Do tego momentu zainstalowaliśmy i skonfigurowaliśmy serwer MySQL. Nauczyłeś się administrować nim oraz odczytywać i modyfikować zapisane w bazach dane. Z ostatnich odcinków dowiedziałeś się, jak zaprojektować bazę danych, i wcieliłeś projekt w życie. Właściwie masz w pełni funkcjonalną bazę danych i umiesz z niej korzystać. W tym i następnym odcinku dowiesz się, jak tworzyć dodatkowe obiekty baz danych — widoki, funkcje, procedury składowane i wyzwalacze.

Wprowadzenie

Widok jest predefiniowanym, zapisanym po stronie serwera zapytaniem, którego wynik może być wielokrotnie odczytywany. W standardzie ANSI SQL widoki określane są mianem tabel widokowych lub wirtualnych (ang. Viewed, virtual tables), natomiast tabele przechowujące dane noszą miano tabel bazowych (ang. Base tables). Ta nazwa sugeruje, że z perspektywy użytkownika widok niczym nie różni się od tabeli — użytkownik do obu obiektów odwołuje się tak samo.

Widoki nie przechowują kopii zapisanych w tabelach danych. Widok (ang. View) to jedynie zapisane pod podaną nazwą i w określonym schemacie bazy zapytanie (instrukcja SELECT). Serwery bazodanowe przechowują definicje widoków i związane z nimi metadane (m.in. uprawnienia oraz informacje o zależnościach pomiędzy widokami i innymi obiektami bazy danych), ale nie kopie danych zwracanych przez widoki.

Widoki wykorzystywane są, aby:

1. Ułatwić odczytywanie danych użytkownikom. Jeżeli widok pobiera dane na przykład z siedmiu tabel bazowych, odczytanie danych poprzez widok będzie wymagało wykonania prostej instrukcji SELECT.
2. Ułatwić odczytywanie użytkownikom danych obliczonych na podstawie informacji zapisanych w tabelach. Na przykład odczytanie widoku obliczającego średnią liczbę sprzedanych towarów i grupującego wyniki według nazw kategorii produktów będzie wymagało wykonania prostej instrukcji SELECT.
3. Ograniczyć, ze względów bezpieczeństwa, dostęp użytkowników do poufnych danych. Jeżeli widok pobiera dane na przykład jedynie z wybranych kolumn tabeli, przez ten widok możliwe będzie odczytanie jedynie wybranych danych (np. imienia i nazwiska, ale nie pensji pracowników).
4. Ukryć przed użytkownikami strukturę tabel bazy danych. Informacja ta nie jest niezbędna użytkownikom do pracy z bazą, a przez niektóre osoby może zostać wykorzystana w niewłaściwy sposób.
5. Ułatwić zarządzanie uprawnieniami użytkowników.
6. Oddzielić i uniezależnić aplikację kliencką od zmian tabel bazy danych.

Tworzenie widoków

Aby utworzyć widok, należy wykonać instrukcję:

CREATE [OR REPLACE] VIEW Nazwa widoku

[(lista kolumn)]

AS SELECT zapytanie

gdzie:

1. Opcjonalny parametr lista kolumn definiuje nazwy kolumn dla perspektywy. Lista kolumn tworzonej perspektywy musi zawierać dokładnie tyle pozycji, ile kolumn zwraca instrukcja SELECT. Jeżeli w klauzuli SELECT użyliśmy wyrażeń, należy jawnie określić nazwy kolumn widoku.
2. Zapytanie jest dowolną — z kilkoma wyjątkami — poprawnie sformułowaną instrukcją SELECT języka SQL.
3. Jeżeli chcemy zmienić wcześniej utworzony widok, należy po słowie kluczowym CREATE użyć opcji OR REPLACE.

Listing 11.1 pokazuje sposób utworzenia widoku ograniczającego użytkownikom dostęp do poufnych informacji.

Ćwiczenia z tego i następnych odcinków wykonaj w przykładowej bazie test.

