

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Отчет по лабораторной работе № 7

«Изучение механизма транзакций»

по дисциплине «Базы данных»

Работу выполнил:

студент гр. 43501/3

Хуторной Я. В.

Руководитель

Мяснов А. В.

«__» _____ 2015 г

Санкт-Петербург

2015

Цели работы

Познакомить студентов с механизмом транзакций, возможностями ручного управления транзакциями, уровнями изоляции транзакций.

Транзакции

Транзакция - набор логически связанных операций, работающих с данными базы данных, и либо переводящий базу данных из одного целостного состояния в другое, либо нет (т.е. оставляющий БД в целостном состоянии, существовавшем до начала транзакции).

Методом контроля над 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.

Если одновременно выполняются две транзакции, могут возникнуть следующие ошибочные ситуации:

- **Грязное чтение (Dirty Read)** - транзакция T1 модифицировала некий элемент данных. После этого другая транзакция T2 прочитала содержимое этого элемента данных до завершения транзакции T1. Если T1 завершается операцией `ROLLBACK`, то получается, что транзакция T2 прочитала не существующие данные.
- **Неповторяемое (размытое) чтение (Non-repeatable or Fuzzy Read)** - транзакция T1 прочитала содержимое элемента данных. После этого другая транзакция T2 модифицировала или удалила этот элемент. Если T1 прочитает содержимое этого элемента заново, то она получит другое значение или обнаружит, что элемент данных больше не существует.
- **Фантом (фиктивные элементы) (Phantom)** - транзакция T1 прочитала содержимое нескольких элементов данных, удовлетворяющих некому условию. После этого T2 создала элемент данных, удовлетворяющий этому условию и зафиксировалась. Если T1 повторит чтение с тем же условием, она получит другой набор данных.

Принудительное упорядочение транзакций обеспечивается с помощью механизма **блокировок**. Суть этого механизма в следующем: если для выполнения некоторой транзакции необходимо, чтобы некоторый объект базы данных (кортеж, набор кортежей, отношение, набор отношений,..) не изменялся непредсказуемо и без ведома этой транзакции, такой объект блокируется. Основными видами блокировок являются:

- **блокировка со взаимным доступом**, называемая также *S-блокировкой* (от Shared locks) и *блокировкой по чтению*.
- **монопольная блокировка** (без взаимного доступа), называемая также *X-блокировкой* (от eXclusive locks) или *блокировкой по записи*. Этот режим используется при операциях изменения, добавления и удаления объектов.

При этом:

- если транзакция налагает на объект X-блокировку, то любой запрос другой транзакции с блокировкой этого объекта будет отвергнут.
- если транзакция налагает на объект S-блокировку, то
- запрос со стороны другой транзакции с X-блокировкой на этот объект будет отвергнут
- запрос со стороны другой транзакции с S-блокировкой этого объекта будет принят

Доказано, что сериализуемость транзакций (или, иначе, их изоляция) обеспечивается при использовании двухфазного протокола блокировок (2LP - Two-Phase Locks), согласно которому все блокировки, произведенные транзакцией, снимаются только при ее завершении. Т.е. выполнение транзакции разбивается на две фазы: (1) - накопление блокировок, (2) - освобождение блокировок в результате фиксации или отката.

К сожалению, применение механизма блокировки приводит к замедлению обработки транзакций, поскольку система вынуждена ожидать пока освободятся данные, захваченные конкурирующей транзакцией. Решить эту проблему можно за счет уменьшения фрагментов данных, захватываемых транзакцией. В зависимости от захватываемых объектов различают несколько *уровней блокировки*:

- блокируется вся база данных - очевидно, этот вариант неприемлем, поскольку сводит многопользовательский режим работы к однопользовательскому
- блокируются отдельные таблицы
- блокируются страницы (страница - фрагмент таблицы размером обычно 2-4 Кб, единица выделения памяти для обработки данных системой)

- блокируются записи
- блокируются отдельные поля

Язык SQL также предоставляет способ косвенного управления скоростью выполнения 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:

Уровень изоляции	Грязное чтение	Размытое чтение	Фантом
Незафиксированное чтение (READ UNCOMMITTED)	возможно	возможно	возможно
Зафиксированное чтение (READ COMMITTED)	невозможно	возможно	возможно
Повторяемое чтение (REPEATABLE READ)	невозможно	невозможно	возможно
Сериализуемость (SERIALIZABLE)	невозможно	невозможно	невозможно

Для определения характеристик транзакции используется оператор

SET TRANSACTION <режим_доступа>, <уровень_изоляции>

Список уровней изоляции приведен в таблице. Режим доступа по умолчанию используется READ WRITE (чтение запись), если задан уровень изоляции READ UNCOMMITTED, то режим доступа должен быть READ ONLY (только чтение).

