Национальный исследовательский ядерный университет «МИФИ»

(Московский Инженерно-Физический Институт)

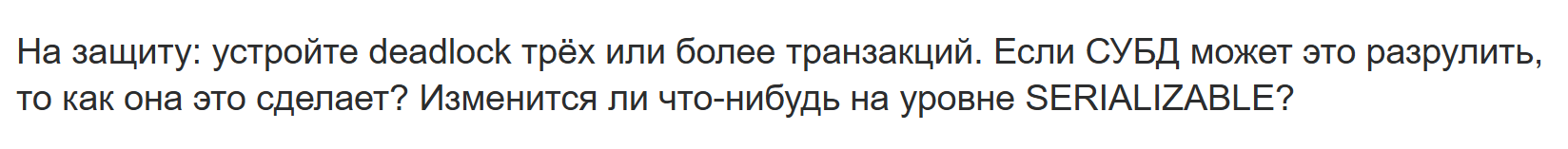
Кафедра №42 «Криптология и кибербезопасность»

**На защиту лабораторной работы №2-2**

**«**Транзакции. Изоляция транзакций**»**

Тимин Александр Б21-515 (2024г.)

**1 Deadlock**



Дедлок — это ситуация, когда две или более транзакции блокируют друг друга, ожидая доступа к ресурсам, занятым другой транзакцией. PostgreSQL обнаруживает дедлоки автоматически и завершает одну из транзакций с ошибкой, чтобы другие могли продолжить выполнение.

**1.1 Deadlock трёх транзакций**

Для эксперимента используем базу данных, в которой создадим таблицу, заполнив ее данными:

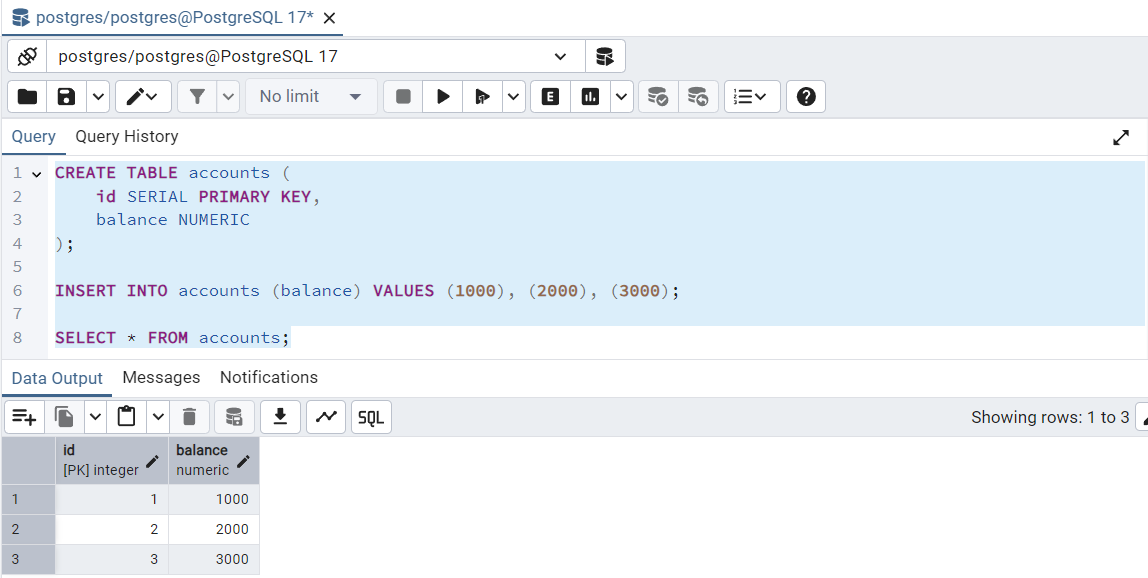
CREATE TABLE accounts (

id SERIAL PRIMARY KEY,

balance NUMERIC

);

INSERT INTO accounts (balance) VALUES (1000), (2000), (3000);



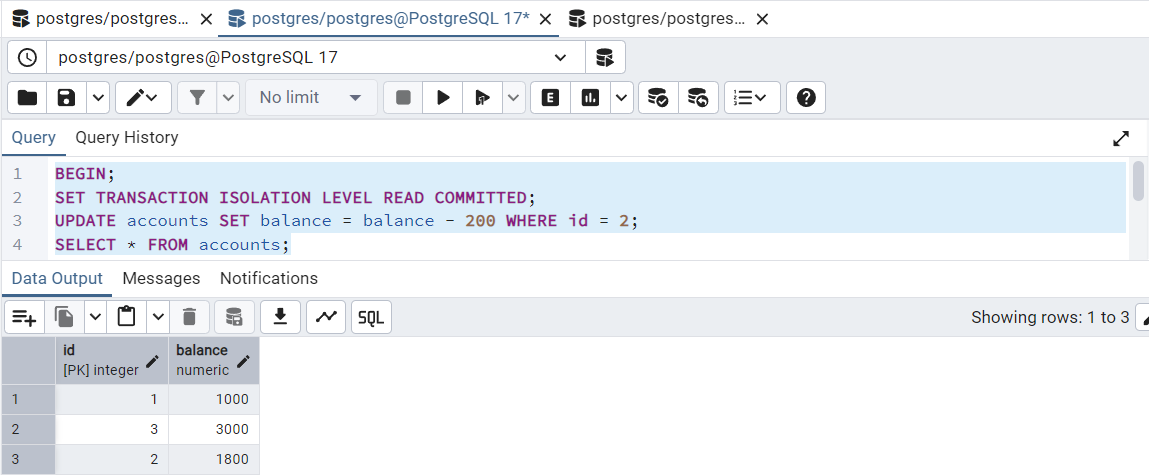
Для одновременного выполнения трёх транзакций было открыто 3 сессии в pgAdmin, в каждой была запущена транзакция и установлен уровень изоляции транзакций. После были обновлены значения строк, соответствующих условному номеру сессии (код из третьей сессии).

