

Факультет инфокоммуникационных технологий

Направление подготовки 11.03.02

Практическая работа №5

Выполнил:

Дощенников Никита Андреевич

Группа: К3221

Проверила:

Татьяна Евгеньевна Войтюк

Цель работы

Изучить механику транзакций в postgresql, понять принципы работы блокировок на уровне строк, разобраться с поведением конкурентных подключений, а также отработать применение операторов `BEGIN`, `COMMIT`, `ROLLBACK`, сохранённых точек и диагностику ошибок, возникающих при конфликте транзакций.

Задачи, решаемые при выполнении работы

- Выполнить создание и модификацию данных в таблицах с использованием транзакций.
- Исследовать работу блокировок при одновременном доступе из нескольких подключений.
- Проверить поведение транзакции при возникновении ошибок.
- Освоить применение `SAVEPOINT`, `ROLLBACK TO SAVEPOINT`, `COMMIT` и полного `ROLLBACK`.
- Проанализировать различия в изоляции и видимости изменений между параллельными сессиями.
- Сделать выводы о корректном использовании транзакций для обеспечения целостности данных.

Исходные данные

1. База данных postgresql, содержащая таблицы схемы “EmployeesDepartments” и таблицу `public.t1`.
2. Доступ к системе с двух независимых подключений для моделирования конкурентной работы.
3. Набор sql-операторов для выполнения транзакций: `BEGIN`, `UPDATE`, `SELECT`, `COMMIT`, `ROLLBACK`, `SAVEPOINT`.
4. Записи, над которыми проводились операции.
5. Средства для выполнения запросов и наблюдения за блокировками и поведением транзакций.

Выполнение работы

Задание 1. Изучение транзакций COMMIT и ROLLBACK в PostgreSQL

Сначала я выключил автокоммит в настройках pgadmin. (Рис. 1)

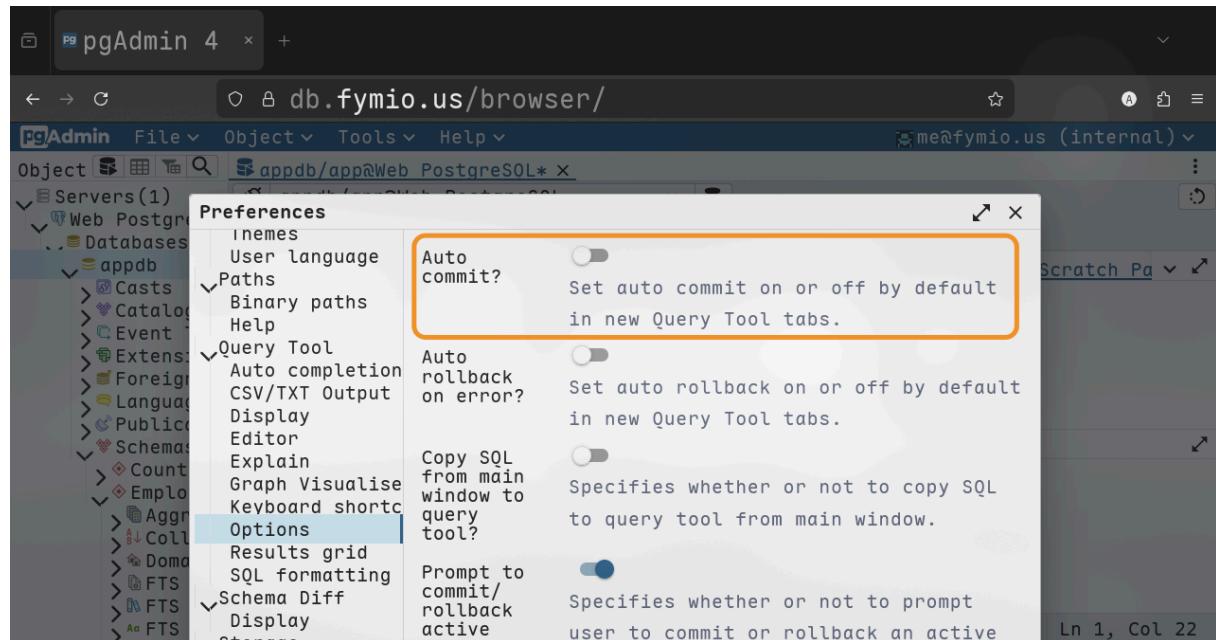


Рис. 1: Выключение автокоммитов.

Я открыл “Querry Tool” и ввел в окне запроса код с Скрипт. 1: (Рис. 2)

```
SELECT 'Before' AS stage, "EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME", "SALARY"
FROM "EmployeesDepartments"."EMPLOYEES"
WHERE "EMPLOYEE_ID" = 101;
```

Скрипт. 1: Выборка данных сотрудника до начала транзакции.

Запрашивает текущее значение полей сотрудника с ID 101, чтобы зафиксировать исходное состояние перед выполнением изменений в транзакции.

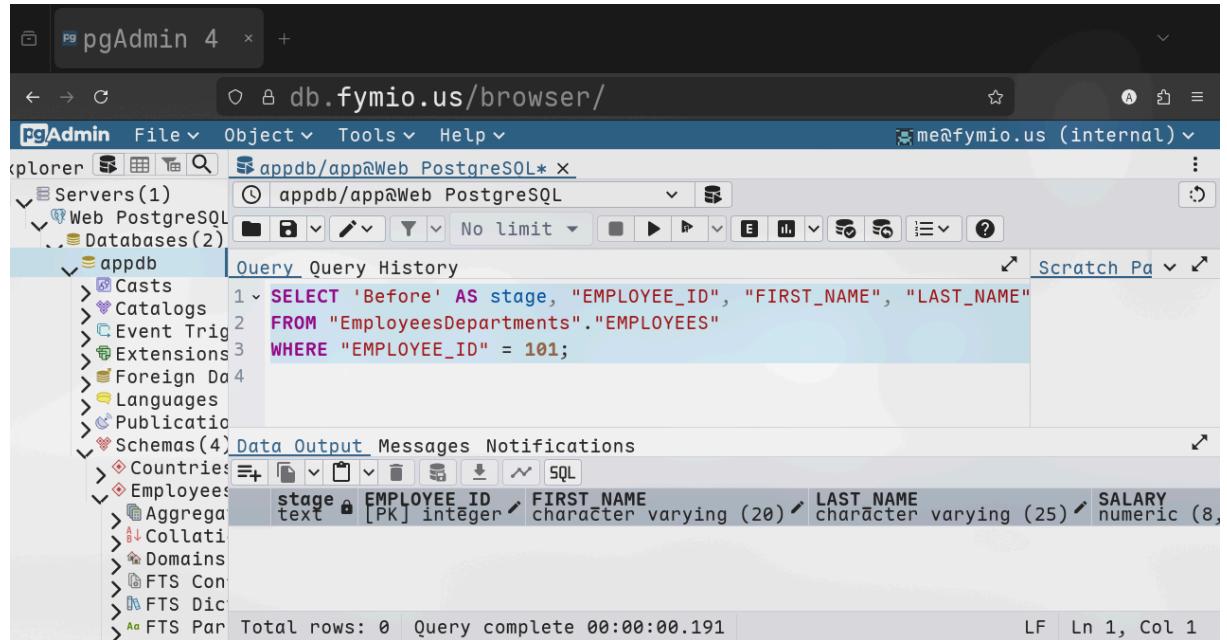


Рис. 2: Результат выполнения Скрипт. 1

BEGIN;

Скрипт. 2: Начало транзакции.

Запускает новую транзакцию, в рамках которой дальнейшие операции будут выполняться атомарно и не будут видны другим сессиям до COMMIT или ROLLBACK.

pgAdmin 4

File Object Tools Help

me@fymio.us (internal)

Explorer Servers(1) appdb/app@Web PostgreSQL Databases(2) appdb

Query History

1 BEGIN;

WARNING: there is already a transaction in progress

BEGIN

Query returned successfully in 248 msec.

Total rows: 0 Query complete 00:00:00.248

LF Ln 1, Col 7

Рис. 3: Результат выполнения Скрипт. 2

```
UPDATE "EmployeesDepartments"."EMPLOYEES"  
SET "SALARY" = "SALARY" * 1.1  
WHERE "EMPLOYEE_ID" = 101;
```

Скрипт. 3: Повышение платы сотрудника на 10%.

Обновляет поле “SALARY” у сотрудника с ID 101, увеличивая его значение в 1.1 раза внутри текущей транзакции. (Рис. 4)

The screenshot shows the pgAdmin 4 interface. The title bar reads "pgAdmin 4" and the address bar shows "db.fymio.us/browser/". The left sidebar displays the database structure under "appdb/app@Web PostgreSQL". The main query editor window contains the following SQL code:

```
1 UPDATE "EmployeesDepartments"."EMPLOYEES"
2 SET "SALARY" = "SALARY" * 1.1
3 WHERE "EMPLOYEE_ID" = 101;
```

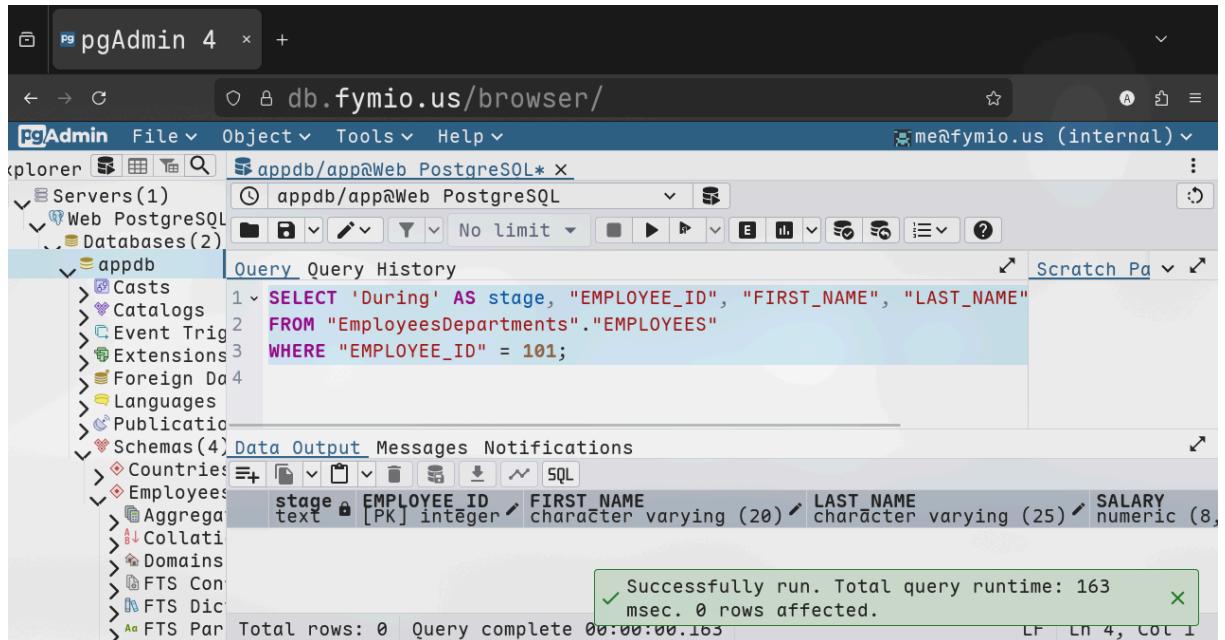
The status bar at the bottom indicates "Query returned successfully in 162 msec." and "Total rows: 1 Query complete 00:00:00.162".

Рис. 4: Результат выполнения Скрипт. 3

```
SELECT 'During' AS stage, "EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME", "SALARY"
FROM "EmployeesDepartments"."EMPLOYEES"
WHERE "EMPLOYEE_ID" = 101;
```

Скрипт. 4: Проверка значения внутри транзакции.

Выводит обновлённые данные сотрудника с ID 101 во время выполнения транзакции, до фиксации изменений. (Рис. 5)



The screenshot shows the pgAdmin 4 interface. The title bar says 'pgAdmin 4'. The left sidebar shows a tree structure of databases, servers, and objects. The main area is a 'Query' window with the following content:

```
1 > SELECT 'During' AS stage, "EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME"
2 > FROM "EmployeesDepartments"."EMPLOYEES"
3 > WHERE "EMPLOYEE_ID" = 101;
```

Below the query window, a table is displayed with the following schema and data:

stage	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
text	[PK] integer	character varying (20)	character varying (25)	numeric (8,2)

A message box at the bottom right of the table area says: 'Successfully run. Total query runtime: 163 msec. 0 rows affected.'

Рис. 5: Результат выполнения Скрипт. 4

COMMIT;

Скрипт. 5: Фиксация изменений транзакции.

Подтверждает все изменения, выполненные в текущей транзакции, делая их постоянными и видимыми для других сессий. (Рис. 6)

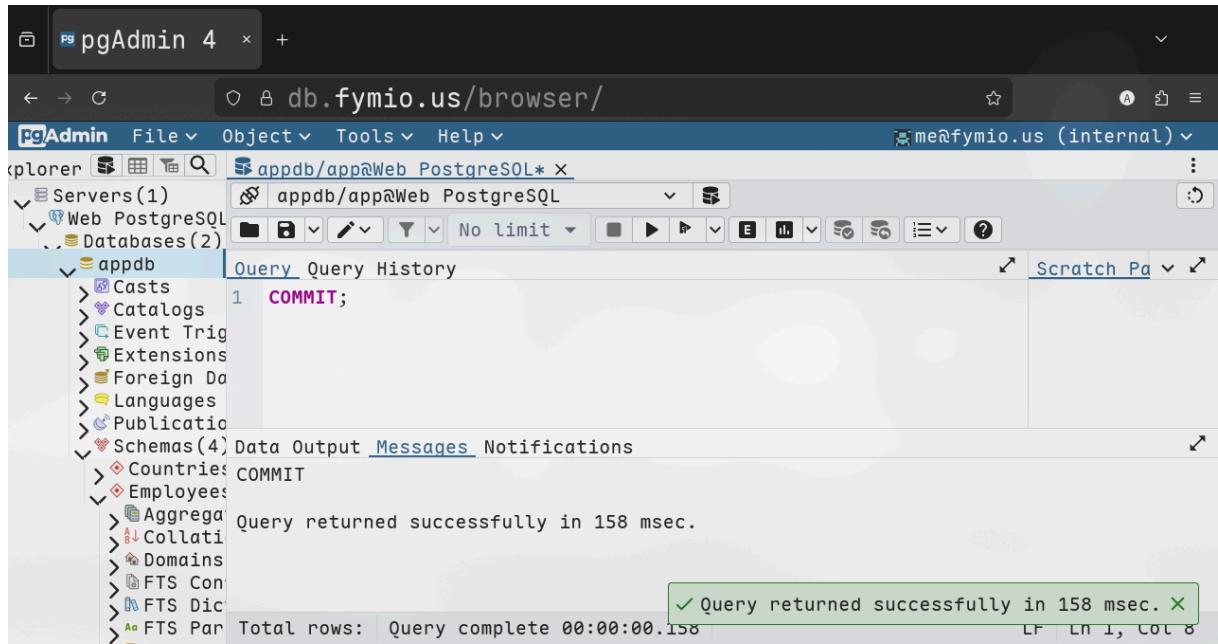


Рис. 6: Результат выполнения Скрипт. 5

```
SELECT 'After' AS stage, "EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME", "SALARY"
FROM "EmployeesDepartments"."EMPLOYEES"
WHERE "EMPLOYEE_ID" = 101;
```

Скрипт. 6: Проверка значения после фиксации транзакции.

