САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
Институт компьютерных наук и технологий
Кафедра Компьютерных Систем и Программных Технологий

Отчет по дисциплине
«Базы данных»

Изучение механизма транзакций

Работу выполнил студент группы №: 43501/3 Крутихин М.В.
Работу принял преподаватель: _____ Мяснов А.В.

Санкт-Петербург
2016 г.

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.

Ход работы.

1. Изучить основные принципы работы транзакций.

Транзакция – задача с определенными характеристиками, включающая любое количество различных операций (SLECT, INSERT, UPDATE, DELETE) над данными 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 (ROLLBACK) транзакции.
Все выполняемые операции должны удовлетворять следующим свойствам:

- Атомарность (либо выполняется все, либо ничего)
- Согласованность (в результате транзакции система переходит из одного абстрактного корректного состояния в другое)
- Изолированность (данные 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 системных сбоев.)

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

Ниже приведены результаты экспериментов по запуску, подтверждению и откату транзакций:

```
SQL > create table temp_table ( someId integer not null primary key);
SQL > commit ;
SQL > insert into temp_table values (1);
SQL > commit ;
SQL > rollback ;
SQL > select * from temp_table ;

SOMEID
============
1

SQL > insert into temp_table values (2);
SQL > select * from temp_table ;

SOMEID
============
1
2

SQL > rollback ;
SQL > select * from temp_table ;

SOMEID
============
1

SQL > insert into temp_table values (2);
SQL > savepoint one;
SQL > delete from temp_table ;
SQL > insert into temp_table values (3);
SQL > select * from temp_table ;
SQL > rollback to one;
SQL > select * from temp_table ;
SOMEID
============
1
2
```

Листинг 1.

В данном эксперименте была создана таблица temp_table. Сначала, в единственное поле этой таблице было добавлено значение – 1 и применена команда – rollback, но эта команда не сработала. (Значение осталось в таблице).

После добавление второго значения в таблицу и повторного использования rollback, это значение оттуда удалилось.

Так же был проведен эксперимент с добавлением точки сохранения, его результат виден на листинге 1.

3. Разобраться с уровнями изоляции транзакций в Firebird.

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

- Read commited – этот уровень изоляции используется, когда необходимо видеть все результаты параллельно выполняющихся (в рамках других транзакций) действий. Этот уровень изоляции гарантирует, что не будет возможности прочитать неподтвержденные данные, измененные в других транзакциях, и делает возможным прочитать подтвержденные данные.
- Snapshot – 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 – 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 ней уже изменяются в контексте других транзакций.

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

Snapshot:

```
SQL > insert into temp_table values (3);
SQL > commit ;
SQL > select * from temp_table ;

SOMEID
```

```
 ============
1
2
3
```

Терминал 1.

```
SQL > set transaction snapshot ;
Commit current transaction (y/n)?n
Rolling back work .
 SQL > select * from temp_table ;

SOMEID
============
1
2
```

Терминал 2.

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

Snapshot table stability

```
SQL > select * from temp_table ;
SOMEID
============
3
4
1
2

SQL > UPDATE TEMP_TABLE SET SOMEID =5 WHERE SOMEID =1;
SQL > select * from temp_table ;

 SOMEID
 ============
 3
 4
 5
 2

SQL > COMMIT ;
SQL > UPDATE TEMP_TABLE SET SOMEID =1 WHERE SOMEID =5;
SQL > COMMIT ;
SQL > select * from temp_table ;
SOMEID
 ============
 3
 4
 1
 2
```

Терминал 1.

```
SQL > set transaction isolation level snapshot table stability ;
Commit current transaction (y/n)?n
Rolling back work .

SQL > select * from temp_table ;

SOMEID
============
3
4
1
2

SQL > UPDATE TEMP_TABLE SET SOMEID =6 WHERE SOMEID =1;
Statement failed , SQLSTATE = 40001
deadlock
-update conflicts with concurrent update
-concurrent transaction number is 64
SQL > UPDATE TEMP_TABLE SET SOMEID =6 WHERE SOMEID =1;
SQL > select * from temp_table ;

SOMEID
============
3
4
6
2
```

Терминал 2.

Видно, 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

```
SQL > SELECT * FROM TEMP_TABLE ;

SOMEID
============
3
4
5
1
2

SQL > insert into temp_table values (6);
SQL > commit ;
```

Терминал 1.

```
SQL > SELECT * FROM TEMP_TABLE ;

SOMEID
============
3
4

5
1
2

SQL > SET TRANSACTION READ COMMITTED ;
Commit current transaction (y/n)?n
Rolling back work .
SQL > SELECT * FROM TEMP_TABLE ;

SOMEID
============
3
4
5
1
2
6
```

Терминал 2.

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.

Недостатки и 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)