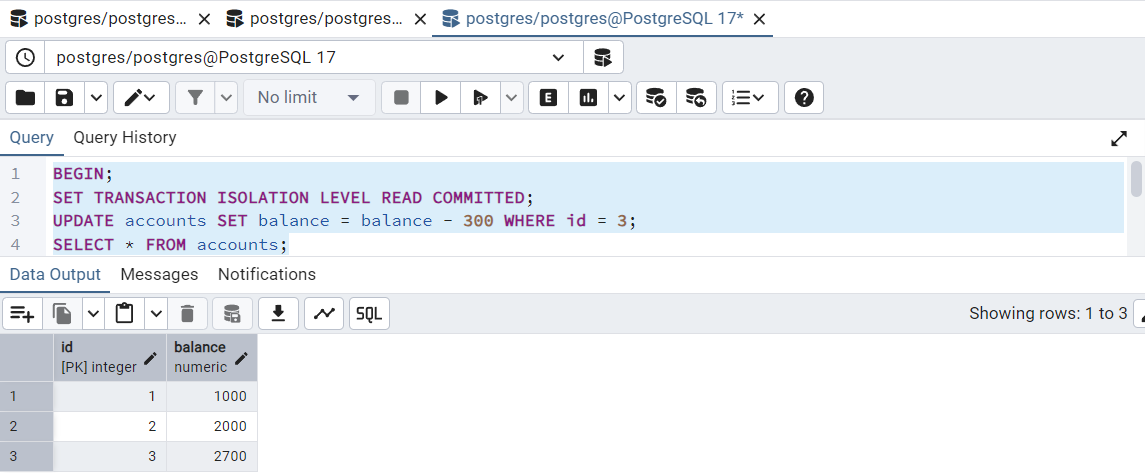
BEGIN;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

UPDATE accounts SET balance = balance - 300 WHERE id = 3;

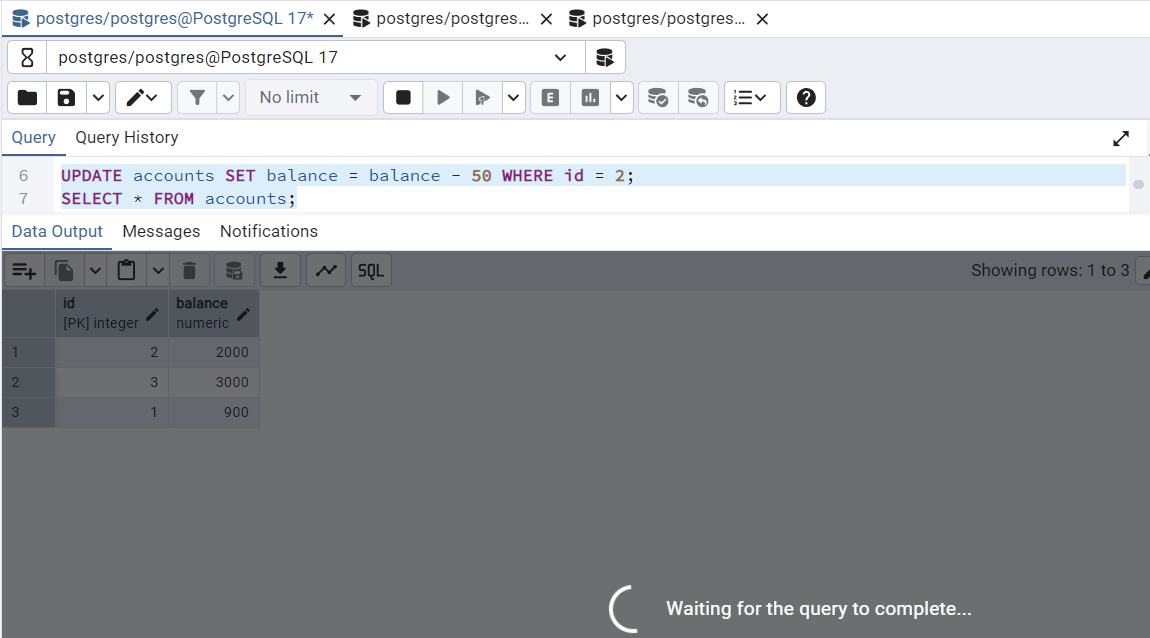


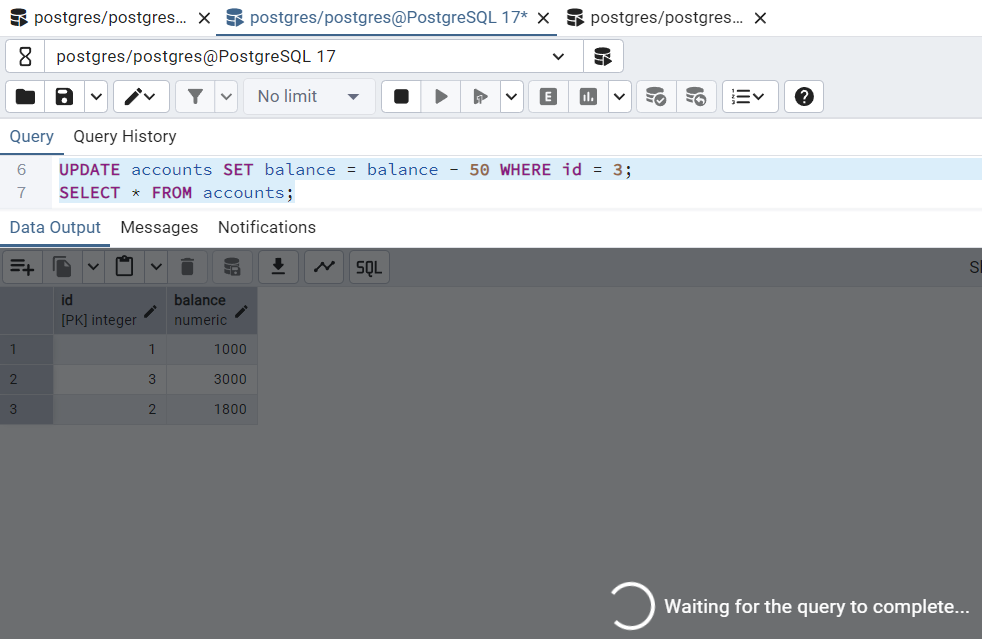




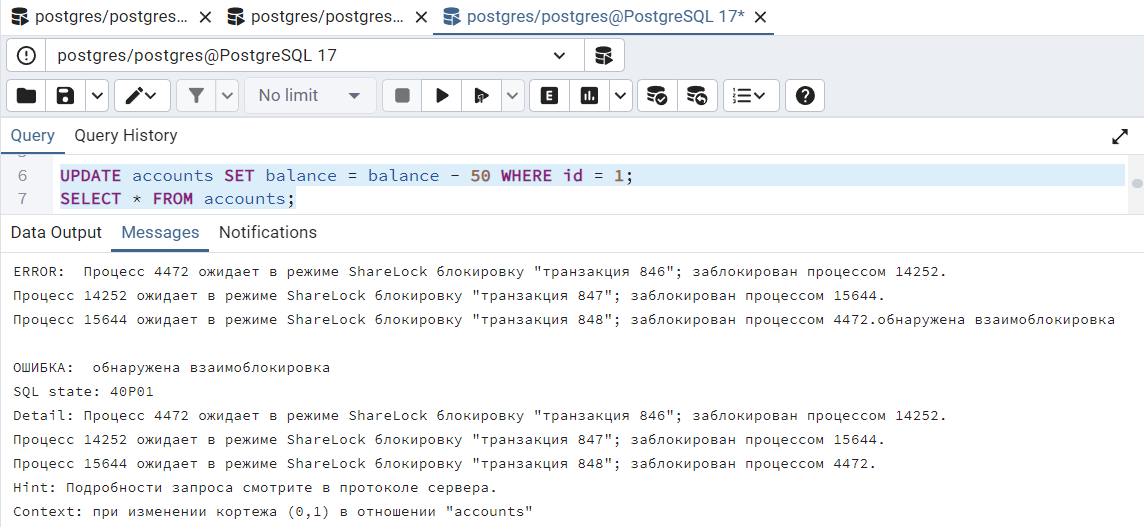
Далее были созданы зависимости транзакций: 1 от 2, 2 от 3, 3 от 1 (код из третьей сессии).