Выводит окончательные данные сотрудника с ID 101 после выполнения `COMMIT`, показывая зафиксированное изменение зарплаты. (Рис. 7)

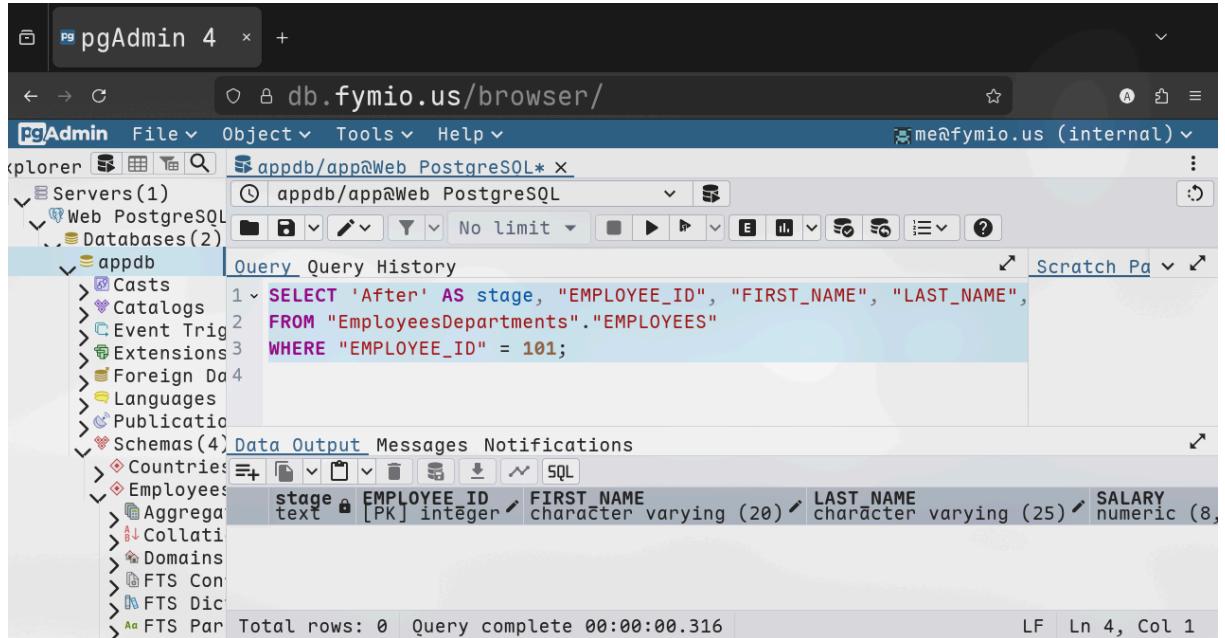


Рис. 7: Результат выполнения Скрипт. 6

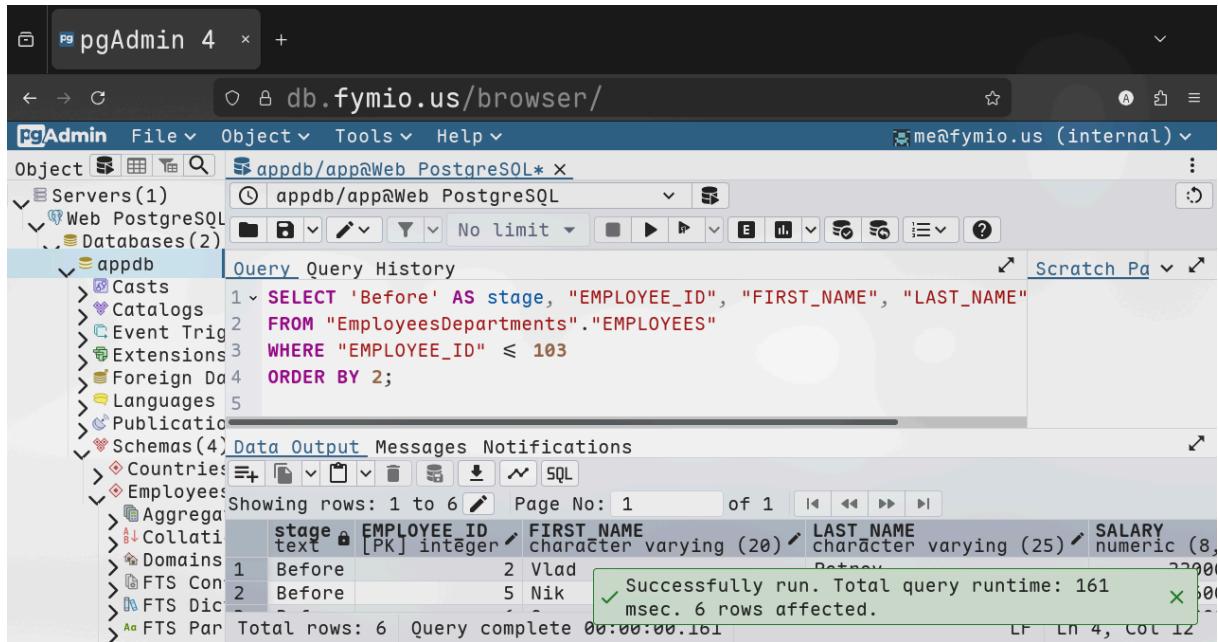
```

SELECT 'Before' AS stage, "EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME", "SALARY"
FROM "EmployeesDepartments"."EMPLOYEES"
WHERE "EMPLOYEE_ID" <= 103
ORDER BY 2;

```

Скрипт. 7: Выборка исходных значений сотрудников до начала транзакции.

Получает текущие данные сотрудников с ID ≤ 103 для фиксации их состояния перед изменениями. (Рис. 8)



The screenshot shows the pgAdmin 4 interface with the following details:

- Header:** pgAdmin 4, db.fymio.us/browser/
- Toolbar:** File, Object, Tools, Help
- Servers:** appdb/app@Web PostgreSQL
- Databases:** appdb
- Object List:** Casts, Catalogs, Event Trig, Extensions, Foreign Da, Languages, Publication, Schemas (4), Countries, Employees.
- Query Editor:** Contains the SQL script:

```

1 SELECT 'Before' AS stage, "EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME"
2 FROM "EmployeesDepartments"."EMPLOYEES"
3 WHERE "EMPLOYEE_ID" <= 103
4 ORDER BY 2;

```
- Result Table:** Shows the output of the query with 6 rows:

stage	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
1	Before	2	Vlad	22000
2	Before	5	Nik	20000
				Successfully run. Total query runtime: 161 msec. 6 rows affected.
				Total rows: 6 Query complete 00:00:00.161

Рис. 8: Результат выполнения Скрипта. 7

```
BEGIN;
```

Скрипт. 8: Начало транзакции.

Открывает новую транзакцию, в пределах которой будут выполняться дальнейшие изменения. (Рис. 9)

The screenshot shows the pgAdmin 4 interface. The title bar reads "pgAdmin 4" and the address bar shows "db.fymio.us/browser/". The left sidebar lists "Servers(1)" and "Web PostgreSQL" under "Object", and "Databases(2)" under "Object". The main pane shows the "Query" tab with the following content:

```
1 BEGIN;
```

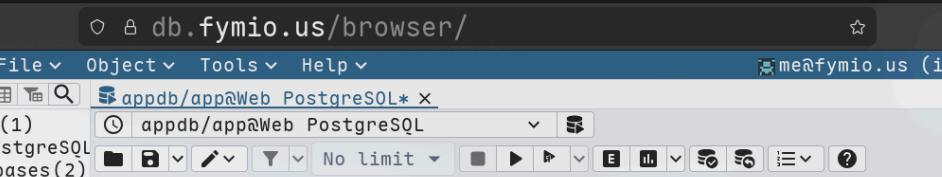
Below the query, a message states: "WARNING: there is already a transaction in progress". The query results table shows "Query returned successfully in 178 msec." and the status bar at the bottom indicates "Total rows: 1 Query complete 00:00:00.178".

Рис. 9: Результат выполнения Скрипт. 8

```
UPDATE "EmployeesDepartments"."EMPLOYEES"  
SET "SALARY" = "SALARY" * 1.1  
WHERE "EMPLOYEE_ID" <= 103;
```

Скрипт. 9: Повышение зарплаты сотрудникам с ID ≤ 103 на 10%

Временно обновляет зарплаты выбранных сотрудников внутри текущей транзакции, увеличивая их значения на 10%. (Рис. 10)



The screenshot shows the pgAdmin 4 interface with the following details:

- Header:** pgAdmin 4, db.fymio.us/browser/ (me@fymio.us (internal)), File, Object, Tools, Help.
- Servers:** Servers(1) - appdb/app@Web PostgreSQL.
- Databases:** Databases(2) - appdb (selected), scratch.
- Query History:** appdb
- Query:**

```
1 UPDATE "EmployeesDepartments"."EMPLOYEES"
2 SET "SALARY" = "SALARY" * 1.1
3 WHERE "EMPLOYEE_ID" <= 103;
```
- Output:** Data, Output, Messages, Notifications. The Messages tab shows "UPDATE 6".
- Message:** Query returned successfully in 126 msec.
- Statistics:** Total rows: 6, Query complete 00:00:00.126, LF, LN 4, Col 1.

Рис. 10: Результат выполнения Скрипт. 9

```

SELECT 'Within transaction' AS stage, "EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME",
"SALARY"
FROM "EmployeesDepartments"."EMPLOYEES"
WHERE "EMPLOYEE_ID" <= 103
ORDER BY 2;

```

Скрипт. 10: Просмотр изменённых данных внутри транзакции

Выводит обновлённые зарплаты сотрудников с $ID \leq 103$ до выполнения `ROLLBACK`, показывая состояние данных внутри незавершённой транзакции. (Рис. 11)

```

SELECT 'Within transaction' AS stage, "EMPLOYEE_ID", "FIRST_NAME",
"LAST_NAME"
FROM "EmployeesDepartments"."EMPLOYEES"
WHERE "EMPLOYEE_ID" <= 103
ORDER BY 2;

```

Рис. 11: Результат выполнения Скрипт. 10

`ROLLBACK;`

Скрипт. 11: Откат транзакции

Отменяет все изменения, сделанные в текущей транзакции, возвращая данные в состояние до `BEGIN`. (Рис. 12)

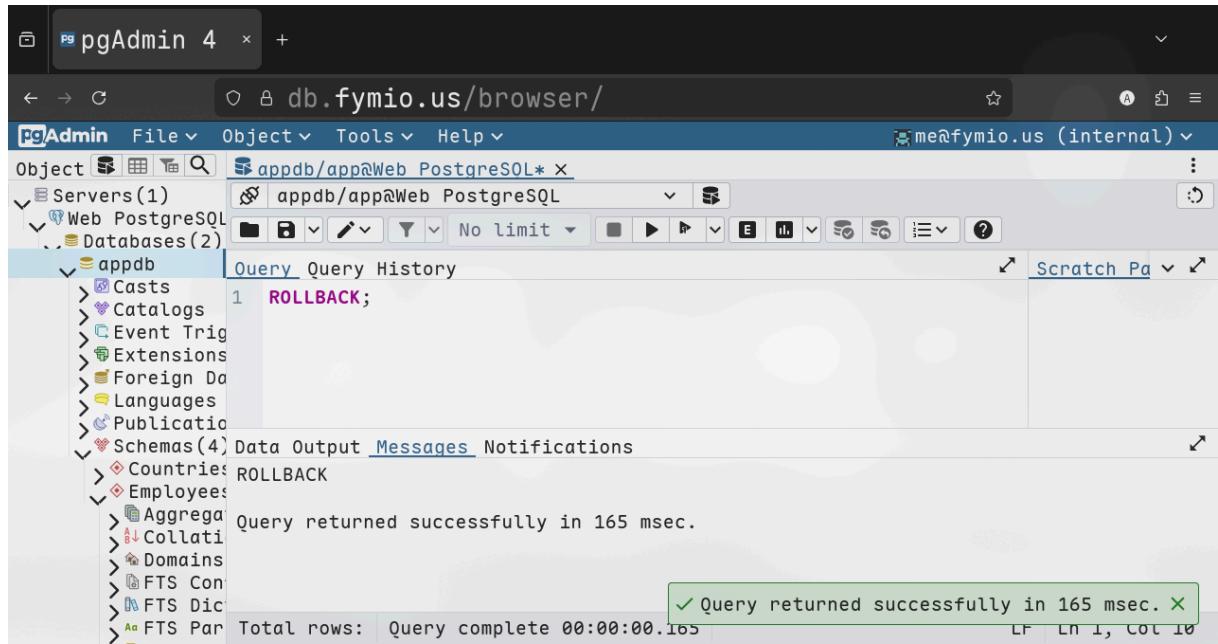


Рис. 12: Результат выполнения Скрипт. 11

```
SELECT 'After' AS stage, "EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME", "SALARY"
FROM "EmployeesDepartments"."EMPLOYEES"
WHERE "EMPLOYEE_ID" <= 103
ORDER BY 2;
```

Скрипт. 12: Проверка данных после отката транзакции

Выводит состояние сотрудников с $ID \leq 103$ после выполнения ROLLBACK, подтверждая, что изменения не были сохранены. (Рис. 13)

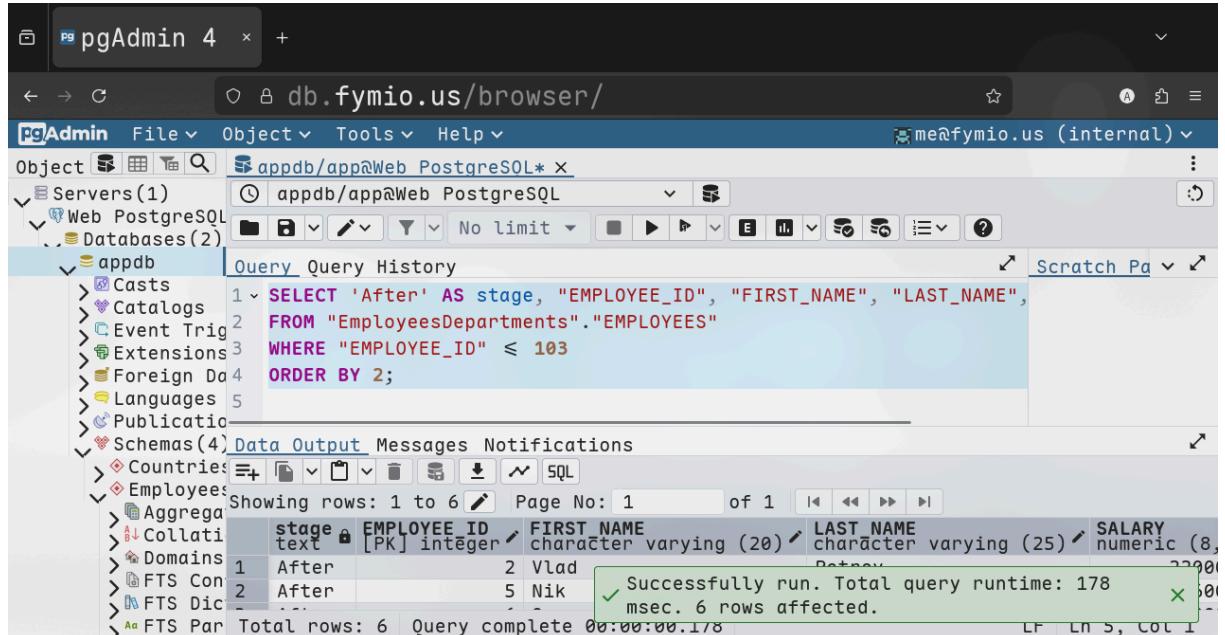


Рис. 13: Результат выполнения Скрипт. 12

```
BEGIN;
```

Скрипт. 13: Начало новой транзакции.

