

Санкт-Петербургский политехнический университет Петра Великого

Институт информационных технологий и управления

Кафедра компьютерных систем и программных технологий

ОТЧЕТ

По лабораторной работе №6

**«Изучение механизма транзакций»**

Базы данных

Студентка гр.43501/32: Бабуркина А.С.

Преподаватель: Мясов А. В.

Санкт-Петербург

2014

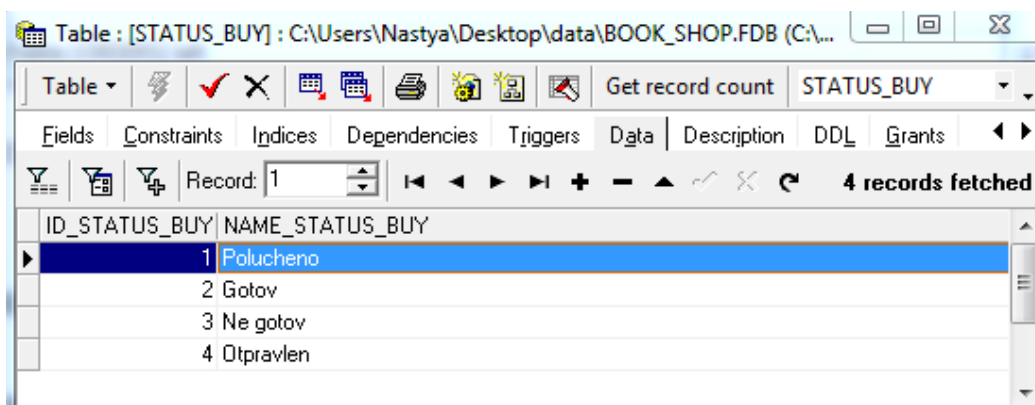
**1. Цели работы:** познакомиться с механизмом транзакций, возможностями ручного управления транзакциями, уровнями изоляции транзакций.

## 2. 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.

## 3. Выполнение работы:

### 3.1. Провести эксперименты по запуску, подтверждению и откату транзакций.

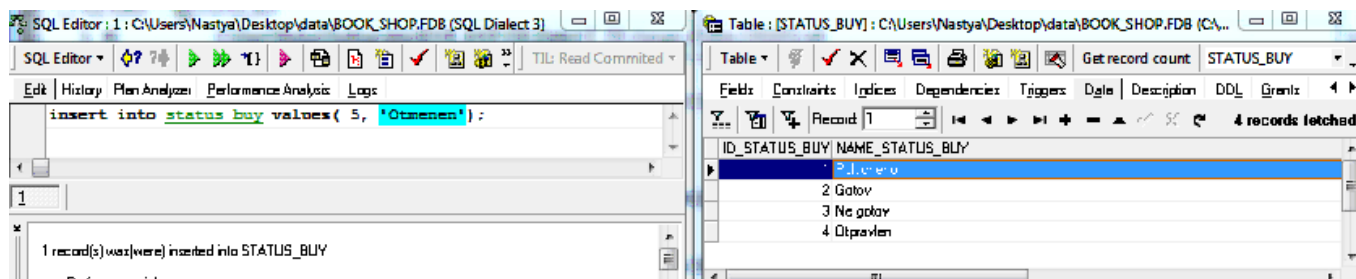


The screenshot shows the Firebird SQL Editor interface. The top toolbar includes icons for table operations and a 'Get record count' button. Below the toolbar, the 'Table' dropdown is set to 'STATUS\_BUY'. The 'Fields' tab is selected, showing the table structure with columns 'ID\_STATUS\_BUY' and 'NAME\_STATUS\_BUY'. The 'Record' dropdown is set to '1', and the status bar indicates '4 records fetched'. The table data is as follows:

ID_STATUS_BUY	NAME_STATUS_BUY
1	Polucheno
2	Gotov
3	Ne gotov
4	Otpravlenn

Рисунок 1. Таблица Status\_buy до проведения эксперимента

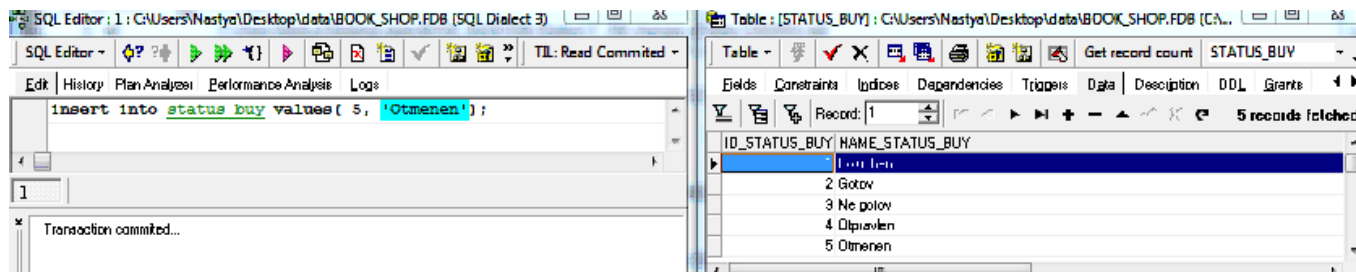
Начнем транзакцию вставив новую запись в таблицу Status\_buy, но не делая commit. В результате БД не была обновлена, так как транзакция еще не закончилась.



The screenshot shows the Firebird SQL Editor with an uncommitted insert operation. The SQL Editor window displays the command: `insert into status_buy values( 5, 'Otmennen');`. The status bar indicates '1 record(s) was(were) inserted into STATUS\_BUY'. The table view on the right shows the same 4 records as before, indicating that the new record has not been committed to the database.

Рисунок 2. Таблица Status\_buy после вставки записи

Для окончания транзакции отправим commit и еще раз проверим таблицу Status\_buy.



The screenshot shows the Firebird SQL Editor after the commit operation. The SQL Editor window displays the command: `insert into status_buy values( 5, 'Otmennen');`. The status bar indicates 'Transaction committed...'. The table view on the right shows the same 4 records as before, indicating that the new record has not been committed to the database.

Рисунок 3. Таблица Status\_buy после отправки команды commit

Теперь новая запись была добавлена, а БД перешла в новое целостное состояние. Транзакция закончена. Добавим в таблицу еще одну запись и сделаем rollback.

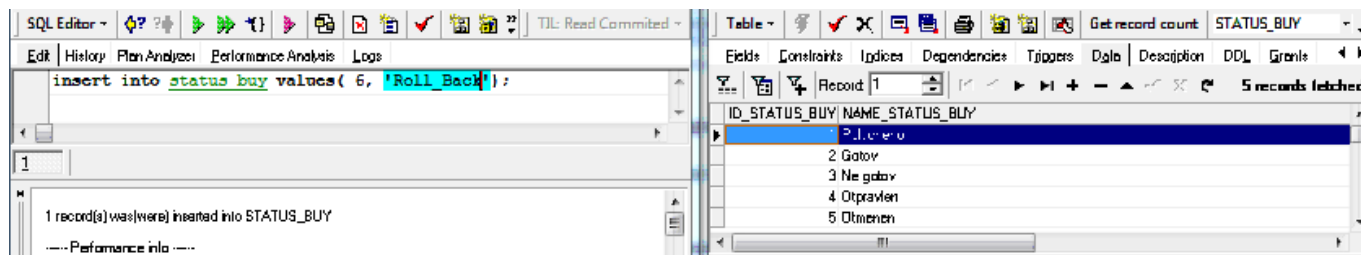


Рисунок 4. Добавление записи в таблицу

После обновления таблицы новая строка не появилась.

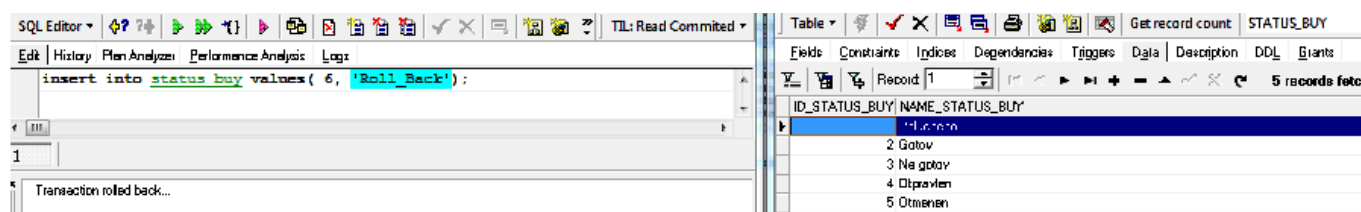


Рисунок 5. Откат транзакции

**3.2.** Спланировать и провести эксперименты, показывающие основные возможности транзакций с различным уровнем 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 в течение всей ее жизни.

