**Санкт- Петербургский политехнический университет Петра Великого**

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Отчет по лабораторной работе

Курс: «Базы данных»

Тема: «Изучение работы транзакций»

**Выполнил:**

Бояркин Н.С. группа 43501/3

**Проверил:**

Мяснов А.В.

Санкт – Петербург

2017

# 1. Цель работы

Познакомить студентов с возможностями реализации более сложной обработки данных на стороне сервера с помощью хранимых процедур и триггеров.

# 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. Ход работы

**Основные принципы работы транзакций**

**Транзакция** – атомарное действие над базой данных, переводящее ее из одного целостного состояния в другое целостное состояние.

Транзакция обладает четырьмя важными свойствами:

1. **Атомарность** – выражается в том, что транзакция должна быть выполнена в целом или не выполнена вовсе.
2. **Согласованность** – гарантирует, что по мере выполнения транзакций, данные переходят из одного согласованного состояния в другое, т. е. транзакция не разрушает взаимной согласованности данных.
3. **Изолированность** – означает, что конкурирующие за доступ к БД транзакции физически обрабатываются последовательно, изолированно друг от друга, но для пользователей это выглядит так, как будто они выполняются параллельно.
4. **Долговечность** – если транзакция завершена успешно, то те изменения, в данных, которые были ею произведены, не могут быть потеряны ни при каких обстоятельствах.

Транзакция начинается автоматически при подключении клиента к базе данных и продолжается до выполнения команд COMMIT или ROLLBACK, либо до отключения клиента или сбоя сервера.

**Эксперименты по запуску, подтверждению и откату транзакций**

Для примера создадим временную базу данных и таблицу в ней. После этого в таблицу добавим значение 30 и сделаем ROLLBACK, добавим значение 40 и сделаем ROLLBACK, добавим точку сохранения.

|  |
| --- |
| ---------------------------------------- CREATE DATABASE ----------------------------------------  Use CONNECT or CREATE DATABASE to specify a database  SQL> CREATE DATABASE '127.0.0.1/3050:D:\temp\TEMP\_DATABASE.fdb' USER 'SYSDBA' PASSWORD 'masterkey';  SQL> CONNECT '127.0.0.1/3050:D:\temp\TEMP\_DATABASE.fdb' USER 'SYSDBA' PASSWORD 'masterkey';  Commit current transaction (y/n)?Y  Committing.  Database: '127.0.0.1/3050:D:\temp\TEMP\_DATABASE.fdb', User: SYSDBA  SQL> CREATE TABLE TEMP\_TABLE (ID\_TEMP\_TABLE INTEGER NOT NULL PRIMARY KEY);  SQL> COMMIT;    ------------------------------------------- ROLLBACK -------------------------------------------    SQL> INSERT INTO TEMP\_TABLE VALUES(30);  SQL> COMMIT;  SQL> SELECT \* FROM TEMP\_TABLE;    ID\_TEMP\_TABLE  =============             30    SQL> ROLLBACK;  SQL> SELECT \* FROM TEMP\_TABLE;    ID\_TEMP\_TABLE  =============             30    SQL> INSERT INTO TEMP\_TABLE VALUES(35);  SQL> SELECT \* FROM TEMP\_TABLE;    ID\_TEMP\_TABLE  =============             30             35    SQL> ROLLBACK;  SQL> SELECT \* FROM TEMP\_TABLE;    ID\_TEMP\_TABLE  =============             30  ------------------------------------------- SAVEPOINT -------------------------------------------    SQL> INSERT INTO TEMP\_TABLE VALUES(40);  SQL> SELECT \* FROM TEMP\_TABLE;    ID\_TEMP\_TABLE  =============             30             40    SQL> SAVEPOINT FIRST\_SAVEPOINT;  SQL> DELETE FROM TEMP\_TABLE;  SQL> SELECT \* FROM TEMP\_TABLE;  SQL> ROLLBACK TO FIRST\_SAVEPOINT;  SQL> SELECT \* FROM TEMP\_TABLE;    ID\_TEMP\_TABLE  =============             30             40 |

В результате, откат добавления единственного значения 30 в таблицу не сработал, в то время как откат добавления второго значения 40 сработал.

Также, успешно был проведен эксперимент с созданием точки сохранения и откату на нее.

**Уровни изоляции транзакций в Firebird**

Firebird предоставляет три уровня изоляции транзакций для определения "глубины" согласованности требований транзакции.

1. **SNAPSHOT** – Дает состояние базы данных на момент старта транзакции. Изменения, выполненные другими транзакциями, в данной транзакции не видны. Транзакция видит все изменения, выполненные в контексте этой транзакции.
2. **READ** **COMMITTED** – Транзакция может видеть самые последние подтвержденные изменения базы данных, выполненные другими транзакциями.
3. **SNAPSHOT** **TABLE** **STABILITY** - Аналогичен уровню SNAPSHOT с тем отличием, что другим транзакциям разрешено чтение данных из таблиц данной транзакции, однако они не могут вносить в них никаких изменений.

