Санкт-Петербургский государственный политехнический университет

Институт информационных технологий и управления

Кафедра компьютерных систем и программных технологий

Отчет по лабораторной работе

По дисциплине «Базы данных»

«Изучение механизма транзакций**»**

Работу выполнил студент группы № 43501/4 Смирнов Л.А.

Работу принял преподаватель Мяснов А.А.

Санкт-Петербург

2015

1. Цель работы

Познакомиться с механизмом транзакций, возможностями ручного управления транзакциями, уровнями изоляции транзакций.

2. 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

3. Выполнение программы работы

3.1 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 WORK (зафиксировать транзакцию).
- Подана команда ROLLBACK WORK (откатить транзакцию).

- Произошло отсоединение пользователя от СУБД.
- Произошел сбой системы.

3.2 Провести эксперименты по запуску, подтверждению и откату транзакций

Рассмотрим на следующем примере: специально для этого введем новую базу данных. База данных будет хранить всего лишь одну таблицу – Kids. В этой таблице мы будем хранить id ребенка, его имя и количество телефонов, которое он имеет.

СКРИПТ:

```
create database 'D:\Firebird\kids.fdb' user 'SYSDBA' password 'masterkey';
commit;
connect 'D:\Firebird\kids.fdb' user 'SYSDBA' password 'masterkey';

create table kids (
id_kid int,
Kid_name varchar(256),
Device_Amt int,
check (Device_Amt >= 0)
);
```

Предположим, что необходимо совершить транзакцию – передать два телефона другому ребенку. Транзакция – процесс передачи мобильных устройств в данном случае. Если будем рассматривать любую транзакцию содержит в себе целый ряд действий. Например, рассмотрим передачу мобильных устройств от одного ребенка к другому: первое действие – уменьшение количества устройств у первого ребенка, второе действие – увеличение количества устройств у второго ребенка. И на протяжении всего этого процесса могут встречаться определенные запреты. Также, например, может произойти сбой в системе в момент передачи устройств. При этом при возникновении любой ошибки – база данных должна вернуться в изначальное состояние. Но если транзакция завершится успешно, у второго ребенка должно быть именно то количество устройств, которое было передано.

Добавим двух человек в базу данных следующим скриптом:

```
insert into KIDS (id_kid, kid_name, device_amt)
values (1, 'Egor', 4);
insert into KIDS (id_kid, kid_name, device_amt)
values (2, 'Misha', 0);
commit;
```

Получилась такая таблица:

ID_KID	KID_NAME	DEVICE_AMT
1	Egor	4
2	Misha	0

Далее совершим транзакцию по передачи двух мобильных устройств человеку с id = 2.

Update KIDS set device_amt = 2 where kid_name = ‘Misha’;

```
SQL> update KIDS set device_amt = 2 where kid_name = 'Misha';
SQL> select * from kids;

      ID_KID KID_NAME   DEVICE_AMT
============ ======== ============
           1 Egor                4
           2 Misha               2
```

Выполним команду rollback;

```
SQL> rollback;
SQL> select * from kids;

      ID_KID KID_NAME   DEVICE_AMT
============ ======== ============
           1 Egor                4
           2 Misha               0
```

Так как мы не подтвердили транзакцию при обновлении поля, то, соответственно, оно не обновление не сохранилось, и мы смогли откатиться назад командой rollback.

С помощью команды commit; подтвердим транзакцию. После коммита попробуем сделать откат транзакции. Как мы видим подтвержденную транзакцию нельзя откатить.

```
SQL> update KIDS set device_amt = 2 where kid_name = 'Misha';
SQL> commit;
SQL> select * from kids;

      ID_KID KID_NAME   DEVICE_AMT
============ ======== ============
           1 Egor                4
           2 Misha               2

SQL> rollback;
SQL> select * from kids;

      ID_KID KID_NAME   DEVICE_AMT
============ ======== ============
           1 Egor                4
           2 Misha               2
```

3.3 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 желаем 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.

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

Проверим свойство изолированности транзакций.

- **Уровень изоляции READ COMMITTED**

Для демонстрации работы откроем два экземпляра программы Firebird. В первом экземпляре попытаемся добавить нового человека в таблицу KIDS не подтверждая данные действия коммитом.

```
SQL> insert into KIDS (id_kid, kid_name, device_amt)
CON> values (3, 'Dima', 2);
```

Выполним второе соединение с базой данных и откроем новую транзакцию с уровнем изоляции READ COMMITED.

```
ISQL Version: WI-V2.5.4.26856 Firebird 2.5
Use CONNECT or CREATE DATABASE to specify a database
SQL> connect 'D:\Firebird\kids.fdb' user 'SYSDBA' password 'masterkey';
Server version:
WI-V2.5.4.26856 Firebird 2.5
WI-V2.5.4.26856 Firebird 2.5/XNet (ЛЕВ-ПК)/P12
WI-V2.5.4.26856 Firebird 2.5/XNet (ЛЕВ-ПК)/P12
Database:  'D:\Firebird\kids.fdb', User: SYSDBA
SQL> set transaction isolation level read committed;
Commit current transaction (y/n)?y
Committing.
```

Выполним select по всем полям таблицы KIDS – select * from kids;

Второй экземпляр установил изоляцию для транзакции READ COMMITTED, тем самым указав, что он хочет видеть только подтвержденные изменения. Но за это придется платить временем. Запрос не будет выполнен до тех пор, пока не будет завершена либо commitom, либо rollback (откатом) та транзакция, которая эту блокировку установила.

