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 Калугина М.О.

Руководитель Мяснов А.В.

“ ” 2016 г.

Санкт -Петербург

2016

1. **Цели работы**

Ознакомиться с возможностями реализации более сложной обработки данных на стороне сервера с помощью хранимых процедур.

# Программа работы

* Изучить основные принципы работы транзакций
* Провести эксперименты по запуску, подтверждению и откату транзакций
* Разобраться с уровнями изоляции транзакций в Firebird
* Спланировать и провести эксперименты, показывающие основные возможности транзакций с различным уровнем изоляции

1. **Основные принципы работы транзакции**

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.

1. **Эксперименты по запуску, подтверждению и откату транзакции**

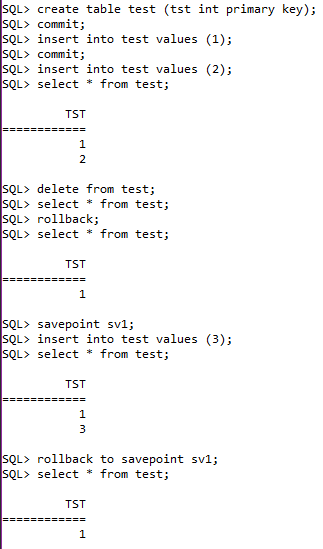


Рис.1. Использование операторов commit, rollback и savepoint

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.

## 5. Уровни изоляции

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 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 Snapshot на двух терминалах:

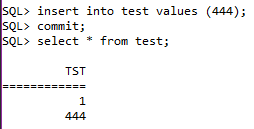


Рис.2. Работа первого терминала

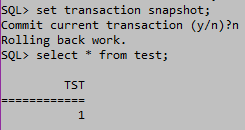


Рис.3. Работа второго терминала

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

Перезапустим транзакцию во втором терминале и увидим наши изменения:

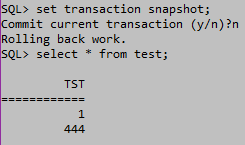


Рис.4. Работа второго терминала

**Уровень изолированности 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.

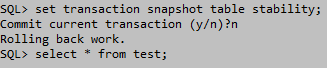


Рис.5. Работа второго терминала

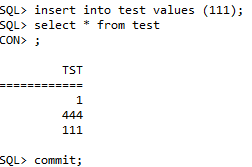


Рис.6. Работа первого терминала

Установим уровень изолированности SNAPSHOT TABLE STABILITY. Если в первом терминале есть незавершенные транзакции, во втором терминале будет невозможно выполнение последнего действия (Рис.5.).

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.

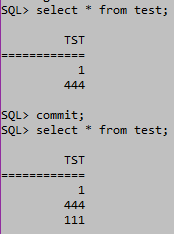


Рис.7. Работа второго терминала

**Уровень изолированности Read committed**

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.

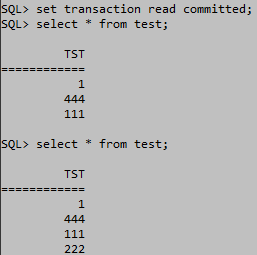


Рис.8. Работа второго терминала



Рис.9. Работа первого терминала

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.

**Record\_version**

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.

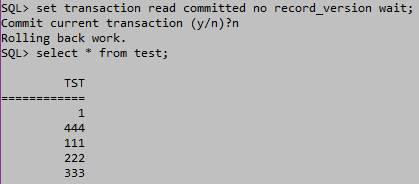


Рис.10. Работа второго терминала

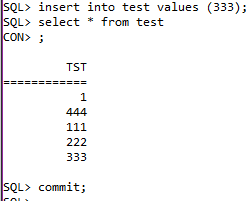


Рис.11. Работа первого терминала

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

Изменим уровни изолированности на no wait:

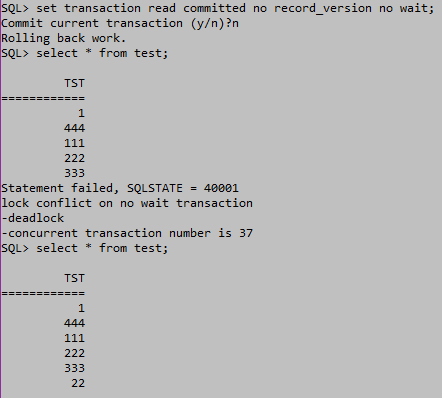


Рис.12. Работа второго терминала

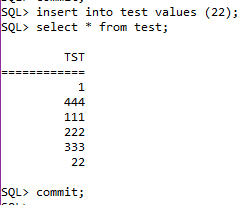


Рис.13. Работа первого терминала

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.