Bazy danych

8. Widoki, klucze obce i wyzwalacze

P. F. Góra

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

2020

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 o ile podzapytania nie są skorelowane, 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 można było 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)
```

Widoki zmaterializowane

Widoki są wirtualnymi tabelami, zawierającymi rezultaty zapytań — same nie przechowują żadnych danych. Zdarzają się jednak sytuacje, w których przydatne może być fizyczne przechowywanie danych z widoku. Takie widoki nazywamy *widokami zmatrializowanymi* (ang.*materialized views*). W ten sposób możemy tworzyć lokalną kopię danych, będących wynikiem zapytania zadanego tabeli (tabelom) zlokalizowanej na odległym serwerze lub, jeszcze częściej, w systemach rozproszonych, w których różne części danych przechowywane są na różnych serwerach. Przy powtórnym odwołaniu się do takiego widoku, korzystamy z kopii lokalnej, bez konieczności realizowania kosztownej i czasochłonnej komunikacji przez sieć. Od czasu do czasu widok taki odświeżamy aby zapewnić aktualność danych. *Może się zdarzyć*, że dane na odległym serwerze zostaną zmienione zanim przeprowadzimy odświeżenie widoku, ale *per saldo* może i tak

to być opłacalne, zwłaszcza jeśli komunikacja jest powolna, a procent nieaktualnych danych niewielki. Korzystanie z widoków zmaterializowanych zmniejsza konieczność korzystania z sieci w systemach rozległych za cenę ryzyka, iż jakaś część danych nie będzie aktualna.

Widoki zmaterializowane (chwilowo) mogą łamać wymóg spójności danych. Systemy z zaimplementowanymi widokami zmaterializowanymi *formalnie* należałoby zaliczyć do baz danych typu NoSQL.

MySQL nie implementuje widoków zmaterializowanych, ale można to obejść za pomocą wyzwalaczy i procedur składowanych — patrz na przykład `http://www.fromdual.com/mysql-materialized-views`.

Dygresja: Metody przechowywania tabel w MySQL

Tabele w MySQL mogą być przechowywane na kilka sposobów. Sposób ten (żargonowo: silnik, *engine*) jest definiowany przy zakładaniu tabeli. Najważniejsze dwa sposoby to

1. MyISAM — domyślny (natywny) system MySQL. Dostęp do tabel jest szybki, ale sposób ten nie realizuje niektórych rzeczy zdefiniowanych w standardzie SQL, w szczególności nie obsługuje mechanizmu kluczy obcych ani transakcji, co jest przedmiotem niniejszego wykładu.
2. InnoDB — dostęp do tabel może być wyraźnie wolniejszy, ale ten sposób pozwala na stosowanie kluczy obcych i transakcji.

```
CREATE TABLE Fubar(...)  ENGINE=InnoDB;
```

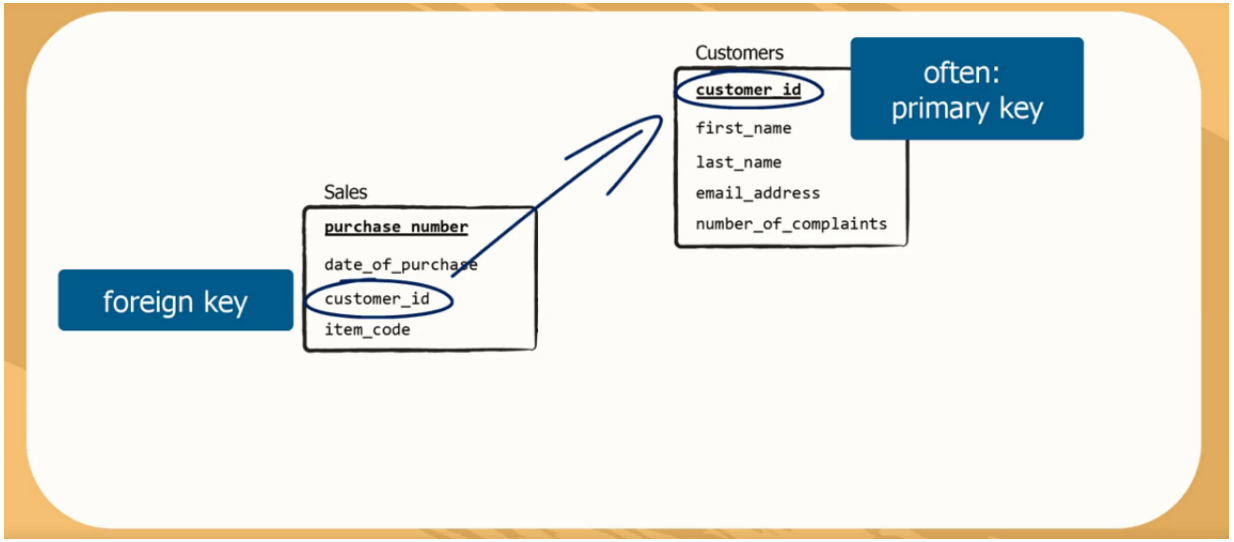
II. Klucze obce

Klucze obce — powiązanie indeksowanej kolumny jakiejś tabeli z indeksowaną kolumną innej tabeli, co pozwala na automatyczne dokonywanie zmian w powiązanych tabelach lub uniemożliwia dokonanie zmian naruszających ograniczenia. *Uwaga: To jest bardzo wygodne, ale* **bardzo** *niebezpieczne!* A poza tym spowalnia działanie bazy.

Mechanizm kluczy obcych jest sposobem na realizację w SQL więzów integralności referencyjnej.

W MySQL działa tylko dla tabel InnoDB.

Integralność referencyjna



Przy próbie dodania krotki do tabeli `Sales`, istnienie klucza obcego wymusi sprawdzenie, czy w tabeli `Customers` istnieje krotka o podanym `customer_id` i uniemożliwi wstawienie nowej krotki do `Sales`, jeśli w `Customers` nie ma wskazywanej wartości `customer_id`.

Składnia

