Санкт-Петербургский политехнический университет Петра Великого
Кафедра компьютерных систем и программных технологий

Отчёт по лабораторной работе

Дисциплина: Базы данных

Тема: Изучение работы транзакций

Выполнил студент гр. 43501/1 _____ Г. А. Жемелев
(подпись)

Преподаватель _____ А. В. Мяснов
(подпись)

“16” декабря 2016 г.

Санкт-Петербург
2016

Цель работы

Познакомиться 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

Транзакция – это атомарное действие над базой данных, переводящее ее из одного целостного состояния в другое целостное состояние. Транзакция включает в себя последовательность операций, которые либо выполняются все, либо не выполняется ни одна из них (например, если произошла ошибка).

Порождение транзакций может быть как явным – с помощью оператора SET TRANSACTION в Firebird SQL, – так и неявным: новая транзакция начинается автоматически после подключения к БД и после каждого оператора COMMIT, говорящей об успешном завершении транзакции. Неуспешное завершение требует так называемого отката транзакции, то есть возвращения БД в состояние, в котором она находилась на момент начала транзакции – для этого используется оператор ROLLBACK. В рамках транзакции можно создавать точки сохранения, откат к которым также производится с помощью оператора ROLLBACK с указанием имени интересующей точки.

Проведем несколько экспериментов по запуску, подтверждению и откату транзакций на тестовой базе данных из одной таблицы:

```
SQL> create table ttab (ID INTEGER PRIMARY KEY, Val VARCHAR(20));
```

```
SQL> insert into ttab values(1, 'First record');
SQL> insert into ttab values(2, 'Second record');
SQL> insert into ttab values(3, 'Third record');
SQL> select * from ttab;

          ID VAL
============ ====================
           1 First record
           2 Second record
           3 Third record

SQL> commit;
SQL> insert into ttab values(4, 'Fourth record');
SQL> update ttab set ID = 33 where ID = 3;
SQL> select * from ttab;

          ID VAL
============ ====================
           1 First record
           2 Second record
          33 Third record
           4 Fourth record

SQL> rollback;
SQL> select * from ttab;

          ID VAL
============ ====================
           1 First record
           2 Second record
           3 Third record

SQL> insert into ttab values(4, 'Fourth record');
SQL> commit;
SQL> rollback;
SQL> select * from ttab;

          ID VAL
============ ====================
           1 First record
           2 Second record
           3 Third record
           4 Fourth record

SQL> delete from ttab where ID = 3;
SQL> set transaction;
Commit current transaction (y/n)?y
Committing.
SQL> select * from ttab;

          ID VAL
============ ====================
           1 First record
           2 Second record
           4 Fourth record
```

```
SQL> insert into ttab values(5, 'Fifth record');
SQL> savepoint sp1;
SQL> insert into ttab values(6, 'Sixth record');
SQL> select * from ttab;

          ID VAL
============ ====================
           1 First record
           2 Second record
           5 Fifth record
           4 Fourth record
           6 Sixth record

SQL> rollback to savepoint sp1;
SQL> select * from ttab;

          ID VAL
============ ====================
           1 First record
           2 Second record
           5 Fifth record
           4 Fourth record

SQL> rollback;
SQL> select * from ttab;

          ID VAL
============ ====================
           1 First record
           2 Second record
           4 Fourth record

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

          ID VAL
============ ====================
           1 First record
           2 Second record
           4 Fourth record

SQL> drop table ttab;
Statement failed, SQLSTATE = 42000
unsuccessful metadata update
-object TABLE "TTAB" is in use
SQL> commit;
SQL> drop table ttab;
SQL> show tables;
There are no tables in this database
SQL> rollback;
SQL> show tables;
There are no tables in this database
```

В ходе эксперимента помимо операторов SQL-DML был испытан оператор

DROP TABLE из SQL-DDL. Попытка удалить таблицу в незавершенной транзакции (содержавшей один оператор SELECT) была пресечена СУБД, а вот после завершения транзакции таблица была успешно удалена и возвратить её с помощью ROLLBACK не удалось – таким образом, можно заключить, что при работе с SQL-DDL нельзя полагаться на механизм транзакций как на защиту от неосторожного изменения схемы БД. В то же время, при манипуляции данными механизм транзакций успешно выполняет свои функции.

В одной базе данных может одновременно проводиться множество транзакций (по меньшей мере одна транзакция для каждого пользователя). Поэтому возникает проблема синхронизации данных в БД. В связи с этим каждая транзакция характеризуется уровнем изоляции – значением, которое определяет уровень, при котором в транзакции допускаются несогласованные данные, то есть степень изолированности одной транзакции от другой.

В Firebird определены три уровня изоляции:

- READ COMMITED – позволяет в транзакции без её перезапуска видеть все подтверждённые изменения данных, выполненные в других параллельных транзакциях. Неподтверждённые изменения не видны.
- SNAPSHOT – текущей 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 ее перезапуска.
- SNAPSHOT TABLE STABILITY – позволяет, как и в случае с уровнем SNAPSHOT, 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 базы данных, уже каким-либо образом измененных первой транзакцией.

При этом существует ключевое слово RESERVING, которое позволяет ослабить или ужесточить степень изоляции конкретных таблиц в рамках транзакции.

Для испытания различных уровней изоляции воспользуемся программой IBExpert, позволяющей создать несколько сессий связи с БД одновременно, имитируя таким образом, работу нескольких пользователей с одной базой данных.

Дальнейшие эксперименты представлены в виде последовательности рисунков 1-13, иллюстрирующих выполняемые действия.

