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. 43501/3 Никитенко А.П.

(подпись)

Руководитель А.В. Мяснов

(подпись)

“ ” 2016 г.

Санкт-Петербург

2016

**Цель работы**

Познакомить студентов с языком создания запросов управления данными SQL-DML.

**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.

**Транзакции**

Транзакция - это неделимая, с точки зрения воздействия на СУБД, последовательность операций манипулирования данными. Для пользователя транзакция выполняется по принципу "*все или ничего*", т.е. либо транзакция выполняется целиком и переводит базу данных из одного *целостного состояния* в другое *целостное состояние*, либо, если по каким-либо причинам, одно из действий транзакции невыполнимо, или произошло какое-либо нарушение работы системы, база данных возвращается в исходное состояние, которое было до начала транзакции (происходит откат транзакции). С этой точки зрения, транзакции важны как в многопользовательских, так и в однопользовательских системах. В однопользовательских системах транзакции - это логические единицы работы, после выполнения которых база данных остается *в целостном состоянии*. Транзакции также являются *единицами восстановления* данных после сбоев - восстанавливаясь, система ликвидирует следы транзакций, не успевших успешно завершиться в результате программного или аппаратного сбоя. Эти два свойства транзакций определяют атомарность (неделимость) транзакции. В многопользовательских системах, кроме того, транзакции служат для обеспечения *изолированной* работы отдельных пользователей - пользователям, одновременно работающим с одной базой данных, кажется, что они работают как бы в однопользовательской системе и не мешают друг другу.

Транзакция обладает четырьмя важными свойствами, известными как ***свойства АСИД***:

* ***(А) Атомарность***. Транзакция выполняется как атомарная операция - либо выполняется вся транзакция целиком, либо она целиком не выполняется.
* ***(С) Согласованность***. Транзакция переводит базу данных из одного согласованного (целостного) состояния в другое согласованное (целостное) состояние. Внутри транзакции согласованность базы данных может нарушаться.
* ***(И) Изоляция***. Транзакции разных пользователей не должны мешать друг другу (например, как если бы они выполнялись строго по очереди).
* ***(Д) Долговечность***. Если транзакция выполнена, то результаты ее работы должны сохраниться в базе данных, даже если в следующий момент произойдет сбой системы.

Транзакция обычно начинается автоматически с момента присоединения пользователя к СУБД и продолжается до тех пор, пока не произойдет одно из следующих событий:

* Подана команда COMMIT WORK (зафиксировать транзакцию).
* Подана команда ROLLBACK WORK (откатить транзакцию).
* Произошло отсоединение пользователя от СУБД.
* Произошел сбой системы.

Команда COMMIT WORK завершает текущую транзакцию и автоматически начинает новую транзакцию. При этом гарантируется, что результаты работы завершенной транзакции фиксируются, т.е. сохраняются в базе данных.

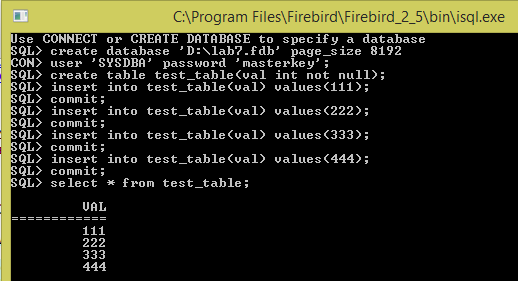
Команда ROLLBACK WORK приводит к тому, что все изменения, сделанные текущей транзакцией откатываются, т.е. отменяются так, *как будто их вообще не было*. При этом автоматически начинается новая транзакция.

При отсоединении пользователя от СУБД происходит автоматическая фиксация транзакций.

При сбое системы происходят более сложные процессы. Кратко суть их сводится к тому, что при последующем запуске системы происходит анализ выполнявшихся до момента сбоя транзакций. Те транзакции, для которых была подана команда COMMIT WORK, но *результаты работы которых не были занесены в базу данных* выполняются снова (накатываются). Те транзакции, для которых не была подана команда COMMIT WORK, откатываются. Более подробно восстановление после сбоев рассматривается далее.