Открывает транзакционный блок, внутри которого будут выполняться операции с возможностью частичного отката через `SAVEPOINT`. (Рис. 14)

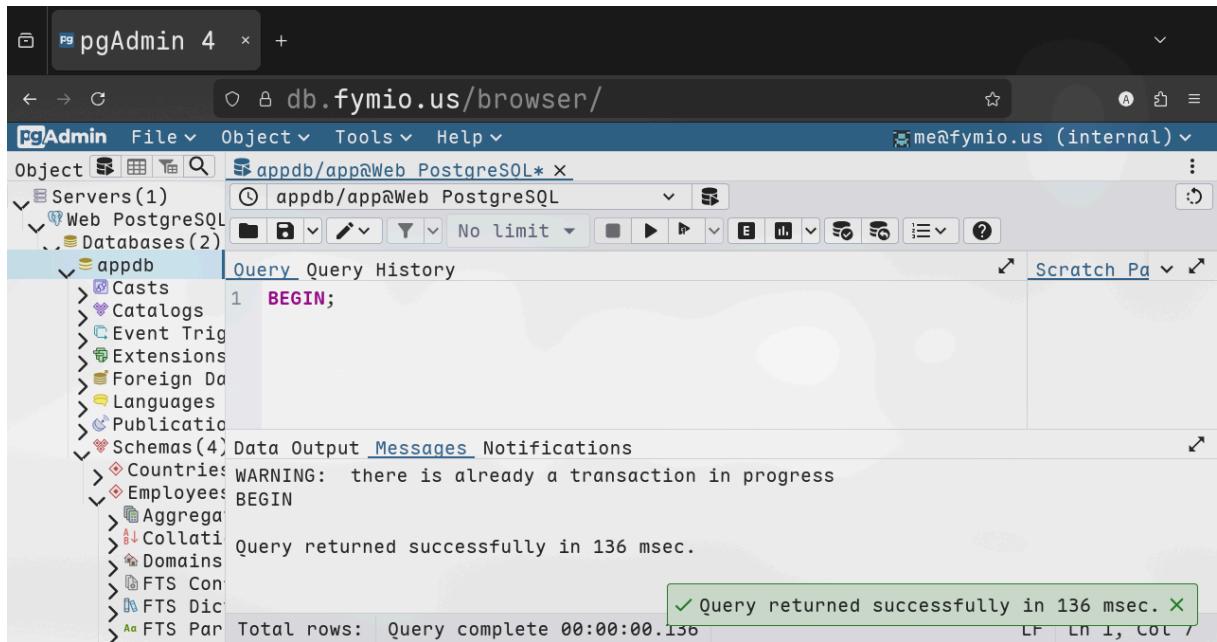


Рис. 14: Результат выполнения Скрипта. 13

```
SELECT 'Trans started' AS stage, "EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME",  
"SALARY"  
FROM "EmployeesDepartments"."EMPLOYEES"  
WHERE "EMPLOYEE_ID" = 104;
```

Скрипт. 14: Выборка исходных данных сотрудника перед работой с savepoint

Получает текущее состояние сотрудника с ID 104 в момент начала транзакции, чтобы зафиксировать отправную точку перед изменениями. (Рис. 15)

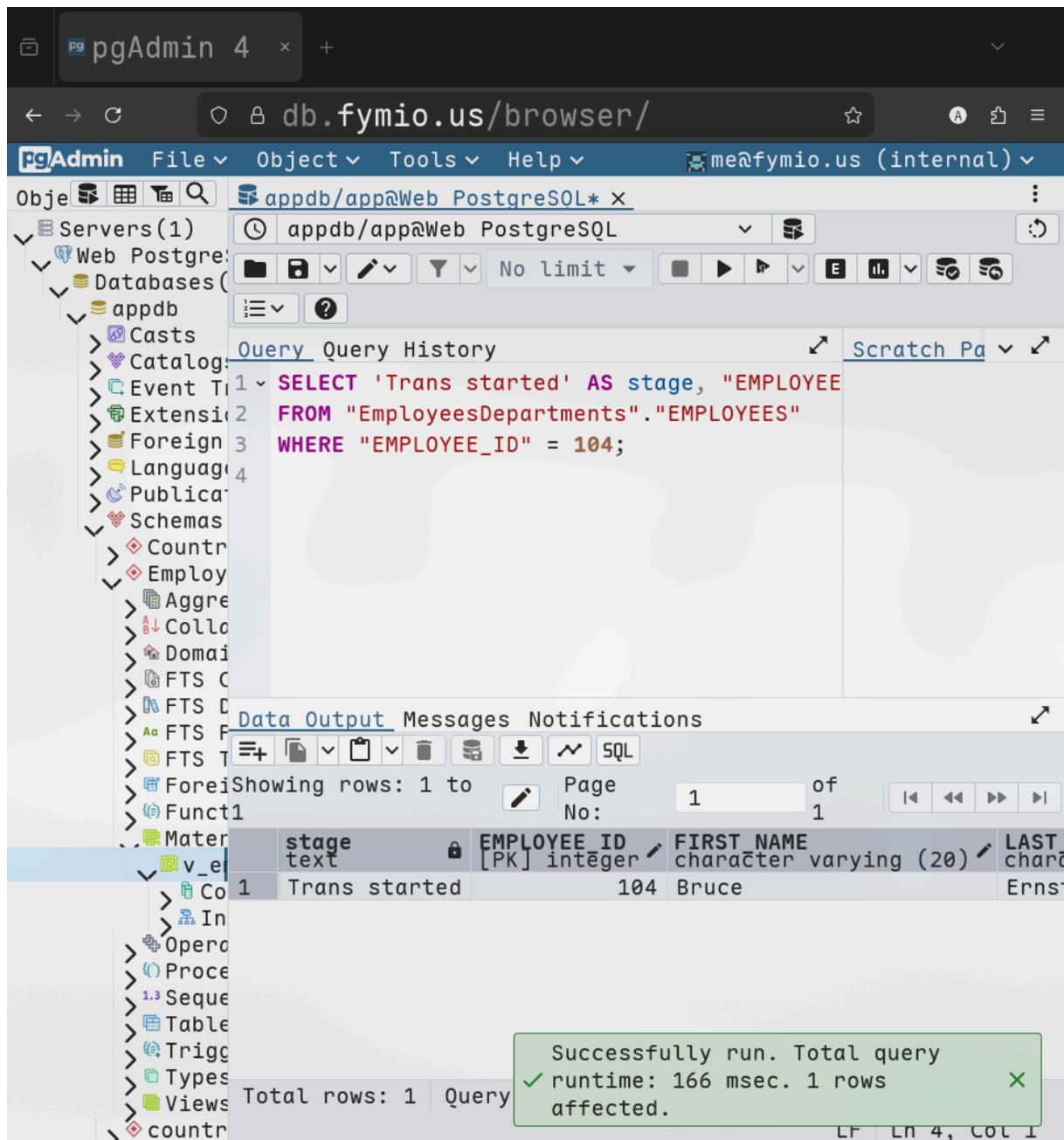


Рис. 15: Результат выполнения Скрипта. 14

```
SAVEPOINT sp1;
```

Скрипт. 15: Создание первой точки сохранения

Устанавливает savepoint sp1 внутри текущей транзакции, позволяя откатить изменения до этой точки при необходимости. (Рис. 16)

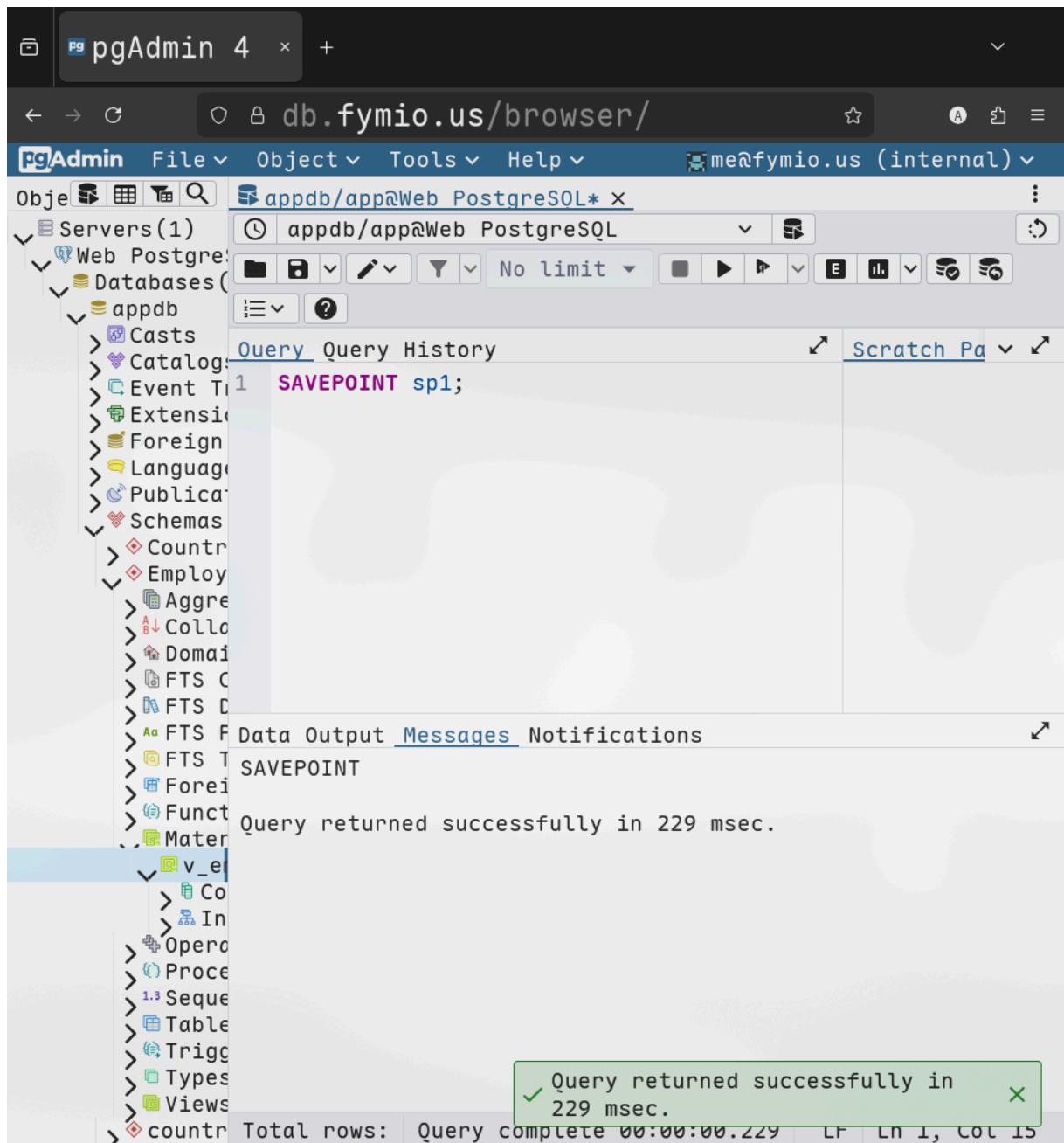


Рис. 16: Результат выполнения Скрипт. 15

```

UPDATE "EmployeesDepartments"."EMPLOYEES"
SET "SALARY" = 3000
WHERE "EMPLOYEE_ID" = 104;

```

Скрипт. 16: Установка новой зарплаты сотруднику после sp1

Обновляет поле SALARY сотрудника с ID 104 до 3000 внутри текущей транзакции, после создания первой точки сохранения sp1. (Рис. 17)

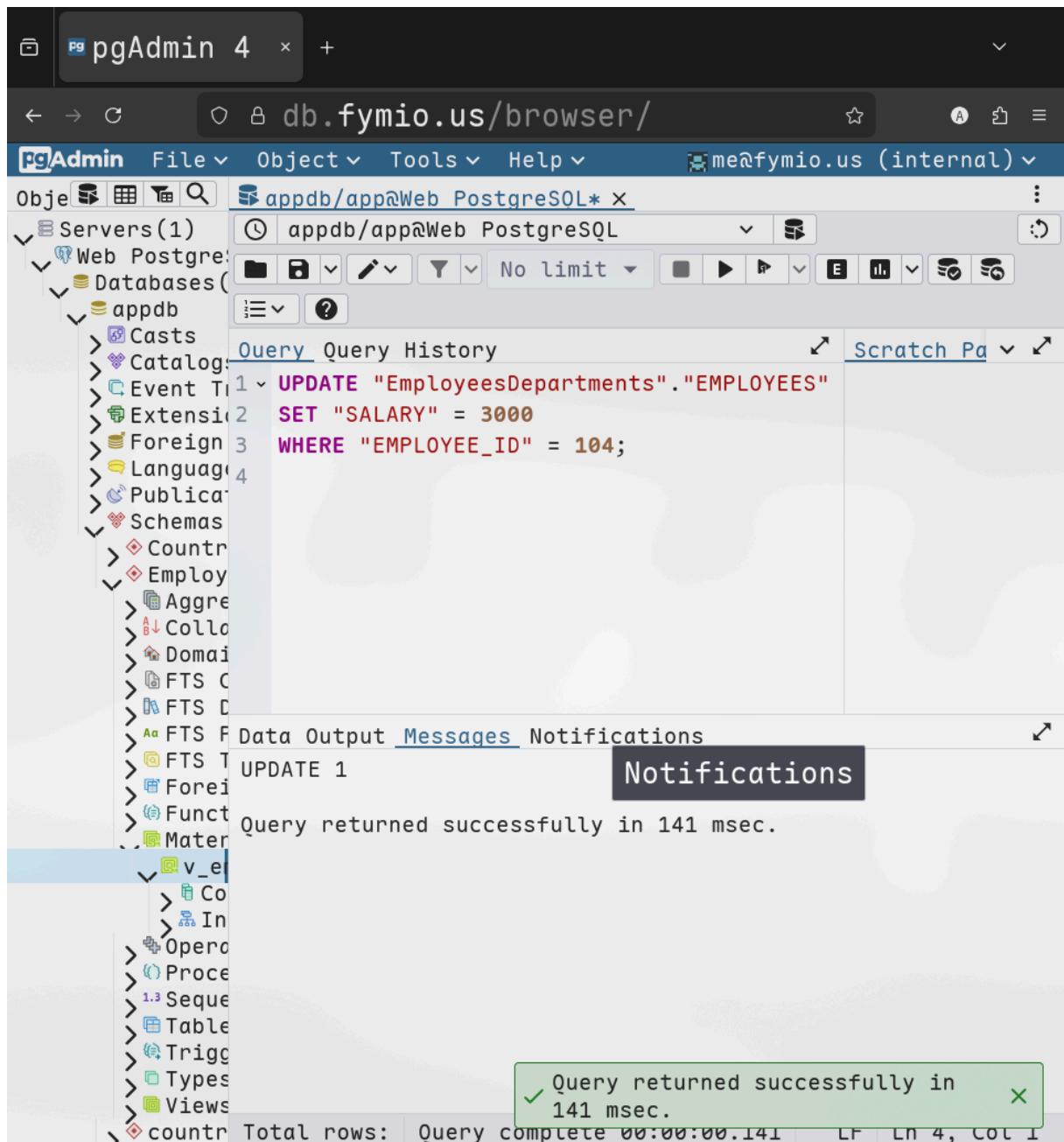


Рис. 17: Результат выполнения Скрипт. 16

```
SELECT 'After SAVEP sp1' AS stage, "EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME",
"SALARY"
FROM "EmployeesDepartments"."EMPLOYEES"
WHERE "EMPLOYEE_ID" = 104;
```

Скрипт. 17: Проверка данных после первой точки сохранения

Выводит состояние сотрудника с ID 104 после обновления зарплаты до 3000, показывая результат после savepoint sp1. (Рис. 18)

