

**ФГАОУ ВО «МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ»**

Лабораторная работа №6

Транзакции и согласованность базы данных

По дисциплине:

Базы данных

Выполнил студент 1-го курса группы 243-323

Онищенко А. А.

Проверил

_____ Красникова И.Н.

Москва, 2024

Пример транзакции

• Упражнение 6.1

Начните транзакцию (командой BEGIN) и создайте новое бронирование в таблице bookings сегодняшней датой. Добавьте два электронных билета в таблицу tickets, связанных с созданным бронированием.

Представьте, что пользователь не подтвердил бронирование и все введенные данные необходимо отменить. Выполните отмену транзакции и проверьте, что никакой добавленной вами информации действительно не осталось.

The screenshot displays the pgAdmin 4 interface with a transaction script in the Query tab. The script performs the following actions:

- Declares a variable `failed_book_ref` of type `character(6)`.
- Starts a transaction with `BEGIN`.
- Inserts a new booking into the `bookings` table with `book_ref` '000004', `book_date` '2015-10-29 10:08:00+03', and `total_amount` 95800.00. The `RETURNING` clause assigns the `book_ref` to `failed_book_ref`.
- Inserts two tickets into the `tickets` table, both linked to `book_ref` '000004'.
- Attempts to commit the transaction with `ROLLBACK` (labeled as 'пользователь не подтвердил' - user did not confirm).
- Ends the transaction with `END $$`.

Below the script, the 'Data Output' tab shows the results of the transaction. The first query (the `INSERT INTO bookings`) returned 5 rows:

| book_ref [PK] character(6) | book_date timestamp with time zone | total_amount numeric(10,2) |
|-------------------------------|---------------------------------------|-------------------------------|
| 1 000004 | 2015-10-12 14:40:00+03 | 55800.00 |
| 2 00000F | 2016-09-02 02:12:00+03 | 265700.00 |
| 3 000010 | 2016-03-08 18:45:00+03 | 50900.00 |
| 4 000012 | 2016-09-11 08:02:00+03 | 37900.00 |
| 5 000025 | 2015-10-29 10:08:00+03 | 95800.00 |

The second query (the `SELECT` statements) returned 5 rows, showing the state of the `bookings` and `tickets` tables after the transaction. The `bookings` table shows the new booking with `book_ref` '000004' and `total_amount` 95800.00. The `tickets` table shows two tickets linked to `book_ref` '000004'.

pgAdmin 4

File Object Tools Edit View Window Help

Welcome demo/postgres@PostgreSQL 17 X

demo/postgres@PostgreSQL 17

Query Query History

```

1  -- заведем переменную
2  DO $$
3  DECLARE failed_book_ref character(6);
4
5  -- начало транзакции
6  BEGIN
7
8  -- новое бронирование
9  INSERT INTO bookings.bookings (book_ref, book_date, total_amount)
10 VALUES ('000002', CURRENT_DATE, 987654.00)
11 RETURNING book_ref INTO failed_book_ref;
12
13 -- два новых билета
14 INSERT INTO bookings.tickets (ticket_no, book_ref, passenger_id, passenger_name, contact_data)
15 VALUES
16 ('0005432001234', failed_book_ref, '1234 567890', 'Pipurkov Ivan', '{"email": "pipurkov@mail.ru"}'),
17 ('0005432001235', failed_book_ref, '1234 567890', 'Pipurkova Ivanesa', '{"email": "pipurkova@mail.ru"}');
18
19 -- пользователь не подтвердил
20 ROLLBACK;
21
22 END $$;
23
24 SELECT * FROM bookings.bookings ORDER BY book_ref LIMIT 5;
25 SELECT * FROM bookings.tickets ORDER BY ticket_no LIMIT 5;
26
27 SELECT * FROM bookings.bookings WHERE book_ref = failed_book_ref;
28 SELECT * FROM bookings.tickets WHERE book_ref = failed_book_ref;

```

Data Output Messages Notifications

Showing rows: 1 to 5 Page No: 1 of 1

| ticket_no [PK] character (13) | book_ref character (6) | passenger_id character varying (20) | passenger_name text | contact_data jsonb |
|-------------------------------------|---------------------------|--|------------------------|---|
| 1 0005432000284 | 1A40A1 | 4030 855525 | MIKHAIL SEMENOV | {"phone": "+70110137563"} |
| 2 0005432000285 | 12736D | 8360 311602 | ELENA ZAKHAROVA | {"phone": "+705670013989"} |
| 3 0005432000286 | DC89BC | 4510 377533 | ILYA PAVLOV | {"phone": "+70624013335"} |
| 4 0005432000287 | CDE0BB | 5952 253588 | ELENA BELOVA | {"email": "e.belova.07121974@postgresspro.ru", "phone": "+70340423946"} |
| 5 0005432000288 | BEF9D0 | 4313 788533 | VYACHESLAV IVANOV | {"email": "vyacheslav-ivanov051968@postgresspro.ru", "phone": "+70417078841"} |

Total rows: 5 Query complete 00:00:00.047

CRLF Ln 25, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

Welcome demo/postgres@PostgreSQL 17 X

demo/postgres@PostgreSQL 17

Query Query History

```

3  DECLARE failed_book_ref character(6);
4
5  -- начало транзакции
6  BEGIN
7
8  -- новое бронирование
9  INSERT INTO bookings.bookings (book_ref, book_date, total_amount)
10 VALUES ('000002', CURRENT_DATE, 987654.00)
11 RETURNING book_ref INTO failed_book_ref;
12
13 -- два новых билета
14 INSERT INTO bookings.tickets (ticket_no, book_ref, passenger_id, passenger_name, contact_data)
15 VALUES
16 ('0005432001234', failed_book_ref, '1234 567890', 'Pipurkov Ivan', '{"email": "pipurkov@mail.ru"}'),
17 ('0005432001235', failed_book_ref, '1234 567890', 'Pipurkova Ivanesa', '{"email": "pipurkova@mail.ru"}');
18
19 -- пользователь не подтвердил
20 ROLLBACK;
21
22 END $$;
23
24 SELECT * FROM bookings.bookings ORDER BY book_ref LIMIT 5;
25 SELECT * FROM bookings.tickets ORDER BY ticket_no LIMIT 5;
26
27 SELECT * FROM bookings.bookings WHERE book_ref = failed_book_ref;
28 SELECT * FROM bookings.tickets WHERE book_ref = failed_book_ref;
29
30

```

Data Output Messages Notifications

ERROR: column "failed_book_ref" не существует
LINE 1: SELECT * FROM bookings.bookings WHERE book_ref = failed_book...

DETAIL: column "failed_book_ref" не существует
SQL state: 42703
Character: 58

Total rows: 5 Query complete 00:00:00.036

CRLF Ln 27, Col 66

• Упражнение 6.2

Теперь представьте сценарий, в котором нужно отменить не все данные, а только последний из добавленных электронных билетов. Для этого повторите все действия из предыдущего упражнения, но перед добавлением каждого билета создавайте точку сохранения (с одним и тем же именем). После ввода второго билета выполните откат к точке сохранения. Проверьте, что бронирование и первый билет остались.

```
-- kaxano tpaspassage
BEGIN;

-- kaxano tpaspassage
INSERT INTO bookings.bookings (book_ref, book_date, total_amount)
VALUES ('000002', CURRENT_DATE, 987654.00);

-- kaxano tpaspassage
SAVEPOINT sp_first;

-- kaxano tpaspassage
INSERT INTO bookings.tickets (ticket_no, book_ref, passenger_id, passenger_name, contact_data)
VALUES ('0005432001234', '000002', '1234 567890', 'Pipurkov Ivan', '{"email": "pipurkov@mail.ru"}');

-- kaxano tpaspassage
SAVEPOINT sp_second;

-- kaxano tpaspassage
INSERT INTO bookings.tickets (ticket_no, book_ref, passenger_id, passenger_name, contact_data)
VALUES ('0005432001235', '000002', '1234 567890', 'Pipurkova Ivanesa', '{"email": "pipurkova@mail.ru"}');

-- kaxano tpaspassage
ROLLBACK TO sp_second;
COMMIT;
```

Data Output Messages Notifications

COMMIT

Query returned successfully in 29 msec.

```
INSERT INTO bookings.bookings (book_ref, book_date, total_amount)
VALUES ('000002', CURRENT_DATE, 987654.00);

-- kaxano tpaspassage
SAVEPOINT sp_first;

-- kaxano tpaspassage
INSERT INTO bookings.tickets (ticket_no, book_ref, passenger_id, passenger_name, contact_data)
VALUES ('0005432001234', '000002', '1234 567890', 'Pipurkov Ivan', '{"email": "pipurkov@mail.ru"}');

-- kaxano tpaspassage
SAVEPOINT sp_second;

-- kaxano tpaspassage
INSERT INTO bookings.tickets (ticket_no, book_ref, passenger_id, passenger_name, contact_data)
VALUES ('0005432001235', '000002', '1234 567890', 'Pipurkova Ivanesa', '{"email": "pipurkova@mail.ru"}');

-- kaxano tpaspassage
ROLLBACK TO sp_second;
COMMIT;

SELECT * FROM bookings.bookings WHERE book_ref = '000002';
SELECT * FROM bookings.tickets WHERE book_ref = '000002';
```

Data Output Messages Notifications

| book_ref | book_date | total_amount |
|--------------------|--------------------------|--------------|
| [PQ] character (3) | Timestamp with time zone | numeric (12) |
| 1 | 2025-05-05 00:00:00+03 | 987654.00 |

```
INSERT INTO bookings.bookings (book_ref, book_date, total_amount)
VALUES ('000002', CURRENT_DATE, 987654.00);

-- kaxano tpaspassage
SAVEPOINT sp_first;

-- kaxano tpaspassage
INSERT INTO bookings.tickets (ticket_no, book_ref, passenger_id, passenger_name, contact_data)
VALUES ('0005432001234', '000002', '1234 567890', 'Pipurkov Ivan', '{"email": "pipurkov@mail.ru"}');

-- kaxano tpaspassage
SAVEPOINT sp_second;

-- kaxano tpaspassage
INSERT INTO bookings.tickets (ticket_no, book_ref, passenger_id, passenger_name, contact_data)
VALUES ('0005432001235', '000002', '1234 567890', 'Pipurkova Ivanesa', '{"email": "pipurkova@mail.ru"}');

-- kaxano tpaspassage
ROLLBACK TO sp_second;
COMMIT;

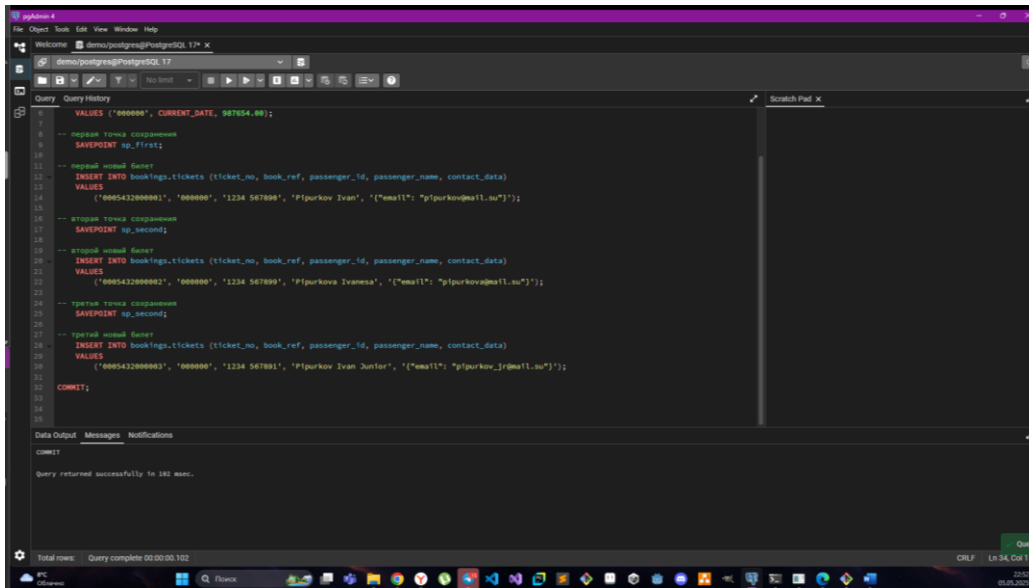
SELECT * FROM bookings.bookings WHERE book_ref = '000002';
SELECT * FROM bookings.tickets WHERE book_ref = '000002';
```

