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

Отчёт о лабораторной работе №7

Дисциплина: Базы данных

Тема: Изучение работы транзакций

Выполнил студент гр. 43501/1 А.А. Нагорнов

Руководитель А.В. Мяснов

Санкт -Петербург

2016

1. Цели работы

Познакомить студентов 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.

3. Ход работы

3.1. Основные принципы работы транзакций

Транзакцией называется последовательность операций, производимых над базой данных и переводящих базу данных из одного непротиворечивого (согласованного) состояния в другое непротиворечивое (согласованное) состояние. Используются для многопользовательских баз данных. То есть при сетевых запросах к ней.

Свойства:

- Atomicity(Атомарность) — транзакции атомарны, то есть либо все изменения транзакции фиксируются (commit), либо все откатываются (rollback);
- Consistency(Непротиворечивость) — транзакции не нарушают согласованность данных, то есть они переводят базу данных из одного корректного состояния в другое.
- Isolation(Изолированность) — работающие одновременно транзакции не влияют друг на друга, то есть многопоточная обработка транзакций производится таким образом, чтобы результат их параллельного исполнения соответствовал результату их последовательного исполнения;
- Durability(Долговременность) — если транзакция была успешно завершена, никакое внешнее событие не должно привести к потере совершенных ей изменений.

3.2. Эксперимент по запуску и откату транзакций

```
CREATE TABLE people (
  ID INT NOT NULL,
  PRIMARY KEY (ID)
);
insert into people values (1);
insert into people values (2);
commit;

SAVEPOINT one;
select * from people;
        ID
============
         1
         2
delete from people;
select * from people;
rollback to one;
select * from people;
        ID
============
         1
         2
```

После добавления в таблицу двух записей, была создана точка сохранения “one”. После чего из таблицы были удалены все данные. Для возврата к точке сохранения была использована команда rollback.

3.3. Уровни изоляции

Уровень изолированности транзакций — значение, определяющее уровень, при котором в транзакции допускаются несогласованные данные, то есть степень изолированности одной транзакции от другой. Более высокий уровень изолированности повышает точность данных, но при этом может снижаться количество параллельно выполняемых транзакций. С другой стороны, более низкий уровень изолированности позволяет выполнять больше параллельных транзакций, но снижает точность данных.

В Firebird уровень изоляции может быть:

* READ COMMITTED:
• RECORD_VERSION;
• NO RECORD_VERSION;
* SNAPSHOT;
* SNAPSHOT TABLE STABILITY.

Самым низким уровнем изоляции является READ COMMITTED. Только на этом уровне изоляции вид состояния базы данных, измененного в процессе выполнения транзакции, изменяется каждый раз, когда подтверждаются версии записей.

Уровень изоляции READ COMMITTED обеспечивает неповторяемое чтение и не избавляет от феномена фантомных строк. Это наиболее полезный уровень для операций реального времени над большим объемом данных, т. к. сокращается количество конфликтов данных, однако он не является подходящим для задач, которым нужен воспроизводимый вид.

* При установке RECORD_VERSION (флаг по умолчанию) сервер позволяет транзакции читать самую последнюю подтвержденную версию записи. Если транзакция имеет режим READ WRITE (чтение/запись), то она будет способна перезаписывать самую последнюю подтвержденную версию записи, если ее идентификатор (TID) более поздний, чем идентификатор транзакции, подтвердившей самую последнюю версию записи.

* При установке NO_RECORD_VERSION сервер эффективно имитирует поведение систем, которые используют двухфазную блокировку для управления параллельностью. Он блокирует для текущей транзакции чтение строки, если для нее существует изменение, ожидающее завершения. Разрешение ситуации зависит от установки разрешения блокировки.

*RECORD_VERSION

Первый терминал:

```
SQL> insert into people values (7);
SQL>
```

Второй терминал:

```
SQL> set transaction read committed;
Commit current transaction (y/n)?y
Committing.
SQL> select * from people;

          ID
============
           1
           2
           3
           4

SQL> select * from people;
```

```
SQL> select * from people;

          ID
============
           1
           2
           3
           4
           7
```

До того момента, пока в первом терминале не будет подтверждения транзакции (commit), во втором терминале мы не увидим изменение данных (будем ожидать ее завершения).

*NO_RECORD_VERSION

Изменим на втором терминале уровень изолированности:

```
SQL> set transaction read committed no record_version no wait;
Commit current transaction (y/n)?y
Committing.
SQL> select * from people;

          ID
============
           1
           2
           3
           4
           7
```

На первом терминале добавим новое значение:

```
SQL> insert into people values (9);
SQL>
```

Теперь, при обращении второго терминала к данным, появится исключение:

```
          ID
============
           1
           2
           3
           4
           7
Statement failed, SQLSTATE = 40001
lock conflict on no wait transaction
-deadlock
-concurrent transaction number is 32
SQL>
```

Snapshot - уровень изоляции по умолчанию. Он позволяет видеть неизменное состояние базы данных на момент старта транзакции. Изменения, выполненные другими транзакциями, в этой транзакции не видны. Свои изменения транзакция видит.

В первом терминале установим уровень изолированности Snapshot:

```
SQL> set transaction snapshot;
SQL> select * from people;

          ID
============
           1
           2
           3
           4
           7
           9

SQL> select * from people;

          ID
============
           1
           2
           3
           4
           7
           9
```

Во втором терминале добавим несколько значений и сделаем подтверждение:

```
SQL> select * from people;

          ID
============
           1
           2
           3
           4
           7
           9

SQL> insert into people values(100);
SQL> insert into people values(200);
SQL> commit;
SQL>
```

В данном примере с помощью второго терминала в таблицу были записаны новые значения, но первый терминал их не увидел. Это будет до тех пор, пока во втором терминале не будет введена команда commit.

```
SQL> commit;
SQL> select * from people;

          ID
============
           1
           2
           3
           4
           7
           9
         100
         200
```

Уровень изоляции транзакции SNAPSHOT TABLE STABILITY аналогичен уровню SNAPSHOT с той лишь разницей, что в данном случае другие транзакции независимо от их уровня изоляции могут только читать данные таблиц, включенных в операции этой транзакции, но не могут их изменять.

После того как уровень изоляции установлен, для того чтобы завершить транзакцию все клиентские транзакции должны быть завершены (commit;).

На первом терминале установим уровень изоляции:

```
SQL> set transaction snapshot table stability;
Commit current transaction (y/n)?y
Committing.
SQL>
```

После данного действия, если у одного из клиентов есть незавершенные транзакции, мы не сможем завершить следующую транзакцию.

Первый терминал:

```
SQL> delete from people where ID = 300;
SQL> insert into people values (400);
SQL> select * from people;

          ID
============
           1
           2
           3
           4
           7
           9
         200
         400

SQL> commit;
```

После commit на первом терминале, на втором увидим те же данные:

```
SQL> select * from people;

          ID
============
           1
           2
           3
           4
           7
           9
         100
         200

SQL> commit;
SQL> select * from people;

          ID
============
           1
           2
           3
           4
           7
           9
         200
         400
```

Чтобы увидеть изменения, завершение транзакции нужно и на втором терминале сделать подтверждение.

4. Выводы

В данной лабораторной работе были изучены основные принципы работы транзакций и уровни их изоляции.

При параллельном выполнении нескольких транзакций между ними могут возникать различные конфликты, такие как потерянное обновление, грязное чтение, неповторяющееся чтение и фантомное чтение. Для того, чтобы избежать этих конфликтов и существуют различные уровни изоляции транзакций.