

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Отчет по лабораторной работе №7

По дисциплине «Базы данных»

«Изучение работы транзакций»

Работу выполнили студенты группы №43501/4

Н.С. Шаляпин\_\_\_\_\_

Работу принял преподаватель

А.В. Мяснов\_\_\_\_\_

Санкт-Петербург

2015

# 1.Цель работы

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

## 2.Транзакции

Транзакция (англ, transaction) — группа последовательных операций с базой данных, которая представляет собой логическую единицу работы с данными. Транзакция может быть выполнена либо целиком и успешно, соблюдая целостность данных и независимо от параллельно идущих других транзакций, либо не выполнена вообще и тогда она не должна произвести никакого эффекта. Для транзакций существует два режима доступа к данным базы данных: READWRITEи READONLY,

- При режиме доступа READWRITEоперации в контексте данной транзакции могут быть как операциями чтения, так и операциями изменения данных. Это режим по умолчанию.
- операции выборки данных SELECT, Любая попытка изменения данных в контексте такой транзакции приведёт к исключениям базы данных. Однако это не относится к глобальным временным таблицам (GTT), которые разрешено модифицировать в READ ONLY транзакциях.

При работе с одной и той же базой данных нескольких клиентских приложений могут возникать блокировки. Блокировки могут возникать, когда одна транзакция вносит неподтверждённые изменения в строку таблицы или удаляет строку, а другая транзакция пытается изменять или удалять эту же строку. Такие блокировки называются конфликтом обновления. Блокировки также могут возникнуть и в других ситуациях при использовании некоторых уровней изоляции транзакций.

Существуют два режима разрешения блокировок: WAITи NOWAIT. В режиме WAIT(режим по умолчанию) при появлении конфликта с параллельными транзакциями, выполняющими конкурирующие обновления данных в той же базе данных, такая транзакция будет ожидать завершения конкурирующей транзакции путём её подтверждения (COMMIT) или отката (ROLLBACK).Иными словами, клиентское приложение будет переведено в режим ожидания до момента разрешения конфликта.

Если установлен режим разрешения блокировок NOWAIT, то при появлении конфликта блокировки данная транзакция немедленно вызовет исключение базы данных.