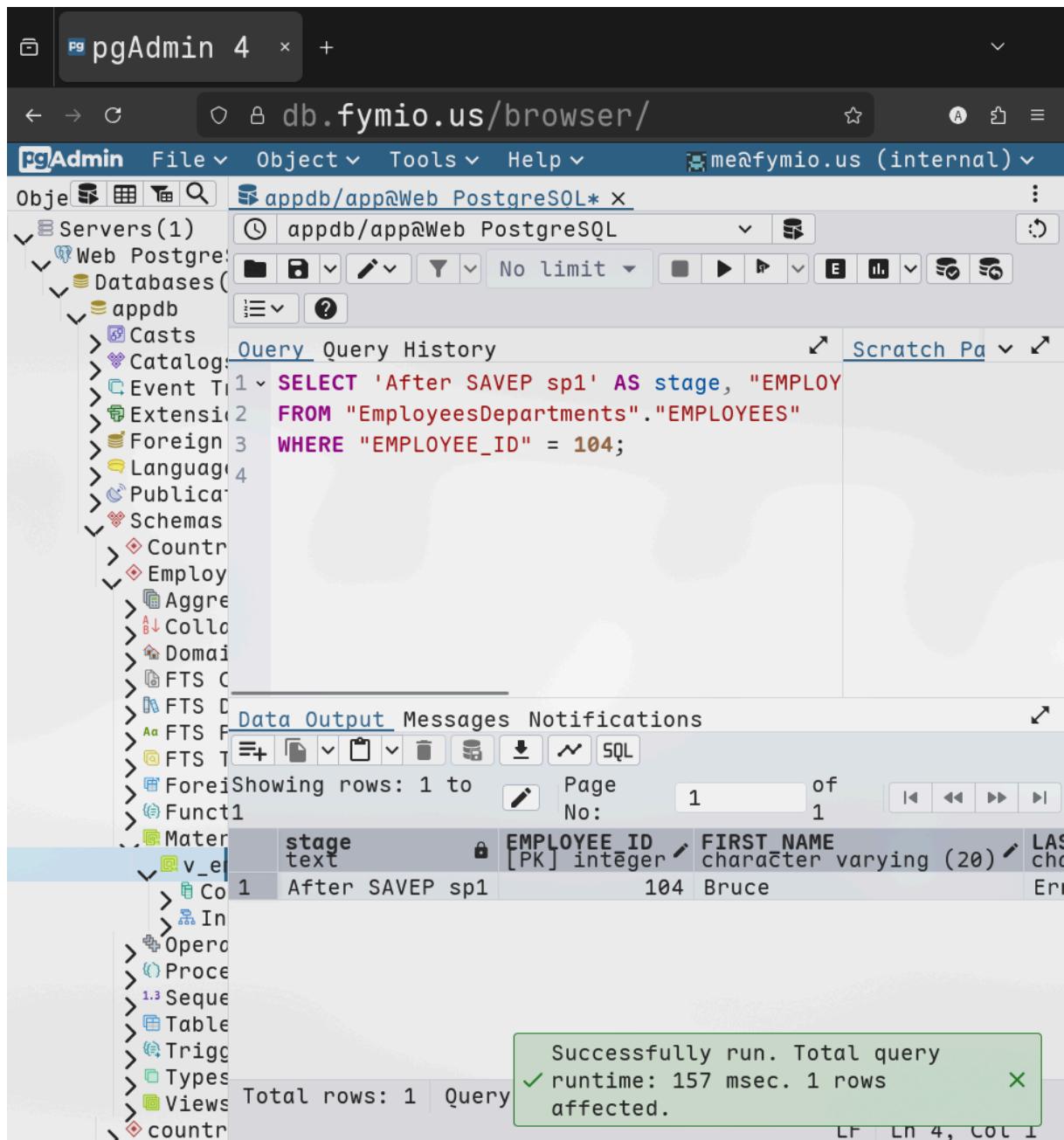


Рис. 18: Результат выполнения Скрипта. 17

```
SAVEPOINT sp2;
```

Скрипт. 18: Создание второй точки сохранения

Устанавливает savepoint sp2 внутри текущей транзакции, чтобы при необходимости можно было откатить изменения только до этой новой точки. (Рис. 19)

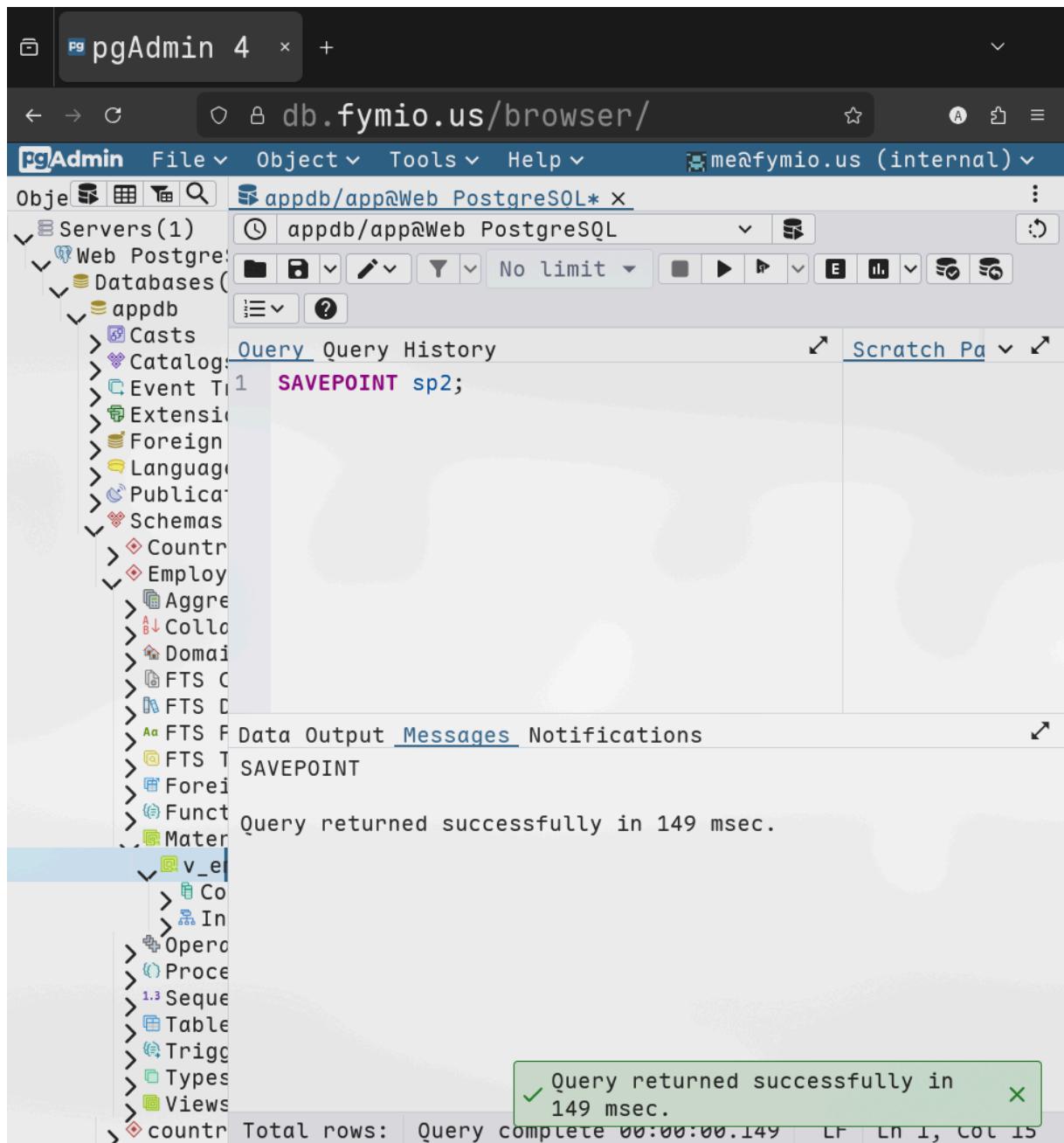


Рис. 19: Результат выполнения Скрипт. 18

```
UPDATE "EmployeesDepartments"."EMPLOYEES"
SET "SALARY" = 10000
WHERE "EMPLOYEE_ID" = 104;
```

Скрипт. 19: Установка новой зарплаты сотруднику после sp2

Обновляет поле SALARY сотрудника с ID 104 до 10000 внутри текущей транзакции, после создания второй точки сохранения sp2. (Рис. 20)

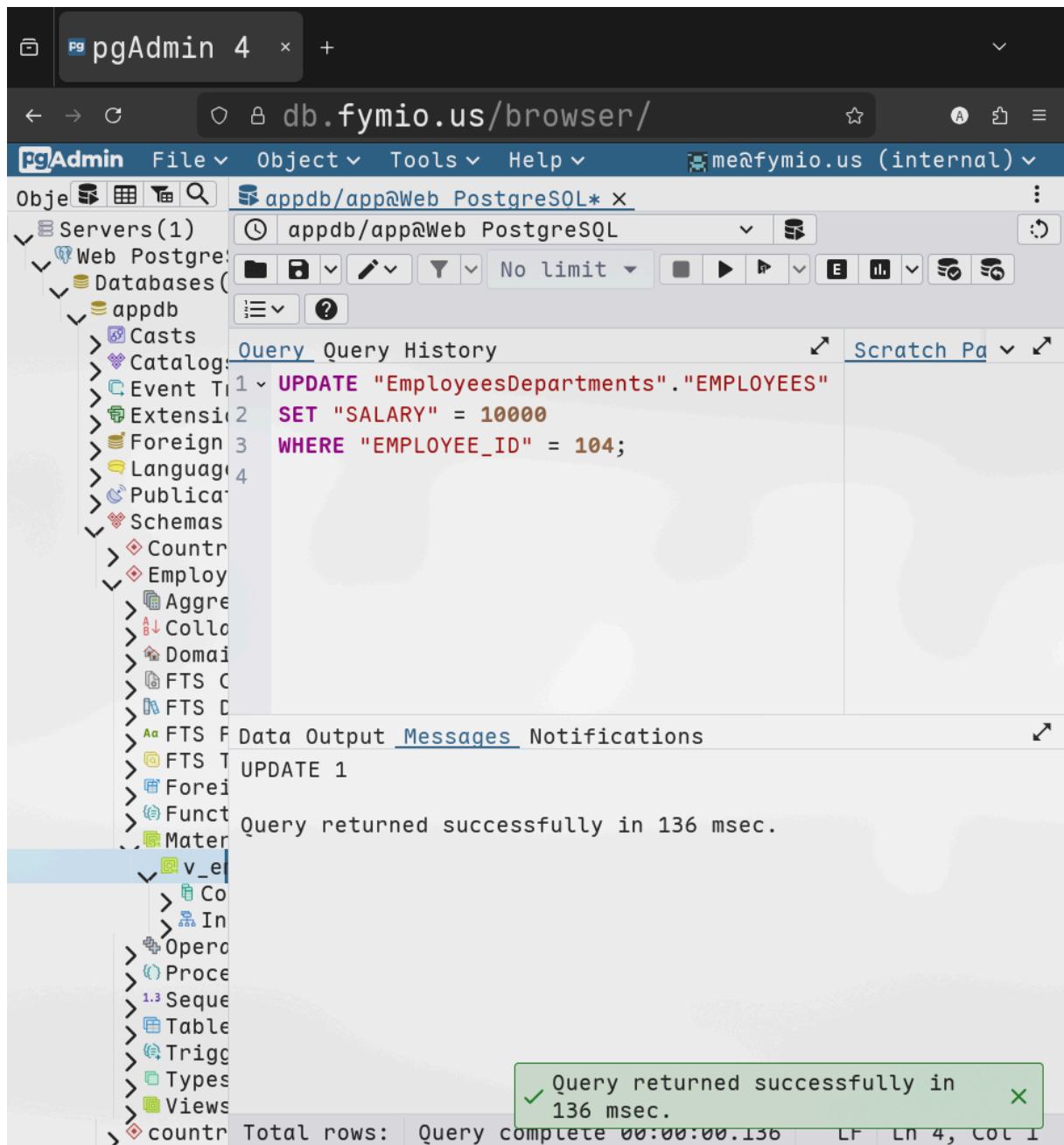


Рис. 20: Результат выполнения Скрипт. 19

```

SELECT 'After SAVEP sp2' AS stage, "EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME",
"SALARY"
FROM "EmployeesDepartments"."EMPLOYEES"
WHERE "EMPLOYEE_ID" = 104;

```

Скрипт. 20: Проверка данных после второй точки сохранения

Выводит состояние сотрудника с ID 104 после обновления зарплаты до 10000, показывая результат после savepoint sp2. (Рис. 21)

```

SELECT 'After SAVEP sp2' AS stage, "EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME"
FROM "EmployeesDepartments"."EMPLOYEES"
WHERE "EMPLOYEE_ID" = 104;
  
```

stage	EMPLOYEE_ID	FIRST_NAME	LAST_NAME
After SAVEP sp2	104	Bruce	Ernest

Successfully run. Total query runtime: 165 msec. 1 rows affected.

Рис. 21: Результат выполнения Скрипт. 20

ROLLBACK TO SAVEPOINT sp2;

Скрипт. 21: Откат к второй точке сохранения

Отменяет все изменения, сделанные после savepoint sp2, возвращая данные к состоянию на момент его создания. (Рис. 22)

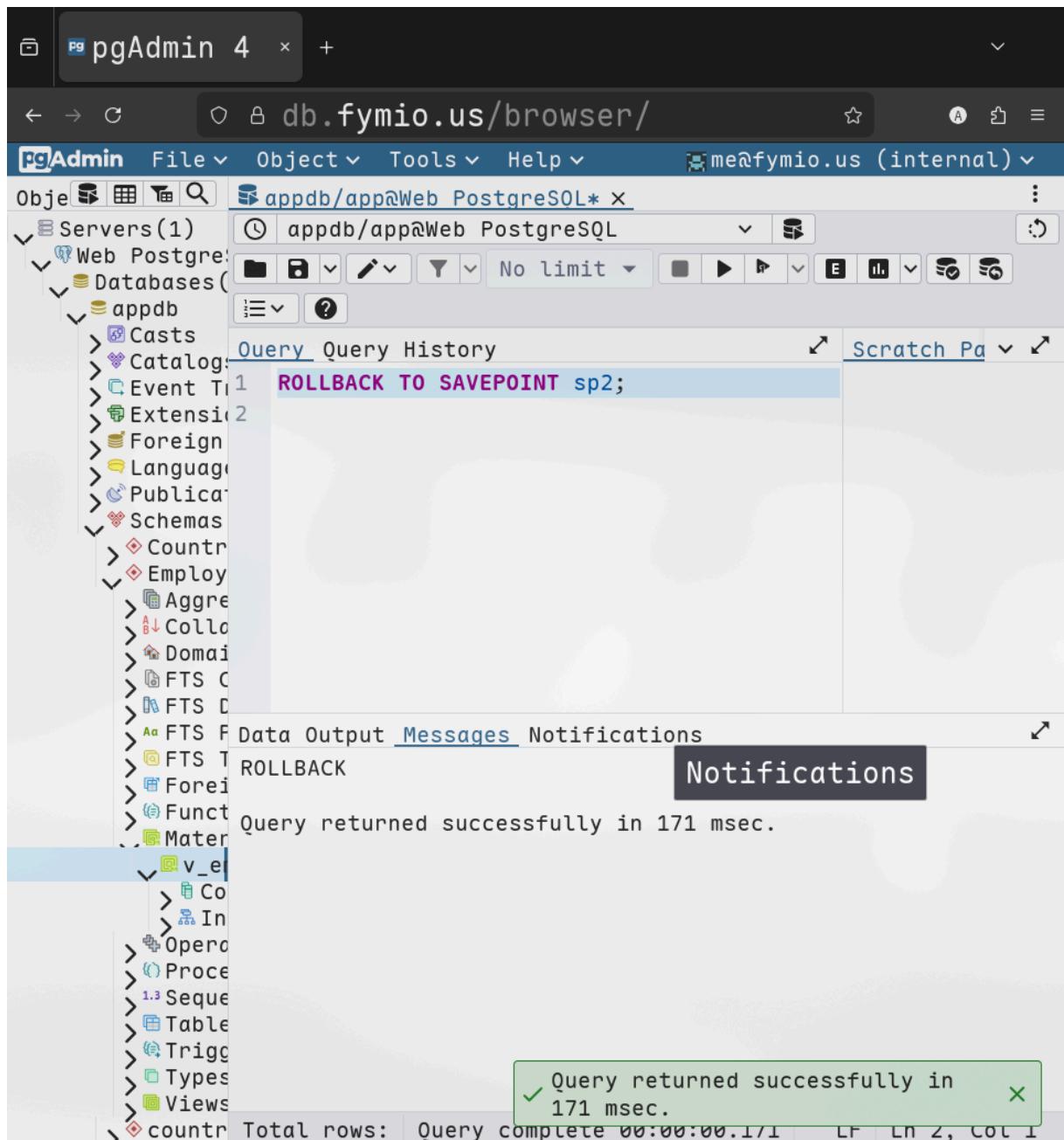


Рис. 22: Результат выполнения Скрипт. 21

```
SELECT 'After ROLLBACK TO SAVEPOINT sp2' AS stage, "EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME", "SALARY"
FROM "EmployeesDepartments"."EMPLOYEES"
WHERE "EMPLOYEE_ID" = 104;
```

