |

| 7       | Fan Large       | 13.36      |  19.95      |

| 8       | Toothbrush      | 0.75       |  1.45       |

| 9       | Roman Coin      | 2.34       |  2.45       |

| 10      | Carrier Bag     | 0.01       |  0.00       |

| 11      | Speakers        | 19.73      |  25.32      |

| 12      | SQL Server 2005 | NULL       |  NULL       |

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

12 rows in set (0.00 sec)


UPDATE item

SET cost_price = 750

WHERE item_id = 12;

Query OK, 1 row affected (0.00 sec)

Rows matched: 1  Changed: 1  Warnings: 0


SELECT *

FROM item

WHERE item_id = 12;

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

| item_id | description     | cost_price |  sell_price |

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

| 12      | SQL Server 2005 | 750.00     |  NULL       |

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

Listing 8.6. Sesja 2. — zmieniamy domyślny poziom izolowania transakcji dla sesji i próbujemy odczytać właśnie modyfikowane dane

```
SET SESSION TRANSACTION ISOLATION LEVEL READ  UNCOMMITTED;

Query OK, 0 rows affected (0.07 sec)
```

```
SELECT *
FROM item
WHERE item_id = 12;
+---------+----------------+------------+------------+
| item_id | description    | cost_price |  sell_price |
+---------+----------------+------------+------------+
| 12      | SQL Server 2005 | 750.00     |  NULL       |
+---------+----------------+------------+------------+
```

Listing 8.7. Sesja 1. — gdybyśmy teraz wycofali transakcję, dane wróciłyby do oryginalnego stanu

```
ROLLBACK WORK;
Query OK, 0 rows affected (0.11 sec)

SELECT *
FROM item
WHERE item_id = 12;
+---------+----------------+------------+------------+
| item_id | description    | cost_price |  sell_price |
+---------+----------------+------------+------------+
| 12      | SQL Server 2005 | NULL       |  NULL       |
+---------+----------------+------------+------------+
```

Listing 8.8. Sesja 2. — gdyby teraz ten sam użytkownik powtórzył zapytanie, uzyskałby inne wyniki

```
SELECT *
FROM item
WHERE item_id = 12;
+---------+----------------+------------+------------+
| item_id | description    | cost_price |  sell_price |
+---------+----------------+------------+------------+
```

| 12 | SQL Server 2005 | NULL | NULL |

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

3. W trybie powtarzalnego odczytu (ang. Repeatable Read) blokady współdzielone utrzymywane są do czasu zakończenia całej transakcji. Dzięki temu inny proces nie może zmodyfikować odczytywanych w jej ramach danych, co eliminuje niepowtarzalne odczyty. Na tym poziomie występują tylko odczyty widma. Tryb REPEATABLE READ gwarantuje, że wielokrotne odczytywanie tych samych danych w ramach tej samej transakcji zwróci te same wyniki.

4. W trybie szeregowania (ang. Serializable) transakcje odwołujące się do tych samych tabel wykonywane są jedna po drugiej. Blokowanie całych obiektów, a nie tylko odczytywanych danych, na czas trwania transakcji pozwala wyeliminować odczyty widma, ale powoduje, że odczytując nawet jeden wiersz tabeli, możemy uniemożliwić pozostałym użytkownikom zmodyfikowanie przechowywanych w niej danych. Tryb SERIALIZABLE jest najbardziej restrykcyjny — zakładana jest w nim blokada wyłączna na całą tabelę, której zawartość użytkownik modyfikuje. W efekcie pozostali użytkownicy muszą czekać, aż transakcja się zakończy, zanim uzyskają dostęp do danych, co daje efekt szeregowego wykonywania instrukcji użytkowników, jednej po drugiej.

Blokowanie tabel

W przypadku typów tabel nieobsługujących transakcji (dotyczy to również tabel typu MyISAM) jedynym sposobem zagwarantowania spójności modyfikowanych danych jest ich blokowanie. W takim przypadku przed rozpoczęciem modyfikacji danych należy jawnie zablokować docelowe tabele instrukcją LOCK TABLES nazwa tabeli1 typ blokady, nazwa tabeli2 typ blokady.