Listing 11.1. Widok zawierający jedynie tytuł, imię, nazwisko i numer telefonu kontrahenta

```
CREATE VIEW V_ks_tele

AS SELECT title, fname, lname

FROM customer;

Query OK, 0 rows affected (0.13 sec)
```

Tworząca widok instrukcja SELECT musi zwracać dane tabelaryczne. Oznacza to, że:

1. Nawet jeżeli dany serwer bazodanowy pozwala zwracać dane w innej postaci (np. dokumentów XML), w definicji widoku nie można skorzystać z tej funkcjonalności.
2. Każda kolumna musi mieć niepowtarzalną nazwę, czyli jeżeli w klauzuli SELECT znajdują się jakieś wyrażenia, trzeba nadać im aliasy.
3. Niemożliwe jest użycie w definicji widoku instrukcji SELECT … INTO (w przeciwnym razie kolejne odwołanie się do widoku spowodowałoby błąd, bo tabela z kopią danych już by istniała, a nazwy tabel muszą być niepowtarzalne).

Z utworzonego widoku można korzystać tak jak ze zwykłej tabeli. Wykonanie instrukcji z listingu 11.2 zwróci dane teleadresowe klientów.

Listing 11.2. Do widoków, tak jak do tabel, odwołujemy się w klauzuli FROM

```
SELECT *

FROM V_ks_tele

WHERE title LIKE 'mr';

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

| title | fname   | lname   |

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

| Mr    | Andrew  | Stones  |

| Mr    | Adrian  | Matthew |

| Mr    | Simon   | Cozens  |

| Mr    | Neil    | Matthew |

| Mr    | Richard | Stones  |

| Mr    | Mike    | Howard  |

| Mr    | Dave    | Jones   |
```

| Mr | Richard | Neill |

| Mr | Bill | Neill |

| Mr | Pink | Hudson |

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

Widoki umożliwiają prezentowanie danych w sposób najbardziej wygodny dla użytkowników, niezależnie od struktury tabel fizycznych. Użytkownikom o wiele łatwiej wpisać polecenie SELECT * nazwa widoku zawierającego wyłącznie kolumny najczęściej przez nich używane, niż wypisywać poszczególne nazwy kolumn przy korzystaniu z tabeli (oczywiście, istnieje również możliwość wybierania konkretnych kolumn z widoku). Jeżeli w trakcie pracy bazy danych okaże się, że bardzo często wykonywany jest pewien typ złączenia lub podzapytania, zapytanie tego typu można zapisać jako widok, który ułatwi pracę użytkownikom bazy. Listing 11.3 pokazuje widok znacznie upraszczający odczytywanie listy towarów zakupionych przez poszczególnych klientów.

Listing 11.3. Widok zwracający listę towarów kupionych w 2000 roku przez poszczególnych klientów. Jak widać, widoki znacznie upraszczają odczytywanie danych

```
CREATE VIEW V_zakupy AS

SELECT lname, fname, description, cost_price, sell_price, sell_price-cost_price AS zysk

FROM customer JOIN orderinfo USING (customer_id)

JOIN orderline USING (orderinfo_id)

JOIN item USING (item_id)

WHERE YEAR(date_placed)=2000;

Query OK, 0 rows affected (0.04 sec)



SELECT *

FROM V_zakupy

WHERE fname LIKE 'Alex';

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

| lname   | fname | description | cost_price | sell_price | zysk  |

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

| Matthew | Alex  | Tissues     | 2.11       | 2.22       | 0.11  |

| Matthew | Alex  | Fan Large   | 13.36      |  11.22     | -2.14 |

| Matthew | Alex  | Roman Coin  | 2.34       | 1.37       |  -0.97 |

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

W języku SQL widoki mogą odczytywać dane nie tylko z tabel bazowych, ale również z innych widoków. Dobrą praktyką programowania jest tworzenie widoków realizujących pojedyncze zadania, takich jak widok z listingu 11.3, jedynie łączący dane zapisane w różnych tabelach. Gdybyśmy teraz chcieli przygotować raport finansowy zawierający informacje o wartości sprzedanych towarów, moglibyśmy utworzyć widok z listingu 11.4.

Listing 11.4. Widok odwołujący się do danych zwracanych przez inny widok

```
CREATE VIEW V_zysk AS

SELECT CONCAT(lname,' ', fname) AS klient,   SUM(zysk)

