Санкт-Петербургский Государственный Политехнический
Университет Петра Великого
Институт Компьютерных Наук и Технологий

Кафедра Компьютерных Систем и Программных Технологий

Отчёт по лабораторной работе
Дисциплина: Базы данных
Тема: Изучение работы транзакций

Выполнил студент группы 43501/3 _____ Круминьш Д.В.
(подпись)

Преподаватель _____ Мяснов А.В.
(подпись)

Санкт-Петербург
2016

Программа работы

1. Изучить основные принципы работы транзакций.
2. Провести эксперименты по запуску, подтверждению и откату транзакций.
3. Разобраться с уровнями изоляции транзакций в Firebird.
4. Спланировать и провести эксперименты, показывающие основные возможности транзакций с различным уровнем изоляции.
5. Продемонстрировать результаты преподавателю, ответить на контрольные вопросы.

Работа проводится в IBExpert. Для проведения экспериментов параллельно запускается несколько сессий связи с БД, в каждой сессии настраивается уровень изоляции транзакций. Выполняются конкурентные операции чтения/изменения данных в различных сессиях, а том числе приводящие к конфликтам.

Ход работы

1 Изучить основные принципы работы транзакций.

Транзакция - это неделимая, с точки зрения воздействия на СУБД, последовательность операций манипулирования данными. Для пользователя транзакция выполняется по принципу "все или ничего т.е. либо транзакция выполняется целиком и переводит базу данных из одного целостного состояния в другое целостное состояние, либо, если по каким-либо причинам, одно из действий транзакции невыполнимо, или произошло какое-либо нарушение работы системы, база данных возвращается в исходное состояние, которое было до начала транзакции (происходит откат транзакции). Транзакция обладает четырьмя важными свойствами:

- **Атомарность**. Транзакция выполняется как атомарная операция - либо выполняется вся транзакция целиком, либо она целиком не выполняется.
- **Согласованность**. Транзакция переводит базу данных из одного согласованного (целостного) состояния в другое согласованное (целостное) состояние. Внутри транзакции согласованность базы данных может нарушаться.
- **Изоляция**. Транзакции разных пользователей не должны мешать друг другу (например, как если бы они выполнялись строго по очереди).
- **Долговечность**. Если транзакция выполнена, то результаты ее работы должны сохраниться в базе данных, даже если в следующий момент произойдет сбой системы.

Транзакция обычно начинается автоматически с момента присоединения пользователя к СУБД и продолжается до тех пор, пока не произойдет одно из следующих событий:

- Подана команда **COMMIT** (зафиксировать транзакцию).
- Подана команда **ROLLBACK** (откатить транзакцию).
- Произошло отсоединение пользователя от СУБД.
- Произошел сбой системы.

2 Провести эксперименты по запуску, подтверждению и откату транзакций.

Были проведены тесты по запуску, подтверждению и откату транзакций.

```
SQL> create table temp_table(someId integer not null primary key);
SQL> commit;
SQL> insert into temp_table values(1);
SQL> commit;
SQL> rollback;
SQL> select * from temp_table;

      SOMEID
============
           1

SQL> insert into temp_table values(2);
SQL> select * from temp_table;

      SOMEID
============
           1
           2

SQL> rollback;
SQL> select * from temp_table;

      SOMEID
============
           1

SQL> insert into temp_table values(2);
SQL> savepoint one;
SQL> delete from temp_table;
SQL> insert into temp_table values(3);
SQL> select * from temp_table;
SQL> rollback to one;
SQL> select * from temp_table;

      SOMEID
============
           1
           2
```

Листинг 1: Лог работы в isql

Была создана таблица temp_table с одним полем someId, в это поле сперва было добавлено значение 1, rollback шедший за commit не сработал и значение осталось в таблице.

При добавлении значения 2 хоть с помощью запроса select оно было показано в таблице, но после команды rollback, оно оттуда удалилось.

Также имеется возможность добавлять точки сохранения для rollback, так данные таблицы были удалены и добавлена новая строка, отличная от предыдущих, но после вызова rollback на savepoint, содержимое таблицы вернулось к исходному состоянию.

3 Разобраться с уровнями изоляции транзакций в Firebird.

Уровень изолированности транзакции определяет, какие изменения, сделанные в других транзакциях, будут видны в данной транзакции. Каждая транзакция имеет свой уровень изоляции, который устанавливается при ее запуске и остается неизменным в течение всей ее жизни. Транзакции в Firebird могут иметь 3 основных возможных уровня изоляции: READ COMMITTED, SNAPSHOT и SNAPSHOT TABLE STABILITY. Каждый из этих трех уровней изоляции определяет правила видимости тех действий, которые выполняются другими транзакциями. Рассмотрим уровни изоляции более подробно.