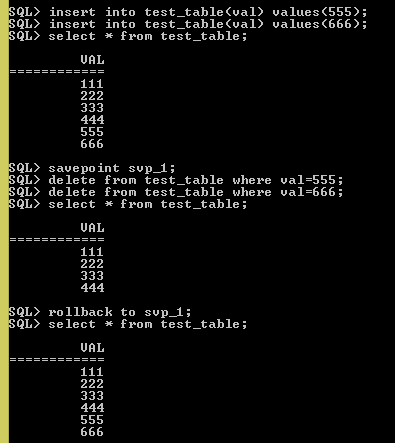
**Провести эксперименты по запуску, подтверждению и откату транзакций.**

Для проведения данных экспериментов была создана БД, состоящая из одной таблицы с одним полем типа INT.



Заполнили таблицу некоторыми данными.

Теперь исследуем возможность создания точки сохранения. Для этого добавим еще данные, создадим точку восстановления, удалим новые данные и вернемся к точке восстановления. Удаленные данные должны будут восстановиться.



Как видно на рисунке, после отката к точке восстановления. удаленные данные были восстановлены.

**Уровни изоляции транзакции.**

Важнейшая характеристика транзакции — **уровень изоляции**. В следующей таблице представлены три существующих уровня изоляции транзакции.

**Уровень изолированности транзакций** — значение, определяющее уровень, при котором в [транзакции](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D0%B7%D0%B0%D0%BA%D1%86%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) 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.

В Firebird поддерживаются следующие уровни изоляции:

1. Read commited - Чтение подтвержденных изменений. Транзакция может видеть самые последние подтвержденные изменения базы данных, выполненные другими транзакциями. При этом уровне изоляции используются еще два взаимоисключающих параметра:

* По умолчанию NO RECORD\_VERSION требует, чтобы было выполнено подтверждение всех измененных другими транзакциями данных.
* RECORD\_VERSION позволяет читать самую последнюю подтвержденную версию изменений, даже если существуют другие неподтвержденные версии.

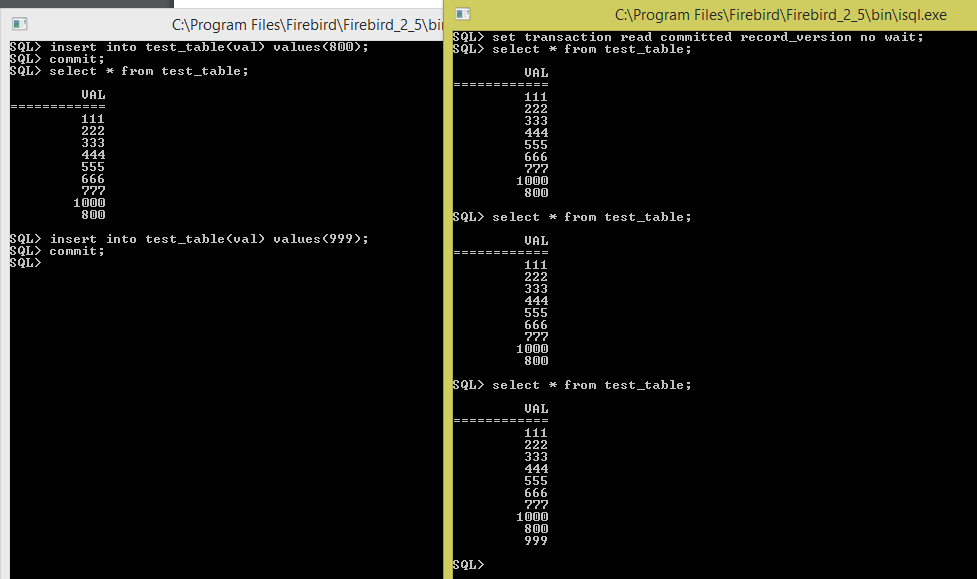
2. Мгновенный снимок (образ). Значение по умолчанию. Другое название - повторяемое чтение (Repeatable Read/Snapshot). Дает состояние базы данных на момент старта транзакции. Изменения, выполненные другими транзакциями, в данной транзакции не видны. Естественно, транзакция «видит» все изменения, выполненные в контексте этой транзакции.

3. Изолированный образ или упорядочиваемый, сериализуемый (Serializable/Snapshot table stability) образ. Аналогичен уровню SNAPSHOT с тем отличием, что другим транзакциям разрешено чтение данных из таблиц данной транзакции, однако они не могут вносить в них никаких изменений.