Эксперимент для READ COMMITED

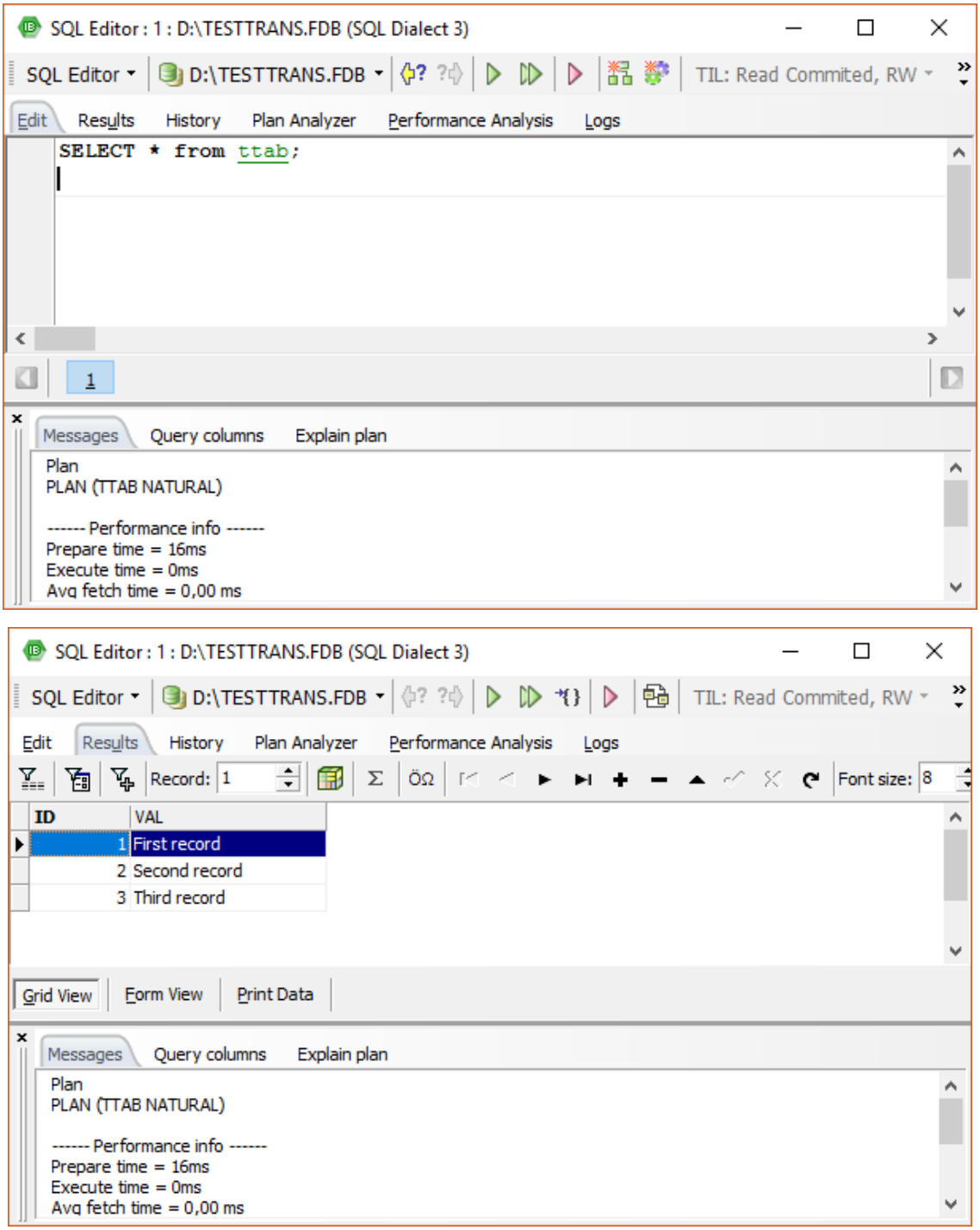

Рис. 1. Запрос из первой транзакции (вверху) и результат (внизу)

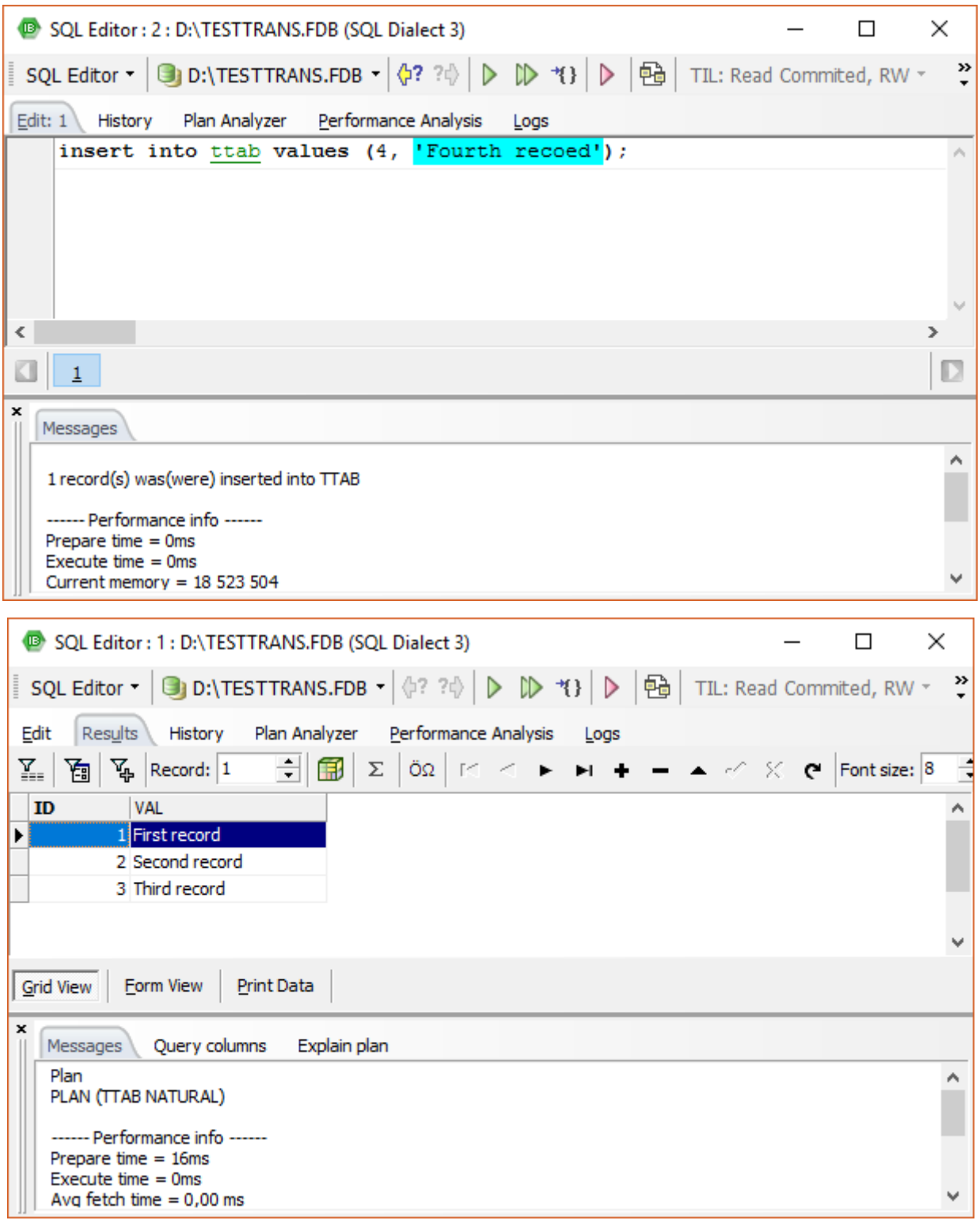

Рис. 2. Запрос из второй транзакции (вверху) и повторный запрос из первой (внизу) – первая транзакция «не видит» неподтвержденные данные