Программа работы

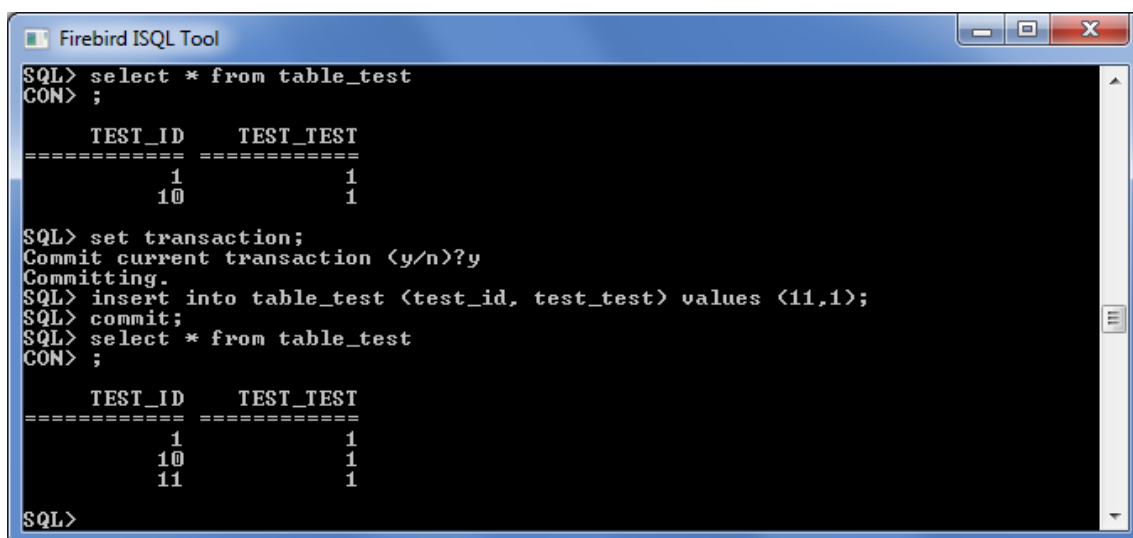
- Изучить основные принципы работы транзакций.
- Провести эксперименты по запуску, подтверждению и откату транзакций.
- Разобраться с уровнями изоляции транзакций в Firebird.
- Спланировать и провести эксперименты, показывающие основные возможности транзакций с различным уровнем изоляции.
- Продемонстрировать результаты преподавателю, ответить на контрольные вопросы.

Выполнение работы

1. Были изучены основные принципы транзакций.
2. Были проведены эксперименты по запуску, подтверждению и откату транзакций.

1) Эксперимент 1. Запуск и подтверждение транзакции.

Транзакция включает в себя команду insert с добавлением записи в таблицу. После подтверждения транзакции, запись успешно добавлена.



```
SQL> select * from table_test
CON> ;

=====
TEST_ID  TEST_TEST
=====
      1         1
     10         1

SQL> set transaction;
Commit current transaction (y/n)?y
Committing.
SQL> insert into table_test (test_id, test_test) values (11,1);
SQL> commit;
SQL> select * from table_test
CON> ;

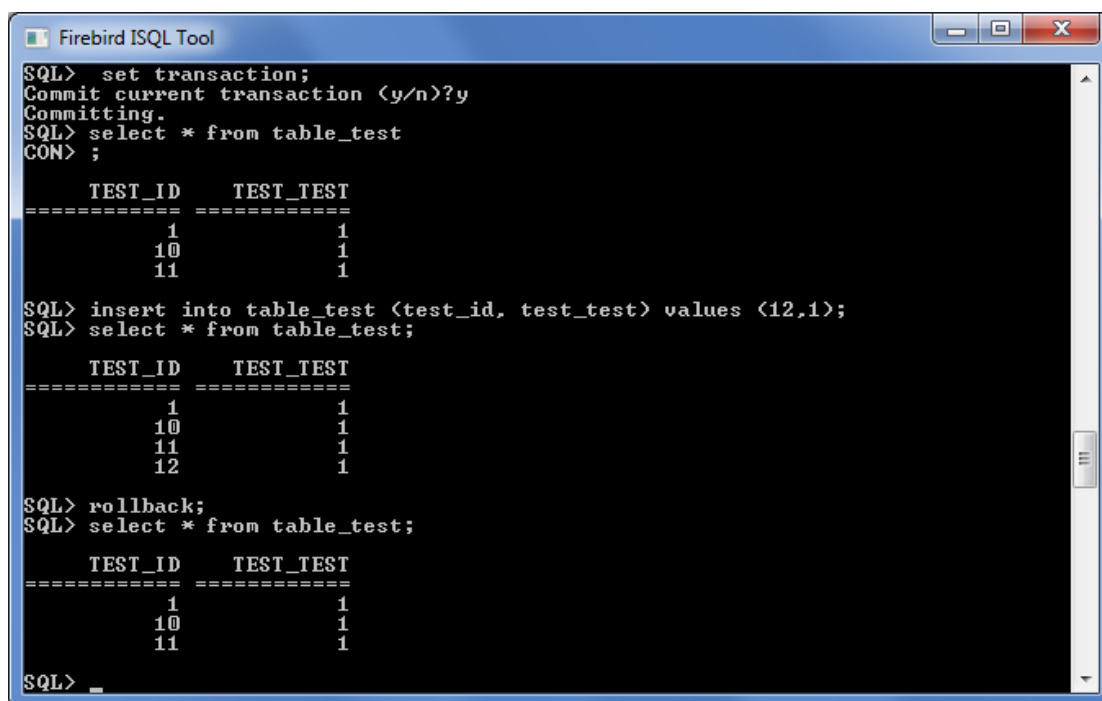
=====
TEST_ID  TEST_TEST
=====
      1         1
     10         1
     11         1

SQL>
```

