Санкт-Петербургский государственный политехнический
университет Петра Великого
**Кафедра компьютерных систем и программных технологий**

**Отчет по лабораторной работе**
**Дисциплина:** Базы данных
**Тема:** Изучение работы транзакций

Выполнил
студент гр. 43501/3 _____ Д. А. Зобков
(подпись)

Преподаватель _____ А. В. Мяснов
(подпись)

“___”_____ 2016 г.

Санкт-Петербург
2016 г.

# 1 Цель работы

Познакомиться 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.

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.

# 3 Ход работы

## 3.1 Изучить основные принципы работы транзакций

Транзакция — задача с определенными характеристиками, включающая любое количество различных операций (SELECT, INSERT, UPDATE, или DELETE) над данными в БД. Транзакция должна рассматриваться как единая операция над данными, т.е. она выполняется либо полностью, либо никак. Так как возможны случаи возникновения ошибок во время выполнения транзакции, приводящих к невозможности ее закончить, необходимы особые операции для подтверждения (COMMIT) или отмены транзакции (ROLLBACK).

Все выполняемые транзакции должны удовлетворять следующим свойствам (ACID критериям):

- *Атомарность.* Выполнение по принципу „все или ничего“;
- *Согласованность.* В результате транзакции система переходит из одного абстрактного корректного состояния в другое;
- *Изолированность.* Данные, находящиеся в несогласованном состоянии, не должны быть видны другим транзакциям, пока изменения не будут завершены;
- *Долговечность.* Если транзакция зафиксирована, то ее результаты должны быть долговечными. Новые состояния всех объектов сохранятся даже в случае аппаратных или системных сбоев.

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

## 3.2 Провести эксперименты по запуску, подтверждению и откату транзакций

Рассмотрим команды COMMIT и ROLLBACK на примере добавления записей в таблицу:

```
/* Таблица для теста */
SQL> SELECT * FROM AWARDS;

    AWARD_ID AW_NAME
============ ====================
           1 Grammy
           2 MTV Hits

/* Добавим запись */
SQL> INSERT INTO AWARDS VALUES (3, 'TEST');
SQL> SELECT * FROM AWARDS;

    AWARD_ID AW_NAME
============ ====================
           1 Grammy
           2 MTV Hits
           3 TEST

/* Отмена транзакции */
SQL> ROLLBACK;
SQL> SELECT * FROM AWARDS;

    AWARD_ID AW_NAME
============ ====================
           1 Grammy
           2 MTV Hits

/* Добавим запись снова, но подтвердим транзакцию */
SQL> INSERT INTO AWARDS VALUES (3, 'TEST');
SQL> COMMIT;
SQL> SELECT * FROM AWARDS;

    AWARD_ID AW_NAME
============ ====================
           1 Grammy
           2 MTV Hits
           3 TEST

/* Удалим запись */
SQL> DELETE FROM AWARDS WHERE AWARD_ID = 3;
SQL> SELECT * FROM AWARDS;

    AWARD_ID AW_NAME
============ ====================
```

```
           1 Grammy
           2 MTV Hits

/* Создадим точку сохранения и добавим запись снова */
SQL> SAVEPOINT FIRST;
SQL> INSERT INTO AWARDS VALUES (3, 'TEST');
/* Повторим процесс */
SQL> SAVEPOINT SEC;
SQL> INSERT INTO AWARDS VALUES (4, 'TEST2');
SQL> SELECT * FROM AWARDS;

    AWARD_ID AW_NAME
============ ====================
           1 Grammy
           2 MTV Hits
           3 TEST
           4 TEST2

/* Откатим транзакцию на 2-ую точку сохранения */
SQL> ROLLBACK TO SEC;
SQL> SELECT * FROM AWARDS;

    AWARD_ID AW_NAME
============ ====================
           1 Grammy
           2 MTV Hits
           3 TEST

/* Откатим транзакцию на 1-ую точку сохранения */
SQL> ROLLBACK TO FIRST;
SQL> SELECT * FROM AWARDS;

    AWARD_ID AW_NAME
============ ====================
           1 Grammy
           2 MTV Hits

/* Завершение транзакции */
SQL> COMMIT;
```

## 3.3 Разобраться с уровнями изоляции транзакций в Firebird

Каждая транзакция имеет свой уровень изоляции, который устанавливается при запуске и остается неизменным в течение всей ее жизни. Firebird SQL предусматривает 3 уровня изоляции:

– *READ COMMITTED* („читать подтвержденные данные“)

Уровень изоляции READ COMMITTED используется, когда мы хотим видеть все подтвержденные результаты параллельно выполняющихся (в рамках других транзакций) действий. Этот уровень изоляции гарантирует,

что мы не сможем прочитать неподтвержденные данные, измененные в других транзакциях, и делает возможным прочитать подтвержденные данные.

– *SNAPSHOT* („моментальный снимок“)

  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 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.

## 3.4 Спланировать и провести эксперименты, показывающие основные возможности транзакций с различным уровнем изоляции

### 3.4.1 READ COMMITED

```
SQL> SELECT * FROM AWARDS;

    AWARD_ID AW_NAME
============ ====================
           1 Grammy
           2 MTV Hits
           3 TEST

SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
Commit current transaction (y/n)?y
Committing.
SQL> SELECT * FROM AWARDS;
/* Запрос "висит" до команды COMMIT на втором терминале */

    AWARD_ID AW_NAME
============ ====================
           1 Grammy
           2 MTV Hits
           3 NEW TEST
```

Листинг 3.1. Терминал 1 (READ COMMITED)

```
SQL> UPDATE AWARDS SET AW_NAME = 'NEW TEST' WHERE AWARD_ID = 3;
SQL> COMMIT;
```

Листинг 3.2. Терминал 2

В ходе эксперимента выполняем неподтвержденное изменение данных таблицы AWARDS. При этом параллельно выполняется транзакция с уровнем изоляции READ COMMITTED, содержащая запрос SELECT. Транзакция не выполняется („зависает“), т.к. она обращается к неподтвержденным данным, но после подтверждения изменения мы увидим измененное содержимое таблицы.