FROM V_zakupy

ORDER BY SUM(zysk);

Query OK, 0 rows affected (0.04 sec)



SELECT * FROM v_zysk;

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

| klient       | SUM(zysk) |

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

| Stones Ann   | -8.40     |

| Hudson Pink  | -3.86     |

| Matthew Alex | -3.00     |

| Hendy Laura  | -1.38     |

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

Złączenie zewnętrzne w definicji widoków

Tworząc widoki pobierające dane z kilku obiektów bazowych, należy unikać łączenia tych obiektów poprzez jakiekolwiek złączenie zewnętrzne. Chociaż poniższa instrukcja jest poprawną instrukcją języka SQL, to odczytanie danych poprzez widok może przynieść nieoczekiwane rezultaty (listing 11.5).

Listing 11.5. Gdyby w tabelach bazowych znajdowały się rekordy zawierające wartości nieokreślone w kolumnach, z których widok pobiera dane, to wyświetlenie poprzez widok wszystkich danych, w których wartością kolumny pobieranej z wewnętrznej tabeli złączenia jest NULL, spowodowałoby przypisanie wartości nieokreślonej do wszystkich wierszy wewnętrznej tabeli

```
CREATE VIEW V_l_zam AS

SELECT CONCAT(lname,' ' ,fname),   COUNT(orderinfo_id) AS 'Liczba zamowien'

FROM customer LEFT OUTER JOIN orderinfo USING  (customer_id)

GROUP BY CONCAT(lname,' ' ,fname);
```

Query OK, 0 rows affected (0.21 sec)

Uporządkowywanie danych poprzez widoki

Widoki są często najprostszym sposobem na „poprawę" bazy danych, której projekt nie spełnia wymogów 1PN. Z reguły administrator wykorzystuje wcześniej i przez kogoś innego utworzoną bazę danych, a polityka firmy nie pozwala na gruntowną przebudowę struktury produkcyjnej bazy.

Ponieważ instrukcja SELECT umożliwia, za pomocą operatora teoriomnogościowego UNION, łączenie wyników wielu zapytań, możemy, bez konieczności tworzenia niewydajnych kursorów po stronie serwera, „uporządkować" tabelę znajdującą się w pierwszej postaci anormalnej. Załóżmy, że zastaliśmy poniższą tabelę w bazie danych:

kursant {imie, kurs_stopnia_1, kurs_stopnia_2, kurs_stopnia_3}

Pomijając wszystkie opisane w odcinku 9. niedogodności związane z tym, że tabela nie spełnia wymogów 1PN, pobieranie i prezentowanie użytkownikom danych z tej tabeli rodzi dodatkowe trudności. Wyobraź sobie kod programu, który na podstawie nazwy kursu (dowolnego stopnia) wyświetli dodatkowe informacje o kursie, takie jak data jego rozpoczęcia czy czas trwania. A teraz wyobraź sobie ten sam kod przy założeniu, że w tabeli kursant zapisane są informacje o kursach 15 różnych stopni.

Rozwiązaniem części tych problemów będzie utworzenie poniższego widoku (listing 11.6).

Listing 11.6. W przykładowej bazie danych nie ma tabeli kursant

```
CREATE VIEW w_kursant
AS SELECT *
FROM (
  SELECT kurs_stopnia_1 as nazwa_kursu
  FROM kursant
  UNION ALL
  SELECT kurs_stopnia_2 as nazwa_kursu
  FROM kursant
  UNION ALL
  SELECT kurs_stopnia_3 as nazwa_kursu
  FROM kursant
  ) n
ORDER BY n.nazwa_kursu
```

W ten sposób nazwy wszystkich kursów można odczytać, odwołując się do jednej kolumny nazwa_kursu.

Modyfikowanie danych poprzez widoki

Ponieważ widok nie przechowuje kopii danych, ale jest jedynie „pryzmatem", przez który odczytujemy dane

zapisane w tabelach, modyfikacja danych widoku jest w rzeczywistości modyfikacją danych zapisanych w tabelach. Widoki, w odniesieniu do których mogą być wykonywane instrukcje INSERT, DELETE i UPDATE, nazywane są widokami modyfikowalnymi (ang. Updatable views).

