Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Факультет инфокоммуникационных технологий

Направление подготовки 11.03.02

Практическая работа №5

Выполнил:

Дощенников Никита Андреевич

Группа: К3221

Проверила:

Татьяна Евгеньевна Войтюк

Санкт-Петербург
2025

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)

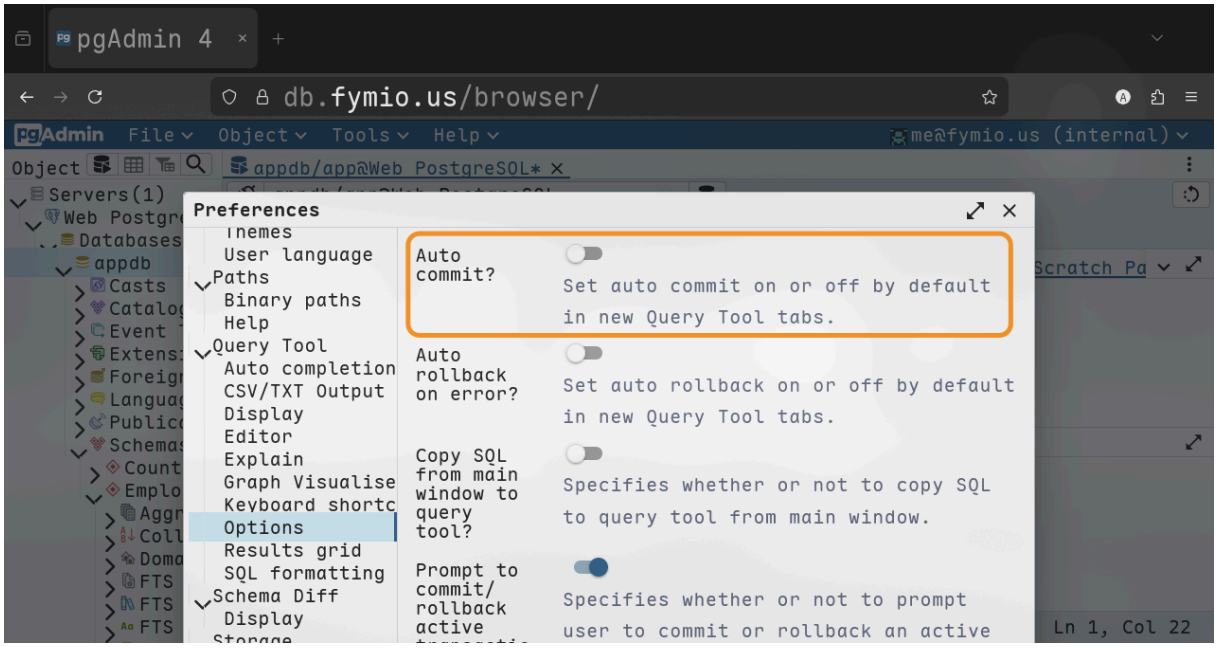

Рис. 1: Выключение автокоммитов.

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.

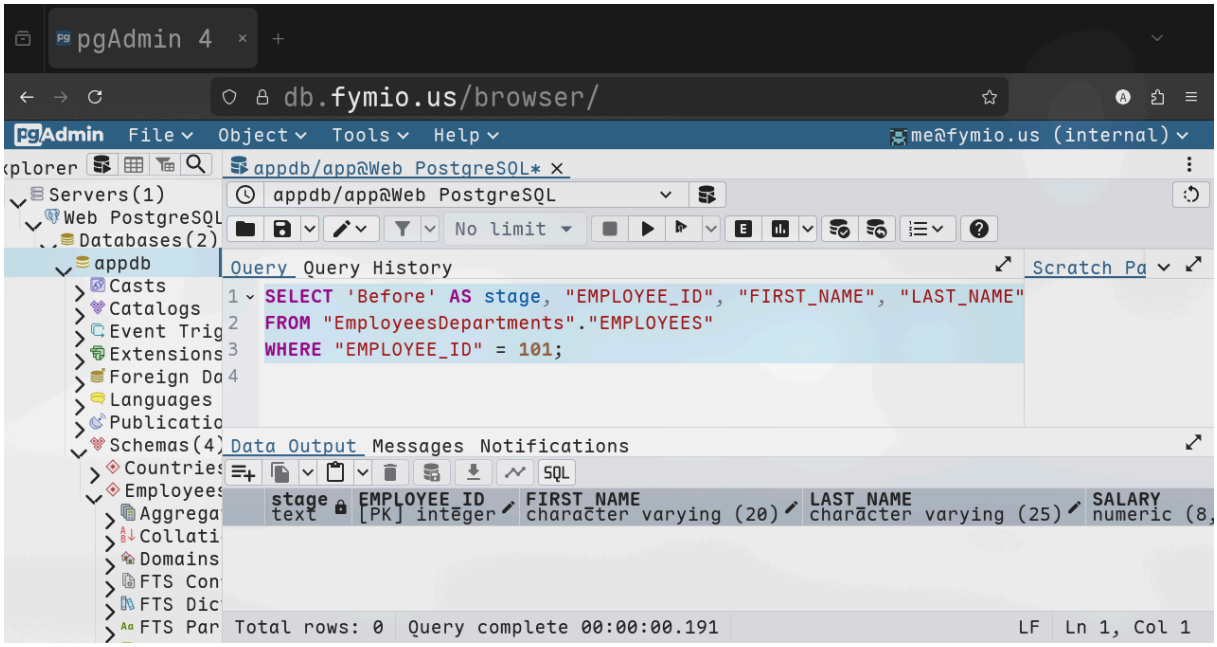

Рис. 2: Результат выполнения Скрипт. 1

```
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`.

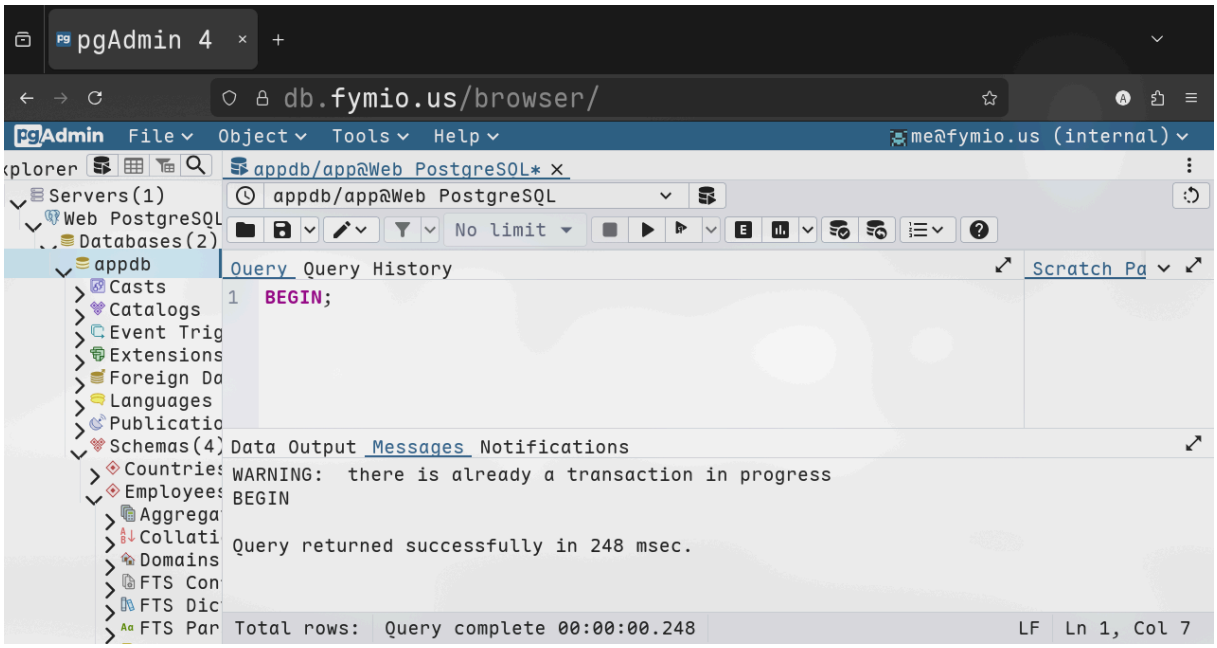

Рис. 3: Результат выполнения Скрипт. 2

```
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)

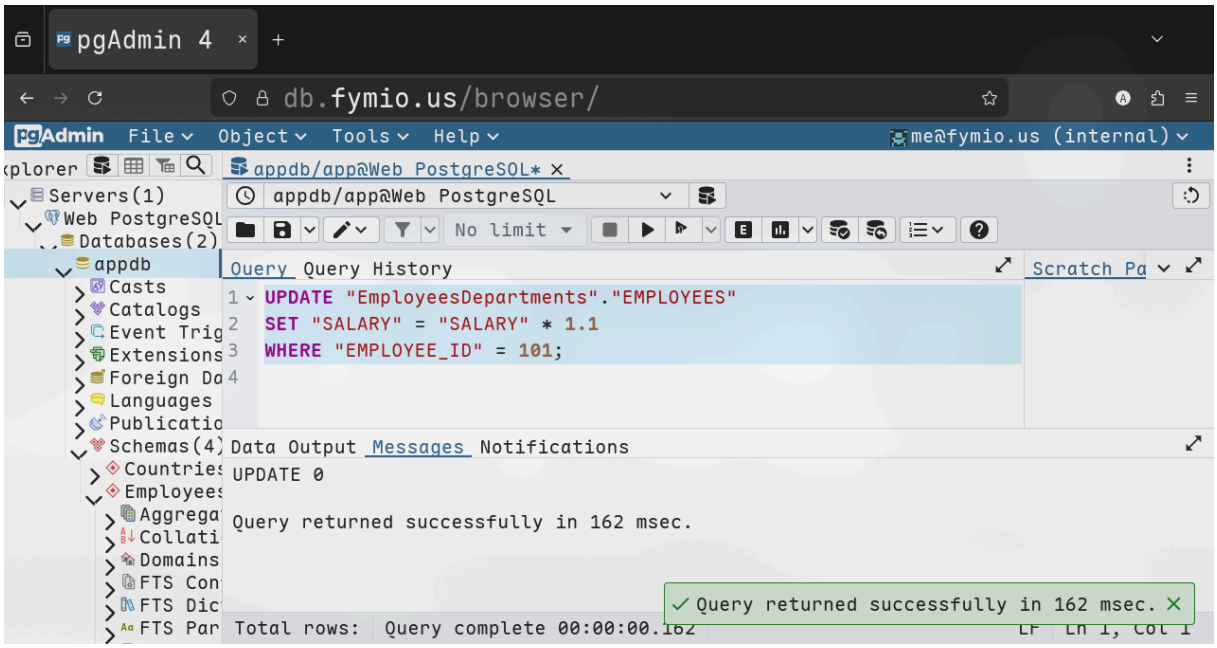

Рис. 4: Результат выполнения Скрипт. 3

```
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)

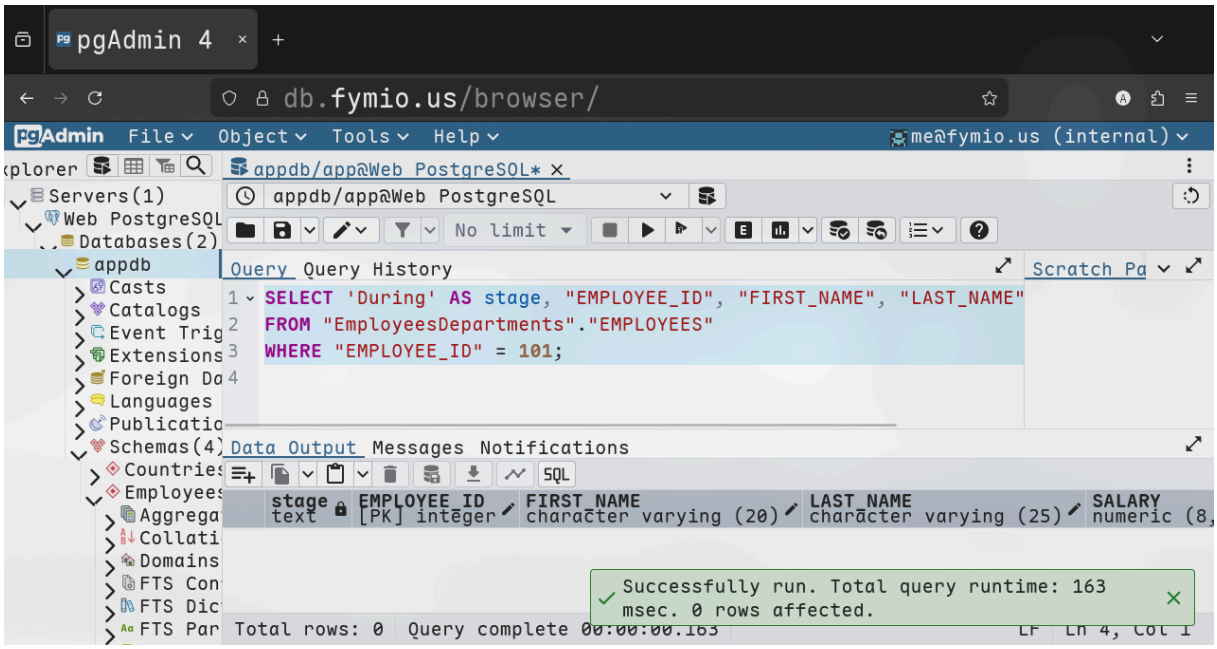

Рис. 5: Результат выполнения Скрипт. 4

```
COMMIT;
```

Скрипт. 5: Фиксация изменений транзакции.

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

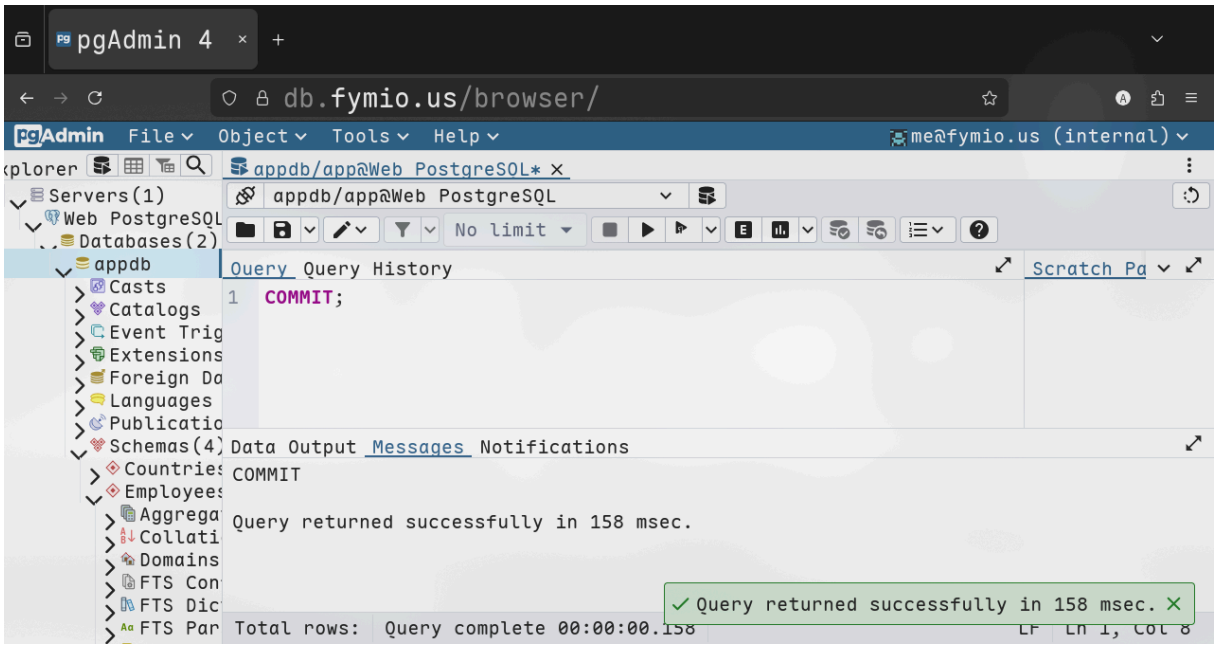

Рис. 6: Результат выполнения Скрипт. 5

```
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)

