Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчет по лабораторной работе № 6

“Изучение механизма транзакций”

по дисциплине "Базы данных"

Группа: 43501/3
Студент: Кирпиченков П.С.
Преподаватель: Мяснов А.В.

Санкт-Петербург
2016

1. Постановка задачи

- 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.

2. Теоретические положения

Транзакция – одна или несколько единых по смыслу операций, которые должны либо выполниться все, либо не выполняться совсем. Транзакции нужны, когда семантически неразделимые операции не являются атомарными с точки зрения СУБД.

Транзакции должны отвечать аббревиатуре ACID – Atomicity, Consistency, Isolation, Durability, т.е. быть неделимыми, непротиворечивыми, изолированными, а внесенные ими изменения должны быть долговечными.

Выделяется несколько уровней изоляции транзакции в зависимости от ее взаимодействия с другими транзакциями:

- Dirty read – транзакция может читать незафиксированные изменения других транзакций. Не поддерживается в Firebird.
- Read commited – чтение только зафиксированных изменений других транзакций.
- Repeatable read (snapshot в Firebird) – повторяемое чтение, т.е. на время транзакции создается слепок состояния базы, на который не могут повлиять другие транзакции. Firebird также поддерживает режим snapshot table stability – в этом режиме транзакция блокирует для записи таблицы, которые использует.
- Serializable – данный уровень гарантирует возможность параллельного выполнения транзакций, т.е. их одновременное выполнение будет эквивалентно последовательному. Не поддерживается в Firebird.

Общий синтаксис транзакции:

```
SET TRANSACTION [NAME transaction]
 [READ WRITE | READ ONLY]
 [WAIT | NO WAIT]
 [[ISOLATION LEVEL] {SNAPSHOT [TABLE STABILITY]
 | READ COMMITTED [[NO] RECORD_VERSION]}]
 [RESERVING <reserving_clause>
 | USING dbhandle [, dbhandle ...]];
 <reserving_clause> = table [, table ...]
 [FOR [SHARED | PROTECTED] {READ | WRITE}] [, <reserving_clause>]
```

Для транзакции можно указать:

- режим доступа (только чтение или чтение/запись)
- наличие/отсутствие ожидания – во втором случае при возникновении конфликтов сразу выдается сообщение об ошибке
- уровень изоляции – выбор одного из трех режимов, поддерживаемых Firebird

- резервируемые таблицы – таблицы, для которых заранее указываются ограничения по их использованию другими транзакциями

3. Выполнение и результаты

Для проверки взаимодействия транзакций с различными уровнями изоляции был проведен ряд опытов.

- Проверка транзакции с доступом только для чтения

```
SQL> set transaction read only no wait read committed;
// попытка записать данные
SQL> insert into country (name) values ('Denmark');
Statement failed, SQLSTATE = 42000
attempted update during read-only transaction
// чтение данных
SQL> select name from country;

NAME
============================================================
Russia
USA
Germany
UK
Belgium
Canada
temp
JP
```

Транзакция допускает выборку данных из таблицы, но не допускает внесение изменений.

- Проверка работы транзакции с уровнем изоляции snapshot

```
// в первой консоли устанавливается уровень snapshot
SQL> set transaction read write no wait snapshot;
// чтение данных до внесения изменений во второй консоли
SQL> select name from country;

NAME
============================================================
Russia
USA
Germany
UK
Belgium
Canada
temp
JP

//чтение данных после внесения изменений во второй консоли (см. далее)
SQL> select name from country;

NAME
============================================================
Russia
USA
Germany
UK
Belgium
Canada
temp
JP

SQL> commit;
```

```
// чтение данных после коммита и переключения на установленный по умолчанию режим read commited
SQL> select name from country;

NAME
================================================================================
Russia
USA
Germany
UK
Belgium
Canada
temp
JP
Denmark

SQL> commit;

// запрос во второй консоли – вводился после первой выборки в транзакции snapshot
SQL> insert into country (name) values ('Denmark');
SQL> select name from country;

NAME
================================================================================
Russia
USA
Germany
UK
Belgium
Canada
temp
JP
Denmark

SQL> commit;
```

Как видно из эксперимента, изменения других транзакций не видны для транзакции с уровнем изоляции snapshot.

