

Лабораторная работа №8

Тема: Глава 9 из [книги](#). Упр 2, 3

Группа: M8O-109CB-24

Выполнил: **Гимазетдинов Дмитрий Русланович**

[вернуться на главную](#)

Упражнение 2

Дано:

Транзакции, работающие на уровне изоляции Read Committed, видят только свои собственные обновления и обновления, зафиксированные параллельными транзакциями. При этом нужно учитывать, что иногда могут возникать ситуации, которые на первый взгляд кажутся парадоксальными, но на самом деле все происходит в строгом соответствии с этим принципом.

Воспользуемся таблицей «Самолеты» (aircrafts) или ее копией. Предположим, что мы решили удалить из таблицы те модели, дальность полета которых менее 2 000 км. В таблице представлена одна такая модель — Cessna 208 Caravan, имеющая дальность полета 1 200 км. Для выполнения удаления мы организовали транзакцию. Однако параллельная транзакция, которая, причем, началась раньше, успела обновить таблицу таким образом, что дальность полета самолета Cessna 208 Caravan стала составлять 2 100 км, а вот для самолета Bombardier CRJ-200 она, напротив, уменьшилась до 1 900 км. Таким образом, в результате выполнения операций обновления в таблице по-прежнему присутствует строка, удовлетворяющая первоначальному условию, т. е. значение атрибута range у которой меньше 2000. Наша задача: проверить, будет ли в результате выполнения двух транзакций удалена какая-либо строка из таблицы.

На первом терминале начнем транзакцию, при этом уровень изоляции Read Committed в команде указывать не будем, т. к. он принят по умолчанию:

```
BEGIN;
```

```
SELECT *  
FROM aircrafts_tmp  
WHERE range < 2000;
```

```
UPDATE aircrafts_tmp  
SET range = 2100  
WHERE aircraft_code = 'CN1';
```

```
UPDATE aircrafts_tmp  
SET range = 1900  
WHERE aircraft_code = 'CR2';
```

На втором терминале начнем вторую транзакцию, которая и будет пытаться удалить строки, у которых значение атрибута range меньше 2000.

```
BEGIN;
```

```
SELECT *  
FROM aircrafts_tmp  
WHERE range < 2000;
```

```
DELETE FROM aircrafts_tmp WHERE range < 2000;
```

Введя команду DELETE, мы видим, что она не завершается, а ожидает, когда со строки, подлежащей удалению, будет снята блокировка. Блокировка, установленная командой UPDATE в первой транзакции, снимается только при завершении транзакции, а завершение может иметь два исхода: фиксацию изменений с помощью команды COMMIT (или END) или отмену изменений с помощью команды ROLLBACK.

Давайте зафиксируем изменения, выполненные первой транзакцией. На первом терминале сделаем так:

```
COMMIT;
```

Тогда на втором терминале мы получим такой результат от команды DELETE:

```
DELETE 0
```

Чем объясняется такой результат? Он кажется нелогичным: ведь команда SELECT, выполненная в этой же второй транзакции, показывала наличие строки, удовлетворяющей условию удаления. Объяснение таково: поскольку вторая транзакция пока еще не видит изменений, произведенных в первой транзакции, то команда DELETE выбирает для удаления строку, описывающую модель Cessna 208 Caravan, однако эта строка была заблокирована в первой транзакции командой UPDATE. Эта команда изменила значение атрибута range в этой строке.

При завершении первой транзакции блокировка с этой строки снимается (со второй строки — тоже), и команда DELETE во второй транзакции получает возможность заблокировать эту строку. При этом

команда DELETE данную строку перечитывает и вновь вычисляет условие WHERE применительно к ней. Однако теперь условие WHERE для данной строки уже не выполняется, следовательно, эту строку удалять нельзя. Конечно, в таблице есть теперь другая строка, для самолета Bombardier CRJ-200, удовлетворяющая условию удаления, однако повторный поиск строк, удовлетворяющих условию WHERE в команде DELETE, не производится.

В результате не удаляется ни одна строка. Таким образом, к сожалению, имеет место нарушение согласованности, которое можно объяснить деталями реализации СУБД.

