Instytut Sterowania i Systemów Informatycznych
Uniwersytet Zielonogórski

Bazy Danych

Ćwiczenie 13: transakcje w bazach danych

opracował: dr hab. inż. Artur Gramacki (a.gramacki@issi.uz.zgora.pl)

1. Uwagi wstępne

Praca wieloużytkownikowa

Sytuacje, gdy z bazy danych korzysta tylko jeden użytkownik występują w praktyce (w tzw. środowiskach produkcyjnych) bardzo rzadko. Zwykle mamy do czynienia z jednoczesnym dostępem wielu użytkowników do tej samej bazy danych. Oczywiście może się zdarzyć, że w tym samym momencie więcej niż jeden użytkownik będzie próbował modyfikować te same dane (te same rekordy w tej samej tabeli). W takiej sytuacji serwer bazy danych musi potrafić jakoś sensownie rozstrzygnąć ten konflikt. Niezależnie jednak od tego, jak fizycznie będzie to zrobione, poszczególni użytkownicy powinni mieć ciągle wrażenie, że mają wyłączny dostęp do zasobów serwera, oraz że nikt im tych zasobów nie blokuje.

Z pomocą przychodzi tutaj tzw. mechanizm blokowania. Jego istotą jest możliwość chwilowego zablokowania modyfikowanych przez jednego użytkownika danych, tak aby do czasu zakończenia modyfikacji nikt inny nie mógł zmienić tych danych. Oczywiście blokowanie takie powinno trwać możliwie jak najkrócej oraz w miarę możliwości odbywać się automatycznie, tak aby użytkownicy mieli zapewniony wystarczający komfort pracy i nie musieli zbyt długo czekać na zwolnienie pożądanych w danym momencie danych.

Transakcje

W kontekście systemów zarządzania bazami danych transakcję będziemy rozumieli jako jedną, lub w ogólności wiele, powiązanych ze sobą instrukcji, które muszą być zawsze traktowanie jako pojedyncza, *niepodzielna jednostka*. Oznacza to, że albo wszystkie zadania wchodzące w skład transakcji zostaną w całości oraz bezbłędnie wykonane, albo też cała transakcja musi zostać bezwarunkowo *anulowana* (wycofana).

Obsługa transakcji w bazach danych jest bardzo pożądaną cechą. Istnieje bowiem bardzo wiele sytuacji, gdzie aplikacje bazodanowe muszą wykonywać wiele powiązanych ze sobą czynności, które razem mają charakter transakcji. Typowym przykładem jest tutaj operacja przelewu bankowego. W dużym uproszczeniu składa się ona z następujących czynności:

- Złożenie zlecenia przez klienta A,
- Sprawdzenie stanu konta klienta A,

- Zablokowanie kwoty przelewu na koncie klienta A,
- Przesłanie kwoty przelewu na konto klienta B,
- Powiększenie o kwotę przelewu salda rachunku klienta B,
- Pomniejszenie o kwotę przelewu salda rachunku klienta A.

Oczywiście nikomu nie trzeba tłumaczyć jakie konsekwencje grożą, gdy z różnych powodów (np. awaria) wykonane zostaną tylko niektóre z powyższych czynności a pozostałych nie uda się wykonać. Nie może mieć również miejsca sytuacja, gdy np. w chwilę po wykonaniu czynności z punktu 3 (ale przed zakończeniem całości operacji) nastąpi wypłata z konta klienta A kwoty, która w połączeniu z kwotą właśnie co wykonywanego przelewu, wprowadza debet na koncie. Z powyższego wynika jasno, że pojęcie transakcji jest bardzo silnie związane z mechanizmem blokowania w kontekście pracy wieloużytkownikowej.