UPDATE accounts SET balance = balance - 50 WHERE id = 1;





Как видно из скриншотов, первые две транзакции подвисли, ожидая завершения транзакций, от которых они зависят.



После же выполнения аналогичного запроса в третьей транзакции, она на полсекунды подвисла (не успел заскринить, чесна), после чего транзакция завершилась с ошибкой:

ERROR: Процесс 4472 ожидает в режиме ShareLock блокировку "транзакция 846"; заблокирован процессом 14252.

Процесс 14252 ожидает в режиме ShareLock блокировку "транзакция 847"; заблокирован процессом 15644.

Процесс 15644 ожидает в режиме ShareLock блокировку "транзакция 848"; заблокирован процессом 4472.обнаружена взаимоблокировка

ОШИБКА: обнаружена взаимоблокировка

SQL state: 40P01

Detail: Процесс 4472 ожидает в режиме ShareLock блокировку "транзакция 846"; заблокирован процессом 14252.

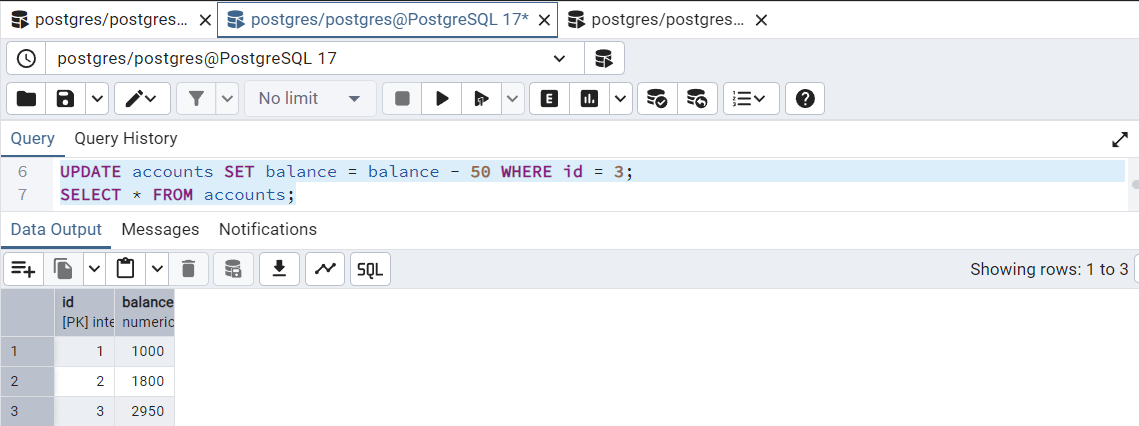
Процесс 14252 ожидает в режиме ShareLock блокировку "транзакция 847"; заблокирован процессом 15644.

Процесс 15644 ожидает в режиме ShareLock блокировку "транзакция 848"; заблокирован процессом 4472.

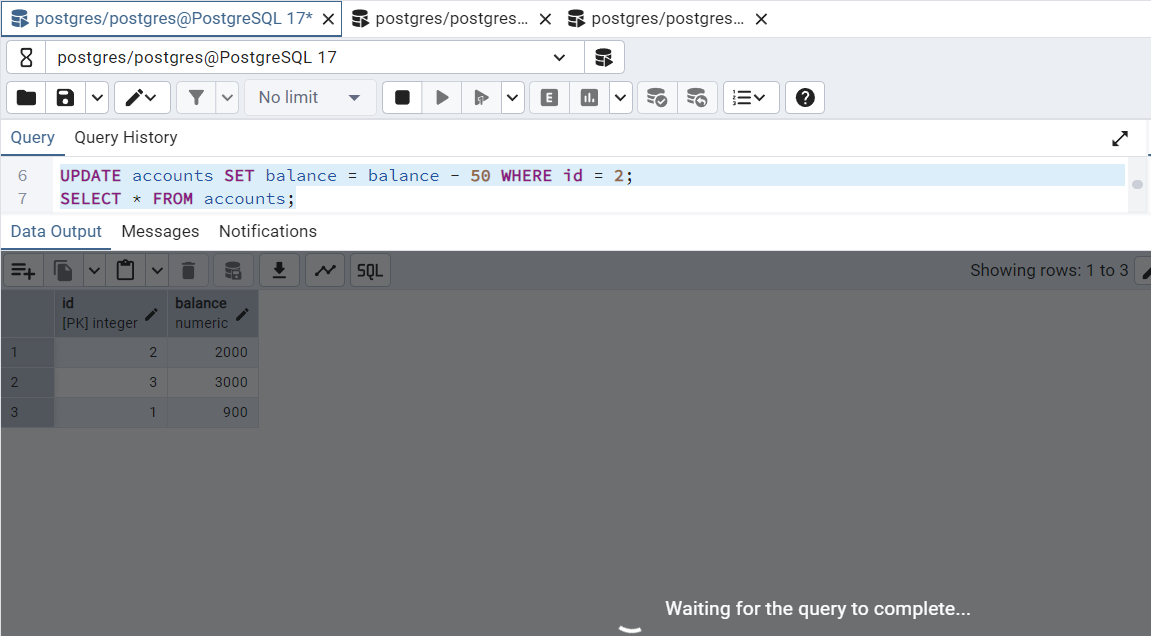
Hint: Подробности запроса смотрите в протоколе сервера.

Context: при изменении кортежа (0,1) в отношении "accounts"

Так как 3 транзакция откатилась, 2 транзакция разблокировалась, и запрос прошел.