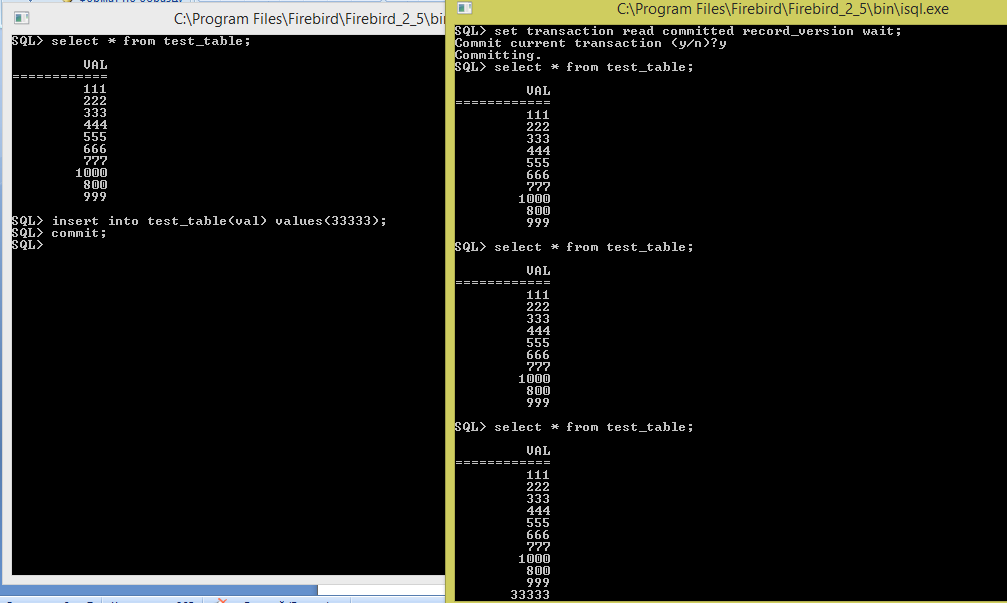
**Спланировать и провести эксперименты, показывающие основные возможности транзакций с различным уровнем изоляции.**

**Уровень READ COMMITED.**

*Флаг RECORD\_VERSION.*

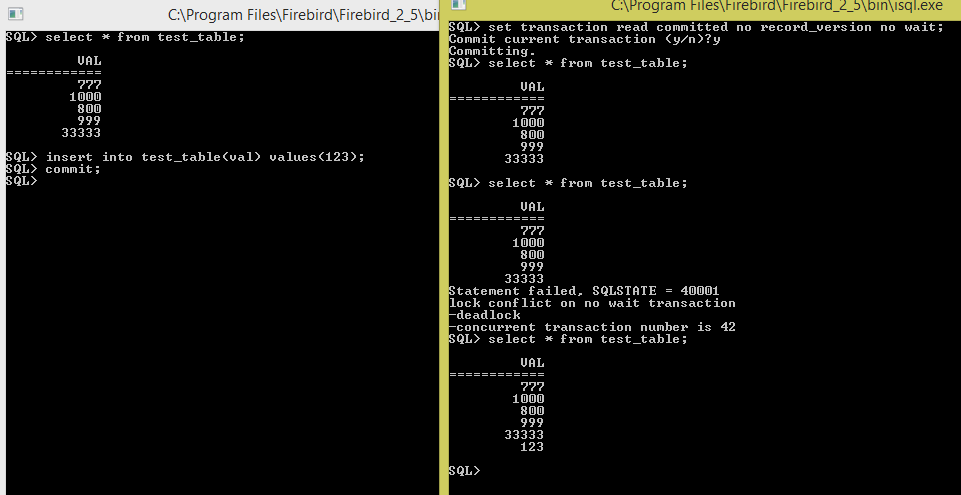


При добавлении данных первым клиентов в таблицу, второй клиент не видит новые данные, пока первый не подтвердит транзакцию. После подтверждения транзакции первым клиентом, второй видит обновленные данные.



При флаге RECORD\_VERSION режим разрешения блокировок на работу транзакций никак не повлиял.