Modyfikowanie danych jest możliwe, o ile:
1. Każdy pojedynczy wiersz widoku odpowiada dokładnie jednemu wierszowi tabeli (widok jest powiązany z tabelami bazowymi związkiem jedno-jednoznacznym). Konsekwencją tego ograniczenia jest to, że widoki, przy których definicji użyto:
1. funkcji grupujących w klauzuli SELECT,
2. operatora DISTINCT,
3. klauzul GROUP BY lub HAVING,
4. wyrażeń matematycznych na liście argumentów klauzuli SELECT,

nie mogą być wykorzystane do modyfikowania danych w tabeli.
1. Dane zostaną jednocześnie zmodyfikowane w tylko jednej tabeli bazowej. Nie oznacza to, że widok musi pobierać dane z tylko jednej tabeli bazowej, ale że jednocześnie mogą być zmodyfikowane wartości kolumn jednej tabeli.
2. Zmiana nie narusza zawężeń zdefiniowanych dla tabeli bazowej, czyli przy modyfikowaniu danych poprzez widoki obowiązują wszystkie więzy integralności zdefiniowane dla tabeli. Jeżeli na przykład widok nie pobiera danych z tabeli, na którą nałożono ograniczenie NOT NULL, wstawianie nowych wierszy poprzez widok będzie niemożliwe. Tak samo niemożliwe jest (zarówno bezpośrednie, jak poprzez widok) wstawienie występujących już w kolumnie wartości, jeżeli nałożono na nią ograniczenie UNIQUE.
3. Niedozwolone jest używanie do modyfikowania danych widoków zawierających odwołania do innych niemodyfikowalnych widoków.
4. Jest zgodne z opcjonalnym, opisanym dalej warunkiem WITH CHECK OPTION.

Listing 11.7 pokazuje, w jaki sposób można zmienić dane poprzez widok.

Listing 11.7. Zmieniamy nazwisko osoby poprzez widok i sprawdzamy zmianę w tabeli bazowej

```
UPDATE V_ks_tele

SET fname = 'King'

WHERE fname = 'Pink';

Query OK, 1 row affected (0.07 sec)

Rows matched: 1  Changed: 1  Warnings: 0



SELECT customer_id, fname

FROM customer

WHERE fname = 'King';

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

| customer_id | fname |
```

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

| 15 | King |

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

Ograniczenie zakresu modyfikowania danych poprzez widoki

Załóżmy, że zdefiniowaliśmy widok zwracający dane o towarach, których cena sprzedaży jest równa 10 lub wyższa (listing 11.8).

Listing 11.8. Ponieważ tak zdefiniowany widok spełnia wszystkie wyżej wymienione warunki, możemy poprzez niego modyfikować przechowywane w tabeli item dane

```
CREATE VIEW V_drogie AS

SELECT *

FROM item

WHERE sell_price >=10;
```

Query OK, 0 rows affected (0.01 sec)

Skoro poprzez widok można modyfikować dane, to w szczególności można zmienić cenę sprzedaży towaru. Ponieważ na liście towarów dostępnych przez widok znajdują się tylko te towary, których cena przekracza 10, obniżenie ceny jednego z towarów spowoduje, że nie będzie on dalej poprzez ten widok dostępny (listing 11.9).

Listing 11.9. Odczytujemy dane poprzez widok, następnie zmieniamy cenę towaru i raz jeszcze odczytujemy zwracane przez ten widok dane

```
SELECT *

FROM V_drogie;
```

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

| item_id | description | cost_price | sell_price |

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

| 1 | Wood Puzzle | 15.23 | 12.35 |

| 7 | Fan Large | 13.36 | 11.22 |

| 11 | Speakers | 19.73 | 14.24 |

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

3 rows in set (0.07 sec)

```
UPDATE V_drogie
```

```
SET sell_price = 8

WHERE item_id=1;

Query OK, 1 row affected (0.03 sec)

Rows matched: 1  Changed: 1  Warnings: 0



SELECT *

FROM V_drogie;

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

| item_id | description | cost_price |  sell_price |

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

| 7       | Fan Large   | 13.36      | 11.22      |

| 11      | Speakers    | 19.73      | 14.24      |

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

Użytkownik odniesie wrażenie niespójności bazy danych — towar, którego cenę właśnie zmienił, zniknął! Aby uniemożliwić przeprowadzenie tego typu zmian, należy podczas definiowania widoku dołączyć klauzulę WITH CHECK OPTION, jak przedstawiono to na listingu 11.10.

Listing 11.10. Warto do wszystkich modyfikowalnych widoków dodać klauzulę WITH CHECK OPTION — na pewno poprawi to wrażenie spójności danych

