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 О.В. Горемыкина

Руководитель А.В. Мяснов

“ ” 2016 г.

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

В базах данных клиент-сервер, таких как Firebird, клиентское приложение никогда не касается данных, которые физически хранятся в страницах базы данных. Вместо этого клиентские приложения ведут общение с системой управления базой данных - с "сервером" - создавая пакеты запросов и ответов внутри транзакций.

Система управления данными, которые изменяются множеством одновременно работающих пользователей, является уязвимой со стороны большого количества проблем целостности данных, если будет работать без какого-либо контроля.

Для решения таких проблем Firebird использует модель управления, которая изолирует каждую задачу внутри уникального контекста, ограничивающего последствия, если работа такой задачи может вызвать перекрытие работы, выполненной другими задачами. Состояние базы данных не может изменяться, если существуют какие-либо конфликты.

Вызовы клиентом COMMIT или ROLLBACK являются единственно возможными способами завершения транзакции. Откат (rollback) отменит все изменения, которые были запрошены в процессе выполнения транзакции, - изменения, которые привели к исключениям, так же как и изменения, которые были успешными и не вызвали исключения.

3.2. Тесты по запуску, подтверждению и откату транзакций

Для тестирования была создана таблица temp с одним полем “tmp”, в это поле были добавлены два значения («10» и «20») после которого была создана точка сохранения save_point1.

```
create table tmp_trans (tmp int not null primary key);
commit;
insert into tmp_trans values (10);
commit;
insert into tmp_trans values (20);
select * from tmp_trans;

         TMP
============
          10
          20
```

```
SQL> delete from tmp_trans;
SQL> select * from tmp_trans;
SQL> rollback to save1;
SQL> select * from tmp_trans;

         TMP
============
          10
          20

SQL> rollback;
SQL> select * from tmp_trans;

         TMP
============
          10
```

Были удалены все данные из таблицы. Затем, вернувшись к точке сохранения и просмотрев данные в таблице убедились, что все данные вернулись. После возвращения к последнему подтверждению транзакции в таблице осталось только одно значение «20».

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.

Snapshot

Это 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.

Первый терминал:

```
SQL> insert into tmp_trans values (777);
SQL> commit;
SQL> select * from tmp_trans;
         TMP
============
          10
          20
         777
```

Второй терминал:

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

         TMP
============
          10
          20
         777
```

При подключении двух клиентов один вставил в таблицу новое значение (777), но второй клиент не видит этих изменений.

Snapshot table stability

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)?n
Rolling back work.
```

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

Просмотреть внесенные изменения, можно только после того как все другой клиент завершит транзакции:

Первый терминал:

```
SQL> delete from tmp_trans;
SQL> insert into tmp_trans values (100);
SQL> select * from tmp_trans;

         TMP
============
         100
```

Второй терминал:

```
SQL> select * from tmp_trans;

         TMP
============
          10
          20
         777
```

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

```
SQL> select * from tmp_trans;

         TMP
============
          10
          20
         777
```

Чтобы увидеть изменения, завершение транзакции нужно и на втором терминале:

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

         TMP
============
         100
```

Read commited

Уровень изолированности 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 транзакции и этого уровня изоляции.

Первый терминал:

```
SQL> set transaction read committed;
Commit current transaction (y/n)?n
Rolling back work.
SQL> select * from tmp_trans;
          TMP
============
          100
```

Второй терминал:

```
SQL> insert into tmp_trans values (15);
SQL> commit;
```

После установления уровня изоляции `read committed` на первом терминале возможно увидеть изменения данных сразу после подтверждения транзакции на втором терминале:

```
SQL> select * from tmp_trans;
          TMP
============
          100
          200
```

Record_Version

Для этого уровня изолированности можно указать один из двух значений дополнительной характеристики в зависимости от желаемого способа разрешения конфликтов: RECORD_VERSION и NO RECORD_VERSION.

• NO RECORD_VERSION (значение по умолчанию) является в некотором роде механизмом двухфазной блокировки. В этом случае транзакция не может прочитать любую запись, которая была изменена параллельной активной (неподтвержденной) транзакцией.

Если указана стратегия разрешения блокировок NO WAIT, то будет немедленно выдано соответствующее исключение. Если указана стратегия разрешения блокировок WAIT, то это приведёт к ожиданию завершения или откату конкурирующей транзакции.

• При задании RECORD_VERSION транзакция всегда читает последнюю подтверждённую версию записей таблиц, независимо от того, существуют ли изменённые и ещё не подтверждённые версии этих записей. В этом случае режим разрешения блокировок (WAIT или NO WAIT) никак не влияет на поведение транзакции при её старте.

Изменим на одном терминале уровень изолированности:

```
SQL> set transaction read committed no record_version wait;
Commit current transaction (y/n)?n
Rolling back work.
```

До того момента, когда на первом терминале не будет завершена транзакция, второй будет ожидать момента ее завершения и не увидит запрашиваемые данные:

Второй терминал:

```
SQL> insert into tmp_trans values (300);
```

Первый терминал:

```
SQL> select * from tmp_trans;
```

Сразу после того, как на втором терминале будет завершена транзакция, первый увидит измененные данные, завершив команду select:

```
SQL> select * from tmp_trans;
          TMP
============
          100
          200
          300
```

Изменим на первом терминале уровень изолированности:

```
SQL> set transaction read committed no record_version no wait;
Commit current transaction (y/n)?n
Rolling back work.
```

На втором терминале добавим данные, не завершая транзакцию:

```
SQL> insert into tmp_trans values (400);
```

При обращении первого терминала к данным таблицы, у него немедленно возникает соответствующее исключение:

```
SQL> select * from tmp_trans;
          TMP
============
           100
           200
           300
Statement failed, SQLSTATE = 40001
lock conflict on no wait transaction
-deadlock
```

4. Выводы

Уровень изоляции SNAPSHOT является довольно неудобным по причине высокой вероятности получения не обновлённых данных.

Если же требуется получить монопольный доступ к отдельным таблицам базы данных, следует выбрать SNAPSHOT TABLE STABILITY. При этом следует помнить, что при запуске такой транзакции и попытке изменить данные можно получить исключение по блокировке, если какая-либо параллельная транзакция выполнила изменения в таблице и не подтвердила транзакцию.

Для обновляющих транзакций наиболее подходящими являются два варианта набора характеристик: использование режима WAIT совместно с NO RECORD_VERSION и NO WAIT совместно с RECORD_VERSION. В первом случае полностью исключаются ошибки блокировки после отмены или подтверждения блокирующей транзакции. Во втором случае мы сразу получаем ошибку, после чего принимаем решение о дальнейших действиях.

Уровни изоляции транзакций уменьшают возможности параллельной обработки данных в таблицах, что исключает получение одним из клиентов некорректных или устаревших данных. Правильное использование уровней транзакций обеспечивает логическую целостность данных и правильность выполнения запросов клиентов.