Bazy danych

10. SQL — Widoki, procedury składowane, kursory i wyzwalacze

P. F. Góra

http://th-www.if.uj.edu.pl/zfs/gora/

semestr letni 2007/08

I. Widoki, *AKA* Perspektywy

W SQL tabela, którą utworzono za pomocą zapytania `CREATE TABLE`, nazywa się *tabelą podstawową* (*base table*). Jej struktura i dane przechowywane są przez system operacyjny.

Wynik każdego zapytania `SELECT` też, formalnie, jest tabelą. Tabelę taką nazywa się *tabelą pochodną* (*derived table*).

Widokiem (lub perspektywą) nazywam trwałą definicję tabeli pochodnej, która to definicja przechowywana jest w bazie danych.

Jak tworzymy widoki?

```
CREATE VIEW nazwa
AS SELECT treść_zapytania_select;
```

Przykład

Tworzymy następujące tabele:

```
mysql> CREATE TABLE arabic
    -> (i INT UNSIGNED NOT NULL, b VARCHAR(8)) CHARSET=cp1250;
Query OK, 0 rows affected (0.87 sec)
mysql> CREATE TABLE roman (i INT UNSIGNED NOT NULL, r VARCHAR(4));
Query OK, 0 rows affected (0.42 sec)
mysql> SET CHARSET cp1250;
Query OK, 0 rows affected (0.04 sec)
mysql> INSERT INTO arabic VALUES
    -> (1,'jeden'),(2,'dwa'),(3,'trzy'),(4,'cztery'),(5,'pięć');
Query OK, 5 rows affected (0.13 sec)
Records: 5  Duplicates: 0  Warnings: 0
mysql> INSERT INTO roman VALUES
    -> (1,'I'),(2,'II'),(3,'III'),(4,'IV');
Query OK, 4 rows affected (1.17 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

```
mysql> SELECT * FROM arabic;
+---+--------+
| i | b      |
+---+--------+
| 1 | jeden  |
| 2 | dwa    |
| 3 | trzy   |
| 4 | cztery |
| 5 | pięć   |
+---+--------+
5 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM roman;
+---+------+
| i | r    |
+---+------+
| 1 | I    |
| 2 | II   |
| 3 | III  |
| 4 | IV   |
+---+------+
4 rows in set (0.00 sec)
```

Tworzymy najprostszy widok

```
mysql> CREATE VIEW v1
    -> AS SELECT * FROM arabic;
Query OK, 0 rows affected (1.14 sec)

mysql> SELECT * FROM v1;
+---+--------+
| i | b      |
+---+--------+
| 1 | jeden  |
| 2 | dwa    |
| 3 | trzy   |
| 4 | cztery |
| 5 | pięć   |
+---+--------+
5 rows in set (1.13 sec)
```

Tak utworzony widok jest w gruncie rzeczy aliasem tabeli.

Widoki można tworzyć na podstawie widoków.

Wyrażenie `SELECT` w definicji widoku może zawierać klauzulę `WHERE`.

Można brać tylko wybrane kolumny. Można nadawać im aliasy.

```
mysql> CREATE VIEW v2
    -> AS SELECT b AS napis FROM v1
    -> WHERE i > 3;
Query OK, 0 rows affected (0.15 sec)

mysql> SELECT * FROM v2;
+--------+
| napis  |
+--------+
| cztery |
| pięć   |
+--------+
2 rows in set (0.05 sec)
```

Underlying tables

Tabele, których użyto do zdefiniowania widoku, są nazywane *underlying tables*.

Widok `v1` ma jedną underlying table: `arabic`. Widok `v2` ma *dwie* underlying tables: tabelę `arabic` oraz widok `v1`.

Widoki a zmiana *underlying table*

Widoki reagują na zmianę danych w tabelach, których użyto do ich stworzenia:

```
mysql> INSERT INTO arabic VALUES(6,'sześć');
Query OK, 1 row affected (0.43 sec)

mysql> SELECT * FROM v2;
+--------+
| napis  |
+--------+
| cztery |
| pięć   |
| sześć  |
+--------+
3 rows in set (0.38 sec)
```

Widoki nie reagują na zmianę *definicji* tabeli, o ile nie narusza ona integralności widoku.

```
mysql> ALTER TABLE arabic ADD COLUMN (x CHAR(1) DEFAULT 'X');
Query OK, 6 rows affected (4.51 sec)
Records: 6  Duplicates: 0  Warnings: 0
mysql> SELECT * FROM arabic WHERE i>4;
+---+--------+------+
| i | b      | x    |
+---+--------+------+
| 5 | pięć   | X    |
| 6 | sześć  | X    |
+---+--------+------+
2 rows in set (0.89 sec)
mysql> SELECT * FROM v1 WHERE i>4;
+---+--------+
| i | b      |
+---+--------+
| 5 | pięć   |
| 6 | sześć  |
+---+--------+
2 rows in set (0.00 sec)
```

Można tworzyć całkiem skomplikowane widoki

```
mysql> CREATE VIEW lewy
    -> AS SELECT arabic.i AS i, b, r FROM arabic LEFT JOIN roman
    -> ON arabic.i=roman.i;
Query OK, 0 rows affected (1.23 sec)
mysql> SELECT * FROM lewy;
+---+--------+------+
| i | b      | r    |
+---+--------+------+
| 1 | jeden  | I    |
| 2 | dwa    | II   |
| 3 | trzy   | III  |
| 4 | cztery | IV   |
| 5 | pięć   |      |
| 6 | sześć  |      |
+---+--------+------+
6 rows in set (0.18 sec)
```

Po co widoki?