Рис. 1. Результат запуска и подтверждения транзакции

2) Эксперимент 2. Запуск и откат транзакции.

Транзакция включает в себя команду insert с добавлением записи в таблицу и команду select для отображения новой добавленной записи. Однако происходит откат транзакции и таблица возвращается в исходное (до начала транзакции) состояние.



```
SQL> set transaction;
Commit current transaction (y/n)?y
Committing.
SQL> select * from table_test
CON> ;

=====
TEST_ID  TEST_TEST
=====
      1         1
     10         1
     11         1

SQL> insert into table_test (test_id, test_test) values (12,1);
SQL> select * from table_test;

=====
TEST_ID  TEST_TEST
=====
      1         1
     10         1
     11         1
     12         1

SQL> rollback;
SQL> select * from table_test;

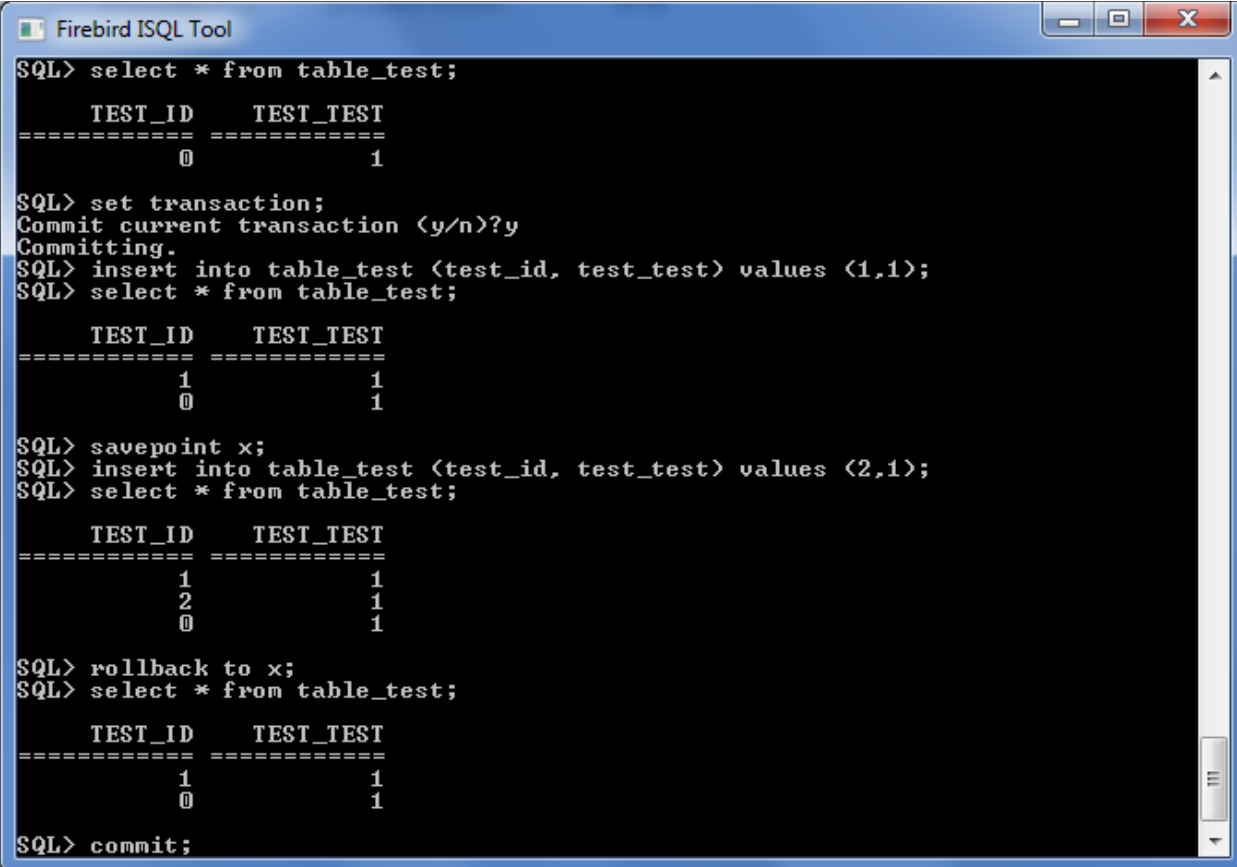
=====
TEST_ID  TEST_TEST
=====
      1         1
     10         1
     11         1

SQL> _
```