Скрипт. 22: Проверка данных после отката к sp2

Выводит состояние сотрудника с ID 104 после отката к savepoint sp2, показывая, что изменения после этой точки отменены. (Рис. 23)

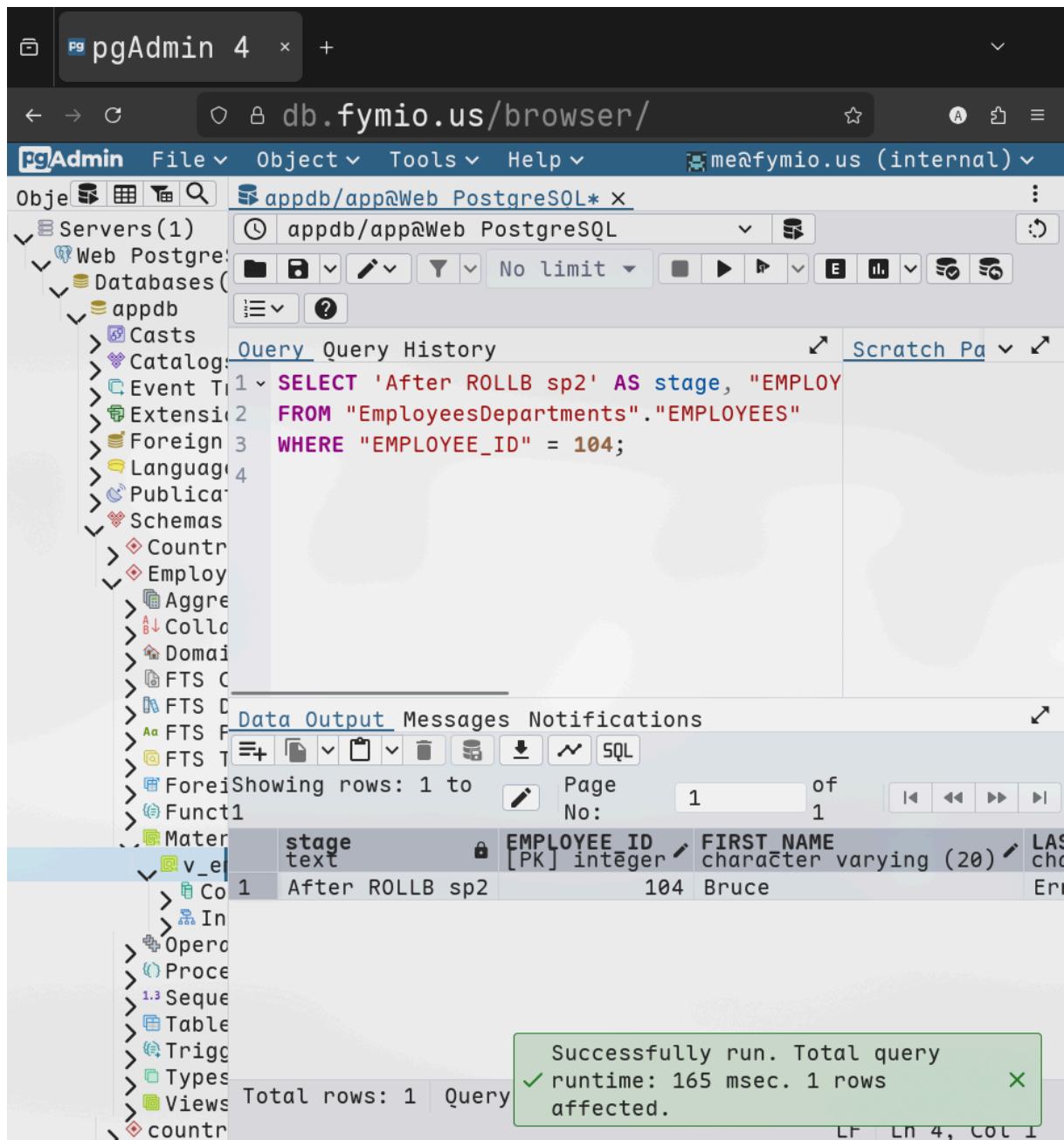


Рис. 23: Результат выполнения Скрипт. 22

```

UPDATE "EmployeesDepartments"."EMPLOYEES"
SET "SALARY" = "SALARY" + 100
WHERE "EMPLOYEE_ID" = 104;
      
```

Скрипт. 23: Увеличение зарплаты сотруднику после отката к sp2

Добавляет 100 к текущей зарплате сотрудника с ID 104 внутри транзакции после отката к savepoint sp2. (Рис. 24)

The screenshot shows the pgAdmin 4 interface with the following details:

- Top Bar:** pgAdmin 4, db.fymio.us/browser/, me@fymio.us (internal).
- Object Browser:** Shows the database structure under "appdb/app@Web PostgreSQL".
- Query Editor:** Contains the following SQL script:


```
UPDATE "EmployeesDepartments"."EMPLOYEES"
SET "SALARY" = "SALARY" + 100
WHERE "EMPLOYEE_ID" = 104;
```
- Result Table:** Shows the result of the update query:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME
104	Bruce	Ernest
- Message Bar:** Total rows: 1 | Query complete 00:00:00.165 | LF | Ln 4, Col 1

Рис. 24: Результат выполнения Скрипт. 23

COMMIT;

Скрипт. 24: Фиксация транзакции после изменений с savepoint

Подтверждает все изменения, выполненные в транзакции, включая корректировку зарплаты после отката к savepoint, делая их постоянными и видимыми для других сессий. (Рис. 25)

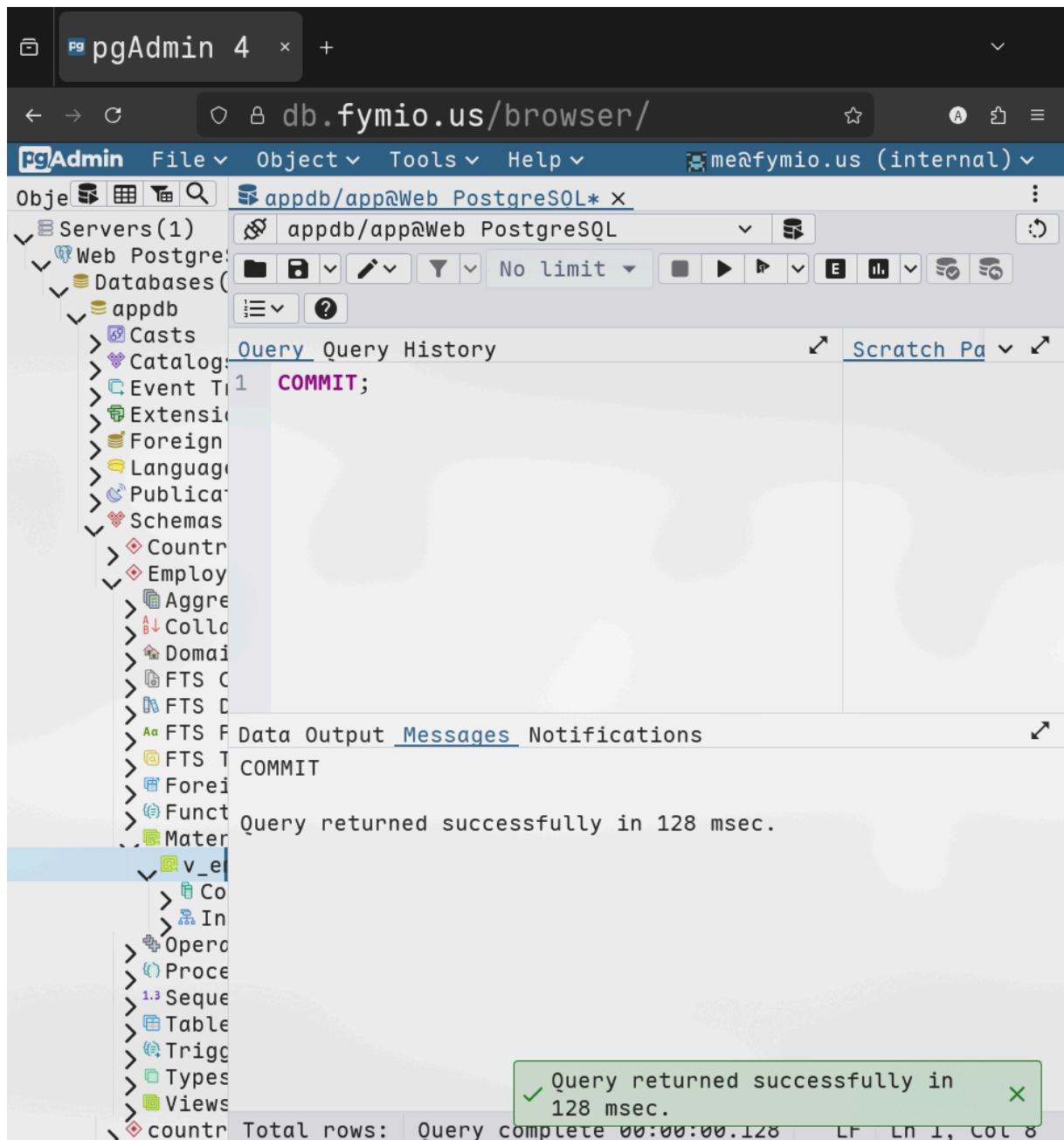


Рис. 25: Результат выполнения Скрипт. 24

```
SELECT 'After trans' AS stage, "EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME", "SALARY"
FROM "EmployeesDepartments"."EMPLOYEES"
WHERE "EMPLOYEE_ID" = 104;
```

Скрипт. 25: Проверка данных после завершения транзакции

Выводит окончательное состояние сотрудника с ID 104 после выполнения всех операций и фиксации транзакции, включая откат к savepoint и последующее увеличение зарплаты. (Рис. 26)

```

pgAdmin 4 × +
← → C ⌂ db.fymio.us/browser/
pgAdmin File Object Tools Help
me@fymio.us (internal)
Object Servers (1) appdb/app@Web PostgreSQL*
Web PostgreSQL appdb/app@Web PostgreSQL
Databases (1) appdb
Casts Catalog
Event Types (1) SELECT 'After trans' AS stage, "EMPLOYEE_ID"
Extensions (2) FROM "EmployeesDepartments"."EMPLOYEES"
Foreign (3) WHERE "EMPLOYEE_ID" = 104;
Languages (4)
Publications (4)
Schemas (1)
Countries (1)
Employees (1)
Aggregates (1)
Collections (1)
Domains (1)
FTS Configuration (1)
FTS Data (1)
FTS Functions (1)
Data Output Messages Notifications
Foreign Keys (1) Showing rows: 1 total
Functions (1) Graph Visualiser 1 of 1
Materialized Views (1) 1
Views (1) Total rows: 1 Query
Tables (1) 1
Triggers (1)
Types (1)
Views (1) Total rows: 1 Query
Counters (1)

```

Successfully run. Total query runtime: 223 msec. 1 rows affected.

Рис. 26: Результат выполнения Скрипт. 25

Задание 2. Поиск и обнаружение блокировок

Я создал 3 вкладки в pgadmin: "Connection1", "Connection2", "Connection3". (Рис. 27)

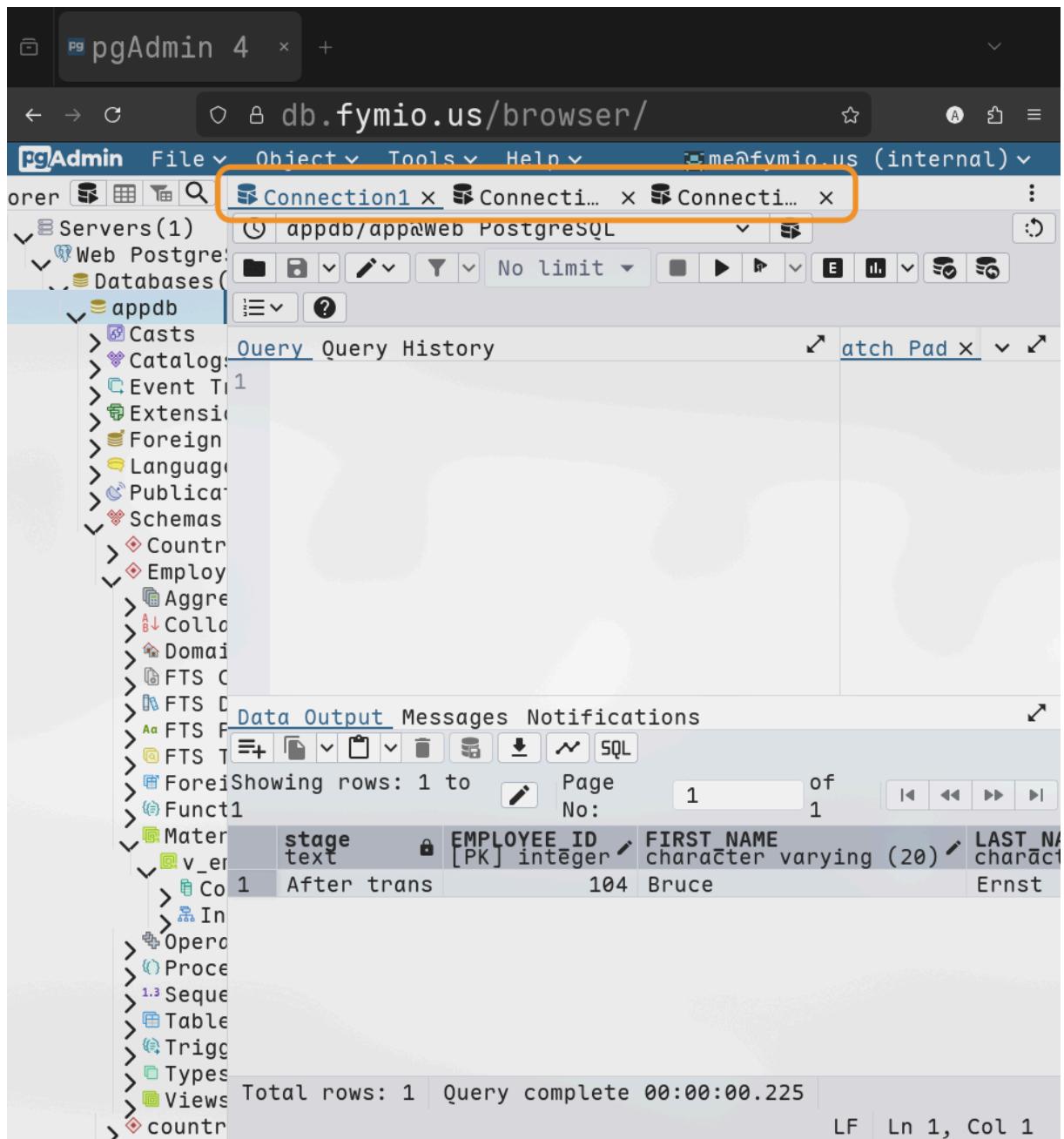


Рис. 27: Вкладки “Connections”