```
mysql> CREATE TABLE Ceny
    -> (ProducentId SMALLINT UNSIGNED NOT NULL,
    -> ProduktId SMALLINT UNSIGNED NOT NULL,
    -> Cena DECIMAL(10,2) NOT NULL,
    -> PRIMARY KEY(ProducentId, ProduktId));
mysql> CREATE TABLE Dostawy
    -> (NrDostawy INT UNSIGNED NOT NULL PRIMARY KEY,
    -> ProducentId SMALLINT UNSIGNED NOT NULL,
    -> ProduktId SMALLINT UNSIGNED NOT NULL,
    -> Ilosc INT UNSIGNED NOT NULL);

mysql> CREATE VIEW ksiegowosc
    -> AS SELECT ProducentID, Ilosc*Cena AS Kwota
    -> FROM Dostawy NATURAL JOIN Ceny;
mysql> CREATE VIEW produkcja
    -> AS SELECT ProduktId, SUM(Ilosc) AS Zapas
    -> FROM Dostawy
    -> GROUP BY ProduktId;
```

W bazie zapamiętane zostaną tylko *definicje* poszczególnych kolumn widoków, nie zaś wartości!

Różni użytkownicy bazy chcą/mogą/powinni mieć różny dostęp do tych samych danych źródłowych.

Definiowanie widoków pozwala na przechowywanie całkiem złożonych definicji w bazie danych i późniejsze wykorzystywanie ich w zapytaniach. Jest to szczególnie wygodne gdy z tą samą bazą współpracują *różne* aplikacje.

Widoki modyfikowalne

Niektóre widoki są *modyfikowalne* — zmiana danych w widoku powoduje zmianę danych w underlying tables.

```
mysql> UPDATE v2 SET napis="six" WHERE napis LIKE "sz%";
Query OK, 1 row affected (1.33 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM arabic;
+---+--------+------+
| i | b      | x    |
+---+--------+------+
| 1 | jeden  | X    |
| 2 | dwa    | X    |
| 3 | trzy   | X    |
| 4 | cztery | X    |
| 5 | pięć   | X    |
| 6 | six    | X    |
+---+--------+------+
6 rows in set (0.00 sec)
```

Ale...

```
mysql> INSERT INTO v2 VALUES ('siedem');
ERROR 1423 (HY000): Field of view 'widoki.v2' underlying table
doesn't have a default value
```

Nie można dodać lub zmienić wiersza widoku modyfikowalnego, jeżeli zmiana taka naruszałaby więzy integralności underlying tables.

Gdyby kolumna `i` tabeli `arabic` była określona jako `AUTO_INCREMENT` lub też gdyby nie była określona jako `NOT NULL`, powyższe wstawienie wiersza do widoku zadziałałoby.

Jeżeli zatem chcemy udostępniać użytkownikom (aplikacjom) widoki, które mają być modyfikowalne, trzeba *dobrze* przemyśleć strukturę całej bazy.

Widoki niemodyfikowalne

Niektóre widoki są z założenia niemodyfikowalne: Niemodyfikowalne są te, które w liście `SELECT` zawierają `UNION`, `UNION ALL`, `DISTINCT`, `DISTINCTROW`, inny widok niemodyfikowalny lub podzapytanie.

Podzapytania w widokach

W MySQL widoki nie mogą zawierać podzapytań, ale problem da się obejść! Jak? tworząc widoki, które spełniają rolę podzapytań. W ten sposób można tworzyć bardzo złożone widoki.

Przykład: W pewnej bazie istnieją tabele

```
mysql> DESCRIBE Studenci;
+----------+---------------------+------+-----+---------+----------------+
| Field    | Type                | Null | Key | Default | Extra          |
+----------+---------------------+------+-----+---------+----------------+
| nr       | tinyint(3) unsigned | NO   | PRI |         | auto_increment |
| Imie     | varchar(16)         | NO   | MUL |         |                |
| Nazwisko | varchar(16)         | NO   |     |         |                |
+----------+---------------------+------+-----+---------+----------------+

mysql> DESCRIBE Bazy;
+-------+---------------------+------+-----+------------+-------+
| Field | Type                | Null | Key | Default    | Extra |
+-------+---------------------+------+-----+------------+-------+
| nr    | tinyint(3) unsigned | NO   | PRI |            |       |
| data  | date                | NO   | PRI | 0000-00-00 |       |
| OK    | char(2)             | YES  |     | OK         |       |
+-------+---------------------+------+-----+------------+-------+
```

Istnieją też tabele `Metnum` oraz `Szeregi`, o strukturze takiej samej, jak tabela `Bazy`.

Teraz możemy utworzyć widoki

```
mysql> CREATE VIEW KtoBazy AS
    -> SELECT Nr, COUNT(*) AS C FROM Bazy WHERE OK="OK" GROUP BY Nr;

mysql> CREATE VIEW KtoMetnum AS
    -> SELECT Nr, COUNT(*) AS C FROM Metnum WHERE OK="OK" GROUP BY Nr;

mysql> CREATE VIEW KtoSzeregi AS
    -> SELECT Nr, COUNT(*) AS C FROM Szeregi WHERE OK="OK" GROUP BY Nr;

mysql> CREATE VIEW KtoIle AS
    -> SELECT Imie, Nazwisko, KtoBazy.C AS Bazy, KtoMetnum.C AS Metnum,
    -> KtoSzeregi.C AS Szeregi
    -> FROM Studenci
    -> LEFT JOIN KtoBazy USING (Nr)
    -> LEFT JOIN KtoMetnum USING (Nr)
    -> LEFT JOIN KtoSzeregi USING (Nr)
    -> ORDER BY Nazwisko;
```

```
mysql> SELECT * FROM KtoIle LIMIT 6;
+-----------+------------+------+--------+---------+
| Imie      | Nazwisko   | Bazy | Metnum | Szeregi |
+-----------+------------+------+--------+---------+
| Karol     | Adamczyk   | 2    | NULL   | 2       |
| Mateusz   | Armatys    | 1    | NULL   | NULL    |
| Katarzyna | Bąk        | NULL | 4      | NULL    |
| Rafał     | Baranowski | NULL | 4      | NULL    |
| Paweł     | Buczkowski | NULL | NULL   | NULL    |
| Kamila    | Czaplicka  | NULL | 4      | NULL    |
+-----------+------------+------+--------+---------+
6 rows in set (0.00 sec)
```

Usuwanie wierszy z widoków

Niekiedy z widoków modyfikowalnych można usuwać wiersze — mianowicie wtedy, gdy SQL potrafi “przetłumaczyć” zapytanie usuwające wiersze z widoku na zapytanie (zapytania) usuwające wiersze z underlying table(s).