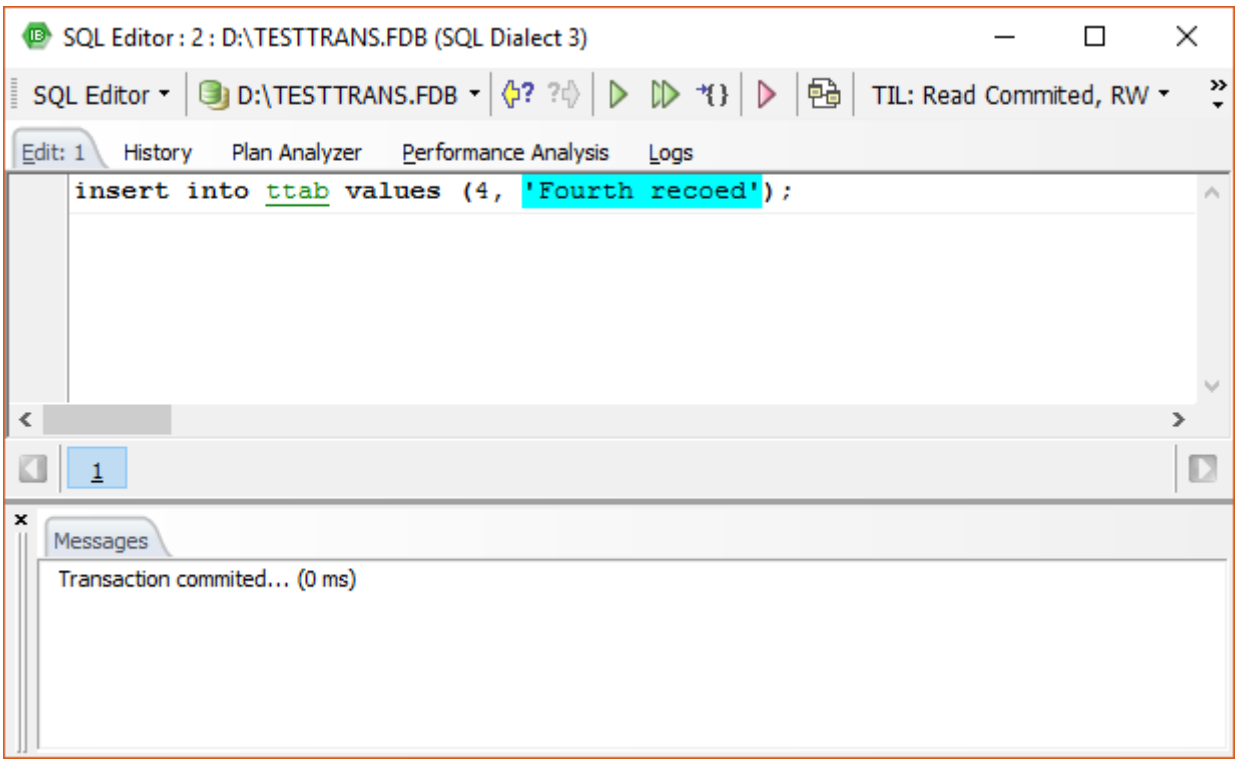

Рис. 3. Завершение второй транзакции

(после выбора пункта меню «SQL Editor/Commit Transaction…»)

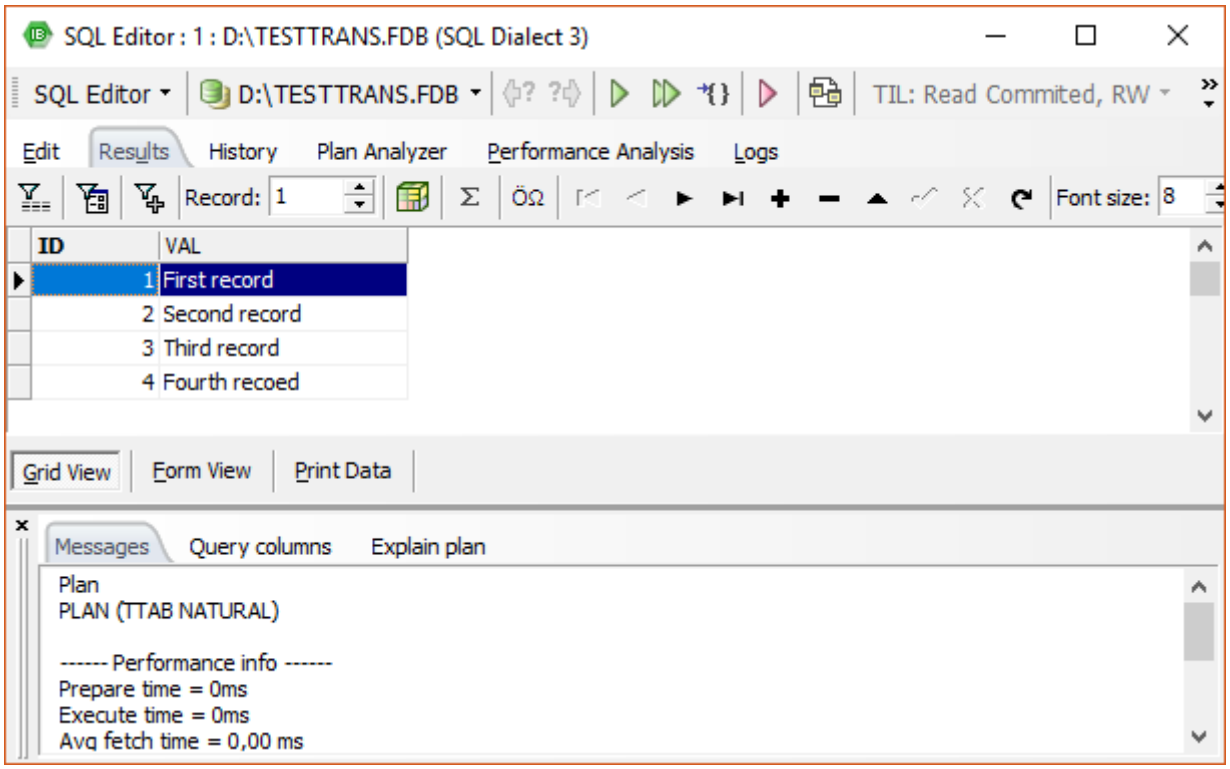

Рис. 4. Результат повторного выполнения SELECT в первой транзакции – новая запись теперь видна, так как эти данные подтверждены

