

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчёт о лабораторной работе №7

Дисциплина: Базы данных

Тема: Изучение работы транзакций

Выполнил студент гр. 43501/1

(подпись) Нгуен Тиен Ву

Руководитель

(подпись) Мяснов А.В.

“__” _____ 2016 г.

Санкт-Петербург

2016

1. Цель работы

Познакомить студентов с механизмом транзакций, возможностями ручного управления транзакциями, уровнями изоляции транзакций.

2. 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.

3. Выполнение работы

А. Были проведены тесты по запуску, подтверждению и откату транзакций:

Для тестирования была создана таблица tmp_trans с одним полем “tmp”, в это поле были добавлены два значения («20» и «40») после которого была создана точка сохранения save1.

```
SQL> create table tmp_trans (tmp int not null primary key);
SQL> commit;
SQL> insert into tmp_trans values (30);
SQL> commit;
SQL> insert into tmp_trans values (60);
SQL> select * from tmp_trans;

      TMP
=====
       30
       60

SQL> savepoint save1;
```

Были удалены все данные из таблицы, затем, вернувшись к точке сохранения, просмотрев данные в таблице мы увидели, что все данные вернулись. Потом, вернувшись к последнему подтверждению транзакции, в таблице осталось только одно значение «20».

```
SQL> delete from tmp_trans;
SQL> select * from tmp_trans;
SQL> rollback to save1;
SQL> select * from tmp_trans;
```

```

      TMP
=====
      30
      60

```

```
SQL> rollback;
SQL> select * from tmp_trans;
```

```

      TMP
=====
      30

```

В. Были проведены эксперименты, показывающие основные возможности транзакций с различным уровнем изоляции:

Snapshot

Это уровень изоляции по умолчанию. Он позволяет видеть неизменное состояние базы данных на момент старта транзакции. Изменения, выполненные другими транзакциями, в этой транзакции не видны. Свои изменения транзакция, разумеется, видит.

Один терминал:

```
SQL> insert into tmp_trans values (123);
SQL> commit;
SQL> select * from tmp_trans;
```

```

      TMP
=====
      30
      60
     123

```

Второй терминал:

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

          TMP
=====
          30
          60
```

По результатам работы видно, что при подключении двух клиентов один вставил в таблицу новое значение (666), но второй клиент не видит этих изменений.

Snapshot table stability

Уровень изоляции транзакции SNAPSHOT TABLE STABILITY аналогичен уровню SNAPSHOT с той лишь разницей, что в данном случае другие транзакции независимо от их уровня изоляции могут только читать данные таблиц, включенных в операции этой транзакции, но не могут их изменять.

После того как уровень изоляции установлен, для того чтобы завершить транзакцию все клиентские транзакции должны быть завершены (commit;).

На первом терминале установим уровень изоляции:

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

После данного действия, если у одного из клиентов есть незавершенные транзакции, мы не сможем завершить следующую транзакцию.

Просмотреть внесенные изменения, можно только после того как все другой клиент завершит транзакции:

Первый терминал:

```
SQL> delete from tmp_trans;
SQL> insert into tmp_trans values (400);
SQL> select * from tmp_trans;
```

| TMP |
|-------|
| ===== |
| 400 |

Второй терминал:

```
SQL> select * from tmp_trans;
```

| TMP |
|-------|
| ===== |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |

После *commit*; на первом терминале, на втором увидим те же данные:

```
SQL> select * from tmp_trans;
```

| TMP |
|-------|
| ===== |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |

Чтобы увидеть изменения, завершение транзакции нужно и на втором терминале (после *commit*; на втором терминале):

```
SQL> select * from tmp_trans;
```

```
      TMP
=====
      400
```

Read committed

Уровень изолированности READ COMMITTED позволяет в транзакции без её перезапуска видеть все подтверждённые изменения данных базы данных, выполненные в других параллельных транзакциях. Неподтверждённые изменения не видны в транзакции и этого уровня изоляции.

Первый терминал:

```
SQL> set transaction read committed;
```

```
Commit current transaction (y/n)?n
```

```
Rolling back work.
```

```
SQL> select * from tmp_trans;
```

```
      TMP
=====
         5
        10
        15
```

Второй терминал:

```
SQL> insert into tmp_trans values (15);
```

```
SQL> commit;
```

После установления уровня изоляции *read committed*, на первом терминале возможно увидеть изменения данных сразу после подтверждения транзакции на втором терминале.

Record_Version

Для этого уровня изолированности можно указать один из двух значений дополнительной характеристики в зависимости от желаемого способа разрешения конфликтов: RECORD_VERSION и NO RECORD_VERSION.

- NO RECORD_VERSION (значение по умолчанию) является в некотором роде механизмом двухфазной блокировки. В этом случае транзакция не может прочитать любую запись, которая была изменена параллельной активной (неподтвержденной) транзакцией.

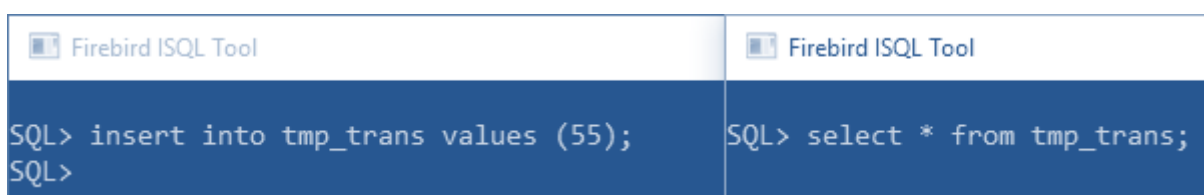
Если указана стратегия разрешения блокировок NO WAIT, то будет немедленно выдано соответствующее исключение. Если указана стратегия разрешения блокировок WAIT, то это приведёт к ожиданию завершения или откату конкурирующей транзакции.

- При задании RECORD_VERSION транзакция всегда читает последнюю подтверждённую версию записей таблиц, независимо от того, существуют ли изменённые и ещё не подтверждённые версии этих записей. В этом случае режим разрешения блокировок (WAIT или NO WAIT) никак не влияет на поведение транзакции при её старте.

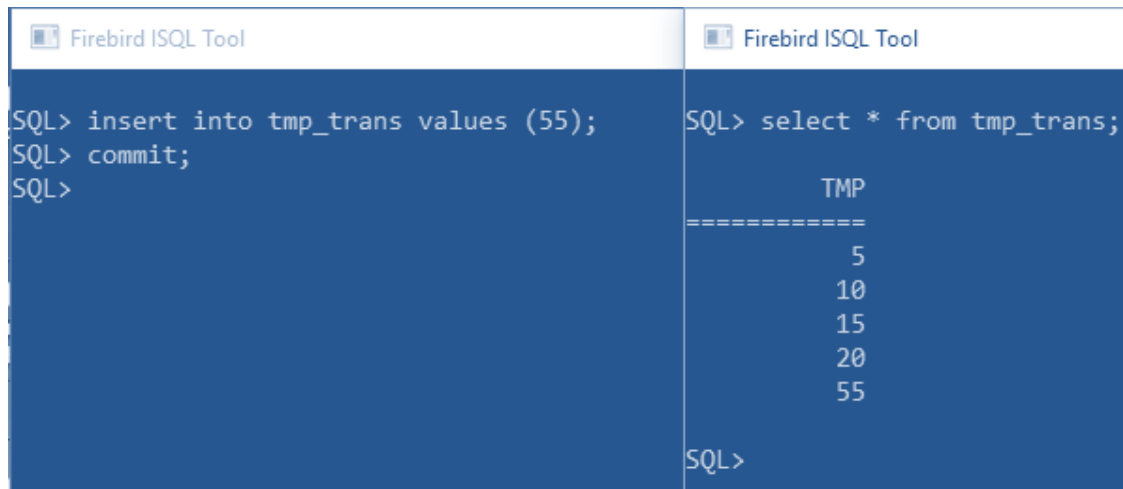
Изменим на одном терминале уровень изолированности:

```
SQL> set transaction read committed no record_version wait;  
Commit current transaction (y/n)?n  
Rolling back work.
```

До того момента, когда на первом терминале не будет завершена транзакция, второй будет ожидать момента ее завершения и не увидит запрашиваемые данные:



Сразу после того, как на первом терминале будет завершена транзакция, второй увидит измененные данные, завершив команду select:



```
SQL> insert into tmp_trans values (55);
SQL> commit;
SQL>
```

```
SQL> select * from tmp_trans;

      TMP
=====
        5
       10
       15
       20
       55

SQL>
```

Изменим на первом терминале уровень изолированности:

```
SQL> set transaction read committed no record_version no wait;
Commit current transaction (y/n)?n
Rolling back work.
```

На втором терминале добавим данные, не завершая транзакцию:

```
SQL> insert into tmp_trans values (100);
SQL>
```

При обращении первого терминала к данным таблицы, у него немедленно возникает соответствующее исключение:

```
SQL> set transaction read committed no record_version no wait;
Commit current transaction (y/n)?n
Rolling back work.
SQL> select * from tmp_trans;

      TMP
=====
        5
       10
       15
       20
       55
```



```
Statement failed, SQLSTATE = 40001  
lock conflict on no wait transaction  
-deadlock
```

4. Вывод

В данной лабораторной работе были изучены основные принципы работы транзакций и уровни изоляции.

Уровень изоляции SNAPSHOT является довольно неудобным по причине высокой вероятности получения неактуальных данных.

Если же требуется получить монополярный доступ к отдельным таблицам базы данных, следует выбрать SNAPSHOT TABLE STABILITY. При этом следует помнить, что при запуске такой транзакции и попытке изменить данные можно получить исключение по блокировке, если какая-либо параллельная транзакция выполнила изменения в таблице и не подтвердила транзакцию.

Для обновляющих транзакций наиболее подходящими являются два варианта набора характеристик: использование режима WAIT совместно с NO RECORD_VERSION и NO WAIT совместно с RECORD_VERSION. В первом случае полностью исключаются ошибки блокировки после отмены или подтверждения блокирующей транзакции. Во втором случае мы сразу получаем ошибку, после чего принимаем решение о дальнейших действиях.

Уровни изоляции транзакций уменьшают возможности параллельной обработки данных в таблицах, что исключает получение одним из клиентов некорректных или устаревших данных. Правильное использование уровней транзакций обеспечивает логическую целостность данных и правильность выполнения запросов клиентов.