```
mysql> DELETE FROM v2 WHERE napis="pięć";
Query OK, 1 row affected (1.24 sec)

mysql> SELECT * FROM arabic;
+---+--------+------+
| i | b      | x    |
+---+--------+------+
| 1 | jeden  | X    |
| 2 | dwa    | X    |
| 3 | trzy   | X    |
| 4 | cztery | X    |
| 6 | six    | X    |
+---+--------+------+
5 rows in set (0.00 sec)
```

Ale...

```
mysql> SELECT * FROM lewy;
+---+--------+------+
| i | b      | r    |
+---+--------+------+
| 1 | jeden  | I    |
| 2 | dwa    | II   |
| 3 | trzy   | III  |
| 4 | cztery | IV   |
| 6 | six    |      |
+---+--------+------+
5 rows in set (1.10 sec)

mysql> DELETE FROM LEWY WHERE r='I';
ERROR 1395 (HY000): Can not delete from join view 'widoki.lewy'
```

Więz CHECK w widokach

```
CREATE VIEW nazwa
AS SELECT treść_zapytania_select WHERE warunek
[WITH [CASCADED | LOCAL] CHECK OPTION];
```

Jeśli w zapytaniu tworzącym widok jest klauzula WHERE, więz CHECK uniemożliwi takie dodanie/zmodyfikowanie danych *do widoku*, które naruszałoby tę klauzulę.

CASCADED i LOCAL mają znaczenie jeśli widok tworzony jest na podstawie innych widoków. Jeżeli wybierzemy CASCADED, klauzule WHERE "podwidoków" też są sprawdzane, nawet jeśli nie nałożono na nie jawnie więzu CHECK.

Przykład

```
mysql> CREATE VIEW v3
    -> AS SELECT i, b FROM arabic
    -> WHERE i < 6;
Query OK, 0 rows affected (1.13 sec)

mysql> SELECT * FROM v3;
+---+--------+
| i | b      |
+---+--------+
| 1 | jeden  |
| 2 | dwa    |
| 3 | trzy   |
| 4 | cztery |
+---+--------+
4 rows in set (0.00 sec)
```

```
mysql> CREATE VIEW v4
    -> AS SELECT i, b FROM arabic
    -> WHERE i < 6
    -> WITH CHECK OPTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM v4;
+---+--------+
| i | b      |
+---+--------+
| 1 | jeden  |
| 2 | dwa    |
| 3 | trzy   |
| 4 | cztery |
+---+--------+
4 rows in set (0.00 sec)
```

Na tym etapie nie ma różnicy — klauzula `WHERE` wybrała odpowiednie wiersze.

```
mysql> INSERT INTO v3 VALUES (7,'siedem');
Query OK, 1 row affected (1.10 sec)

mysql> SELECT * FROM v3;
+---+--------+
| i | b      |
+---+--------+
| 1 | jeden  |
| 2 | dwa    |
| 3 | trzy   |
| 4 | cztery |
+---+--------+
4 rows in set (0.01 sec)

mysql> INSERT INTO v4 VALUES (8,'osiem');
ERROR 1369 (HY000): CHECK OPTION failed 'widoki.v4'
```

Zawartość widoku `v3` nie zmieniła się, ale pozwolił on na modyfikację underlying table, co możnaby sprawdzić wykonując zapytanie `SELECT * FROM arabic`. Widok `v4`, z klauzulą `CHECK`, na to nie pozwolił.

Widoki “dyndające”

Jeżeli usuniemy tabelę, nie usuniemy zaś opartych na niej widoków, lub też tak zmodyfikujemy tabelę, że definicja widoku straci sens, dostaniemy widoki “dyndające” (*dangling views*). Widoki są wypisywane na liście `SHOW TABLES`;

```
mysql> SHOW TABLES;
+------------------+
| Tables_in_widoki |
+------------------+
| arabic           |
| ceny             |
| dostawy          |
| ksiegowosc       |
| lewy             |
| produkcja        |
| roman            |
| v1               |
| v2               |
| v3               |
| v4               |
+------------------+
11 rows in set (1.12 sec)
```

Aby dowiedzieć się czy dana tabela lub widok nie są uszkodzone, dajemy polecenie

```
mysql> CHECK TABLE roman;
+--------------+-------+----------+----------+
| Table        | Op    | Msg_type | Msg_text |
+--------------+-------+----------+----------+
| widoki.roman | check | status   | OK       |
+--------------+-------+----------+----------+
1 row in set (1.23 sec)

mysql> CHECK TABLE lewy;
+-------------+-------+----------+----------+
| Table       | Op    | Msg_type | Msg_text |
+-------------+-------+----------+----------+
| widoki.lewy | check | status   | OK       |
+-------------+-------+----------+----------+
1 row in set (0.16 sec)
```

W tym momencie wszystko jest w porządku.

Ale...

```
mysql> ALTER TABLE roman DROP COLUMN r;
Query OK, 4 rows affected (1.88 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> CHECK TABLE lewy;
+-------------+-------+----------+-----------------------------------------+
| Table       | Op    | Msg_type | Msg_text                                |
+-------------+-------+----------+-----------------------------------------+
| widoki.lewy | check | error    | View 'widoki.lewy' references invalid   |
|             |       |          | table(s) or column(s) or function(s) or |
|             |       |          | definer/invoker of view lack rights to use |
|             |       |          | them                                    |
+-------------+-------+----------+-----------------------------------------+
1 row in set (0.02 sec)

mysql> DROP VIEW lewy;
Query OK, 0 rows affected (0.01 sec)
```

II. Procedury składowane (*stored procedures*)

Procedury składowane stanowią część schematu bazy danych. Stosuje się je do wykonowania powtarzających się, logicznie takich samych operacji na (bazie) danych, nie wymagających ingerencji ze strony użytkownika.

Zalety używania procedur składowanych:

- Różne aplikacje korzystające z tej samej bazy danych korzystają z tej samej procedury — mniejsze ryzyko popełnienia błędu.
- Zwiększone bezpieczeństwo: Nie udostępniamy użytkownikom bezpośrednio tabel (hacker mógłby dobrać się do tabeli z hasłami!), zezwalamy *tylko* na wykonywanie predefiniowanych procedur.
- Mniejsze koszty uruchomienia i konserwacji.
- Z punktu widzenia wydajności
  - procedura wykonywana jest przez *wolniejszy* język, ale na *szybszym* serwerze,
  - *znaczne zmniejszenie kosztu przesyłu danych*.

Najprostszy przykład