Рис. 2. Результат запуска и отката транзакции

3) Эксперимент 3. Запуск и откат до контрольной точки (save point).

Сначала происходит добавление новой записи в таблицу, после чего создается savepoint. После этого в таблицу добавляется еще одна запись, но после этого происходит откат к savepoint. В таблице остается только первая запись.



```
SQL> select * from table_test;

=====
TEST_ID  TEST_TEST
=====
0        1

SQL> set transaction;
Commit current transaction (y/n)?y
Committing.
SQL> insert into table_test (test_id, test_test) values (1,1);
SQL> select * from table_test;

=====
TEST_ID  TEST_TEST
=====
1        1
0        1

SQL> savepoint x;
SQL> insert into table_test (test_id, test_test) values (2,1);
SQL> select * from table_test;

=====
TEST_ID  TEST_TEST
=====
1        1
2        1
0        1

SQL> rollback to x;
SQL> select * from table_test;

=====
TEST_ID  TEST_TEST
=====
1        1
0        1

SQL> commit;
```

Рис. 3. Результат запуск транзакции и откат до контрольной точки (save point).

3. Были спланированы и проведены эксперименты, показывающие основные возможности транзакций с различным уровнем изоляции.

1) Эксперимент 1. Уровень изоляции - READ COMMITTED.

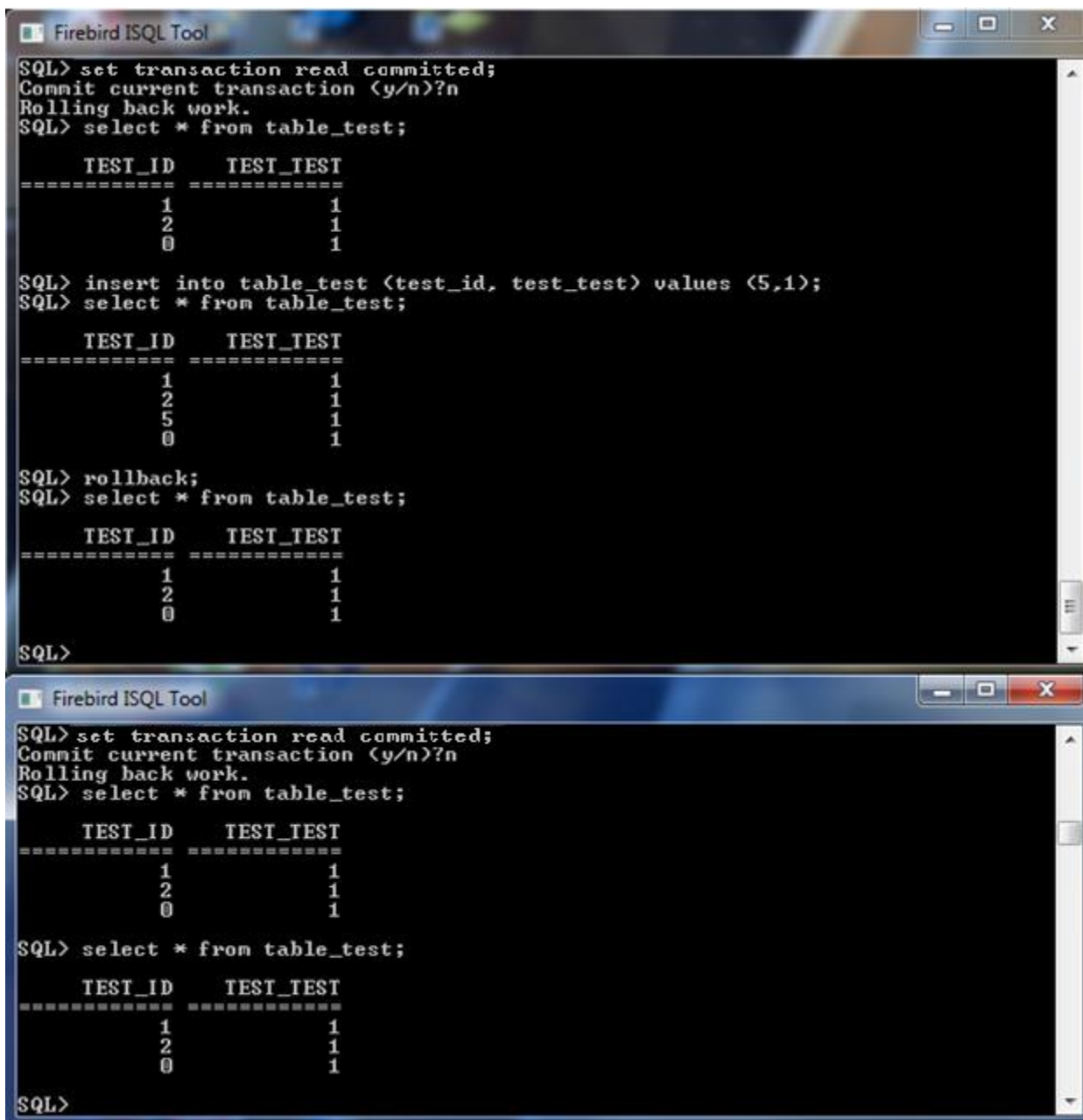
Данный уровень изоляции позволяет избавиться от ошибок грязного чтения.

