

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

На защиту: устройте deadlock трёх или более транзакций. Если СУБД может это разрулить, то как она это делает? Изменится ли что-нибудь на уровне SERIALIZABLE?

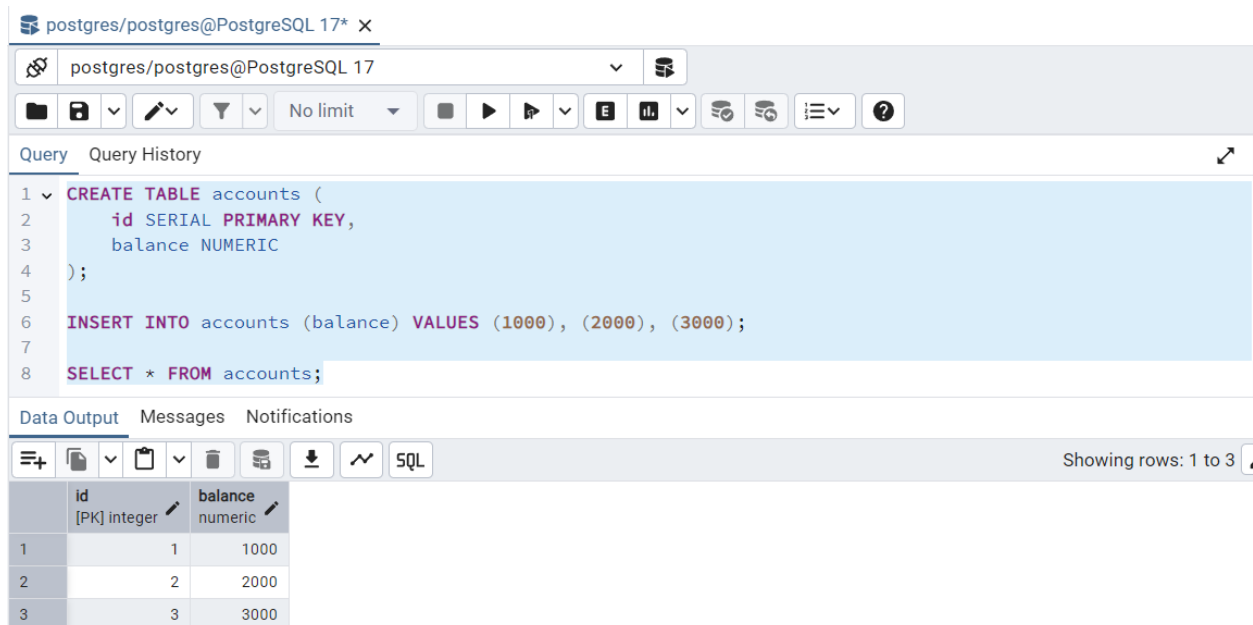
Дедлок — это ситуация, когда две или более транзакции блокируют друг друга, ожидая доступа к ресурсам, занятым другой транзакцией. PostgreSQL обнаруживает дедлоки автоматически и завершает одну из транзакций с ошибкой, чтобы другие могли продолжить выполнение.

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);
```



The screenshot shows a PostgreSQL client window titled 'postgres/postgres@PostgreSQL 17*'. The interface includes a toolbar with icons for file operations, query execution, and settings. The 'Query' tab is active, displaying the following SQL script:

```
1 CREATE TABLE accounts (  
2     id SERIAL PRIMARY KEY,  
3     balance NUMERIC  
4 );  
5  
6 INSERT INTO accounts (balance) VALUES (1000), (2000), (3000);  
7  
8 SELECT * FROM accounts;
```

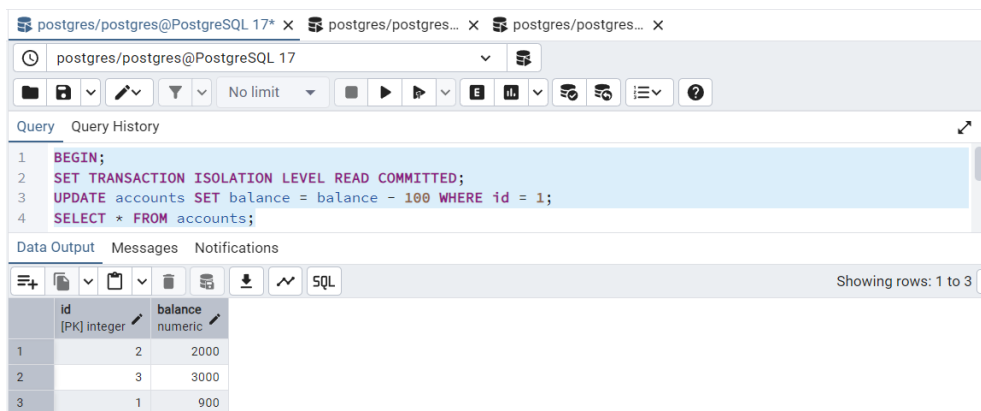
Below the query editor, the 'Data Output' tab is selected, showing the results of the 'SELECT * FROM accounts;' query. The output is a table with two columns: 'id' (integer, primary key) and 'balance' (numeric). The data is as follows:

	id [PK] integer	balance numeric
1	1	1000
2	2	2000
3	3	3000

The bottom right corner of the window indicates 'Showing rows: 1 to 3'.

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;
```

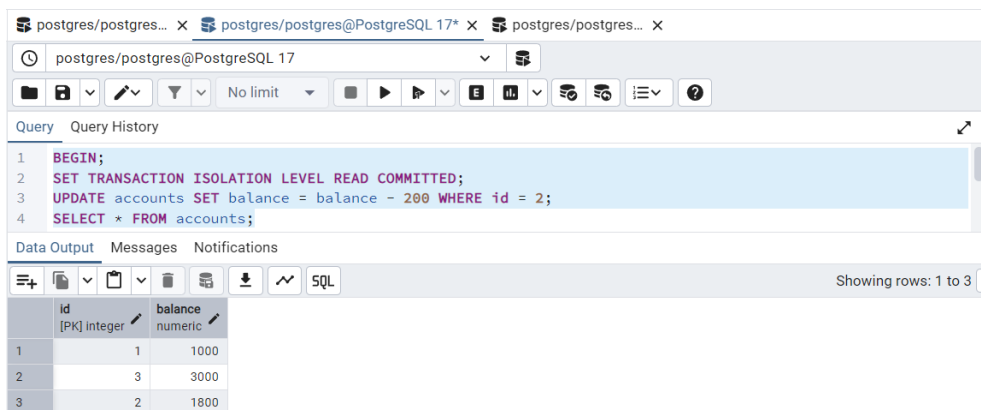


The screenshot shows the pgAdmin interface with a SQL query executed. The query is as follows:

```
1 BEGIN;  
2 SET TRANSACTION ISOLATION LEVEL READ COMMITTED;  
3 UPDATE accounts SET balance = balance - 100 WHERE id = 1;  
4 SELECT * FROM accounts;
```

The results are displayed in a table with 3 rows:

	id [PK] integer	balance numeric
1	2	2000
2	3	3000
3	1	900

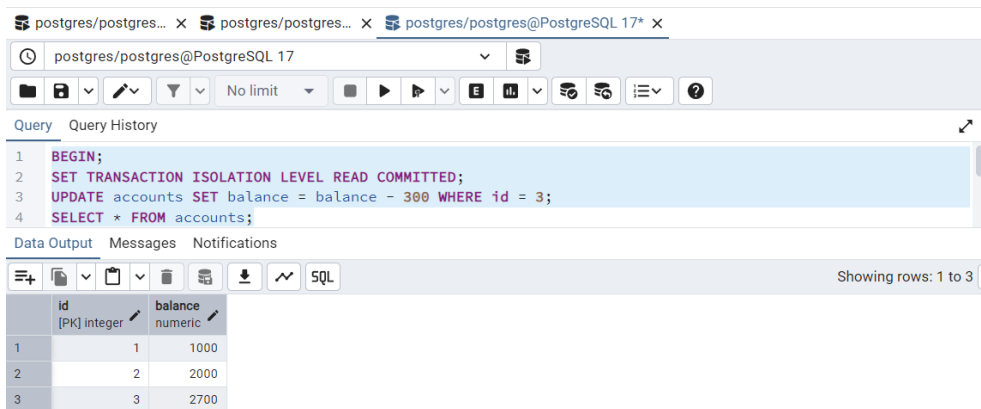


The screenshot shows the pgAdmin interface with a SQL query executed. The query is as follows:

```
1 BEGIN;  
2 SET TRANSACTION ISOLATION LEVEL READ COMMITTED;  
3 UPDATE accounts SET balance = balance - 200 WHERE id = 2;  
4 SELECT * FROM accounts;
```