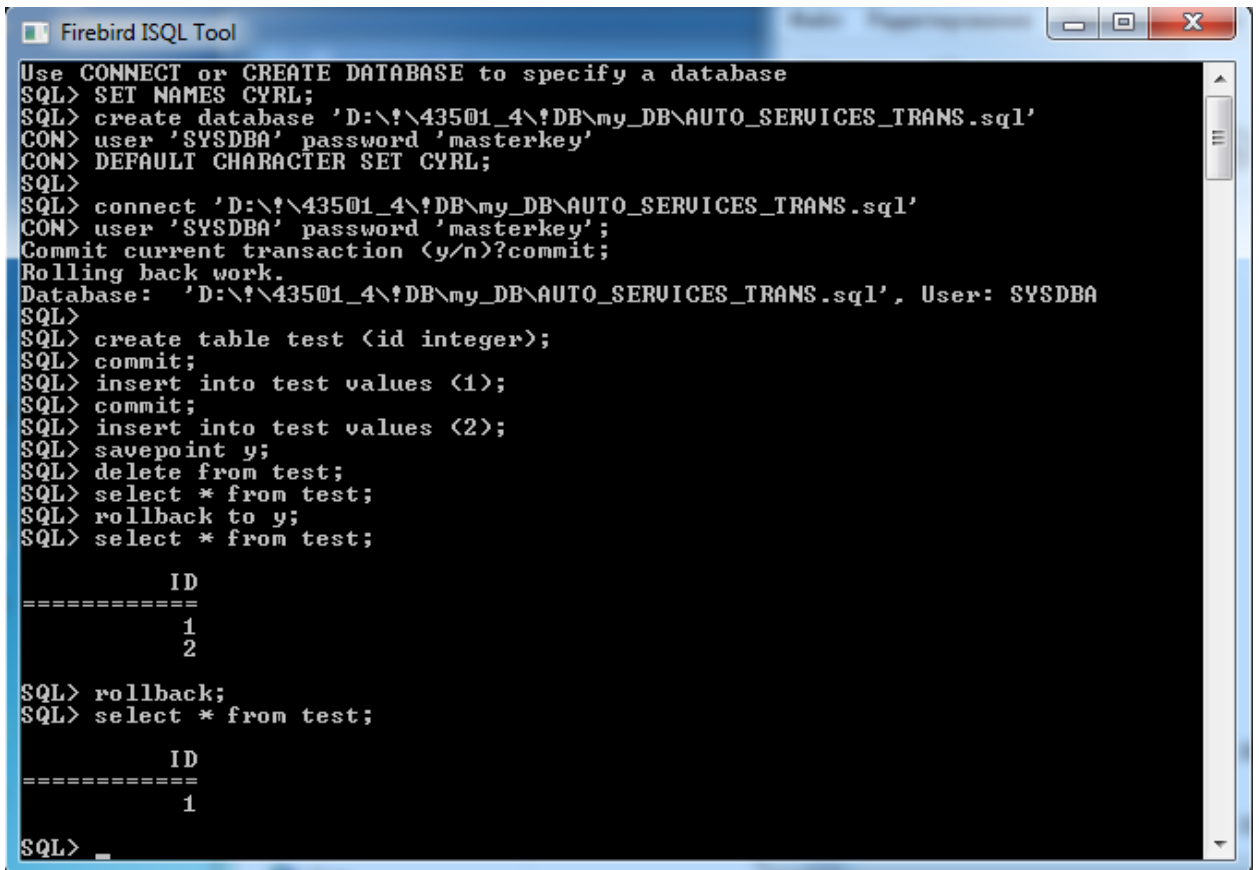
Уровень изолированности транзакций — значение, определяющее уровень, при котором в транзакции допускаются несогласованные данные, то есть степень изолированности одной транзакции от другой. Изменения, внесённые некоторым оператором, будут видны всем последующим операторам, запущенным в рамках этой же транзакции, независимо от её уровня изолированности. Изменения, произведённые в рамках другой транзакции остаются невидимыми для текущей транзакции до тех пор пока они не подтверждены. Уровень изолированности, а иногда, другие атрибуты, определяет, как транзакции будут взаимодействовать с другой транзакцией, которая хочет подтвердить изменения.

### 3. Программа работы

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

### 4. Ход работы

1. Были изучены основные принципы работы транзакций.
2. Были проведены эксперименты по запуску, подтверждению и откату транзакций:



```
Firebird ISQL Tool
Use CONNECT or CREATE DATABASE to specify a database
SQL> SET NAMES CYRL;
SQL> create database 'D:\!\43501_4\!DB\my_DB\AUTO_SERVICES_TRANS.sql'
CON> user 'SYSDBA' password 'masterkey'
CON> DEFAULT CHARACTER SET CYRL;
SQL>
SQL> connect 'D:\!\43501_4\!DB\my_DB\AUTO_SERVICES_TRANS.sql'
CON> user 'SYSDBA' password 'masterkey';
Commit current transaction (y/n)?commit;
Rolling back work.
Database: 'D:\!\43501_4\!DB\my_DB\AUTO_SERVICES_TRANS.sql', User: SYSDBA
SQL>
SQL> create table test (id integer);
SQL> commit;
SQL> insert into test values (1);
SQL> commit;
SQL> insert into test values (2);
SQL> savepoint y;
SQL> delete from test;
SQL> select * from test;
SQL> rollback to y;
SQL> select * from test;

      ID
=====
      1
      2

SQL> rollback;
SQL> select * from test;

      ID
=====
      1

SQL> _
```

Рис. 1: Опыты с подтверждением и запуском транзакций.

3. Были изучены уровни изоляции транзакций в Firebird.
4. Были проведены эксперименты с различными уровнями изоляции транзакций.

Опыты с уровнем изоляции snapshot: позволяет видеть только 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 (рисунок 2).

```

SQL> insert into test values (4);
SQL> commit;
SQL>

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

      ID
=====
      1
      2
      3

SQL>
  
```

Рис. 2: Два клиента: один выполнил вставку в таблицу, второй при этом не видит произведенных изменений.

