Санкт-Петербургский Государственный Политехнический Университет

Институт Информационных Технологий и Управления

Кафедра Компьютерных Систем и Программных Технологий

**Отчет**

**о лабораторной работе №6**

**Дисциплина:** Базы данных

**Тема:** Транзакции

**Выполнил**: гр.43501/1 Саитов Илья

**Преподаватель:** Мяснов А.В.

Санкт-Петербург 2015

1. **Цель работы:**

Ознакомиться с возможностями с механизмом транзакций, возможностями ручного управления транзакциями, уровнями изоляции транзакций.

1. **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.
* Продемонстрировать результаты преподавателю.

1. **Выполнение работы:**

**а) Проведем эксперименты по запуску, подтверждению и откату транзакций:**

С помощью утилиты FireBird ISQL Tool реализуем 2 сеанса подключения к нашей БД. Наблюдение, на протяжении всей работы, будем вести двухоконном режиме. С каждого окна ведем подключение к БД.

Итак, создадим первый сеанс подключения и в таблицу GroupRec (таблица справочник, которая хранит список категорий годности военнослужащих) добавим новую группу здоровья командой:

INSERT INTO GroupRec(Group\_Id, NameGroup) VALUES (10, ‘E’);

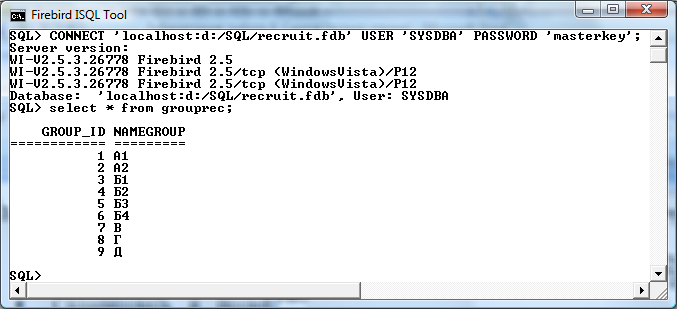


Рис.3.1. Подкючение 1 сеанса и исходное состояние таблицы GroupRec.

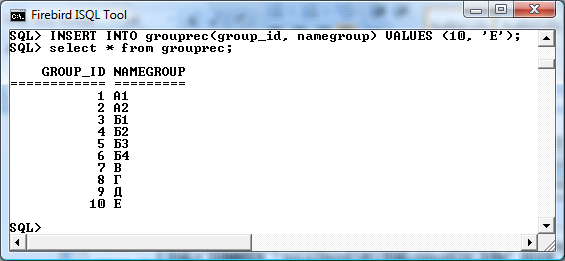


Рис.3.2. Сеанс №1. Вставка новой группы и наблюдение изменения таблицы GroupRec.

Теперь запустим второй сеанс подключения и посмотрим изменения таблицы GroupRec:

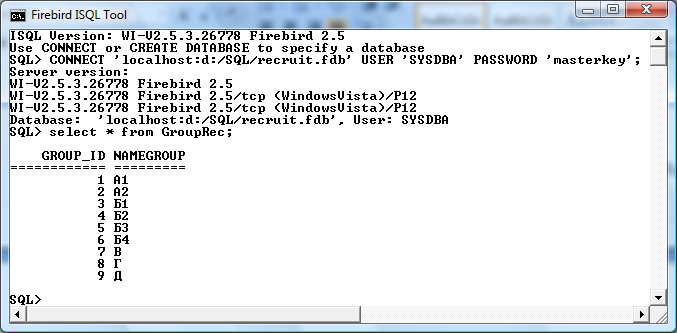


Рис.3.3. Сеанс №2. Подключение и наблюдение изменения таблицы GroupRec.

Видно, что в первом сеансе мы наблюдаем изменения, а во втором нет. Все потому, что после внесения изменения не было подтвеждения транзакций командой *commit*.

Выполним команду *rollback* в первом сеансе, что приведет к отмене изменений в первом сеансе подключения:

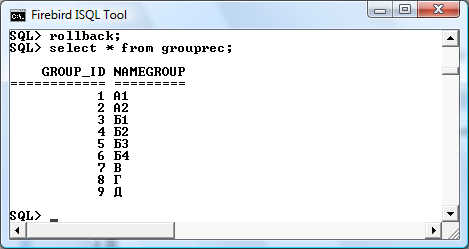


Рис.3.4. Сеанс №1. Откат изменений и наблюдение таблицы GroupRec.

