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

Отчёт о лабораторной работе

Дисциплина: Базы данных

Тема: Изучение механизма транзакций

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

Руководитель Мяснов А.В.

Санкт –Петербург

2016

1. Цель 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

Всё в Firebird выполняется в рамках транзакций. Транзакция — логическая единица изолированной работы группы последовательных операций над базой данных. Изменения над данными остаются обратимыми до тех пор, пока клиентское приложение не выдаст серверу инструкцию COMMIT.

Каждая из транзакций может быть представлена вариантами состояний: 00-active, 01-commited, 10-roller back, 11-limbo (распределенные 2-фазные транзакции).

Оператор COMMIT подтверждает все изменения в данных, выполненные в контексте данной транзакции (добавления, изменения, удаления). Новые версии записей становятся доступными для других транзакций, и, если предложение RETAIN не используется освобождаются все ресурсы сервера, связанные с выполнением данной транзакции.

Если в процессе подтверждения транзакции возникли ошибки в базе данных, то транзакция не подтверждается. Пользовательская программа должна обработать ошибочную ситуацию и заново подтвердить транзакцию или выполнить ее откат.

Оператор ROLLBACK отменяет все изменения данных базы данных (добавление, изменение, удаление), выполненные в контексте этой транзакции. Оператор ROLLBACK никогда не вызывает ошибок. Если не указано предложение

RETAIN, то при его выполнении освобождаются все ресурсы сервера, связанные с выполнением данной транзакции.

С помощью оператора SAVEPOINT можно создать совместимую точку сохранения, к которой можно позже откатывать работу с базой данных, не отменяя все действия, выполненные с момента старта транзакции. Механизмы точки сохранения также известны под термином "вложенные транзакции" ("nested transactions").

Если имя точки сохранения уже существует в рамках транзакции, то существующая точка сохранения будет удалена, и создаётся новая с тем же именем.

С помощью оператора ROLLBACK TO SAVEPOINT можно вернуться к нужной точке сохранения.

3.2 эксперименты по запуску, подтверждению и откату транзакций

Ниже представлен пример использования операторов commit, rollback, savepoint:

```
SQL> create table lab7 ( num int primary key);
SQL> insert into lab7 values (1);
SQL> insert into lab7 values (2);
SQL> commit;
SQL> select * from lab7;

         NUM
============
           1
           2

SQL> delete from lab7;
SQL> select * from lab7;
SQL> rollback;
SQL> select * from lab7;

         NUM
============
           1
           2

SQL> insert into lab7 values (3);
SQL> commit;
SQL> insert into lab7 values (4);
SQL> select * from lab7;

         NUM
============
           1
           2
           3
           4
```

```
SQL> delete from lab7;
SQL> select * from lab7;
SQL> rollback;
SQL> select * from lab7;

         NUM
============
           1
           2
           3

SQL> savepoint sp1;
SQL> insert into lab7 values (5);
SQL> select * from lab7;

         NUM
============
           1
           2
           3
           5

SQL> rollback to savepoint sp1;
SQL> select * from lab7;

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

3.3 Уровни изоляции транзакций в Firebird

Уровень изолированности транзакций — значение, определяющее уровень, при котором в транзакции допускаются несогласованные данные, то есть степень изолированности одной транзакции от другой. Изменения, внесённые некоторым оператором, будут видны всем последующим операторам, запущенным в рамках этой же транзакции, независимо от её уровня изолированности. Изменения, произведённые в рамках другой транзакции остаются невидимыми для текущей транзакции до тех пор, пока они не подтверждены. Уровень изолированности, а иногда, другие атрибуты, определяет, как транзакции будут взаимодействовать с другой транзакцией, которая хочет подтвердить изменения.

Snapshot

Уровень изолированности SNAPSHOT (уровень изолированности по умолчанию) означает, что этой транзакции видны лишь те изменения, фиксация которых произошла не позднее момента старта этой транзакции. Любые подтверждённые изменения, сделанные другими конкурирующими транзакциями, не будут видны в такой транзакции в процессе ее активности без её перезапуска. Чтобы

увидеть эти изменения, нужно завершить транзакцию (подтвердить её или выполнить полный откат, но не откат на точку сохранения) и запустить транзакцию заново.

Проверим работу:

Работа первого терминала:

```
SQL> select * from lab7;

         NUM
============
           1
           2
           3

SQL> insert into lab7 values (4);
SQL> select * from lab7;

         NUM
============
           1
           2
           3
           4
```

Работа второго терминала:

```
SQL> set transaction snapshot;
SQL> select * from lab7;

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

Из представленного выше, видно, второй клиент не видит изменения, сделанные первым клиентом. Это связано с тем, транзакция с уровнем Snapshot была запущена до того, как в первой транзакции произошли изменения.