Эксперимент для SNAPSHOT

Уровень изоляции текущей транзакции можно изменить путем выбора соответствующего пункта в меню «TIL» в правом верхнем углу окна SQL Editor. Установим «TIL: Snapshot, RW» и проделаем аналогичный эксперимент.

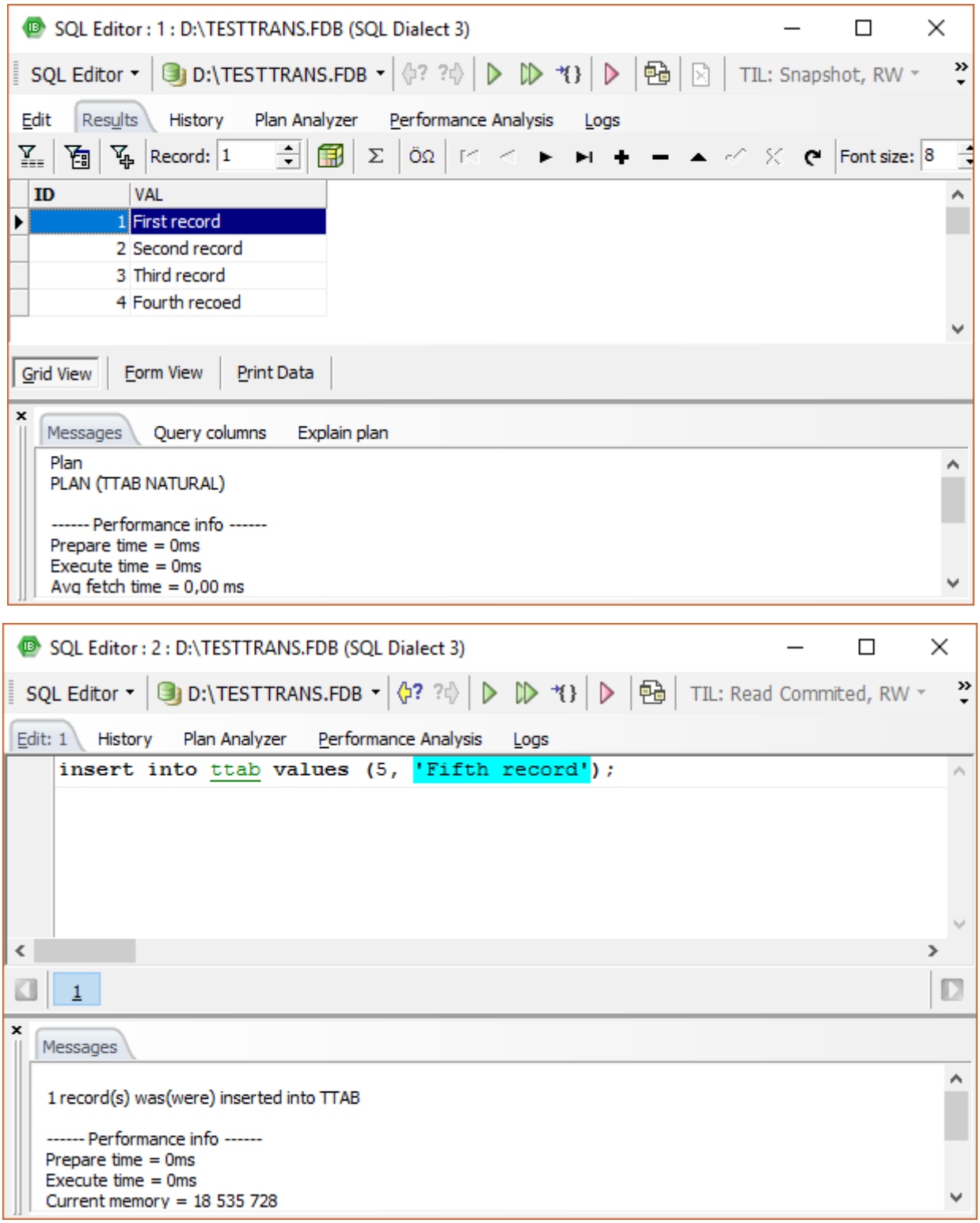

Рис. 5. Результаты запросов из первой транзакции (вверху) и второй (внизу)