```
FOREIGN KEY (nazwa_kolumny_indeksowanej, ...)
  REFERENCES nazwa_tabeli (nazwa_kolumny_indeksowanej, ...)
  [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
  [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

Znaczenie: **dodatkowe** funkcjonalności

`ON DELETE` — przy usuwaniu krotki sprawdza stan tabel powiązanych i określa, co się z nimi dzieje.
`ON UPDATE` — przy modyfikacji krotki (atrybutu wchodzącego w skład klucza obcego) sprawdza stan tabel powiązanych i określa, co się z nimi dzieje.

`RESTRICT` — nie pozwala na dokonanie zmian naruszających powiązanie.
`CASCADE` — nakazuje zmianom na propagację kaskadową wzdłuż drzewa powiązanych tabel (potencjalnie *bardzo niebezpieczne* przy usuwaniu krotek!).
`SET NULL` — ustawia odpowiednie atrybuty powiązanych tabel, dotąd wskazujące na usuwany/modyfikowany element klucza obcego, na wartość `NULL`, jeśli definicja tabeli to dopuszcza.
`NO ACTION` — wyłącza mechanizm klucza obcego dla danej operacji.

Przykład

```
mysql> CREATE TABLE X
    -> (IdX TINYINT UNSIGNED NOT NULL PRIMARY KEY)
    -> ENGINE=InnoDB;

Query OK, 0 rows affected (0.16 sec)

mysql> CREATE TABLE Y
    -> (IdY CHAR(1) NOT NULL PRIMARY KEY,
    ->  IdX TINYINT UNSIGNED NOT NULL,
    ->  INDEX (IdX),
    ->  FOREIGN KEY (IdX) REFERENCES X (IdX)
    ->    ON DELETE CASCADE
    ->    ON UPDATE CASCADE)
    -> ENGINE=InnoDB;

Query OK, 0 rows affected (0.22 sec)
```

```
mysql> SELECT * FROM Y;
+-----+-----+
| IdY | IdX |
+-----+-----+
| A   | 1   |
| B   | 2   |
| C   | 3   |
| D   | 3   |
| E   | 5   |
+-----+-----+
5 rows in set (0.07 sec)
mysql> UPDATE X Set IdX=9 WHERE IdX=2;
Query OK, 1 row affected (1.19 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> SELECT * FROM Y;
+-----+-----+
| IdY | IdX |
+-----+-----+
| A   | 1   |
| C   | 3   |
| D   | 3   |
| E   | 5   |
| B   | 9   |
+-----+-----+
5 rows in set (0.00 sec)
```



```
mysql> DELETE FROM X WHERE IdX=3;
Query OK, 1 row affected (0.07 sec)

mysql> SELECT * FROM Y;
+-----+-----+
| IdY | IdX |
+-----+-----+
| A   | 1   |
| E   | 5   |
| B   | 9   |
+-----+-----+
3 rows in set (0.00 sec)
```