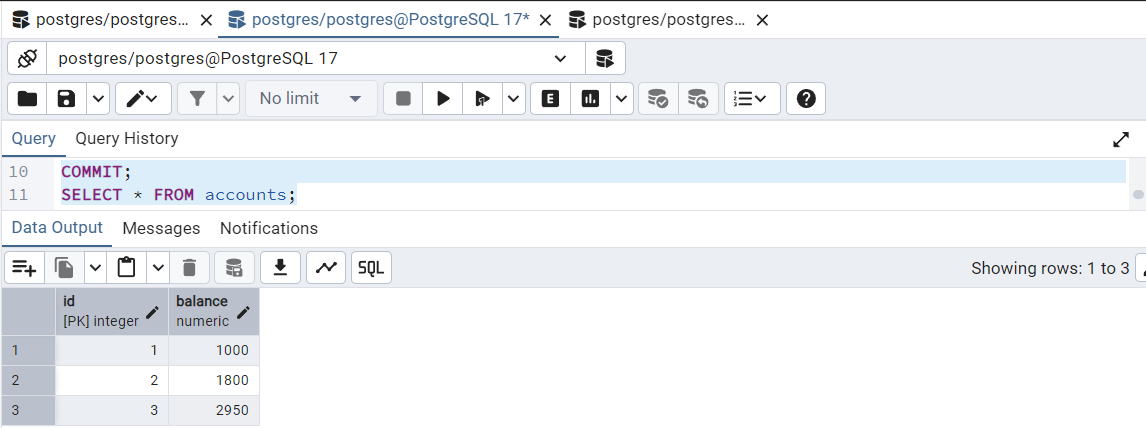


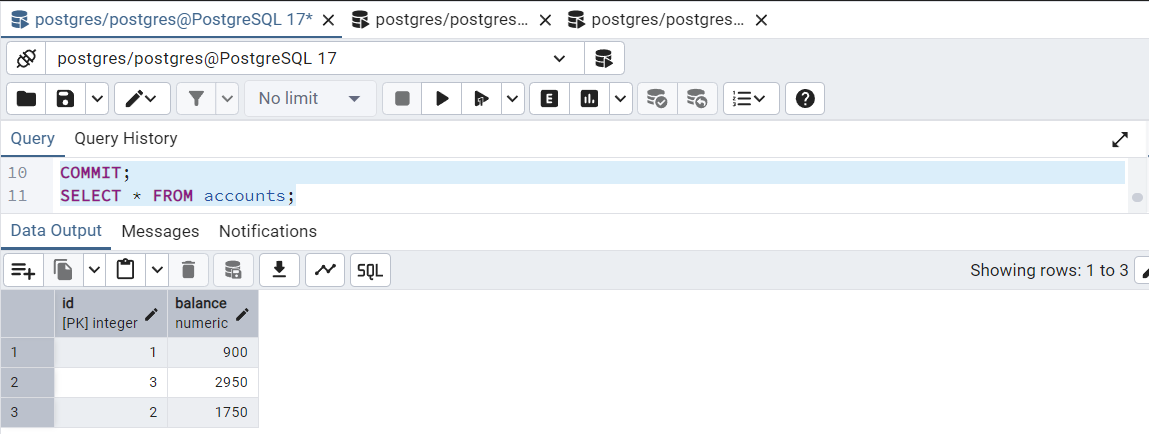
Транзакция 1 по-прежнему висит.



Далее 2 и 1 транзакции были последовательно завершены.

COMMIT;





Что же в итоге произошло? Постгрес где-то под капотом отслеживает зависимости транзакций (вероятно, строит дерево). При возникновении цикла СУБД рандомно выбирает жертву и выдаёт пользаку соответствующую ошибку, блокируя все его запросы за исключением ROLLBACK. При этом освобождаются все транзакции, зависимые от данной. Таким образом, цикл прерывается, и транзакции могут успешно продолжить свое выполнение.

Также обратим внимание на порядок выполнения запросов. Хоть они и были изолированы в отдельных транзакциях, выполнялись они по времени именно в том порядке, в котором были запущены (на это указывает порядок в последнем выводе – постргрес выдает строки в порядке последнего их изменения). Предположительно, на уровне SERIALIZABLE запросы будут выполняться блочно в порядке начала (или завершения – на данный момент не уверен) транзакций.

**1.2 А что на SERIALIZABLE?**

Сбросим таблицу в исходное состояние.

DROP TABLE accounts;