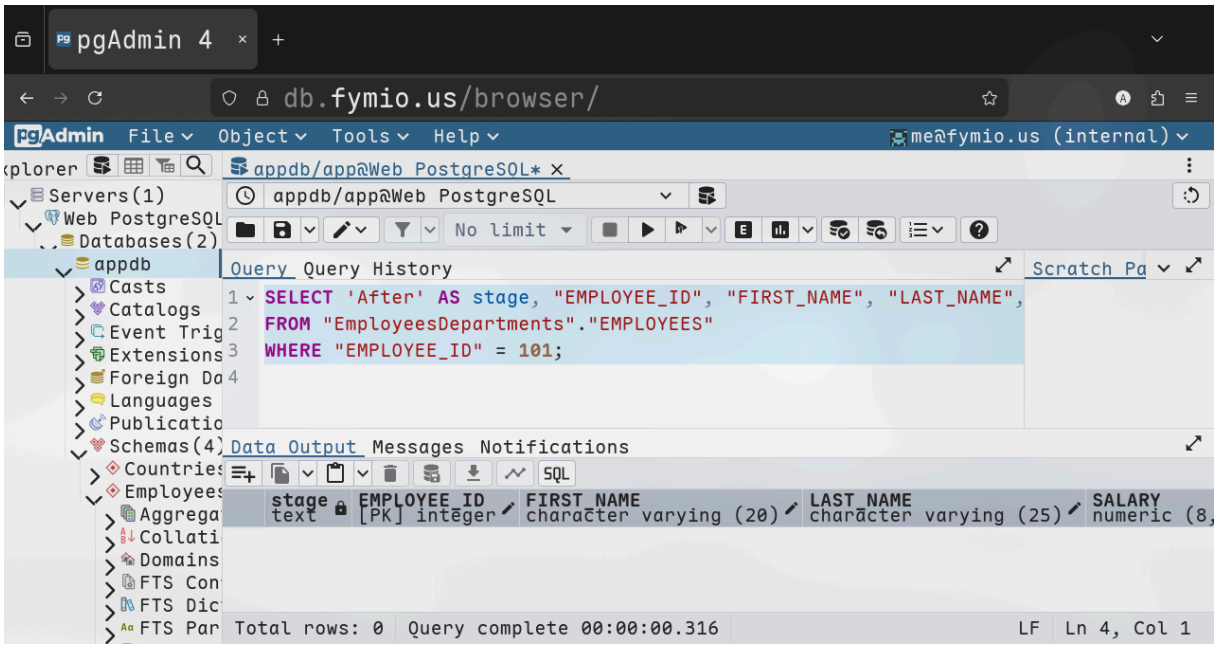

Рис. 7: Результат выполнения Скрипт. 6

```
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)

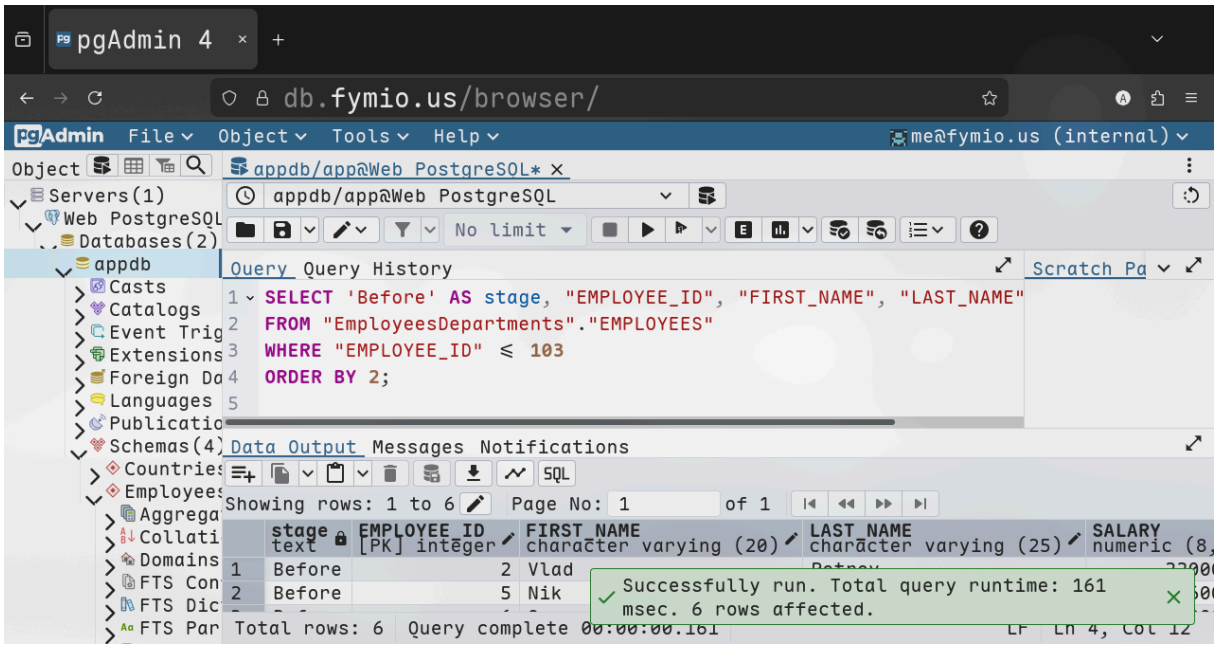

Рис. 8: Результат выполнения Скрипт. 7

```
BEGIN;
```

Скрипт. 8: Начало транзакции.

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

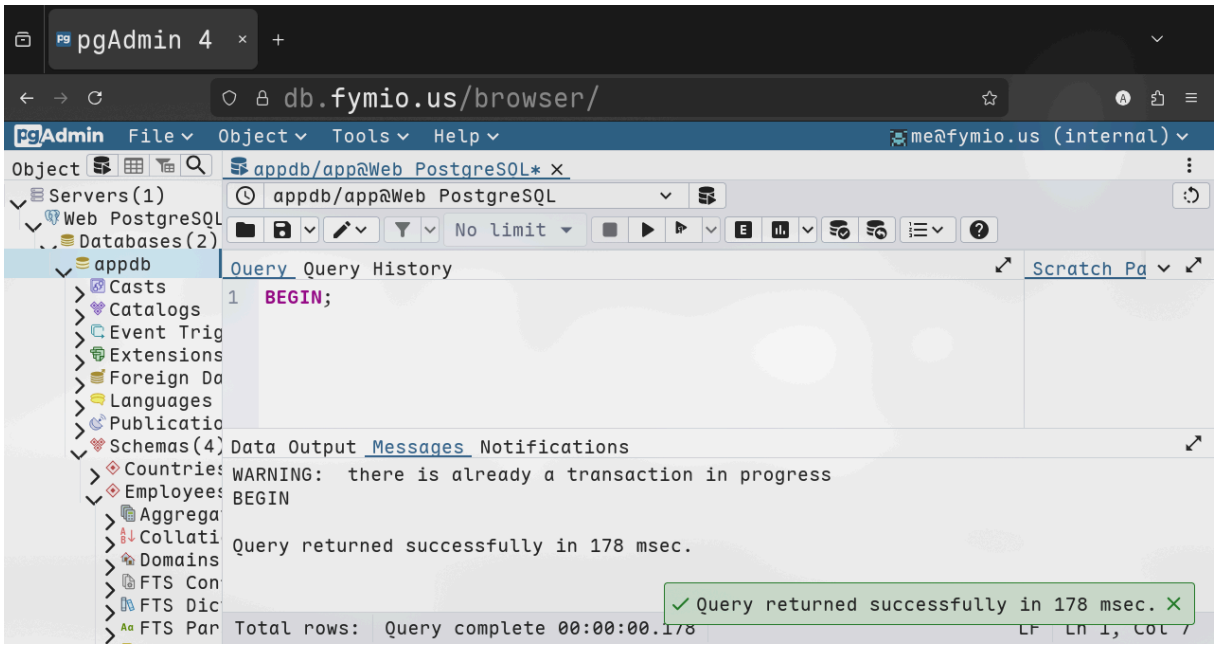

Рис. 9: Результат выполнения Скрипт. 8

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

Скрипт. 9: Повышение зарплаты сотрудникам с ID $\leq$ 103 на 10%

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

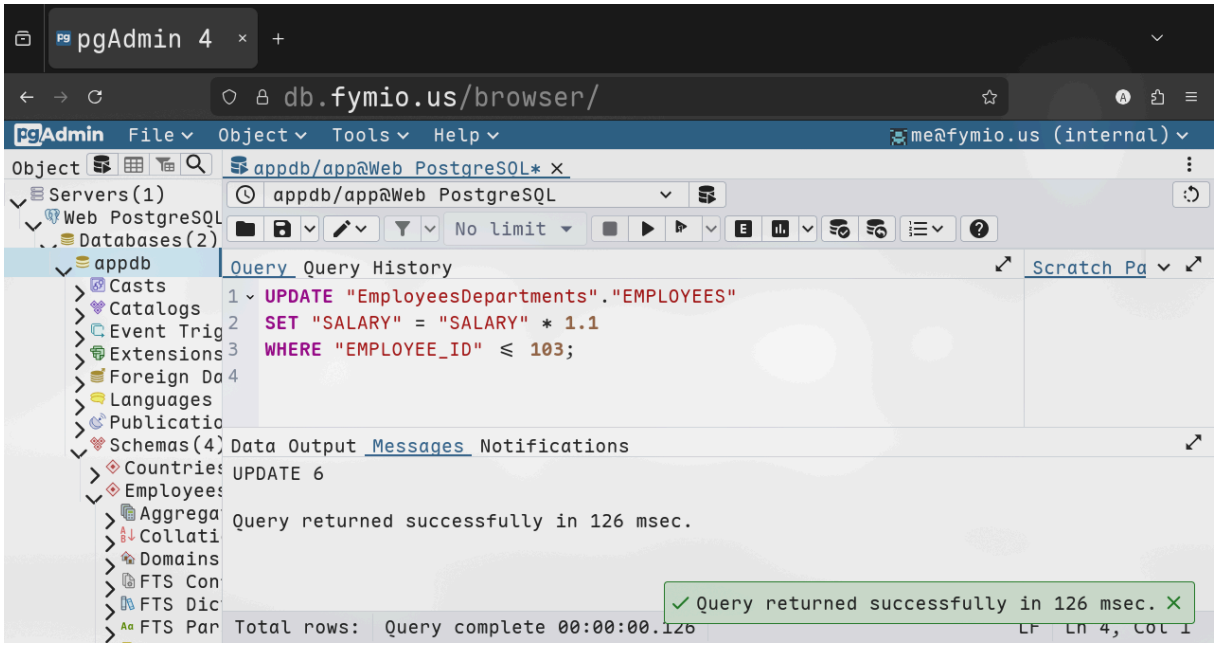

Рис. 10: Результат выполнения Скрипт. 9

```
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 ≤ 103 до выполнения `ROLLBACK`, показывая состояние данных внутри незавершённой транзакции. (Рис. 11)

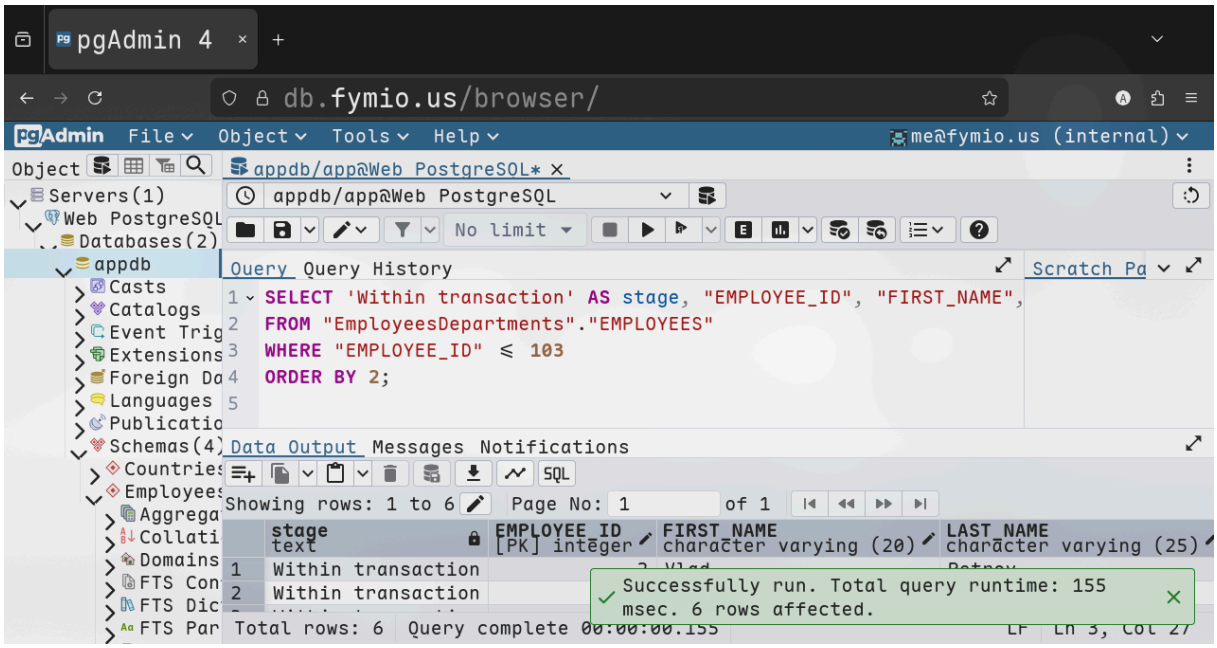

Рис. 11: Результат выполнения Скрипт. 10

```
ROLLBACK;
```

Скрипт. 11: Откат транзакции

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

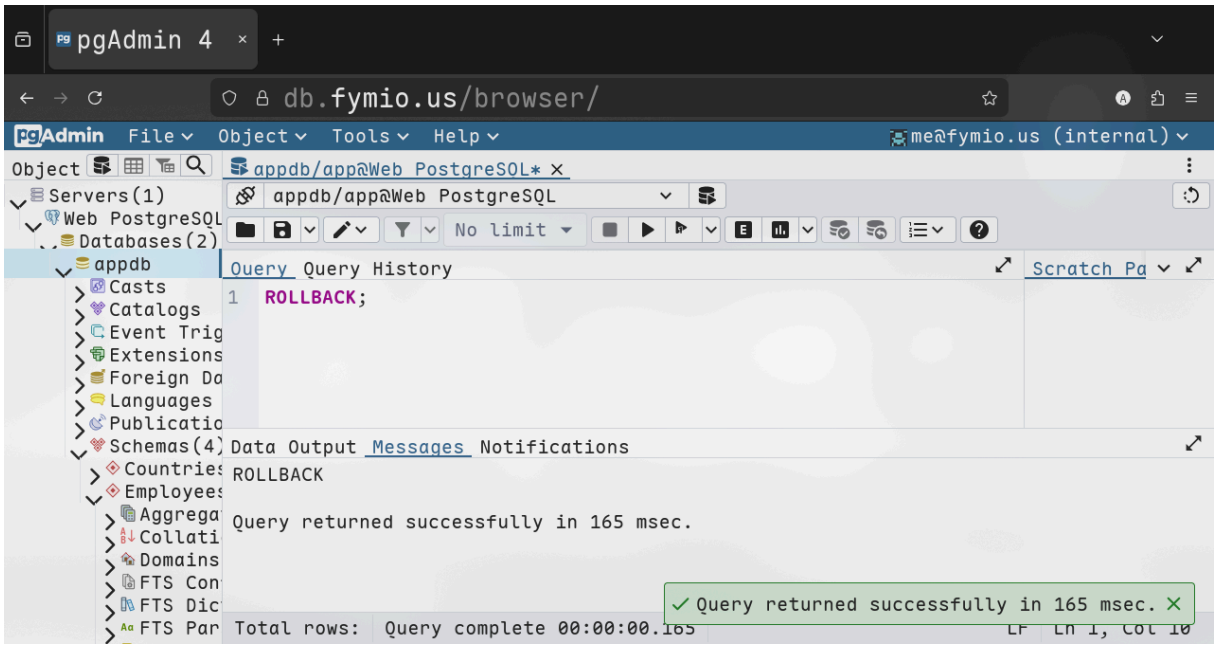

Рис. 12: Результат выполнения Скрипт. 11

```sql
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 ≤ 103 после выполнения ROLLBACK, подтверждая, что изменения не были сохранены. (Рис. 13)

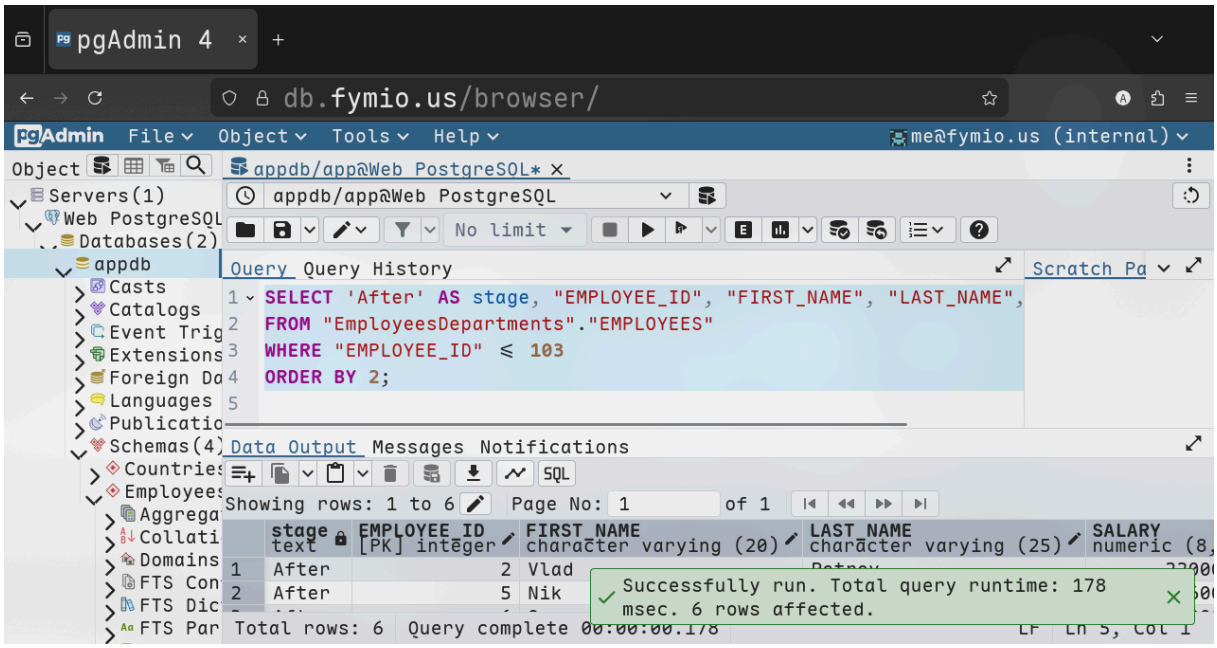

Рис. 13: Результат выполнения Скрипт. 12