Tabele mogą zostać zablokowane do odczytu (typ blokady READ) — wtedy zakładana jest na nie blokada współdzielona — lub do zapisu (typ blokady WRITE) — wtedy zakładana jest na nie blokada wyłączna. Blokada współdzielona jest kompatybilna z innymi blokadami współdzielonymi, czyli po jej założeniu inni użytkownicy będą mogli odczytywać dane z zablokowanej w ten sposób tabeli (listingi 8.9 i 8.10).

Listing 8.9. Sesja 1. — dwie tabele zostały zablokowane do odczytu

```
LOCK TABLES orderinfo READ, orderline READ;

Query OK, 0 rows affected (0.00 sec)
```

Listing 8.10. Sesja 2. — zawartość tabeli zablokowanej do odczytu może być odczytywana przez wszystkich użytkowników bazy danych

```
SELECT *

FROM orderinfo;

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

| orderinfo_id | customer_id | date_placed | date_shipped | shipping |

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

|            1 |           3 | 2000-03-13  | 2000-03-17   |     2.99 |

|            2 |           8 | 2000-06-23  | 2000-06-23   |     0.00 |

|            3 |          15 | 2000-09-02  | 2000-09-12   |     3.99 |

|            4 |          13 | 2000-09-03  | 2000-09-10   |     2.99 |
```

| 5 | 8 | 2000-07-21 | 2000-07-24 | 0.00 |

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

Natomiast próba zmodyfikowania struktury lub zawartości tabeli zablokowanej do odczytu będzie wstrzymana do momentu zdjęcia blokady (listingi 8.11 i 8.12).

Listing 8.11. Sesja 2. — próba usunięcia wiersza z zablokowanej tabeli — wykonanie instrukcji zostało wstrzymane

DELETE FROM orderinfo

WHERE orderinfo_id=5;

Listing 8.12. Sesja 1. — zdjęcie blokady współdzielonej spowodowało, że instrukcja DELETE została w końcu wykonana i zgłosiła błąd

UNLOCK TABLES;

Query OK, 0 rows affected (0.07 sec)

/*Sesja 2. */

ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constrai

nt fails (`test`.`orderline`, CONSTRAINT `orderline_orderlinfo_id_fk` FOREIGN KE

Y (`orderinfo_id`) REFERENCES `orderinfo` (`orderinfo_id`)) **Wstawianie danych**

W porównaniu z bardzo rozbudowaną instrukcją SELECT składnia instrukcji INSERT (wstawiającej nowe wiersze), DELETE (usuwającej wiersze) i UPDATE (modyfikującej wartość pól) jest bardzo prosta. W tym punkcie zajmiemy się sprawdzaniem poprawności danych i ich wstawianiem. **Weryfikacja danych**

Dokładność, za którą tak wysoko cenimy programy komputerowe, a w szczególności bazy danych, ma też swoje drugie oblicze. Żaden serwer baz danych w przeciwieństwie do zdecydowanej większości ludzi nie jest w stanie poprawnie zinterpretować błędnych rekordów jako danych dotyczących tej samej osoby (tabela 8.1).

Tabela 8.1. Błędnie wprowadzone informacje o jednej osobie

Id_osoby

Imię

Nazwisko

Nr_telefonu

Dział

Ostatnia wpłata

1

Tomek

Bronek

233-96-00

Hydraulicy

4000

2

Tomel

Bronek

233-96-00

Hydraulicy

4000

Ponieważ ludzie w przeciwieństwie do komputerów stosunkowo często popełniają różne drobne błędy, np. takie jak w omawianym przypadku „literówka" w imieniu pracownika, dane powinny zostać sprawdzone przed wprowadzeniem ich do bazy. W przeciwnym razie, chociaż wszystkie informacje będą przechowywane w bazie danych, przy okazji przygotowywania raportów lub zestawień statystycznych otrzymamy wyniki mające niewiele wspólnego z prawidłowymi. Drugim punktem oprócz ludzkich błędów, na który należy zwrócić szczególną uwagę, jest opracowanie i konsekwentne egzekwowanie konwencji formatowania danych, zrozumiałej dla wszystkich użytkowników. Jeżeli na przykład zdefiniujemy dla kolumny Nr_telefonu typ danych varchar bez żadnych dodatkowych warunków lub zaleceń dla użytkowników, to możemy spodziewać się danych mniej więcej takich, jakie przedstawiono w tabeli 8.2.