Data Output Messages Notifications

| ticket_no | book_ref | passenger_id | passenger_name | contact_data | |
|---------------------|---------------|--------------|----------------|---------------|---------------------------------|
| [PQ] character (13) | character (3) | text | text | jsonb | |
| 1 | 0005432001234 | 000002 | 1234 567890 | Pipurkov Ivan | '{"email": "pipurkov@mail.ru"}' |

• Упражнение 6.3

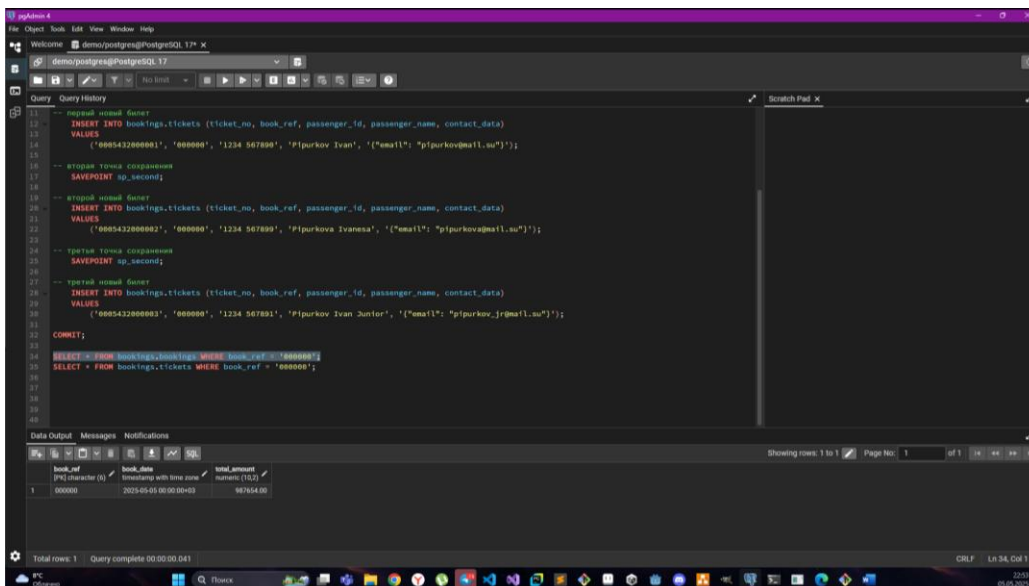
В рамках той же транзакции добавьте еще один электронный билет и зафиксируйте транзакцию. Обратите внимание на то, что после этой операции отменить внесенные транзакцией изменения будет уже невозможно.



```
1  VALUES ('000000', CURRENT_DATE, 987654.00);
2
3  -- rollback точка сохранения
4  SAVEPOINT sp_first;
5
6  -- rollback второй билет
7  INSERT INTO bookings.tickets (ticket_no, book_ref, passenger_id, passenger_name, contact_data)
8  VALUES
9  ('0005432000001', '000000', '1234 567890', 'Pipurkov Ivan', ('email': 'pipurkov@mail.ru'));
10
11 -- stopan точка сохранения
12 SAVEPOINT sp_second;
13
14 -- rollback второй билет
15 INSERT INTO bookings.tickets (ticket_no, book_ref, passenger_id, passenger_name, contact_data)
16 VALUES
17 ('0005432000002', '000000', '1234 567890', 'Pipurkova Ivanesa', ('email': 'pipurkova@mail.ru'));
18
19 -- точка точка сохранения
20 SAVEPOINT sp_second;
21
22 -- rollback второй билет
23 INSERT INTO bookings.tickets (ticket_no, book_ref, passenger_id, passenger_name, contact_data)
24 VALUES
25 ('0005432000003', '000000', '1234 567891', 'Pipurkov Ivan Junior', ('email': 'pipurkov_jr@mail.ru'));
26
27 COMMIT;
```

COMMIT

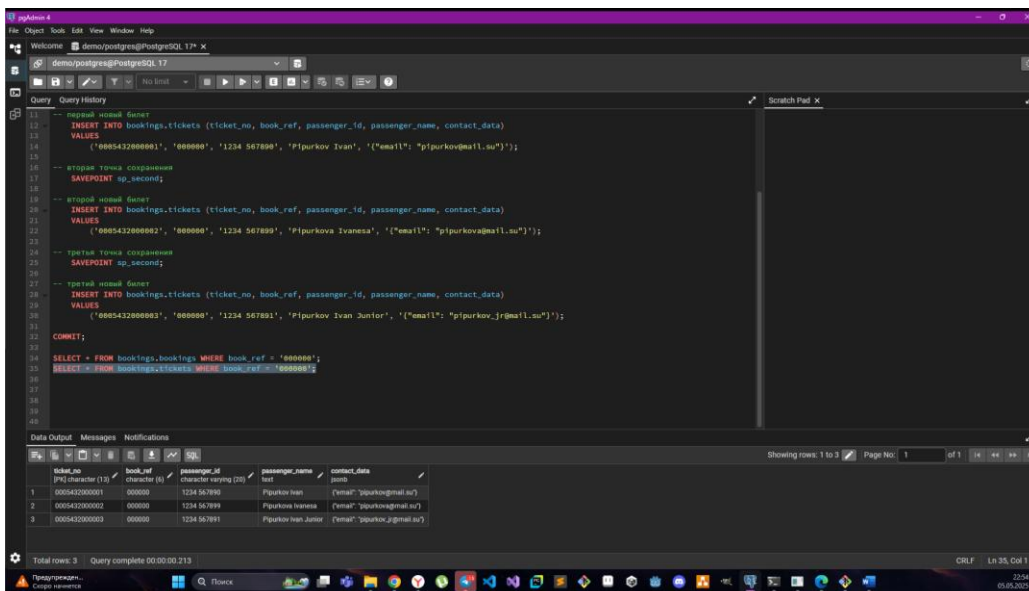
Query returned successfully in 182 msec.



```
1  -- rollback второй билет
2  INSERT INTO bookings.tickets (ticket_no, book_ref, passenger_id, passenger_name, contact_data)
3  VALUES
4  ('0005432000001', '000000', '1234 567890', 'Pipurkov Ivan', ('email': 'pipurkov@mail.ru'));
5
6  -- stopan точка сохранения
7  SAVEPOINT sp_second;
8
9  -- rollback второй билет
10 INSERT INTO bookings.tickets (ticket_no, book_ref, passenger_id, passenger_name, contact_data)
11 VALUES
12 ('0005432000002', '000000', '1234 567890', 'Pipurkova Ivanesa', ('email': 'pipurkova@mail.ru'));
13
14 -- точка точка сохранения
15 SAVEPOINT sp_second;
16
17 -- rollback второй билет
18 INSERT INTO bookings.tickets (ticket_no, book_ref, passenger_id, passenger_name, contact_data)
19 VALUES
20 ('0005432000003', '000000', '1234 567891', 'Pipurkov Ivan Junior', ('email': 'pipurkov_jr@mail.ru'));
21
22 COMMIT;
```

```
23 SELECT * FROM bookings.tickets WHERE book_ref = '000000';
24 SELECT * FROM bookings.tickets WHERE book_ref = '000000';
```

| book_ref | book_date | total_amount |
|----------|------------------------|--------------|
| 000000 | 2023-05-01 00:00:00+03 | 987654.00 |



```
1  -- rollback второй билет
2  INSERT INTO bookings.tickets (ticket_no, book_ref, passenger_id, passenger_name, contact_data)
3  VALUES
4  ('0005432000001', '000000', '1234 567890', 'Pipurkov Ivan', ('email': 'pipurkov@mail.ru'));
5
6  -- stopan точка сохранения
7  SAVEPOINT sp_second;
8
9  -- rollback второй билет
10 INSERT INTO bookings.tickets (ticket_no, book_ref, passenger_id, passenger_name, contact_data)
11 VALUES
12 ('0005432000002', '000000', '1234 567890', 'Pipurkova Ivanesa', ('email': 'pipurkova@mail.ru'));
13
14 -- точка точка сохранения
15 SAVEPOINT sp_second;
16
17 -- rollback второй билет
18 INSERT INTO bookings.tickets (ticket_no, book_ref, passenger_id, passenger_name, contact_data)
19 VALUES
20 ('0005432000003', '000000', '1234 567891', 'Pipurkov Ivan Junior', ('email': 'pipurkov_jr@mail.ru'));
21
22 COMMIT;
```

```
23 SELECT * FROM bookings.tickets WHERE book_ref = '000000';
24 SELECT * FROM bookings.tickets WHERE book_ref = '000000';
```

| ticket_no | book_ref | passenger_id | passenger_name | contact_data |
|---------------|----------|--------------|----------------------|----------------------------------|
| 0005432000001 | 000000 | 1234 567890 | Pipurkov Ivan | ('email': 'pipurkov@mail.ru') |
| 0005432000002 | 000000 | 1234 567890 | Pipurkova Ivanesa | ('email': 'pipurkova@mail.ru') |
| 0005432000003 | 000000 | 1234 567891 | Pipurkov Ivan Junior | ('email': 'pipurkov_jr@mail.ru') |

