

Федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический университет
Петра Великого»
Институт компьютерных наук и технологий
Кафедра Компьютерных систем и программных технологий

Отчёт по лабораторной №7

Дисциплина: Базы данных

Тема: Транзакции

Выполнил студент гр. 43501/4

(подпись) В.С. Филиппов

Руководитель

(подпись) А.В. Мяснов

“ ” _____ 2015 г.

Санкт-Петербург

2015

Цель 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.

Выполнение работы

1. Основные принципы 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.

Транзакции – это главный инструмент, поддерживающий целостность данных.

Основные особенности транзакций:

- 1) Приложение может выполнить одно или несколько операций в контексте транзакции, где каждое должно быть завершено в определённой последовательности. Операции выполняются как правило из sql-операций таких как SELECT, INSERT, UPDATE, DELETE.
- 2) Изменения, выполненные транзакциями, могут быть зафиксированы (committed), если все операции в транзакции были завершены. Пока результат транзакции не зафиксирован (commit), изменения в базе данных не видимы для других пользователей.
- 3) Транзакции также могут быть отменены (rolled back). В этом случае, на сколько другие пользователи не были бы заинтересованы, данные никогда не изменятся.

SQL операторы для работы с транзакциями в Firebird:

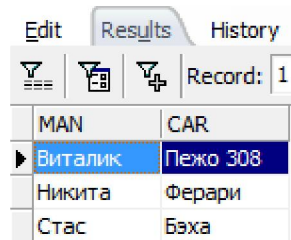
- SET TRANSACTION— задание параметров транзакции и её старт;
- COMMIT— завершение транзакции и сохранение изменений;
- ROLLBACK— отмена изменений, произошедший в рамках транзакции;
- SAVEPOINT— установка точки сохранения для частичного отката изменений, если

это необходимо;

- RELEASE SAVEPOINT— удаление точки сохранения.

2. Эксперименты по запуску, подтверждению и откату транзакций.

```
select * from man car;
```



MAN	CAR
Виталик	Пежо 308
Никита	Ферари
Стас	Бэха

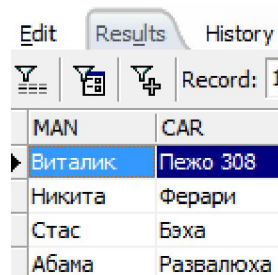
```
commit
```

Видим в таблице 3 записи

```
insert into man car values ('Абама', 'Развалюха')
```

Вставили ещё одну

```
select * from man car
```



MAN	CAR
Виталик	Пежо 308
Никита	Ферари
Стас	Бэха
Абама	Развалюха

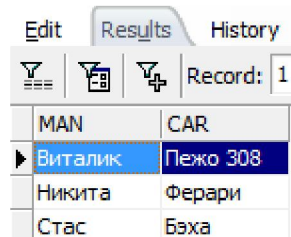
Сохраним точку

```
savepoint lol
```

Удалим добавленную в транзакции запись

```
delete from man car where man = 'Абама'
```

```
select * from man car
```



MAN	CAR
Виталик	Пежо 308
Никита	Ферари
Стас	Бэха

Отменим изменения до точки lol

```
rollback to lol
```

```
select * from man car
```

Edit

Results

History

Y==

Y=

Y+

Record: 1

MAN	CAR
Виталик	Пежо 308
Никита	Ферари
Стас	Бэха
Абама	Развалюха

Откатим транзакцию

```
rollback
select * from man_car;
```

Edit

Results

History

Y==

Y=

Y+

Record: 1

MAN	CAR
Виталик	Пежо 308
Никита	Ферари
Стас	Бэха

3. Эксперименты, показывающие основные возможности транзакций с различным уровнем изоляции

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.

1 клиент

```
SQL> SET TRANSACTION isolation level snapshot;
SQL> select * from man_car;
```

MAN	CAR
Виталик	Пежо 308
Никита	Ферари
Стас	Бэха

2 клиент

```
SQL> SET TRANSACTION isolation level snapshot;
SQL> select * from man_car;
```

MAN	CAR
Виталик	Пежо 308
Никита	Ферари
Стас	Бэха

Как видим в начале таблицы совпадают

```
SQL> insert into man_car values('Абама', 'Развалюха');
SQL> commit;
SQL> select * from man_car;
```

MAN	CAR
Виталик	Пежо 308
Никита	Ферари
Стас	Бэха
??	?

```
SQL>
```

Вставили запись в 1 клиенте и сделали commit

```
SQL> select * from man_car;
```

MAN	CAR
Виталик	Пежо 308
Никита	Ферари
Стас	Бэха

```
SQL> commit;
```

```
SQL> select * from man_car;
```

MAN	CAR
Виталик	Пежо 308
Никита	Ферари
Стас	Бэха
??	?

```
SQL>
```

2 клиент, смотрим таблицу, и не видим изменения только после завершения транзакции (после commit)

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.