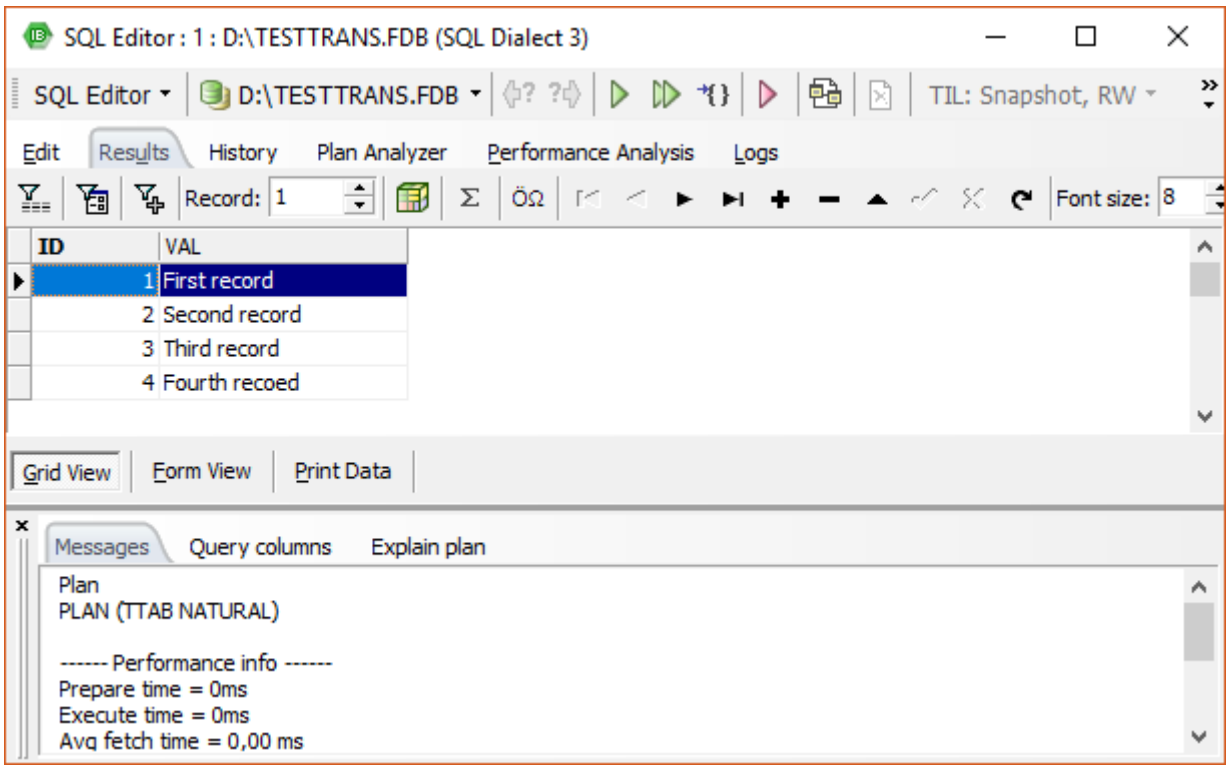

Рис. 6. Результат повторного вызова SELECT в первой транзакции

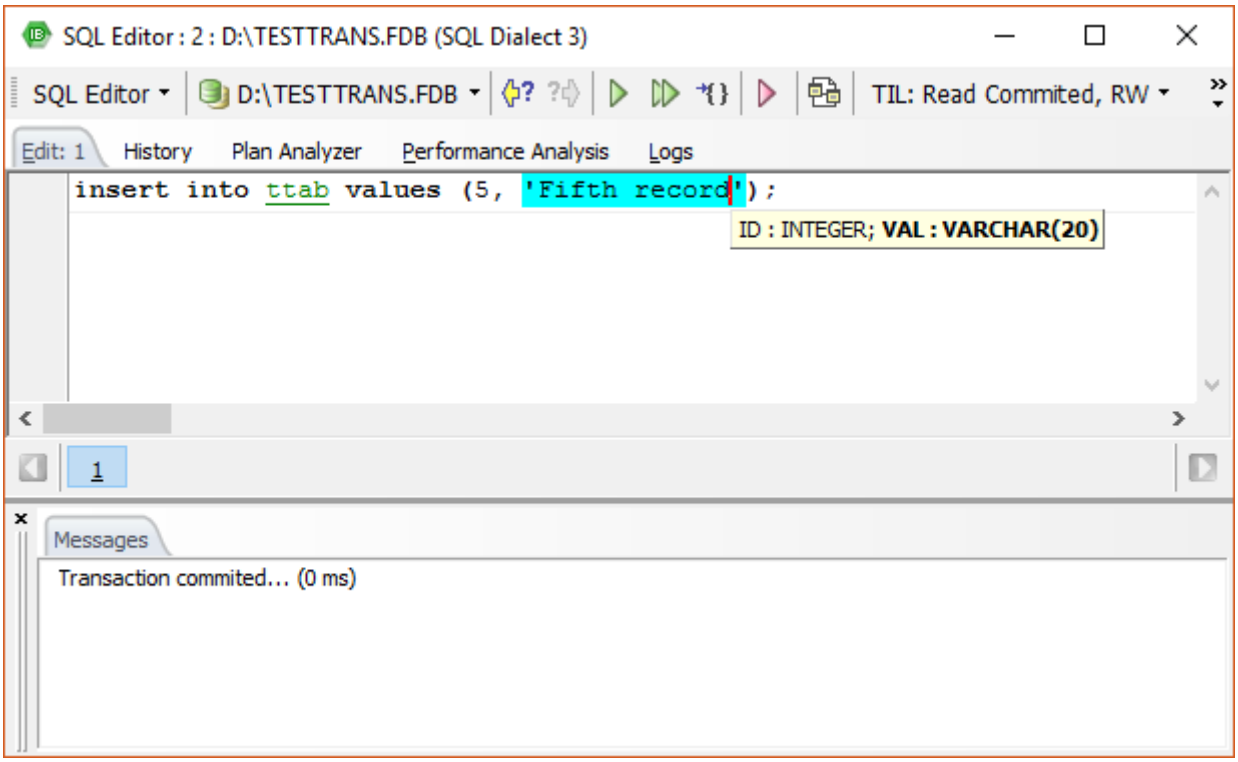

Рис. 7. Завершение второй транзакции

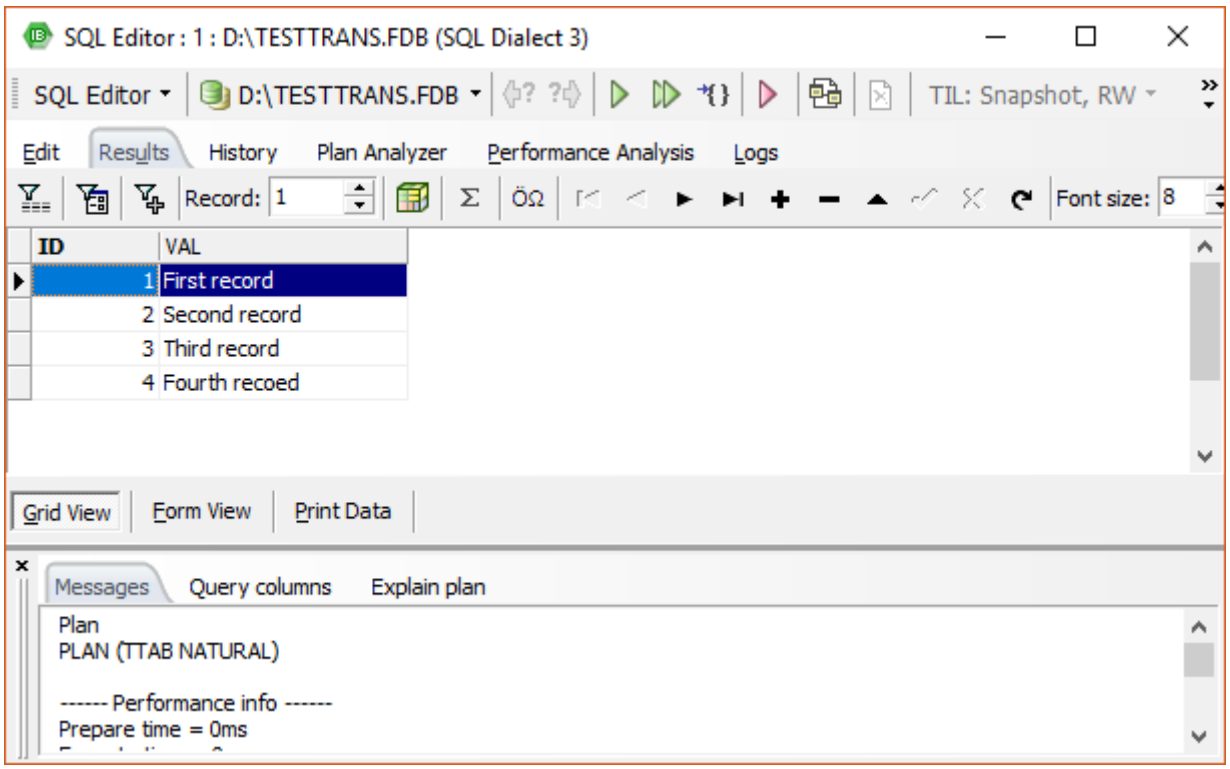

Рис. 8. Результат еще одного вызова SELECT в первой транзакции – теперь новая запись не видна, несмотря на то что подтверждена