```
BEGIN;
```

Скрипт. 13: Начало новой транзакции.

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

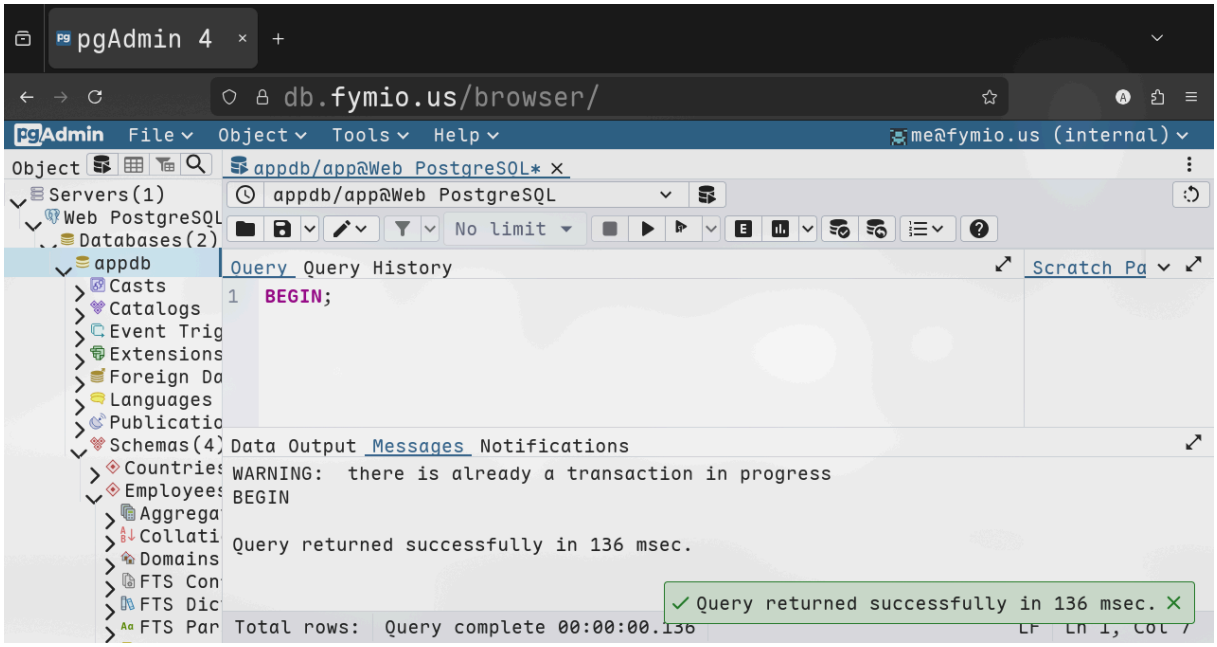

Рис. 14: Результат выполнения Скрипт. 13

```
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)

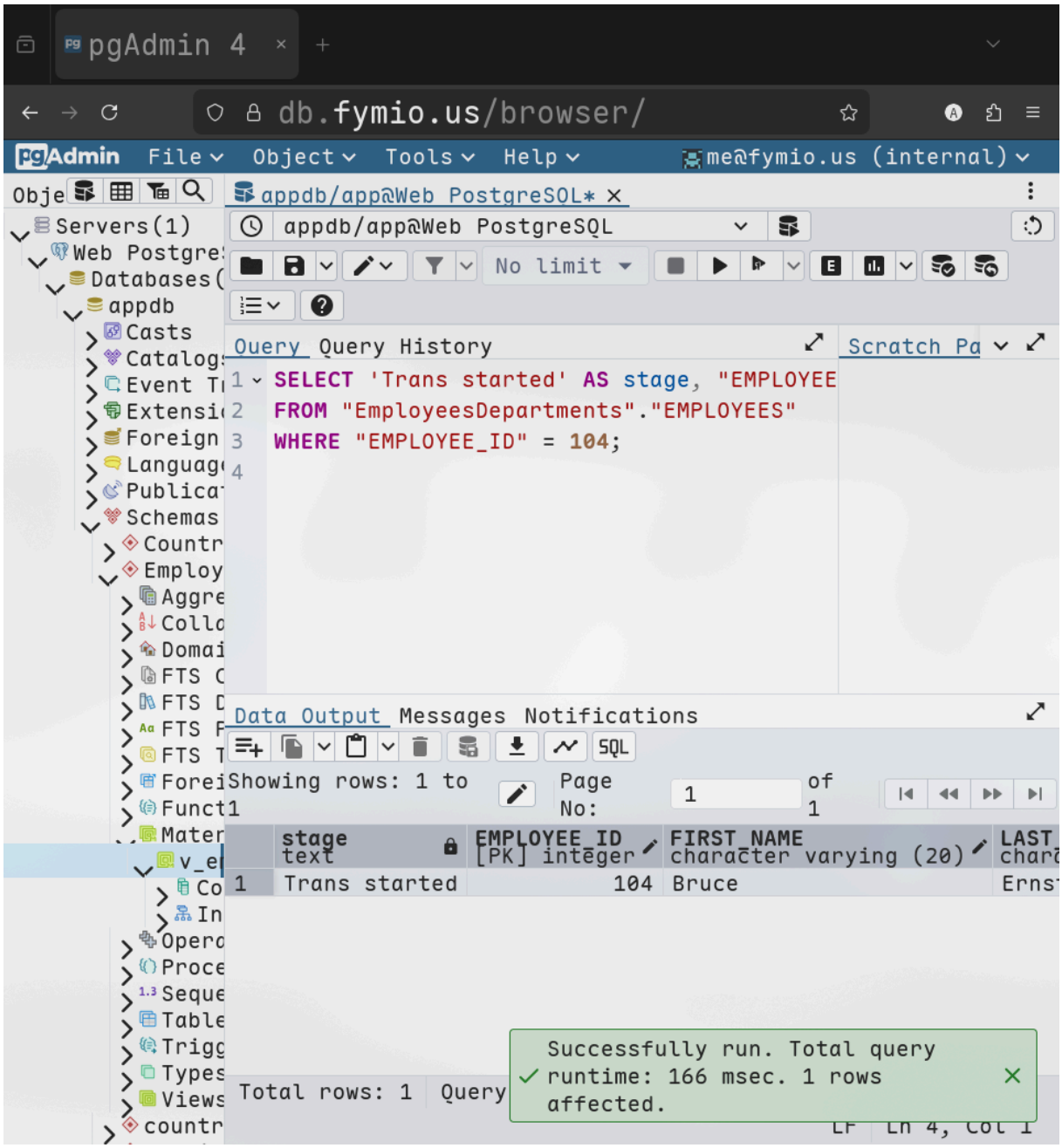

Рис. 15: Результат выполнения Скрипт. 14

```
SAVEPOINT sp1;
```

Скрипт. 15: Создание первой точки сохранения

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

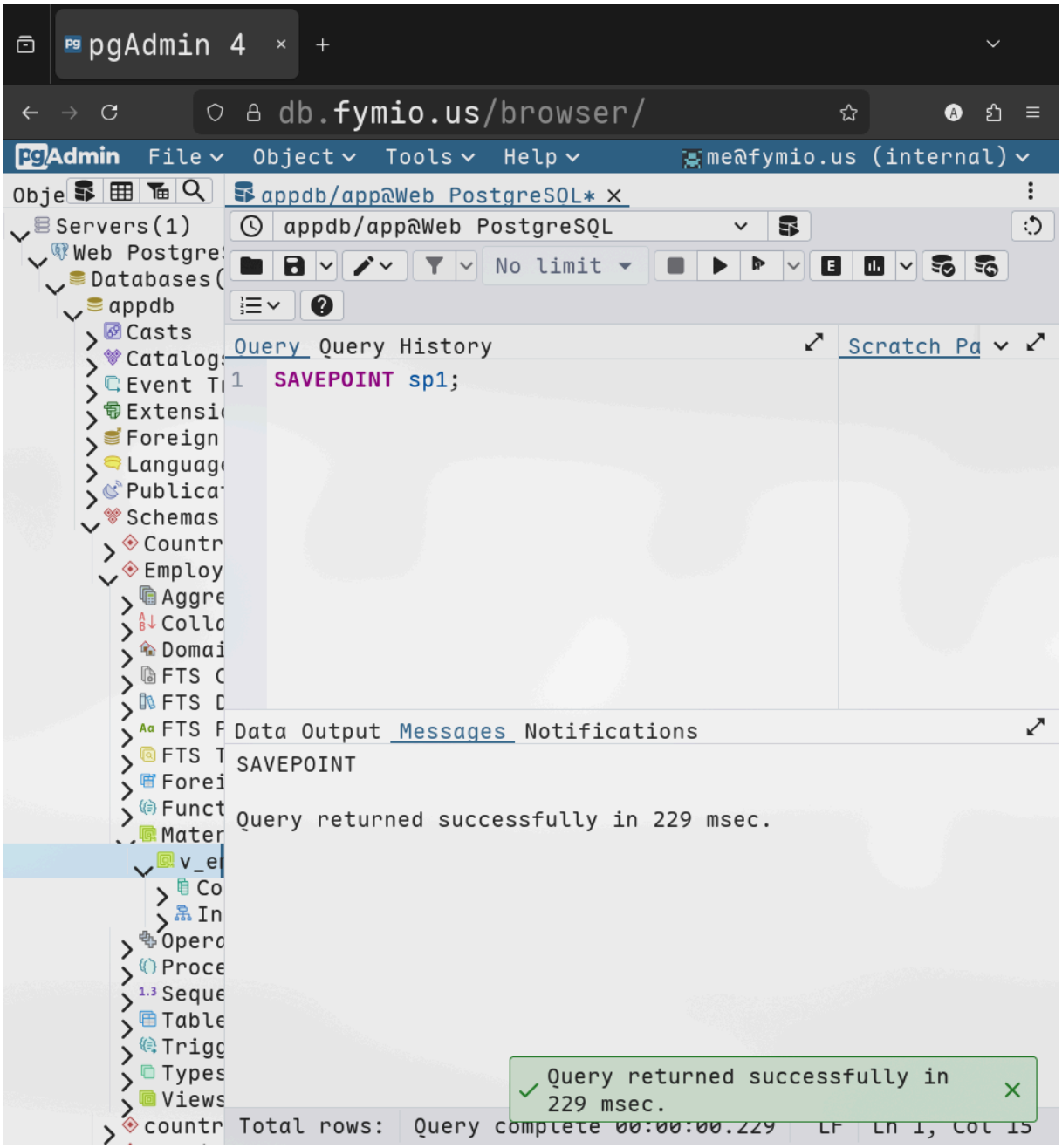

Рис. 16: Результат выполнения Скрипт. 15

```
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)

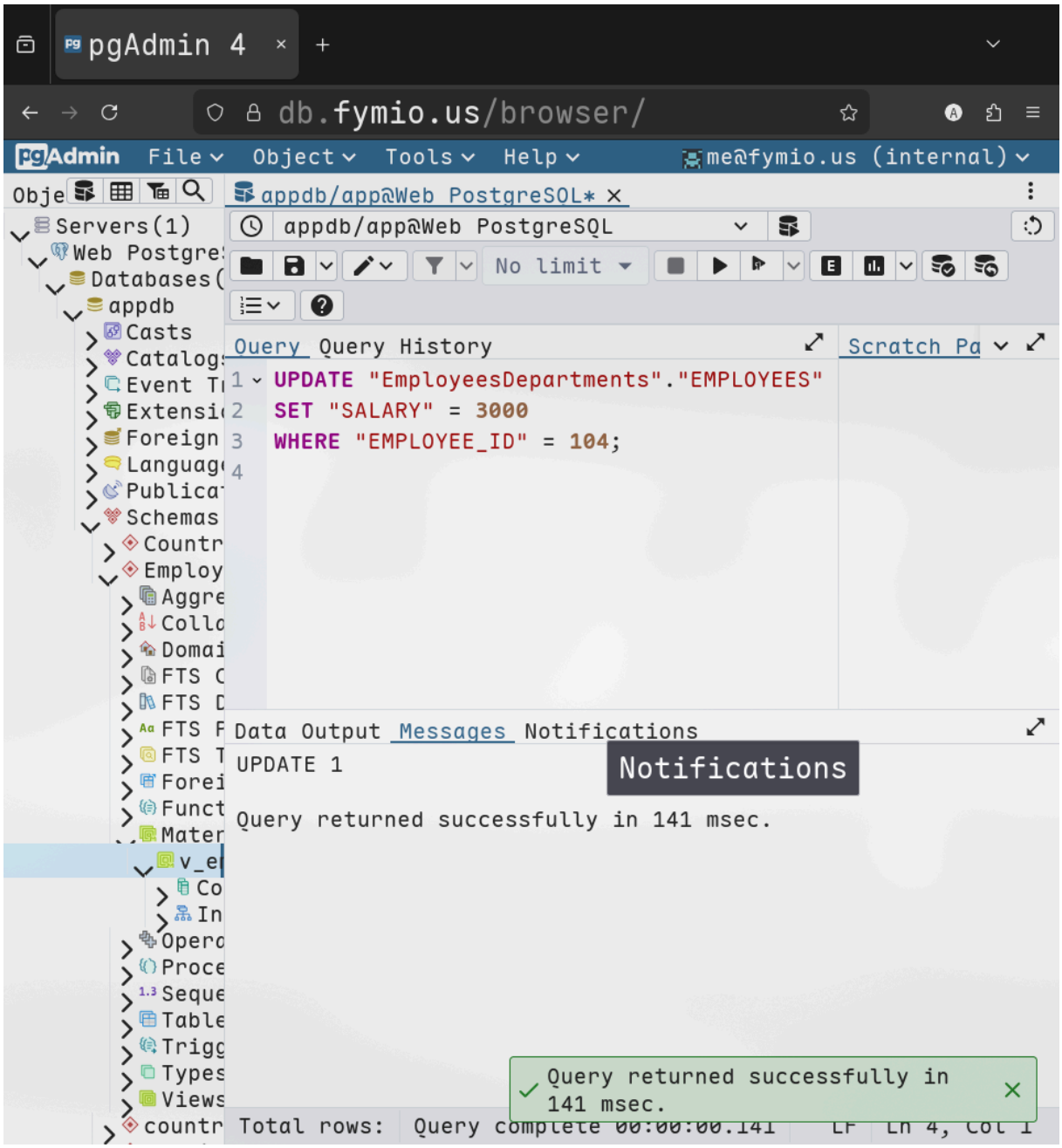

Рис. 17: Результат выполнения Скрипт. 16

```
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)