- Проверка уровня snapshot table stability

```
SQL> set transaction read write no wait snapshot table stability;
SQL> select name from country;

NAME
================================================================
Russia
USA
Germany
UK
Belgium
Canada
temp
JP
Denmark
// действия в другой транзакции проводились в этот момент
SQL> insert into country (name) values ('Poland');
SQL> commit;

// во второй консоли в процессе выполнения первой транзакции
SQL> set transaction read write no wait read committed;
SQL> insert into country (name) values ('Poland');
Statement failed, SQLSTATE = 40001
lock conflict on no wait transaction
-Acquire lock for relation (COUNTRY) failed
SQL> insert into country (name) values ('Poland');
Statement failed, SQLSTATE = 23000
violation of PRIMARY or UNIQUE KEY constraint "INTEG_6" on table "COUNTRY"
-Problematic key value is ("NAME" = 'Poland')
```

Видим, что таблица была заблокирована операцией чтения. После коммита в первой транзакции чтение было разблокировано.

- Проверка параллельной записи конфликтующий значений и откатов к предыдущему состоянию

```
SQL> set transaction read write no wait read committed;
SQL> select id, name from country;

          ID NAME
============ ==========
           1 Russia
           2 USA
           3 Germany
           4 UK
           5 Belgium
           8 Canada
           7 temp
           9 JP
          10 Denmark
          11 Poland
```

```
SQL> set transaction read write no wait read committed;
SQL> select id, name from country;

          ID NAME
============ ==========
           1 Russia
           2 USA
           3 Germany
           4 UK
           5 Belgium
           8 Canada
           7 temp
           9 JP
          10 Denmark
          11 Poland

SQL> insert into country (id, name) values ('12', 'China');
```

```
SQL> insert into country (id, name) values ('12', 'Spain');
Statement failed, SQLSTATE = 23000
violation of PRIMARY or UNIQUE KEY constraint "INTEG_5" on table "COUNTRY"
-Problematic key value is ("ID" = 12)
SQL> select id, name from country
CON> ;

          ID NAME
============ ==========
           1 Russia
           2 USA
           3 Germany
           4 UK
           5 Belgium
           8 Canada
           7 temp
           9 JP
          10 Denmark
          11 Poland
Statement failed, SQLSTATE = 40001
lock conflict on no wait transaction
-deadlock
-concurrent transaction number is 3696
```

```
SQL> rollback;
```

```
SQL> insert into country (id, name) values ('12', 'Spain');
SQL> select id, name from country;
```

```
          ID NAME
============ ==================
           1 Russia
           2 USA
           3 Germany
           4 UK
           5 Belgium
           8 Canada
           7 temp
           9 JP
          10 Denmark
          11 Poland
          12 Spain

SQL> rollback;
SQL> select id, name from country;

          ID NAME
============ ==================
           1 Russia
           2 USA
           3 Germany
           4 UK
           5 Belgium
           8 Canada
           7 temp
           9 JP
          10 Denmark
          11 Poland
```

На основании опыта можно сделать следующие выводы:

- Уникальность первичного ключа проверяется вне контекста транзакции. При попытке добавить запись с повторяющимся ключом была получена ошибка, несмотря на незафиксированное состояние добавленной записи.
- После отката транзакции добавление произошло успешно.
- Откат транзакции успешно удаляет изменения, хотя во время транзакции внутри нее видны новые добавленные данные.

4. Выводы

Механизм транзакций позволяет бороться с противоречивыми данными в базе при параллельной работе нескольких клиентов. В зависимости от назначения задачи и возможных ошибок при ее совместном выполнении с другими задачами заранее устанавливаются настройки транзакции. Для предотвращения ошибок может использоваться разный набор правил и действий.

Минимальный доступный в Firebird вариант – чтение только подтвержденных данных, не допускающее появления промежуточных данных других транзакций в текущей. За счет этого внутри транзакции можно безбоязненно вносить промежуточные изменения в саму базу, зная, что они не повлияют на других клиентов до коммита. Более строгой настройкой является фиксирование состояние базы на момент начала транзакции, если для действий транзакции требуется гарантия одинаковости начальных данных на

всем времени работы. В случае, если транзакция требует гарантии присутствия в базе именно данных из взятого ею слепка, дополнительно происходит блокировка для записи другими транзакциями всех читаемых ею таблиц.

Каждый последующий режим сильнее предыдущего связывает ограничениями совместную работу транзакций. Если уровень изоляции транзакции заявлен, но не используется, то работа с базой становится менее эффективной. С другой стороны, если семантика транзакции требует гарантий стабильности данных, а необходимые меры защиты не были сделаны, могут возникнуть противоречия в данных и логические ошибки при параллельной работе с базой нескольких клиентов.