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 гр. 43501/1 С.В. Смирнов

Руководитель А.В. Мяснов

“ ” 2016 г.

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.
5. Продемонстрировать результаты преподавателю, ответить на контрольные вопросы.

# Ход работы

## Основные принципы работы транзакций

Любая база данных годна к использованию только тогда, когда ее состояние соответствует состоянию предметной области. Такие состояния называют целостными. Очевидно, что при изменении данных БД должна переходить от одного целостного состояния к другому. Однако, в процессе обновления данных возможны ситуации, когда состояние целостности нарушается.

Во избежание таких ситуаций в СУБД вводится понятие **транзакции** - атомарного действия над БД, переводящего ее из одного целостного состояния в другое целостное состояние. Другими словами, транзакция - это последовательность операций, которые должны быть или все выполнены, или все не выполнены (все или ничего).

Методом контроля за транзакциями является ведение *журнала*, в котором фиксируются все изменения, совершаемые транзакцией в БД. Если во время обработки транзакции происходит сбой, транзакция откатывается - из журнала восстанавливается состояние БД на момент начала транзакции.

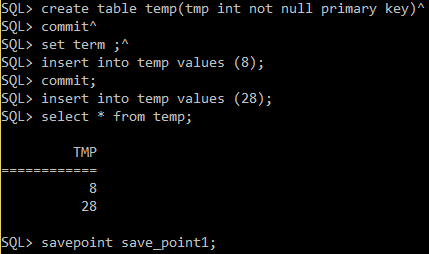
В СУБД различных поставщиков начало транзакции может задаваться явно (например, командой **BEGIN TRANSACTION**), либо предполагаться неявным (так определено в стандарте SQL), т.е. очередная транзакция открывается автоматически сразу же после удачного или неудачного завершения предыдущей. Для завершения транзакции обычно используют команды SQL:

**COMMIT** - успешно завершить транзакцию

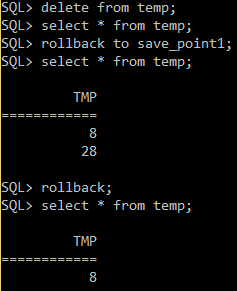
**ROLLBACK** - откатить транзакцию, т.е. вернуть БД в состояние, в котором она находилась на момент начала транзакции.

## Тесты по запуску, подтверждению и откату транзакций

Для тестирования была создана таблица temp с одним полем “tmp”, в это поле были добавлены два значения («8» и «28») после которого была создана точка сохранения save\_point1.



Были удалены все данные из таблицы, затем, вернувшись к точке сохранения, просмотрев данные в таблице мы увидели, что все данные вернулись. Потом, вернувшись к последнему подтверждению транзакции, в таблице осталось только одно значение «8».



## Уровни изоляции

Язык SQL также предоставляет способ косвенного управления скоростью выполения транзакций с помощью указания *уровня изоляции* транзакции. Под уровнем изоляции транзакции понимается возможность возникновения одной из описанных выше ошибочных ситуаций. В стандарте SQL определены 4 уровня изоляции:

**SNAPSHOT** (константа concurrency) – уровень изоляции, не допускающий фантомов. Все операции в транзакции с данным уровнем изоляции видят только те данные, которые существовали (committed) на момент старта этой транзакции (даже если они впоследствии были изменены или удалены другими транзакциями). По умолчанию в IB/FB API.

**SNAPSHOT TABLE STABILITY** (константа consistency) – изолированность образа (воспроизводимое чтение), при обращении к таблицам блокирует к ним доступ (как минимум на изменения, целиком для всей таблицы, см. дальше резервирование таблиц).

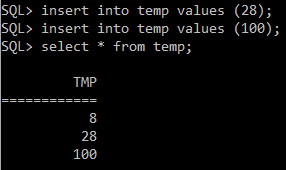
**READ COMMITTED** (константа read\_committed) –в данной транзакции все изменения, которые были подтверждены другими транзакциями, будут видны немедленно. Имеет две опции:

* **NO RECORD\_VERSION** (константа no\_rec\_version) – если при чтении пакета версий записи обнаруживается non-committed версия, то выдается или deadlock (в режиме no wait) или транзакция зависает на блокировке (в режиме wait). По умолчанию для режима READ COMMITTED в IB API.
* **RECORD\_VERSION** (константа REC\_VERSION) – игнорирует non-committed версии, читая последнюю committed-версию (см. выше no\_rec\_version). Именно этот режим является умолчательным в BDE, и рекомендуется для нормальной работы в режиме read committed.