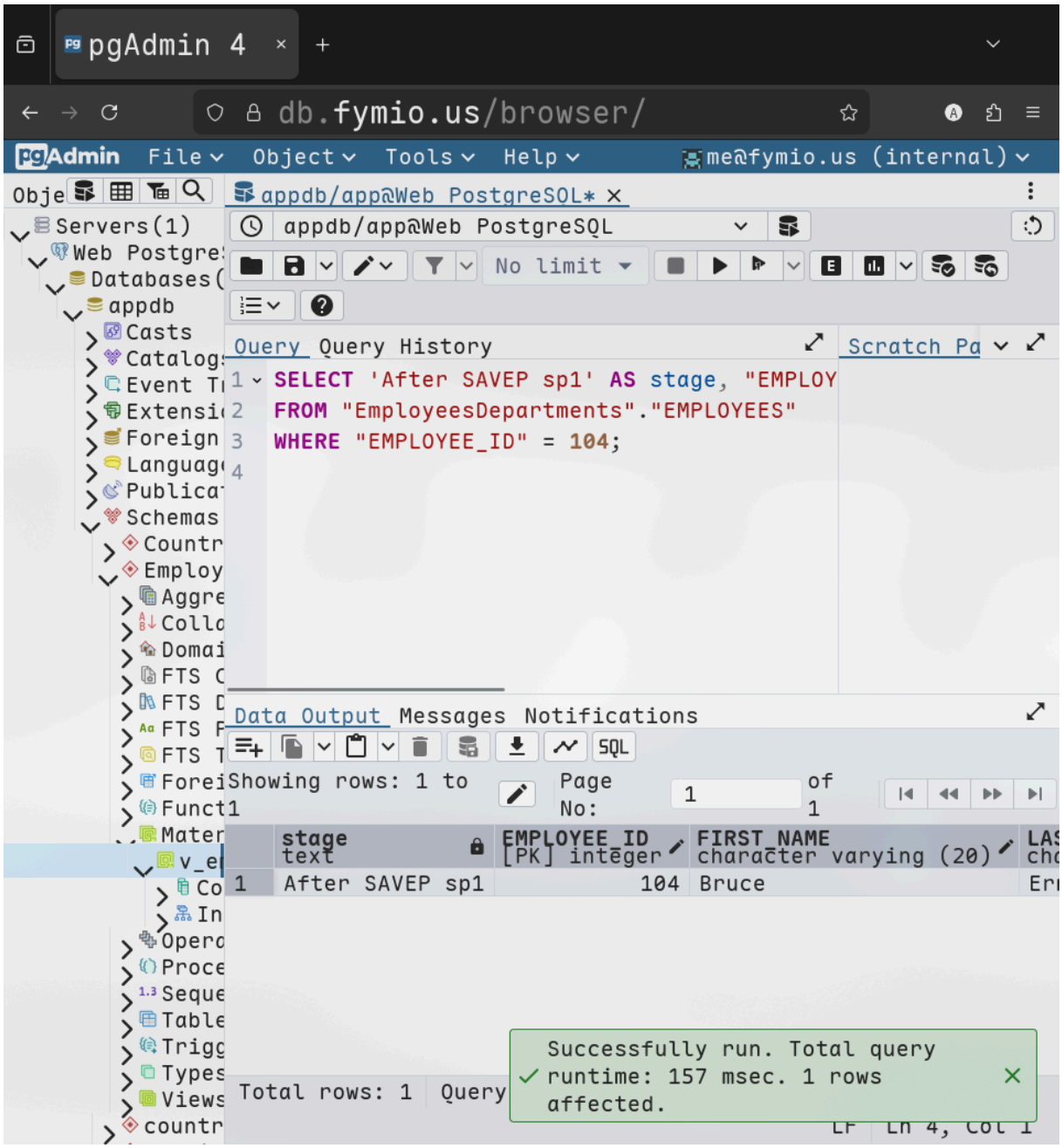

Рис. 18: Результат выполнения Скрипт. 17

```
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)

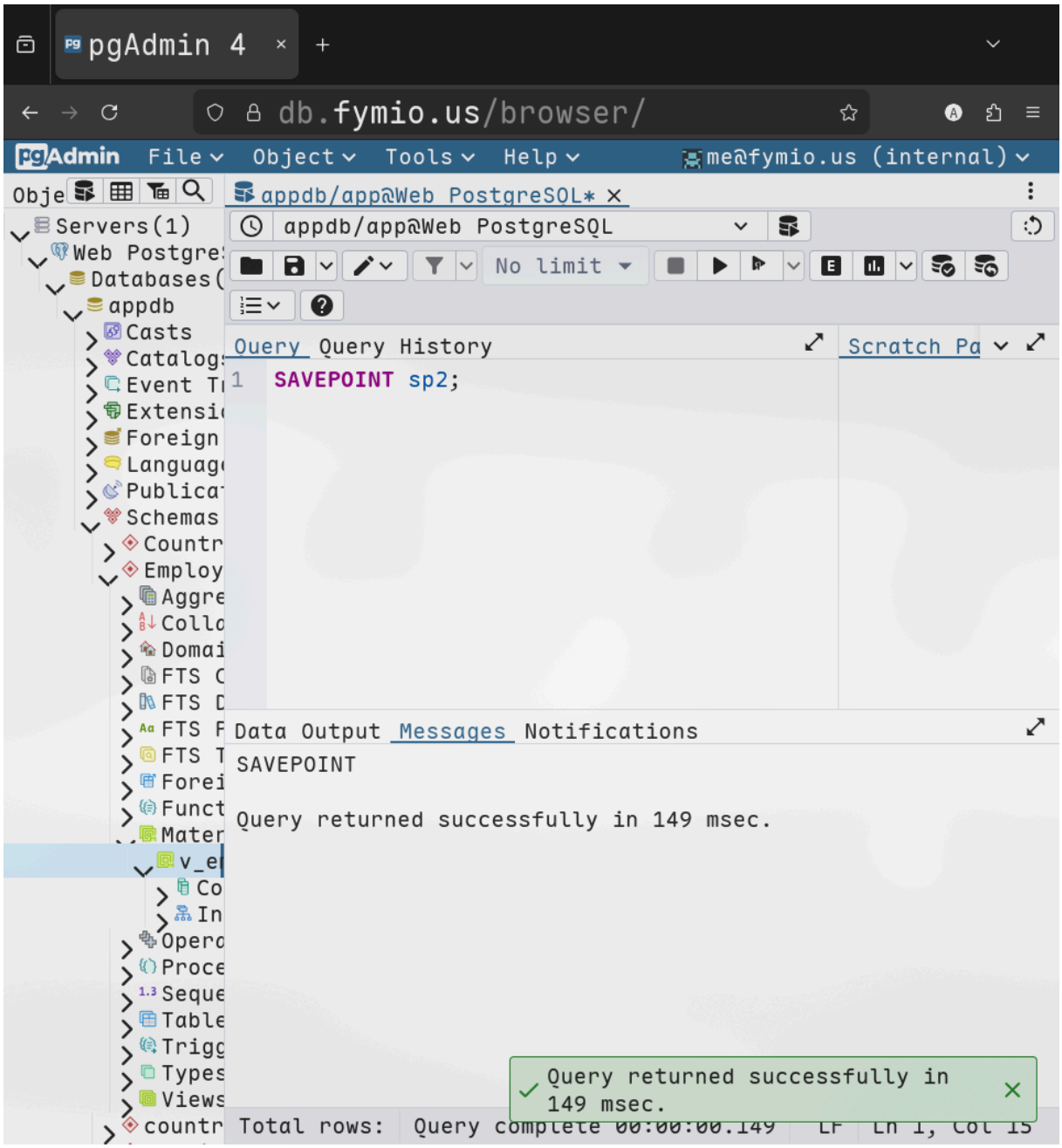

Рис. 19: Результат выполнения Скрипт. 18

```
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)

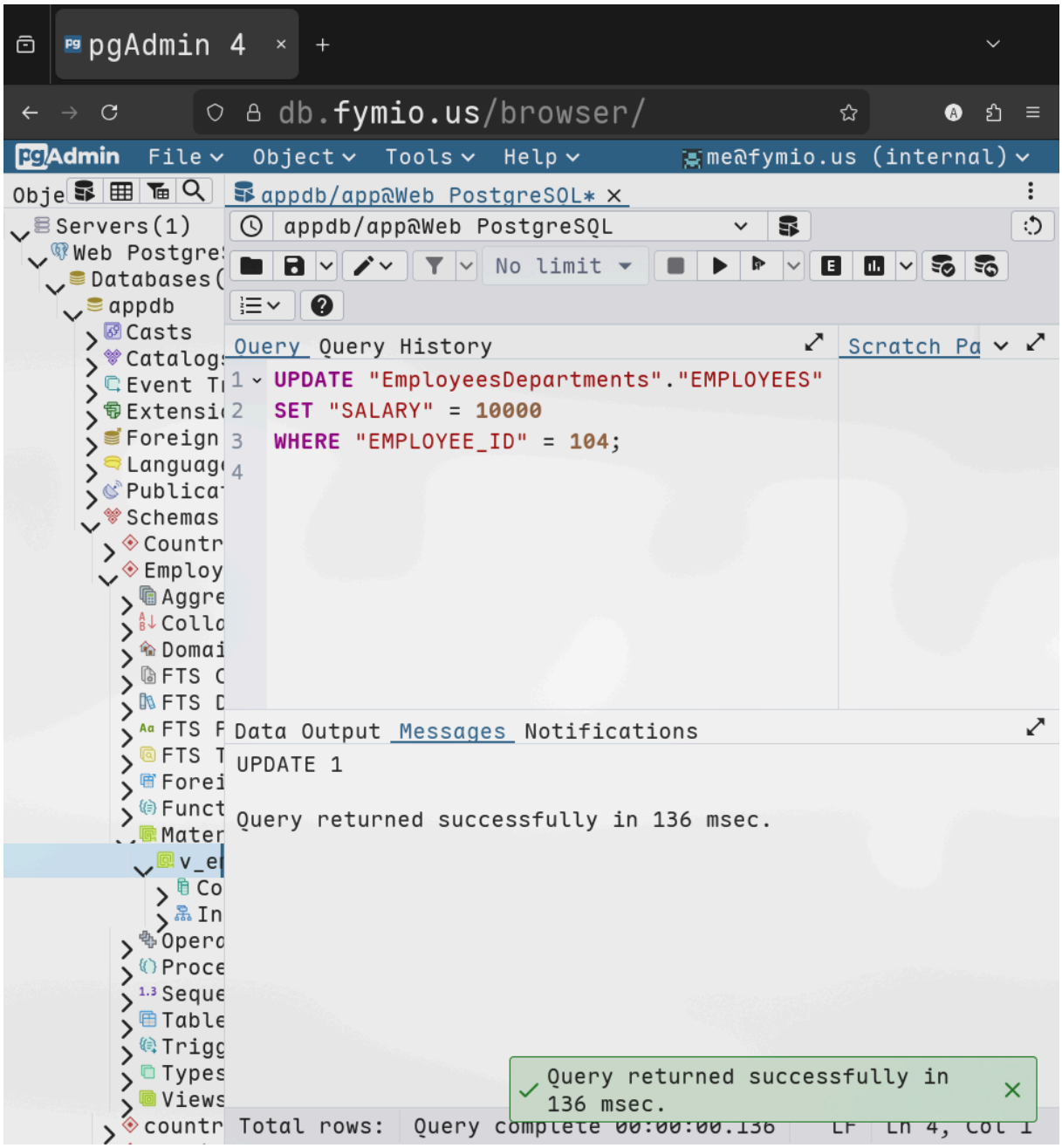

Рис. 20: Результат выполнения Скрипт. 19

```
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)

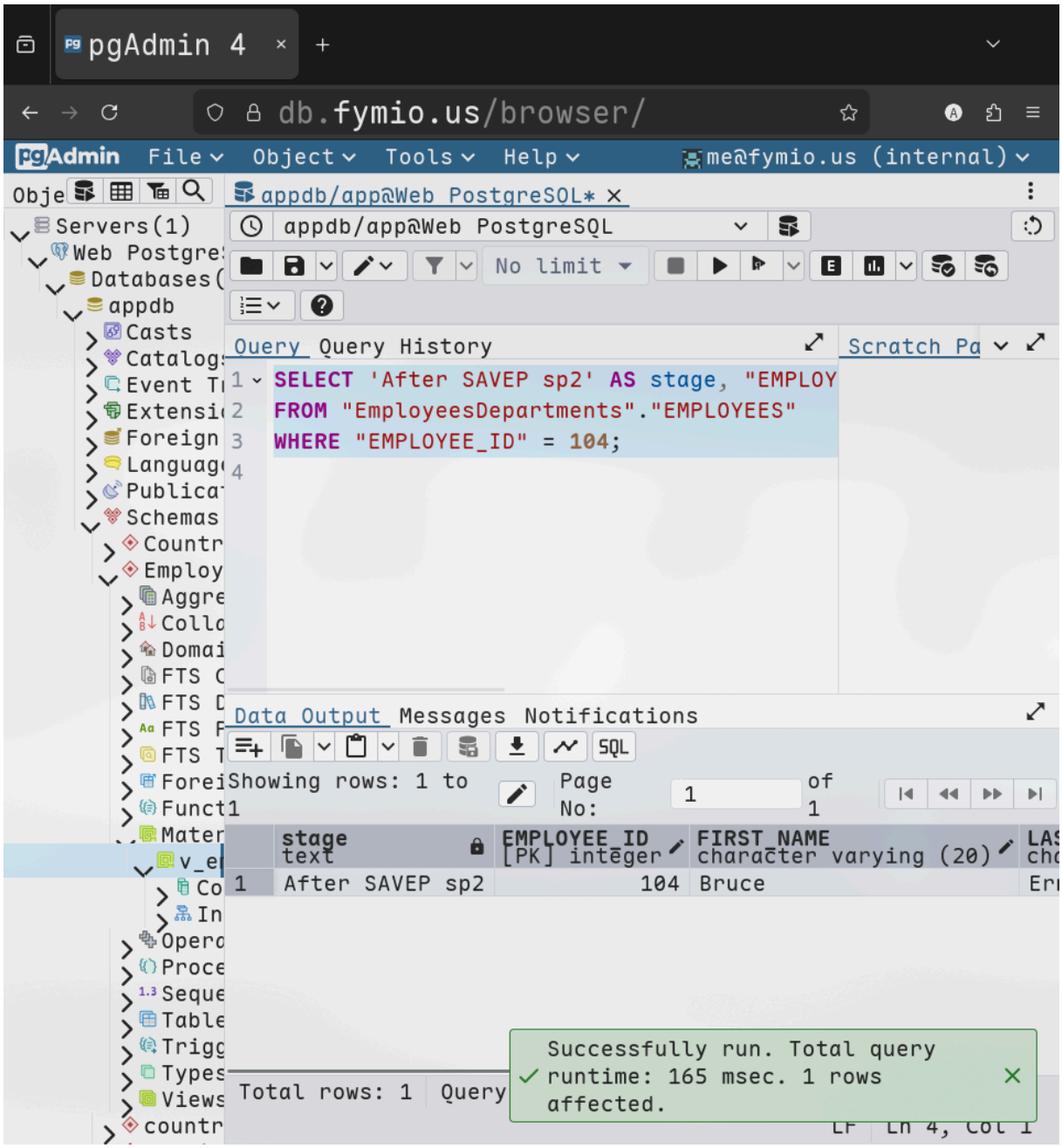

Рис. 21: Результат выполнения Скрипт. 20

```
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)

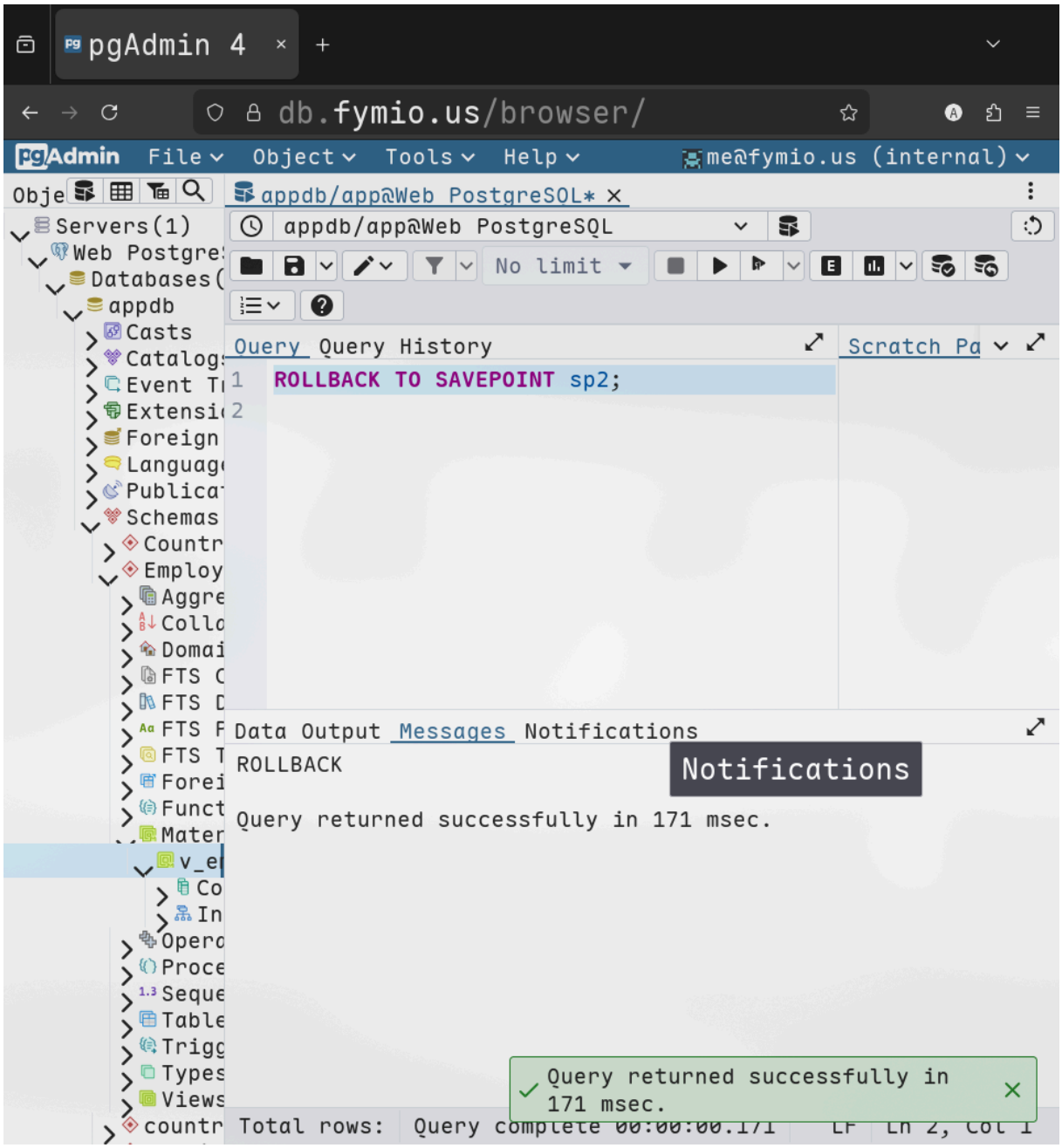

Рис. 22: Результат выполнения Скрипт. 21

```
SELECT 'After ROLLB 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)

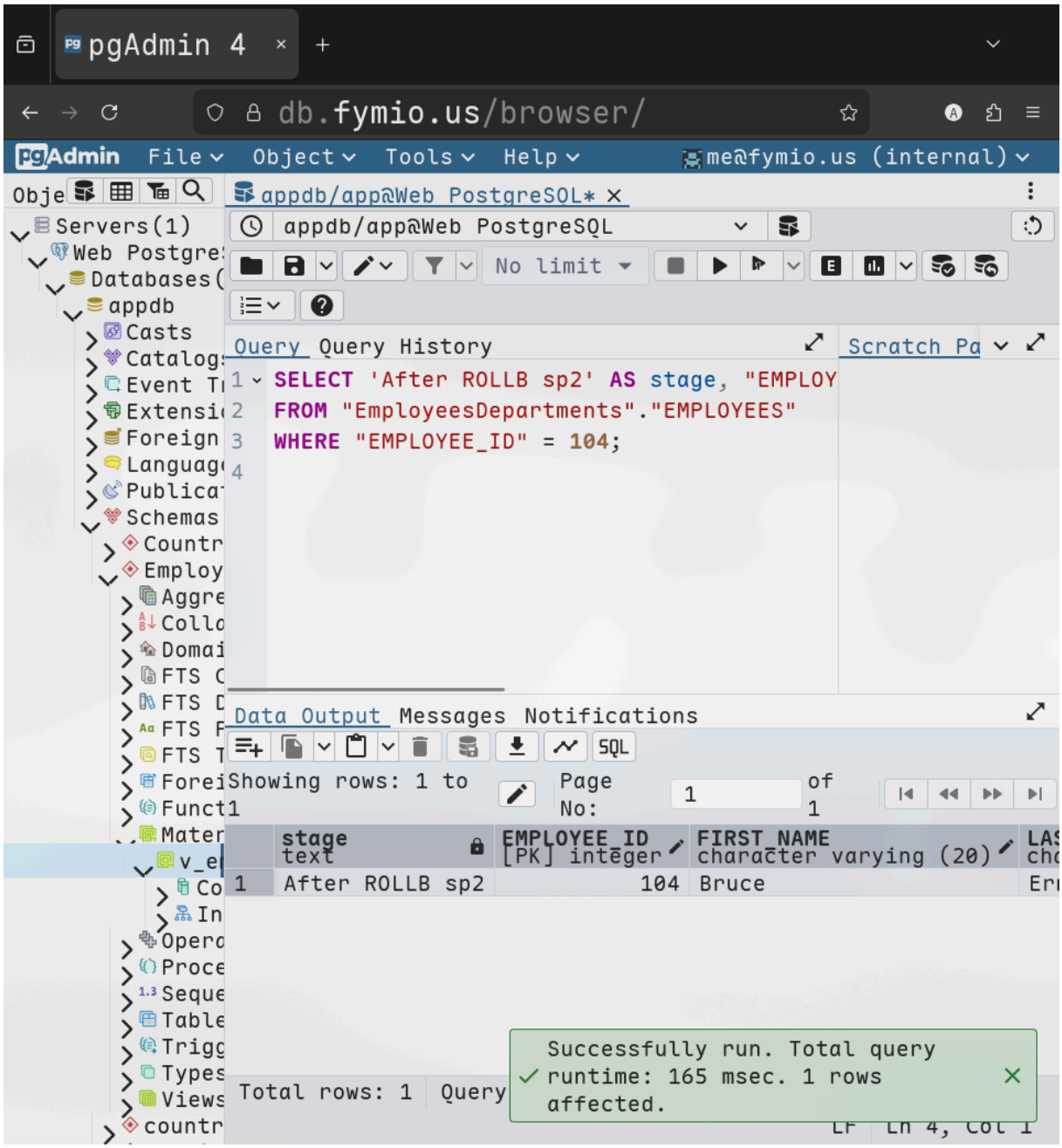

Рис. 23: Результат выполнения Скрипт. 22