```
mysql> DELIMITER//
```

Przed zdefiniowaniem procedury należy “redefiniować średnik”, żeby średniki w ciele procedury nie były interpretowane jako koniec zapytania definiującego procedurę.

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE proc01 ()
    -> /* To jest komentarz */
    -> SELECT * FROM arabic;   /* "arabic" jest nazwą tabeli */
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
```

Wywołanie procedury — instrukcja `CALL`

```
mysql> CALL proc01 ();
+----+-----------+------+
| i  | b         | x    |
+----+-----------+------+
| 1  | jeden     | X    |
| 2  | dwa       | X    |
| 4  | cztery    | X    |
| 3  | trzy      | X    |
| 5  | pięć      | X    |
| 6  | sześć     | X    |
| 7  | siedem    | X    |
| 8  | osiem     | X    |
| 9  | dziewięć  | X    |
| 10 | dziesięć  | X    |
| 12 | dwanaście | X    |
+----+-----------+------+
11 rows in set (0.00 sec)

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

Klauzule definicyjne

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE proc02()
    -> LANGUAGE SQL
    -> NOT DETERMINISTIC
    -> SQL SECURITY INVOKER
    -> COMMENT 'Przykład'
    -> SELECT RAND();
    -> //
Query OK, 0 rows affected (1.14 sec)
mysql> DELIMITER ;
mysql> CALL proc02 ();
+------------------+
| RAND()           |
+------------------+
| 0.16568405103468 |
+------------------+
1 row in set (0.02 sec)
Query OK, 0 rows affected (0.02 sec)
```

`LANGUAGE SQL` wymagane ze względu na kompatybilność

`NOT DETERMINISTIC` bo przy takich samych parametrach może dać różne wyniki
`DETERMINISTIC` jeśli parametry wywołania jednoznacznie determinują wynik

`SQL SECURITY INVOKER` przy wywołaniu sprawdzaj przywileje wywołującego
`SQL SECURITY DEFINER` przy wywołaniu sprawdzaj przywileje użytkownika, który stworzył procedurę

Instrukcje złożone. Parametry wywołania.

```
mysql> CREATE PROCEDURE proc03 (IN i INT)
    -> LANGUAGE SQL
    -> DETERMINISTIC
    -> BEGIN
    ->   SELECT i + 2;
    ->   SELECT i - 4;
    -> END; //
Query OK, 0 rows affected (1.71 sec)
mysql> CALL proc03 (2)//
+-------+
| i + 2 |
+-------+
| 4     |
+-------+
1 row in set (0.10 sec)
+-------+
| i - 4 |
+-------+
| -2    |
+-------+
1 row in set (0.10 sec)
Query OK, 0 rows affected (0.10 sec)
```

```
mysql> CREATE PROCEDURE proc04 (INOUT j FLOAT)
    -> LANGUAGE SQL
    -> DETERMINISTIC
    -> BEGIN
    ->   DECLARE z FLOAT;  /* zmienna lokalna */
    ->   SET z = SIN(j);
    ->   SELECT z AS 'zet';
    ->   SET j = j*z;
    -> END; //
Query OK, 0 rows affected (0.00 sec)
```

Zmienne lokalne mają zakres ograniczoney do swojego bloku `BEGIN`–`END`.

```
mysql> SET @x = 2.0//
Query OK, 0 rows affected (0.00 sec)

mysql> CALL proc04 (@x) //
+----------+
| zet      |
+----------+
| 0.909297 |
+----------+
1 row in set (1.12 sec)

Query OK, 0 rows affected (1.13 sec)

mysql> SELECT @x //
+-----------------+
| @x              |
+-----------------+
| 1.8185948133469 |
+-----------------+
1 row in set (0.00 sec)
```

Procedury mogą uzyskać dostęp do tabel

```
mysql> CREATE PROCEDURE proc05 (IN z CHAR(1))
    -> UPDATE arabic SET x=z WHERE MOD(i,2)=0 LIMIT 4;//
Query OK, 0 rows affected (1.12 sec)
mysql> CALL proc05 ('P') //
Query OK, 4 rows affected (0.08 sec)
mysql> SELECT * FROM arabic //
+----+-----------+------+
| i  | b         | x    |
+----+-----------+------+
| 1  | jeden     | X    |
| 2  | dwa       | P    |
| 4  | cztery    | P    |
| 3  | trzy      | X    |
| 5  | pięć      | X    |
| 6  | sześć     | P    |
| 7  | siedem    | X    |
| 8  | osiem     | P    |
| 9  | dziewięć  | X    |
| 10 | dziesięć  | X    |
| 12 | dwanaście | X    |
+----+-----------+------+
11 rows in set (0.00 sec)
```

Czego procedurom *nie wolno* robić?

Procedury nie mogą zmieniać innych procedur. W ciele procedury w MySQL nielegalne są instrukcje `CREATE | ALTER | DROP PROCEDURE`, `CREATE | ALTER | DROP FUNCTION`, `CREATE | ALTER | DROP TRIGGER`.

W MySQL wewnątrz procedury nielegalna jest instrukcja `USE`, można jednak odwoływać się do tabel innej bazy danych niż baza bieżąca (oczywiście o ile mamy do tego uprawnienia).

Procedury mogą natomiast tworzyć, usuwać i modyfikować definicje tabel, widoków i baz danych.