Tabela 8.2. Niesformatowane dane o numerach telefonów

Id_osoby

Imię

Nazwisko

Nr_telefonu

Dział

1

Tomek

Bronek

2339600

Hydraulicy

2

Tom

Bimbom

233-96-00

Lotnicy

3

Stan

B.

(33) 233-96-88

Obcy

4

Zuzanna

J.

33.233.96.88

Położne

Inne elementy wpływające na jakość sformatowania danych to: używanie wielkich liter zgodnie z przyjętymi zasadami oraz unikanie spacji poprzedzających lub kończących wprowadzane łańcuchy i niewnoszących żadnych dodatkowych informacji.

Jak widzimy, przydatność informacji zgromadzonych w bazie danych w dużym stopniu zależy od poprawnego i zgodnego z przyjętymi jednolitymi konwencjami formatowania, które musi nastąpić przed zapisaniem danych do bazy.

Ponieważ pomysłowość ludzka w przeciwieństwie do możliwości obliczeniowych komputera nie ma granic, to nie istnieje w pełni automatyczna metoda usuwania błędów powstałych podczas modyfikowania danych. Za pomocą (przedstawionych w dalszej części kursu) właściwe dobranych typów kolumn, ograniczeń (ang. Check) i wyzwalaczy (ang. Triggers) możemy jednak wyeliminować 90% takich błędów. **Wstawianie pojedynczych wierszy**

Do dodawania nowych wierszy do tabel służy instrukcja INSERT. Wykonanie instrukcji powoduje dodanie wierszy do tabeli — niemożliwe jest dodanie tylko wybranych pól. Podstawowa struktura instrukcji INSERT wygląda następująco:

INSERT INTO nazwa tabeli [(lista kolumn)]

VALUES (lista_wartości)

MySQL obsługuje niestandardowe rozszerzenie instrukcji INSERT DELEYED INTO …, które pozwala użytkownikom wstawić dane do tabel typów MyISAM, MEMORY lub ARCHIVE bez oczekiwania na koniec wykonania instrukcji. Rozszerzenie to stosowane jest głównie w przypadku baz danych zapisujących dane w czasie rzeczywistym, np. bazy dziennika jakiejś aplikacji.

Listing 8.13 pokazuje prostą instrukcję INSERT.

Listing 8.13. Wstawiamy do bazy dodatkowy kod kreskowy towaru o identyfikatorze 1

INSERT INTO barcode (barcode_ean, item_id)

VALUES ('9879879337429', 1);

Query OK, 1 row affected (0.13 sec)

Wstawiane dane muszą uwzględniać zawężenia nałożone na tabelę. Na przykład próba dodania wiersza bez podania wartości dla kolumny z nałożonym zawężeniem NOT NULL zakończy się błędem.

Jeżeli wstawiamy dane do wszystkich kolumn określonej tabeli, tak jak miało to miejsce w powyższym przykładzie, możemy pominąć parametr lista kolumn. Jednak w takim przypadku kolejne wartości w klauzuli VALUES muszą być wprowadzane w kolejności odpowiadającej kolumnom tabeli, a dodatkowo, jeżeli zmieni się struktura tabeli, to instrukcja będzie działać nieprawidłowo. Z tego powodu podczas wprowadzania wierszy zaleca się podawanie listy kolumn (listing 8.14).