```
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)

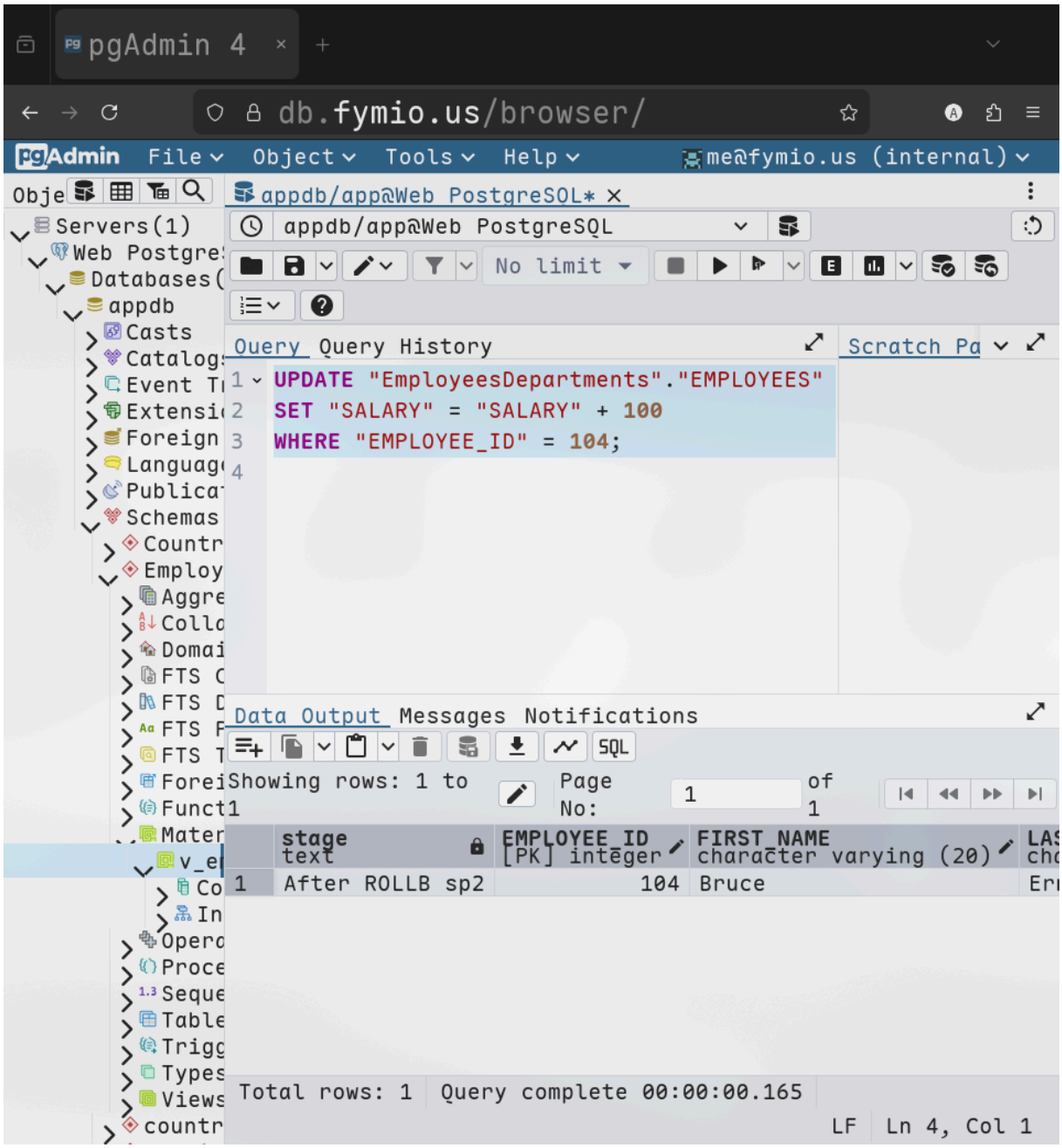

Рис. 24: Результат выполнения Скрипт. 23

```
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)

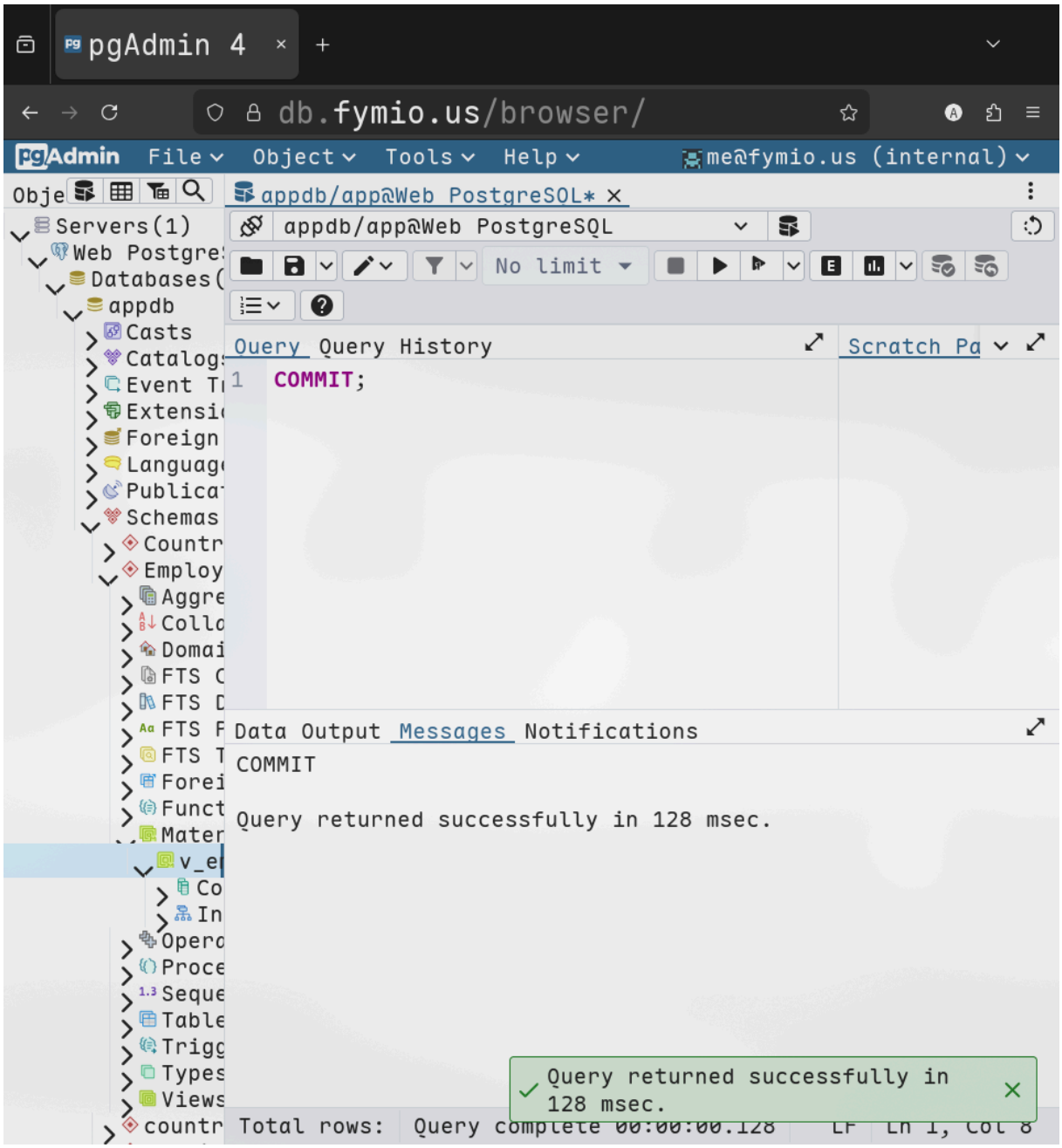

Рис. 25: Результат выполнения Скрипт. 24

```
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)

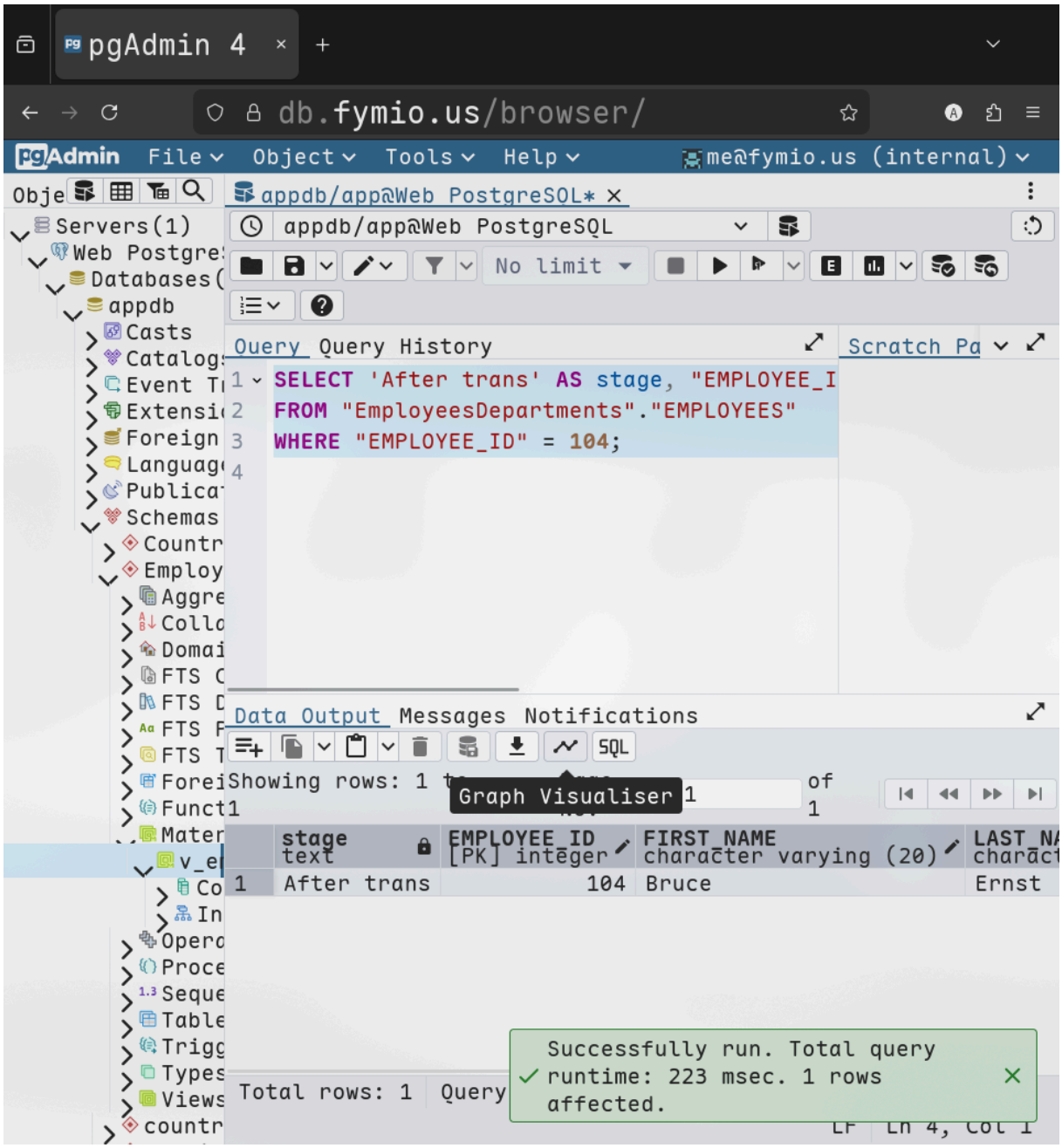

Рис. 26: Результат выполнения Скрипт. 25

Задание 2. Поиск и обнаружение блокировок

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

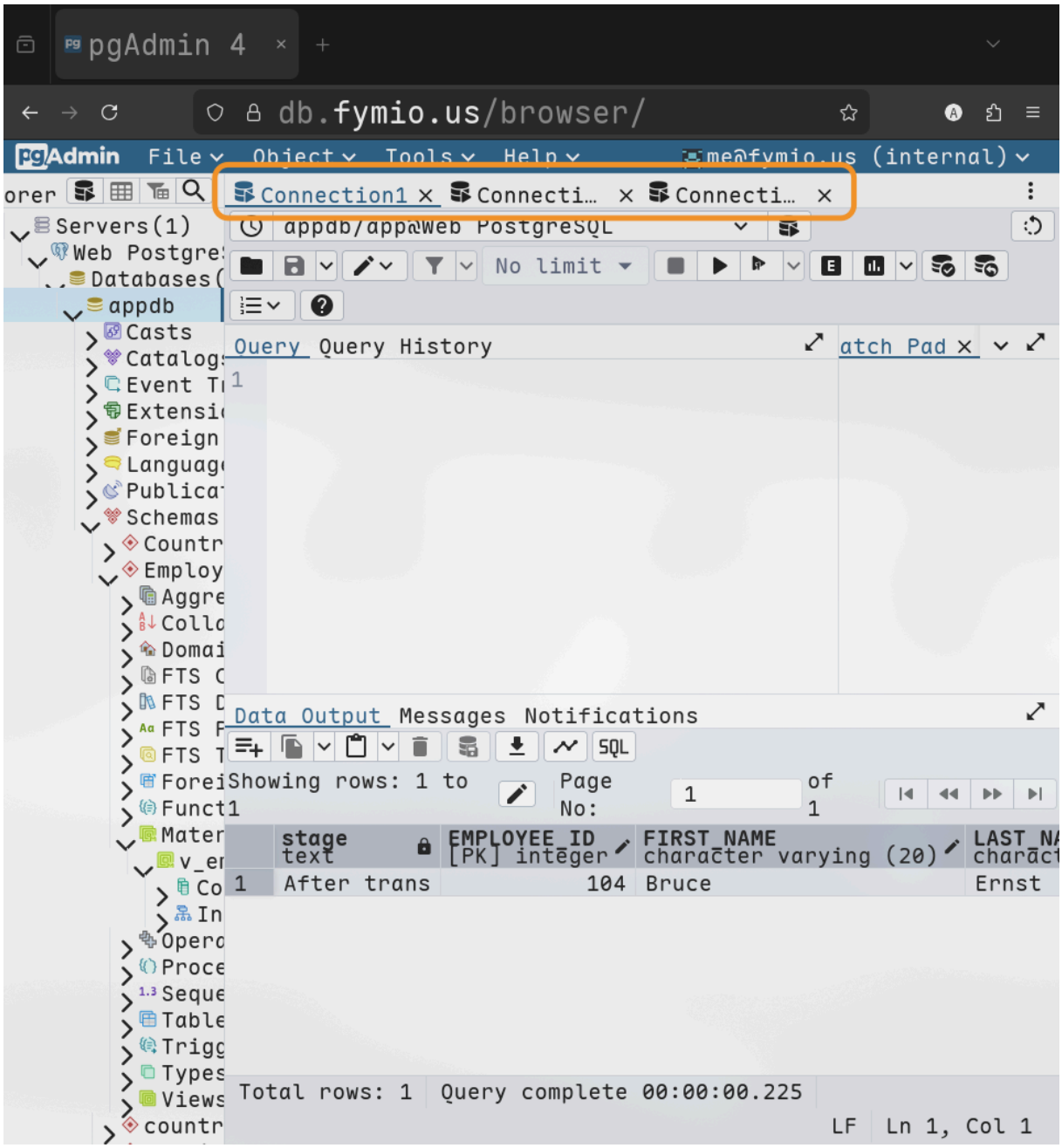

Рис. 27: Вкладки “Connections”