The results are displayed in a table with 3 rows:

	id [PK] integer	balance numeric
1	1	1000
2	3	3000
3	2	1800



The screenshot shows the pgAdmin interface with a SQL query executed. The query is as follows:

```
1 BEGIN;  
2 SET TRANSACTION ISOLATION LEVEL READ COMMITTED;  
3 UPDATE accounts SET balance = balance - 300 WHERE id = 3;  
4 SELECT * FROM accounts;
```

The results are displayed in a table with 3 rows:

	id [PK] integer	balance numeric
1	1	1000
2	2	2000
3	3	2700

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;
```

postgres/postgres@PostgreSQL 17* x postgres/postgres... x postgres/postgres... x

postgres/postgres@PostgreSQL 17

Query Query History

```
6 UPDATE accounts SET balance = balance - 50 WHERE id = 2;  
7 SELECT * FROM accounts;
```

Data Output Messages Notifications

	id [PK] integer	balance numeric
1	2	2000
2	3	3000
3	1	900

Showing rows: 1 to 3

Waiting for the query to complete...

postgres/postgres... x postgres/postgres@PostgreSQL 17* x postgres/postgres... x

postgres/postgres@PostgreSQL 17

Query Query History

```
6 UPDATE accounts SET balance = balance - 50 WHERE id = 3;  
7 SELECT * FROM accounts;
```

Data Output Messages Notifications

	id [PK] integer	balance numeric
1	1	1000
2	3	3000
3	2	1800

Waiting for the query to complete...

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

The screenshot shows a PostgreSQL client window with the title 'postgres/postgres@PostgreSQL 17*'. The query editor contains two lines of SQL: `UPDATE accounts SET balance = balance - 50 WHERE id = 1;` on line 6 and `SELECT * FROM accounts;` on line 7. The 'Messages' tab is selected, displaying an error message in Russian. The error text is: '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' followed by '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.', and 'Context: при изменении кортежа (0,1) в отношении "accounts"'.

```
postgres/postgres... X postgres/postgres... X postgres/postgres@PostgreSQL 17* X
postgres/postgres@PostgreSQL 17
Query History
6 UPDATE accounts SET balance = balance - 50 WHERE id = 1;
7 SELECT * FROM accounts;
Data Output Messages Notifications
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"
```

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.

The screenshot shows a PostgreSQL client window with the title 'postgres/postgres@PostgreSQL 17'. The query editor contains two lines of SQL: `UPDATE accounts SET balance = balance - 50 WHERE id = 3;` and `SELECT * FROM accounts;`. The 'Data Output' tab is active, displaying a table with 3 rows and 2 columns: 'id [PK] integer' and 'balance numeric'. The rows are: (1, 1000), (2, 1800), and (3, 2950). The status bar at the bottom right indicates 'Showing rows: 1 to 3'.

	id [PK] integer	balance numeric
1	1	1000
2	2	1800
3	3	2950

Транзакция 1 по-прежнему висит.

The screenshot shows the same PostgreSQL client window. The query editor now contains: `UPDATE accounts SET balance = balance - 50 WHERE id = 2;` and `SELECT * FROM accounts;`. The 'Data Output' tab is active, but the table is empty. The status bar at the bottom right indicates 'Showing rows: 1 to 3' and 'Waiting for the query to complete...'. The table schema is visible: 'id [PK] integer' and 'balance numeric'.

	id [PK] integer	balance numeric
--	-----------------	-----------------

Далее 2 и 1 транзакции были последовательно завершены.

COMMIT;

postgres/postgres@PostgreSQL 17