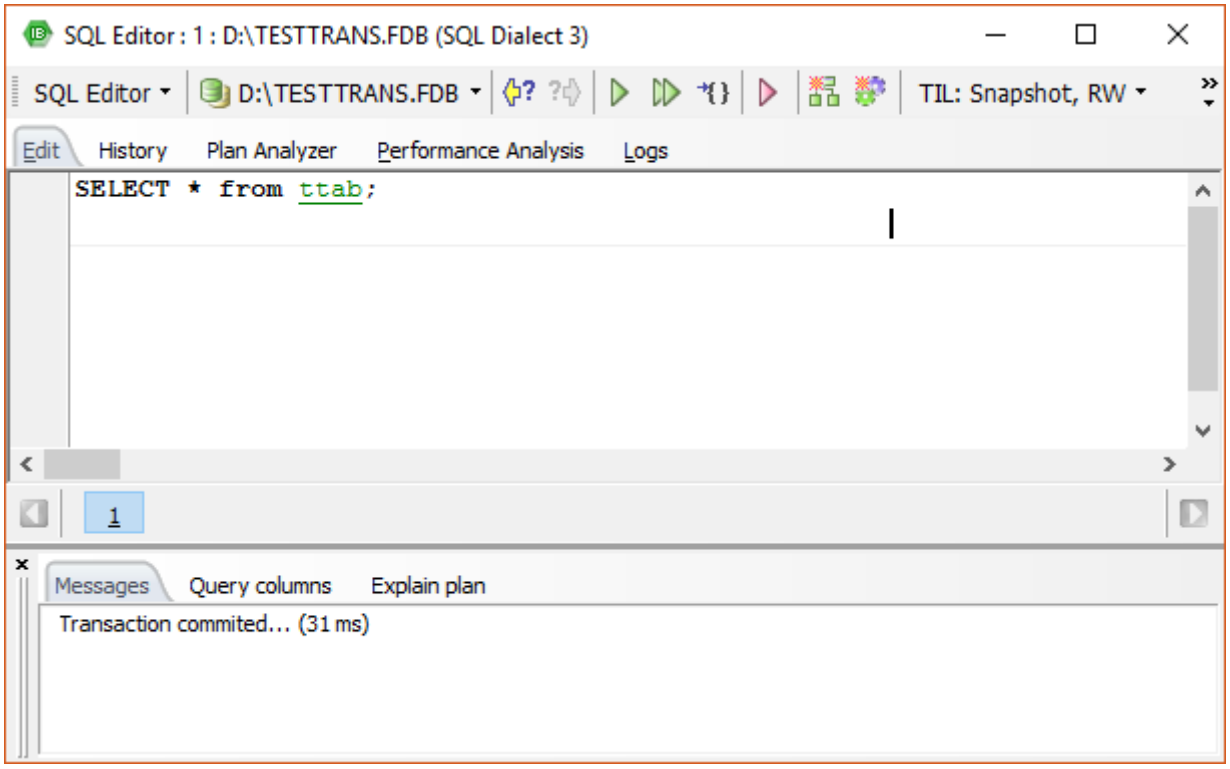

Рис. 9. Завершение первой транзакции

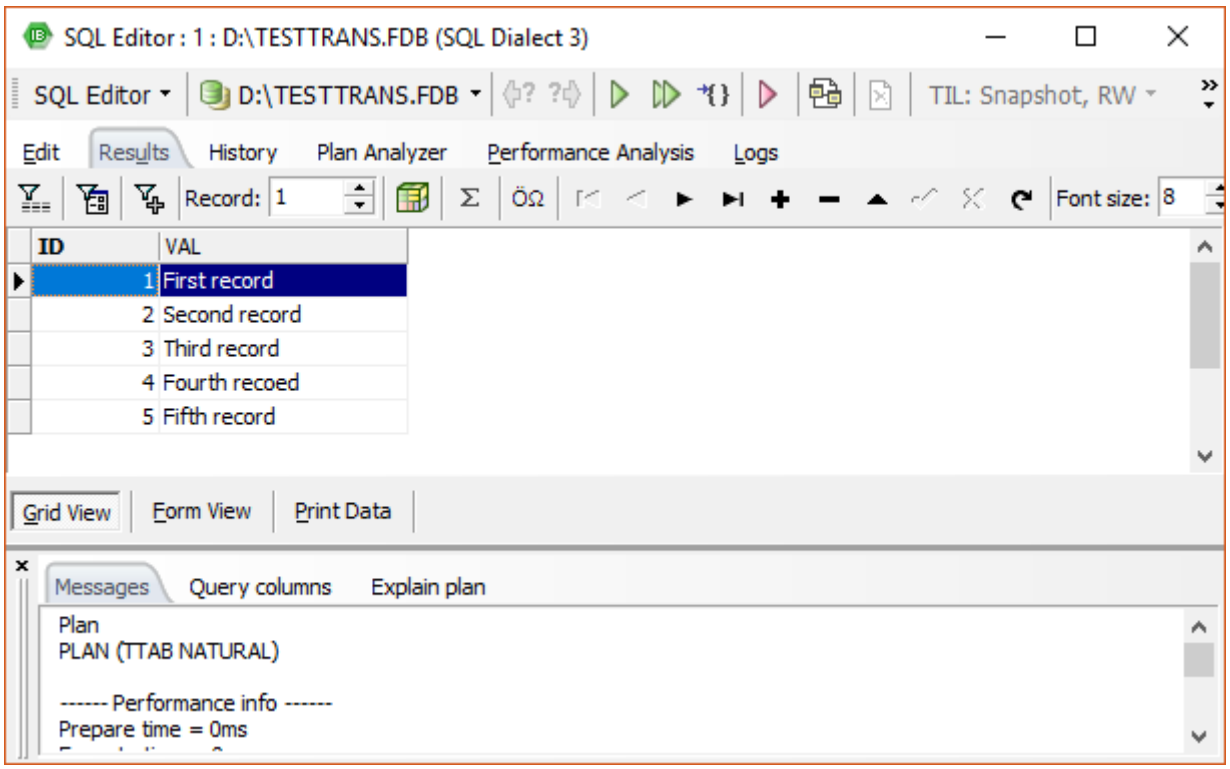

Рис. 10. После порождения новой транзакции в первом окне SQL Editor и вызова SELECT пятая запись стала видна