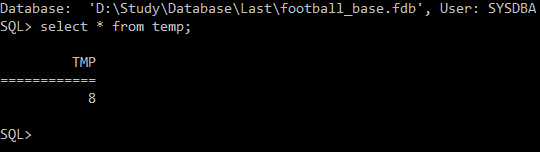
**Snapshot**

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:



Второй терминал:



По результатам работы видно, что при подключении двух клиентов один занес в таблицу два значения: 28 и 100, но второй клиент не видит этих изменений, пока транзакция не была завершена.

**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;).

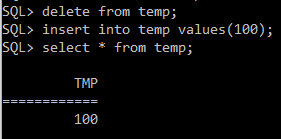
На первом терминале установим уровень изоляции:



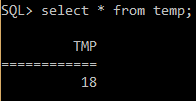
После данного действия, если у одного из клиентов есть незавершенные транзакции, мы не сможем завершить следующую транзакцию.

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

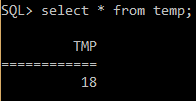
Первый терминал:



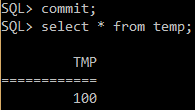
Второй терминал:



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



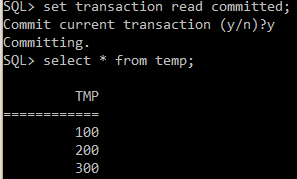
Чтобы увидеть изменения, завершение транзакции нужно и на втором терминале (после *commit;* на втором терминале):



**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.

Первый терминал:



Второй терминал:



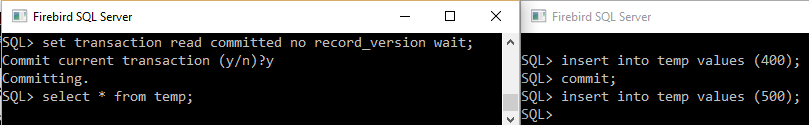
После установления уровня изоляции **read committed**, на первом терминале возможно увидеть изменения данных сразу после подтверждения транзакции на втором терминале.

**Record\_Version**

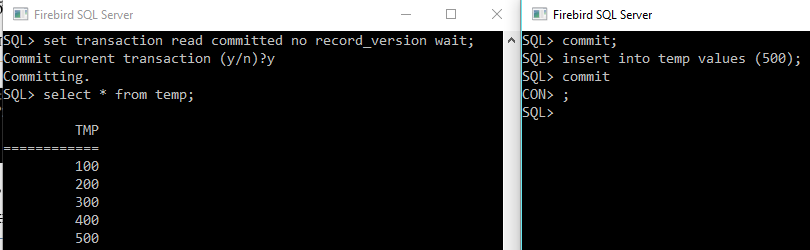
Изменим на одном терминале уровень изолированности:



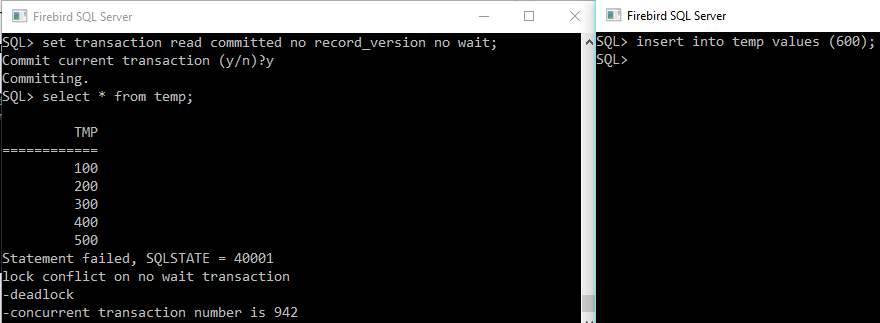
До того момента, когда на первом терминале не будет завершена транзакция, второй будет ожидать момента ее завершения и не увидит запрашиваемые данные:



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



Изменим на первом терминале уровень изолированности:



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

# 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.

Чем больше уровень изоляции транзакции – тем более изолированы транзакции друг от друга. При самом высоком уровне изоляции параллелизм в выполнении транзакций становится невозможен, так как любое параллельное выполнение транзакций может вызвать конфликт.

При работе с базой данных необходимо выбирать самый подходящий уровень изоляции транзакций, который будет золотой серединой между производительностью и надёжностью транзакций.