```

10 COMMIT;
11 SELECT * FROM accounts;

```

Data Output Messages Notifications

Showing rows: 1 to 3

	id [PK] integer	balance numeric
1	1	1000
2	2	1800
3	3	2950

postgres/postgres@PostgreSQL 17*

```

10 COMMIT;
11 SELECT * FROM accounts;

```

Data Output Messages Notifications

Showing rows: 1 to 3

	id [PK] integer	balance numeric
1	1	900
2	3	2950
3	2	1750

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

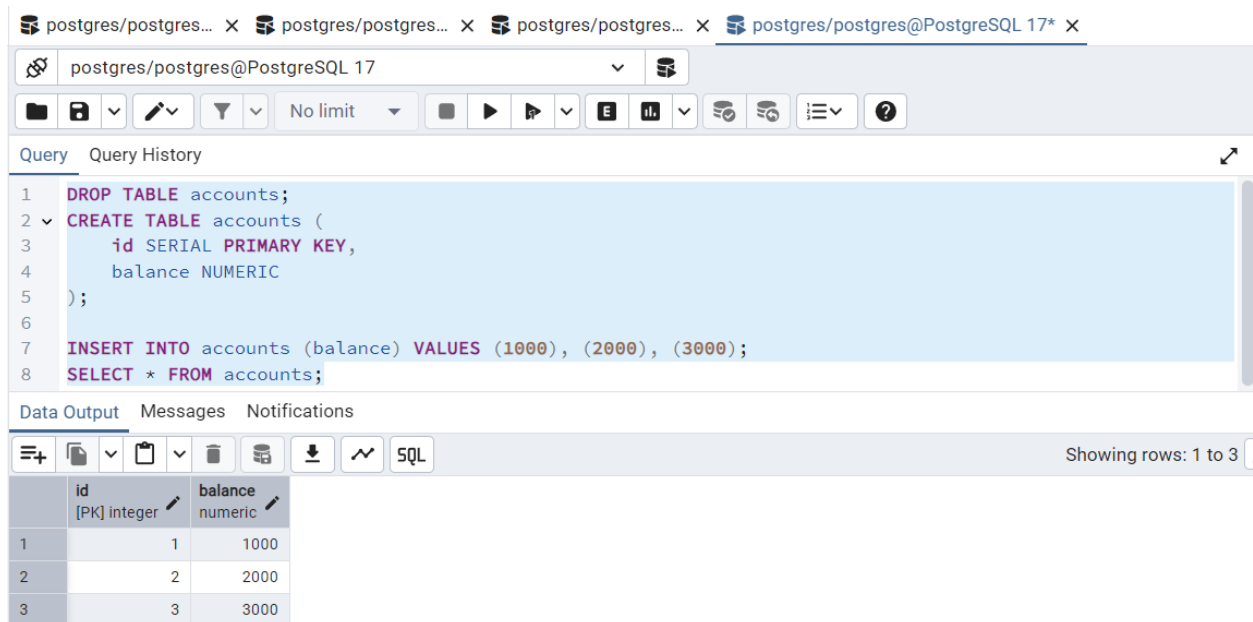
Также обратим внимание на порядок выполнения запросов. Хотя они и были изолированы в отдельных транзакциях, выполнялись они по времени именно в том порядке, в котором были запущены (на это указывает порядок в последнем выводе – постгрес выдает строки в порядке последнего их изменения). Предположительно, 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);
```



The screenshot shows a PostgreSQL client interface with multiple tabs. The active tab is 'postgres/postgres@PostgreSQL 17*'. The query editor contains the following SQL statements:

```
1 DROP TABLE accounts;  
2 CREATE TABLE accounts (  
3     id SERIAL PRIMARY KEY,  
4     balance NUMERIC  
5 );  
6  
7 INSERT INTO accounts (balance) VALUES (1000), (2000), (3000);  
8 SELECT * FROM accounts;
```

The 'Data Output' tab is selected, showing the results of the last query. The table has two columns: 'id' (integer, primary key) and 'balance' (numeric). The results are as follows:

	id [PK] integer	balance numeric
1	1	1000
2	2	2000
3	3	3000

The interface also shows a toolbar with various icons for query execution and a status bar indicating 'Showing rows: 1 to 3'.

