

Poziomy izolacji transakcji i replikacje w systemie bazodanowym MySQL

Bartosz Chodorowski <166142@student.pwr.wroc.pl>

Wrocław, 13 grudnia 2009

1 Poziomy izolacji

1.1 Treść zadania

Opisz poziomy izolacji transakcji dla dowolnie wybranego systemu bazy danych. Napisz odpowiednie przykłady pokazujące działanie transakcji na każdym z tych poziomów. Napisz to wszystko w formie sprawozdania.

1.2 Wstęp teoretyczny

Transakcją nazywamy zbiór operacji na bazie danych, które stanowią w istocie pewną całość i jako takie powinny być wykonane wszystkie lub żadna z nich. Warunki jakie powinny spełniać transakcje bardziej szczegółowo opisują zasady ACID (Atomicity, Consistency, Isolation, Durability – Atomowość, Spójność, Izolacja, Trwałość).

Przykładem transakcji może być transakcja bankowa jaką jest przelew. Muszą tu zostać dokonane 2 operacje – zabranie pieniędzy z jednego konta oraz dopisanie ich do drugiego. W przypadku niepowodzenia żadna z tych operacji nie powinna być zatwierdzona, gdyż zajęcie tylko jednej powodowałoby nieprawidłowości w bazie danych (pojawienie się lub zniknięcie pieniędzy).

Transakcja składa się zawsze z 3 etapów:

- rozpoczęcia
- wykonania
- zamknięcia

W systemach bazodanowych istotne jest, aby transakcja trwała jak najkrócej, ponieważ równolegle może być dokonywanych wiele transakcji i część operacji musi zostać wykonana w pewnej kolejności. Każdy etap transakcji jest logowany, dzięki czemu w razie awarii systemu (dzięki zawartości logów), można odtworzyć stan bazy danych sprzed transakcji, która nie została zamknięta.

1.3 Transakcje a MySQL

MySQL korzysta z zaawansowanego mechanizmu składowania danych zwanego **InnoDB**. Umożliwia on między innymi wykonywanie transakcji. Zgodnie ze standardem SQL:1992, InnoDB oferuje cztery poziomy transakcji:

- **READ UNCOMMITTED** – dopuszcza czytanie przez transakcje danych nie zatwierdzonych, tj. danych które zostały zmienione przez transakcję jeszcze nie zatwierdzoną
- **READ COMMITTED** – wprowadza zakaz czytania danych z nie zatwierdzonych transakcji, a więc czytać można tylko dane zatwierdzone
- **REPEATABLE READ** – wprowadza zakaz zapisywania w transakcjach nie zatwierdzonych. Jeśli więc transakcja nie zatwierdzona przeczytała jakąś daną, to dana ta może być tylko czytana przez inną transakcję
- **SERIALIZABLE** – za niekonfliktowe uznaje się tylko te operacje, gdy działanie jednej z nich nie powoduje zmiany zbioru danych, na których działa druga

Domyślnym poziomem transakcji jest **REPEATABLE READ**.

1.4 Przykłady

Stworzono następującą tabelę z przykładowymi danymi:

```
1 mysql> CREATE TABLE Procesor(Nazwa VARCHAR(50), Cena INT, Stan INT)
2 -> ENGINE = InnoDB;
3 Query OK, 0 rows affected (0.05 sec)
4
5 mysql> INSERT INTO Procesor VALUES('200MMX', 320, 20);
6 Query OK, 1 row affected (0.00 sec)
7
8 mysql> INSERT INTO Procesor VALUES('233MMX', 370, 50);
9 Query OK, 1 row affected (0.00 sec)
```

