Bazy danych

7. Klucze obce

Transakcje

P. F. Góra

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

2011

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;
```

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.

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

`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 ☺

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)
```

Transakcje

- Pojedynczy użytkownik — ochrona szczególnie wrażliwych fragmentów. *Transakcja wykonuje się albo w całości, albo wcale.* Jeżeli w trakcie wykonywania transakcji wystąpi jakiś błąd, całą sekwencję operacji można odwołać, przywracając bazę do stanu sprzed rozpoczęcia tej sekwencji.

- System wielodostępny

  1. Jak wyżej.

  2. Różne procesy klienckie odwołujące się do tych samych tabel nie mogą się ze sobą kłócić.

Zasady ACID (T. Hearder, A. Reuter, 1983)

A Atomicity — atomowość.
Transakcja jest niepodzielna, albo wszystko, albo nic.

C Consistency — spójność.
Transakcja nie może naruszać integralności danych (więzów narzuconych na dane w tabelach).

I Isolation — izolacja.
Jedna transakcja nie może widzieć wyników działania jakiejś innej, niezatwierdzonej transakcji. Można powiedzieć, że transkacja musi odbywać się tak, jakby żadna inna transakcja nie miała miejsca w tym samym czasie.

D Durability — trwałość.
Zmiany wprowadzone w transakcji muszą być trwałe, niezależnie od możliwych późniejszych błędów sprzętu lub oprogramowania.

Jak to robimy w SQL?

```
START TRANSACTION;
zapytanie_1;
zapytanie_2;
...
zapytanie_N;
COMMIT;
```

Zmiany zostają zatwierdzone

```
START TRANSACTION;
zapytanie_1;
zapytanie_2;
...
zapytanie_N;
ROLLBACK;
```

Zmiany zostają odwołane

Wielodostępność — co może pójść źle?

1. Niespójność odczytów — jedna transakcja może odczytać dane zmieniane przez drugą transakcję, chociaż transakcja ta nie zatwierdziła jeszcze zmian.

2. Niepowtarzalność odczytów — transakcja odczytuje dane, nieco później odczytuje je ponownie, a odczytane dane są inne, mimo iż transakcja odczytująca nie została jeszcze zatwierdzona.

3. Odczyty fantomowe — jedna tabela dodaje wiersz, druga transakcja aktualizuje wiersze. Nowy wiersz powinien być zaktualizowany, a nie jest.

Jak realizujemy izolację?

W celu zapewnienia izolacji w systemie wielodostępnym, transakcje blokują tabele (fragmenty tabel), które są im potrzebne. Najczęściej stosowane mechanizmy to:

2PL *Strict two-phase locking*: Każda transakcja zakłada blokadę na każdy rekord, który chce odczytać, przed dokonaniem odczytu. Blokady do odczytu mogą być współdzielone z innymi transakcjami. Każda transakcja zakłada też wyłączną blokadę na każdy fragment danych, który chce zapisać. Wszystkie blokady są utrzymywane aż do zakończenia transakcji.

OCC *Optimistic Concurrency Control*: Wiele transakcji mogą odczytywać i modyfikować fragment danych bez zakładania blokad. Transakcje zapamiętują historię dokonywanych odczytów i zapisów. Przed zatwierdzeniem transakcja sprawdza historię w celu wykrycia ewentualnych konfliktów z innymi

transakcjami. Jeśli jakieś konflikty zostaną wykryte, jedna z transakcji wywołujących konflikt zostaje odwołana.

Aby zminimalizować blokowanie (i opóźnienie innych transakcji), stosuje się mechanizm OCC. Działa on dobrze gdy konflikty są rzadkie, ale może jednak wygenerować duży koszt, jeżeli konflikty *nie* są rzadkie; w tych wypadkach mechanizm 2PL jest efektywnie szybszy.

Poziomy izolacji ANSI

Poziom izolacji	Niespójność odczytów	Niepowtarzalność odczytów	Odczyty fantomowe
Read uncommitted	OK	OK	OK
Read committed	NIE	OK	OK
Repeatable read	NIE	NIE	OK
Serializable	NIE	NIE	NIE

Serializowalność oznacza, że wynik sekwencji przeplatających się działań, wykonywanych przez zatwierdzane transakcje, musi ściśle odpowiadać sytuacji, w której wszystkie transakcje wykonywane są kolejno, jedna po zakończeniu drugiej.

Uwaga

- Instrukcje DDL (Data Description Language), czyli instrukcje tworzące i usuwające bazy oraz tworzące, usuwające i modyfukujące tabele nie są “transakcyjne” — nie można ich wycofać.

- W MySQL tabele, które chcemy zabezpieczać transakcjami, muszą być typu InnoDB.

Ryzyko związane z transakcjami

1. Długo działające transakcje blokują dostęp innych użytkowników do danych, na których działa transakcja, dopóki nie zostanie ona zatwierdzona lub odwołana.

2. Należy unikać transakcji wtedy, gdy wymagana jest interakcja z użytkownikiem — należy najpierw zebrać wszytskie dane, a dopiero potem rozpoczynać transakcję.

(B)lokowanie tabel

`LOCK TABLES` nazwa_tabeli `[READ | [LOW PRIORITY] WRITE];`

- Tryb `READ` — chcę czytać tabelę i w tym czasie nie zezwalam innym na zapis.
- Tryb `WRITE` — chcę zmieniać zawartość tabeli i w tym czasie nie zezwalam innym ani na zapis, ani na odczyt.

- Tryb `LOW PRIORITY WRITE` — pozwala innym wątkom na założenie blokady `READ`; w tym czasie wątek, który chce nałożyć blokadę `LOW PRIORITY WRITE`, musi czekać, aż tamten wątek zwolni blokadę.

`UNLOCK TABLES;` — zwalnia wszystkie zablokowane przez dany wątek tabele.

Tabel nie należy blokować zbyt długo lub niepotrzebnie.

Uwaga praktyczna: Aplikacja powinna *najpierw* zebrać wszystkie potrzebne dane od użytkownika, *później* inicjować transakcję lub blokować tabele.