2. Mechanizmy składowania

Aby móc zapewnić obsługę omówionych powyżej mechanizmów, serwer bazodanowy musi posiadać odpowiednie ku temu możliwości. W MySQL-u rozwiązano to w taki sposób, że może on obsługiwać różne tzw. *mechanizmy składowania* (ang. *storage engines*). Dwa najbardziej popularne to *MyISAM* oraz *InnoDB*[1]. *MyISAM* na dzień dzisiejszy (styczeń 2016) nie obsługuje transakcji, drugi natomiast (*InnoDB*) zawiera taką obsługą. Ponadto mechanizm *InnoDB* obsługuje klucze obce, czego jak na razie nie potrafi mechanizm *MyISAM* (i nie wiadomo, czy kiedykolwiek taka funkcjonalność zostanie tam wprowadzona).

Mechanizmy nie wspierające transakcji, mimo iż pozornie wydają się mniej atrakcyjne, są nadal bardzo często używane. Ich niewątpliwą zaletą jest większa szybkość działania. Trzeba bowiem pamiętać, że obsługa transakcji wymaga od bazy danych większego zaangażowania zasobów sprzętowych i w związku z tym działają one po prostu wolniej. Zmiany w szybkości działania mogą być jednak zauważone w zasadzie tylko wówczas, gdy przechowujemy naprawdę duże ilości danych (liczone w dziesiątkach czy setkach tysięcy rekordów). Przy mniejszych ilościach danych w zasadzie trudno jest dostrzec jakieś istotniejsze różnice w szybkości działania. Dlatego też sugerujemy, aby w pierwszej kolejności spróbować użyć mechanizmu *InnoDB*, a dopiero gdy okaże się on zbyt wolny, migrować do mechanizmu nietransakcyjnego.

Niewątpliwą zaletą *InnoDB* jest również obsługa kluczy obcych. Jest to funkcjonalność na tyle atrakcyjna, że często może warto rozważyć stosowanie tego mechanizmy nawet wówczas, gdy jest on nieco wolniejszy od nieobsługującego kluczy obcych mechanizmu *MyISAM.*

3. Mechanizm składowania *MyISAM*

Mechanizm *MyISAM* nie wspiera transakcji, w zamian natomiast oferuje nieco większą szybkość działania. Poszczególne instrukcje traktowane są jak transakcje, więc nie ma problemu, gdy np. użytkownik aktualizuje kilka wierszy w tabeli i w tym samym czasie jakiś inny użytkownik robi to

[1] Pozostałe to: *MEMORY*, *CSV*, *ARCHIVE*, *BLACKHOLE*, *MERGE*, *FEDERATED*, *EXAMPLE*. Są one jednak zdecydowanie rzadziej używane, wiec nie będziemy ich tutaj omawiać.

samo. Serwer MySQL po prostu automatycznie *blokuje* wymagane zasoby. Nie można jednak grupować w transakcje kilku instrukcji. Mechanizm *MyISAM* dostarcza mechanizmy umożliwiające w miarę wygodną pracę w środowisku wieloużytkownikowym. Mamy mianowicie do dyspozycji polecenia do ręcznego *blokowania tabel*.

Użytkownik blokując dane tabele (wydając polecenie `LOCK`) powoduje, że do czasu zwolnienia blokady żaden inny użytkownik nie będzie mógł zmieniać w nich danych. Blokowanie odbywa się więc na życzenie użytkownika, więc użytkownik ten powinien również zadbać o to, aby tabele nie były zablokowane zbyt długo. Tabele zablokowane pozostają bowiem niedostępne dla innych użytkowników. W tym więc sensie omawiany tu mechanizm można uważać za dość niewygodny i mogący sprawiać pewne kłopoty.

Blokowanie może odbywać się w trybie `READ` oraz `WRITE`. Tryb `READ`. Gwarantuje, że nikt inny nie będzie mógł modyfikować danych, jednak nie ma przeciwwskazań, aby inni użytkownicy nie mogli w tym czasie odczytywać danych. Bardziej restrykcyjny jest tryb `WRITE`, gdyż do czasu zwolnienia blokady żaden inny użytkownik nie będzie mógł ani modyfikować danych, ani też ich oglądać. Można więc ten tryb uważać, za żądanie wyłączności dostępu do blokowanej tabeli.