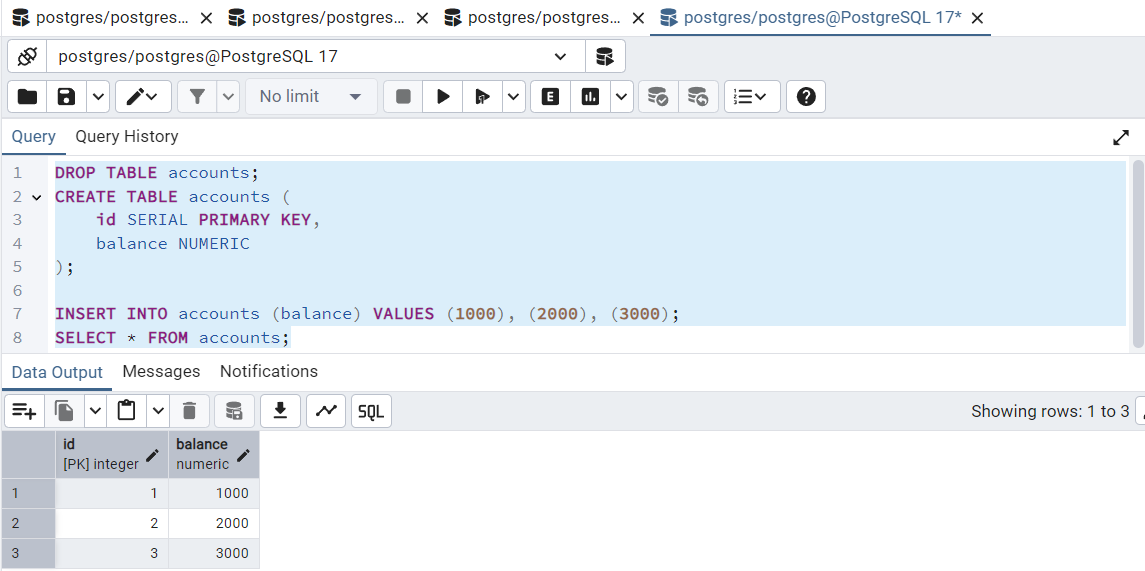
CREATE TABLE accounts (

id SERIAL PRIMARY KEY,

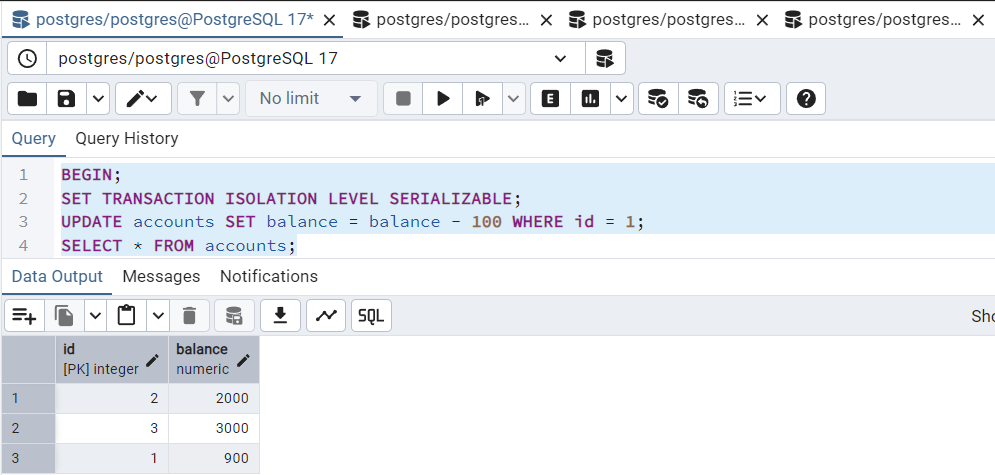
balance NUMERIC

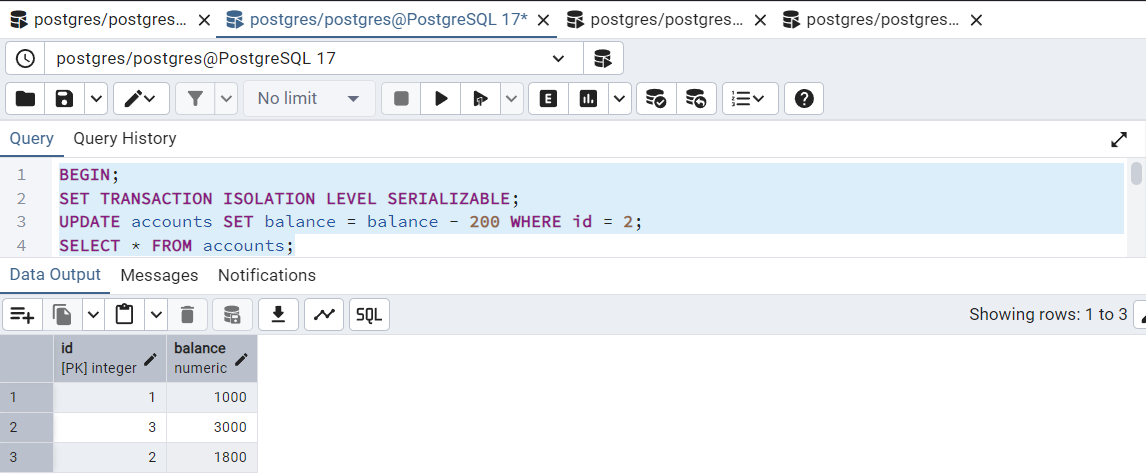
);

INSERT INTO accounts (balance) VALUES (1000), (2000), (3000);