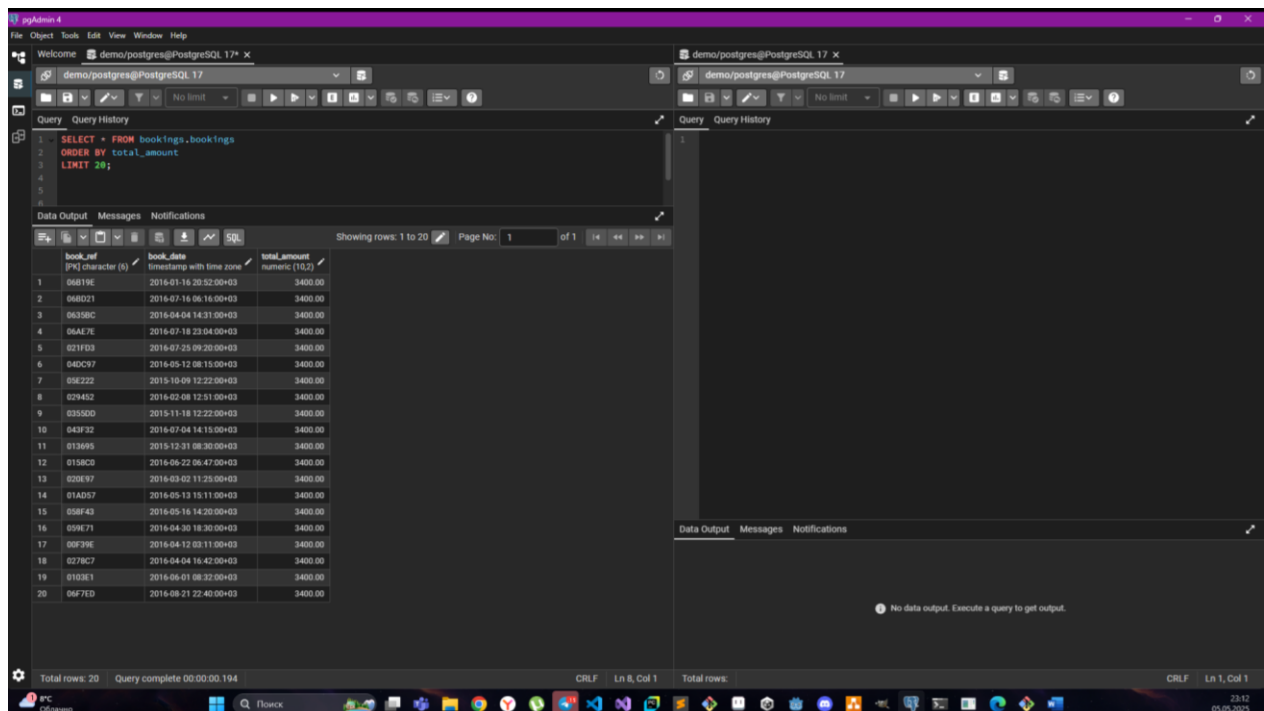
Уровень изоляции Read Committed

• Упражнение 6.4

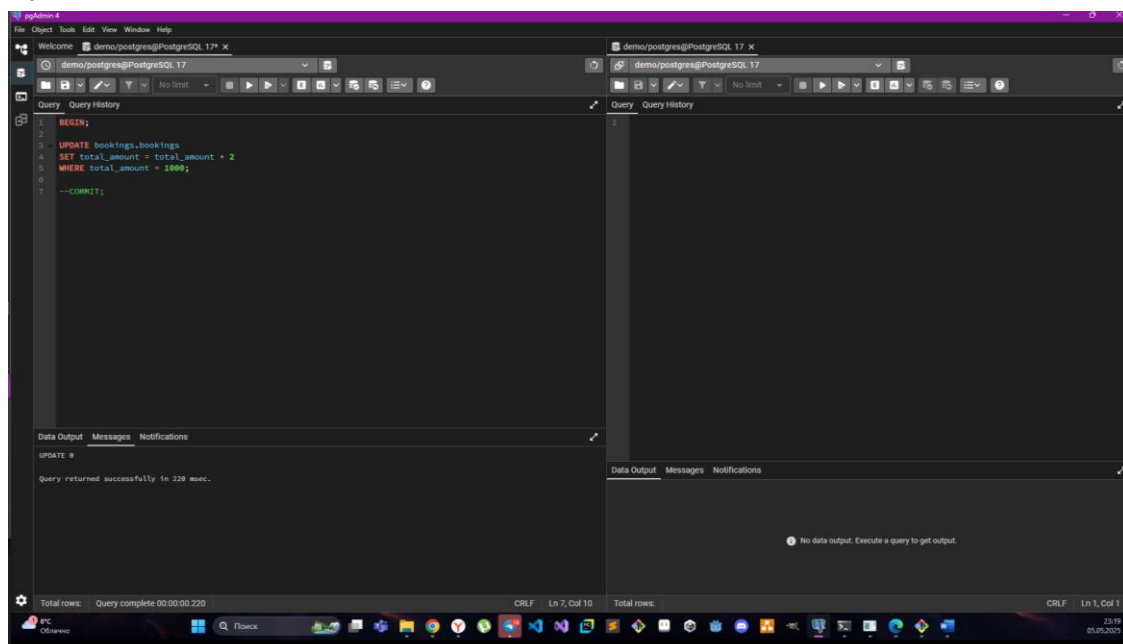
Перед началом выполнения задания проверьте, что в таблице bookings нет бронирований на сумму total_amount 1 000 рублей.

1. В первом сеансе начните транзакцию (командой BEGIN). Выполните обновление таблицы bookings: увеличьте total_amount в два раза в тех строках, где сумма равна 1 000 рублей.
2. Во втором сеансе (откройте новое окно psql) вставьте в таблицу bookings новое бронирование на 1 000 рублей и зафиксируйте транзакцию.
3. В первом сеансе повторите обновление таблицы bookings и зафиксируйте транзакцию.

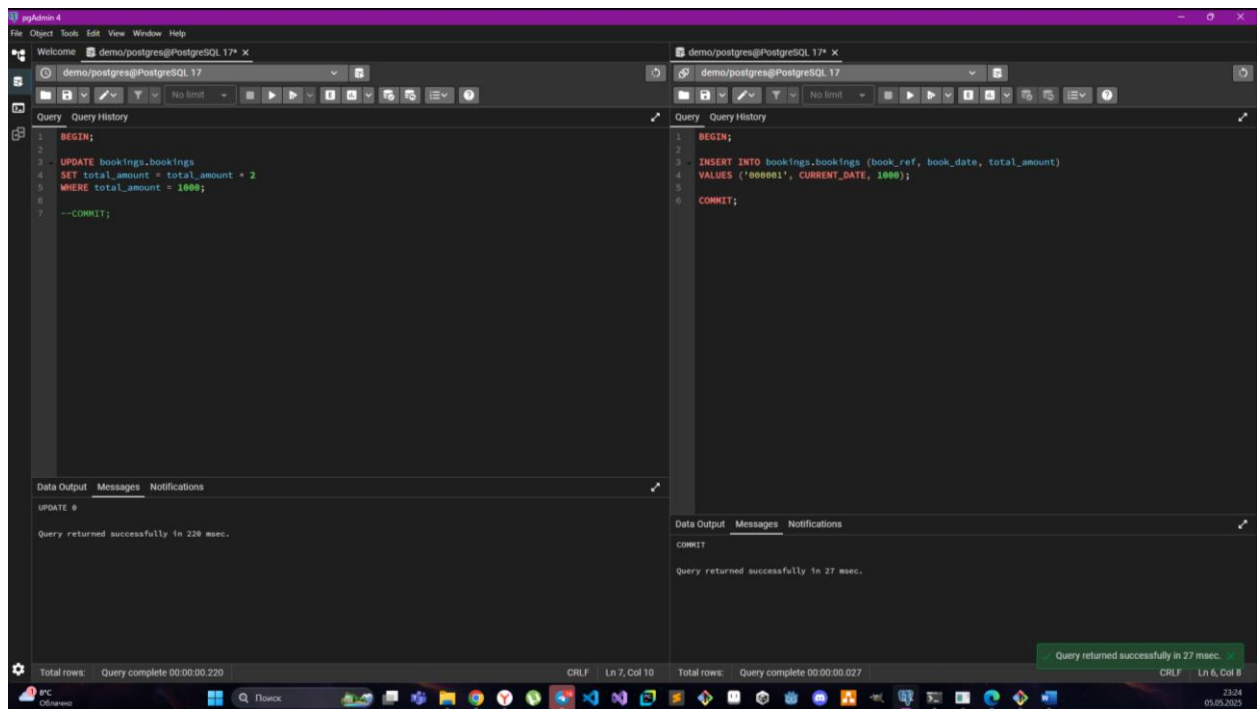
Осталась ли сумма добавленного бронирования равной 1 000 рублей? Почему это не так?



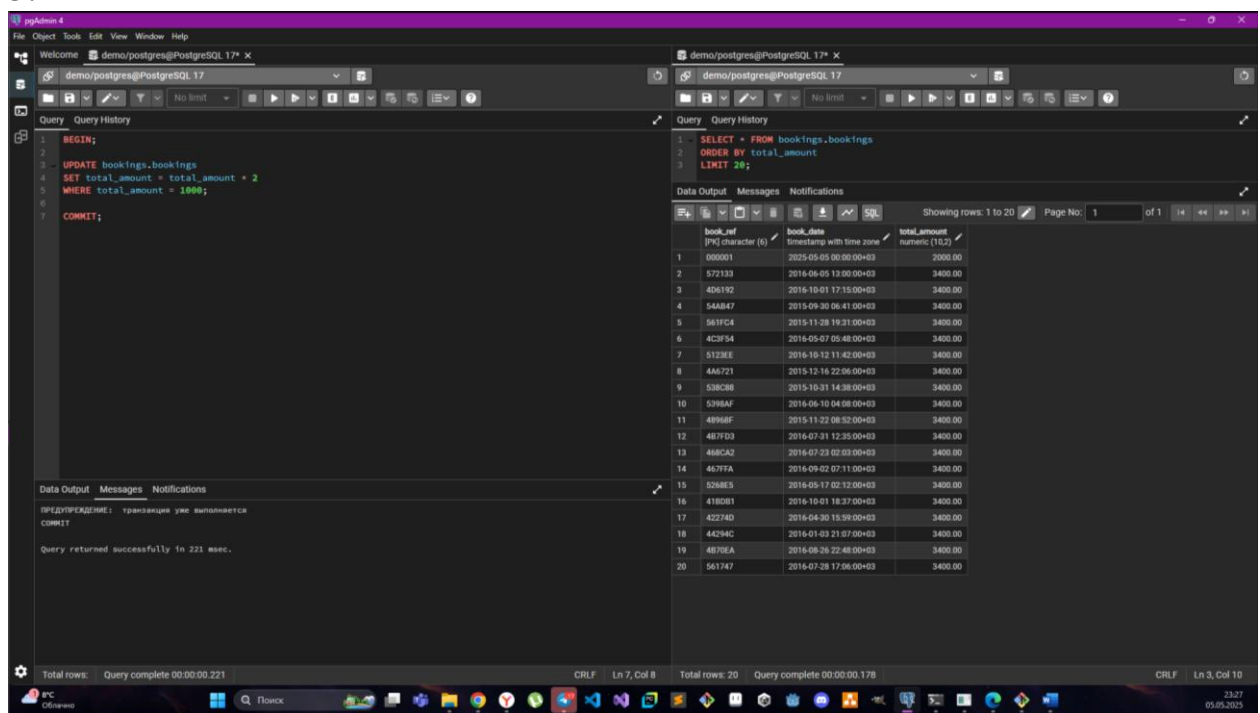
1.



2.



3.