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

postgres/postgres@PostgreSQL 17* X postgres/postgres... X postgres/postgres... X postgres/postgres... X

postgres/postgres@PostgreSQL 17

Query Query History

```
1 BEGIN;
2 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
3 UPDATE accounts SET balance = balance - 100 WHERE id = 1;
4 SELECT * FROM accounts;
```

Data Output Messages Notifications

	id [PK] integer	balance numeric
1	2	2000
2	3	3000
3	1	900

postgres/postgres... X postgres/postgres@PostgreSQL 17* X postgres/postgres... X postgres/postgres... X

postgres/postgres@PostgreSQL 17

Query Query History

```
1 BEGIN;
2 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
3 UPDATE accounts SET balance = balance - 200 WHERE id = 2;
4 SELECT * FROM accounts;
```

Data Output Messages Notifications

	id [PK] integer	balance numeric
1	1	1000
2	3	3000
3	2	1800

postgres/postgres... X postgres/postgres... X postgres/postgres@PostgreSQL 17* X postgres/postgres... X

postgres/postgres@PostgreSQL 17

Query Query History

```
1 BEGIN;
2 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
3 UPDATE accounts SET balance = balance - 300 WHERE id = 3;
4 SELECT * FROM accounts;
```

Data Output Messages Notifications

	id [PK] integer	balance numeric
1	1	1000
2	2	2000
3	3	2700

postgres/postgres@PostgreSQL 17* x postgres/postgres... x postgres/postgres... x postgres/postgres... x

postgres/postgres@PostgreSQL 17

Query Query History Scratch Pad x

```
6 UPDATE accounts SET balance = balance - 50 WHERE id = 2;
7 SELECT * FROM accounts;
```

Data Output Messages Notifications

	id [PK] integer	balance numeric
1	2	2000
2	3	3000
3	1	900

Showing rows: 1 to 3 Page No: 1 of 1

Waiting for the query to complete...

postgres/postgres... x postgres/postgres@PostgreSQL 17* x postgres/postgres... x postgres/postgres... x

postgres/postgres@PostgreSQL 17

Query Query History Scratch Pad x

```
6 UPDATE accounts SET balance = balance - 50 WHERE id = 3;
7 SELECT * FROM accounts;
```

Data Output Messages Notifications

	id [PK] integer	balance numeric
1	1	1000
2	3	3000
3	2	1800

Showing rows: 1 to 3 Page No: 1 of 1

Waiting for the query to complete...

postgres/postgres... x postgres/postgres... x postgres/postgres@PostgreSQL 17* x postgres/postgres... x

postgres/postgres@PostgreSQL 17

Query Query History Scratch Pad x

```
6 UPDATE accounts SET balance = balance - 50 WHERE id = 1;
7 SELECT * FROM accounts;
```

Data Output Messages Notifications

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"

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.

The screenshot shows a PostgreSQL client interface with a query execution error. The query was:

```
6 UPDATE accounts SET balance = balance - 50 WHERE id = 3;  
7 SELECT * FROM accounts;
```

The error message is: ERROR: Процесс 4472 ожидает в режиме ShareLock блокировку "транзакция 856"; заблокирован процессом 14252. SQL state: 40P01. The hint is: Подробности запроса смотрите в протоколе сервера. The context is: при изменении кортежа (0,1) в отношении "accounts".

The data output shows the following table:

	id	balance
1	1	1000
2	2	1800
3	3	2950

А первая все еще ждет вторую.

postgres/postgres@PostgreSQL 17* x postgres/postgres... x postgres/postgres... x postgres/postgres... x

postgres/postgres@PostgreSQL 17

Query Query History Scratch Pad x

```
6 UPDATE accounts SET balance = balance - 50 WHERE id = 2;
7 SELECT * FROM accounts;
```

Data Output Messages Notifications

	id [PK] integer	balance numeric
1	2	2000
2	3	3000
3	1	900

Showing rows: 1 to 3 Page No: 1 of 1

Waiting for the query to complete...