Napisano następujące przykłady, które obrazują wykorzystanie wszystkich czterech poziomów izolacji:

```
1 DELIMITER $$
2
3 SET autocommit=0;
4
5 #####
6
7 DROP PROCEDURE IF EXISTS przyklad1_1;
8 CREATE PROCEDURE przyklad1_1()
9 BEGIN
10     SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
11     START TRANSACTION;
12     UPDATE Procesor set Cena = 300 WHERE Nazwa = '200MMX';
13     SET @a = (SELECT SLEEP(4));
14     ROLLBACK;
15 END;
16
17 DROP PROCEDURE IF EXISTS przyklad1_2;
18 CREATE PROCEDURE przyklad1_2()
19 BEGIN
20     SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
21     START TRANSACTION;
```

```

22         SET @a = (SELECT SLEEP(2));
23         SELECT * FROM Procesor where Nazwa = '200MMX';
24         COMMIT;
25 END;
26
27 #####
28
29 DROP PROCEDURE IF EXISTS przyklad2_1;
30 CREATE PROCEDURE przyklad2_1()
31 BEGIN
32     SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
33     START TRANSACTION;
34     SELECT sum(Cena*Stan) FROM Procesor WHERE Nazwa = '200MMX';
35     SET @a = (SELECT SLEEP(4));
36     SELECT sum(Cena*Stan) FROM Procesor WHERE Nazwa = '200MMX';
37     COMMIT;
38 END;
39
40 DROP PROCEDURE IF EXISTS przyklad2_2;
41 CREATE PROCEDURE przyklad2_2()
42 BEGIN
43     SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
44     START TRANSACTION;
45     SET @a = (SELECT SLEEP(2));
46     UPDATE Procesor SET Cena = 310 WHERE Nazwa = '200MMX';
47     SET @a = (SELECT SLEEP(4));
48     COMMIT;
49 END;
50
51 #####
52
53 DROP PROCEDURE IF EXISTS przyklad3_1;
54 CREATE PROCEDURE przyklad3_1()
55 BEGIN
56     SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
57     START TRANSACTION;
58     SELECT sum(Cena*Stan) FROM Procesor WHERE Nazwa = '200MMX';
59     SET @a = (SELECT SLEEP(4));
60     SELECT sum(Cena*Stan) FROM Procesor WHERE Nazwa = '200MMX';
61     COMMIT;
62 END;
63
64 DROP PROCEDURE IF EXISTS przyklad3_2;
65 CREATE PROCEDURE przyklad3_2()
66 BEGIN
67     SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
68     START TRANSACTION;
69     SET @a = (SELECT SLEEP(2));
70     UPDATE Procesor SET Cena = 310 WHERE Nazwa = '200MMX';
71     SET @a = (SELECT SLEEP(4));
72     COMMIT;
73 END;
74
75 #####
76
77 DROP PROCEDURE IF EXISTS przyklad4_1;
78 CREATE PROCEDURE przyklad4_1()
79 BEGIN
80     SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
81     START TRANSACTION;
82     SELECT sum(Cena*Stan) FROM Procesor WHERE Nazwa = '200MMX';
83     SET @a = (SELECT SLEEP(4));

```

```

84      SELECT sum(Cena*Stan) FROM Procesor WHERE Nazwa = '200MMX';
85      COMMIT;
86 END;
87
88 DROP PROCEDURE IF EXISTS przyklad4_2;
89 CREATE PROCEDURE przyklad4_2()
90 BEGIN
91     SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
92     START TRANSACTION;
93     SET @a = (SELECT SLEEP(2));
94     INSERT INTO Procesor VALUES('200MMX', 250, 10);
95     COMMIT;
96 END;
97
98 #####
99
100 $$
101
102 DELIMITER ;

```

Przykład pierwszy pokazuje błędnie użyty poziom izolacji `READ UNCOMMITTED`. Jednoczesne uruchomienie procedur `przyklad1_1` oraz `przyklad1_2` powoduje wypisanie niepoprawnej informacji o cenie (300 zamiast 320). W tej sytuacji powinno się użyć kolejnego poziomu izolacji.

Przykład drugi pokazuje, w jakiej sytuacji zawodzi `READ COMMITED`. Równoczesne uruchomienie procedur `przyklad2_1` oraz `przyklad2_2` powoduje wypisanie różnych wartości. Zastosowanie trybu `REPEATABLE READ` (procedury `przyklad3_1` oraz `przyklad3_2`) powoduje wypisanie tej samej wartości.