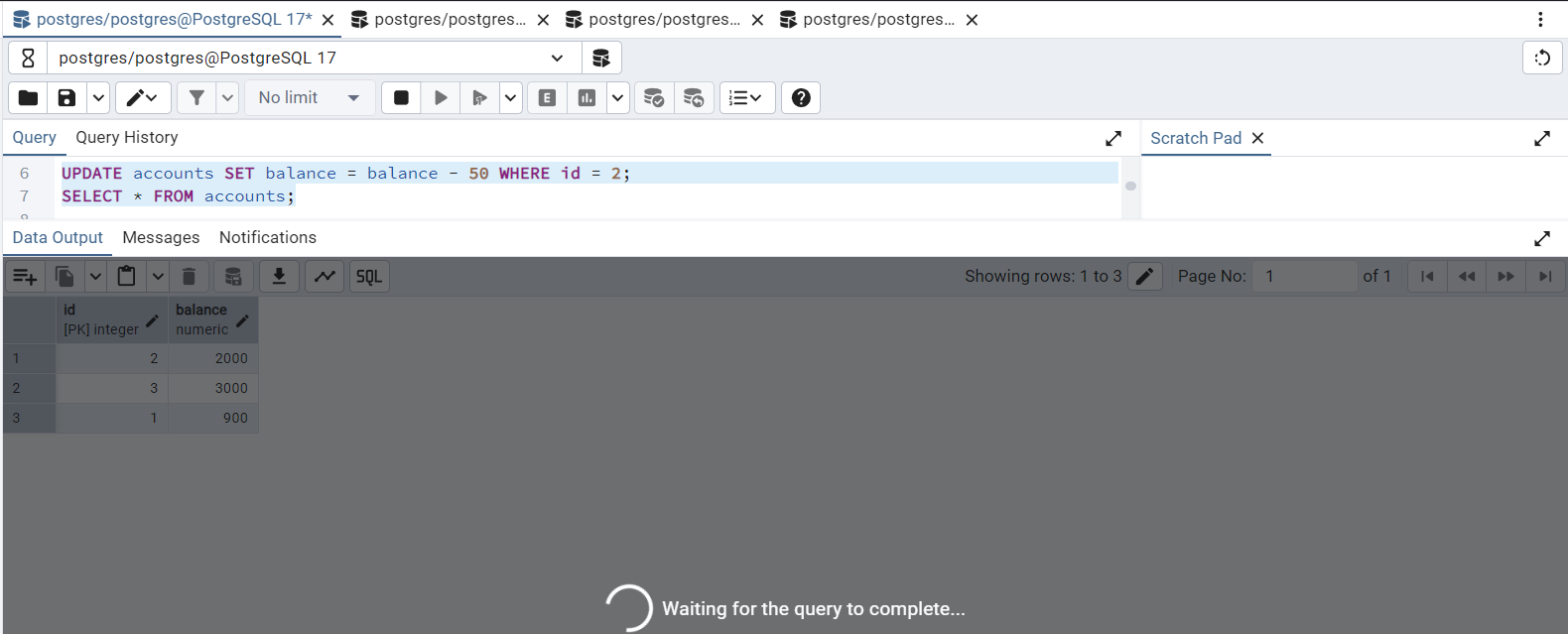


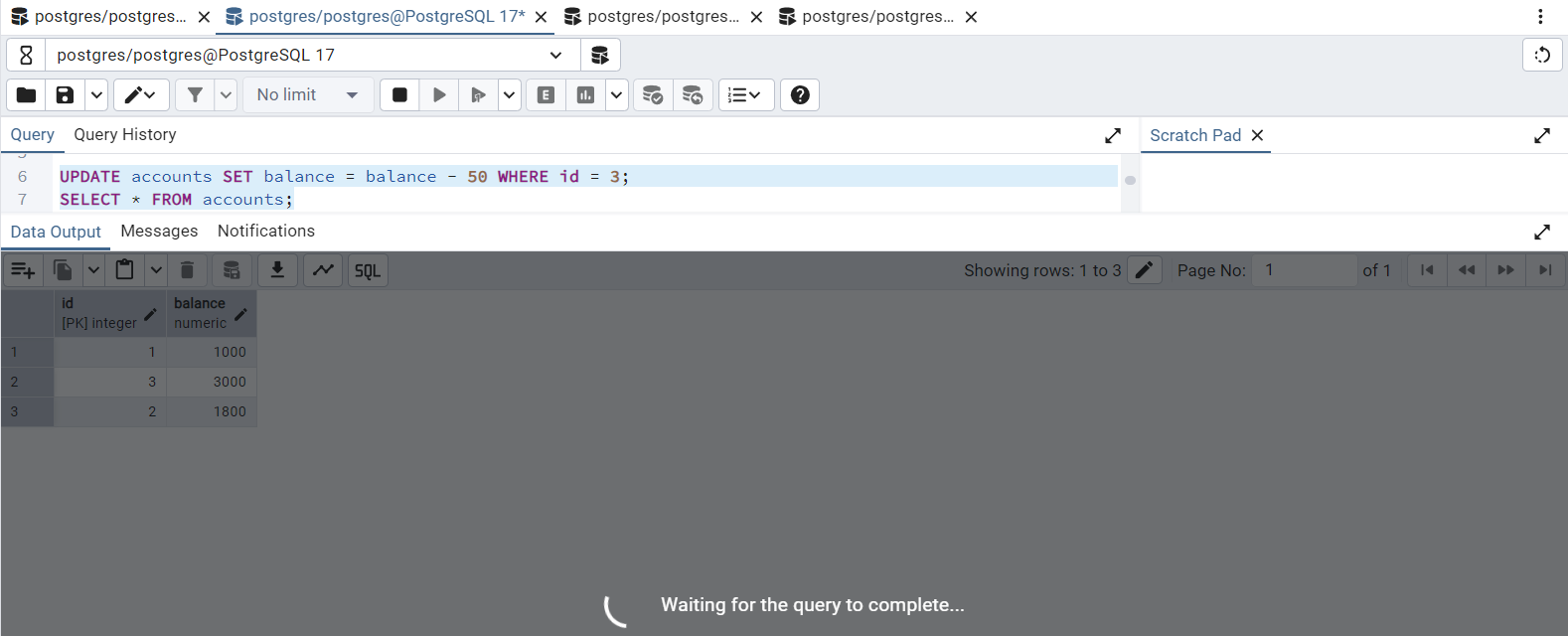
Далее все действия будут аналогичны предыдущему эксперименту, за исключением уровня изоляции транзакций.

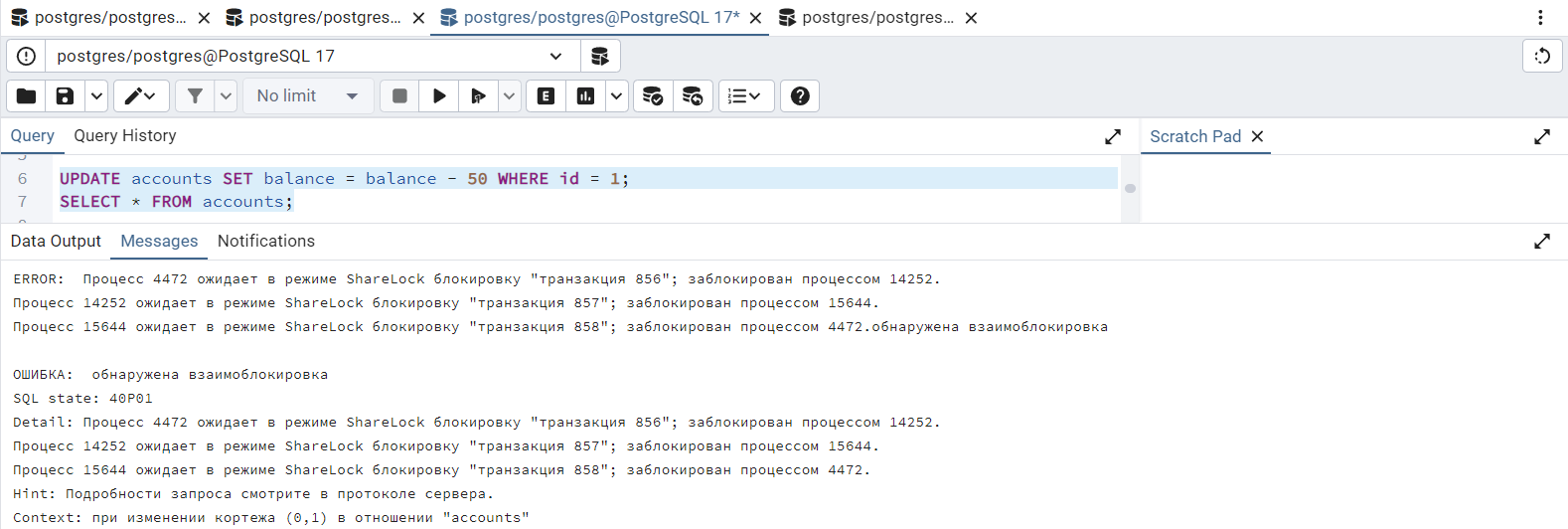












Так же, как и в предыдущем эксперименте, выскочила ошибка с сообщением о взаимной блокировке транзакций

ERROR: Процесс 4472 ожидает в режиме ShareLock блокировку "транзакция 856"; заблокирован процессом 14252.

Процесс 14252 ожидает в режиме ShareLock блокировку "транзакция 857"; заблокирован процессом 15644.

Процесс 15644 ожидает в режиме ShareLock блокировку "транзакция 858"; заблокирован процессом 4472.обнаружена взаимоблокировка

ОШИБКА: обнаружена взаимоблокировка

SQL state: 40P01

Detail: Процесс 4472 ожидает в режиме ShareLock блокировку "транзакция 856"; заблокирован процессом 14252.

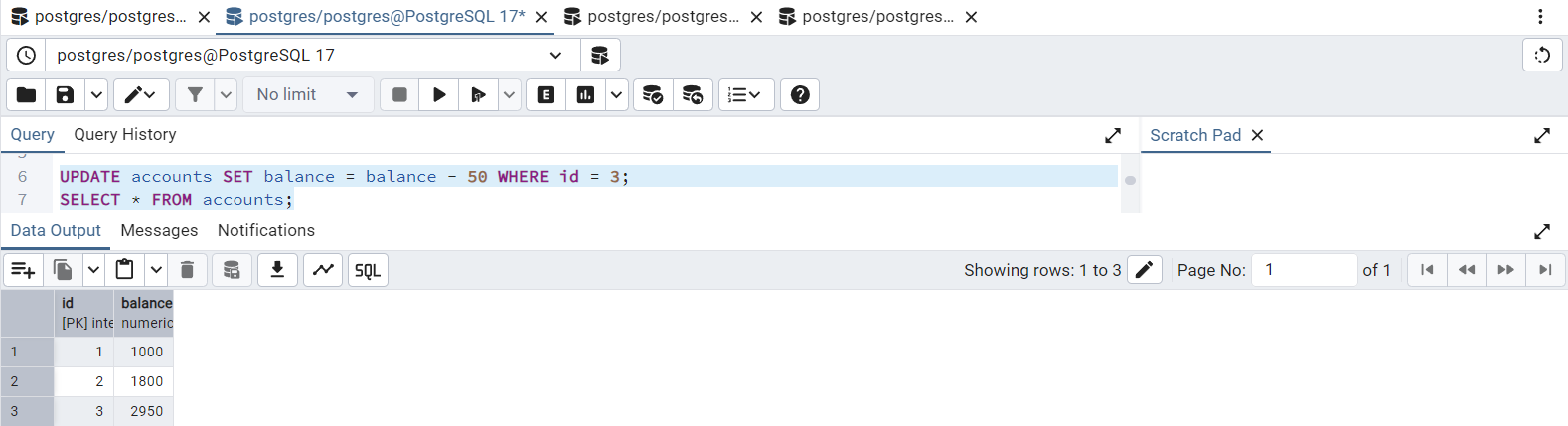
Процесс 14252 ожидает в режиме ShareLock блокировку "транзакция 857"; заблокирован процессом 15644.