Транзакции в Firebird 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** ( "читать подтвержденные данные"). Уровень изоляции READ COMMITTED 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 прочесть неподтвержденные данные, измененные в других транзакциях, и делает возможным прочесть подтвержденные данные.
- **SNAPSHOT**. 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**. Это уровень изоляции также создает "моментальный" снимок базы данных, но одновременно блокирует на запись 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

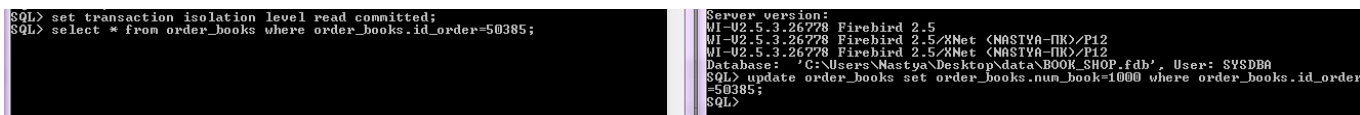
изменены в других параллельных транзакциях. Кроме того, транзакции с уровнем изоляции SNAPSHOT TABLE STABILITY не могут получить доступ к таблице, если данные в них уже изменяются в контексте других транзакций.

### 3.2.1. READ COMMITTED

В первой сессии задаем уровень изолированности READ COMMITTED командой:

```
set transaction isolation level read committed;
```

Затем во второй сессии модифицируем строку и не делаем commit. В результате при попытке прочитать ту же строку терминал первой сессии переходит в ожидание.

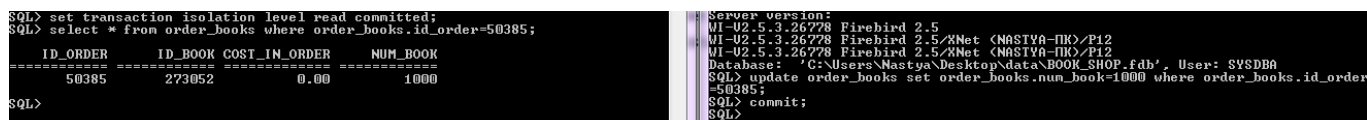


```
SQL> set transaction isolation level read committed;
SQL> select * from order_books where order_books.id_order=50385;

Server version:
MI-U2.5.3.26778 Firebird 2.5
MI-U2.5.3.26778 Firebird 2.5/Net (NASTYA-ПК)/P12
MI-U2.5.3.26778 Firebird 2.5/Net (NASTYA-ПК)/P12
Database: 'C:\Users\Nastya\Desktop\data\BOOK_SHOP.fdb', User: SYSDBA
SQL> update order_books set order_books.num_book=1000 where order_books.id_order=50385;
SQL>
```

Рисунок 6. READ COMMITTED. После обновления

Вторая сессия посылает commit, завершая транзакцию, тогда первая разблокируется и считывает необходимое значение.



```
SQL> set transaction isolation level read committed;
SQL> select * from order_books where order_books.id_order=50385;

ID_ORDER  ID_BOOK  COST_IN_ORDER  NUM_BOOK
=====
50385      273052      0.00          1000

SQL>

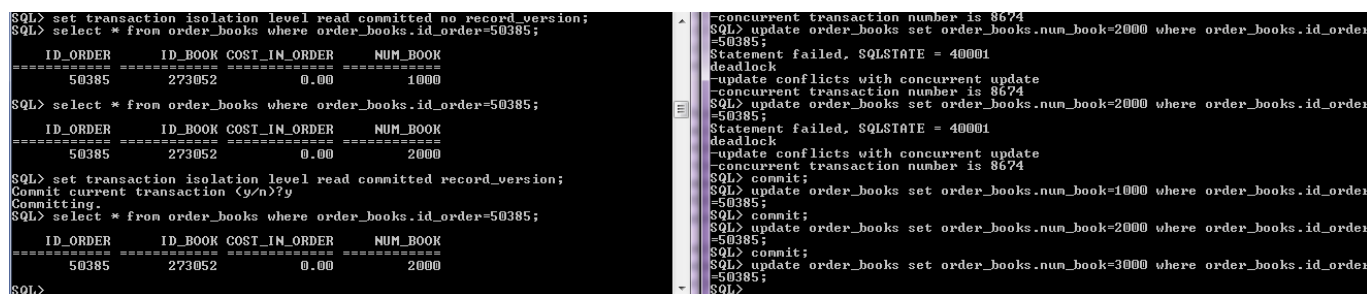
Server version:
MI-U2.5.3.26778 Firebird 2.5
MI-U2.5.3.26778 Firebird 2.5/Net (NASTYA-ПК)/P12
MI-U2.5.3.26778 Firebird 2.5/Net (NASTYA-ПК)/P12
Database: 'C:\Users\Nastya\Desktop\data\BOOK_SHOP.fdb', User: SYSDBA
SQL> update order_books set order_books.num_book=1000 where order_books.id_order=50385;
SQL> commit;
SQL>
```