Теперь, выполним ту же операцию добавления записи, но только в этот раз выполним команду подтвеждения транзакций *commit* в двух сеансах:

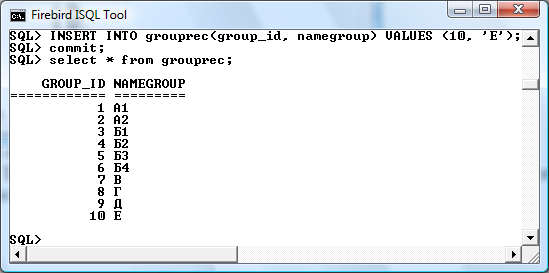


Рис.3.5. Сеанс №1. Внесение изменений, подтверждение и наблюдение таблицы GroupRec.

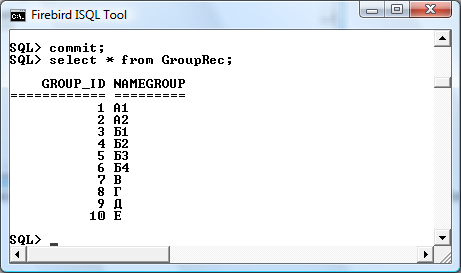


Рис.3.6. Сеанс №2. Подтверждение транзакций и наблюдение таблицы GroupRec.

Как видно, изменения теперь видны в обоих сеансах подключения. Что и требовалось показать.

**б) Уровни изоляции в FireBird:**

Транзакция — это группа последовательных операций с базой данных, которая представляет собой логическую единицу работы с данными. Транзакция может быть выполнена либо целиком и успешно, соблюдая целостность данных и независимо от параллельно идущих других транзакций, либо не выполнена вообще и тогда она не должна произвести никакого эффекта. Все зависит от уровня изоляции. Уровень изолированности транзакции определяет, какие изменения, сделанные в других транзакциях, будут видны в данной транзакции. Каждая транзакция имеет свой уровень изоляции, который устанавливается при ее запуске и остается неизменным.

Касательно в Firebird, транзакции могут иметь 3 основных возможных уровня изоляции: *READ COMMITTED*, *SNAPSHOT* и *SNAPSHOT* *TABLE* *STABILITY*. Каждый из этих трех уровней изоляции определяет правила видимости тех действий, которые выполняются другими транзакциями:

• **READ COMMITTED** (с анлг. "читать подтвержденные данные"). Уровень изоляции READCOMMITTED используется, когда мы хотим видеть все подтвержденные результаты параллельно выполняющихся (т. е. в рамках других транзакций) действий. Этот уровень изоляции гарантирует, что мы не сможем прочитать неподтвержденные данные, измененные в других транзакциях.

• **SNAPSHOT** (с анлг. "моментальный снимок"). Этот уровень изоляции используется для создания "моментального" снимка базы данных. Все операции чтения данных, выполняемые в рамках транзакции с уровнем изоляции SNAPSHOT, будут видеть только состояние базы данных на момент начала запуска транзакции. Все изменения, сделанные в параллельных транзакциях, не видны в этой транзакции. В то же время SNAPSHOT не блокирует данные, которые он не изменяет.

• **SNAPSHOT TABLE STABILITY**. Данный уровень изоляции также создает "моментальный" снимок базы данных, но одновременно блокирует на запись и чтение данные, задействованные в операциях, выполняемые данной транзакцией. Это означает, что если транзакция SNAPSHOT TABLE STABILITY изменила данные в какой-нибудь таблице, то после этого данные в этой таблице уже не могут быть изменены в других параллельных транзакциях. Кроме того, транзакции с уровнем изоляции SNAPSHOT TABLE STABILITY не могут получить доступ к таблице, если данные в них уже изменяются в контексте других транзакций.

**в) Проведем эксперименты, показывающие основные возможности транзакций с различным уровнем изоляции:**

**1) Изоляция уровня READ COMMITED:**

|  |
| --- |
| Сеанс №1:  > SET transaction isolation level READ COMMITTED;  Сеанс №2:  > INSERT INTO GroupRec(Group\_Id, NameGroup) VALUES (11, ‘M’);  Сеанс №1:  > SELECT \* FROM GroupRec;  (Введя команду, никаких ответных реакций не последовало. Лишь только моргал курсор. Произошла блокировка первой сессии, до завершения подтверждения транзакции во второй сессии.)  Сеанс №2:  > commit;  Сеанс №1:  (Выполнилась команда SELECT, которая была введена выше) |

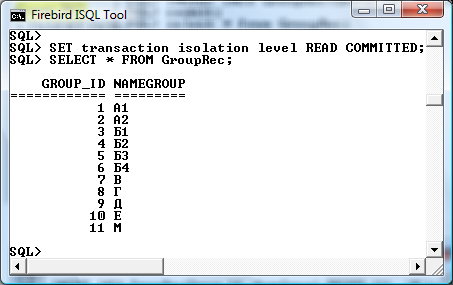


Рис.3.7. Сеанс №1. Установка изоляции READ COMMITED и попытка выполнить команду.