Процесс 15644 ожидает в режиме ShareLock блокировку "транзакция 858"; заблокирован процессом 4472.

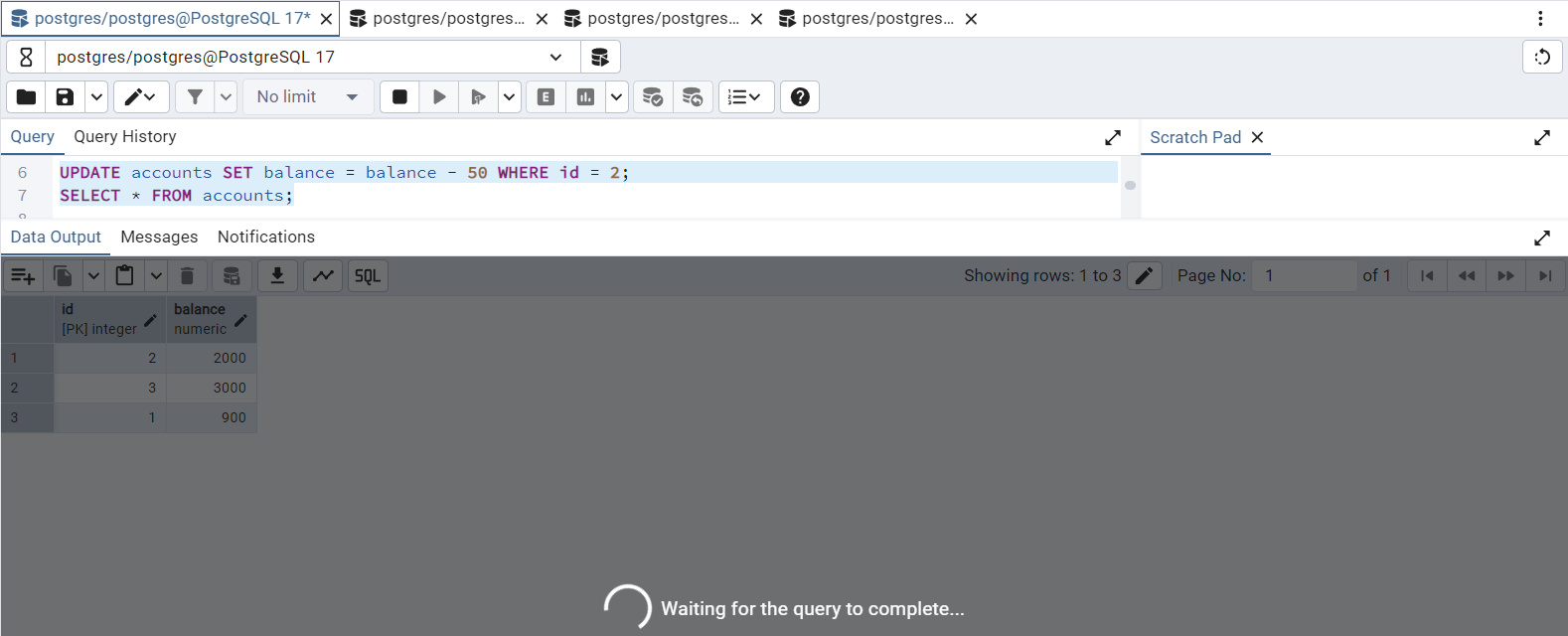
Hint: Подробности запроса смотрите в протоколе сервера.

Context: при изменении кортежа (0,1) в отношении "accounts"

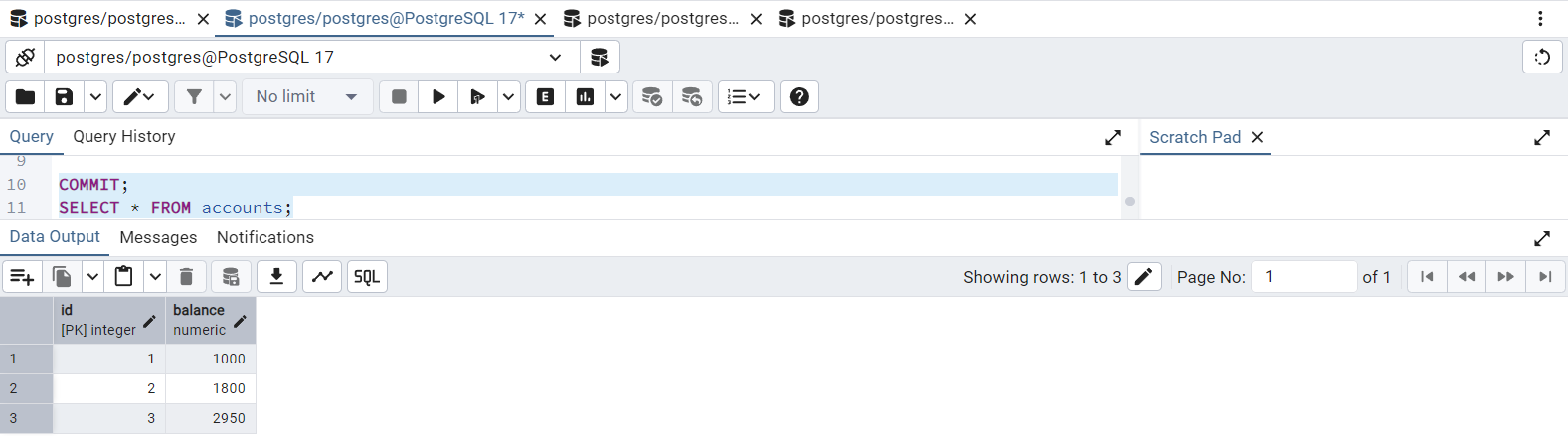
Вторая транзакция освободилась.