Рисунок 7. READ COMMITTED. После commit

Сценарий эксперимента:

1. set transaction isolation level read committed;
2. update order\_books set order\_books.num\_book=1000 where order\_books.id\_order=50385;
1. select \* from order\_books where order\_books.id\_order=50385;
2. commit

Транзакции уровня изоляции Read Committed имеют в IB два режима - NO RECORD VERSION и RECORD VERSION. В первом случае, если при чтении записи ядро IB обнаруживает наличие неподтвержденной (uncommitted) версии этой записи, то возвращает сообщение о deadlock. В режиме RECORD VERSION наличие неподтвержденных версий записей игнорируется, и всегда возвращается старая версия записи.



```
SQL> set transaction isolation level read committed no record_version;
SQL> select * from order_books where order_books.id_order=50385;

ID_ORDER  ID_BOOK  COST_IN_ORDER  NUM_BOOK
=====
50385      273052      0.00          1000

SQL> select * from order_books where order_books.id_order=50385;

ID_ORDER  ID_BOOK  COST_IN_ORDER  NUM_BOOK
=====
50385      273052      0.00          2000

SQL> set transaction isolation level read committed record_version;
Commit current transaction (y/n)?y
Committing.
SQL> select * from order_books where order_books.id_order=50385;

ID_ORDER  ID_BOOK  COST_IN_ORDER  NUM_BOOK
=====
50385      273052      0.00          2000

SQL>

-concurrent transaction number is 8674
SQL> update order_books set order_books.num_book=2000 where order_books.id_order=50385;
Statement failed, SQLSTATE = 40001
deadlock
-update conflicts with concurrent update
-concurrent transaction number is 8674
SQL> update order_books set order_books.num_book=2000 where order_books.id_order=50385;
Statement failed, SQLSTATE = 40001
deadlock
-update conflicts with concurrent update
-concurrent transaction number is 8674
SQL> commit;
SQL> update order_books set order_books.num_book=1000 where order_books.id_order=50385;
SQL> commit;
SQL> update order_books set order_books.num_book=2000 where order_books.id_order=50385;
SQL> commit;
SQL> update order_books set order_books.num_book=3000 where order_books.id_order=50385;
SQL>
```

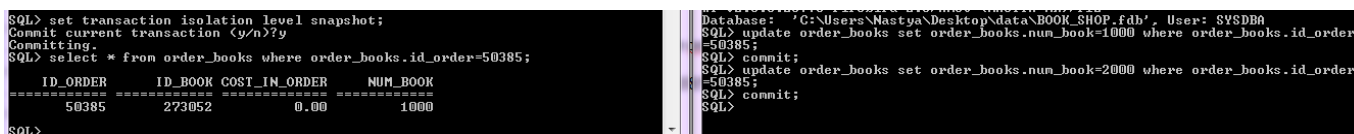
Рисунок 8. . READ COMMITTED. 2 режима

### 3.2.2. SNAPSHOT

Сначала установим уровень изоляции во второй сессии SNAPSHOT. В этот момент данной транзакцией был сделан некий "снимок" БД. Теперь когда в первой сессии изменим значение количества книг и сделаем commit во втором терминале все равно отобразятся старые значения, так как изменение строки произошло позже фиксации значений БД.

Сценарий эксперимента:

1. set transaction isolation level snapshot;
2. update order\_books set order\_books.num\_book=2000 where order\_books.id\_order=50385;
2. commit;
1. select \* from order\_books where order\_books.id\_order=50385;



```
SQL> set transaction isolation level snapshot;
Commit current transaction (y/n)?y
Committing.
SQL> select * from order_books where order_books.id_order=50385;

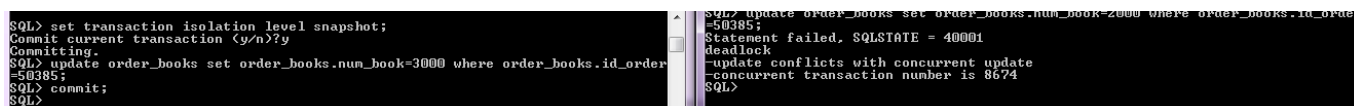
=====
ID_ORDER  ID_BOOK  COST_IN_ORDER  NUM_BOOK
=====
50385     273052         0.00         1000

SQL>

Database: 'C:\Users\Nastya\Desktop\data\BOOK_SHOP.fdb', User: SYSDBA
SQL> update order_books set order_books.num_book=1000 where order_books.id_order=50385;
SQL> commit;
SQL> update order_books set order_books.num_book=2000 where order_books.id_order=50385;
SQL> commit;
SQL>
```

Рисунок 9. SNAPSHOT. Результат эксперимента

При таком подходе возникает проблема одновременного изменения данных разными транзакциями. Пример, где возникает ошибка. Т.е. пока одна транзакция не завершит изменения, другая будет не в состоянии изменить эти данные.