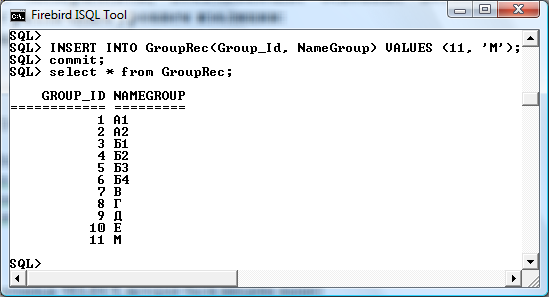


Рис.3.8. Сеанс №2. Внесение изменений и подтвержение транзакций.

Также, уровень изоляции READ COMMITTED находится в одном из двух режимов - *NO RECORD VERSION* и *RECORD VERSION*. В первом случае, если при чтении записи в БД обнаруживается наличие неподтвержденной версии этой записи, то блокируется транзакция чтения данной записи. В режиме RECORD VERSION наличие неподтвержденных версий записей игнорируется, и всегда возвращается старая версия записи. По умолчанию включен режим NO RECORD VERSION.

Повторим эксперимент, включив второй режим:

|  |
| --- |
| Сеанс №1:  > SET transaction isolation level READ COMMITTED RECORD\_VERSION;  Commit current transaction (y/n)?y  Committing.  Сеанс №2:  > INSERT INTO GroupRec(Group\_Id, NameGroup) VALUES (12, ‘T’);  Сеанс №1:  > SELECT \* FROM GroupRec;  (Вывелась неизмененная таблица).  Сеанс №2:  >commit; |

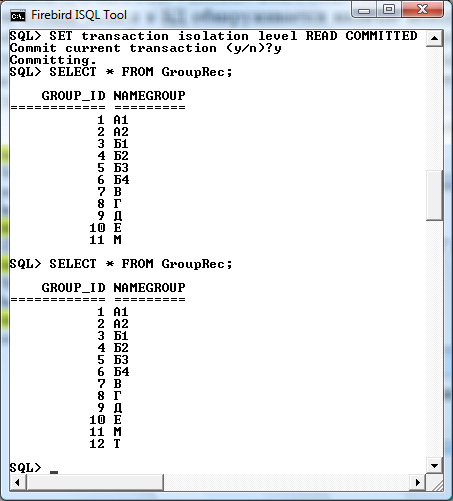
****

Рис.3.9. Сеанс №1. Установка режима READ COMMITED в RECORD\_VERSION и выполнения чтения таблицы до и после commit.

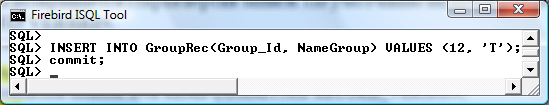
****

Рис.3.10. Сеанс №2. Внесение изменений и подтвержение транзакций.

**2) Изоляция уровня SNAPSHOT:**

|  |
| --- |
| Сеанс №1:  > SET transaction isolation level SNAPSHOT;  Commit current transaction (y/n)?y  Committing.  Сеанс №2:  > DELETE FROM GroupRec WHERE group\_id>9;  > commit;  Сеанс №1:  > SELECT \* FROM GroupRec;  ( Произошло чтение старого состояния таблицы записи, так как моментальный снимок базы данных был сделан до выплнения команды DELETE.)  Сеанс №2:  > SELECT \* FROM GroupRec;  ( Здесь вывелось новое состояние таблицы);  Сеанс №1:  > commit;  > SELECT \* FROM GroupRec;  ( Вывелось уже новое состояние таблицы). |

|  |  |
| --- | --- |
|  | Рис.3.12. Сеанс №2. Удаление записей и подтверждение транзакций. |
| Рис.3.11. Сеанс №1. Установка уровня SNAPSHOT и проверка изменений. |  |

Из результата видно, что изменения вступили в силу только после выполнения команды подтверждения транзакций на обоих сеансах подключения.

