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 Д. С. Раскин

(подпись)

Руководитель А.В. Мяснов

(подпись)

“ ” 2015 г.

Санкт-Петербург

2015

1. **Цель работы**

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

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

Были проведены тесты по запуску, подтверждению и откату транзакций:

SQL> create table test\_transact(

CON> test int not null primary key);

SQL> commit;

SQL> insert into test\_transact values (1);

SQL> commit;

SQL> insert into test\_transact values (2);

SQL> savepoint one;

SQL> delete from test\_transact;

SQL> select \* from test\_transact;

SQL> rollback to one;

SQL> select \* from test\_transact;

TEST

============

1

2

SQL> rollback;

SQL> select \* from test\_transact;

TEST

============

1

Для тестирования была создана таблица test\_transact с одним полем “test”, в это поле были добавлены по очереди два значения, сначала 1, и был выполнен коммит, потом 2, после которого создали точку сохранения. Удалив все данные из таблицы, проверили что их не существует, затем вернулись к точке сохранения, вызвали запрос получения данных и увидели, что все данные остались в таблице. Потом вернулись к последнему подтверждению транзакции, в таблице осталось только одно значение 1.

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

**Snapshot**

Уровень изолированности SNAPSHOT (уровень изолированности по умолчанию) означает, что этой транзакции видны лишь те изменения, фиксация которых произошла не позднее момента старта этой транзакции. Любые подтверждённые изменения, сделанные другими конкурирующими транзакциями, не будут видны в такой транзакции в процессе ее активности без её перезапуска. Чтобы увидеть эти изменения, нужно завершить транзакцию (подтвердить её или выполнить полный откат, но не откат на точку сохранения) и запустить транзакцию заново**.**

Один терминал:

SQL> insert into test\_transact values(7);

SQL> commit;

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 test\_transact;

TEST

============

1

Видно, что при подключении двух клиентов один вставил в таблицу новое значение (7), но второй клиент не видит этих изменений.

**Snapshot table stability**

Уровень изоляции транзакции SNAPSHOT TABLE STABILITY позволяет, как и в случае SNAPSHOT, также видеть только те изменения, фиксация которых произошла непозднее момента старта этой транзакции. При этом после старта такой транзакции в других клиентских транзакциях невозможно выполнение изменений ни в каких таблицах этой базы данных, уже каким-либо образом измененных первой транзакцией. Все такие попытки в параллельных транзакциях приведут к исключениям базы данных. Просматривать любые данные другие транзакции могут совершенно свободно.

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

На одном из терминалов установим уровень изоляции:

QL> 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:

Один терминал:

SQL> delete from test\_transact;

SQL> insert into test\_transact values(1);

SQL> select \* from test\_transact;

TEST

============

1

Второй терминал:

SQL> select \* from test\_transact;

TEST

============

1

7

5

3

11

9

12

13

15

16

17

После commit; на первом терминале на втором увидим:

Все еще:

SQL> select \* from test\_transact;

TEST

============

1

7

5

3

11

9

12

13

15

16

17

Чтобы увидеть изменения завершение транзакции нужны и на втором терминале(после commit; на втором терминале):

SQL> select \* from test\_transact;

TEST

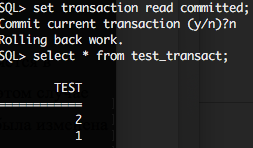
============

1

**Read commited**

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. Для получения обновлённого списка строк интересующей таблицы необходимо лишь повторное выполнение оператора SELECT в рамках активной транзакции READ COMMITTED без её перезапуска.





На рисунках видно, что один клиент выполняет вставку данных в таблицу. Второй клиент видит изменения сразу после подтверждения транзакции первым клиентом.

**Record\_Version**

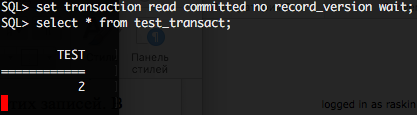
Для этого уровня изолированности можно указать один из двух значений дополнительной характеристики в зависимости от желаемого способа разрешения конфликтов: RECORD\_VERSION и NO RECORD\_VERSION. Как видно из их имён они являются взаимоисключающими.

• 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.

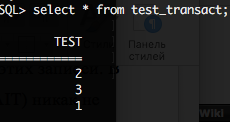
Если указана стратегия разрешения блокировок NO WAIT, то будет немедленно выдано соответствующее исключение. Если указана стратегия разрешения блокировок WAIT, то это приведёт к ожиданию завершения или откату конкурирующей транзакции. Если конкурирующая транзакция откатывается, или, если она завершается и её идентификатор старее (меньше), чем идентификатор текущей транзакции, то изменения в текущей транзакции допускаются. Если конкурирующая транзакция завершается и её идентификатор новее (больше), чем идентификатор текущей транзакции, то будет выдана ошибка конфликта блокировок.

• 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.



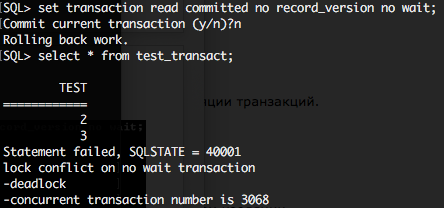


Из рисунка видно, что первый клиент добавляет данные в таблицу, второй клиент не может выполнить команду select, пока первый клиент не закончит.



После подтверждения первым клиентом транзакции, второй клиент завершает выполнение команды select.





На рисунке видно, что первый клиент добавляет данные в таблицу. При обращении вторым клиентом к таблице у него появляется исключение.

1. **Вывод**

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

Уровни изоляции транзакций уменьшают возможности параллельной обработки данных в таблицах, что исключает получение одним из клиентов некорректных или устаревших данных. Транзакции также обеспечивают гарантию того, что незаконченные обновления или искаженные действия не будут фиксироваться в базе данных(выполнение одного большего скрипта или набора команд, к примеру с одного локального компьютера превращается в атомарную операцию, которая либо выполнится целиком либо не выполнится вообще). Однако, надо понимать что при использование механизма транзакций присутствует потеря в скорости, которая, в сравнении с исходной, может достигать 3-5 раз .