В одном крайнем случае транзакция может получить исключительный доступ по записи ко всей таблице, в то время как в другом крайнем случае неподтвержденная транзакция получает доступ к любым внешним изменениям состояния базы данных. Никакая транзакция в Firebird не сможет видеть неподтвержденные изменения данных от других транзакций.

**Провести эксперименты, показывающие основные возможности транзакций с различным уровнем изоляции (SNAPSHOT).**

Первый процесс:

|  |
| --- |
| SQL> DELETE FROM TEMP\_TABLE;  SQL> COMMIT;  SQL> INSERT INTO TEMP\_TABLE VALUES(1);  SQL> INSERT INTO TEMP\_TABLE VALUES(2);  SQL> INSERT INTO TEMP\_TABLE VALUES(3);  SQL> COMMIT;  SQL> SELECT \* FROM TEMP\_TABLE;    ID\_TEMP\_TABLE  =============              1              2              3 |

Второй процесс:

|  |
| --- |
| SQL> CONNECT '127.0.0.1/3050:D:\temp\TEMP\_DATABASE.fdb' USER 'SYSDBA' PASSWORD 'masterkey';  Database: '127.0.0.1/3050:D:\temp\TEMP\_DATABASE.fdb', User: SYSDBA  SQL> SET TRANSACTION SNAPSHOT;  Commit current transaction (y/n)?N  Rolling back work.  SQL> SELECT \* FROM TEMP\_TABLE;    ID\_TEMP\_TABLE  =============              1              2 |

Во втором процессе новой записи не появилось, в то время как в первом успешно добавилось.

**Провести эксперименты, показывающие основные возможности транзакций с различным уровнем изоляции (READ COMITTED).**

Первый процесс:

|  |
| --- |
| SQL> SELECT \* FROM TEMP\_TABLE;    ID\_TEMP\_TABLE  =============              1              2              3    SQL> INSERT INTO TEMP\_TABLE VALUES(4);  SQL> COMMIT;  SQL> SELECT \* FROM TEMP\_TABLE;    ID\_TEMP\_TABLE  =============              1              2              3              4 |

Второй процесс:

|  |
| --- |
| SQL> SELECT \* FROM TEMP\_TABLE;    ID\_TEMP\_TABLE  =============              1              2              3    SQL> SET TRANSACTION READ COMMITTED;  Commit current transaction (y/n)?N  Rolling back work.  SQL> SELECT \* FROM TEMP\_TABLE;    ID\_TEMP\_TABLE  =============              1              2              3              4 |

Второй процесс увидел изменения в таблице сразу же после подтверждения транзакции в первом процессе.

**Провести эксперименты, показывающие основные возможности транзакций с различным уровнем изоляции (SNAPSHOT TABLE STABILITY).**

Первый процесс:

|  |
| --- |
| SQL> SELECT \* FROM TEMP\_TABLE;    ID\_TEMP\_TABLE  =============              1              2              3    SQL> UPDATE TEMP\_TABLE SET ID\_TEMP\_TABLE = 4 WHERE ID\_TEMP\_TABLE = 3;  SQL> COMMIT;  SQL> SELECT \* FROM TEMP\_TABLE;    ID\_TEMP\_TABLE  =============              1              2              4    SQL> UPDATE TEMP\_TABLE SET ID\_TEMP\_TABLE = 3 WHERE ID\_TEMP\_TABLE = 4;  SQL> COMMIT;  SQL> SELECT \* FROM TEMP\_TABLE;    ID\_TEMP\_TABLE  =============              1              2              3 |

Второй процесс:

|  |
| --- |
| SQL> SET TRANSACTION ISOLATION LEVEL SNAPSHOT TABLE STABILITY;  Commit current transaction (y/n)?N  Rolling back work.  SQL> SELECT \* FROM TEMP\_TABLE;    ID\_TEMP\_TABLE  =============              1              2              3    SQL> UPDATE TEMP\_TABLE SET ID\_TEMP\_TABLE = 4 WHERE ID\_TEMP\_TABLE = 3;  SQL> SELECT \* FROM TEMP\_TABLE;    ID\_TEMP\_TABLE  =============              1              2              4 |

Возвращение в первый процесс:

|  |
| --- |
| SQL> SELECT \* FROM TEMP\_TABLE;    ID\_TEMP\_TABLE  =============              1              2              3 |

Изменения в транзакции с уровнем изоляции SNAPSHOT TABLE STABILITY (второй процесс) целиком блокируются для всех остальных транзакций (первый процесс) до окончания выполнения транзакции.

# 4. Вывод

Механизм транзакций позволяет поддерживать целостность данных при параллельной работе нескольких клиентов с базой данных.

Достоинства транзакций:

1. Обеспечение корректной работы в многопользовательских системах при параллельном обращении к одним и тем же данным.
2. Восстанавливаясь после сбоев, система ликвидирует следы транзакций, не успевших успешно завершиться в результате программного или аппаратного сбоя.
3. Поддержание логической целостности базы данных.

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