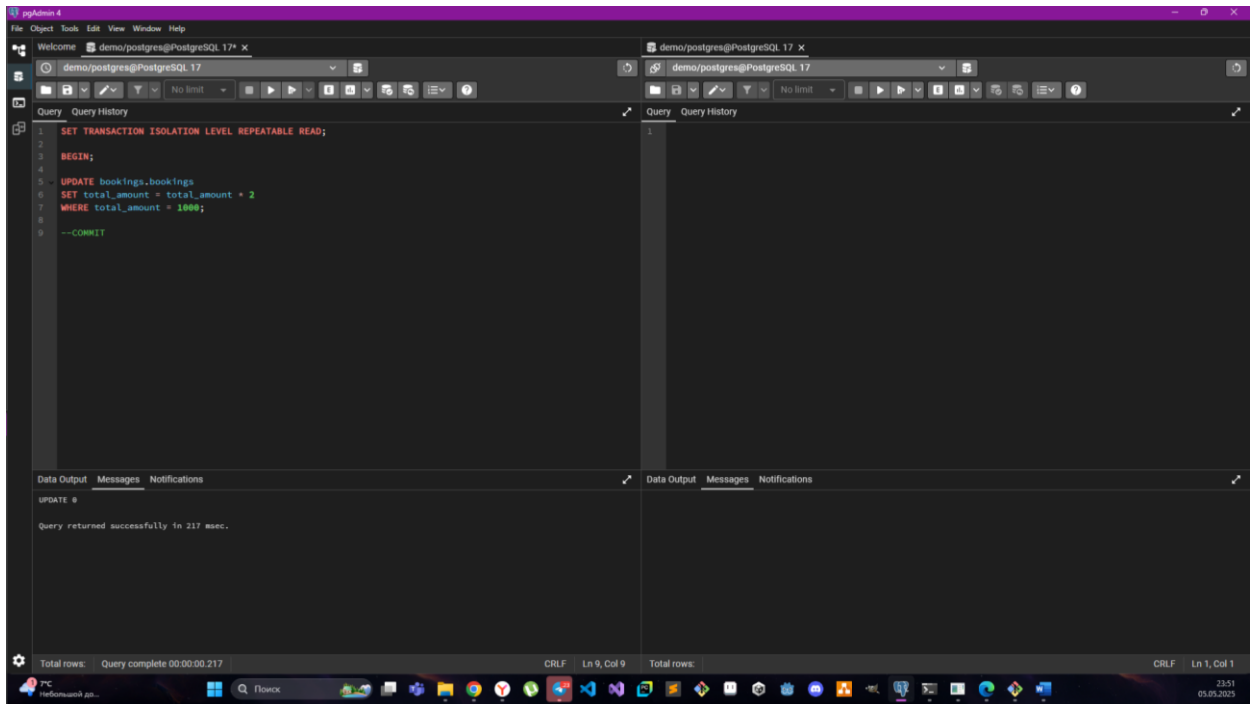


Все происходит из-за использования PostgreSQL по умолчанию уровня изоляции Read Committed, то есть результаты других транзакций становятся доступными после их фиксации. Поэтому сначала мы пытаемся обновить значения, потом фиксируем добавление новой строки, а после повторно проверяем и обновляем значения.

Уровень изоляции Repeatable Read

• Упражнение 6.5

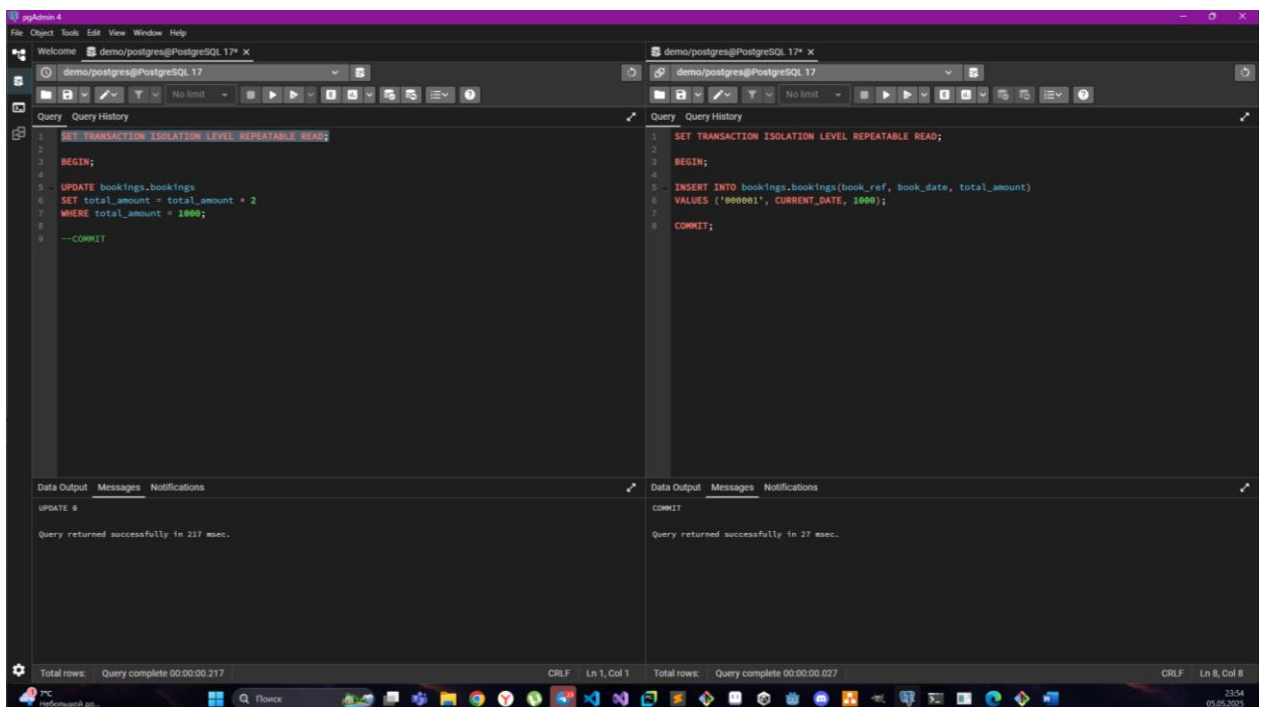
Повторите предыдущее упражнение, но начните транзакцию в первом сеансе с уровнем изоляции транзакций Repeatable Read. Объясните различие полученных результатов.



The screenshot shows the pgAdmin 4 interface with two sessions connected to a PostgreSQL 17 database. The left session, titled 'demo/postgres@PostgreSQL 17', contains a transaction with the following SQL statements:

```
1 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
2 BEGIN;
3
4
5 UPDATE bookings.bookings
6 SET total_amount = total_amount + 2
7 WHERE total_amount = 1000;
8
9 --COMMIT
```

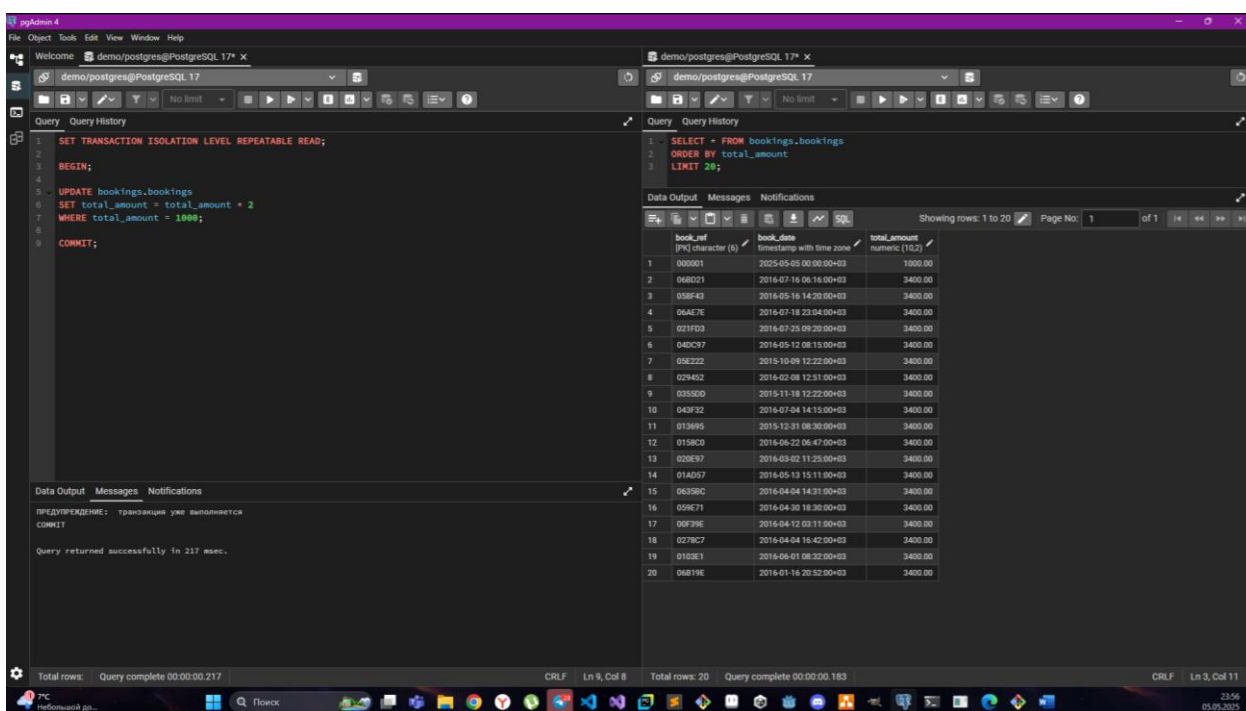
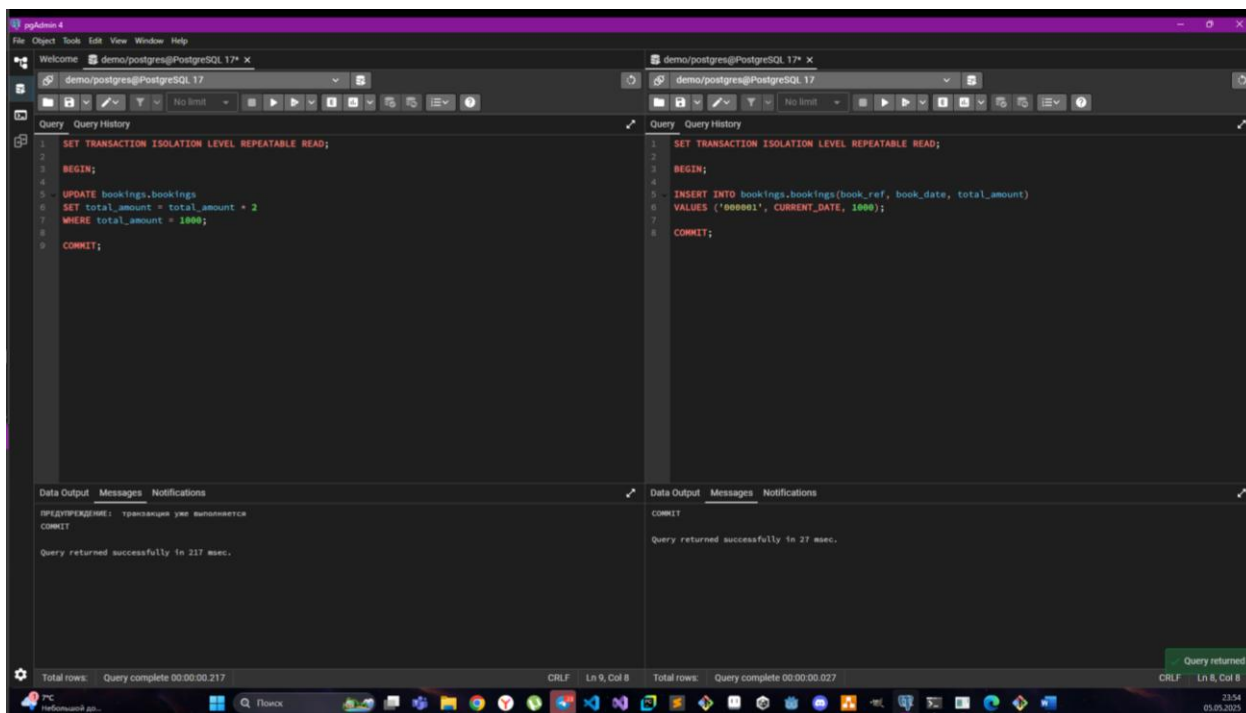
The 'Data Output' tab for this session shows the message: 'UPDATE 0' and 'Query returned successfully in 217 msec.' The status bar at the bottom indicates 'Total rows: Query complete 00:00:00.217'.