Опыты с уровнем изоляции snapshottablestability: позволяет видеть только те изменения, фиксация которых произошла не позднее 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 (рисунки 3-5).

```

SQL> drop table test;
SQL> create table test (id integer);
SQL> commit;
SQL> set transaction snapshot table stability;
SQL> insert into test values (1);
SQL> select * from test;

      ID
=====
      1

SQL>

SQL> commit;
SQL> set transaction snapshot table stability;
SQL> insert into test values (2);
SQL>
  
```

Рис. 3: Два клиента: один выполни вставку в таблицу, второй при этом не может завершить операцию вставки.

```

SQL> drop table test;
SQL> create table test (id integer);
SQL> commit;
SQL> set transaction snapshot table stability;
SQL> insert into test values (1);
SQL> select * from test;

      ID
=====
      1

SQL> commit;
SQL>

SQL> commit;
SQL> set transaction snapshot table stability;
SQL> insert into test values (2);
SQL>
  
```

Рис. 4: Два клиента: после того, как первый клиент зафиксировал изменения, второй смог выполнить операцию вставки.

```
SQL> select * from test;
=====
ID
=====
1
SQL> commit;
SQL> select * from test;
=====
ID
=====
1
2
SQL>

SQL> set transaction snapshot table stability;
SQL> insert into test values (2);
SQL> select * from test;
=====
ID
=====
2
SQL> commit;
SQL> select * from test;
=====
ID
=====
1
2
SQL> _
```

Рис. 5: Два клиента: изменения, произведенные в других транзакциях видны только после того, как изменения были зафиксированы.

Опыты с уровнем изоляции readcommitted: 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.

С опцией 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 (WAITили XOWAIT) никак не влияет на поведение транзакции при её старте (рисунок 6).

```
SQL> insert into test values (5);
SQL> commit;
SQL> insert into test values (6);
SQL> commit;
SQL>

SQL> set transaction read committed;
SQL> select * from test;
=====
ID
=====
1
2
3
4
5
SQL> select * from test;
=====
ID
=====
1
2
3
4
5
6
SQL>
```

Рис. 6: Два клиента: один выполнил вставку в таблицу, второй видит ее сразу после коммита.

С опцией **no record\_version wait**: 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 (WAITили NO WAIT) никак не влияет на поведение транзакции при её старте (рисунки 7,8).

```

SQL> commit;
SQL> insert into test values (4);
SQL>
SQL> set transaction read committed no record_version wait;
Commit current transaction (y/n)?
Rolling back work.
SQL> select * from test;

```

Рис. 7: Два клиента: один выполнил вставку в таблицу, второй не может выполнить select пока первый не закоммитит.

```

SQL> insert into test values (1);
SQL> commit;
SQL> delete from test where id > 3;
SQL> commit;
SQL> insert into test values (4);
SQL> commit;
SQL>
SQL> set transaction read committed no
Commit current transaction (y/n)?
Rolling back work.
SQL> select * from test;

```

ID
1
2
3
4

```

SQL> _

```

Рис. 8: Два клиента: первый зафиксирован изменения, второй сразу завершил выполнение select.

Сопцией `no record_version nowait`: при обращении к таблице, измененной в другой неподтвержденной транзакции, база выбросит исключение (рисунок 9).

```

SQL> insert into test values (5);
SQL>
SQL> set transaction read committed no record_version no wait;
SQL> select * from test;

```

ID
1
2
3
4

```

Statement failed, SQLSTATE = 40001
lock conflict on no wait transaction
-deadlock
-concurrent transaction number is 7030
SQL>

```

Рис. 9: Два клиента: один выполнил вставку в таблицу, у второго вылетело исключение при обращении к этой таблице.

## 5. Вывод

В данной работе был изучен механизм транзакций, возможности управления транзакциями и уровни изоляции транзакций в Firebird. Транзакции позволяют сохранять целостность БД при подключении к ней нескольких клиентов. Задавая различные типы разрешения конфликтов и уровни изоляции можно управлять видимостью изменений, произошедших в базе, для разных пользователей.