Listing 8.14. Krótsza, ale niepolecana wersja poprzedniej instrukcji

INSERT INTO barcode

VALUES ('9871179337429', 1);

Query OK, 1 row affected (0.11 sec)

Język SQL dopuszcza używanie w klauzuli VALUES prostych funkcji, o ile ich wynikiem jest pojedyncza wartość. Natomiast niedozwolone jest stosowanie podzapytań. Aby na przykład dodać zamówienie z bieżącą datą, możemy napisać (listing 8.15):

Listing 8.15. W klauzuli VALUES można stosować dowolne funkcje skalarne

INSERT INTO orderinfo (orderinfo_id, customer_id, date_placed)

VALUES (100,1,NOW());

Query OK, 1 row affected (0.08 sec)

SELECT *

FROM orderinfo

WHERE orderinfo_id=100;

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

| orderinfo_id | customer_id | date_placed | date_shipped | shipping |

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

| 100 | 1 | 2006-04-24 | | NULL |

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

Konstruktor wierszy

Wstawianie wierszy jeden po drugim jest uciążliwe i mało wydajne. Dlatego niektóre serwery bazodanowe (w tym MySQL) umożliwiają wymienienie w klauzuli VALUES danych, które zostaną wstawione do wielu wierszy jednocześnie (listing 8.16).

Listing 8.16. Instrukcja wstawiająca dwa wiersze

INSERT INTO barcode (barcode_ean,item_id)

VALUES ('6264537346173',12),('9473627464534',13);

Query OK, 2 rows affected (0.07 sec)

Records: 2 Duplicates: 0 Warnings: 0

Wstawianie wartości Null

W języku SQL dysponujemy dwiema metodami wprowadzania wartości Null do pól tabeli. W pierwszym przypadku, jeżeli pole ma zadeklarowaną wartość domyślną jako Null, możemy użyć instrukcji INSERT, pomijając wybraną kolumnę na liście kolumn (listing 8.17).

Listing 8.17. Ponieważ w tabeli customer tylko trzy kolumny są wymagane, a wartości jednej z nich, customer_id, są automatycznie generowane przez MySQL, nowego klienta możemy dodać w taki sposób

INSERT INTO customer (lname,zipcode)

VALUES ('Floyd','AABBCCDD');

SELECT customer_id, lname, fname, zipcode

FROM customer

WHERE customer_id>16;

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

| customer_id | lname | fname | zipcode |

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

| 17 | Floyd | | AABBCCDD |

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

Jeżeli dla danej kolumny zdefiniowana jest wartość domyślna, to pominięcie jej w instrukcji INSERT spowoduje wstawienie tej wartości. Aby w takim wypadku wprowadzić wartość Null do tego pola, należy jawnie podać nazwę kolumny oraz wartość Null (listing 8.18).

Listing 8.18. Bezpieczniejszy sposób wstawiania wartości Null

```
INSERT INTO customer (lname,zipcode, fname)

VALUES ('Crimson','AABBDDZZ', Null);

Query OK, 1 row affected (0.11 sec)


SELECT customer_id, lname, fname, zipcode

FROM customer

WHERE customer_id>16;

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

| customer_id | lname   | fname | zipcode  |

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

| 17          | Floyd   |       | AABBCCDD |

| 18          | Crimson |       | AABBDDZZ |

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

Zwróć uwagę na brak apostrofów przy wartości Null — gdybyśmy ją umieścili w apostrofach, MySQL potraktowałby ją jako dosłowny ciąg znaków, a nie zinterpretował jako wartość nieokreśloną (listing 8.19).

Listing 8.19. Wartość Null i ciąg znaków 'Null' to dwie zupełnie różne rzeczy

```
INSERT INTO customer (lname,zipcode, fname)

VALUES ('Waits','XXBBDDZZ', 'Null');

Query OK, 1 row affected (0.10 sec)


SELECT customer_id, lname, fname, zipcode

FROM customer WHERE customer_id>16;

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

| customer_id | lname | fname | zipcode |

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

| 17 | Floyd | | AABBCCDD |

| 18 | Crimson | | AABBDDZZ |

| 19 | Waits | Null | XXBBDDZZ |

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

Wstawianie danych wybranych w zapytaniu

Język SQL umożliwia wstawianie do tabeli wierszy, które są wynikiem wykonania instrukcji SELECT. W ten sposób za pomocą pojedynczej instrukcji INSERT możemy dodać do tabeli dowolną liczbę wierszy. Ogólna składnia takiej instrukcji wygląda następująco:

INSERT INTO nazwa tabeli

[(lista kolumn)]

SELECT zapytanie

Instrukcja INSERT z klauzulą SELECT pozwala na dodanie do tabeli dowolnej liczby wierszy wybranych w zapytaniu.