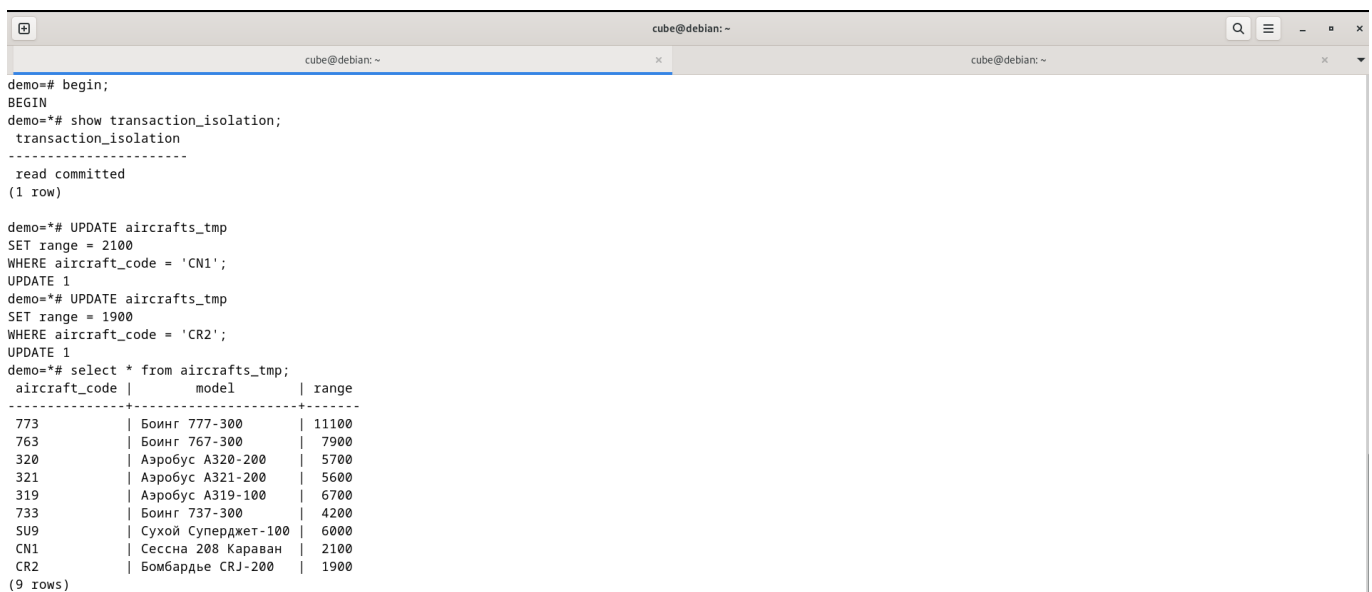
Завершим вторую транзакцию:

```
END;
```

Задание. Модифицируйте сценарий выполнения транзакций: в первой транзакции вместо фиксации изменений выполните их отмену с помощью команды ROLLBACK и посмотрите, будет ли удалена строка и какая конкретно.

Решение:

Создадим транзакцию (**bash-1**) и выполним команды обновления в таблице **aircrafts_tmp**:



```
cube@debian: ~  
demo=# begin;  
BEGIN  
demo=# show transaction_isolation;  
transaction_isolation  
-----  
read committed  
(1 row)  
  
demo=# UPDATE aircrafts_tmp  
SET range = 2100  
WHERE aircraft_code = 'CN1';  
UPDATE 1  
demo=# UPDATE aircrafts_tmp  
SET range = 1900  
WHERE aircraft_code = 'CR2';  
UPDATE 1  
demo=# select * from aircrafts_tmp;  
aircraft_code | model | range  
-----  
773 | Боинг 777-300 | 11100  
763 | Боинг 767-300 | 7900  
320 | Аэробус A320-200 | 5700  
321 | Аэробус A321-200 | 5600  
319 | Аэробус A319-100 | 6700  
733 | Боинг 737-300 | 4200  
SU9 | Сухой Суперджет-100 | 6000  
CN1 | Сессна 208 Караван | 2100  
CR2 | Бомбардье CRJ-200 | 1900  
(9 rows)
```

Теперь проведем новую транзакцию для второго пользователя (**bash-2**):

```
cube@debian: ~  
demo=# begin;  
BEGIN  
demo=# show tra  
track_activities      track_counts      track_wal_io_timing  transaction_read_only  
track_activity_query_size track_functions    transaction_deferrable transform_null_equals  
track_commit_timestamp track_io_timing    transaction_isolation  
demo=# show tra  
track_activities      track_counts      track_wal_io_timing  transaction_read_only  
track_activity_query_size track_functions    transaction_deferrable transform_null_equals  
track_commit_timestamp track_io_timing    transaction_isolation  
demo=# show transaction_isolation;  
-----  
read committed  
(1 row)  
  
demo=# SELECT *  
FROM aircrafts_tmp  
WHERE range < 2000;  
aircraft_code | model          | range  
-----  
CN1           | Сессна 208 Караван | 1200  
(1 row)  
  
demo=# DELETE FROM aircrafts_tmp WHERE range < 2000;  
█
```

Как видим, выполнение второй транзакции застыла, т.к. мы заблокировали доступ на изменения первой транзакцией!