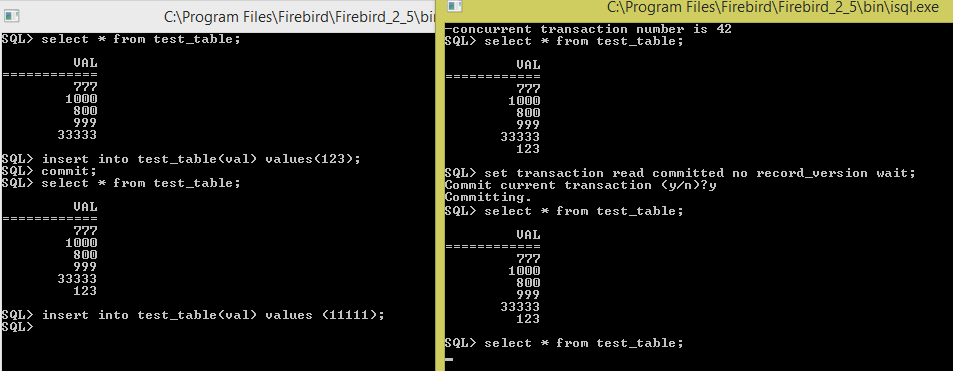
*Флаг NO RECORD\_VERSION.*



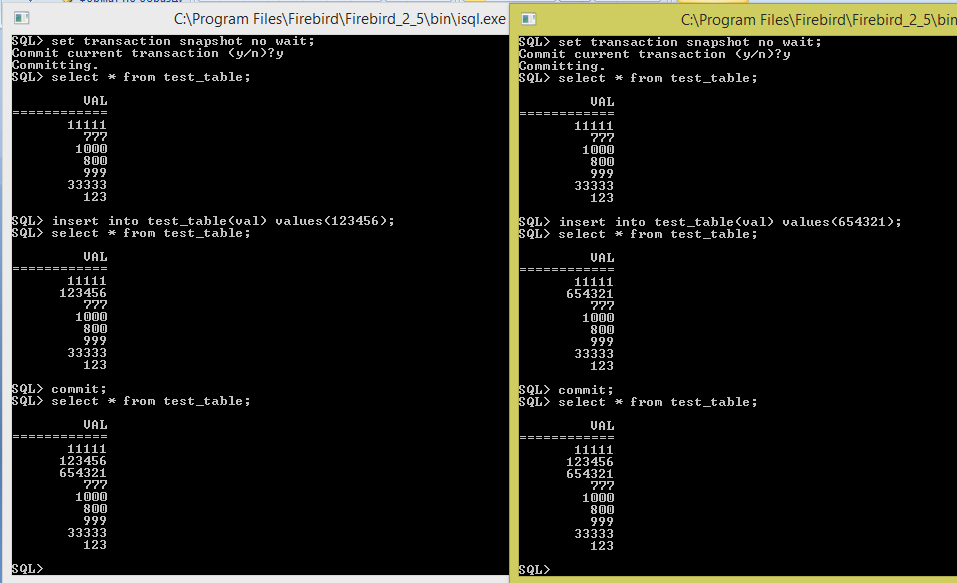
Режим разрешения блокировок no wait.

При флаге NO RECORD\_VERSION и установленном режиме блокировок no wait после внесения изменений в таблице первым клиентом, второй клиент не может посмотреть ни обновленные данные, ни последние подтвержденные данные.

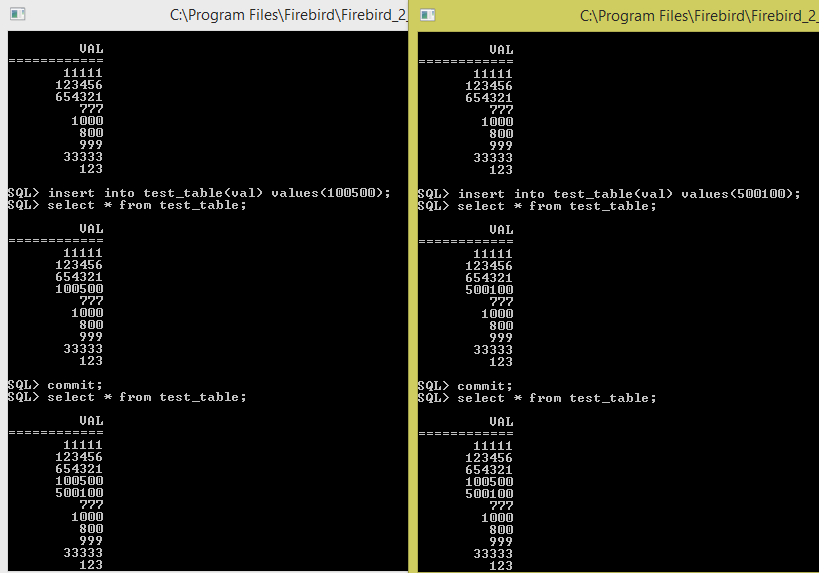
После подтверждения изменений первым клиентом, второй клиент смог посмотреть обновленную и подтвержденную информацию.