Aby przeprowadzić odpowiednie ćwiczenia musimy mieć otwarte *dwie sesje* MySQL. Najlepiej w obu sesjach zalogować się na konto tego samego użytkownika. Poniżej pierwszy użytkownik blokuje dwie wybrane tabele (każdą w innym trybie) a drugi próbuje je odczytać. Następnie pierwszy użytkownik odblokowuje tabele a drugi ponownie stara się je odczytać.

Aby utworzyć tabelę *MyISAM* powinniśmy użyć następującej składni[2].

```
CREATE TABLE nazwa-tabeli (
...
) engine = MyISAM;
```

Aby upewnić się z jakiego typu tabelami mamy do czynienia możemy wykonać następujące polecenie[3]:

```
mysql> SELECT table_name, engine
    -> FROM information_schema.`TABLES`
    -> WHERE table_schema = 'lab' AND table_name LIKE 'studenci';

+------------+--------+
| table_name | engine |
+------------+--------+
| studenci   | MyISAM |
+------------+--------+
1 row in set (0.03 sec)
```

[2] W wersji 5.xserwera MySQL domyślnie wszystkie tabele tworzone są jako *InnoDB*. We wcześniejszych wersjach domyślnym typem był *MyISAM*. Oczywiście jawne wskazanie typu tabeli nie jest błędem.

[3] korzystamy tutaj ze specjalnej „systemowej" bazy danych o nazwie *information_schema*). W bazie tej przechowywane są różne informacje na temat innych baz. Jest to więc swego *rodzaju baza metadanych*. Pojawiła się ona dopiero w wersji 5 serwera MySQL. Szczegóły patrz dokumentacja serwera.

```
                ---------------                          ---------------
                |   SESJA A   |                          |   SESJA B   |
                ---------------                          ---------------

mysql> LOCK TABLES region READ, dept WRITE;
Query OK, 0 rows affected (0.00 sec)

                                              mysql> SELECT * FROM region
                                                  -> WHERE name LIKE 'Europe';
                                              +----+--------+
                                              | id | name   |
                                              +----+--------+
                                              |  5 | Europe |
                                              +----+--------+
                                              1 row in set (0.00 sec)


                                              mysql> SELECT * FROM dept
                                                  -> WHERE name LIKE 'Finance';
                                                 -------------------------
                                                 --- sesja zablokowana ---
                                                 -------------------------

mysql> UNLOCK TABLES;
Query OK, 0 rows affected (0.00 sec)


                                                 ----------------------
                                                 --- sesja 'puszcza' ---
                                                 ----------------------
                                                +----+---------+-----------+
                                                | id | name    | region_id |
                                                +----+---------+-----------+
                                                | 10 | Finance |         1 |
                                                +----+---------+-----------+
                                                1 row in set (3.46 sec)
```

Uwaga: blokowanie tabel za pomocą polecenie `LOCK` ma w zasadzie sens tylko w przypadku korzystania z mechanizmu *MyISAM* (lub innego nie wspierającego transakcji). Używanie polecenie `LOCK` dla tabel transakcyjnych (np. *InnoDB*) jest możliwe, jednak należy zachować ostrożność.

4. Mechanizm składowania *InnoDB* oraz transakcje

Działanie transakcji

Mechanizm *InnoDB* uważany jest za jeden z najszybszych na świecie mechanizmów obsługujących transakcje. Stworzony został przez fińską firmę *InnoBase Oy*. Firma ta została wykupiona przez potentata bazodanowego, czyli firmę *Oracle*.

Transakcję rozpoczynamy wydając polecenie `START TRANSACTION` lub jego synonimy `BEGIN WORK` oraz `BEGIN`. Zalecamy używanie tej pierwszej postaci, gdyż jest ona zgodna z normą SQL-99.