```
CREATE VIEW V_drogie_v2 AS

SELECT *

FROM item

WHERE sell_price >=10

WITH CHECK OPTION;

Query OK, 0 rows affected (0.03 sec)



UPDATE V_drogie_v2

SET sell_price = 4

WHERE item_id=7;

ERROR 1369 (HY000): CHECK OPTION failed  'test.v_drogie_v2'
```

Oczywiście, zmiany, które nie spowodują „zniknięcia” wiersza, nadal można przeprowadzać poprzez widok (listing 11.11).

Listing 11.11. Aktualizacja danych poprzez zmodyfikowany widok

```
UPDATE V_drogie_v2

SET cost_price = 4

WHERE item_id=7;

Query OK, 1 row affected (0.02 sec)

Rows matched: 1  Changed: 1  Warnings: 0



SELECT *

FROM v_drogie_v2;

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

| item_id | description | cost_price |  sell_price |

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

| 1       | Wood Puzzle | 15.23      |  13.00      |

| 7       | Fan Large   | 4.00       |  11.22      |

| 11      | Speakers    | 19.73      |  14.24      |

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

Modyfikowanie widoków

Definicję widoku możemy zmienić poprzez wykonanie instrukcji ALTER VIEW. W ten sposób zmodyfikujemy definicje widoku, ale nadane do niego użytkownikom uprawnienia pozostaną (listing 11.12).

Listing 11.12. Zmiana definicji widoku

```
ALTER VIEW V_drogie AS

SELECT *

FROM item

WHERE sell_price >=15

WITH CHECK OPTION;

Query OK, 0 rows affected (0.28 sec)
```

Usuwanie widoków

Niepotrzebne widoki można usunąć z bazy danych. Ponieważ usunięcie tabeli bazowej nie spowoduje automatycznego usunięcia powiązanych z nią widoków, z reguły instrukcje DROP TABLE i DROP VIEW wykonywane są łącznie (listing 11.13).

Listing 11.13. Usuwamy niepotrzebny widok

```
DROP VIEW v_drogie_v2;

Query OK, 0 rows affected (0.01 sec)
```

Podobnie jak w przypadku usuwania tabel, aby MySQL usunął widok, musimy mieć nadane uprawnienia administratora baz danych lub być właścicielem tego widoku.

Podsumowując, należy stwierdzić, że użytkownicy nie powinni mieć bezpośredniego dostępu do tabel. Zamiast tego powinni odczytywać i modyfikować dane poprzez widoki, funkcje lub procedury. Uzyskamy w ten sposób:

1. Funkcjonalny mechanizm kontroli dostępu do danych. Jeżeli tylko część danych z tabeli powinna być udostępniona wszystkim użytkownikom (np. imiona i nazwiska pracowników, ale nie ich adresy czy numery telefonów), to tworząc widok zwracający wyłącznie ogólnodostępne dane i odbierając użytkownikom bezpośredni dostęp do tabeli, zapewnimy poufność pozostałych danych.

2. Warstwę abstrakcji pozwalającą na zmianę struktury tabel bez konieczności aktualizacji programów klienckich. Schemat każdej produkcyjnej bazy danych prędzej czy później będzie zmieniany w związku z nowymi wymaganiami albo na potrzeby optymalizacji. Jeżeli użytkownicy od początku odczytywali dane poprzez widoki, zmiana struktury tabel może zostać skompensowana odpowiednią zmianą widoków i nie będzie wtedy wymagała aktualizacji wszystkich programów klienckich.

3. Możliwość łatwiejszego i szybszego przesyłania zapytań przez sieć — zamiast długich zapytań przesyłane będą wywołania procedur lub funkcji czy proste zapytania kierowane do widoków.