Последовательность действий эксперимента:

- 1) Транзакция 1 читает таблицу table_test (select).
- 2) Транзакция 2 добавляет в таблицу table_test новую запись.
- 3) Транзакция 1 читает таблицу table_test (select).
- 4) Транзакция 1 производит откат.

Уровень изоляции READ UNCOMMITTED не позволяет контролировать ошибки грязного чтения. Уровень изоляции READ COMMITTED позволяет контролировать ошибки грязного чтения. READ UNCOMMITTED не поддерживается в Firebird, READ COMMITTED соответствует стандарту.

Поэтому в результате повторного чтения таблицы транзакцией 1, результат должен соответствовать первому чтению. Если бы уровень изоляции был READ UNCOMMITTED, то при повторном чтении транзакция 1 бы зафиксировала новые данные, но после того, как транзакция 2 сделает откат, эти данные будут не соответствовать реальным (грязные).



```
SQL> set transaction read committed;
Commit current transaction (y/n)?n
Rolling back work.
SQL> select * from table_test;

=====
TEST_ID  TEST_TEST
=====
1         1
2         1
3         1

SQL> insert into table_test (test_id, test_test) values (5,1);
SQL> select * from table_test;

=====
TEST_ID  TEST_TEST
=====
1         1
2         1
5         1
3         1

SQL> rollback;
SQL> select * from table_test;

=====
TEST_ID  TEST_TEST
=====
1         1
2         1
3         1

SQL>
```

```
SQL> set transaction read committed;
Commit current transaction (y/n)?n
Rolling back work.
SQL> select * from table_test;

=====
TEST_ID  TEST_TEST
=====
1         1
2         1
3         1

SQL> select * from table_test;

=====
TEST_ID  TEST_TEST
=====
1         1
2         1
3         1

SQL>
```

Рис. 4. Результат эксперимента. Сверху – транзакция 2. Снизу – транзакция 1

2) Эксперимент 2. Уровень изоляции - READ COMMITTED.

Данный эксперимент демонстрирует ситуацию, в которой допускается ошибка размытого чтения.

Последовательность действий эксперимента:

- 1) Транзакция 1 читает таблицу table_test (select).
- 2) Транзакция 2 добавляет в таблицу table_test новую запись.
- 3) Транзакция 1 читает таблицу table_test (select).

В результате эксперимента при повторном чтении транзакция 1 прочитает неверные данные:

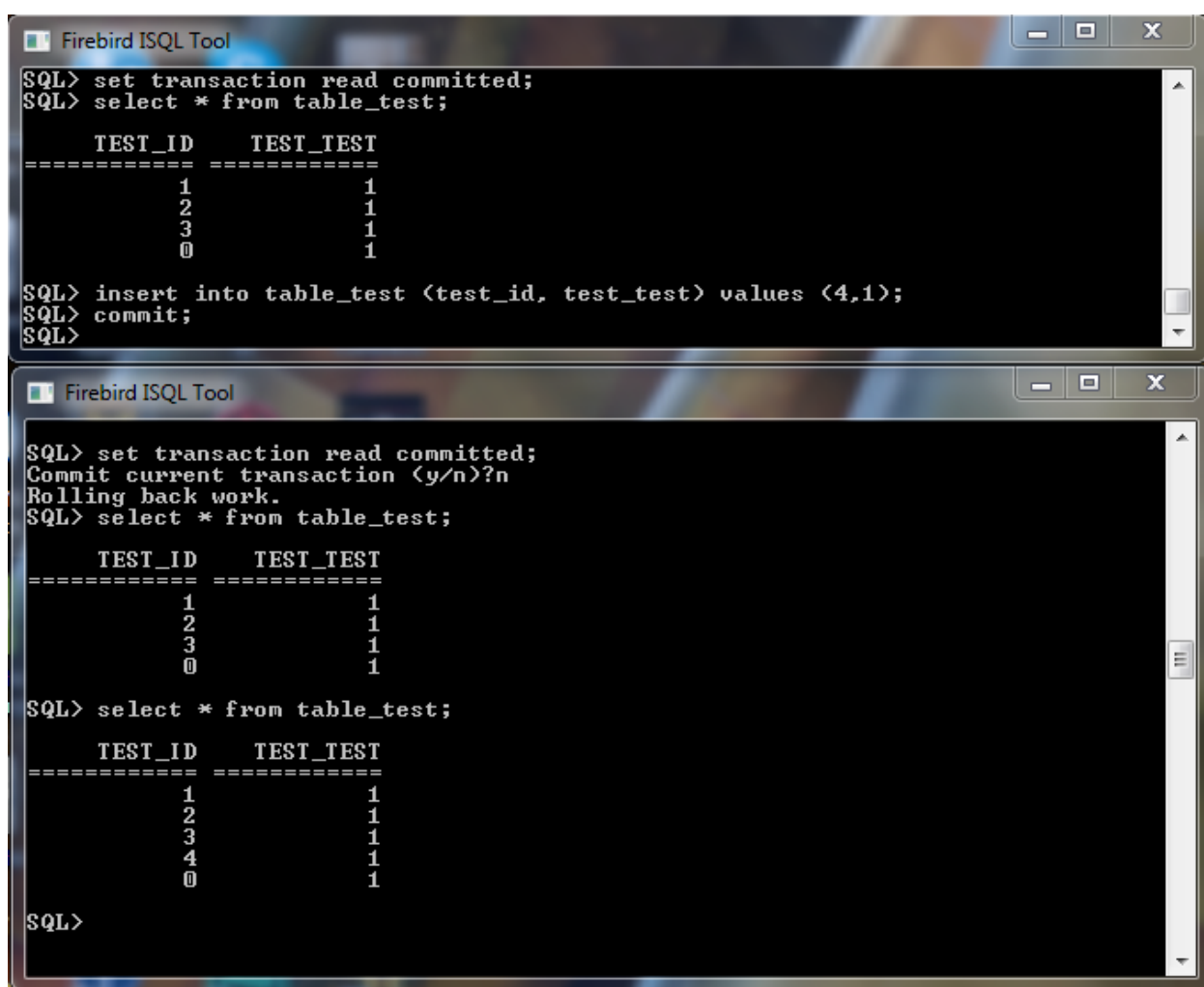


Рис. 5. Результат эксперимента. Сверху – транзакция 2. Снизу – транзакция 1

4) Эксперимент 4. Уровень изоляции - REPEATABLE READ (snapshot).

Уровень изоляции snapshot позволяет видеть только те изменения, фиксация которых произошла не позднее момента старта этой транзакции. Любые подтвержденные изменения, сделанные другими конкурирующими

транзакциями, не будут видны в такой транзакции в процессе ее активности без ее перезапуска.

Данный эксперимент показывает, как при помощи нового уровня изоляции избавиться от ошибок предыдущего эксперимента.

Последовательность действий эксперимента:

- 1) Транзакция 1 читает таблицу table_test (select).
- 2) Транзакция 2 добавляет в таблицу table_test новую запись.
- 3) Транзакция 2 читает таблицу table_test (select).
- 4) Транзакция 1 читает таблицу table_test (select).

```
SQL> set transaction snapshot;
SQL> insert into table_test (test_id, test_test) values (5,1);
SQL> select * from table_test;

TEST_ID  TEST_TEST
=====
1         1
2         1
3         1
4         1
5         1
0         1

SQL> commit;

SQL> set transaction snapshot;
Commit current transaction (y/n)?n
Rolling back work.
SQL> select * from table_test;

TEST_ID  TEST_TEST
=====
1         1
2         1
3         1
4         1
0         1

SQL> select * from table_test;

TEST_ID  TEST_TEST
=====
1         1
2         1
3         1
4         1
0         1

SQL> select * from table_test;

TEST_ID  TEST_TEST
=====
1         1
2         1
3         1
4         1
0         1

SQL> commit;
```

Рис. 6. Результат эксперимента. Сверху – транзакция 2. Снизу – транзакция 1

В результате эксперимента удалось избавиться от ошибки размытого чтения.

- 5) Эксперимент 5. Уровень изоляции – snapshot table stability.

Уровень изоляции snapshot table stability позволяет видеть только те изменения, фиксация которых произошла не позднее момента старта этой транзакции. При этом после старта такой транзакции в других клиентских транзакциях невозможно выполнения изменений ни в каких таблицах этой базы данных, уже каким-либо образом измененных первой транзакцией.