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

```sql
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.

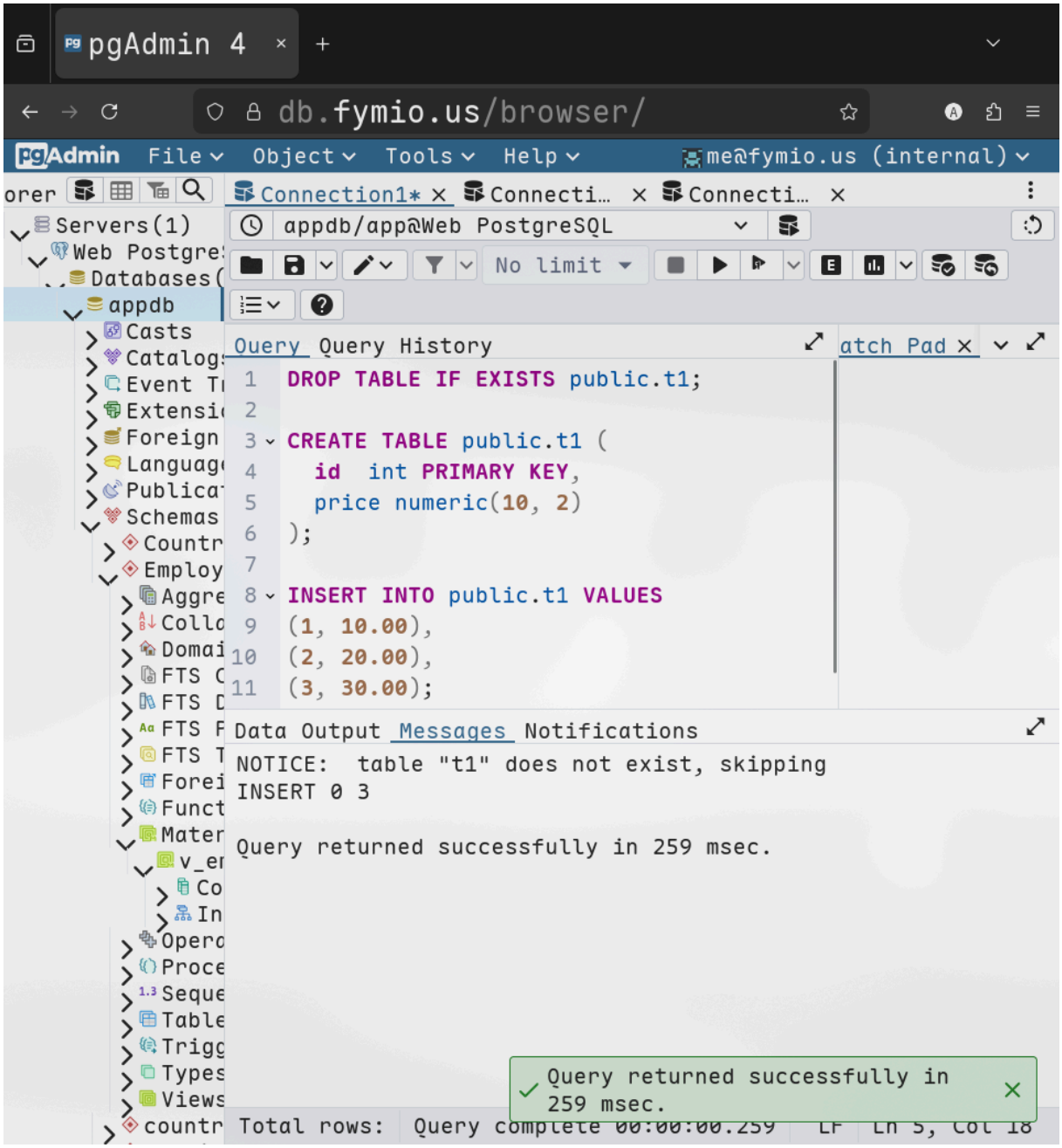

Рис. 28: Выполнение Скрипт. 26

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

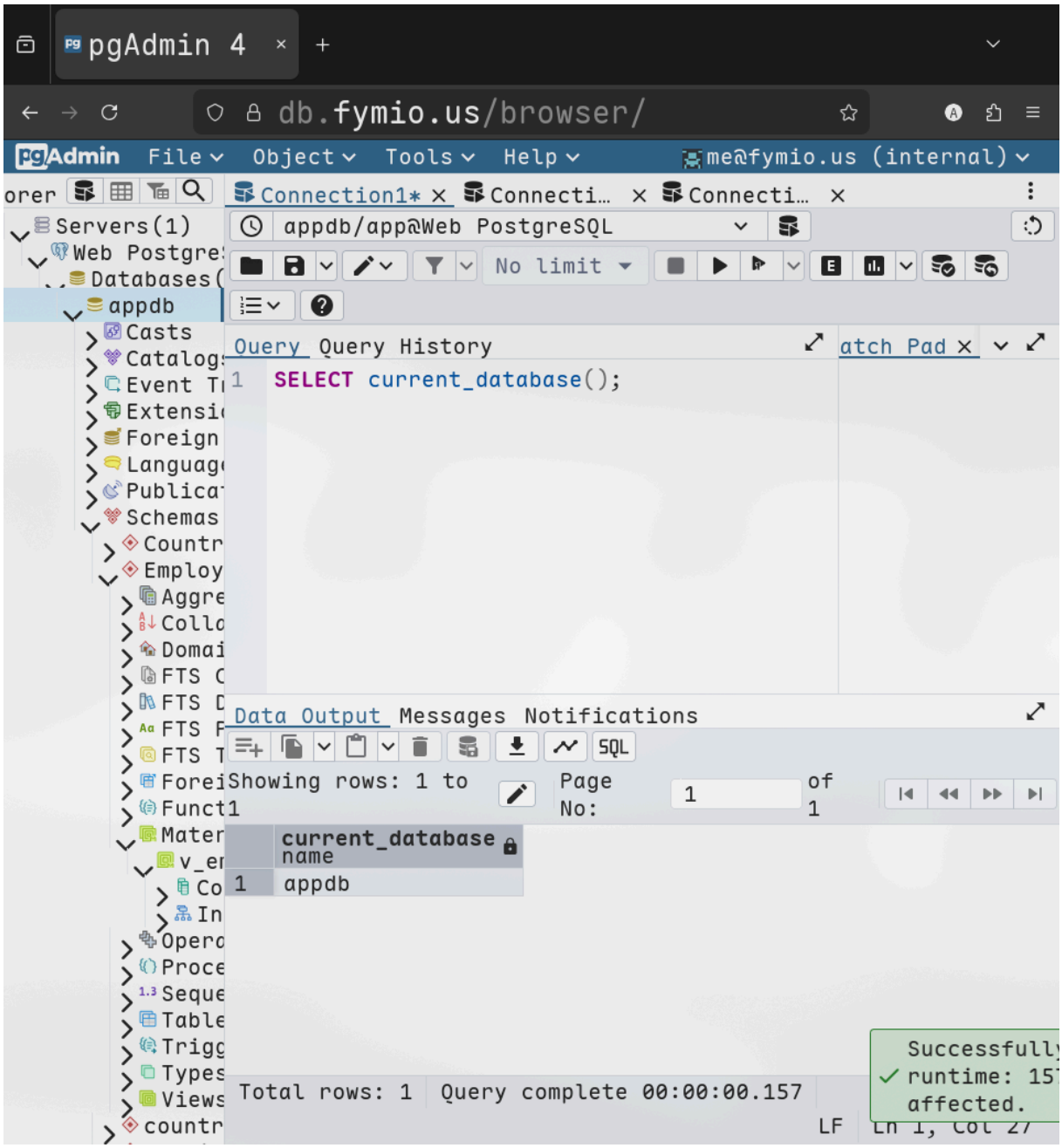

Рис. 29: Проверка подключения.

Затем я заблокировал первую строку при помощи Скрипт. 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.

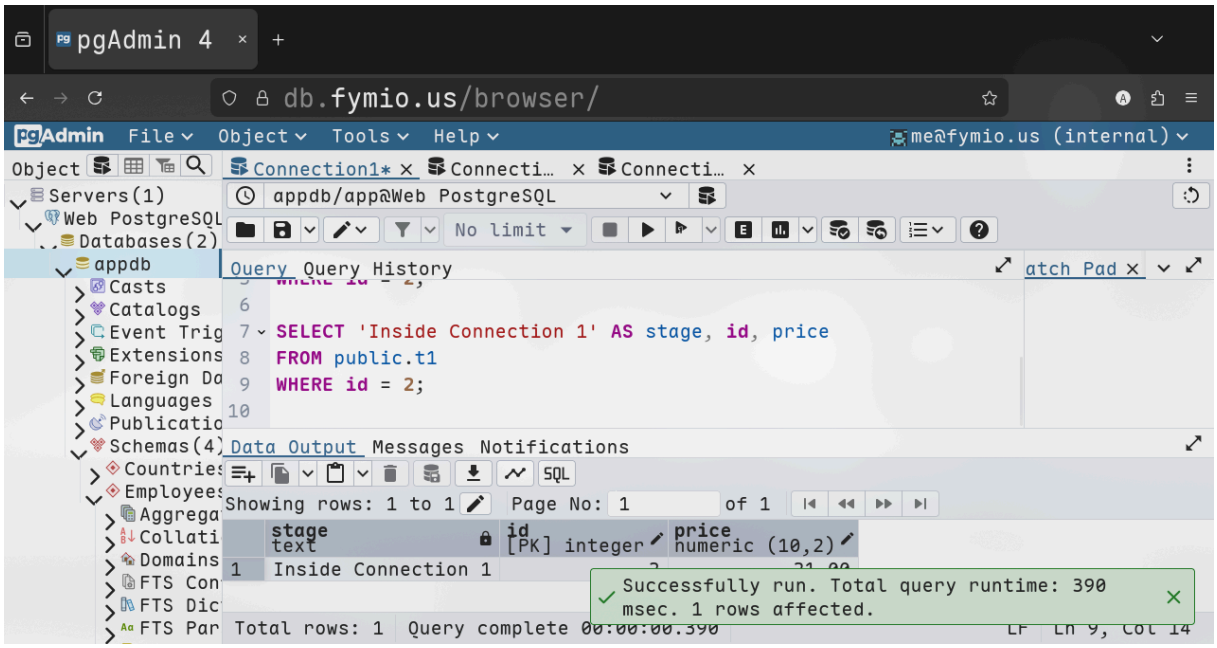

Рис. 30: Выполнение Скрипт. 27

Затем, из “Connection2” я попытался обновить ту же строку с помощью Скрипт. 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”.

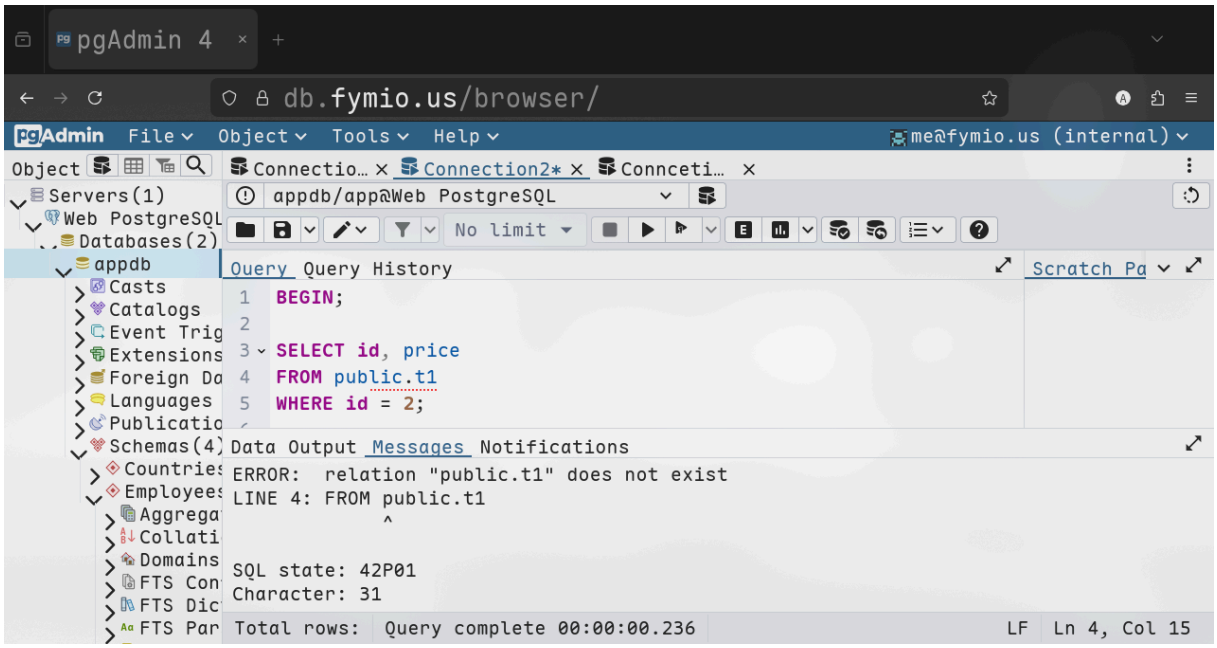

Рис. 31: Результат выполнения Скрипт. 28

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,
       usename 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”.

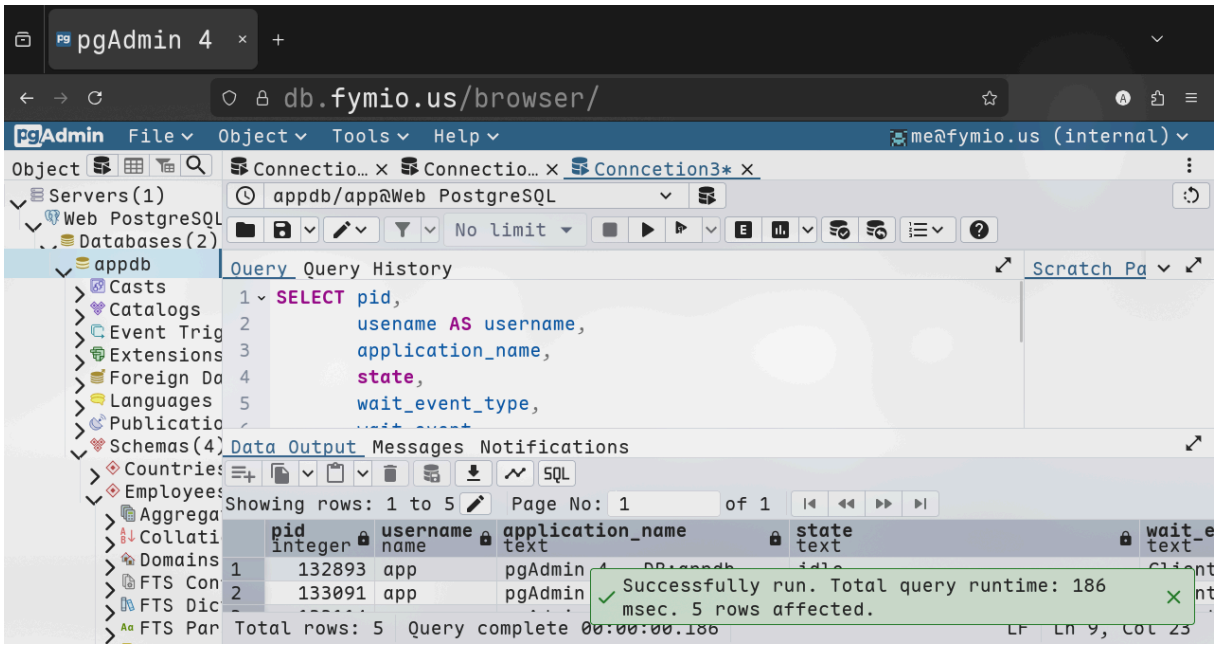

Рис. 32: Результат выполнения Скрипт. 29

- `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 Скрипт. 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`.

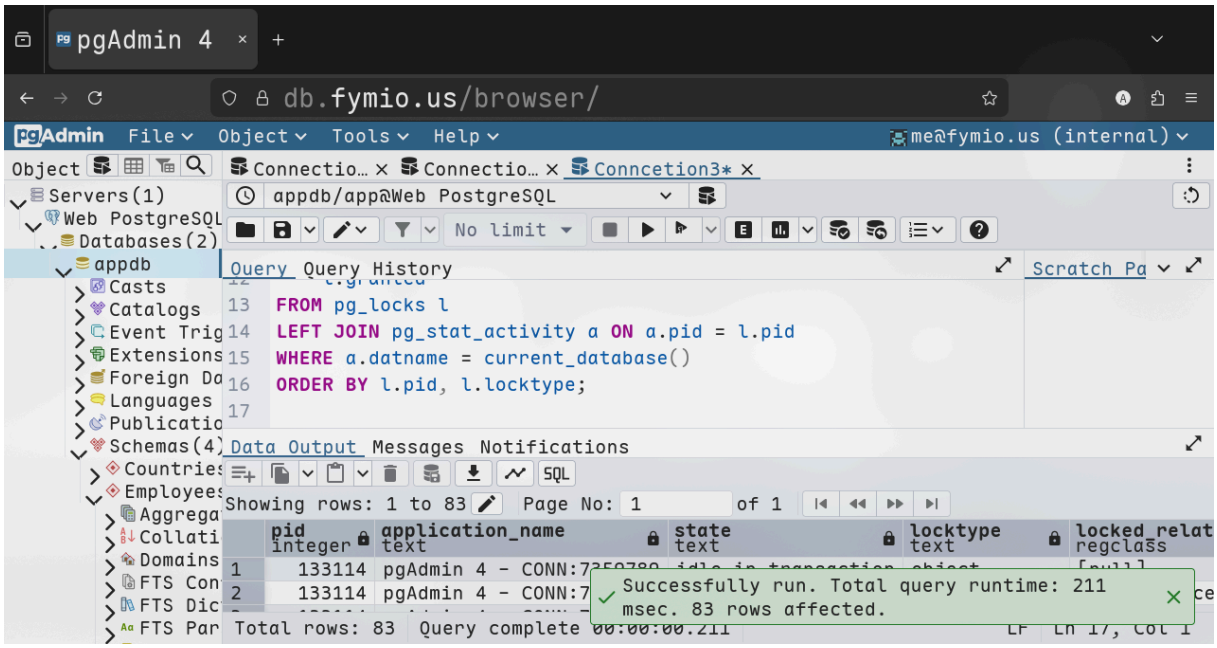

Рис. 33: Результат выполнения Скрипт. 30

- `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)

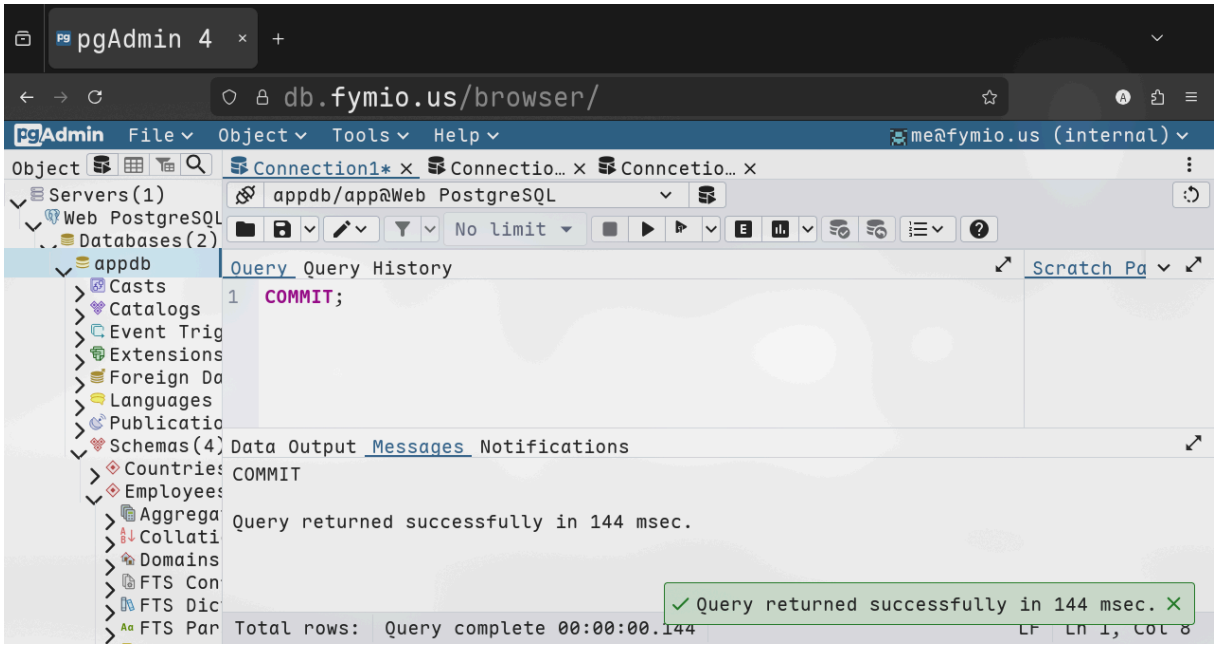

Рис. 34: Завершение транзакции.

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

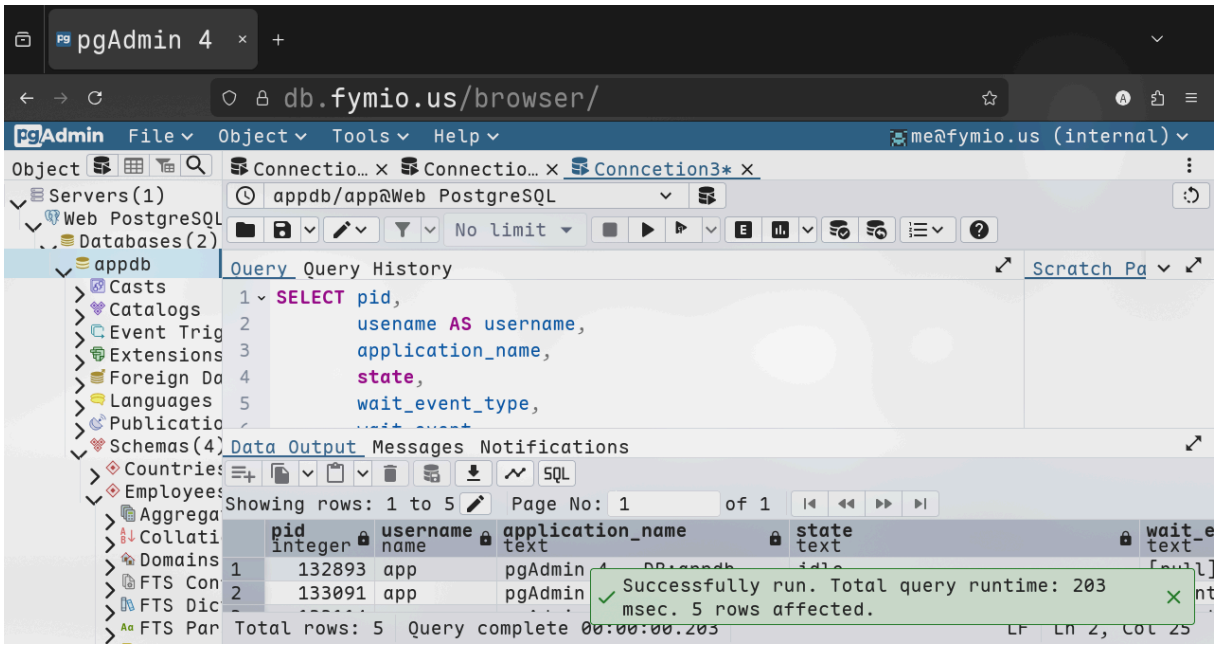

Рис. 35: Результат выполнения Скрипт. 29.