Перезапустим транзакцию во втором терминале:

```
SQL> set transaction snapshot;
Commit current transaction (y/n)?y
Committing.
SQL> select * from lab7;

         NUM
============
           1
           2
           3
           4
```

Snapshot table stability

Уровень изоляции транзакции SNAPSHOT TABLE STABILITY позволяет, как и в случае SNAPSHOT, также видеть только те изменения, фиксация которых произошла не позднее момента старта этой транзакции. При этом после старта такой

транзакции в других клиентских транзакциях невозможно выполнение изменений ни в каких таблицах этой базы данных, уже каким-либо образом измененных первой транзакцией. Все такие попытки в параллельных транзакциях приведут к исключениям базы данных. Просматривать любые данные другие транзакции могут совершенно свободно.

Работа первого терминала:

```
SQL> set transaction snapshot table stability;
Commit current transaction (y/n)?y
Committing.
SQL> select * from lab7;

         NUM
============
           1
           2
           3
           4

SQL> select * from lab7;

         NUM
============
           1
           2
           3
           4

SQL> commit;
SQL> select * from lab7;

         NUM
============
           1
           2
           3
           4
           5
```

Работа второго терминала:

```
SQL> commit;
SQL> insert into lab7 values(5);
SQL> commit;
SQL> select * from lab7;

         NUM
============
           1
           2
           3
           4
           5

SQL>
```

Установим уровень изолированности SNAPSHOT TABLE STABILITY. Если во втором терминале есть незавершенные транзакции, в первом терминале будет невозможно выполнение последнего действия.

Добавим информацию в таблицу. Пока мы не сделаем commit, первый терминал не увидит содержимое таблицы. Второй терминал может производить транзакции.

После commit на первом терминале мы увидим содержимое таблицы, но до ее изменения. Сделаем commit еще раз и повторим запрос select, после чего видим обновленную таблицу.

Read committed

Для этого уровня изолированности можно указать один из двух значений дополнительной характеристики в зависимости от желаемого способа разрешения конфликтов:

• 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.

Wait:

Работа первого терминала:

```
SQL> set transaction read committed no record_version wait;
Commit current transaction (y/n)?n
Rolling back work.
SQL> select * from lab7;

         NUM
============
           1
           2
           3
           4
           5
           6
           7

SQL> select * from lab7;

         NUM
============
           1
           2
           3
           4
           5
           6
           7
           8

SQL> select * from lab7;
```

Работа второго терминала:

```
SQL> commit;
SQL> select * from lab7;

         NUM
============
           1
           2
           3
           4
           5
           6
           7

SQL> insert into lab7 values(8);
SQL> select * from lab7;

         NUM
============
           1
           2
           3
           4
           5
           6
           7
           8

SQL> commit;
SQL> insert into lab7 values(9);
SQL>
```

Видно, что первый терминал ожидает завершения транзакции на втором терминале.

No wait:

Работа первого терминала:

```
Commit current transaction (y/n)?n
Rolling back work.
SQL> select * from lab7;

         NUM
============
           1
           2
           3
           4
           5
           6
           7
           8
           9

SQL> select * from lab7;

         NUM
============
           1
           2
           3
           4
           5
           6
           7
           8
           9
Statement failed, SQLSTATE = 40001
lock conflict on no wait transaction
-deadlock
-concurrent transaction number is 72
SQL> select * from lab7;

         NUM
============
           1
           2
           3
           4
           5
           6
           7
           8
           9
          10
```

Если у второго терминала есть незавершенные транзакции, то в первом терминале при запросе select выдаются старые значения таблицы и исключения. При завершение транзакции во втором терминале, в первом при запросе select выдаются обновленные значения таблицы.

4. Выводы

Одним из наиболее распространённых наборов требований к транзакциям и транзакционным системам является набор ACID:

- Атомарность (Atomicity) гарантирует, что никакая транзакция не будет зафиксирована в системе частично. Будут либо выполнены все её подоперации, либо не выполнено ни одной.
- Согласованность (Consistency) - транзакция, достигающая своего нормального завершения и, тем самым, фиксирующая свои результаты, сохраняет согласованность базы данных. Другими словами, каждая успешная транзакция по определению фиксирует только допустимые результаты. Это условие является необходимым для поддержки этого свойства.
- Изолированность (Isolation) - во время выполнения транзакции параллельные транзакции не должны оказывать влияние на её результат.
- Долговечность (Durability) - независимо от проблем на нижних уровнях изменения, сделанные успешно завершённой транзакцией, должны остаться сохранёнными после возвращения системы в работу. Другими словами, если пользователь получил подтверждение от системы, что транзакция выполнена, он может быть уверен, что сделанные им изменения не будут отменены из-за какого-либо сбоя.

Уровни изолированности транзакции уменьшают возможность параллельной обработки информации, что исключает чтение устаревших данных.