Во вкладке “Connection1” я выполнил Скрипт. 26 (Рис. 28)

```

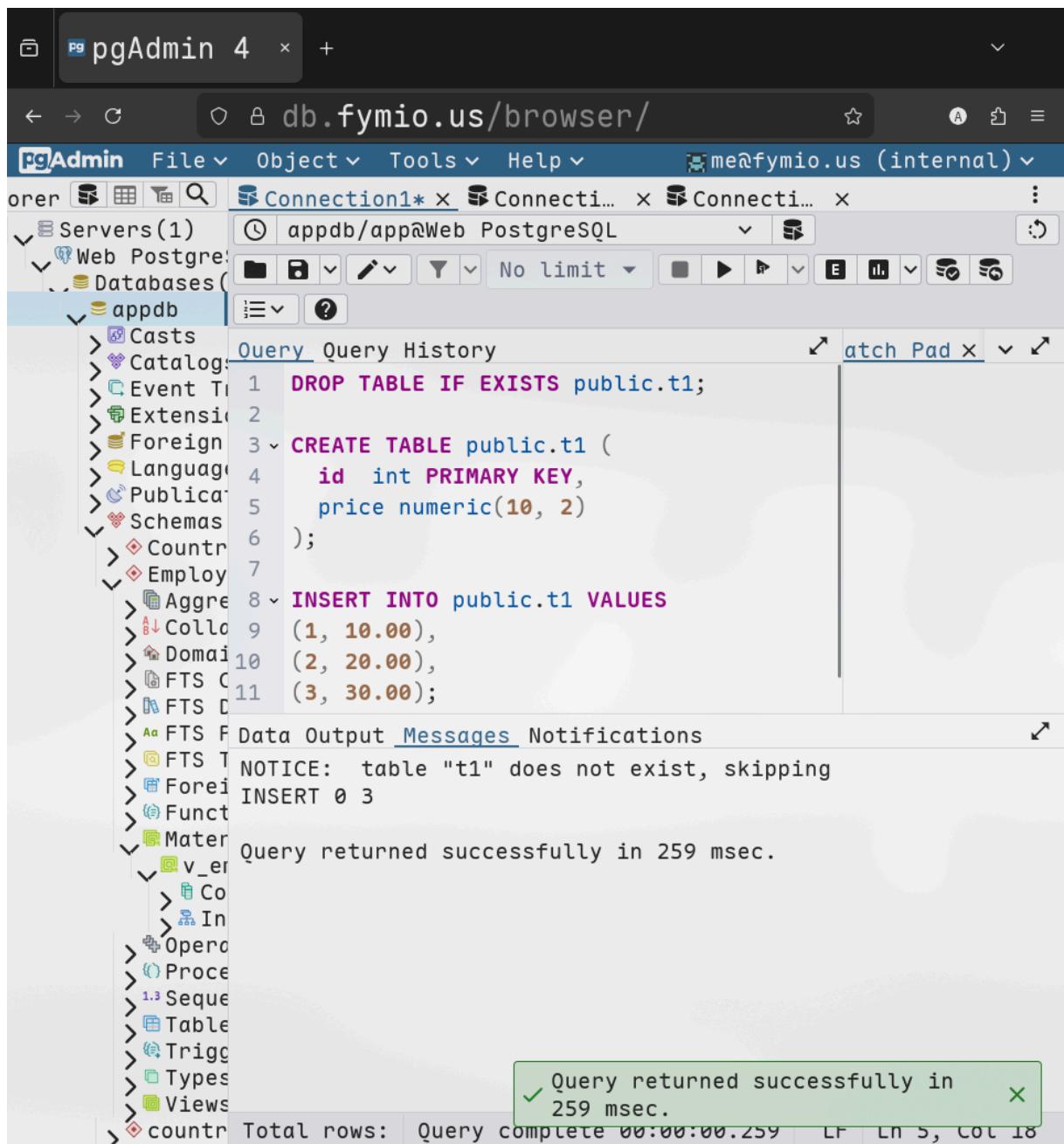
DROP TABLE IF EXISTS public.t1;

CREATE TABLE public.t1 (
    id int PRIMARY KEY,
    price numeric(10, 2)
);

INSERT INTO public.t1 VALUES
(1, 10.00),
(2, 20.00),
(3, 30.00);

```

Скрипт. 26: Создание тестовой таблицы с тремя записями.



The screenshot shows the pgAdmin 4 interface with a connection to 'appdb/app@Web PostgreSQL'. The left sidebar shows the database structure under 'appdb', including 'Schemas' (public), 'Tables' (t1), and other objects like 'Functions', 'Types', and 'Views'. The main query editor window contains the following SQL script:

```

DROP TABLE IF EXISTS public.t1;

CREATE TABLE public.t1 (
    id int PRIMARY KEY,
    price numeric(10, 2)
);

INSERT INTO public.t1 VALUES
(1, 10.00),
(2, 20.00),
(3, 30.00);

```

The script is being executed, and the status bar at the bottom right indicates: 'Query returned successfully in 259 msec.' and 'Query complete 00:00:00.259 LF Ln 5, Col 18'.

Рис. 28: Выполнение Скрипта. 26

При помощи команды `SELECT current_database();` я проверил подключение к нужной базе. (Рис. 29)

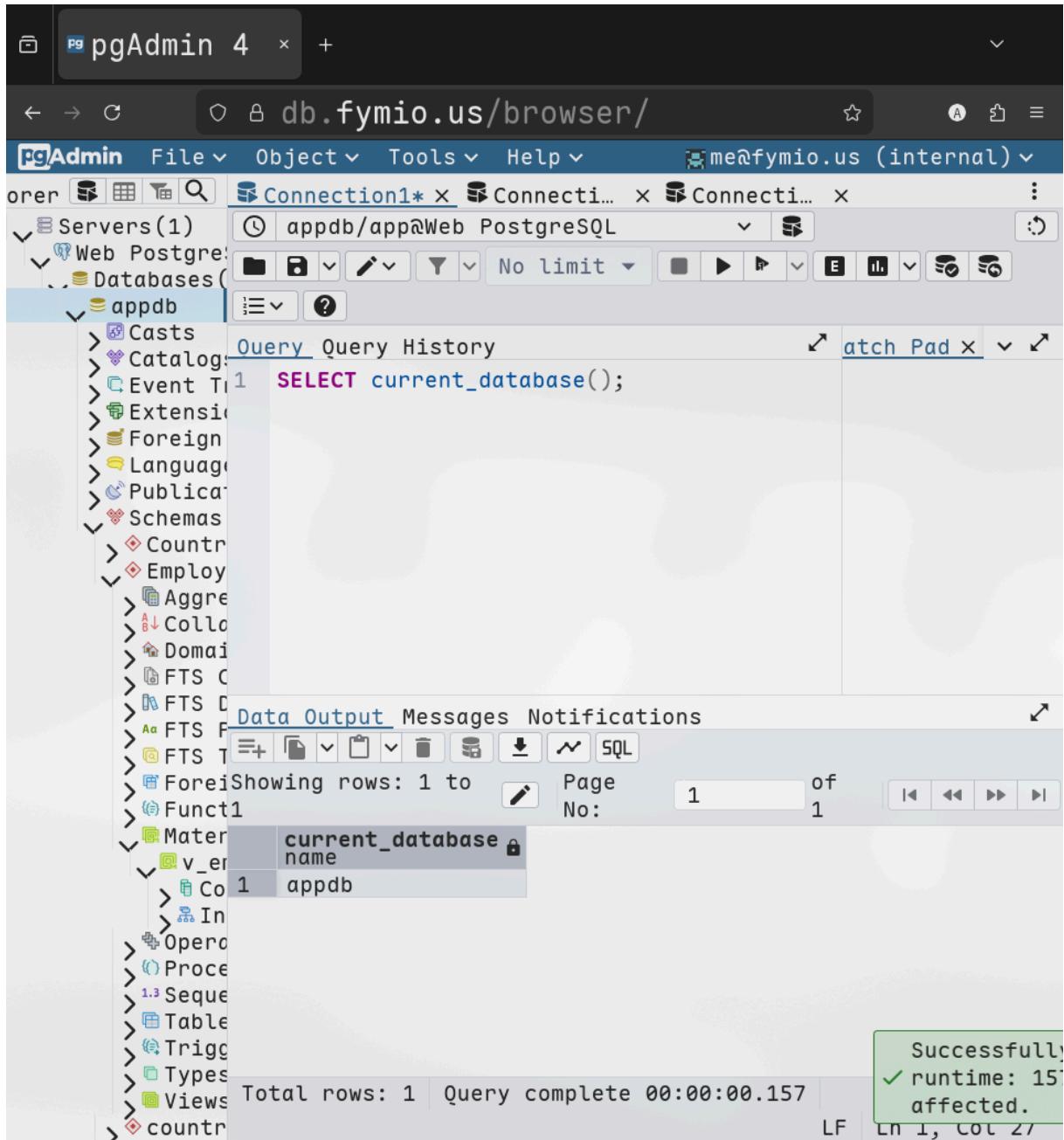


Рис. 29: Проверка подключения.

Затем я заблокировал первую строку при помощи Скрипта. 27 (Рис. 30)

```

BEGIN;

UPDATE public.t1
SET price = price + 1.00
WHERE id = 2;

SELECT 'Inside Connection 1' AS stage, id, price
FROM public.t1
WHERE id = 2;

```

Скрипт. 27: Блокировка строки.

Connection1 установит RowExclusiveLock на строку с id = 2 и удерживает его до `COMMIT` или `ROLLBACK`.

Пока транзакция открыта, блокировка активна.

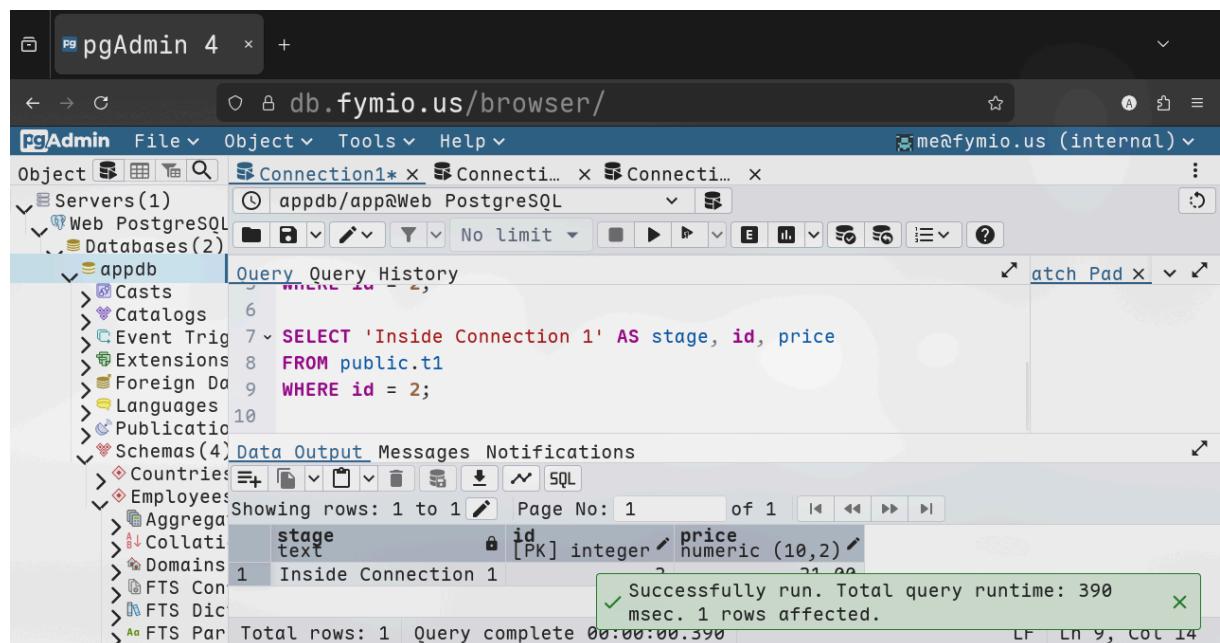


Рис. 30: Выполнение Скрипта. 27

Затем, из “Connection2” я попытался обновить ту же строку с помощью Скрипта. 28 (Рис. 31)

```

BEGIN;

SELECT id, price
FROM public.t1
WHERE id = 2;

UPDATE public.t1
SET price = 0
WHERE id = 2;

SELECT id, price
FROM public.t1
WHERE id = 2;

```

Скрипт. 28: Обновление строки из “Connection2”.

```

pgAdmin 4
db.fymio.us/browser/
Object File Object Tools Help
Servers(1) Connection2* Connection3*
Web PostgreSQL Connection2* Connection3*
Databases(2) appdb
Query Query History
1 BEGIN;
2
3 SELECT id, price
4 FROM public.t1
5 WHERE id = 2;
Schemas(4) Data Output Messages Notifications
Countries: ERROR: relation "public.t1" does not exist
Employees: LINE 4: FROM public.t1
Aggrega
Collati
Domains: SQL state: 42P01
FTS Con: Character: 31
FTS Dic
FTS Par
Total rows: Query complete 00:00:00.236
Scratch Pa
LF Ln 4, Col 15

```

Рис. 31: Результат выполнения Скрипт. 28

Я получил ошибку, так как “Connection1” держит открытый `BEGIN` с `UPDATE` той же строки.

Затем, я проверил блокировки из “Connection3” при помощи Скрипт. 29. (Рис. 32)

```

SELECT pid,
       username AS username,
       application_name,
       state,
       wait_event_type,
       wait_event,
       query
  FROM pg_stat_activity
 WHERE datname = current_database()
 ORDER BY pid;

```

Скрипт. 29: Проверка блокировок из “Connection3”.

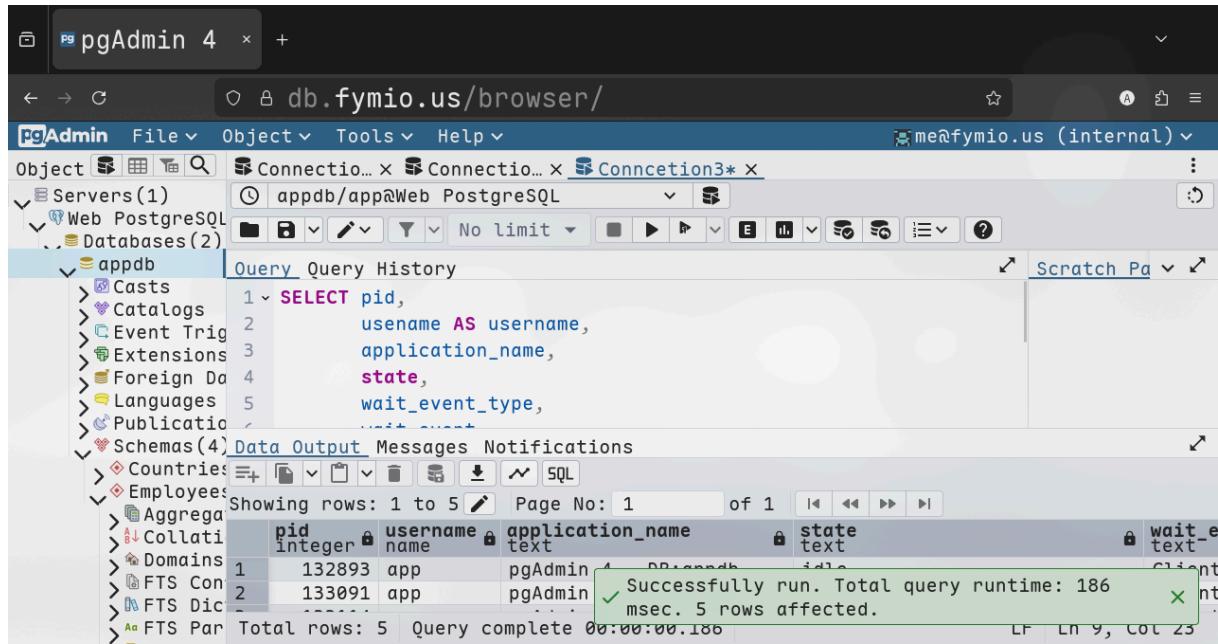


Рис. 32: Результат выполнения Скрипта. 29

- `pid` - идентификатор процесса.
- `state` - состояние сеанса.
- `wait_event_type` = Lock, если сеанс ждёт блокировку.
- `query` показывает последний выполняемый SQL.