Zakres zmiennych

```
mysql> CREATE PROCEDURE proc06 ()
    -> BEGIN
    ->   DECLARE napis CHAR(4) DEFAULT 'zewn';
    ->   BEGIN
    ->     DECLARE napis CHAR(4) DEFAULT 'wewn';
    ->     SELECT napis;
    ->   END;
    ->   SELECT napis;
    ->   SET napis = 'pqrs';
    ->   SELECT napis;
    -> END;//
Query OK, 0 rows affected (0.62 sec)
```

```
mysql> CALL proc06 () //
+-------+
| napis |
+-------+
| wewn  |
+-------+
1 row in set (0.00 sec)

+-------+
| napis |
+-------+
| zewn  |
+-------+
1 row in set (0.00 sec)

+-------+
| napis |
+-------+
| pqrs  |
+-------+
1 row in set (0.00 sec)
Query OK, 0 rows affected (0.16 sec)
```

Instrukcja warunkowa

```
mysql> CREATE PROCEDURE proc07 (IN j INT)
    -> BEGIN
    ->   DECLARE m INT;
    ->   SET m = (SELECT MAX(i) FROM arabic);
    ->   IF j > m THEN
    ->      INSERT INTO arabic (i,b) VALUES (j,'tekst');
    ->   ELSE
    ->      UPDATE arabic SET i = i + m WHERE i = j;
    ->   END IF;
    -> END;//
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> CALL proc07 (6) //
Query OK, 1 row affected (1.75 sec)
mysql> SELECT * FROM arabic //
+----+-----------+------+
| i  | b         | x    |
+----+-----------+------+
| 1  | jeden     | X    |
| 2  | dwa       | P    |
| 4  | cztery    | P    |
| 3  | trzy      | X    |
| 5  | pięć      | X    |
| 18 | sześć     | P    |
| 7  | siedem    | X    |
| 8  | osiem     | P    |
| 9  | dziewięć  | X    |
| 10 | dziesięć  | X    |
| 12 | dwanaście | X    |
+----+-----------+------+
11 rows in set (0.02 sec)
```

```
mysql> CALL proc07 (19) //
Query OK, 1 row affected (0.13 sec)
mysql> SELECT * FROM arabic //
+----+-----------+------+
| i  | b         | x    |
+----+-----------+------+
| 1  | jeden     | X    |
| 2  | dwa       | P    |
| 4  | cztery    | P    |
| 3  | trzy      | X    |
| 5  | pięć      | X    |
| 18 | sześć     | P    |
| 7  | siedem    | X    |
| 8  | osiem     | P    |
| 9  | dziewięć  | X    |
| 10 | dziesięć  | X    |
| 12 | dwanaście | X    |
| 19 | tekst     | X    |
+----+-----------+------+
12 rows in set (0.00 sec)
```

Instrukcja CASE

```
mysql> CREATE PROCEDURE proc08 (IN j INT)
    -> CASE j
    ->   WHEN 0 THEN SELECT j AS 'wynik';
    ->   WHEN 1 THEN SELECT 5*j AS 'wynik';
    ->   ELSE SELECT 10*j AS 'wynik';
    -> END CASE; //
Query OK, 0 rows affected (0.00 sec)
mysql> CALL proc08 (0) //
+-------+
| wynik |
+-------+
| 0     |
+-------+
1 row in set (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
mysql> CALL proc08 (2) //
+-------+
| wynik |
+-------+
| 20    |
+-------+
1 row in set (0.00 sec)
Query OK, 0 rows affected (0.01 sec)
```

Pętla `WHILE`

```
mysql> CREATE PROCEDURE proc09 ()
    -> BEGIN
    ->   DECLARE v INT;
    ->   SET v = 1;
    ->   WHILE v < 3 DO
    ->     SELECT b FROM arabic
    ->       WHERE i = v;
    ->     SET v = v + 1;
    ->   END WHILE;
    -> END; //
Query OK, 0 rows affected (1.27 sec)
```

```
mysql> CALL proc09 () //
+-------+
| b     |
+-------+
| jeden |
+-------+
1 row in set (0.45 sec)

+------+
| b    |
+------+
| dwa  |
+------+
1 row in set (0.47 sec)

Query OK, 0 rows affected (0.47 sec)
```

Pętla REPEAT

```
mysql> CREATE PROCEDURE proc10 (OUT s INT)
    -> BEGIN
    ->   DECLARE j INT DEFAULT 0;
    ->   SET s = 1;
    ->   REPEAT
    ->     SET j = j + 1;
    ->     SET s = s * j;
    ->     UNTIL j > 5
    ->   END REPEAT;
    -> END; //
Query OK, 0 rows affected (0.97 sec)

mysql> CALL proc10(@s) //
Query OK, 0 rows affected (0.08 sec)

mysql> SELECT @s //
+------+
| @s   |
+------+
| 720  |
+------+
1 row in set (0.00 sec)
```

Pętla LOOP, instrukcja LEAVE i etykiety

```
mysql> CREATE PROCEDURE proc11 (IN k INT)
    -> BEGIN
    ->   DECLARE s, v INT DEFAULT 1;
    ->   etykieta: LOOP
    ->     SET s = s*v;
    ->     SET v = v+1;
    ->     IF v > k THEN LEAVE etykieta; END IF;
    ->   END LOOP;
    ->   SELECT k, s;
    -> END; //
Query OK, 0 rows affected (0.94 sec)

mysql> CALL proc11 (6) //
+------+------+
| k    | s    |
+------+------+
| 6    | 720  |
+------+------+
1 row in set (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
```

Etykiety można stawiać przed BEGIN, WHILE, REPEAT i LOOP.