1 клиент

```
SQL> SET TRANSACTION isolation level snapshot table stability;
SQL> select * from man_car;
```

MAN	CAR
Виталик	Пежо 308
Никита	Ферари
Стас	Бэха

2 клиент

```

SQL> select * from man_car;

MAN          CAR
=====
Виталик      Пежо 308
Никита       Ферари
Стас         Бэха

SQL> insert into man_car values('Чак','Джип');

На этом второй клиент «завис»,
т.к. не выполнилась транзакция
SNAPSHOT TABLE STABILITY

SQL> commit;
Завершили транзакцию

Теперь клиент «развис»
SQL> commit;
SQL> select * from man_car;

MAN          CAR
=====
Виталик      Пежо 308
Никита       Ферари
Стас         Бэха
- €         " | ЁЇ

```

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 этого уровня изолированности можно указать один из двух значений дополнительной характеристики в зависимости от желаемого способа разрешения конфликтов: **RECORD_VERSION** и **NO RECORD_VERSION**.

NO RECORD_VERSION (значение по умолчанию) В этом случае транзакция не может прочитать любую запись, которая была изменена параллельной активной (неподтвержденной) транзакцией.

Если указана стратегия разрешения блокировок **WAIT** (по умолчанию), то это приведёт к ожиданию завершения или откату конкурирующей транзакции.

```

1 клиент
SQL> SET TRANSACTION isolation level READ COMMITTED;
SQL> select * from man_car;

MAN          CAR
=====
Виталик      Пежо 308
Никита       Ферари
Стас         Бэха

2 клиент
SQL> insert into man_car values('Чак','Джип');
SQL> commit;
SQL>

```

```
SQL> select * from man_car;
```

MAN	CAR
Виталик	Пежо 308
Никита	Ферари
Стас	Бэха
- €	" EI

```
SQL> █
```

Внутри транзакции увидели изменения,
совершённые в другой транзакции

```
SQL> delete from man_car where man = 'Чак';
```

Удалили запись, но не подтвердили

```
SQL> select * from man_car;
```

Завис

```
SQL> commit;
```

Подтвердили

MAN	CAR
Виталик	Пежо 308
Никита	Ферари
Стас	Бэха

```
SQL>
```

развис

Если указана стратегия разрешения блокировок **NO WAIT**, то будет немедленно
выдано соответствующее исключение.

1 клиент

```
SQL> SET TRANSACTION isolation level READ COMMITTED NO WAIT;
```

```
SQL> select * from man_car;
```

MAN	CAR
Виталик	Пежо 308
Никита	Ферари
Стас	Бэха

2 клиент

```
SQL> insert into man_car values('Чак','Джип');
```

```
SQL> select * from man_car;
```

MAN	CAR
Виталик	Пежо 308
Никита	Ферари
Стас	Бэха

Statement failed, SQLSTATE = 40001
lock conflict on no wait transaction
-deadlock
-concurrent transaction number is 330

ошибка, т.к. другая транзакция изменила таблицу
и не подтвердила изменения

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.

```
1 клиент
SQL> SET TRANSACTION isolation level
CON> READ COMMITTED RECORD_VERSION;
SQL> select * from man_car;
```

MAN	CAR
Виталик	Пежо 308
Никита	Ферари
Стас	Бэха

```
2 клиент
SQL> insert into man_car values('Чак','Джип');
SQL> select * from man_car;
```

MAN	CAR
Виталик	Пежо 308
Никита	Ферари
Стас	Бэха

Не видим не подтверждённое изменение

```
SQL> commit;
```

```
SQL> select * from man_car;
```

MAN	CAR
Виталик	Пежо 308
Никита	Ферари
Стас	Бэха
- €	" Eİ

```
SQL>
```

Видим подтверждённое изменение

Вывод

Большинство действий с базами данных включает в себя несколько запросов внутри одной транзакции. Транзакция гарантирует, что все её запросы будут выполнены или не выполнены совсем. Простейшим примером важности транзакций является банковская система. При переводе средств с одного счета на другой необходимо совершить два действия: прибавить сумму на одном счете и вычесть на другом. Одно из действий может быть недоступно, тогда и второе не должно быть выполнено. Также это предотвратит изменения в БД при неожиданном обрыве канала связи с сервером. Не возникнет ситуации, когда выполнена только часть транзакции.

Помимо этого, транзакции позволяют нескольким пользователям работать с одной БД одновременно. При этом можно задавать разные способы разрешения конфликтов

(уровни изолированности), при одновременной работе с данными. Уровни изолированности транзакций позволяют нам разгрузить систему.

К примеру, заставить ожидать всех пользователей пока один из них не изменит таблицу (**READ COMMITTED NO RECORD_VERSION WAIT**). В таком случае изменения наиболее важны, и обеспечивается параллельная работа транзакций. Можно разрешить работу всех пользователей с какой-либо фиксированной версией БД, и обновить ее по окончании транзакций (**SNAPSHOT**), что тоже позволяет работать параллельно, но изолированно друг от друга. При уровне изоляции **SNAPSHOT TABLE STABILITY** мы запрещаем изменять таблицу другим транзакциям. Это обеспечивает максимальную безопасность, но сильно нагружает систему.