- **READ COMMITTED**. Буквально переводится как "читать подтвержденные данные однако это не совсем (точнее, не всегда) так. Уровень изоляции READ COMMITTED используется, когда мы желаем видеть все подтвержденные результаты параллельно выполняющихся (т. е. в рамках других транзакций) действий. Этот уровень изоляции гарантирует, что мы НЕ сможем прочитать неподтвержденные данные, измененные в других транзакциях, и делает ВОЗМОЖНЫМ чтение подтвержденных данных.
- **SNAPSHOT**. Этот уровень изоляции используется для создания "моментального"снимка базы данных. Все операции чтения данных, выполняемые в рамках транзакции с уровнем изоляции SNAPSHOT, будут видеть только состояние базы данных на момент начала запуска транзакции. Все изменения, сделанные в параллельных подтвержденных (и разумеется, неподтвержденных) транзакциях, не видны в этой транзакции. В то же время SNAPSHOT не блокирует данные, которые он не изменяет.
- **SNAPSHOT TABLE STABILITY**. Это уровень изоляции также создает "моментальный"снимок базы данных, но одновременно блокирует на запись данные, задействованные в операциях, выполняемые данной транзакцией. Это означает, что если транзакция SNAPSHOT TABLE STABILITY изменила данные в какой-нибудь таблице, то после этого данные в этой таблице уже не могут быть изменены в других параллельных транзакциях. Кроме того, транзакции с уровнем изоляции SNAPSHOT TABLE STABILITY не могут получить доступ к таблице, если данные в них уже изменяются в контексте других транзакций.

4 Спланировать и провести эксперименты, показывающие основные возможности транзакций с различным уровнем изоляции.

Snapshot

```
1 SQL> insert into temp_table values(3);
2 SQL> commit;
3 SQL> select * from temp_table;
4
5        SOMEID
6 ============
7             1
8             2
9             3
```

Листинг 2: Терминал 1. Snapshot

```
SQL> set transaction snapshot;
Commit current transaction (y/n)?n
Rolling back work.
SQL> select * from temp_table;

      SOMEID
============
           1
           2
```

Листинг 3: Терминал 2. Snapshot

Как видно из лога терминала 2, новой записи в таблице не появилось, тогда как в терминале 1, новое значение успешно вывелось.

Snapshot table stability

```
SQL> select * from temp_table;

      SOMEID
============
           3
           4
           1
           2

SQL> UPDATE TEMP_TABLE SET SOMEID=5 WHERE SOMEID=1;
SQL> select * from temp_table;

      SOMEID
============
           3
           4
           5
           2

SQL> COMMIT;
SQL> UPDATE TEMP_TABLE SET SOMEID=1 WHERE SOMEID=5;
SQL> COMMIT;
SQL> select * from temp_table;

      SOMEID
============
           3
           4
           1
           2
```

Листинг 4: Терминал 1. Snapshot table stability

```
SQL> set transaction isolation level snapshot table stability;
Commit current transaction (y/n)?n
Rolling back work.
```

```
SQL> select * from temp_table;

      SOMEID
============
           3
           4
           1
           2

SQL> UPDATE TEMP_TABLE SET SOMEID=6 WHERE SOMEID=1;
Statement failed, SQLSTATE = 40001
deadlock
-update conflicts with concurrent update
-concurrent transaction number is 64
SQL> UPDATE TEMP_TABLE SET SOMEID=6 WHERE SOMEID=1;
SQL> select * from temp_table;

      SOMEID
============
           3
           4
           6
           2
```

Листинг 5: Терминал 2. Snapshot table stability

Видим, что результаты соответствуют ожиданиям. Изменяемая в транзакции с уровнем изоляции snapshot table stability целиком блокируется для всех остальных транзакций до окончания выполнения транзакции.

Read committed

```
SQL> SELECT * FROM TEMP_TABLE;

      SOMEID
============
           3
           4
           5
           1
           2

SQL> insert into temp_table values(6);
SQL> commit;
```

Листинг 6: Терминал 1. Read committed

```
SQL> SELECT * FROM TEMP_TABLE;

      SOMEID
============
           3
           4
```

```
            5
            1
            2

SQL> SET TRANSACTION READ COMMITTED;
Commit current transaction (y/n)?n
Rolling back work.
SQL> SELECT * FROM TEMP_TABLE;

      SOMEID
============
            3
            4
            5
            1
            2
            6
```

Листинг 7: Терминал 2. Read committed

Как видно из логов, в терминале 1 произошла вставка данных в таблицу. Терминал 2 увидел изменения сразу-же после подтверждения транзакции в терминале 1.

Вывод

Механизм транзакции незаменим при работе с крупными базами данных. Он позволяет поддерживать целостность данных при параллельной работе нескольких клиентов с базой данных. Достоинства транзакций:

1. Механизм транзакций позволяет обеспечить логическую целостность данных в БД. Другими словами, транзакции – логические единицы работы, после выполнения которых БД остается в целостном состоянии.

2. Транзакции являются единицами восстановления данных. Восстанавливаясь после сбоев, система ликвидирует следы транзакций, не успевших успешно завершиться в результате программного или аппаратного сбоя.

3. Механизм транзакций обеспечивает правильность работы в многопользовательских системах при параллельном обращении нескольких пользователей к одним и тем же данным.

Недостатки и проблемы: При параллельном выполнении транзакций возможны следующие проблемы:

1. При одновременном изменении одного блока данных разными транзакциями одно из изменений теряется;

2. При повторном чтении в рамках одной транзакции ранее прочитанные данные оказываются изменёнными(другой транзакцией)

3. Если транзакция не завершается в момент когда она должна быть завершена, то соответственно записи блокируются на лишнее время, другой поток не может их изменить.