Instrukcja `ITERATE`

`ITERATE` — zignoruj resztę ciała pętli i przejdź do następnej iteracji.

```
mysql> CREATE PROCEDURE proc12 ()
    -> BEGIN
    ->   DECLARE j INT DEFAULT 1;
    ->   ett: REPEAT
    ->     SET j = j + 1;
    ->     IF j = 4  THEN ITERATE ett; END IF;
    ->     SELECT i, b FROM arabic WHERE i=j;
    ->     UNTIL j = 6
    ->   END REPEAT ett;
    -> END; //
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> CALL proc12 () //
+---+------+
| i | b    |
+---+------+
| 2 | dwa  |
+---+------+
1 row in set (0.00 sec)

+---+------+
| i | b    |
+---+------+
| 3 | trzy |
+---+------+
1 row in set (0.00 sec)

+---+------+
| i | b    |
+---+------+
| 5 | pięć |
+---+------+
1 row in set (0.00 sec)

Empty set (0.00 sec)

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

Obsługa błędów (*Error handling*)

"Naturalnym" zastosowanie procedur składowanych jest obsługa błędów, jakie mogą pojawić się przy pracy z bazą danych, na przykład na skutek naruszenia więzów. Rozpatrzmy przykład:

```
mysql> CREATE TABLE tabela
    -> (K SMALLINT UNSIGNED NOT NULL PRIMARY KEY,
    ->  W INT UNSIGNED);
Query OK, 0 rows affected (1.14 sec)

mysql> INSERT INTO tabela VALUES (1,10);
Query OK, 1 row affected (0.12 sec)

mysql> INSERT INTO tabela VALUES (2,-1);
ERROR 1264 (22003): Out of range value adjusted for column 'W' at row 1
mysql> INSERT INTO tabela VALUES (1,5);
ERROR 1062 (23000): Duplicate entry '1' for key 1
```

Pojawienie się błędów przy nie-interaktywnym wprowadzaniu danych może spowodować problemy, których chcemy uniknąć. Zarazem chcemy wiedzieć jakie błędy się pojawiły.

```
mysql> CREATE TABLE log_bledow (B VARCHAR(80)) CHARSET cp1250;
Query OK, 0 rows affected (1.31 sec)

mysql> DELIMITER //
mysql> CREATE PROCEDURE safeinsert (IN k INT, IN w INT)
    -> BEGIN
    ->   /* Pierwszy handler */
    ->   DECLARE EXIT HANDLER FOR 1062
    ->   BEGIN
    ->     SET CHARSET cp1250;
    ->     INSERT INTO log_bledow VALUES
    ->       (CONCAT('Godzina: ',current_time,' powtórzony klucz ',k));
    ->   END;
    ->   /* Drugi handler */
    ->   DECLARE EXIT HANDLER FOR 1264
    ->   BEGIN
    ->     SET CHARSET cp1250;
    ->     INSERT INTO log_bledow VALUES
    ->       (CONCAT('Godzina: ',current_time,' ujemna wartość ',w));
    ->   END;
    ->   /* Ciało procedury - wstawianie */
    ->   INSERT INTO tabela VALUES (k, w);
    -> END; //
Query OK, 0 rows affected (0.49 sec)
```

```
mysql> CALL safeinsert (1,5);
Query OK, 1 row affected (0.09 sec)
mysql> CALL safeinsert (1,6);
Query OK, 1 row affected (0.11 sec)
mysql> CALL safeinsert (2,7);
Query OK, 1 row affected (0.04 sec)
mysql> CALL safeinsert (3,-1);
Query OK, 1 row affected (0.03 sec)

mysql> SELECT * FROM tabela;
+---+------+
| K | W    |
+---+------+
| 1 | 5    |
| 2 | 7    |
+---+------+
2 rows in set (0.00 sec)
mysql> SELECT * FROM log_bledow;
+-------------------------------------+
| B                                   |
+-------------------------------------+
| Godzina: 09:44:06 powtórzony klucz 1 |
| Godzina: 09:44:36 ujemna wartość -1  |
+-------------------------------------+
2 rows in set (0.00 sec)
```

Polecenia `PREPARE` i `EXECUTE`

Przypuśmy, że mamy kilka tabel o *takiej samej* strukturze, na przykład

```
mysql> CREATE TABLE Kowalski
    -> (Data DATE NOT NULL,
    ->  Miasto VARCHAR(16) NOT NULL,
    ->  Kwota DECIMAL(8,2) UNSIGNED NOT NULL,
    ->  PRIMARY KEY(Data, Miasto));
Query OK, 0 rows affected (1.10 sec)

mysql> CREATE TABLE Nowak LIKE Kowalski;
Query OK, 0 rows affected (0.16 sec)

mysql> INSERT INTO Nowak VALUES (...);

mysql> INSERT INTO Kowalski VALUES (...);
```

Tabele `Kowalski` i `Nowak` mają taką samą strukturę, a więc, logicznie rzecz biorąc, można ich użyć w “takim samym” zapytaniu — w zapytaniu różniącym się tylko nazwą tabeli. Po stronie aplikacji zapytanie takie jest bardzo łatwo zbudować jako odpowiedni łańcuch, a następnie przesłać je do serwera SQL. Jednak traci się w ten sposób możliwość korzystania z procedur składowanych, a zatem wszystkie korzyści wynikające z używania takich procedur.

Pytanie:

Czy w SQL nazwy tabel mogą być parametrami procedur składowanych?

Tak — należy jednak wykorzystać polecenia `PREPARE` i `EXECUTE`.

Procedura mająca nazwę tabeli jako argument

```
CREATE PROCEDURE monthsales (IN nazwisko VARCHAR(16), IN miesiac TINYINT UNSIGNED)
LANGUAGE SQL
NOT DETERMINISTIC
SQL SECURITY DEFINER
BEGIN
  DECLARE EXIT HANDLER FOR 1146
  BEGIN
    SELECT nazwisko AS "Nazwisko", "Nie ma takiej tabeli" AS "Brak tabeli";
  END;
  /* tekst tworzonej komendy przechowuję _w_zmiennej_tymczasowej_! */
  SET @tmp = CONCAT('SELECT "',nazwisko,'" AS Nazwisko, SUM(KWOTA) FROM ',nazwisko,
             ' WHERE MONTH(Data)=',miesiac,';');
  SELECT @tmp AS 'Wykonam następującą komendę';
  /* utworzenie zapytania */
  /* "komenda" jest >>handlerem<< utworzonego zapytania */
  PREPARE komenda FROM @tmp;
  /* wykonanie utworzonego zapytania */
  EXECUTE komenda;
  /* usunięcie >>handlera<< */
  DEALLOCATE PREPARE komenda;
END;
```