Zwróć uwagę na kilka spraw:

1.

Wynik zapytania w całości będzie dodany do tabeli docelowej.

2.

Tabela docelowa musi być utworzona wcześniej.

3.

Próba dodania danych innych typów niż typy kolumn tabeli docelowej zakończy się błędem.

4.

Wstawiane dane muszą uwzględniać zawężenia nałożone na tabelę.

Przykład z listingu 8.20 pokazuje, jak utworzyć tabelę i wstawić do niej wybrane w zapytaniu dane.

Listing 8.20. Najpierw tworzymy tymczasową tabelę, potem wstawiamy do niej zwrócone przez zapytanie dane, a następnie odczytujemy zawartość utworzonej tabeli

CREATE TABLE temp

(name varchar(64),

price numeric(7,2));

Query OK, 0 rows affected (0.28 sec)

```
INSERT INTO temp

SELECT description, sell_price

FROM item

WHERE sell_price>10;

Query OK, 5 rows affected (0.14 sec)

Records: 5  Duplicates: 0  Warnings: 0



SELECT *

FROM temp;

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

| name        | price |

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

| Wood Puzzle | 21.95 |

| Rubik Cube  | 11.49 |

| Fan Small   | 15.75 |

| Fan Large   | 19.95 |

| Speakers    | 25.32 |

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

Instrukcja SELECT zagnieżdżona w instrukcji INSERT jest szczególnie przydatna podczas modyfikowania tabel — niektóre zmiany struktury tabeli wymagają wykonania kilku operacji. Po pierwsze, należy utworzyć kopię tabeli, która ma być modyfikowana. Następnie należy skopiować do niej wszystkie dane i usunąć tabelę źródłową. W końcu należy utworzyć nową tabelę, odzwierciedlającą pożądane zmiany, i skopiować do niej dane zapisane w tymczasowej tabeli.

Usuwanie danych

Informacje przechowywane w bazie danych możemy usuwać za pomocą instrukcji DELETE lub TRUNCATE. W pierwszym przypadku istnieje możliwość określenia, które dane będą usunięte z tabeli, w drugim — wszystkie dane z tabeli zostaną nieodwracalne skasowanie.

Instrukcja DELETE

Instrukcja DELETE usuwa wybrane wiersze tabeli. Składnia instrukcji jest bardzo prosta:

```
DELETE FROM nazwa tabeli

[WHERE warunek]
```

Instrukcja DELETE usuwa całe wiersze; nie ma możliwości usunięcia wybranych pół tabeli. Opcjonalna część z klauzulą WHERE jest wykorzystywana do ograniczenia wierszy, które mają być usunięte. Pominięcie tej części powoduje, że wszystkie wiersze tabeli zostaną skasowane (listing 8.21).

Listing 8.21. Usuwamy całą zawartość tabeli temp

```
DELETE FROM temp;

Query OK, 5 rows affected (0.03 sec)


SELECT *

FROM temp;

Empty set (0.04 sec)
```

Używając klauzuli WHERE, można określić warunki potrzebne do usunięcia rekordu (listing 8.22).

Listing 8.22. Usuwamy kody wybranego towaru

```
DELETE FROM barcode

WHERE item_id=1;

Query OK, 5 rows affected (0.05 sec)


SELECT *

FROM barcode;

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

| barcode_ean   | item_id |

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

| 6241574635234 | 2       |

| 6241527746363 | 3       |

| 6264537836173 | 3       |

| 7465743843764 | 4       |

| 3453458677628 | 5       |

| 6434564564544 | 6       |

| 8476736836876 | 7       |

| 6241234586487 | 8       |
```

```
| 9473625532534 | 8       |

| 9473627464543 | 8       |

| 4587263646878 | 9       |

| 2239872376872 | 11      |

| 9879879837489 | 11      |

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

MySQL usuwa wiersze z tabeli jeden po drugim, co dość niekorzystnie wpływa na wydajność całej operacji, ale może być wyzyskane do określenia usuwanych wierszy. Jeżeli wymusimy kolejność usuwania klauzulą ORDER BY, to za pomocą klauzuli LIMIT będziemy mogli sensownie ograniczyć liczbę usuwanych wierszy, tak samo jak miało to miejsce w instrukcji SELECT (listing 8.23).