Aby utworzyć tabelę *InnoDB* powinniśmy użyć składni:

```
CREATE TABLE nazwa-tabeli (
...
```

```
) engine = InnoDB;
```

MySQL może pracować w dwóch trybach. Pierwszy, zwany *autocommit*, powoduje, że każde wydawane polecenie jest traktowana jako transakcja. Oznacza to, że wynik działania tego polecenia jest automatycznie zatwierdzany w bazie danych i nie ma możliwości jego wycofania (ang. *rollback*). Można więc uważać ,że każde polecenie poprzedzone jest niejawnym zapoczątkowaniem transakcji oraz zakończone niejawnym jej zatwierdzeniem, czyli:

```
START TRANSACTION;
   instrukcja SQL
COMMIT;
```

W tym trybie mechanizm *InnoDB* pracuje w sposób podobny do mechanizmu *MyISAM*, gdyż użytkownik nie ma możliwości grupowania w jedną transakcję więcej niż jednego polecenia SQL. Aby wyłączyć działanie trybu *autocommit* należy wydać polecenie:

```
SET AUTOCOMMIT = 0;
```

Jeżeli zależy nam na bardziej trwałym ustawieniu trybu zatwierdzania zmian, możemy umieścić odpowiedni wpis w pliku konfiguracyjnym serwera MySQL *my.ini* lub *my.cnf*. W sekcji *[mysqld]* należy umieścić wpis

```
init_connect = 'SET AUTOCOMMIT=0')
```

Wówczas można samodzielnie wydawać polecenia `START TRANSACTION`, `COMMIT` oraz ROLLBACK}. Pierwsze z nich rozpoczyna nową transakcję[4], drugie zatwierdza wszystkie zmiany wprowadzone w ramach bieżącej transakcji. Ostatnie natomiast polecenie wycofuje wszystkie zmiany wprowadzone w ramach bieżącej transakcji.

Po wykonaniu instrukcji

```
SET AUTOCOMMIT = 0
```

MySQL zakłada, że od tej pory wszystkie transakcje będą przez użytkownika jawnie kończone poleceniem `COMMIT`. Należy o tym zawsze pamiętać, bo gdy o tym zapomnimy i zakończymy sesję (np. zamykając aplikację kliencką), MySQL automatycznie wycofa wszystkie zmiany wprowadzone w czasie całej sesji!

Ponowny powrót do trybu *autocommit* spowoduje oczywiście polecenie:

```
SET AUTOCOMMIT = 1;
```

---

[4] Standard języka SQL nie definiuje instrukcji do rozpoczynania transakcji. Po prostu zakłada się, że transakcja rozpoczyna się automatycznie w momencie wykonania pierwszej instrukcji SQL i kończy po wykonaniu polecenie `COMMIT` lub `ROLLBACK`. Mimo to wielu producentów serwerów baz danych implementuje instrukcje do jawnego zapoczątkowywania transakcji

W każdej chwili, odpowiednim poleceniem, możliwe jest sprawdzenie jaki tryb zatwierdzania transakcji obowiązuje w danym momencie:

```
mysql> SELECT @@AUTOCOMMIT;
+--------------+
| @@AUTOCOMMIT |
+--------------+
|            1 |
+--------------+
1 row in set (0.00 sec)
```

Zwróćmy jeszcze uwagę, że pracując z tabelami nietransakcyjnymi (zwykle *MyISAM*) lub też z tabelami *InnoDB* ale z ustawionym trybem *autocommit*, możliwe jest wydawanie poleceń `START TRANSACTION`, `COMMIT` oraz `ROLLBACK`, jednak nie niosą one ze sobą żadnych skutków. Można się zastanowić, dlaczego twórcy serwera MySQL dali nam taką możliwość. Wydaje się bowiem, że sensowniej byłoby generować jakiś komunikat ostrzegawczy. Pewnym wytłumaczeniem takiego stanu rzeczy może być potrzeba zapewnienia przenaszalności aplikacji dla MySQL w ramach różnych mechanizmów składowania.