Затем я провел детальный просмотр блокировок с `pg_locks` при помощи Скрипта. 30. (Рис. 33)

```

SELECT
    l.pid,
    a.application_name,
    a.state,
    l.locktype,
    l.relation::regclass AS locked_relation,
    l.page,
    l.tuple,
    l.virtualxid,
    l.transactionid,
    l.mode,
    l.granted
FROM pg_locks l
LEFT JOIN pg_stat_activity a ON a.pid = l.pid
WHERE a.datname = current_database()
ORDER BY l.pid, l.locktype;

```

Скрипт. 30: Просмотр блокировок с `pg_locks`.

```

13  FROM pg_locks l
14  LEFT JOIN pg_stat_activity a ON a.pid = l.pid
15  WHERE a.datname = current_database()
16  ORDER BY l.pid, l.locktype;
17

```

	pid	application_name	state	locktype	locked_relation
1	133114	pgAdmin 4 - CONN:7250780	idle in transaction object	rowlock	public
2	133114	pgAdmin 4 - CONN:7	Successfully run. Total query runtime: 211 msec.		

Рис. 33: Результат выполнения Скрипт. 30

- `mode = RowExclusiveLock` - монопольная блокировка на строку.
- `granted = true` - блокировка удерживается; `false` - ждёт получения.
- `locked_relation` показывает таблицу, к которой применена блокировка.
- `tuple` и `page` — идентификаторы строки и страницы.

Затем, при помощи `COMMIT` я завершил транзакцию в “Connection1”. (Рис. 34)

```

1 COMMIT;

```

Query returned successfully in 144 msec.
--

Рис. 34: Завершение транзакции.

Я повторил запросы Скрипт. 29 и Скрипт. 30 в “Connection3” чтобы проверить состояние после завершения транзакций. (Рис. 35, Рис. 36)

```

1 > SELECT pid,
2       username AS username,
3       application_name,
4       state,
5       wait_event_type,

```

	pid	username	application_name	state	wait_event_text
1	132893	app	pgAdmin 4 - CONN:8	idle	
2	133091	app	pgAdmin 4 - CONN:8	idle	

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

Successfully run. Total query runtime: 203 msec. 5 rows affected.

Total rows: 5 Query complete 00:00:00.203

Рис. 35: Результат выполнения Скрипт. 29.

```

1 > SELECT
2       l.pid,
3       a.application_name,
4       a.state,
5       l.locktype,

```

	pid	application_name	state	locktype	locked_relation
1	133116	pgAdmin 4 - CONN:8	idle	none	no authid and index
2	133116	pgAdmin 4 - CONN:8	idle	none	no authid and index

Showing rows: 1 to 41 Page No: 1 of 1

Successfully run. Total query runtime: 145 msec. 41 rows affected.

Total rows: 41 Query complete 00:00:00.145

Рис. 36: Результат выполнения Скрипт. 30.

Задание 3. Уровни изоляции READ UNCOMMITTED и READ COMMITTED

Я создал две сессии с названиями “Session1” и “Session2”. (Рис. 37)

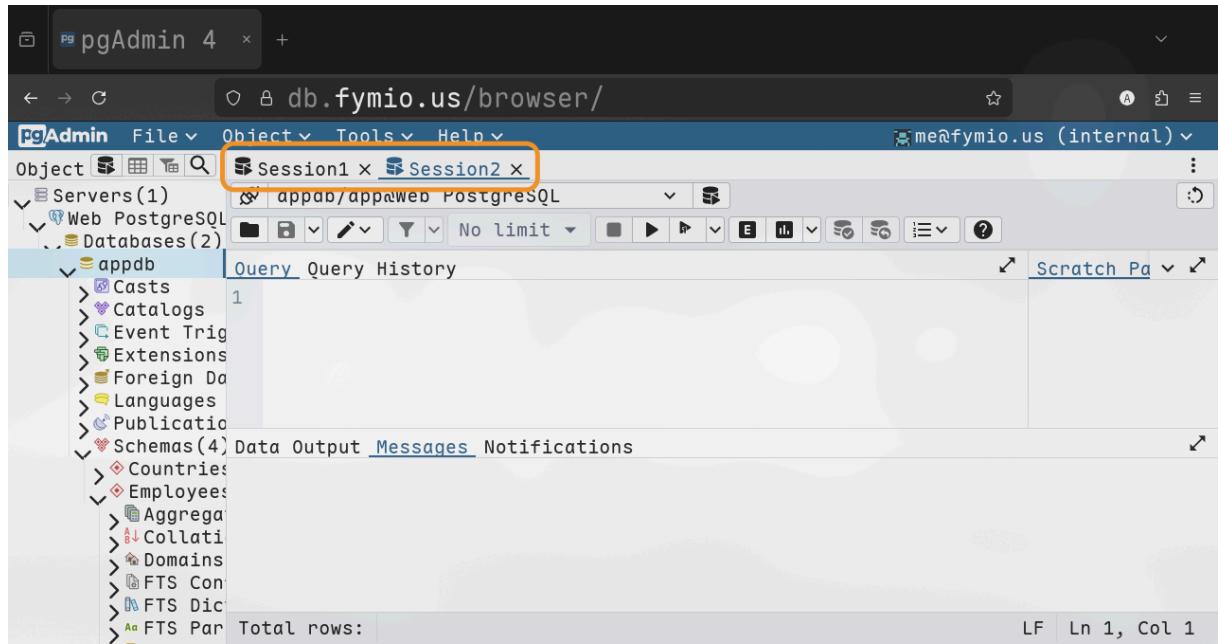


Рис. 37: Сессии “Sessions”.

В “Session1” я начал транзакцию и изменение данных при помощи Скрипта 31. (Рис. 38)

```
BEGIN;

SELECT id, price
FROM public.t1
WHERE id = 3;

UPDATE public.t1
SET price = 100
WHERE id = 3;

SELECT id, price
FROM public.t1
WHERE id = 3;
```

Скрипт. 31: Блокировка строки.

```

pgAdmin 4
db.fymio.us/browser/
Session1* Session2*
appdb/appWeb PostgreSQL
Query History
1 BEGIN;
2
3 SELECT id, price
4 FROM public.t1
5 WHERE id = 3;

```

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

	id	price
1	3	100.00

Successfully run. Total query runtime: 227 msec. 1 rows affected.

Total rows: 1 Query complete 00:00:00.227

Рис. 38: Результат выполнения Скрипт. 31

- На этом этапе строка с `id = 3` в Session1 заблокирована монопольно.
- Любые другие `UPDATE` этой строки в других сессиях будут ждать `COMMIT` или `ROLLBACK`.

Затем в “Session2” я установил уровень изоляции `READ COMMITTED` при помощи Скрипт. 32. (Рис. 39)

```

BEGIN ISOLATION LEVEL READ COMMITTED;

SELECT id, price
FROM public.t1
WHERE id = 3;

```

Скрипт. 32: Установка уровня `READ COMMITTED`.

```

pgAdmin 4
db.fymio.us/browser/
Session1* Session2*
appdb/appWeb PostgreSQL
Query History
1 BEGIN ISOLATION LEVEL READ COMMITTED;
2
3 SELECT id, price
4 FROM public.t1
5 WHERE id = 3;

```

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

	id	price
1	3	30.00

Successfully run. Total query runtime: 172 msec. 1 rows affected.

Total rows: 1 Query complete 00:00:00.172

Рис. 39: Результат выполнения Скрипт. 32

`READ COMMITTED` гарантирует, что каждая команда `SELECT` видит только зафиксированные изменения других транзакций.

Командой `COMMIT` в первой сессии я зафиксировал изменения. (Рис. 40)

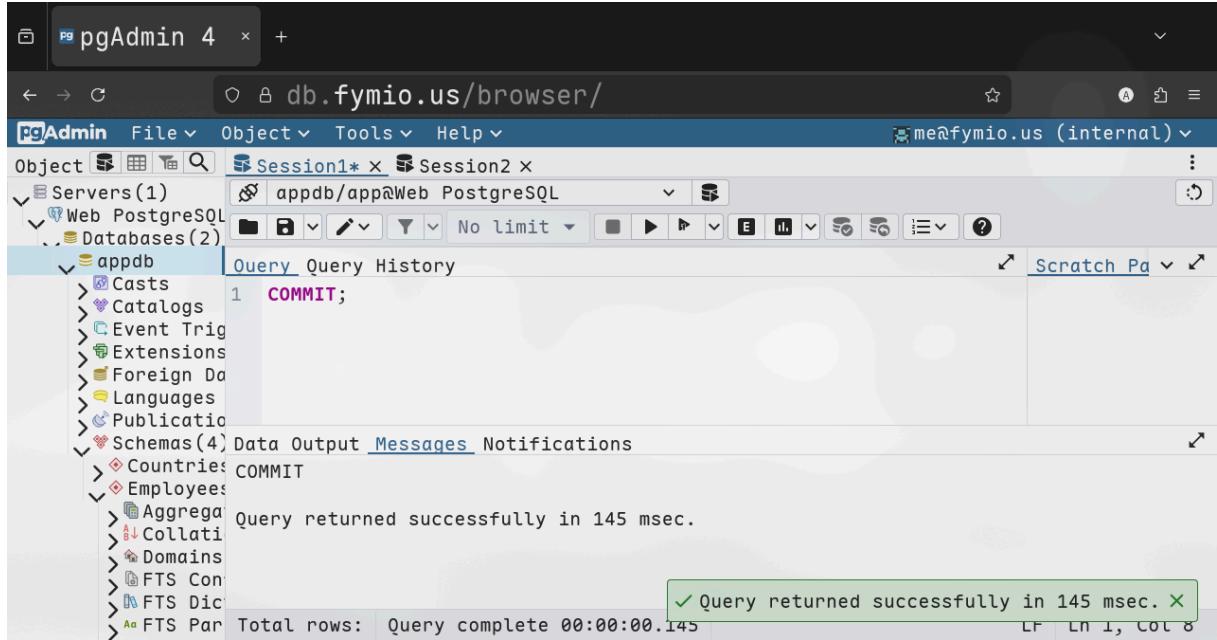


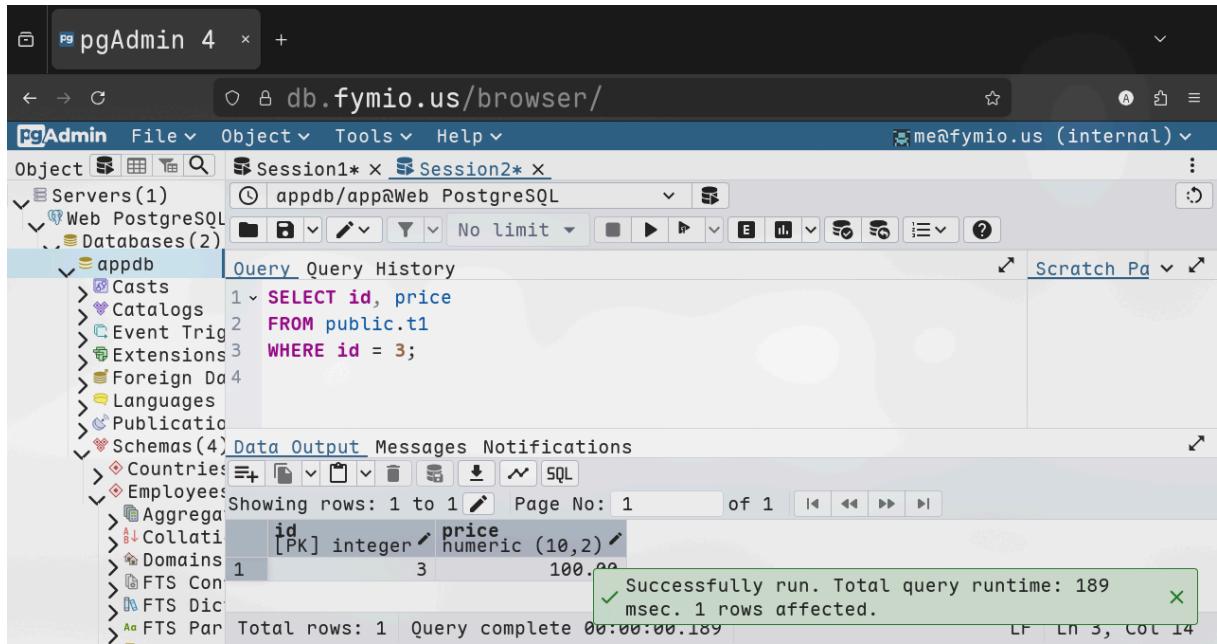
Рис. 40: Фиксирование изменений.

Теперь транзакция Session1 завершена. Монопольная блокировка снята, а новые данные зафиксированы.

Затем я провел повторное чтение в “Session2” при помощи Скрипта 33. (Рис. 41)

```
SELECT id, price
FROM public.t1
WHERE id = 3;
```

Скрипт. 33: Чтение в “Session2”.



The screenshot shows the pgAdmin 4 interface with two sessions: Session1 and Session2. Session2 is active. In the query history, a SELECT statement is run:

```

1 > SELECT id, price
2   FROM public.t1
3   WHERE id = 3;

```

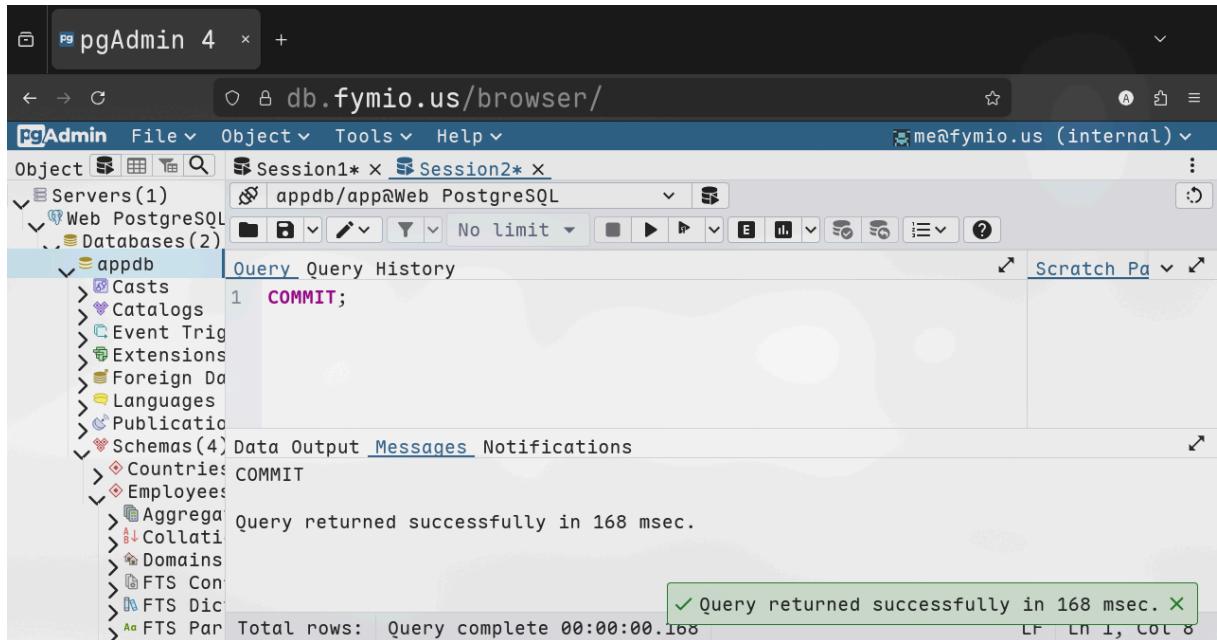
The results table shows one row:

	id	price
1	3	100.00

A green message box at the bottom right indicates the query was successfully run and completed in 189 msec.

Рис. 41: Результат использования Скрипта. 33.