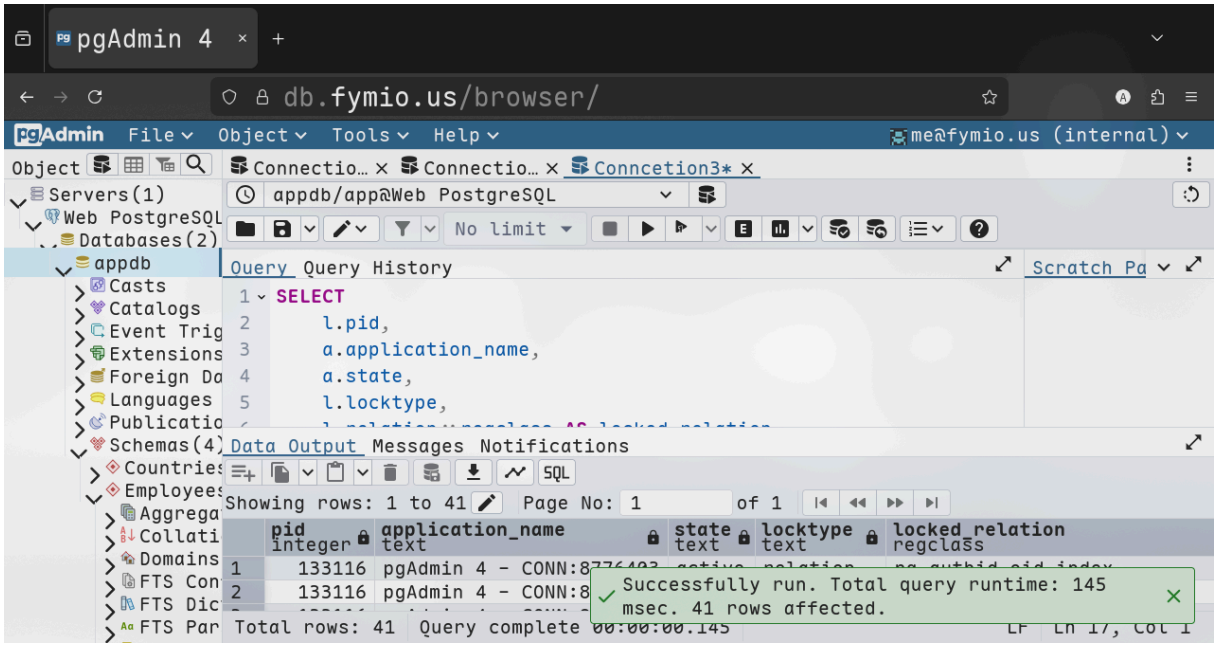

Рис. 36: Результат выполнения Скрипт. 30.

Задание 3. Уровени изоляции READ UNCOMMITTED и READ COMMITTED

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

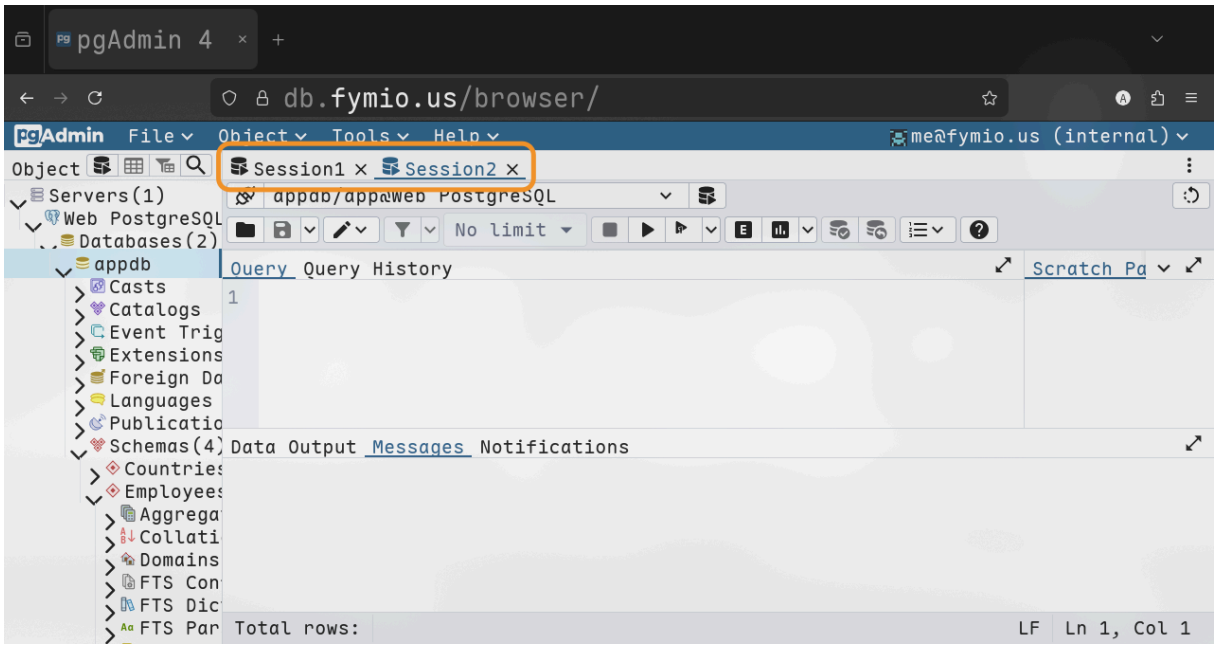

Рис. 37: Сессии “Sessions”.

В “Session1” я начал транзакцию и изменение данных при помощи Скрипт. 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.

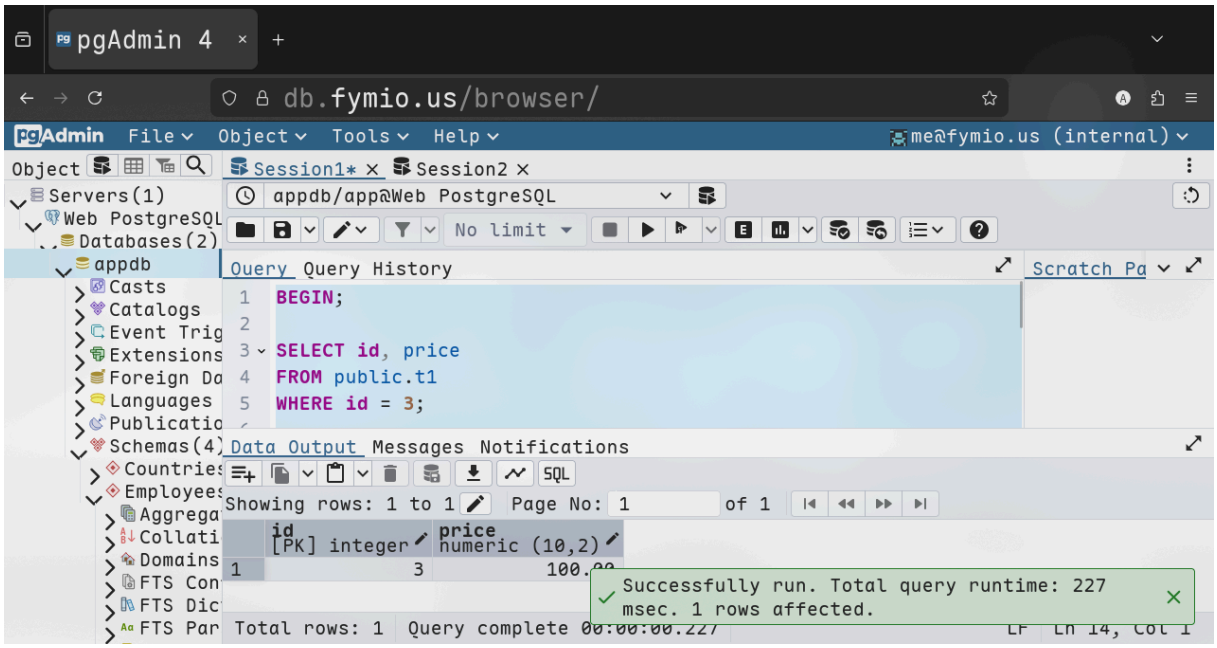

Рис. 38: Результат выполнения Скрипт. 31

- 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 COMMITED` 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 COMMITED`.

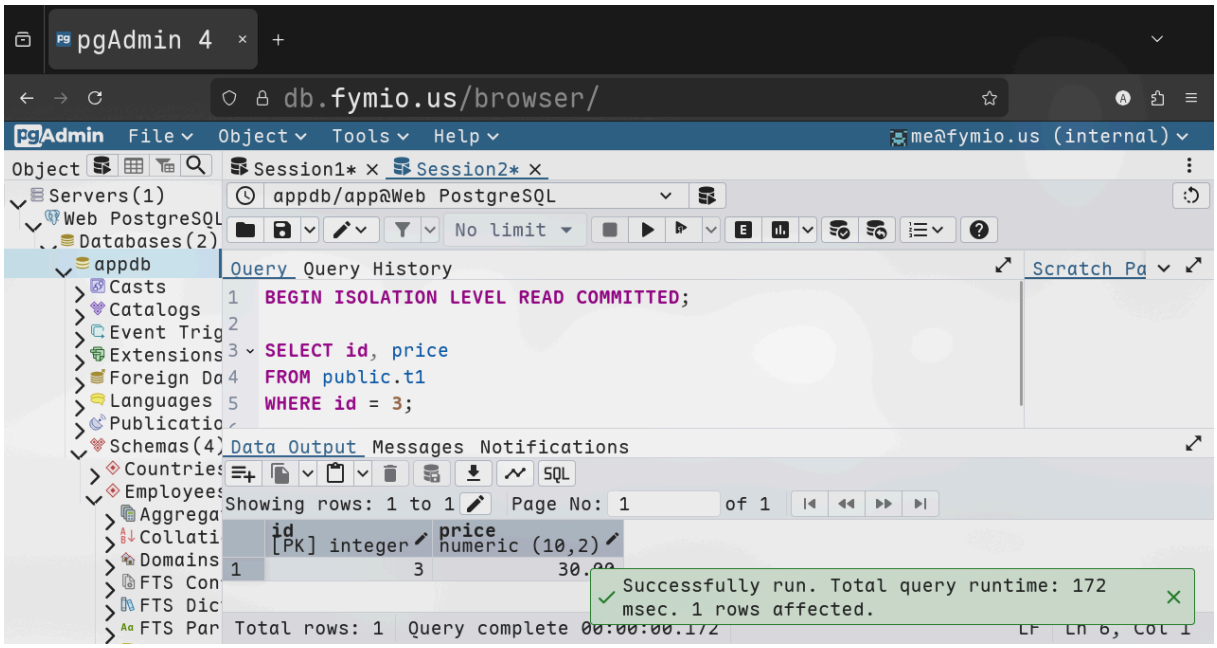

Рис. 39: Результат выполнения Скрипт. 32

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

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

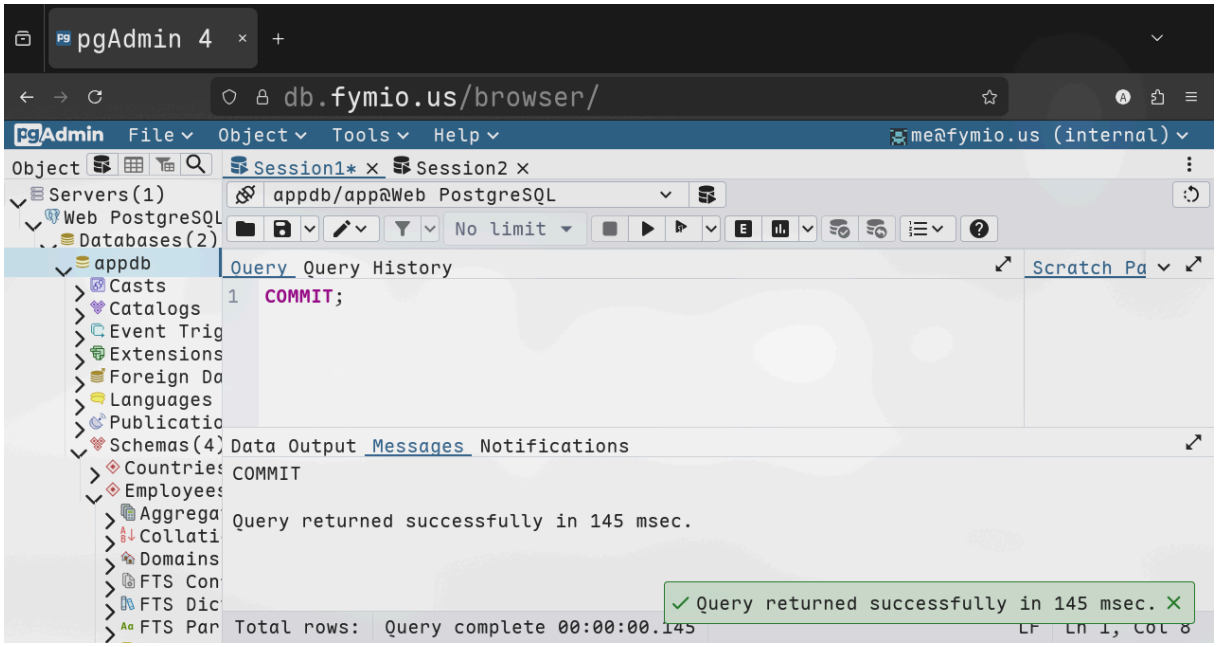

Рис. 40: Фиксирование изменений.

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

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

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

Скрипт. 33: Чтение в “Session2”.

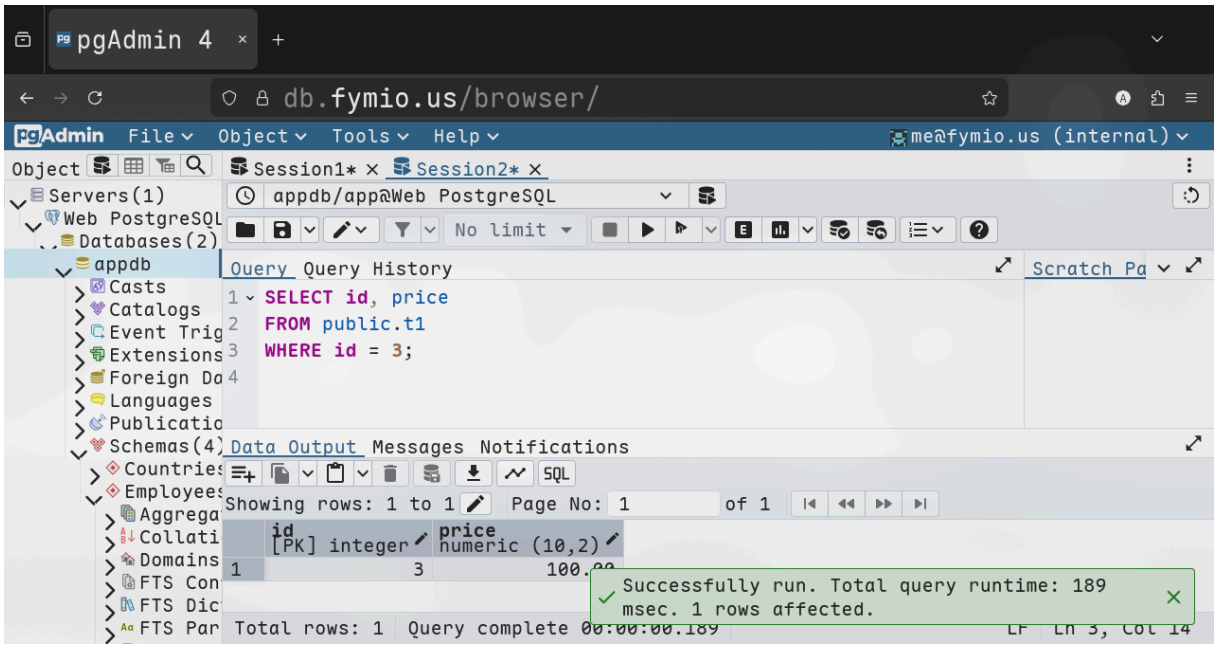

Рис. 41: Результат использования Скрипт. 33.

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

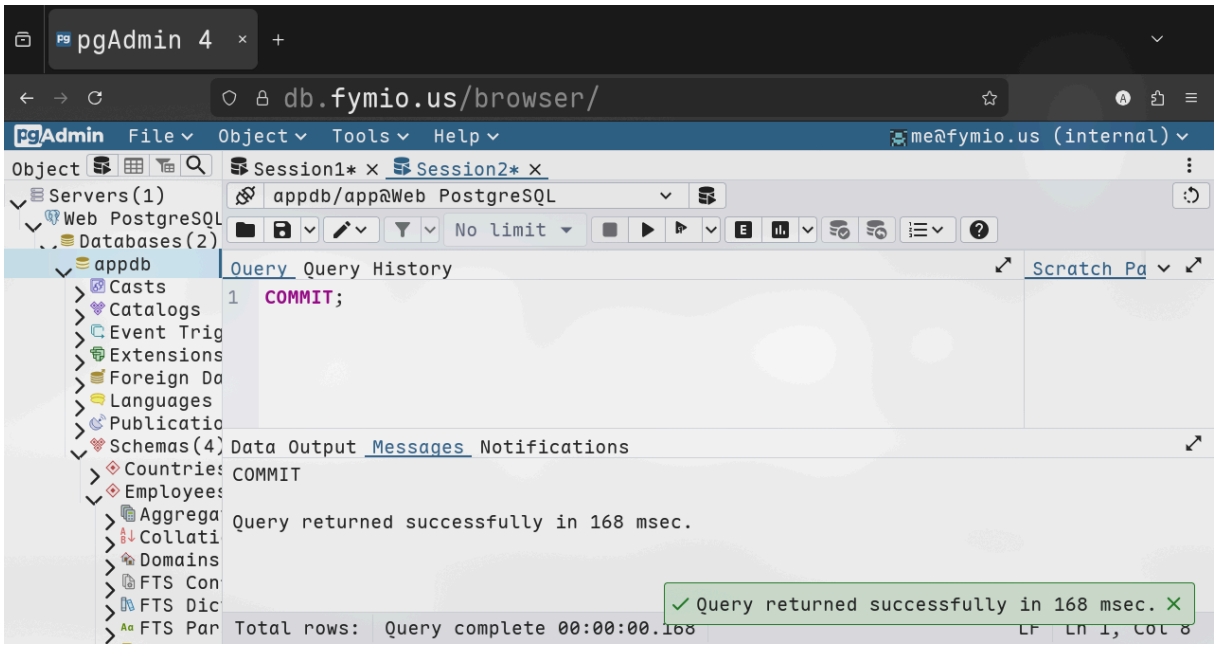

Рис. 42: Закрытие транзакции в “Session2”.

Задание 4. Уровень изоляции REPEATABLE READ

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