Один клиент выполнил вставку в таблицу, второй при этом не может завершить операцию вставки.

```
SQL> set transaction snapshot table stability;
SQL> insert into table_test (test_id, test_test) values (55,1);
SQL> select * from table_test;

=====
TEST_ID  TEST_TEST
=====
1         1
2         1
3         1
4         1
5         1
10        1
55        1
0         1

SQL>
```

```
SQL> set transaction snapshot table stability;
Commit current transaction (y/n)?n
Rolling back work.
SQL> insert into table_test (test_id, test_test) values (65,1);
```

Рис. 7. Результат эксперимента

И только после фиксации изменений первым клиентом, второй смог завершить операцию вставки.

```
SQL> set transaction snapshot table stability;
SQL> insert into table_test (test_id, test_test) values (55,1);
SQL> select * from table_test;

=====
TEST_ID  TEST_TEST
=====
1         1
2         1
3         1
4         1
5         1
10        1
55        1
0         1

SQL> commit;
```

```
SQL> set transaction snapshot table stability;
Commit current transaction (y/n)?n
Rolling back work.
SQL> insert into table_test (test_id, test_test) values (65,1);
SQL> commit;
SQL> select * from table_test;

=====
TEST_ID  TEST_TEST
=====
1         1
2         1
3         1
4         1
5         1
10        1
55        1
65        1
0         1

SQL>
```

Рис. 8. Результат эксперимента

6) Эксперимент 6. Уровень изоляции - READ COMMITTED no record version и READ COMMITTED record version.

NO RECORD_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) (рис. 9).