Przykład 2

```
mysql> CREATE TABLE Up
    -> (Nr SMALLINT UNSIGNED NOT NULL PRIMARY KEY,
    -> A CHAR(1) NOT NULL,
    -> INDEX(A))
    -> ENGINE=InnoDB;
Query OK, 0 rows affected (0.44 sec)

mysql> CREATE TABLE Down
    -> (Lp SMALLINT UNSIGNED NOT NULL PRIMARY KEY,
    -> A CHAR(1),
    -> B CHAR(1),
    -> INDEX(A,B),
    -> FOREIGN KEY (A) REFERENCES Up (A)
    ->   ON DELETE RESTRICT
    ->   ON UPDATE SET NULL)
    -> ENGINE=InnoDB;
Query OK, 0 rows affected (0.34 sec)
```

W tabeli `Down` indeks założony jest na dwóch kolumnach. Jeśli tylko jedna z nich wchodzi do klucza obcego, musi to być pierwsza kolumna.

```
mysql> SELECT * FROM Up;
+----+---+
| Nr | A |
+----+---+
|  1 | a |
|  2 | b |
|  3 | b |
|  4 | w |
|  5 | z |
+----+---+
5 rows in set (0.09 sec)
```

```
mysql> SELECT * FROM Down ORDER BY Lp;
+----+------+------+
| Lp | A    | B    |
+----+------+------+
|  1 | z    | P    |
|  2 | w    | Q    |
|  3 | b    | R    |
|  4 | b    | S    |
|  5 | b    | T    |
|  6 | a    | U    |
+----+------+------+
6 rows in set (0.01 sec)
```

```
mysql> INSERT INTO Down VALUES (7,'c','X');
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint
fails (`klucze/down`, CONSTRAINT `down_ibfk_1` FOREIGN KEY (`A`) REFERENCES `up`
(`A`) ON UPDATE SET NULL)
```

Dodanie nowego wiersza do `Down` nie powiodło się, gdyż w `Up` nie ma wiersza zawierającego 'c'.

```
mysql> DELETE FROM Up WHERE A='w';
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint
fails (`klucze/down`, CONSTRAINT `down_ibfk_1` FOREIGN KEY (`A`) REFERENCES `up`
(`A`) ON UPDATE SET NULL)
```

Usunięcie wiersza z `Up` nie udało się, gdyż `Down` zawiera wiersz z 'w'.


```
mysql> UPDATE Up SET A='w' WHERE A='b';
Query OK, 2 rows affected (0.04 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

```
mysql> SELECT * FROM Up;
+----+---+
| Nr | A |
+----+---+
|  1 | a |
|  2 | w |
|  3 | w |
|  4 | w |
|  5 | z |
+----+---+
5 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM Down ORDER BY Lp;
+----+------+------+
| Lp | A    | B    |
+----+------+------+
|  1 | z    | P    |
|  2 | w    | Q    |
|  3 | NULL | R    |
|  4 | NULL | S    |
|  5 | NULL | T    |
|  6 | a    | U    |
+----+------+------+
6 rows in set (0.00 sec)
```

Pola tabeli `Down`, zawierające dotąd wartości 'b', zostały ustawione na `NULL`.

Przywracam tabelom `Up`, `Down` pierwotną zawartość i uruchamiam zapytanie

```
mysql> UPDATE Up SET A='w' WHERE Nr=3;
Query OK, 1 row affected (0.10 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql> SELECT * FROM Up;
+----+---+
| Nr | A |
+----+---+
|  1 | a |
|  2 | b |
|  3 | w |
|  4 | w |
|  5 | z |
+----+---+
5 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM Down ORDER BY Lp;
+----+------+------+
| Lp | A    | B    |
+----+------+------+
|  1 | z    | P    |
|  2 | w    | Q    |
|  3 | NULL | R    |
|  4 | NULL | S    |
|  5 | NULL | T    |
|  6 | a    | U    |
+----+------+------+
6 rows in set (0.00 sec)
```

W tabeli `Down` wszystkie ’b’ zostały zamienione na `NULL`, mimo iż w tabeli `Up` pozostała jedna wartość ’b’. Wynika to z zasady przeszukiwania indeksu: Warunek `ON UPDATE` odpala, gdy zostanie znaleziona choć jedna pasująca wartość, bez sprawdzania, czy są jakieś inne krotki o tej samej wartości zmienianego atrybutu.

Ograniczenia “zapętlone”

Czasami potrzebna jest sytuacja, w której pierwsza tabela odwołuje się do drugiej, druga zaś do pierwszej. Klucze obce można definiować tylko w odniesieniu do *istniejących* kolumn w *istniejących* tabelach. Jak obejść ten problem?