The screenshot shows the same pgAdmin 4 interface. The right session, also titled 'demo/postgres@PostgreSQL 17', contains a transaction with the following SQL statements:

```
1 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
2 BEGIN;
3
4 INSERT INTO bookings.bookings(book_ref, book_date, total_amount)
5 VALUES ('0000001', CURRENT_DATE, 1000);
6
7 COMMIT;
```

The 'Data Output' tab for this session shows the message: 'COMMIT' and 'Query returned successfully in 27 msec.' The status bar at the bottom indicates 'Total rows: Query complete 00:00:00.027'.



Так как в этот раз использовался уровень изоляции Repeatable Read, который позволяет видеть только те данные, которые были доступны на момент начала выполнения транзакции, то получается, что на момент начала первой транзакции строки со значение total_amount = 1000 не существовало, поэтому значение и не обновилось.

• Упражнение 6.6

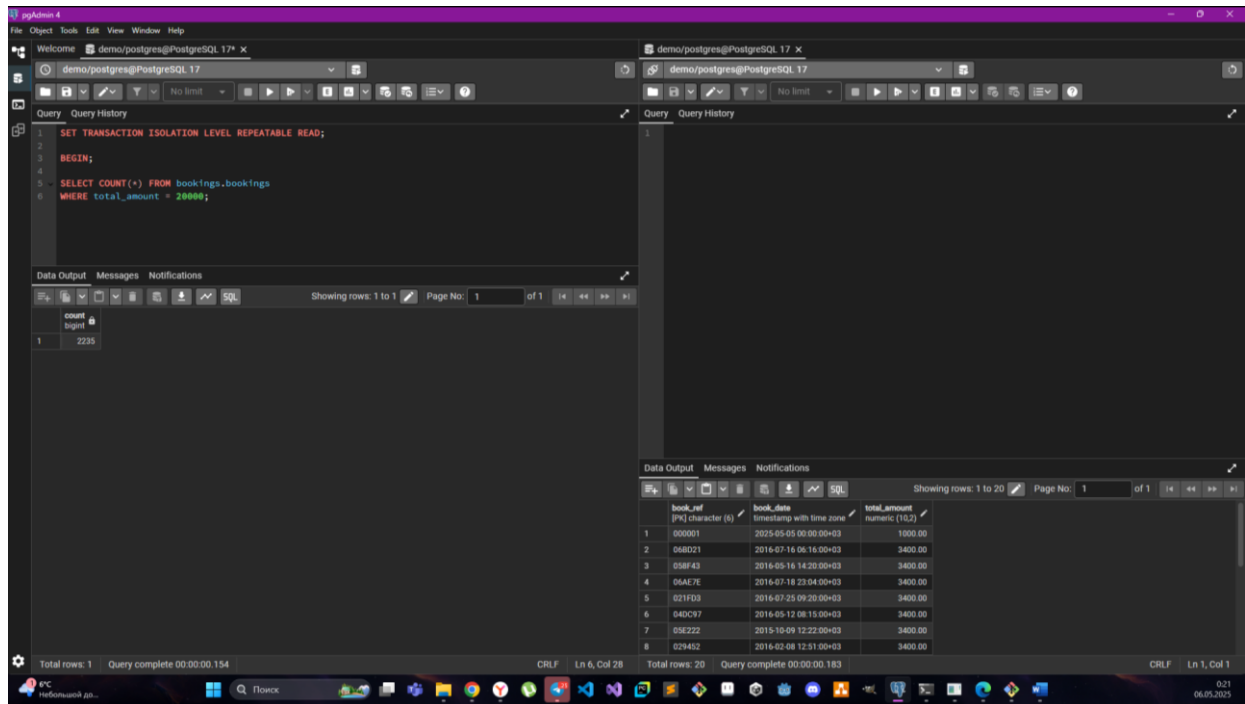
Выполните указанные действия в двух сеансах:

1. В первом сеансе начните новую транзакцию с уровнем изоляции Repeatable Read. Вычислите количество бронирований с суммой 20 000 рублей.
2. Во втором сеансе начните новую транзакцию с уровнем изоляции Repeatable Read. Вычислите количество бронирований с суммой 30 000 рублей.
3. В первом сеансе добавьте новое бронирование на 30 000 рублей и снова вычислите количество бронирований с суммой 20 000 рублей.

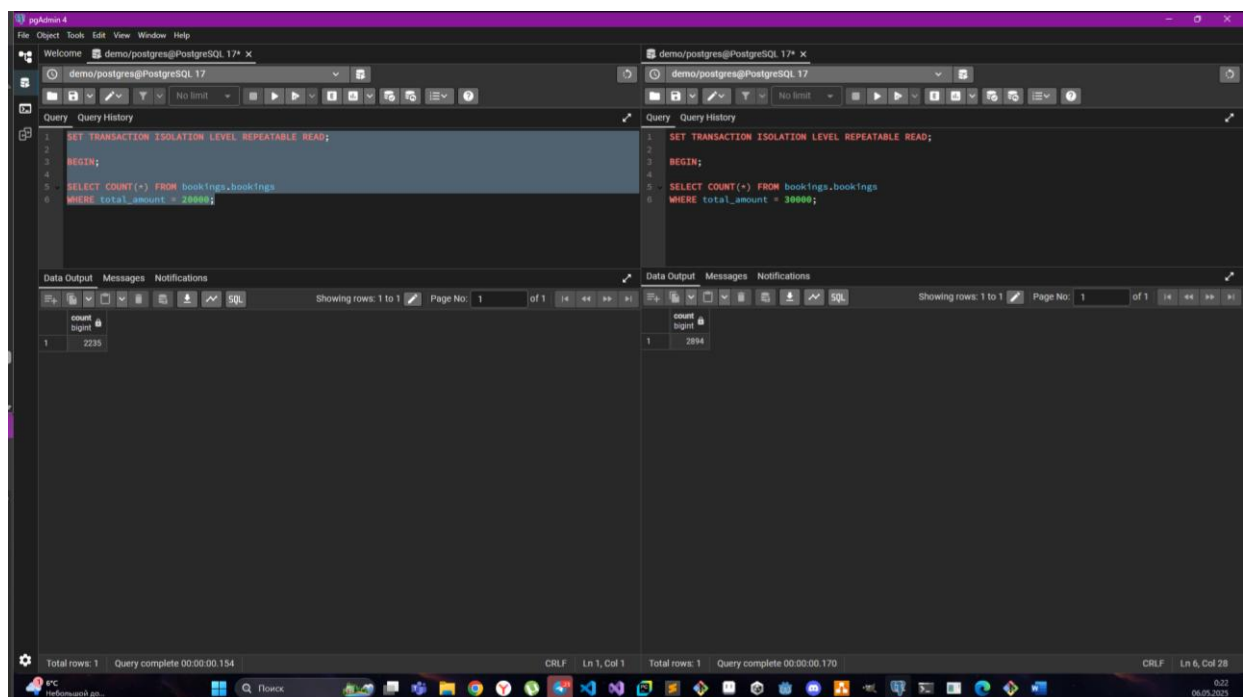
4. Во втором сеансе добавьте новое бронирование на 20 000 рублей и снова вычислите количество бронирований с суммой 30 000 рублей.
5. Зафиксируйте транзакции в обоих сеансах.

Соответствует ли результат ожиданиями? Можно ли сериализовать эти транзакции (иными словами, можно ли представить такой порядок последовательного выполнения этих транзакций, при котором результат совпадет с тем, что получился при параллельном выполнении)?

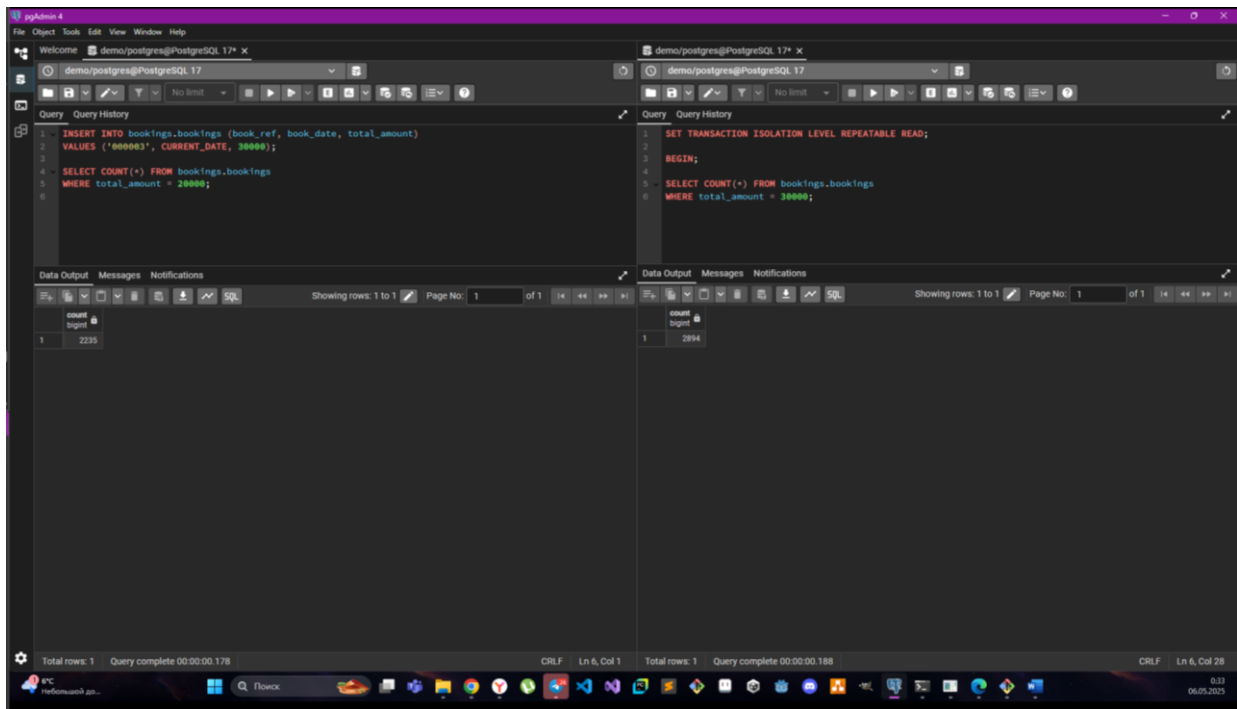
1.



