

Санкт-Петербургский Государственный Политехнический  
Университет Петра Великого  
Институт Компьютерных Наук и Технологий  
**Кафедра Компьютерных Систем и Программных Технологий**

**Отчёт по лабораторной работе**

**Дисциплина:** Базы данных

**Тема:** Изучение работы транзакций

Выполнил студент группы 43501/3

\_\_\_\_\_ Круминьш Д.В.  
(подпись)

Преподаватель

\_\_\_\_\_ Мяснов А.В.  
(подпись)

# 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.

## Ход работы

### 1 Изучить основные принципы работы транзакций.

**Транзакция** - это неделимая, с точки зрения воздействия на СУБД, последовательность операций манипулирования данными. Для пользователя транзакция выполняется по принципу "все или ничего" т.е. либо транзакция выполняется целиком и переводит базу данных из одного целостного состояния в другое целостное состояние, либо, если по каким-либо причинам, одно из действий транзакции невыполнимо, или произошло какое-либо нарушение работы системы, база данных возвращается в исходное состояние, которое было до начала транзакции (происходит откат транзакции). Транзакция обладает четырьмя важными свойствами:

- **Атомарность.** Транзакция выполняется как атомарная операция - либо выполняется вся транзакция целиком, либо она целиком не выполняется.
- **Согласованность.** Транзакция переводит базу данных из одного согласованного (целостного) состояния в другое согласованное (целостное) состояние. Внутри транзакции согласованность базы данных может нарушаться.
- **Изоляция.** Транзакции разных пользователей не должны мешать друг другу (например, как если бы они выполнялись строго по очереди).
- **Долговечность.** Если транзакция выполнена, то результаты ее работы должны сохраниться в базе данных, даже если в следующий момент произойдет сбой системы.

Транзакция обычно начинается автоматически с момента присоединения пользователя к СУБД и продолжается до тех пор, пока не произойдет одно из следующих событий:

- Подана команда **COMMIT** (зафиксировать транзакцию).
- Подана команда **ROLLBACK** (откатить транзакцию).
- Произошло отсоединение пользователя от СУБД.
- Произошел сбой системы.

## 2 Провести эксперименты по запуску, подтверждению и откату транзакций.

Были проведены тесты по запуску, подтверждению и откату транзакций.

```
1 SQL> create table temp_table(someId integer not null primary key);
2 SQL> commit;
3 SQL> insert into temp_table values(1);
4 SQL> commit;
5 SQL> rollback;
6 SQL> select * from temp_table;
7
8         SOMEID
9 =====
10             1
11
12 SQL> insert into temp_table values(2);
13 SQL> select * from temp_table;
14
15         SOMEID
16 =====
17             1
18             2
19
20 SQL> rollback;
21 SQL> select * from temp_table;
22
23         SOMEID
24 =====
25             1
26
27 SQL> insert into temp_table values(2);
28 SQL> savepoint one;
29 SQL> delete from temp_table;
30 SQL> insert into temp_table values(3);
31 SQL> select * from temp_table;
32 SQL> rollback to one;
33 SQL> select * from temp_table;
34
35         SOMEID
36 =====
37             1
38             2
```

Листинг 1: Лог работы в isql

Была создана таблица temp\_table с одним полем someId, в это поле сперва было добавлено значение 1, rollback шедший за commit не сработал и значение осталось в таблице.

При добавлении значения 2 хоть с помощью запроса select оно было показано в таблице, но после команды rollback, оно оттуда удалилось.

Также имеется возможность добавлять точки сохранения для rollback, так данные таблицы были удалены и добавлена новая строка, отличная от предыдущих, но после вызова rollback на savepoint, содержимое таблицы вернулось к исходному состоянию.

### 3 Разобраться с уровнями изоляции транзакций в Firebird.

Уровень изолированности транзакции определяет, какие изменения, сделанные в других транзакциях, будут видны в данной транзакции. Каждая транзакция имеет свой уровень изоляции, который устанавливается при ее запуске и остается неизменным в течение всей ее жизни. Транзакции в Firebird могут иметь 3 основных возможных уровня изоляции: READ COMMITTED, SNAPSHOT и SNAPSHOT TABLE STABILITY. Каждый из этих трех уровней изоляции определяет правила видимости тех действий, которые выполняются другими транзакциями. Рассмотрим уровни изоляции более подробно.

- **READ COMMITTED.** Буквально переводится как "читать подтвержденные данные" однако это не совсем (точнее, не всегда) так. Уровень изоляции READ COMMITTED используется, когда мы хотим видеть все подтвержденные результаты параллельно выполняющихся (т. е. в рамках других транзакций) действий. Этот уровень изоляции гарантирует, что мы НЕ сможем прочитать неподтвержденные данные, измененные в других транзакциях, и делает ВОЗМОЖНЫМ чтение подтвержденных данных.
- **SNAPSHOT.** 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 подтвержденных (и разумеется, неподтвержденных) 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.