```
mysql> CREATE TABLE Jeden
    -> (A SMALLINT UNSIGNED NOT NULL PRIMARY KEY,
    ->  B CHAR(1) NOT NULL,
    ->  INDEX(B))
    -> ENGINE=InnoDB;
Query OK, 0 rows affected (1.24 sec)
mysql> CREATE TABLE Dwa
    -> (B CHAR(1) NOT NULL PRIMARY KEY,
    ->  A SMALLINT UNSIGNED NOT NULL,
    ->  INDEX(A),
    ->  FOREIGN KEY (A) REFERENCES Jeden (A)
    ->    ON DELETE RESTRICT
    ->    ON UPDATE CASCADE)
    -> ENGINE=InnoDB;
Query OK, 0 rows affected (1.25 sec)
```

```
mysql> ALTER TABLE Jeden
    -> ADD CONSTRAINT
    -> FOREIGN KEY (B) REFERENCES Dwa (B)
    ->   ON DELETE RESTRICT
    ->   ON UPDATE CASCADE;
Query OK, 0 rows affected (1.43 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Ale z tak zapętlonymi tabelami będzie sporo problemów ☺

Klucze obce — podsumowanie

- Tabela *podrzędna* bierze klucz obcy z tabeli *nadrzędnej*.
- Do tabeli *podrzędnej* nie można wstawić wiersza wskazującego na nieistniejącą wartość klucza obcego w tabeli *nadrzędnej*. Nie mozna dokonać też modyfikacji tabeli *podrzędnej*, która doprowadziłaby do takiej sytuacji. W ten sposób SQL/relacyjne bazy danych implementują więzy integralności referencyjnej.
- Zmiany dokonywane w tabeli *nadrzędnej* mogą "wzdłuż kluczy obcych" kaskadowo wywoływać zmiany w swoich tabelach *podrzędnych* (i w ewentualnych tabelach podrzędnych niższego rzędu). Zależy to od tego, jak klucz obcy został zdefiniowany w tabeli *podrzędnej*.

Dygresja - funkcja CASE

Funkcja CASE służy do kontroli przepływu. W SQL ma dwie postacie. Pierwsza postać:

```
CASE wartość
WHEN wartość-porównywana THEN wynik
WHEN wartość-Porównywana THEN wynik
...
END
```

Druga postać:

```
CASE
WHEN warunek THEN wynik
WHEN warunek THEN wynik
...
ELSE wynik
END
```

Klauzula ELSE jest opcjonalna.


```
mysql> SELECT * FROM Litery;
+--------+--------+
| litera | liczba |
+--------+--------+
| A      |      1 |
| B      |      2 |
| C      |      3 |
| D      |      4 |
| E      |      5 |
| F      |      6 |
| G      |      7 |
| H      |      8 |
| I      |     14 |
| J      |     15 |
| K      |      6 |
+--------+--------+
11 rows in set (0.01 sec)
```

```
mysql> SELECT CASE litera
    -> WHEN 'A' THEN 'A'
    -> WHEN 'B' THEN 'Be'
    -> WHEN 'C' THEN 'Ce'
    -> WHEN 'D' THEN 'De'
    -> WHEN 'E' THEN 'E'
    -> WHEN 'F' THEN 'Ef'
    -> WHEN 'G' THEN 'Gie'
    -> WHEN 'I' THEN 'I'
    -> WHEN 'J' THEN 'Jot'
    -> END AS 'Nazwa Litery'
    -> From Litery;
```

```
+--------------+
| Nazwa Litery |
+--------------+
| A            |
| Be           |
| Ce           |
| De           |
| E            |
| Ef           |
| Gie          |
| NULL         |
| I            |
| Jot          |
| NULL         |
+--------------+
11 rows in set (0.00 sec)
```

```
mysql> SELECT CASE
    -> WHEN litera < 'D' THEN 'niska'
    -> WHEN litera > 'H' THEN 'wysoka'
    -> ELSE 'inna'
    -> END AS 'Jaka Litera'
    -> FROM Litery;
```

```
+--------------+
| Jaka Litera  |
+--------------+
| niska        |
| niska        |
| niska        |
| inna         |
| inna         |
| inna         |
| inna         |
| inna         |
| wysoka       |
| wysoka       |
| wysoka       |
+--------------+
11 rows in set (0.00 sec)
```