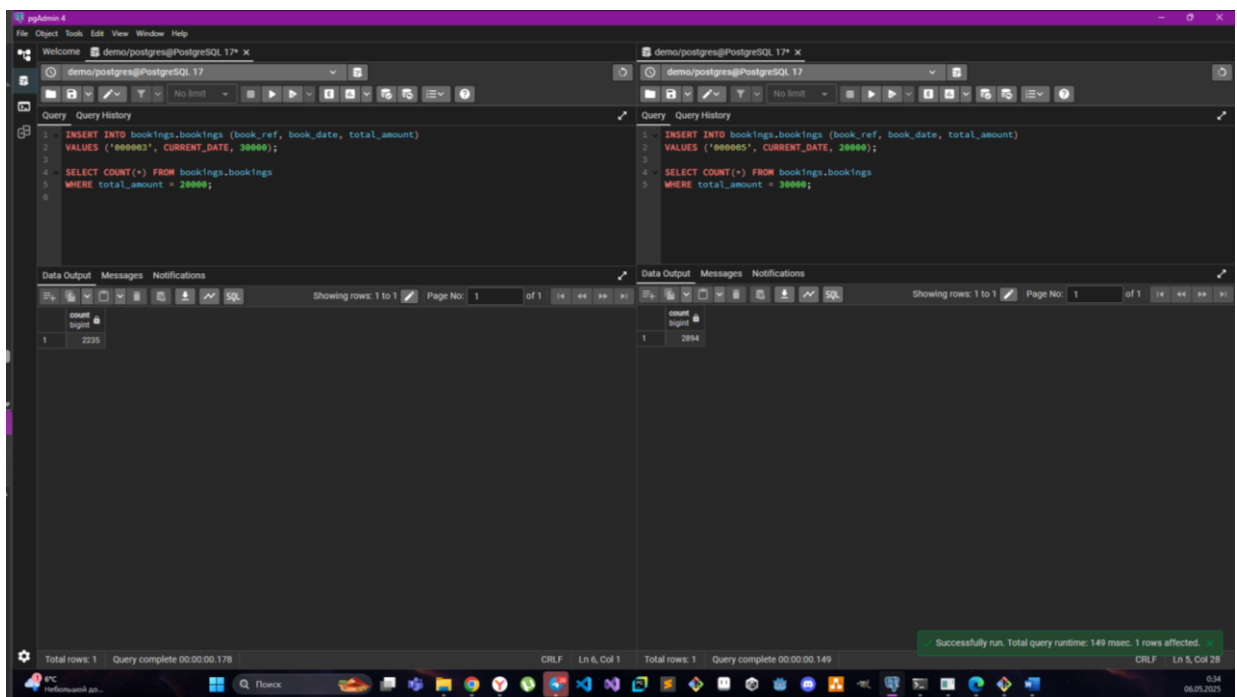
2.



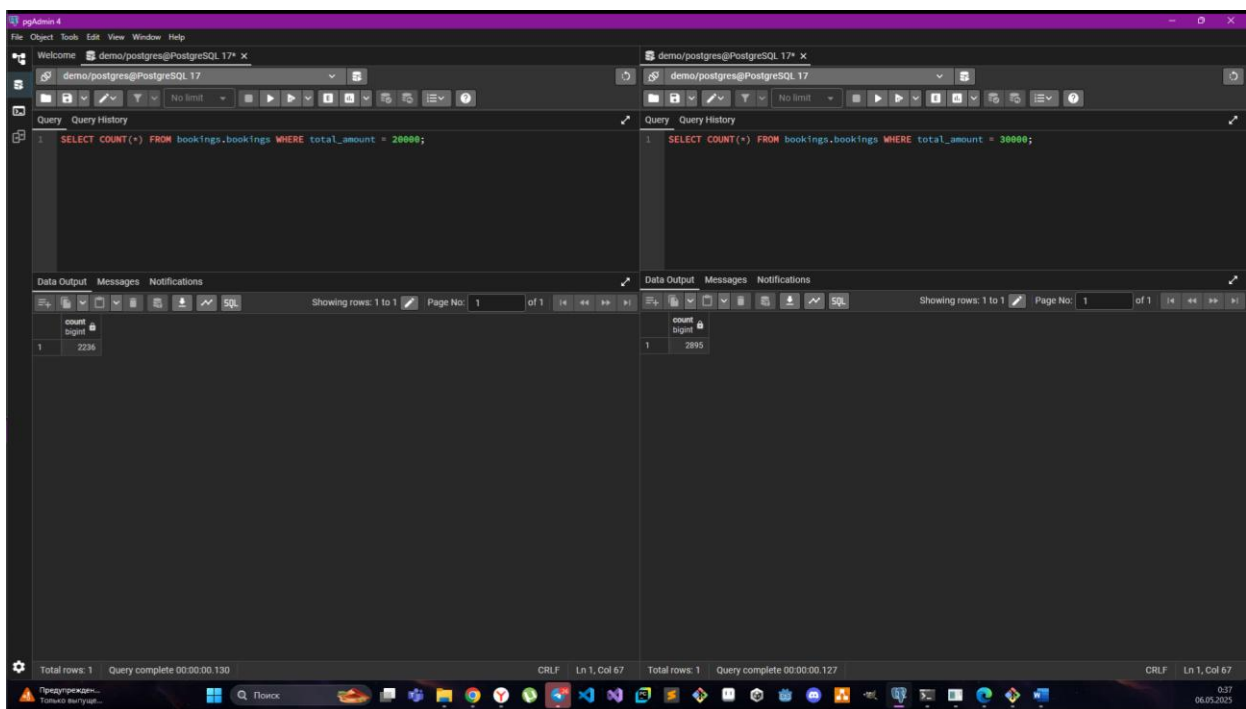
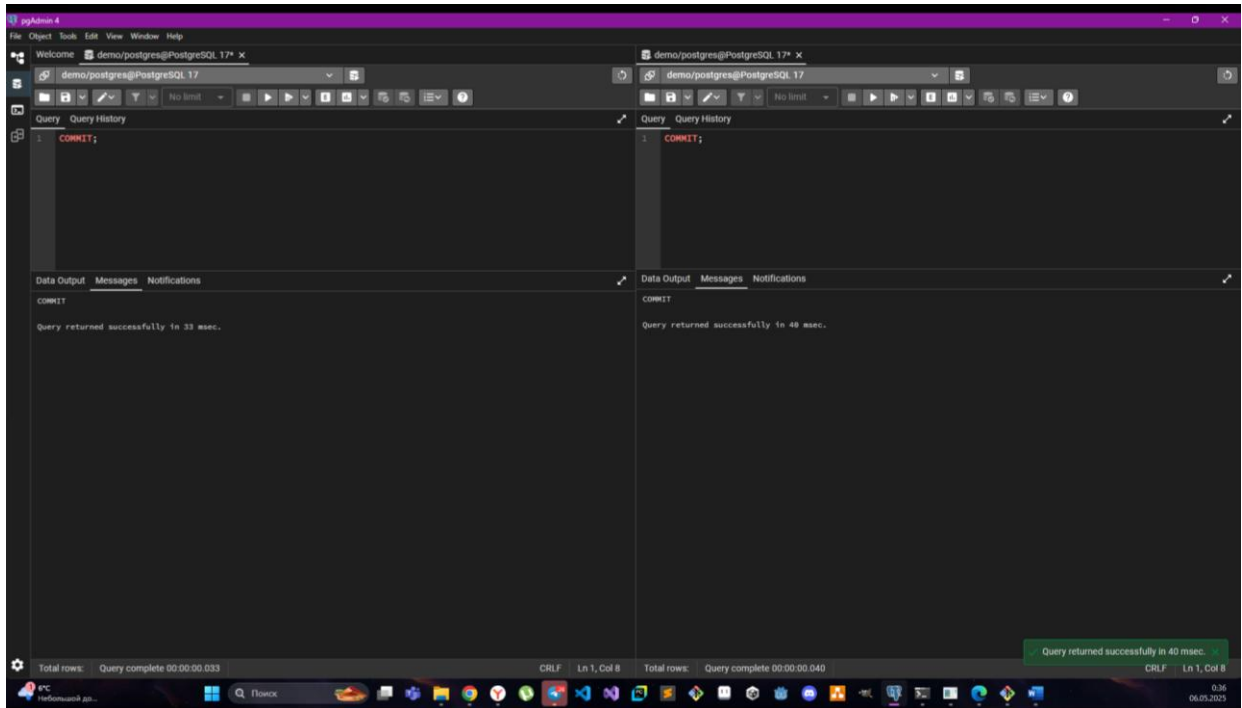
3.



4.



5.