После этого, я завершил транзакцию в “Session2” при помощи команды `COMMIT;` . (Рис. 42)



The screenshot shows the pgAdmin 4 interface with two sessions: Session1 and Session2. Session2 is active. The query history shows the `COMMIT;` command:

```

1 > COMMIT;

```

A green message box at the bottom right indicates the query returned successfully in 168 msec.

Рис. 42: Закрытие транзакции в “Session2”.

Задание 4. Уровень изоляции REPEATABLE READ

В “Session1” я запустил транзакцию `REPEATABLE READ` при помощи Скрипта. 34. (Рис. 43)

```

BEGIN ISOLATION LEVEL REPEATABLE READ;

SELECT id, price
  FROM public.t1
 WHERE id = 3;

```

Скрипт. 34: Запуск транзакции в “Session1”.

pgAdmin 4

Session1* x Session2* x

Object Servers(1) Web PostgreSQL Databases(2) appdb

Query History

```

1 BEGIN ISOLATION LEVEL REPEATABLE READ;
2
3 v SELECT id, price
4   FROM public.t1
5 WHERE id = 3;

```

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

	id	price
1	3	100.00

Successfully run. Total query runtime: 181 msec. 1 rows affected.

Total rows: 1 Query complete 00:00:00.181

Рис. 43: Результат выполнения Скрипт. 34.

Затем в “Session2” я попытался изменить ту же строку при помощи Скрипт. 35. (Рис. 44)

```

BEGIN;
UPDATE public.t1
SET price = 1
WHERE id = 3;
COMMIT;

```

Скрипт. 35: Изменение строки в “Session2”.

pgAdmin 4

Session1* x Session2* x

Object Servers(1) Web PostgreSQL Databases(2) appdb

Query History

```

1 BEGIN;
2 v UPDATE public.t1
3 SET price = 1
4 WHERE id = 3;
5 COMMIT;

```

Notifications

Query returned successfully in 156 msec.

Query returned successfully in 156 msec. x

Total rows: 1 Query complete 00:00:00.156

Рис. 44: Результат выполнения Скрипт. 35.

- `UPDATE` в “Session2” получит монопольную блокировку строки.
- Сессия “Session1” держит совместную блокировку, но она не мешает `UPDATE`.

- После `COMMIT` изменения вступают в силу для других будущих транзакций, но не для “Session1”.

После, я повторил `SELECT` в “Session1” при помощи Скрипт. 36. (Рис. 45)

```
SELECT id, price
FROM public.t1
WHERE id = 3;

COMMIT;
```

Скрипт. 36: Повторный `SELECT` в “Session1”.

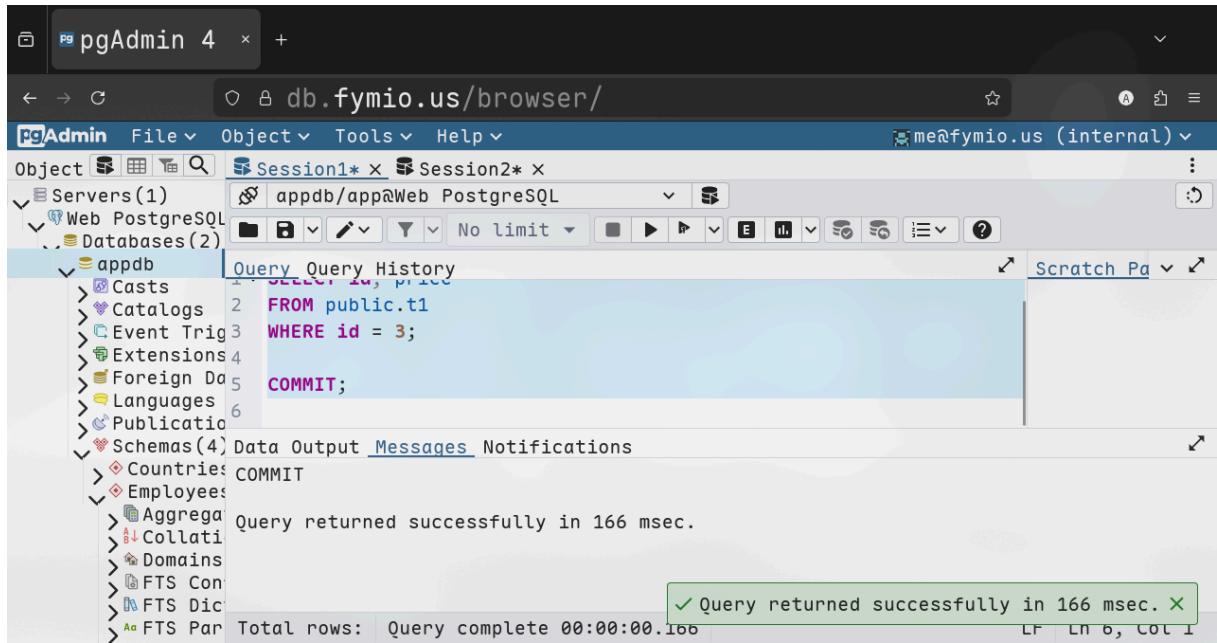


Рис. 45: Результат выполнения Скрипта. 36.

Результат остался тот, который был на момент начала транзакции “Session1”.

Для демонстрации отсутствия фантомов, я начал новую транзакцию в “Session1” при помощи Скрипта. 37. (Рис. 46)

```
BEGIN ISOLATION LEVEL REPEATABLE READ;

SELECT id, price
FROM public.t1
WHERE price = 1;
```

Скрипт. 37: Начало транзакции в “Session1”

```

BEGIN ISOLATION LEVEL REPEATABLE READ;
SELECT id, price
FROM public.t1
WHERE price = 1;

```

	id	price
1	3	1.00

Successfully run. Total query runtime: 372 msec. 1 rows affected.

Рис. 46: Результат выполнения Скрипт. 37.

В “Session2” добавил новую строку при помощи Скрипт. 38. (Рис. 47)

```

BEGIN;
INSERT INTO public.t1 VALUES (4, 1);
COMMIT;

```

Скрипт. 38: Добавление новой строки.

```

BEGIN;
INSERT INTO public.t1 VALUES (4, 1);
COMMIT;

```

Query returned successfully in 148 msec.

Query returned successfully in 148 msec.

Рис. 47: Результат выполнения Скрипт. 38.

Затем повторил `SELECT` для “Session1” при помощи Скрипт. 39. (Рис. 48)

```

SELECT id, price
FROM public.t1
WHERE price = 1;

COMMIT;

```

Скрипт. 39: Повторный `SELECT` для “Session1”.

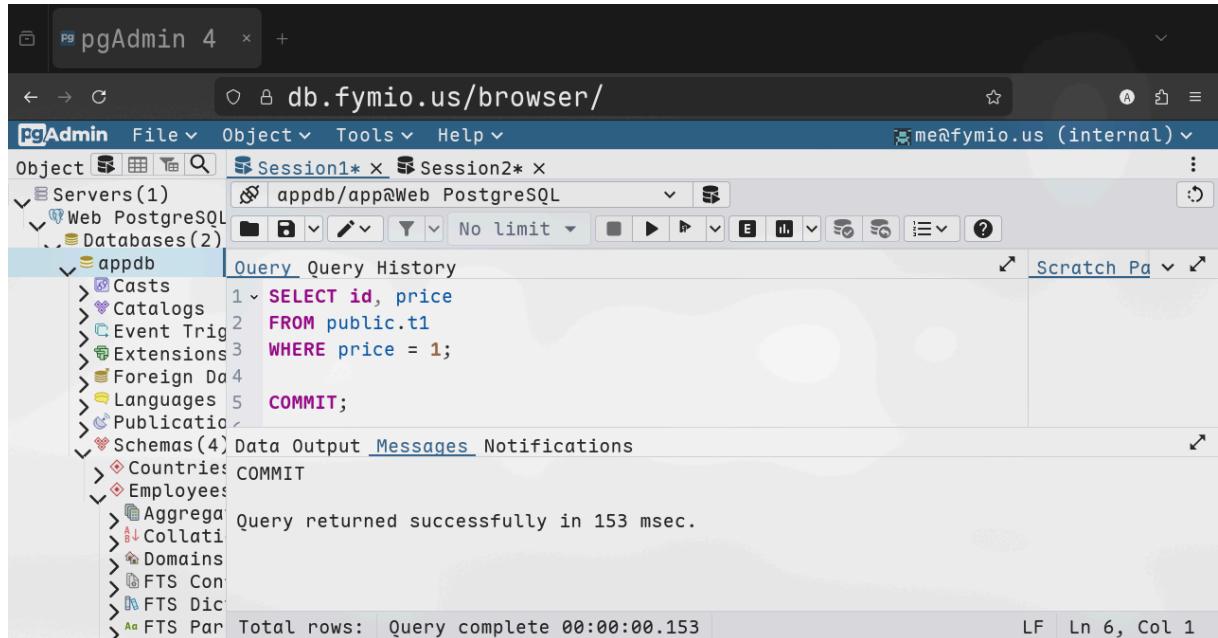


Рис. 48: Результат выполнения Скрипта. 39.

Задание 5. Уровень изоляции SERIALIZABLE

Я обновил строку в “Session1” и прочитал ее при помощи Скрипта. 40. (Рис. 49)

```

BEGIN ISOLATION LEVEL SERIALIZABLE;

UPDATE public.t1
SET price = 111
WHERE id = 1;

SELECT id, price
FROM public.t1
WHERE id = 1;

```

Скрипт. 40: Обновление и чтение строки в “Session1”.

The screenshot shows the pgAdmin 4 interface. The title bar says "pgAdmin 4" and the address bar says "db.fymio.us/browser/". The left sidebar shows a tree view of the database structure under "appdb", including "Casts", "Catalogs", "Event Trig", "Extensions", "Foreign Da", "Languages", "Publication", and "Schemas(4)". The "Schemas" node is expanded, showing "Countries" and "Employees". The "Employees" node is expanded, showing "Aggrega", "Collati", "Domains", "FTS Con", "FTS Dic", and "FTS Par". The "Domains" node is selected and its details are shown in the main pane: "id" is the primary key (PK) of type integer, and "price" is of type numeric (10,2). The value for "id" is 1 and for "price" is 111.00. The status bar at the bottom says "Total rows: 1 Query complete 00:00:00.181". The right pane shows a "Scratch Pad" tab.

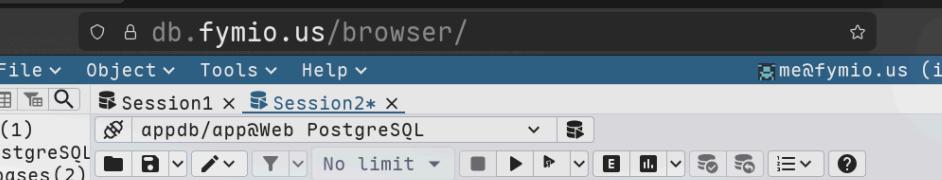
Рис. 49: Результат выполнения Скрипт. 40

У транзакции "Session1" теперь есть монопольная блокировка на строку `id=1` до `COMMIT`.

Затем из "Session2" я попытался обновить ту же строку при помощи Скрипта 41. (Рис. 50)

```
BEGIN ISOLATION LEVEL SERIALIZABLE;  
  
UPDATE public.t1  
SET price = 222  
WHERE id = 1;  
  
COMMIT;
```

Скрипт. 41: Попытка обновления строки из “Session2”.



The screenshot shows the pgAdmin 4 interface. The title bar reads "pgAdmin 4". The address bar shows the connection URL "db.fymio.us/browser/". The top menu bar includes "File", "Object", "Tools", and "Help". The left sidebar shows the database structure: "Servers(1)" (Web PostgreSQL), "Databases(2)" (appdb, public), "appdb" (Casts, Catalogs, Event Trig, Extensions, Foreign Da, Languages, Publication, Schemas(4) - Countries, Employees, Aggrega, Collati, Domains, FTS Con, FTS Dic, FTS Par), and "public" (No limit). The main query editor window contains the following SQL code:

```
4 SET price = 222
5 WHERE id = 1;
6
7 COMMIT;
```

The status bar at the bottom right shows a green message: "Query returned successfully in 164 msec." and "Total rows: 1 Query complete 00:00:00.164".

Рис. 50: Результат выполнения Скрипт. 41.

Сделал `SELECT` и `COMMIT` в “Session1” при помощи Скрипта. 42. (Рис. 51)

```
SELECT id, price
FROM public.t1
WHERE id = 1;

COMMIT;
```

Скрипт. 42: `SELECT` и `COMMIT` в “Session1”.

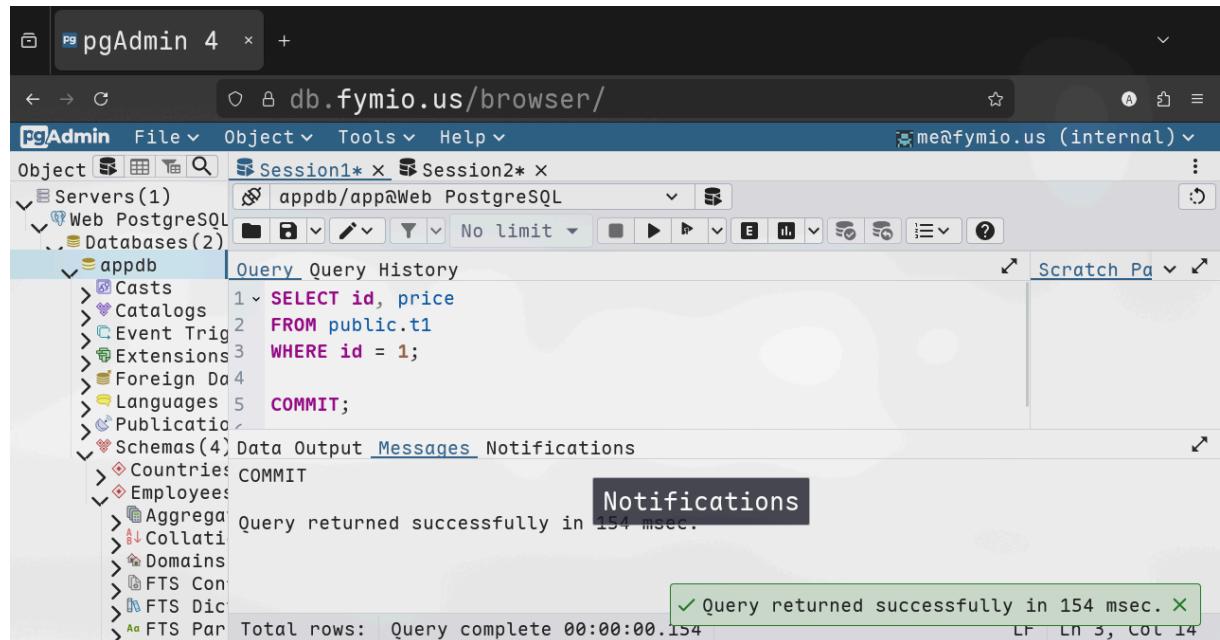


Рис. 51: Результат выполнения Скрипта. 42

Выводы и анализ результатов работы

В процессе выполнения работы были изучены основные механизмы транзакций postgresql: фиксация и откат изменений, работа с точками сохранения, а также влияние ошибок на состояние транзакции. На практике подтверждено, что изменения внутри транзакции не видны другим сессиям до выполнения `COMMIT`, а любая ошибка переводит транзакцию в состояние `aborted`, после чего дальнейшие команды игнорируются до выполнения `ROLLBACK`.

Также было проверено поведение нескольких подключений к базе: конкурентные транзакции блокируют доступ к одним и тем же строкам, что обеспечивает целостность данных. Работа с `SAVEPOINT` показала возможность частичного отката без потери всей транзакции.

Основные трудности были связаны с ошибками из-за неверной схемы таблиц и пониманием состояния `aborted`, но они были устранены в процессе выполнения. В итоге цели работы полностью достигнуты.