Zobaczmy jak w praktyce działa mechanizm transakcji. Wykonamy kilka poleceń zmieniających dane a następnie spróbujemy wycofać wszystkie wprowadzone zmiany. Aby przeprowadzić odpowiednie ćwiczenia musimy mieć otwarte dwie sesje MySQL - podobnie jak to było przy eksperymentach z blokowaniem tabel.

```
               ---------------                              ---------------
               |   SESJA A   |                              |   SESJA B   |
               ---------------                              ---------------
mysql> SET AUTOCOMMIT = 0;

mysql> START TRANSACTION;

                                          mysql> SELECT * FROM test;
                                          +------+
                                          | id   |
                                          +------+
                                          |    1 |
                                          +------+

mysql> UPDATE test SET id = 2;

                                          mysql> SELECT * FROM test;
                                          +------+
                                          | id   |
                                          +------+
                                          |    1 |
                                          +------+

mysql> DELETE FROM test;

                                          mysql> SELECT * FROM test;
                                          +------+
                                          | id   |
                                          +------+
                                          |    1 |
                                          +------+

                                          mysql> UPDATE test SET id = 2;
```

```
                                          -------------------------
                                          --- sesja zablokowana ---
                                          -------------------------
mysql> ROLLBACK;

                                          ------------------------
                                          --- sesja 'puszcza' ---
                                          ------------------------
                                        mysql>  UPDATE test SET id = 2;
                                        Query OK, 1 row affected (2.13 sec)
                                        Rows matched: 1  Changed: 1  Warnings: 0

                                        mysql> SELECT * FROM test;
                                        +------+
                                        | id   |
                                        +------+
                                        |    2 |
                                        +------+
```

Uwaga: MySQL stara się, jeżeli jest to tylko możliwe, samodzielnie odblokowywać zablokowane sesje. Gdy więc po wydaniu w drugiej sesji polecenia

```
UPDATE test SET id = 2;
```

„dostatecznie długo" nic się nie dzieje w sesji pierwszej, serwer odblokowuje „wiszącą" sesję i wycofuje przy okazji transakcję, która tą sesję zablokowała. Wyświetlany jest przy tym komunikat:

```
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction.
```

Domyślnie czas oczekiwania na odblokowanie wynosi 10 sekund. Można się o tym przekonać wydając następujące polecenie:

```
mysql> SHOW VARIABLES WHERE variable_name LIKE 'connect_timeout';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| connect_timeout | 5     |
+-----------------+-------+
```

Zakleszczanie

Do tzw. zakleszczenia} dochodzi wówczas, gdy dwie różne sesje (dwaj różni użytkownicy) próbują zmieniać sobie nawzajem te same dane. Wówczas obie sesje zostają zablokowane i użytkownicy nie mogą pracować. Serwer MySQL próbuje wówczas, podobnie jak to było opisane powyżej, odblokować zablokowane sesje. Skutkuje to tym, że w jednej z sesji (użytkownik nie ma na to wpływu) serwer automatycznie wycofuje transakcje.

Przeprowadzamy następujące ćwiczenie:

```
mysql> DROP TABLE test;
mysql> CREATE TABLE test (id INT PRIMARY KEY) ENGINE = InnoDB;
mysql> INSERT INTO test VALUES (1), (2);
mysql> COMMIT;
mysql> SELECT * FROM test;
+----+
```

```
| id |
+----+
|  1 |
|  2 |
+----+
2 rows in set (0.00 sec)
```

SESJA A	SESJA B
`mysql> SET AUTOCOMMIT = 0;`	`mysql> SET AUTOCOMMIT = 0;`
`mysql> START TRANSACTION;`	`mysql> START TRANSACTION;`
`mysql> UPDATE test SET id=10 WHERE id=1;`	
	`mysql> UPDATE test SET id=20 WHERE id=2;`
`mysql> UPDATE test SET id=20 WHERE id=2;`	
`--- sesja zablokowana ---`	
	`mysql> UPDATE test SET id=10 WHERE id=1;`
`--- sesja 'puszcza' ---`	
`Query OK, 1 row affected (21.44 sec)`	
`Rows matched: 1  Changed: 1  Warnings: 0`	
	`mysql> UPDATE test SET id=10 WHERE id=1;`
	`ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction`
	`mysql>`
`mysql> SELECT * FROM test;`	
`+----+`	
`| id |`	
`+----+`	
`| 10 |`	
`| 20 |`	
`+----+`	
	`mysql> SELECT * FROM test;`
	`+----+`
	`| id |`
	`+----+`
	`|  1 |`
	`|  2 |`
	`+----+`
`mysql> COMMIT;`	
`Query OK, 0 rows affected (0.00 sec)`	
	`mysql> SELECT * FROM test;`
	`+----+`
	`| id |`
	`+----+`
	`|  1 |`
	`|  2 |`
	`+----+`
	`mysql> COMMIT;`