### 3.4.2 SNAPSHOT

```
SQL> SELECT * FROM AWARDS;

    AWARD_ID AW_NAME
============ ====================
           1 Grammy
           2 MTV Hits
           3 TEST

SQL> SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
Commit current transaction (y/n)?y
Committing.
/* Выполняем после COMMIT во 2 терминале */
SQL> SELECT * FROM AWARDS;

    AWARD_ID AW_NAME
============ ====================
           1 Grammy
           2 MTV Hits
           3 TEST

SQL> UPDATE AWARDS SET AW_NAME = 'TEST' WHERE AWARD_ID = 3;
Statement failed, SQLSTATE = 40001
deadlock
-update conflicts with concurrent update
-concurrent transaction number is 13816
```

Листинг 3.3. Терминал 1 (SNAPSHOT)

```
SQL> UPDATE AWARDS SET AW_NAME = 'NEW TEST' WHERE AWARD_ID = 3;
SQL> COMMIT;
```

Листинг 3.4. Терминал 2

Видим, что транзакция, содержащая SELECT, вывела „старые“ данные, подтвержденные на момент старта транзакции. При таком уровне изоляции даже подтвержденные изменения данных из других транзакций не видны. Можно заметить,

что при изменении данных в рамках сделанного снимка базы возникает конфликт между двумя транзакциями. В конечном итоге в силу вступили изменения из второго терминала.

### 3.4.3 SNAPSHOT TABLE STABILITY

```
SQL> SELECT * FROM AWARDS;

    AWARD_ID AW_NAME
============ ====================
           1 Grammy
           2 MTV Hits
           3 TEST

SQL> SET TRANSACTION ISOLATION LEVEL SNAPSHOT TABLE STABILITY;
Commit current transaction (y/n)?y
Committing.
SQL> UPDATE AWARDS SET AW_NAME = 'NEW TEST' WHERE AWARD_ID = 3;
SQL> COMMIT;
SQL> SELECT * FROM AWARDS;

    AWARD_ID AW_NAME
============ ====================
           1 Grammy
           2 MTV Hits
           3 NEW TEST
```

Листинг 3.5. Терминал 1 (SNAPSHOT TABLE STABILITY)

```
SQL> UPDATE AWARDS SET AW_NAME = 'NEW TEST' WHERE AWARD_ID = 3;
/* Запрос "висит" до команды COMMIT на первом терминале */
Statement failed, SQLSTATE = 40001
deadlock
-update conflicts with concurrent update
-concurrent transaction number is 13795
```

Листинг 3.6. Терминал 2

Также, как и в предыдущем случае, возник конфликт между двумя транзакциями, но в этот раз транзакция с уровнем изоляции SNAPSHOT TABLE STABILITY запретила изменение таблицы из других транзакций. В этом случае в силу вступили изменения первого терминала.

## 4 Выводы

В ходе выполнения работы было изучено создание и управление транзакциями, а также на примерах рассмотрены уровни изоляции.

Транзакции позволяют выполнять контроль целостности данных при выполнении различных запросов, но если требовать последовательной обработки транзакций, то возможно уменьшение производительности, т.к. при параллельном выполнении транзакций возможны следующие проблемы:

– Грязное чтение (Dirty Read) — чтение данных, добавленных или измененных транзакцией, которая впоследствии не подтвердится (откатится);
– Размытое чтение (Fuzzy Read) — при повторном чтении в рамках одной транзакции, ранее прочитанные данные оказываются измененными;
– Фантомное чтение (Phantom Read) — одна транзакция в ходе своего выполнения несколько раз выбирает множество строк по одним и тем же критериям, другая транзакция в интервалах между этими выборками добавляет или удаляет строки или изменяет столбцы некоторых строк, используемых в критериях выборки первой транзакции, и успешно заканчивается; в результате получится, что одни и те же выборки в первой транзакции дают разные множества строк.

Для достижения компромисса между быстродействием существуют несколько уровней изоляции транзакций. При этом можно выделить соответствие между уровнями изоляции и возможными конфликтами:

| **Уровень изоляции** | **Dirty Read** | **Fuzzy Read** | **Phantom Read** |
|---|---|---|---|
| READ UNCOMMITTED | Возможно | Возможно | Возможно |
| READ COMMITED | Невозможно | Возможно | Возможно |
| REPEATABLE READ | Невозможно | Невозможно | Возможно |
| SERIALIZABLE | Невозможно | Невозможно | Невозможно |

Таким образом, сериализуемость — это способность к упорядочению параллельной обработки транзакций. Также можно установить соответствие между уровнями изоляции по SQL-стандарту и уровнями изоляции транзакций в Firebird SQL:

| **Уровни изоляции по SQL** | **Уровни изоляции транзакций в Firebird** |
|---|---|
| READ COMMITED | READ COMMITED |
| REPEATABLE READ | SNAPSHOT |
| SERIALIZABLE | SNAPSHOT TABLE STABILITY |