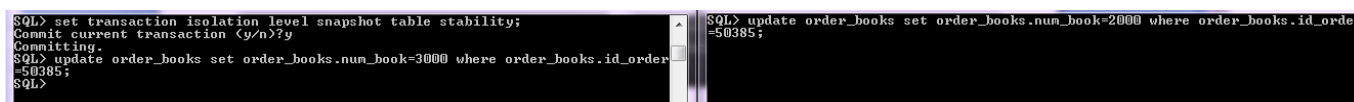
```
SQL> set transaction isolation level snapshot;
Commit current transaction (y/n)?y
Committing.
SQL> update order_books set order_books.num_book=3000 where order_books.id_order=50385;
SQL> commit;
SQL>

SQL> update order_books set order_books.num_book=2000 where order_books.id_order=50385;
Statement failed, SQLSTATE = 40001
deadlock
-update conflicts with concurrent update
-concurrent transaction number is 8674
SQL>
```

Рисунок 10. SNAPSHOT. Возникновение ошибки

### 3.2.3. SNAPSHOT TABLE STABILITY

Сначала установим уровень изоляции во второй сессии SNAPSHOT TABLE STABILITY и обновим строку в таблице заказов. Затем попробуем другим терминалом также модифицировать ту же запись. Из рисунков видно, что транзакция перешла в ожидание.

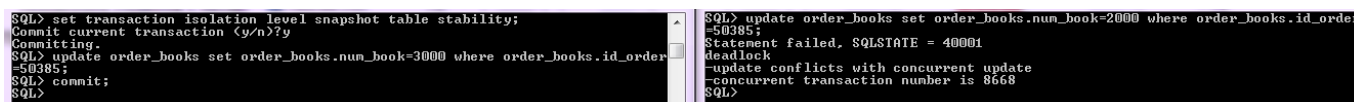


```
SQL> set transaction isolation level snapshot table stability;
Commit current transaction (y/n)?y
Committing.
SQL> update order_books set order_books.num_book=3000 where order_books.id_order=50385;
SQL>

SQL> update order_books set order_books.num_book=2000 where order_books.id_order=50385;
```

Рисунок 11. SNAPSHOT TABLE STABILITY. Попытка модифицировать одну строку

После завершения транзакции во второй сессии, первая получает сообщение об ошибке( deadlock).



```
SQL> set transaction isolation level snapshot table stability;
Commit current transaction (y/n)?y
Committing.
SQL> update order_books set order_books.num_book=3000 where order_books.id_order=50385;
SQL> commit;
SQL>

SQL> update order_books set order_books.num_book=2000 where order_books.id_order=50385;
Statement failed, SQLSTATE = 40001
deadlock
-update conflicts with concurrent update
-concurrent transaction number is 8668
SQL>
```

Рисунок 12. SNAPSHOT TABLE STABILITY. Завершение транзакции.

Из рисунка ниже видно, что изменения из первого терминала так и не вступили в силу.

```
SQL> select * from order_books where order_books.id_order=50385;
=====
ID_ORDER  ID_BOOK  COST_IN_ORDER  NUM_BOOK
=====
50385     273052      0.00         3000
SQL>
```

Рисунок 13. SNAPSHOT TABLE STABILITY.

Сценарий эксперимента:

1. set transaction isolation level snapshot TABLE STABILITY;
1. update order\_books set order\_books.num\_book=3000 where order\_books.id\_order=50385;
2. update order\_books set order\_books.num\_book=2000 where order\_books.id\_order=50385;
1. commit;

### Выводы:

Транзакция - это неделимая последовательность операций манипулирования данными. 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).

Транзакция обладает четырьмя свойствами:

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

Уровень изолированности транзакций — значение, определяющее уровень, при котором в транзакции допускаются несогласованные данные. 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.

К достоинствам транзакций можно отнести: транзакции позволяют обеспечить логическую целостность данных в БД( после выполнения транзакций БД остается в целостном состоянии), обеспечивают правильность работы в системах при параллельном обращении нескольких пользователей к одним и тем же данным.