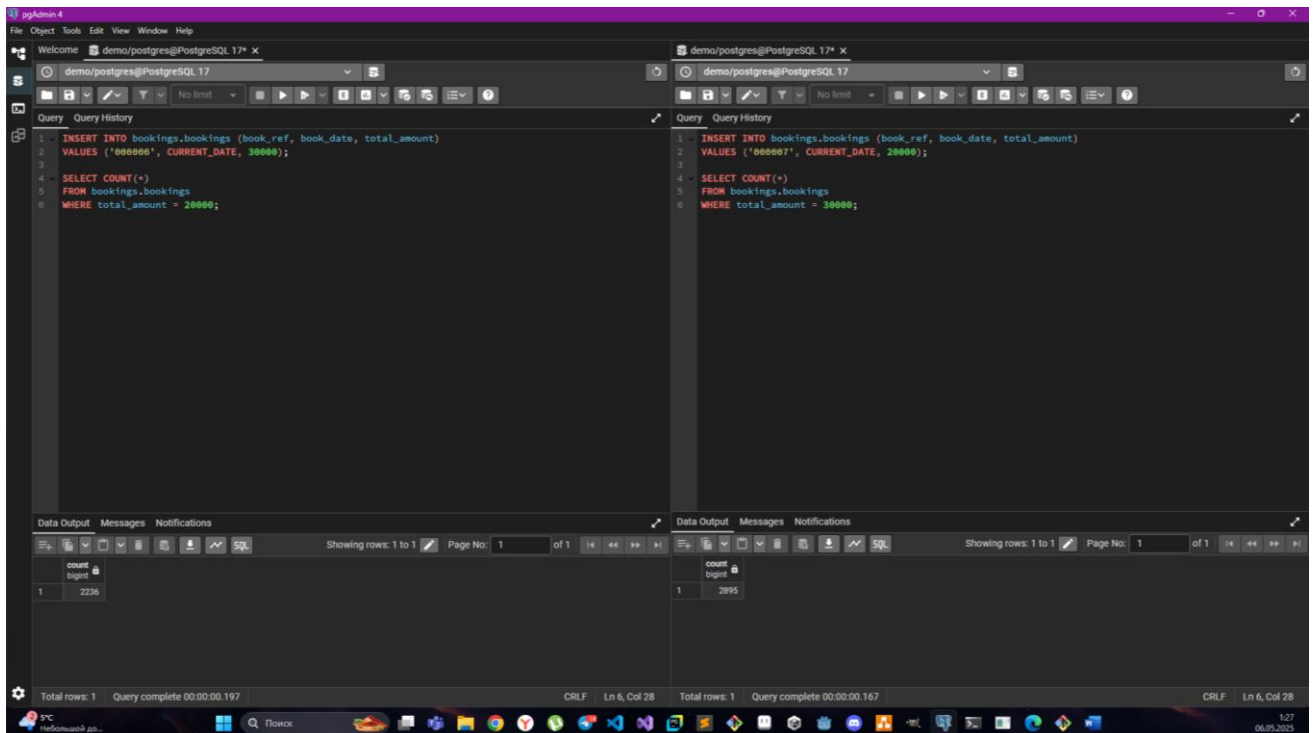
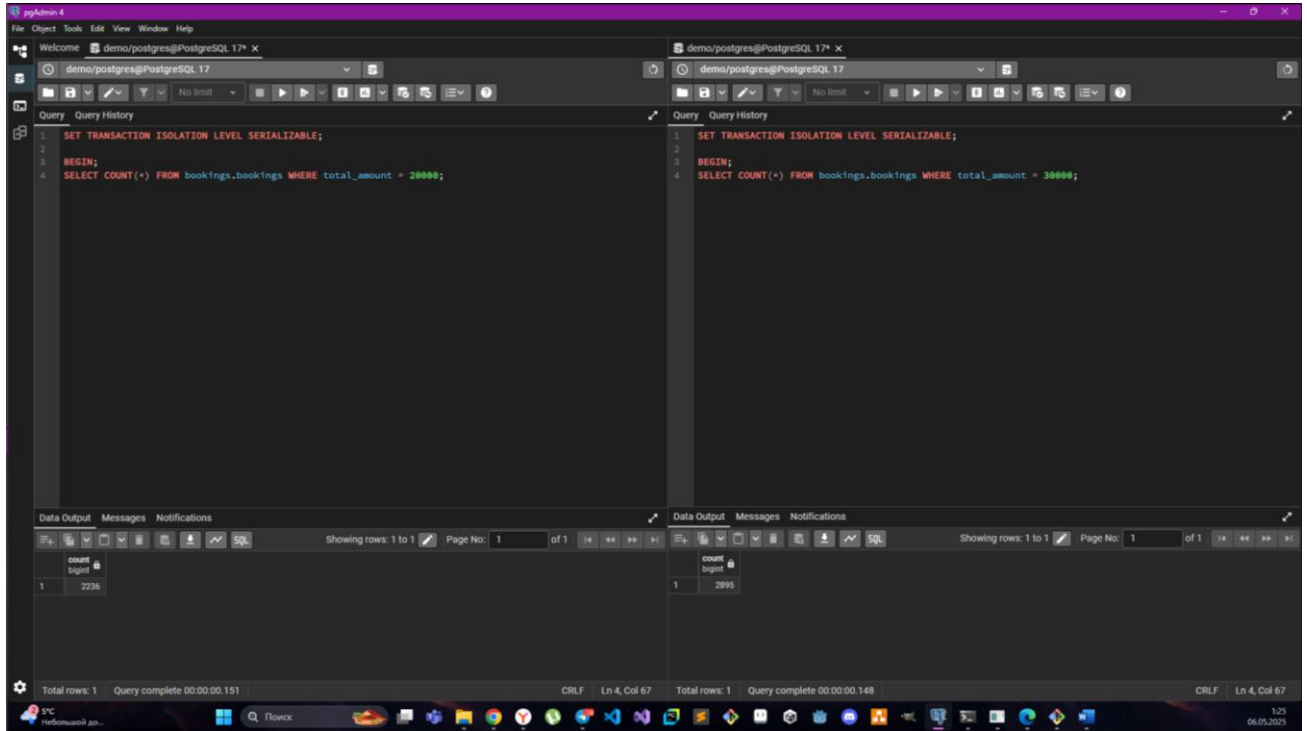


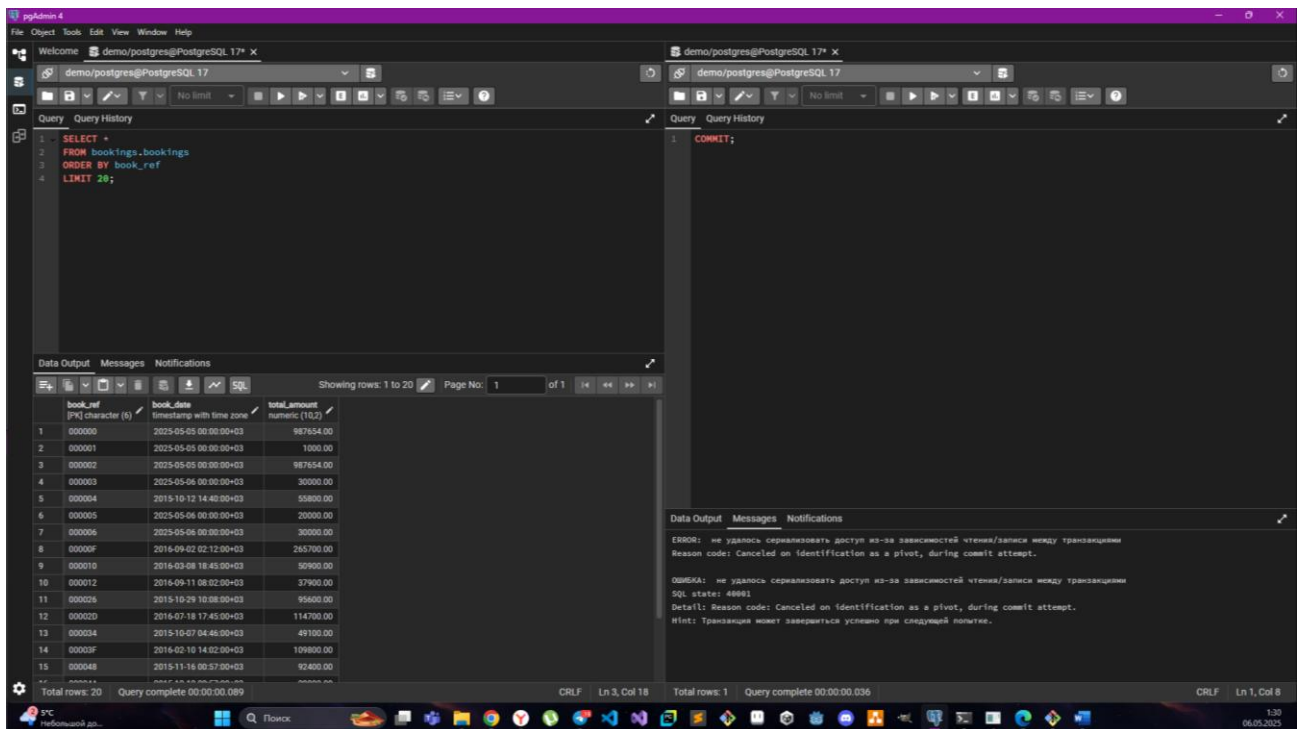
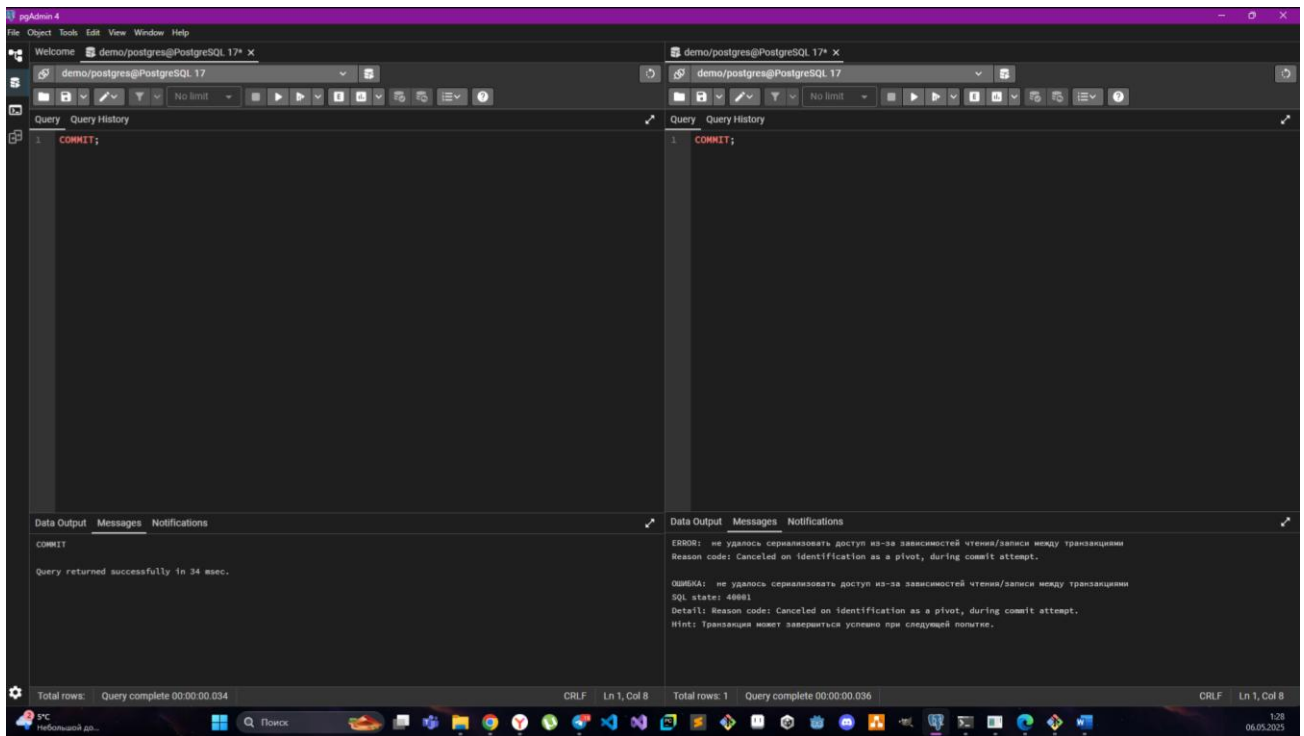
Результаты выполнения двух транзакций соответствуют ожиданиям, а именно тому, что запросы выполняются параллельно и не влияют на работу друг друга. Serializable эмулирует последовательное выполнение транзакций. В данном случае не получится представить эти транзакции последовательным образом и получить схожие с параллельным выполнением результат, так как при попытке повторного чтения в одном из сеансов потенциально возникнет конфликт, в чем мы убедимся в следующем упражнении.

Уровень изоляции Serializable

• Упражнение 6.7

Повторите предыдущее упражнение, но транзакции в обоих сеансах начните с уровнем изоляции Serializable. Если вы правильно ответили на его последний вопрос, вы поймете, почему теперь эти действия приводят к ошибке. Если же результат этого упражнения стал для вас неожиданностью, четко сформулируйте различие уровней Repeatable Read и Serializable.