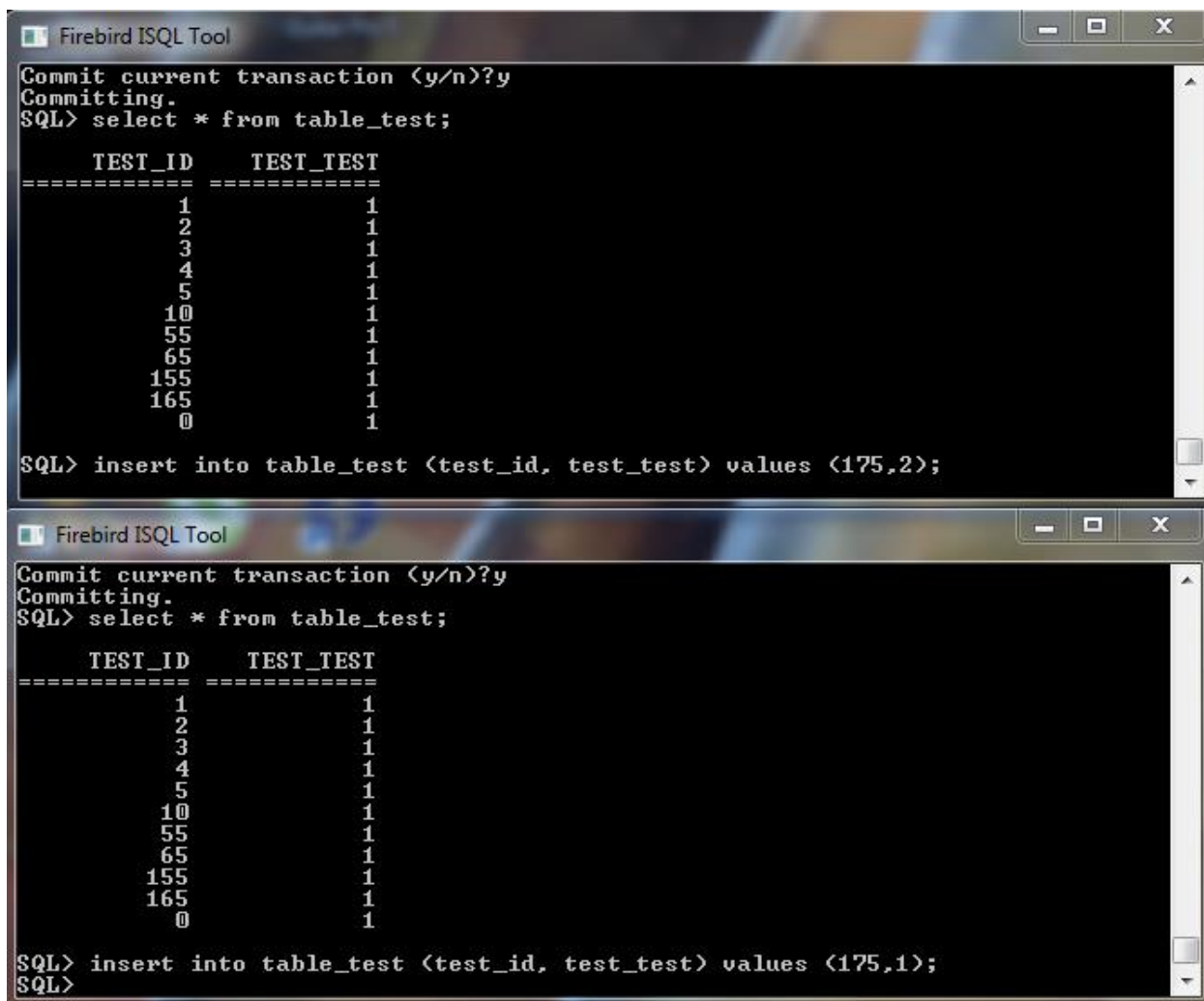


Рис. 9. Результат эксперимента

Уровень изоляции - READ COMMITTED record version.

RECORD_VERSION - игнорирует non-committed версии, читая последнюю committed-версию (рис. 10).

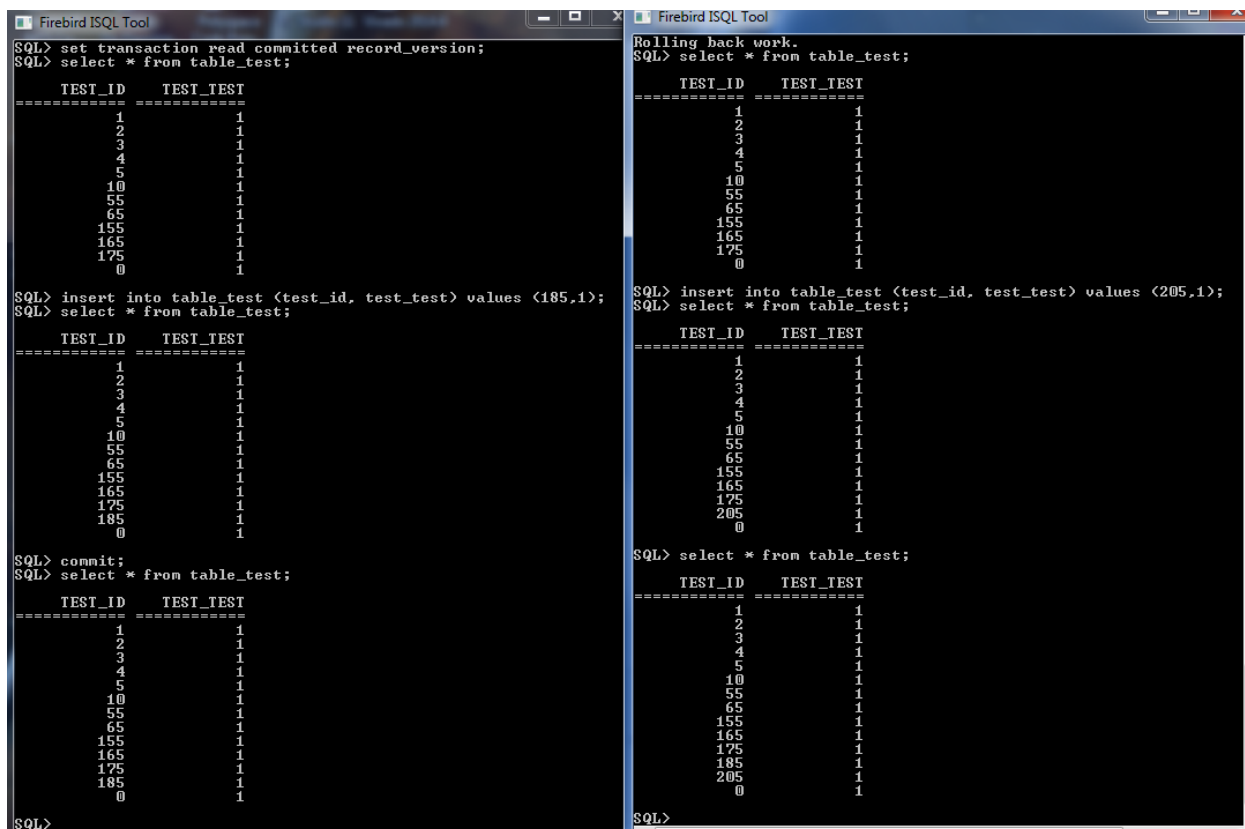


Рис. 10. Результат эксперимента

Вывод

В данной работе был изучен механизм транзакций, возможности управления транзакциями и уровни изоляции транзакций в Firebird. Транзакции позволяют сохранить целостность БД при подключении к ней нескольких клиентов. Задавая различные типы разрешения конфликтов и уровни изоляции можно управлять видимостью изменений, произошедших в базе для разных пользователей.