Эксперимент для SNAPSHOT TABLE STABILITY

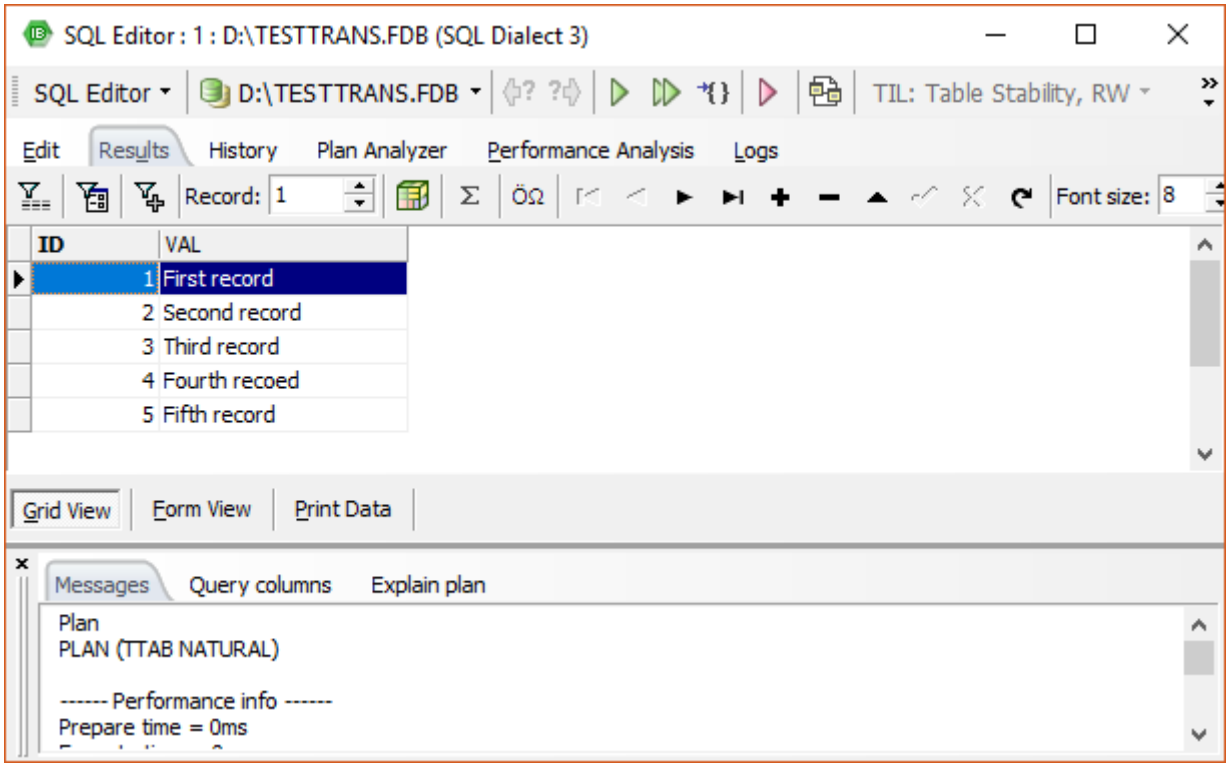

Рис. 11. Результат выполнения запроса SELECT в первой транзакции после установки уровня изоляции Snapshot Table Stability