Стоит отметить, что на данном уровене изоляции может возникнуть блокировка ресурсов (Deadlock) БД, когда 2 запроса в разных транзакциях одновременно обращаются к одной записи. Промоделируем данную ситуацию:

|  |  |
| --- | --- |
| Сеанс №1:  > SET transaction isolation level READ COMMITTED RECORD\_VERSION;  Commit current transaction (y/n)?y  Committing.  Сеанс №2:  > UPDATE GroupRec SET namegroup='T' WHERE group\_id=10;  Сеанс №1:  > UPDATE GroupRec SET namegroup='M' WHERE group\_id=10;  (Окно повисло в ожидании. Транзакция ждет освобождения ресурса от другого сеанса подключения)  Сеанс №2:  >commit;  Сеанс №1:  Statement failed, SQLSTATE = 40001  deadlock  -update conflicts with concurrent update  -concurrent transaction number is 23350  > SELECT \* FROM GroupRec;  (Вывелась неизмененная таблица).  >commit  > SELECT \* FROM GroupRec;  (Вывелась измененная таблица. Выполнилась команда второго сеанса). | |
|  | | Рис.3.14. Сеанс №2. Выполнение команды UPDATE и подтверждение транзакций. | |
| Рис.3.13. Сеанс №1. Установка уровня SNAPSHOT и попытка провести команду UPDATE параллельно другой транзакции к одной и той же записи. | |  | |

Видно, что второй сеанс первый занял ресурс и у первого сеанса возник конфликт с конкурирующей транзакцией. Та транзакция, которая первая заняла ресурс «победила» в этой конкуренции.

**3) Изоляция уровня SNAPSHOT TABLE STABILITY:**

|  |
| --- |
| Сеанс №1:  > SET transaction isolation level SNAPSHOT TABLE STABILITY;  Commit current transaction (y/n)?y  Committing.  Сеанс №2:  > SET transaction isolation level SNAPSHOT TABLE STABILITY;  Commit current transaction (y/n)?y  Committing.  > UPDATE GroupRec SET namegroup='H' WHERE group\_id=10;  (Окно повисло в ожидании. Транзакция ждет освобождения ресурса от другого сеанса подключения)  Сеанс №1:  > SELECT \* FROM GroupRec;  (Окно повисло в ожидании. Произошла блокировка чтения таблицы, так как происходит ее модификация во второй)  Сеанс №2:  >commit  Сеанс №1:  (Вывелась старая таблица).  >commit  > SELECT \* FROM GroupRec;  (Вывелась измененная таблица). |

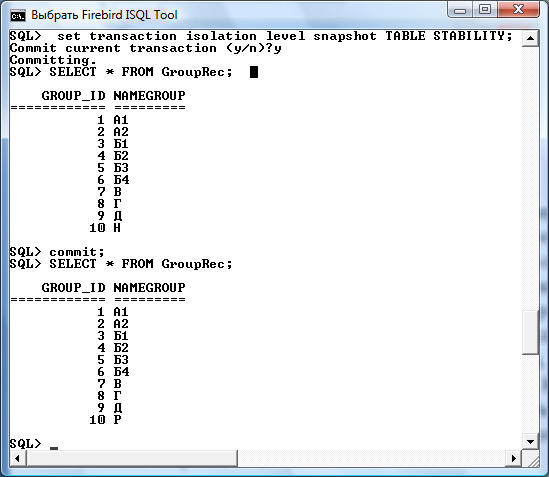
****

Рис.3.15. Сеанс №1. Установка уровня SNAPSHOT TABLE STABILITY и попытка провести команду SELECT к содержимому таблице.

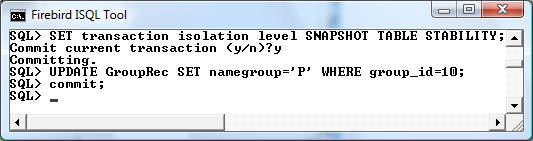
****

Рис.3.16. Сеанс №2. Установка уровня SNAPSHOT TABLE STABILITY, выполнение команды UPDATE и подтверждение транзакций.

Из результатов видно, что при данном уровне изоляции обеспечивается безопасность данных при их изменении, путем их блокировки от других транзакций. Однако в этом случае мы имеем крайне низкую пропускную способность к данным. Если кто-то начал начал записывать или обновлять, то другим придется подождать пока не завершится начатая транзакция.

1. **Вывод:**

Выполнив лабораторную работу №6, мы ознакомились с механизмом транзакций, возможностями ручного управления транзакциями и уровнями изоляции транзакций.

Выяснено, что транзакции применяются для обеспечения целостности данных (транзакция выполняется полностью или не выполняется вообще), корректной работы с данными при обращении нескольких пользователей к одним и тем же данным за счет использования уровней изоляции.

Уровень изолированности транзакций — это есть степень изолированности одной транзакции от другой. Более высокий уровень изолированности повышает точность данных, но при этом снижается количество параллельно выполняемых транзакций. С другой стороны, более низкий уровень изолированности позволяет выполнять больше параллельных транзакций, но при этом возникает большая вероятность наличия несогласованных данных в БД, что приводит к полной неразберихе при чтении/записи данных. Все зависит от степени важности данных и назначения самой БД.

Из выше сказанного следует, что цель данной работы достигнута. Полученные знания могут быть использованы в дальнейшем при работе с СУБД.