```
mysql> CALL monthsales ('Kowalski',5);

+-----------------------------------------------------------------------------+
| Wykonam następującą komendę                                                 |
+-----------------------------------------------------------------------------+
| SELECT "Kowalski" AS Nazwisko, SUM(KWOTA) FROM Kowalski WHERE MONTH(Data)=5; |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)

+----------+------------+
| Nazwisko | SUM(KWOTA) |
+----------+------------+
| Kowalski | 1741.30    |
+----------+------------+
1 row in set (0.01 sec)

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

```
mysql> CALL monthsales ('Nowak', 5);

+-----------------------------------------------------------------------+
| Wykonam następującą komendę                                           |
+-----------------------------------------------------------------------+
| SELECT "Nowak" AS Nazwisko, SUM(KWOTA) FROM Nowak WHERE MONTH(Data)=5; |
+-----------------------------------------------------------------------+
1 row in set (0.00 sec)

+----------+------------+
| Nazwisko | SUM(KWOTA) |
+----------+------------+
| Nowak    | 4404.51    |
+----------+------------+
1 row in set (0.00 sec)

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

```
mysql> CALL monthsales ('XYZ',4);

+-----------------------------------------------------------------+
| Wykonam następującą komendę                                     |
+-----------------------------------------------------------------+
| SELECT "XYZ" AS Nazwisko, SUM(KWOTA) FROM XYZ WHERE MONTH(Data)=4; |
+-----------------------------------------------------------------+
1 row in set (0.00 sec)

+----------+----------------------+
| Nazwisko | Brak tabeli          |
+----------+----------------------+
| XYZ      | Nie ma takiej tabeli |
+----------+----------------------+
1 row in set (0.01 sec)

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

Funkcje składowane

Ważne ograniczenie w MySQL: *funkcje składowane nie mają dostępu do danych przechowywanych w bazie* ☹

```
mysql> CREATE FUNCTION sumasinusow (n INT)
    ->   RETURNS FLOAT
    ->   DETERMINISTIC
    -> BEGIN
    ->   DECLARE sumasinusow FLOAT DEFAULT 0;
    ->   DECLARE i INT DEFAULT 1;
    ->   WHILE i < n DO
    ->     SET sumasinusow = sumasinusow + SIN(i);
    ->     SET i = i + 1;
    ->   END WHILE;
    ->   RETURN sumasinusow;
    -> END; //
Query OK, 0 rows affected (1.01 sec)
```

Ważne: Zmienna zwracana musi być zainicjalizowana (przez `DEFAULT` lub `SET`), gdyż w przeciwnym razie funkcja zwróci `NULL`.

```
mysql> SELECT sumasinusow(15) //
+------------------+
| sumasinusow(15)  |
+------------------+
|  1.09421646595   |
+------------------+
1 row in set (0.43 sec)
```

III. Kursory

SQL jest językiem imperatywnym: Określamy *co* chcemy zrobić, nie zaś *jak* to zrobić.

Kursory pozwalają na obsługę tabel wiersz po wierszu, a więc stanowią odejście od paradygmatu imperatywnego w stronę paradygmaty deklaratywnego. Obsługa wiersz po wierszu jest typowa dla aplikacji pisanych w językach wysokiego poziomu. Dlaczego więc robi się to w SQL? Żeby zmniejszyć koszt związany z przesyłaniem znacznej ilości danych.

W MySQL kursory realizowane są za pomocą procedur składowanych.

Co można zrobić z kursorem?

- Zadeklarować
- Otworzyć
- Pobrać dane, następnie zaś przejść do następnego wiersza
- Zamknąć

Kursorów warto używać tylko wtedy, gdy “przesuwają się” po kolumnie indeksowanej z unikalnymi wartościami — najlepiej jeśli jest kluczem głównym.

Suma narastająca poprzez samozłączenie

```
mysql> CREATE TEMPORARY TABLE RunningTotal
    ->   (I SMALLINT UNSIGNED NOT NULL PRIMARY KEY,
    ->    X FLOAT, SumaNarastajaca FLOAT)
    -> SELECT a.I, a.X, SUM(b.X) AS SumaNarastajaca
    -> FROM przebieg AS a, przebieg AS b
    -> WHERE b.I <= a.I
    -> GROUP BY a.I;
Query OK, 12 rows affected (0.18 sec)
Records: 12  Duplicates: 0  Warnings: 0
```

```
mysql> SELECT * FROM RunningTotal;
+----+-------+-----------------+
| I  | X     | SumaNarastajaca |
+----+-------+-----------------+
| 1  | 1     | 1               |
| 2  | 1.25  | 2.25            |
| 3  | 1.5   | 3.75            |
| 4  | 1.75  | 5.5             |
| 5  | 2     | 7.5             |
| 6  | -1.1  | 6.4             |
| 7  | -0.85 | 5.55            |
| 8  | -0.6  | 4.95            |
| 9  | -0.35 | 4.6             |
| 10 | -0.3  | 4.3             |
| 11 | -0.5  | 3.8             |
| 12 | -0.75 | 3.05            |
+----+-------+-----------------+
12 rows in set (0.00 sec)
```