1. Гарантии целостности данных

Repeatable Read:

- Гарантирует, что в пределах одной транзакции повторные чтения одних и тех же данных вернут одинаковые результаты
- Предотвращает "грязное чтение" и "неповторяющееся чтение"
- Не предотвращает фантомное чтение (появление новых строк, удовлетворяющих условиям запроса)

Serializable:

- Гарантирует, что результат параллельного выполнения транзакций будет эквивалентен их последовательному выполнению
- Предотвращает все аномалии: "грязное чтение", "неповторяющееся чтение" и "фантомное чтение"
- Обеспечивает самый строгий уровень изоляции

2. Механизм работы

Repeatable Read:

- Использует снимок данных на момент начала первого запроса в транзакции
- Блокирует только фактически читаемые строки

Serializable:

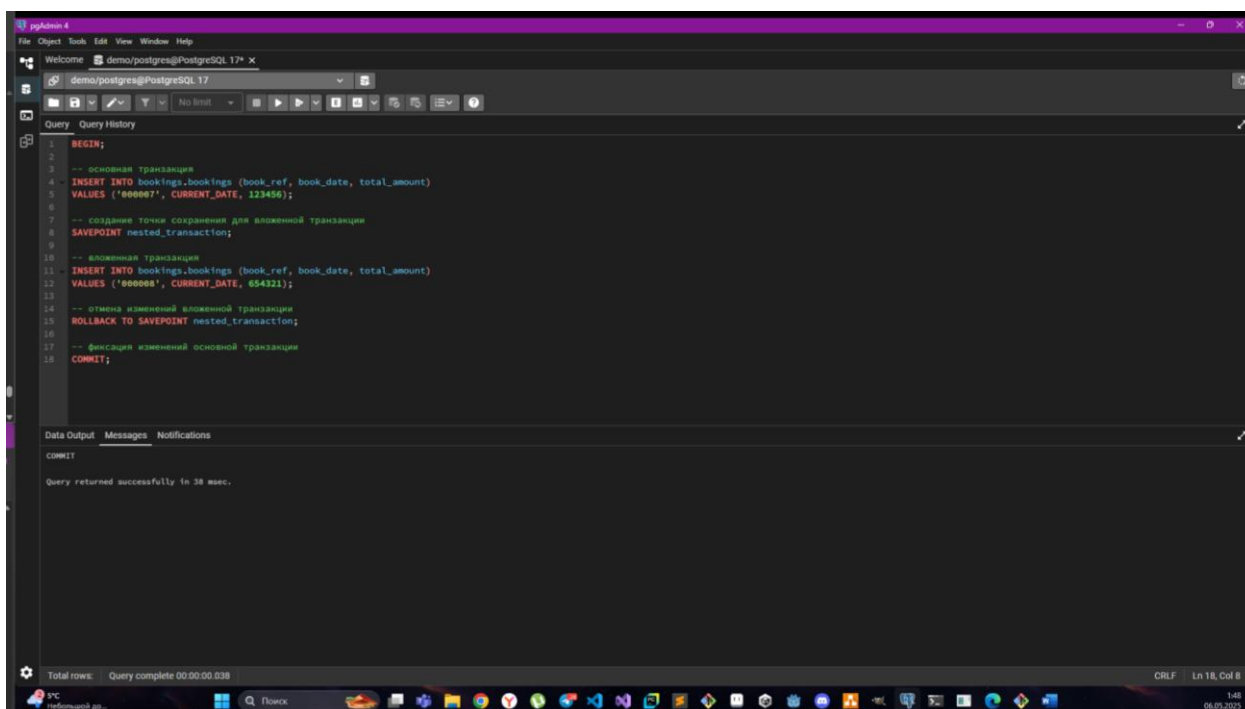
- Использует более сложные механизмы (часто - предикатные блокировки)
- Может блокировать не только существующие данные, но и потенциальные диапазоны значений

Вложенные транзакции

• Упражнение 6.8

Некоторые СУБД (но не PostgreSQL) позволяют использовать вложенные транзакции. Если начать вторую транзакцию, не завершая уже открытую первую, то вторая транзакция будет считаться вложенной: ее результат фиксируется только в том случае, если фиксируется первая транзакция, но при этом ее результаты можно отменить независимо от первой транзакции.

Реализуйте такое поведение для PostgreSQL, т. е. предложите, какие команды следует выполнять для открытия вложенной транзакции, для ее отмены и для фиксации. Подсказка: используйте оператор SAVEPOINT.



```
1 BEGIN;
2
3 -- основная транзакция
4 INSERT INTO bookings.bookings (book_ref, book_date, total_amount)
5 VALUES ('600007', CURRENT_DATE, 123456);
6
7 -- создание точки сохранения для вложенной транзакции
8 SAVEPOINT nested_transaction;
9
10 -- вложенная транзакция
11 INSERT INTO bookings.bookings (book_ref, book_date, total_amount)
12 VALUES ('600008', CURRENT_DATE, 654321);
13
14 -- отмена изменений вложенной транзакции
15 ROLLBACK TO SAVEPOINT nested_transaction;
16
17 -- фиксация изменений основной транзакции
18 COMMIT;
```

Data Output Messages Notifications

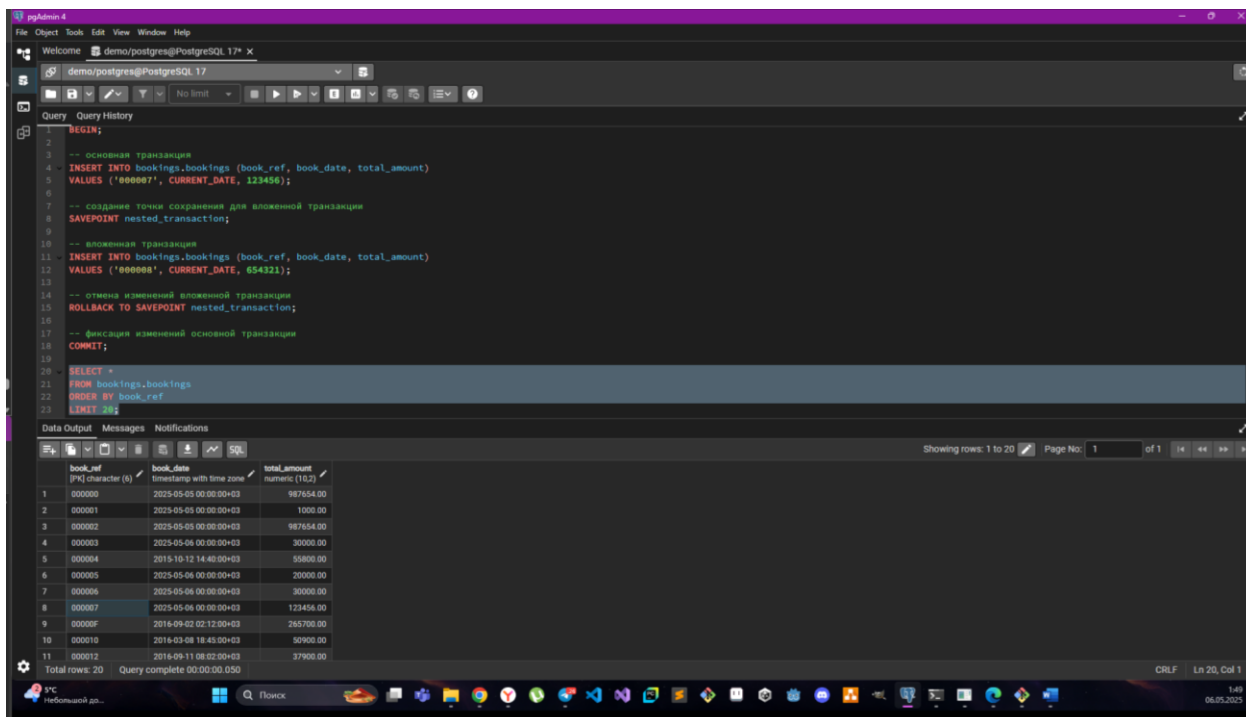
COMMIT

Query returned successfully in 38 msec.

Total rows: Query complete 00:00:00.038

Ln 18, Col 8

06.01.2025



Так же при необходимости можно использовать оператор `RELEASE SAVEPOINT`, для отмены точек сохранения и подтверждения вложенной транзакции. Это особенно актуально при особо сложных транзакциях, где может понадобиться отменить только часть изменений.

Ответы на контрольные вопросы

1. Что такое транзакция? Приведите примеры транзакции.

Транзакция — это последовательность операций, выполняемых над базой данных, которая рассматривается как единое целое. Транзакция должна быть завершена полностью (коммит) или не выполнена вовсе (откат). Это гарантирует целостность данных.

Примеры транзакции:

Перевод денег с одного банковского счета на другой. Операции включают вычитание суммы с одного счета и добавление ее на другой. Если одна из операций не удастся, обе операции должны быть отменены.

Добавление нового заказа в систему, где необходимо обновить запасы товаров и создать запись о заказе. Если обновление запасов не удалось, запись о заказе не должна сохраняться.

2. Что такое атомарность?

Атомарность — это свойство транзакции, которое гарантирует, что все операции внутри транзакции выполняются полностью или не выполняются вовсе. Это означает, что транзакция является неделимой: если одна часть транзакции не может быть выполнена, вся транзакция откатывается.

3. Назовите основные требования к транзакциям? Приведите примеры.

Основные требования к транзакциям известны как ACID-принципы:

Атомарность: все операции в транзакции выполняются полностью или не выполняются вовсе. Пример: если перевод денег не удался, ни одна из операций не должна быть зафиксирована.

Согласованность: транзакция переводит базу данных из одного согласованного состояния в другое. Пример: если добавляется новый заказ, запасы товаров должны быть обновлены, чтобы отразить это изменение.

Изолированность: результаты транзакции не видны другим транзакциям до ее завершения. Пример: если одна транзакция обновляет данные, другая транзакция не должна видеть промежуточные результаты.

Долговечность: после фиксации транзакции изменения сохраняются в базе данных даже в случае сбоя системы. Пример: после успешного выполнения транзакции по переводу денег, изменения должны остаться даже при перезагрузке сервера.

4. Что такое аномалии конкурентного выполнения?

Аномалии конкурентного выполнения — это проблемы, возникающие при одновременном выполнении нескольких транзакций, которые могут привести к неконсистентному состоянию базы данных. Эти аномалии могут возникать из-за недостаточной изоляции транзакций.

5. Какая аномалия называется потерянным обновлением?

Потерянное обновление — это аномалия, которая происходит, когда одно обновление данных перезаписывает изменения, сделанные другой транзакцией. Например, если два пользователя одновременно обновляют одну и ту же запись, и одно из обновлений теряется, это приводит к некорректному состоянию данных.

6. Перечислите несколько видов аномалий конкурентного выполнения.

Потерянное обновление.

Фантомные чтения - когда одна транзакция видит разные данные при повторных запросах из-за изменений, сделанных другой транзакцией.

Неправильное чтение - когда транзакция читает данные, которые были изменены другой транзакцией, но еще не зафиксированы.

7. Что такое протокол двухфазного блокирования?

Протокол двухфазного блокирования (2PL) — это метод управления конкурентным доступом к данным, который гарантирует изолированность транзакций.

Он состоит из двух фаз:

Фаза роста: транзакция может запрашивать блокировки, но не может их освобождать.

Фаза сжатия: транзакция может освобождать блокировки, но не может запрашивать новые.

Этот протокол помогает избежать аномалий конкурентного выполнения, но может привести к взаимным блокировкам.

8. Какие уровни изоляции предусмотрены стандартом?

Read Uncommitted: позволяет читать незафиксированные данные, что может привести к аномалиям.

Read Committed: позволяет читать только зафиксированные данные, предотвращая "потерянные обновления".

Repeatable Read: гарантирует, что данные, прочитанные в транзакции, останутся неизменными до ее завершения.

Serializable: самый строгий уровень, который предотвращает все аномалии конкурентного выполнения, обеспечивая полную изоляцию транзакций.