Listing 8.23. Usuwamy informacje o ostatnio złożonym zamówieniu

```
SELECT *

FROM orderinfo

ORDER BY date_placed DESC

LIMIT 1;

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

| orderinfo_id | customer_id | date_placed |  date_shipped | shipping |

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

| 100          | 1           | 2006-04-24  |               | NULL     |

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

1 row in set (0.01 sec)


DELETE FROM orderinfo

ORDER BY date_placed DESC

LIMIT 1;

Query OK, 1 row affected (0.01 sec)


SELECT * FROM orderinfo

ORDER BY date_placed DESC
```

LIMIT 1;

```
+--------------+------------+------------+--------------+----------+
| orderinfo_id | customer_id | date_placed |  date_shipped | shipping |
+--------------+------------+------------+--------------+----------+
| 4            | 13         | 2000-09-03  | 2000-09-10   |  2.99    |
+--------------+------------+------------+--------------+----------+
```

Usuwanie danych wybranych w zapytaniu

Język SQL umożliwia usuwanie danych spełniających jakiś warunek bez konieczności ich wypisywania. Wystarczy zagnieździć zwracającą pojedynczą wartość instrukcję SELECT w instrukcji INSERT, tak jak pokazuje to listing 8.24.

Listing 8.24. Usuwamy zamówienia pracowników o podanych nazwiskach

```
DELETE FROM orderinfo
WHERE customer_id IN (SELECT customer_id
FROM customer
WHERE lname IN ('Waits','Floyd','Howard'));
```

Instrukcja TRUNCATE TABLE

Możemy usunąć z wybranej tabeli wszystkie dane, wykonując instrukcję TRUNCATE TABLE. Chociaż w przypadku tabel typu InnoDB funkcjonalnym odpowiednikiem wykonania instrukcji TRUNCATE TABLE jest wykonanie instrukcji DELETE bez klauzuli WHERE, to w przypadku innych typów tabel jest między nimi istotna różnica — TRUNCATE TABLE usuwa, a następnie odtwarza pustą tabelę, co jest o wiele szybsze niż usuwanie wierszy tabeli po kolei (listing 8.25).

Listing 8.25. Szybkie usuwanie całej zawartości tabeli

```
TRUNCATE TABLE orderline;
Query OK, 12 rows affected (0.12 sec)
```

Modyfikowanie danych

Baza danych jest przydatna tylko wtedy, gdy zawiera kompletne i aktualne dane. Serwery baz danych ułatwiają przechowywanie i przetwarzanie danych, ale w zamian nakładają na użytkowników i administratora obowiązek ciągłego ich uaktualniania — dopisywania nowych informacji, usuwania błędnych lub niepotrzebnych danych i modyfikowania istniejących. Do wprowadzania zmian w zapisanych danych służy instrukcja UPDATE.

```
UPDATE nazwa tabeli
SET nazwa kolumny = wartość | podzapytanie , ...
[WHERE warunek]
```

Instrukcja składa się z trzech klauzul:

1. W pierwszej części określamy, która tabela będzie aktualizowana.

2. Druga część — klauzula SET — służy do podania listy kolumn, które będą zmieniane, i do naniesienia nowych wartości, które zostaną przypisane tym kolumnom.
3. W ostatniej części za pomocą klauzuli WHERE określamy wiersze tabeli, w których nastąpi zmiana wartości.

Zapamiętaj — Zaktualizowane będą wszystkie dane spełniające warunek podany w klauzuli WHERE. Brak tej klauzuli spowoduje zmianę wartości wszystkich danych zapisanych w tabeli. Oczywiście, aktualizowane dane muszą uwzględniać zawężenia nałożone na tabelę, a próba zmiany typu danych zakończy się błędem, chyba że MySQL będzie mógł dokonać niejawnej konwersji nowego typu do typu zdefiniowanego dla danej kolumny. Ponadto niemożliwe jest jednoczesne aktualizowanie danych zapisanych w kilku tabelach. Listing 8.26 pokazuje, w jaki sposób można zmienić imię wskazanego klienta.