Przykład: Suma narastająca poprzez kursor

```
mysql> CREATE PROCEDURE kurs1 ()
    -> BEGIN
    ->   /* Najpierw deklarujemy zmienne */
    ->   DECLARE koniec, j SMALLINT UNSIGNED;  /* zmienna koniec ma wartość NULL */
    ->   DECLARE suma, z FLOAT DEFAULT 0.0;
    ->   /* Potem deklarujemy kursor */
    ->   /* Zapytanie SELECT może być bardzo skomplikowane */
    ->   DECLARE k1 CURSOR FOR SELECT I, X FROM przebieg ORDER BY I;
    ->   /* Co zrobić gdy dojdziemy do ostatniego wiersza */
    ->   DECLARE CONTINUE HANDLER FOR NOT FOUND
    ->     SET koniec = 1;
    ->   /* Zakładamy tabelę tymczasową */
    ->   DROP TEMPORARY TABLE IF EXISTS RunningTotal;
    ->   CREATE TEMPORARY TABLE RunningTotal
    ->     (I SMALLINT UNSIGNED NOT NULL PRIMARY KEY,
    ->      X FLOAT, SumaNarastajaca FLOAT);
```

To nie koniec!

```
->   /* Otwieramy kursor */
->   OPEN k1;
->   /* Pętla wiersz po wierszu */
->   petla: LOOP
->     /* pobranie danych do kursora */
->     FETCH k1 INTO j, z;
->     /* wyjdź jeśli już skończyły się wiersze */
->     IF koniec = 1 THEN
->        LEAVE petla;
->     END IF;
->     /* oblicz i wstaw dane do tabeli tymczasowej */
->     SET suma = suma + z;
->     INSERT INTO RunningTotal VALUES (j,z,suma);
->   END LOOP petla;
->   /* koniec pętli po wierszach */
-> END; //
```

```
mysql> CALL kurs1 () ;
Query OK, 1 row affected (2.17 sec)

mysql> SELECT * FROM RunningTotal;
+----+-------+-----------------+
| I  | X     | SumaNarastajaca |
+----+-------+-----------------+
| 1  | 1     | 1               |
| 2  | 1.25  | 2.25            |
| 3  | 1.5   | 3.75            |
| 4  | 1.75  | 5.5             |
| 5  | 2     | 7.5             |
| 6  | -1.1  | 6.4             |
| 7  | -0.85 | 5.55            |
| 8  | -0.6  | 4.95            |
| 9  | -0.35 | 4.6             |
| 10 | -0.3  | 4.3             |
| 11 | -0.5  | 3.8             |
| 12 | -0.75 | 3.05            |
+----+-------+-----------------+
12 rows in set (0.09 sec)
```

Odstępstwa od standardu SQL

Kursory w MySQL (na razie?) *nie* są zgodne ze standardem SQL. Najważniejsze rzeczy, których brakuje, to

1. Kursory w MySQL mogą być *wrażliwe*, ale nawet nie wiadomo czy są ☹ Jeżeli po otwarciu kursora inny wątek zmieni dane wczytane do kursora, kursor *może*, ale *nie musi*, zauważyć wprowadzone zmiany. Takie kursory nazywane są *asensitive*. Zgodnie ze standardem SQL, kursor można zdefiniować jako *niewrażliwy*:
   `DECLARE` nazwa_kursora `INSENSITIVE CURSOR FOR` ...
   Taki kursor nie widzi zmian wprowadzonych po otwarciu przez inne wątki.

2. Kursory w MySQL są *nieprzewijalne*. Kursor po wczytaniu krotki posuwa się o jedną krotkę do przodu. Zgodnie ze standardem SQL, raz otwarty kursor może dowolnie przesuwać się po swoim zakresie.
   `DECLARE` nazwa_kursora `SCROLL CURSOR FOR ...`
   Wówczas legalne są polecenia `FETCH NEXT`, `FETCH PRIOR`, `FETCH LAST`, `FETCH FIRST`, `FETCH RELATIVE` liczba, `FETCH ABSOLUTE` liczba.

3. Kursory są tylko do odczytu — za pomocą kursorów nie można zmieniać wartości otwartej przez kursor tabeli. Zgodnie ze zstandardem SQL, legalne są polecenia typu
   `UPDATE` tabela `SET ... WHERE CURRENT OF CURSOR` nazwa_kursora;

IV. Wyzwalacze (triggers)

Wyzwalacze są procedurami wykonywanymi automatycznie po zmianie zawartości wskazanej tabeli.

```
CREATE TRIGGER nazwa_wyzwalacza
{BEFORE | AFTER }
{INSERT | UPDATE | DELETE }
ON nazwa_tabeli
FOR EACH ROW
wywoływane wyrażenie SQL;
```

```
mysql> CREATE TRIGGER tabela_trig
    -> BEFORE UPDATE ON tabela
    -> FOR EACH ROW
    -> BEGIN
    ->   SET @s = OLD.w;
    ->   SET @n = NEW.w;
    -> END; //
Query OK, 0 rows affected (1.41 sec)
```

```
mysql> UPDATE tabela SET W = W + 8
    -> WHERE K = 1;
Query OK, 1 row affected (1.40 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT @s, @n;
+------+------+
| @s   | @n   |
+------+------+
| 5    | 13   |
+------+------+
1 row in set (0.06 sec)
```

Kwalifikatory `OLD`, `NEW` odnoszą się do zawartości tabeli przed zmianą i po zmianie. Przy `INSERT` legalny jest tylko `NEW`, przy `DELETE` tylko `OLD`, przy `UPDATE` oba.

Wyzwalacze mogą odwoływać się tylko do trwałych tabel podstawowych (nie do widoków ani tabel tymczasowych). Każda tabela może mieć co najwyżej jeden wyzwalacz z określoną kombinacją `BEFORE | AFTER` + `INSERT | UPDATE | DELETE`.

Wyzwalacz `INSERT` jest uruchamiany także przez `LOAD DATA`. Wyzwalacze `UPDATE, DELETE` nie są uruchamiane przez zapytania usuwające tabele lub modyfikujące jej schemat. Wyzwalacze nie są uruchamiane przez zdarzenia kaskadowe wywoływane przez klucze obce.

Zgodnie ze standardem SQL, możliwe jest definiowanie wyzwalaczy bez klauzuli `FOR EACH ROW` — taki wyzwalacz wywoływany jest raz po każdym zapytaniu wstawiającym/modyfikującym/usuwającym dane, nie zaś dla każdego wstawianego/modyfikowanego/usuwanego wiersza. Ta cecha nie jest zaimplementowana w MySQL.