Закоммитим вторую транзакцию.

postgres/postgres... x postgres/postgres@PostgreSQL 17* x postgres/postgres... x postgres/postgres... x

postgres/postgres@PostgreSQL 17

Query Query History Scratch Pad x

```
9
10 COMMIT;
11 SELECT * FROM accounts;
```

Data Output Messages Notifications

	id [PK] integer	balance numeric
1	1	1000
2	2	1800
3	3	2950

Showing rows: 1 to 3 Page No: 1 of 1

И вдруг возникла ошибка в первой транзакции.

postgres/postgres@PostgreSQL 17* x postgres/postgres... x postgres/postgres... x postgres/postgres... x

postgres/postgres@PostgreSQL 17

Data Output Messages Notifications

ERROR: не удалось сериализовать доступ из-за параллельного изменения

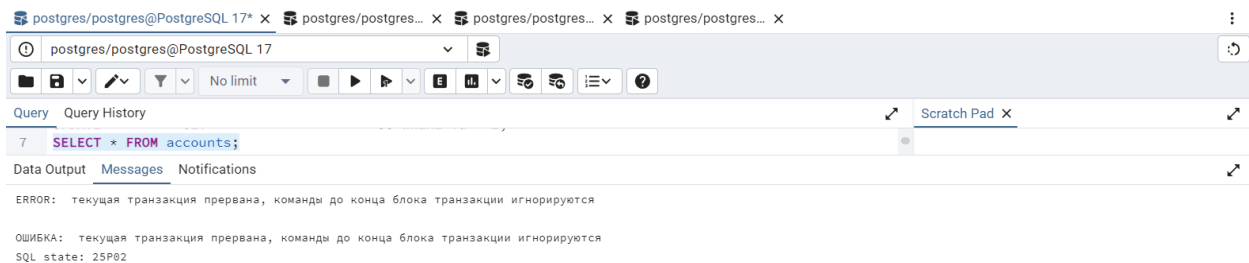
ОШИБКА: не удалось сериализовать доступ из-за параллельного изменения
SQL state: 40001

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.

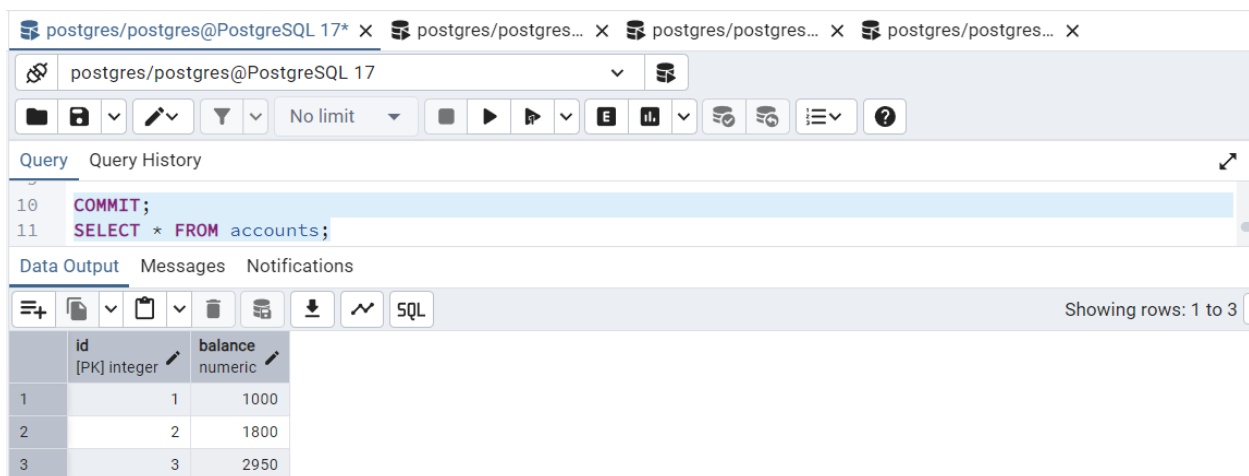
Видим ошибку. СУБД ожидает запрос завершения транзакции (аналогичная ситуация была в третьей сессии в сразу после предыдущего эксперимента – я не заскринил).



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;
- Sql-скрипт сброса состояния таблицы для экспериментов 4.1-4.3;
- Sql-скрипт первой сессии для эксперимента 4.1;
- Sql-скрипт второй сессии для экспериментов 4.1-4.3;
- отчет (docx);
- отчет (pdf).