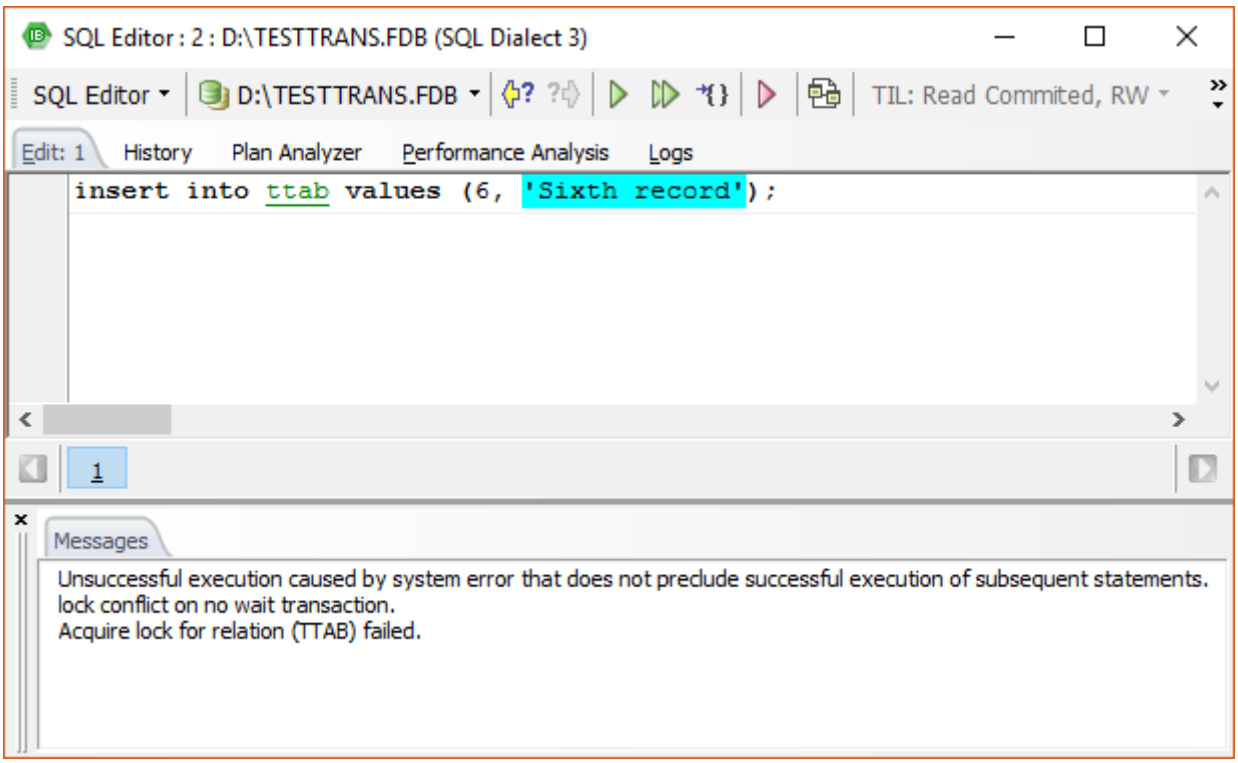

Рис. 12. Попытка внесения изменений второй транзакцией не удалась

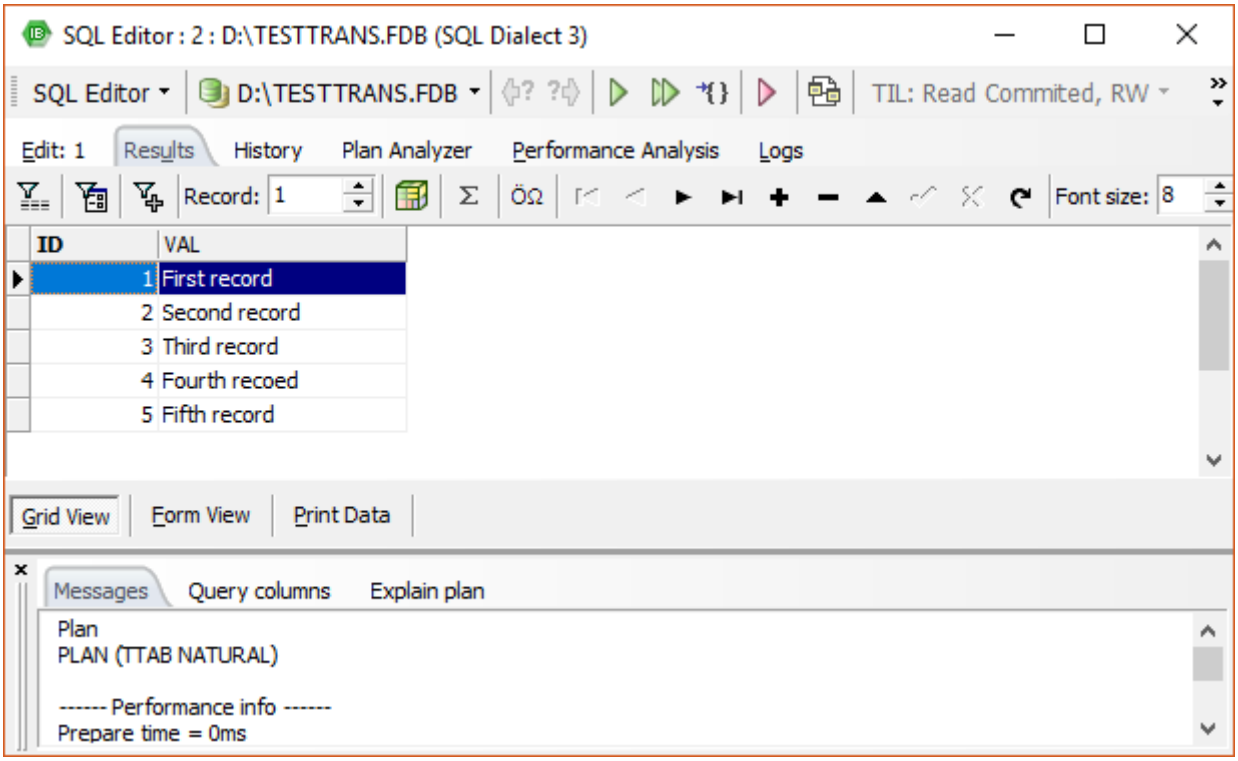

Рис. 13. Чтение из второй транзакции прошло успешно
(как и ожидалось согласно документации)

Таким образом, удалось на практике наблюдать работу всех трех поддерживаемых в Firebird уровней изоляции.

Выводы

Механизм транзакций играет очень важную роль при работе с базами данных. В современных СУБД все действия выполняются в рамках транзакций: изменения над данными остаются обратимыми до тех пор, пока клиентское приложение не выдаст серверу инструкцию COMMIT.

Транзакции должны обладать следующими свойствами («ACID»):

- Атомарность (Atomicity) – выполнение по принципу «все или ничего».
- Согласованность (Consistency) – в результате транзакции система переходит из одного целостного состояния в другое целостное состояние.
- Изолированность (Isolation) – неподтвержденные данные не должны быть видны другим транзакциям, пока изменения не будут завершены.
- Долговечность (Durability) – если транзакция зафиксирована, то ее результаты должны сохраняться даже в случае сбоев.

Возможность одновременного выполнения множества транзакций требует специальных механизмов для синхронизации и разрешения конфликтов между множеством клиентов при работе с общими данными. Для этого предусмотрены уровни изоляции, определяющие степень изолированности одной транзакции от другой. Чем больше эта степень, тем менее вероятно возникновение ошибочных ситуаций (грязное чтение, размытое чтение или фантомное чтение), но при этом и скорость работы уменьшается (чем больше уровень изоляции, тем меньше возможностей для распараллеливания). Выбор уровня изоляции необходимо осуществлять исходя из требований решаемой задачи.