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

                                        mysql> SELECT * FROM test;
                                        +----+
                                        | id |
                                        +----+
                                        | 10 |
                                        | 20 |
                                        +----+
```

Zwróćmy uwagę, że tabela używana w tym ćwiczeniu miała założony klucz główny. Powtórz ćwiczenie dla tabeli bez klucza głównego. Zaobserwuj różnice i postaraj się wyjaśnić zachowanie serwera MySQL. Problem ten jest dokładnie wyjaśniony w dokumentacji serwera MySQL i dlatego postaraj się dokładnie „doczytać" te informacje.

Blokowanie wierszy

Mechanizm *InnoDB* również pozwala na jawne blokowanie danych. Inaczej jednak niż w mechanizmie *MyISAM*, blokować można dane tylko na poziomie wierszy a nie całych tabel. Choć oczywiście możliwe jest zablokowanie całej tabeli, poprzez zablokowanie wszystkich jej wierszy. Zobaczmy, jak to wygląda w praktyce (zwróćmy uwagę na klauzulę `FOR UPDATE`):

```
               ---------------                         ---------------
               |   SESJA A   |                         |   SESJA B   |
               ---------------                         ---------------

mysql> SELECT id FROM test;
+----+
| id |
+----+
|  1 |
|  2 |
+----+

mysql> SELECT id FROM test
    -> WHERE id = 1 FOR UPDATE;
+----+
| id |
+----+
|  1 |
+----+

                                       mysql> UPDATE test SET id=20 WHERE id=2;
                                       Query OK, 1 row affected (0.00 sec)
                                       Rows matched: 1  Changed: 1  Warnings: 0

                                       mysql> UPDATE test SET id=10 WHERE id=1;
                                       -------------------------
                                       --- sesja zablokowana ---
                                       -------------------------

mysql> ROLLBACK;
                                       ------------------------
                                       --- sesja 'puszcza' ---
                                       ------------------------
                                       Query OK, 1 row affected (8.89 sec)
                                       Rows matched: 1  Changed: 1  Warnings: 0

                                       mysql> COMMIT;
                                       Query OK, 0 rows affected (0.03 sec)
```

```
mysql> SELECT id FROM test;
+----+
| id |
+----+
|  1 |
|  2 |
+----+
2 rows in set (0.01 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT id FROM test;
+----+
| id |
+----+
| 10 |
| 20 |
+----+
2 rows in set (0.00 sec)
```

Uwaga: w praktyce powinniśmy blokować tylko tyle wierszy, ile jest rzeczywiście konieczne. Zablokowanie zbyt dużej ich liczby może utrudniać pracę innym użytkownikom.

Zwróćmy uwagę, że gdy wykonaliśmy polecenie `COMMIT` w sesji B, sesja A „nie widzi" jeszcze wprowadzonych zmian. Zmiany stają się widoczne dopiero po wydaniu polecenia `SELECT` z poziomu sesji A. Wyjaśnij to pozornie dziwne zachowanie!

Używając polecenia `SELECT` do blokowania wierszy, można powiedzieć, że blokujemy je jakby na przyszłość -- samo bowiem polecenie `SELECT` nie wprowadza przecież żadnych zmian w zgromadzonych danych. Blokada zostaje zdjęta po wykonaniu polecenie `COMMIT` lub też `ROLLBACK`.