При режиме разрешения блокировок wait, после внесения изменений первым клиентом, второй клиент остается в "подвешенном" состоянии, ожидая подтверждения изменений первым клиентом. После подтверждения изменений первым клиентом, второй клиент видит обновленные и подтвержденные данные.

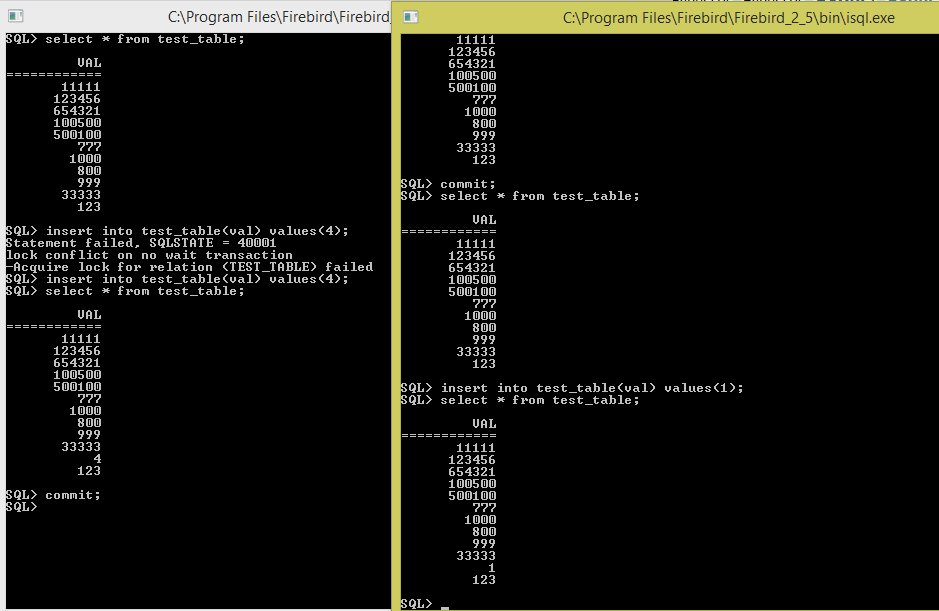
**Уровень SNAPSHOT.**

Сделаем изменения в таблице test\_table каждым из клиентов. Пока клиенты не подтвердили изменения, они могут видеть транзакции, совершенные другим клиентом. После подтверждения транзакций, оба клиента видят также данные, записанные другим клиентом.

Режим разрешения блокировок на результат работы транзакций не повлиял.



**Уровень SNAPSHOT TABLE STABILITY.**



Сначала внесли новую запись в таблицу первым клиентом. Первый клиент видит внесенные изменения до совершения подтверждения, второй клиент наоборот, не видит внесенные первым клиентом изменения. Кроме того, второй клиент не может сам внести никакие изменения в таблицу, он остается в "подвешенном" состоянии, пока первый не подтвердит транзакцию. После того, как первый подтвердил транзакцию, второй клиент смог внести свои данные. Клиенты видят все обновленные данные только после того, как каждый из них подтвердил все свои транзакции.

**Вывод**

В данной лабораторной работе были изучены основные принципы работы транзакций, уровни изоляции.

Уровни изоляции транзакций позволяют управлять параллельной работой с данными. Например, при уровне изоляции snapshot table stability, одновременно изменять данные может только один клиент, остальным придется ждать, пока он не подтвердит транзакции. На уровнях snapshot и read committed разрешают нескольким пользователям одновременно изменять данные.