Listing 8.26. Pominięcie klauzuli WHERE spowodowałoby zmianę imion wszystkich klientów na Pink

```
UPDATE customer

SET fname = 'Pink'

WHERE customer_id=15;

Query OK, 1 row affected (0.10 sec)

Rows matched: 1  Changed: 1  Warnings: 0



SELECT lname, fname, customer_id

FROM customer

WHERE customer_id=15;

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

| lname  | fname | customer_id |

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

| Hudson | Pink  | 15          |

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

Do określenia, które wiersze mają być zmodyfikowane, należy wykorzystywać kolumny przechowujące dla danego obiektu atrybuty-klucze. W ten sposób będziemy mieli pewność, że zmodyfikowaliśmy tylko te dane, które chcieliśmy zmienić.

Modyfikowanie danych w wielu kolumnach

Język SQL umożliwia zmodyfikowanie jednocześnie dowolnej liczby kolumn wskazanych wierszy. Przypuśćmy, że jedna z naszych klientek wyprowadziła się od nowego mieszkania. Dane o adresie przechowywane są w dwóch kolumnach: kolumnie addressline i kolumnie town. Aby uaktualnić dane o adresie Jenny Stone, należy wykonać instrukcję (listing 8.27):

Listing 8.27. Przykład jednoczesnego zmieniania danych w kilku kolumnach

```
UPDATE customer

SET addressline = '11 Holy Wood' , town =  'Ankh'

WHERE lname = 'Stones' AND fname = 'Jenny';

Query OK, 1 row affected (0.14 sec)

Rows matched: 1  Changed: 1  Warnings: 0
```

Modyfikowanie danych na podstawie danych wybranych w zapytaniu

Zamiast podawać nową wartość dla wierszy w poszczególnych kolumnach, możemy wykorzystać dane zwracane przez instrukcję SELECT. Jest to szczególnie przydatne, gdy chcemy zmodyfikować dane, opierając się na informacjach przechowywanych w bazie. Jeżeli na przykład chcemy podnieść o dziesięć procent cenę sprzedaży tych towarów, których dotychczasowa cena sprzedaży była niższa od 10, napiszemy (listing 8.28):

Listing 8.28. W klauzuli SET możemy używać prostych wyrażeń. Ostrzeżenia wynikają z operacji na liczbach dziesiętnych

```
UPDATE item

SET sell_price = sell_price*1.1

WHERE sell_price<10;

Query OK, 5 rows affected, 5 warnings (0.15  sec)

Rows matched: 6  Changed: 5  Warnings: 5
```

MySQL umożliwia zagnieżdżenie instrukcji SELECT w klauzuli SET instrukcji UPDATE. W rezultacie możemy zmienić wartość wierszy na podstawie danych przechowywanych w bazie bez konieczności wcześniejszego sprawdzania tych danych (listing 8.29).

Listing 8.29. W wersji 5.1 nie możemy w podzapytaniu odwoływać się do modyfikowanej tabeli, ale możemy odczytywać zawartość każdej innej tabeli

```
UPDATE temp

SET price = (SELECT cost_price

FROM item

WHERE item_id=1)

WHERE name ='SQL Server 2005';
```

Modyfikowanie danych wybranych w zapytaniu

Język SQL umożliwia również modyfikowanie wierszy wybranych na podstawie wyniku instrukcji SELECT. Dzięki temu możliwe jest zmodyfikowanie danych, które spełniają bardziej skomplikowane kryteria wyboru. Aby na przykład obniżyć cenę sprzedaży towaru kupionego za najniższą cenę, napiszemy (listing 8.30).

Listing 8.30. Dynamiczne określanie modyfikowanych danych. W pierwszej kolejności odczytujemy datę złożenia ostatniego zamówienia, następnie identyfikator tego zamówienia, a na końcu przeceniamy wszystkie towary

sprzedane w ramach tego zamówienia

```
UPDATE item

SET sell_price = sell_price*0.75

WHERE item_id IN (SELECT item_id

FROM orderinfo

WHERE date_placed = (SELECT MAX(date_placed)

FROM orderinfo)

);
```

Query OK, 10 rows affected, 7 warnings (0.08 sec)