Теперь отменим первую транзакцию:

```
bash-1  
demo=# rollback;  
ROLLBACK
```

```
demo=# DELETE FROM aircrafts_tmp WHERE range < 2000;  
DELETE 1
```

На втором терминале, мы получили успешное выполнение команды. Один элемент удален. Давайте посмотрим какой именно:

```
cube@debian: ~  
demo=# begin;  
BEGIN  
demo=# show tra  
track_activities      track_counts      track_wal_io_timing  transaction_read_only  
track_activity_query_size track_functions    transaction_deferrable transform_null_equals  
track_commit_timestamp track_io_timing    transaction_isolation  
demo=# show tra  
track_activities      track_counts      track_wal_io_timing  transaction_read_only  
track_activity_query_size track_functions    transaction_deferrable transform_null_equals  
track_commit_timestamp track_io_timing    transaction_isolation  
demo=# show transaction_isolation;  
-----  
read committed  
(1 row)  
  
demo=# SELECT *  
FROM aircrafts_tmp  
WHERE range < 2000;  
aircraft_code | model          | range  
-----  
CN1           | Сессна 208 Караван | 1200  
(1 row)  
  
demo=# DELETE FROM aircrafts_tmp WHERE range < 2000;  
█
```

Удалилась нужная строка!!!

Упражнение 3

Дано:

Когда говорят о таком феномене, как потерянное обновление, то зачастую в качестве примера приводится операция UPDATE, в которой значение какого-то атрибута изменяется с применением одного из действий арифметики. Например:

```
UPDATE aircrafts_tmp
SET range = range + 200
WHERE aircraft_code = 'CR2';
```

При выполнении двух и более подобных обновлений в рамках параллельных транзакций, использующих, например, уровень изоляции Read Committed, будут учтены все такие изменения (что и было показано в тексте главы). Очевидно, что потерянного обновления не происходит.

Предположим, что в одной транзакции будет просто присваиваться новое значение, например, так:

```
UPDATE aircrafts_tmp
SET range = 2100
WHERE aircraft_code = 'CR2';
```

А в параллельной транзакции будет выполняться аналогичная команда:

```
UPDATE aircrafts_tmp
SET range = 2500
WHERE aircraft_code = 'CR2';
```

Очевидно, что сохранится только одно из значений атрибута range. Можно ли говорить, что в такой ситуации имеет место потерянное обновление? Если оно имеет место, то что можно предпринять для его недопущения? Обоснуйте ваш ответ.

Решение:

Потерянное обновление (Lost Update) — это ситуация, при которой два параллельных процесса читают одно и то же значение из базы данных, модифицируют его и записывают новое значение обратно. В этом случае одно из обновлений перезаписывается другим, что приводит к потере первого изменения. Это происходит, когда оба процесса не знают о существовании другого обновления, и результат одного обновления теряется.

В нашем примере:

```
UPDATE aircrafts_tmp  
SET range = 2100  
WHERE aircraft_code = 'CR2';
```

и параллельно:

```
UPDATE aircrafts_tmp  
SET range = 2500  
WHERE aircraft_code = 'CR2';
```

Обе транзакции модифицируют одно и то же поле `range` для записи с `aircraft_code = 'CR2'`. Если транзакции выполняются параллельно при уровне изоляции **Read Committed**, каждая транзакция видит только **закоммиченные** изменения других транзакций. Однако в этой ситуации каждая транзакция может видеть исходное значение поля `range`, а затем перезаписать его новым значением, не зная о промежуточных изменениях, которые могла сделать другая транзакция. В результате одно из значений (либо 2100, либо 2500) будет потеряно, а другая транзакция "победит".

Является ли это потерянным обновлением? Да, в данном случае можно говорить о **потерянном обновлении**. Это классическая ситуация потерянного обновления, так как одна транзакция перезапишет изменения другой транзакции, не зная об этом. Сохранится только одно значение, а другое будет утрачено.

Как предотвратить потерянное обновление

1. Уровень изоляции **SERIALIZABLE**

Этот уровень изоляции обеспечивает выполнение транзакций так, как если бы они шли последовательно, предотвращая конфликты. При обнаружении конфликта транзакция откатывается с ошибкой "Serialization failure", что требует повторной попытки.

2. **FOR UPDATE**

Использование `SELECT ... FOR UPDATE` блокирует строку на чтение, предотвращая параллельные изменения других транзакций до завершения текущей.

3. **Оптимистическая блокировка**

Добавление поля версии строки и проверка его значения перед обновлением. Если версия изменилась (из-за другой транзакции), обновление отклоняется, и транзакция повторяется.