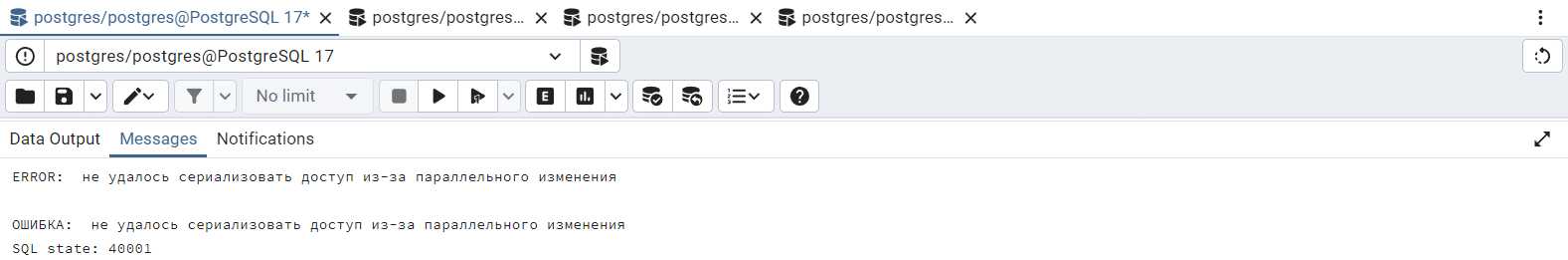
А первая все еще ждет вторую.



Закоммитим вторую транзакцию.



И вдруг возникла ошибка в первой транзакции.

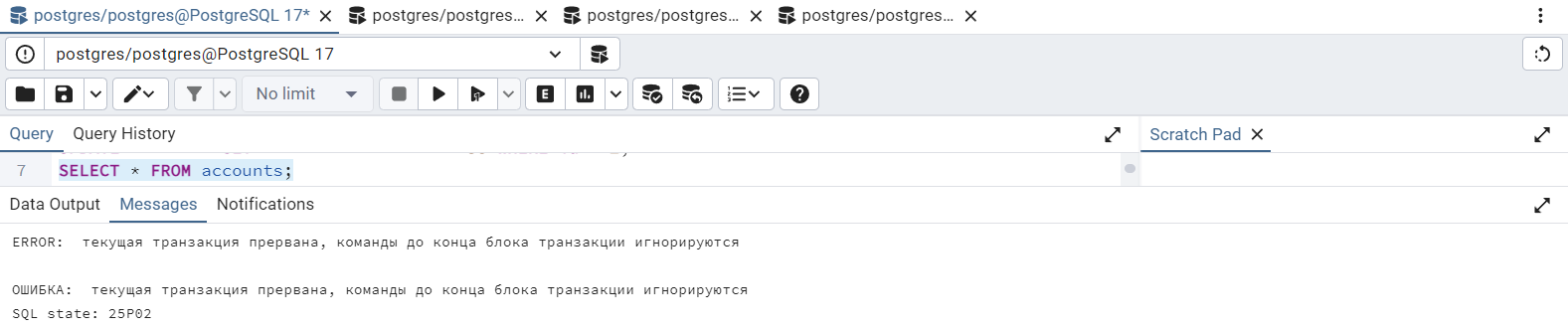


ERROR: не удалось сериализовать доступ из-за параллельного изменения

ОШИБКА: не удалось сериализовать доступ из-за параллельного изменения SQL state: 40001

Посмотрим, что в табличке.

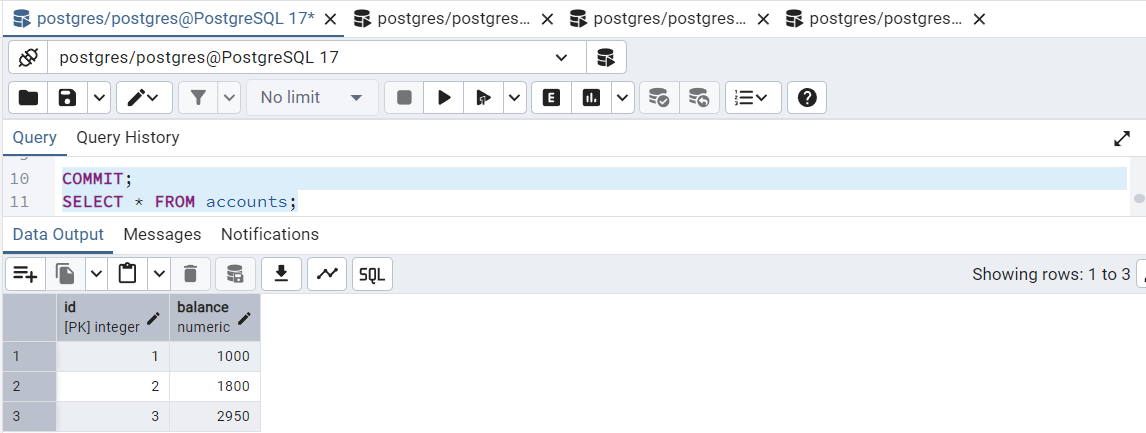
Видим ошибку. СУБД ожидает запрос завершения транзакции (аналогичнаая ситуация была в третьей сессии в сразу после предыдущего эксперимента – я не заскринил).



ERROR: текущая транзакция прервана, команды до конца блока транзакции игнорируются

ОШИБКА: текущая транзакция прервана, команды до конца блока транзакции игнорируются SQL state: 25P02

Закоммитим транзакцию.



Видим, что по итогу применились изменения транзакции только из второй сессии.

Как мне подсказала одна знакомая (нейронка), ошибка сериализации возникла в первой сессии, так как СУБД «не смогла вписать ее в сериализованный порядок выполнения». То есть, оказалось невозможным строго последовательно выполнить транзакции (чего требует данный уровень изоляции) из-за того, что в обеих происходит изменение записи с индексом 2, и при этом вторая транзакция завершилась раньше первой, хотя первая была начата раньше. Другими словами: СУБД не смогла сначала выполнить все запросы первой транзакции, а потом уже второй, потому что вторая была завершена раньше, и при этом в ней менялись те же данные, что и в первой.

Приложение (из отчета о лабораторной работе 2-2)

* [ERD структуры таблиц](https://github.com/KATEHOK/DBS-labs-7/blob/main/1/pics/ERD.png);
* [Sql-скрипт сброса состояния таблицы для экспериментов 4.1-4.3](https://github.com/KATEHOK/DBS-labs-7/blob/main/2/script/4--reset.sql);
* [Sql-скрипт первой сессии для эксперимента 4.1](https://github.com/KATEHOK/DBS-labs-7/blob/main/2/script/4-1-first-session.sql);
* [Sql-скрипт второй сессии для экспериментов 4.1-4.3](https://github.com/KATEHOK/DBS-labs-7/blob/main/2/script/4--second-session.sql);
* [отчет (docx)](https://github.com/KATEHOK/DBS-labs-7/blob/main/2/report.docx);
* [отчет (pdf)](https://github.com/KATEHOK/DBS-labs-7/blob/main/2/report.pdf).