Завершим транзакцию первого экземпляра командой commit; и сразу же мы увидим, что запрос второго экземпляра выполнен.

```
SQL> select * from KIDS;

      ID_KID KID_NAME   DEVICE_AMT
============ ======== ============
           1 Egor                4
           2 Misha               2
           3 Dima                2
```

- **уровень изоляции SNAPSHOT**

Во втором экземпляре откроем транзакцию с уровнем изоляции SNAPSHOT:

```
ISQL Version: WI-V2.5.4.26856 Firebird 2.5
Use CONNECT or CREATE DATABASE to specify a database
SQL> connect 'D:\Firebird\kids.fdb' user 'SYSDBA' password 'masterkey';
Server version:
WI-V2.5.4.26856 Firebird 2.5
WI-V2.5.4.26856 Firebird 2.5/XNet (ЛЕВ-ПК)/P12
WI-V2.5.4.26856 Firebird 2.5/XNet (ЛЕВ-ПК)/P12
Database:  'D:\Firebird\kids.fdb', User: SYSDBA
SQL> set transaction isolation level snapshot;
Commit current transaction (y/n)?y
Committing.
```

В первом экземпляре также добавим строчку добавления данных в таблицу не подтверждая транзакцию коммитом:

```
SQL> insert into KIDS (id_kid, kid_name, device_amt)
CON> values (4, 'Jeka', 1);
```

Сделаем выборку из таблицы во втором экземляре:

```
SQL> select * from KIDS;

      ID_KID KID_NAME   DEVICE_AMT
============ ======== ============
           1 Egor                4
           2 Misha               2
           3 Dima                2
```

Видим, что в данной таблице показаны данные, которые содержались в таблице до выполнения транзакции в первом экземпляре.

Завершим коммитом добавление строки в первом экземпляре и увидим то же самое.

Значения записей остались без изменений, несмотря на то, что транзакция во втором экземпляре была завершена.

- **уровень изоляции SNAPSHOT TABLE STABILITY**

В первом экземпляре добавим еще одну строчку в таблицу KIDS, COMMIT не делаем.

Во втором экземпляре создадим транзакцию с изолирующим уровнем SNAPSHOT TABLE STABILITY. Делаем аналогичную команду во втором экземпляре. Мы увидим, что мы не сможем сделать commit данной команды до тех пор, пока commit не будет сделан на соединении 1.

Как только сделали коммит на первом соединении – сразу получили доступ к записям во втором экземпляре.

В итоге получилась такая таблица:

4. Выводы

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:

- потерянное обновление — при одновременном изменении одного блока данных разными транзакциями одно из изменений теряется;

Транзакция 1	**Транзакция 2**
UPDATE KIDS SET dev_amt=10 WHERE id_kid=1;	UPDATE KIDS SET dev_amt=14 WHERE id_kid=1;

- «грязное» чтение — чтение данных, добавленных или изменённых транзакцией, которая впоследствии не подтвердится (откатится);

Транзакция 1	**Транзакция 2**
	SELECT dev_amt FROM KIDS WHERE id_kid=1;
UPDATE KIDS SET dev_amt=10 WHERE id_kid=1;	
	SELECT dev_amt FROM KIDS WHERE id_kid=1;
ROLLBACK;	

- неповторяющееся чтение — при повторном чтении в рамках одной транзакции ранее прочитанные данные оказываются изменёнными;

```
SQL> update KIDS set device_amt = 5 where id_kid = 1;
SQL> commit
CON> ;
```

```
Committing.
SQL> select * from KiDS;

      ID_KID KID_NAME   DEVICE_AMT
============ ======== ============
           1 Egor                1
           2 Misha               2
           3 Dima                2
           4 Jeka                1
           5 Masha               2
           6 Jora                3

SQL> select * from KiDS;

      ID_KID KID_NAME   DEVICE_AMT
============ ======== ============
           1 Egor                5
           2 Misha               2
           3 Dima                2
           4 Jeka                1
           5 Masha               2
           6 Jora                3
```

Транзакция 1	**Транзакция 2**
	SELECT * FROM KIDS;
UPDATE KIDS SET device_amt WHERE id_kid=1;	
COMMIT;	
	SELECT * FROM KIDS

- фантомное чтение — 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.