### 4 Спланировать и провести 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 старта этой транзакции. Любые подтвержденные изменения, сделанные другими конкурирующими транзакциями, не будут видны в такой транзакции в процессе ее активности без её перезапуска. Данные, которые не изменились, не блокируются.

```
1 SQL> insert into temp_table values(3);
2 SQL> commit;
3 SQL> select * from temp_table;
4
5      SOMEID
```

```

6 =====
7         1
8         2
9         3

```

Листинг 2: Терминал 1. Snapshot

```

1 SQL> set transaction snapshot;
2 Commit current transaction (y/n)?n
3 Rolling back work.
4 SQL> select * from temp_table;
5
6         SOMEID
7 =====
8         1
9         2

```

Листинг 3: Терминал 2. Snapshot

Как видно из лога терминала 2, новой записи в таблице не появилось, тогда как в терминале 1, новое значение успешно вывелось.

## Snapshot table stability

Аналогичен snapshot, за исключением того, что блокируются на запись данные, задействованные в операции. Таблицы, изменяемые данной транзакцией, недоступны другим транзакциям, а изменяемые другими транзакциями - недоступны в рамках данной транзакции.

```

1 SQL> select * from temp_table;
2
3         SOMEID
4 =====
5         3
6         4
7         1
8         2
9
10 SQL> UPDATE TEMP_TABLE SET SOMEID=5 WHERE SOMEID=1;
11 SQL> select * from temp_table;
12
13         SOMEID
14 =====
15         3
16         4
17         5
18         2
19
20 SQL> COMMIT;
21 SQL> UPDATE TEMP_TABLE SET SOMEID=1 WHERE SOMEID=5;
22 SQL> COMMIT;
23 SQL> select * from temp_table;
24
25         SOMEID
26 =====

```

```

27      3
28      4
29      1
30      2

```

Листинг 4: Терминал 1. Snapshot table stability

```

1  SQL> set transaction isolation level snapshot table stability;
2  Commit current transaction (y/n)?n
3  Rolling back work.
4  SQL> select * from temp_table;
5
6      SOMEID
7  =====
8          3
9          4
10         1
11         2
12
13 SQL> UPDATE TEMP_TABLE SET SOMEID=6 WHERE SOMEID=1;
14 Statement failed, SQLSTATE = 40001
15 deadlock
16 -update conflicts with concurrent update
17 -concurrent transaction number is 64
18 SQL> UPDATE TEMP_TABLE SET SOMEID=6 WHERE SOMEID=1;
19 SQL> select * from temp_table;
20
21      SOMEID
22  =====
23          3
24          4
25          6
26          2

```

Листинг 5: Терминал 2. Snapshot table stability

Видим, что результаты соответствуют ожиданиям. Изменяемая в транзакции с уровнем изоляции snapshot table stability целиком блокируется для всех остальных транзакций до окончания выполнения транзакции.

## Read committed

Уровень изоляции 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 транзакциях и этого уровня изоляции.

```

1  SQL> SELECT * FROM TEMP_TABLE;
2
3      SOMEID
4  =====
5          3
6          4
7          5
8          1

```

```

9          2
10
11 SQL> insert into temp_table values(6);
12 SQL> commit;

```

Листинг 6: Терминал 1. Read committed

```

1 SQL> SELECT * FROM TEMP_TABLE;
2
3      SOMEID
4  =====
5           3
6           4
7           5
8           1
9           2
10
11 SQL> SET TRANSACTION READ COMMITTED;
12 Commit current transaction (y/n)?n
13 Rolling back work.
14 SQL> SELECT * FROM TEMP_TABLE;
15
16      SOMEID
17  =====
18           3
19           4
20           5
21           1
22           2
23           6

```

Листинг 7: Терминал 2. Read committed

Как видно из логов, в терминале 1 произошла вставка данных в таблицу. Терминал 2 увидел изменения сразу-же после подтверждения транзакции в терминале 1.

## Вывод

Механизм транзакции незаменим при работе с крупными базами данных. Он позволяет поддерживать целостность данных при параллельной работе нескольких клиентов с базой данных. Достоинства транзакций:

1. Механизм транзакций позволяет обеспечить логическую целостность данных в БД. Другими словами, транзакции – логические единицы работы, после выполнения которых БД остается в целостном состоянии.
2. Транзакции являются единицами восстановления данных. Восстанавливаясь после сбоев, система ликвидирует следы транзакций, не успевших успешно завершиться в результате программного или аппаратного сбоя.
3. Механизм транзакций обеспечивает правильность работы в многопользовательских системах при параллельном обращении нескольких пользователей к одним и тем же данным.

Недостатки и проблемы: 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 изменёнными (другой транзакцией)
3. Если транзакция не завершается в момент когда она должна быть завершена, то соответственно записи блокируются на лишнее время, другой поток не может их изменить.