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

Отчёт
По лабораторной работе №7
Дисциплина: базы данных
Тема: изучение работы транзакций

Выполнил студент группы 43501/1: Евсеев Е.П.
Проверил преподаватель: Мяснов А.В.

Санкт-Петербург
2016

Цели работы

Познакомить студентов с механизмом транзакций, возможностями ручного управления транзакциями, уровнями изоляции транзакций.

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.

Ход работы

1. Изучить 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.

2. Провести эксперименты по запуску, подтверждению и откату транзакций

```
SQL> create table testing (tmp int not null primary key);
SQL> commit;
SQL> insert into testing values (1);
SQL> commit;
SQL> insert into testing values (2);
SQL> commit;
SQL> select * from testing;

         TMP
============
           1
           2

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

         TMP
============
           1
           2

SQL> savepoint sv1;
SQL> insert into testing values (3);
SQL> select * from testing;

         TMP
============
           1
           2
           3

SQL> rollback to savepoint sv1;
SQL> select * from testing;

         TMP
============
           1
           2
```

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. Разобраться с уровнями 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.

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.

Терминал 1

```
SQL> insert into testing values (4);
SQL> commit;
SQL> select * from testing;

         TMP
============
           1
           2
           4

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

         TMP
============
           1
           2
           4
           5
```

Терминал 2

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

         TMP
============
           1
           2
           4

SQL> select * from testing;

         TMP
============
           1
           2
           4
```

Как мы видим, при подключении второго клиента с первого терминала ввели данные в таблицу, однако второй терминал не видит этих изменений, так как транзакция с уровнем Snapshot была запущена до того, как в первой транзакции произошли изменения.

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.

Терминал 2

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

```
SQL> select * from testing;
```

Терминал 1

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

         TMP
============
           1
           2
           4
           5
           6
           7
           8

SQL> insert into testing values (9);
SQL> select * from testing;

         TMP
============
           1
           2
           4
           5
           6
           7
           8
           9

SQL> commit;
```

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

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

После commit на втором терминале мы увидим содержимое таблицы, но до ее изменения. Сделаем commit еще раз и повторим запрос select.

Терминал 2

```
SQL> select * from testing;

         TMP
============
           1
           2
           4
           5
           6
           7

SQL> commit;
SQL> select * from testing;

         TMP
============
           1
           2
           4
           5
           6
           7
           8
           9
```

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.

Для получения обновлённого списка строк интересующей таблицы необходимо лишь повторное выполнение оператора SELECT в рамках активной транзакции READ COMMITTED без её перезапуска.

Терминал 2

```
SQL> set transaction read committed;
SQL> select * from testing;

          TMP
============
           1
           2
           4
           5
           6
           7
           8
           9

SQL> select * from testing;

          TMP
============
           1
           2
           4
           5
           6
           7
           8
           9
           0
```

Терминал 1

```
SQL> insert into testing values (0);
SQL> commit;
```

Во втором терминале установим уровень изолированности Read committed. С помощью select посмотрим данные из таблицы. Данные верны. В первом терминале добавим запись. До тех пор, пока не сделан commit, во втором терминале транзакции не видны. Сделаем commit и во втором терминале повторим запрос select. Данные опять верны.

Для уровня **RECORD_VERSION** 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.

Терминал 2

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

         TMP
============
           1
           2
           4
           5
           6
           7
           8
           9
           0
          10
```

Терминал 1

```
SQL> insert into testing values (10);
SQL> select * from testing;

         TMP
============
           1
           2
           4
           5
           6
           7
           8
           9
           0
          10
```

До того момента, когда на первом терминале не завершена транзакция, второй терминал будет ожидать завершения.

Изменим уровни изолированности на no wait.

Терминал 2

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

         TMP
============
           1
           2
           4
           5
           6
           7
           8
           9
           0
          10
Statement failed, SQLSTATE = 40001
lock conflict on no wait transaction
-deadlock
-concurrent transaction number is 47
SQL> select * from testing;

         TMP
============
           1
           2
           4
           5
           6
           7
           8
           9
           0
          10
          11
```

Терминал 1

```
SQL> insert into testing values (11);
SQL> select * from testing;

         TMP
============
           1
           2
           4
           5
           6
           7
           8
           9
           0
          10
          11

SQL> commit;
```

Если у первого терминала есть незавершенные транзакции, во втором терминале при запросе select выдаются старые значения таблицы и исключение. При commit в первом терминале и повторном запросе select во втором терминале, выдаются правильные данные таблицы.

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.