Chciaż przykład 3 działa już dobrze, przy wstawianiu nowej wartości do bazy należy użyć ostatniego poziomu – `SERIALIZABLE`, aby otrzymać takie samo wyjście dla obu wywołań instrukcji `SELECT`.

2 Replikacje

2.1 Treść zadania

Opisz rodzaje replikacji dla dowolnie wybranego systemu bazy danych. Skonfiguruj jeden z nich.

2.2 Wstęp teoretyczny

Replikacja danych to proces powielania informacji pomiędzy różnymi serwerami baz danych.

2.3 Replikacja a MySQL

Baza danych MySQL oferuje asynchroniczną replikację między serwerem nadrzędnym a podrzędnym. Serwer nadrzędny prowadzi dziennik aktualizacji, a serwer podrzędny odczytuje dziennik i kolejno stosuje aktualizacje.

W MySQL replikacja może opierać się na trzech sposobach: na poziomie instrukcji, wierszy lub trybu mieszanego. Metoda oparta na instrukcjach polega na tym, że serwer nadrzędny rejestruje każdą instrukcję

modyfikującą dane np. INSERT, DELETE, która następnie wykonywana jest przez serwer podrzędny. W metodzie opartej na wierszach rejestrowana jest każda zmiana wiersza tabeli. W trybie mieszanym wykorzystywane są obie metody - serwer nadrzędny decyduje który tryb zastosować dla każdego zapytania.

Początkowo stosowano replikację opartą na wierszach lecz od wersji 5.0 wyparto ją dużo wydajniejszą replikacją na poziomie instrukcji. Zalety replikacji opartej na instrukcjach:

- wymaga przesłania mniejszej ilości danych
- ogranicza rozmiar dziennika aktualizacji
- nie musi uwzględniać formatu wiersza

2.4 Konfiguracja serwera nadrzędnego

W pliku `/etc/mysql/my.cnf` należy ustawić następujące opcje:

```
1 [mysqld]
2 log-bin = mysql-bin
3 server-id = 1
4 max_binlog_size = 20M
5 binlog-do-db = blog
6
7 #bind-address = 127.0.0.1
8 #skip-networking
```

Opcja `binlog-do-db` oznacza bazę danych, którą chcemy replikować.

Następnie w programie `mysql` będąc zalogowanym jako `root` wydajemy polecenie:

```
1 mysql> SHOW MASTER STATUS\G
2 ***** 1. row *****
3 File: mysql-bin.000012
4 Position: 463666
5 Binlog_Do_DB: blog
6 Binlog_Ignore_DB:
7
8 mysql> GRANT REPLICATION SLAVE ON *.*
9 -> TO 'uzytkownik-replikacji@ip-serwera-nadzelnego'
10 -> IDENTIFIED BY 'haslo-uzytkownika-replikacji';
```

Najbardziej interesujące dane to `File` i `Position`: są to współrzędne replikacji, które pozwolą serwerowi podrzelnemu replikować dane właśnie od tego momentu – należy je zapisać ponieważ przydydadzą się przy konfigurowaniu serwera podrzelnego.

2.5 Konfiguracja serwera podrzelnego

W pliku `/etc/mysql/my.cnf` należy ustawić:

```
1 [mysqld]
2 server-id = 2
```

Następnie należy wydać polecenie:

```
1 mysql> CHANGE MASTER TO
2 -> MASTER_HOST='ip-serwera-nadrzédnego',
3 -> MASTER_USER='użytkownik-replikacji',
4 -> MASTER_PASSWORD='haslo-uzytkownika-replikacji',
5 -> MASTER_LOG_FILE='nazwa-pliku-odczytana-z-File',
6 -> MASTER_LOG_POSITON='pozycja-odczytana-z-Position';
```

Ostatnim krokiem jest uruchomienie replikacji:

```
1 mysql> START SLAVE;
```