```
BEGIN ISOLATION LEVEL REPEATABLE READ;

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

Скрипт. 34: Запуск транзакции в “Session1”.

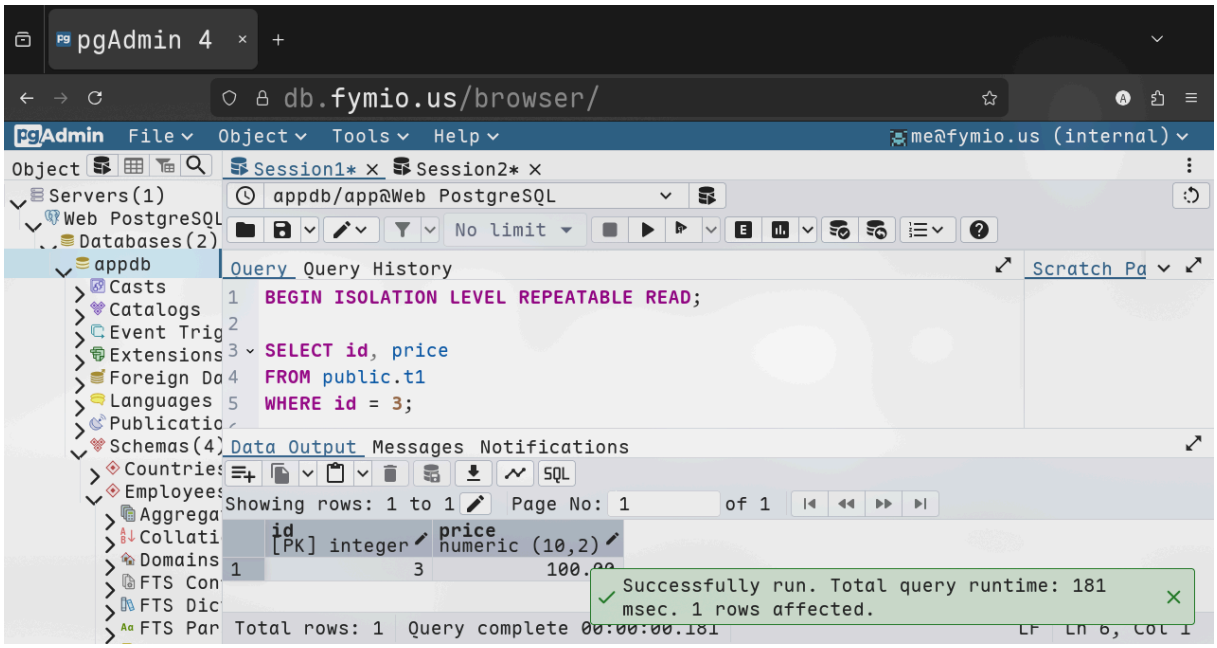

Рис. 43: Результат выполнения Скрипт. 34.

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”.

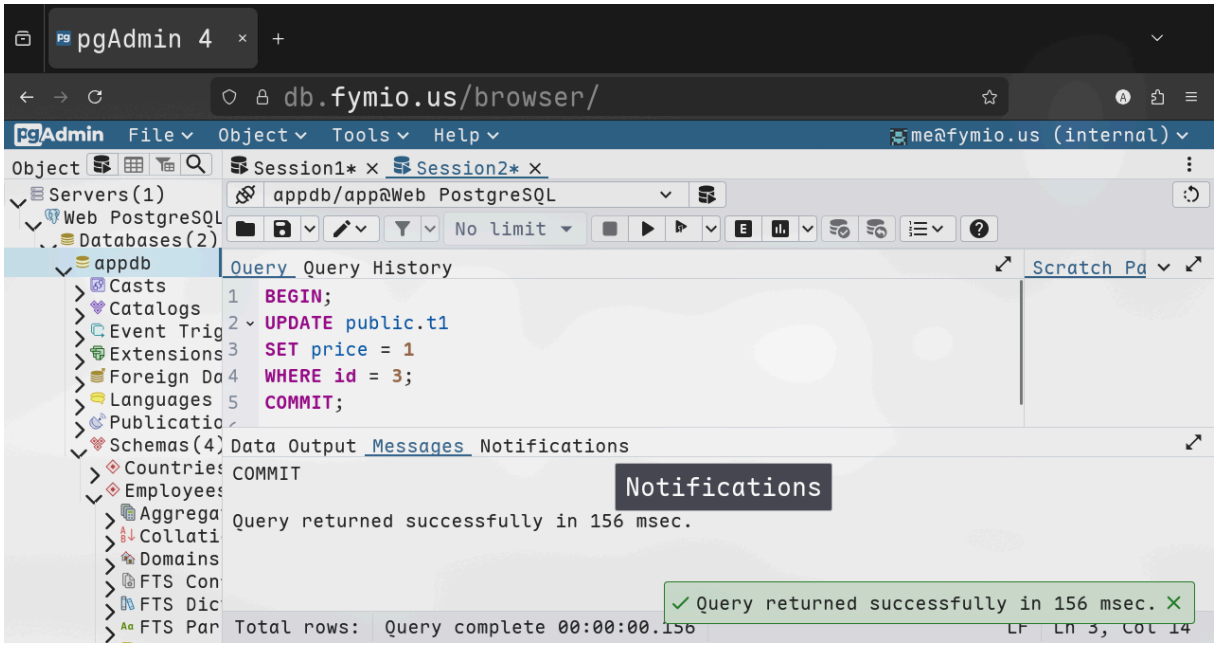

Рис. 44: Результат выполнения Скрипт. 35.

- `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”.

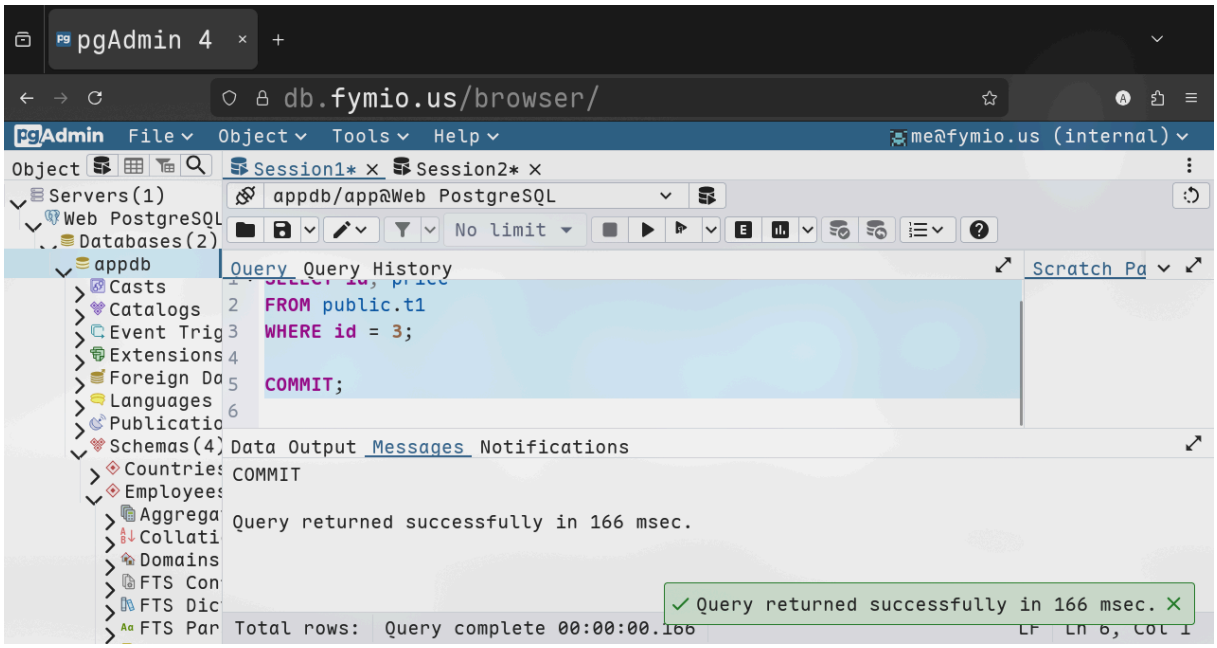

Рис. 45: Результат выполнения Скрипт. 36.

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

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

```
BEGIN ISOLATION LEVEL REPEATABLE READ;

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

Скрипт. 37: Начало транзакции в “Session1”

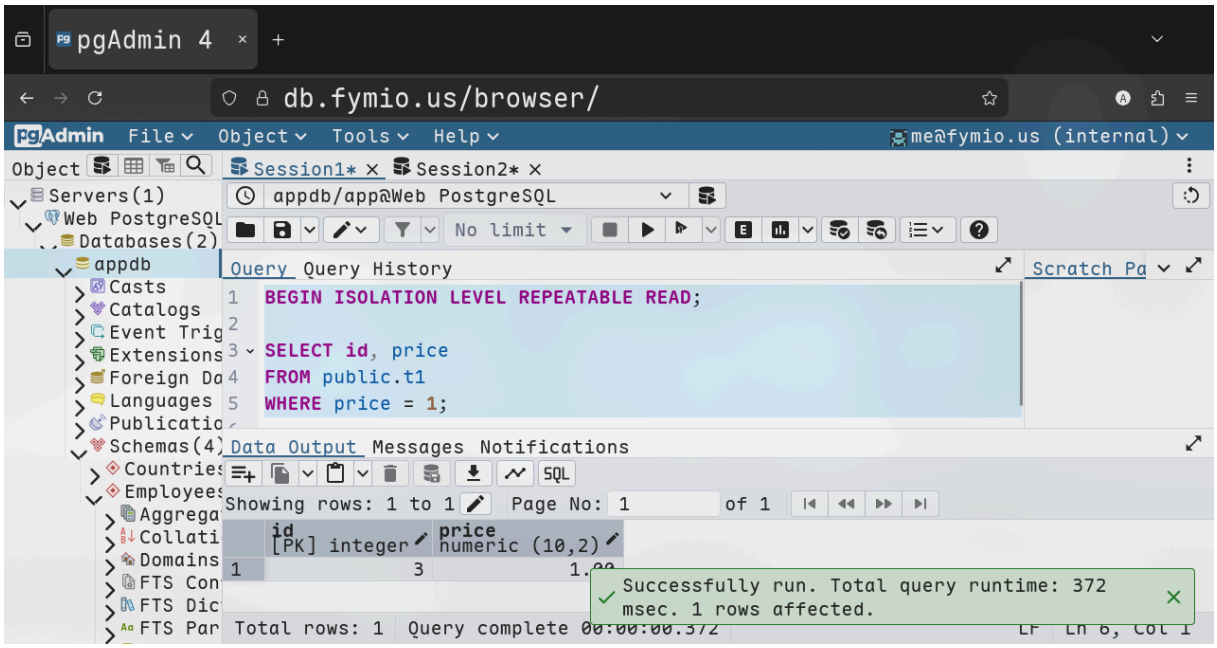

Рис. 46: Результат выполнения Скрипт. 37.

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.

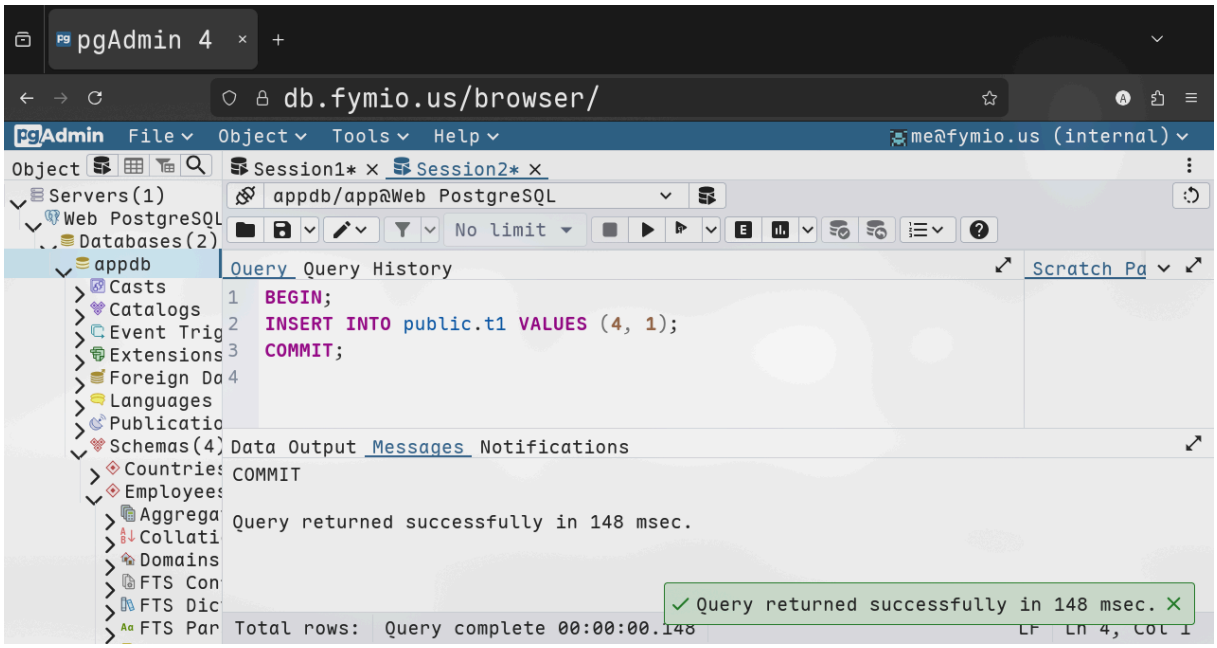

Рис. 47: Результат выполнения Скрипт. 38.

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”.

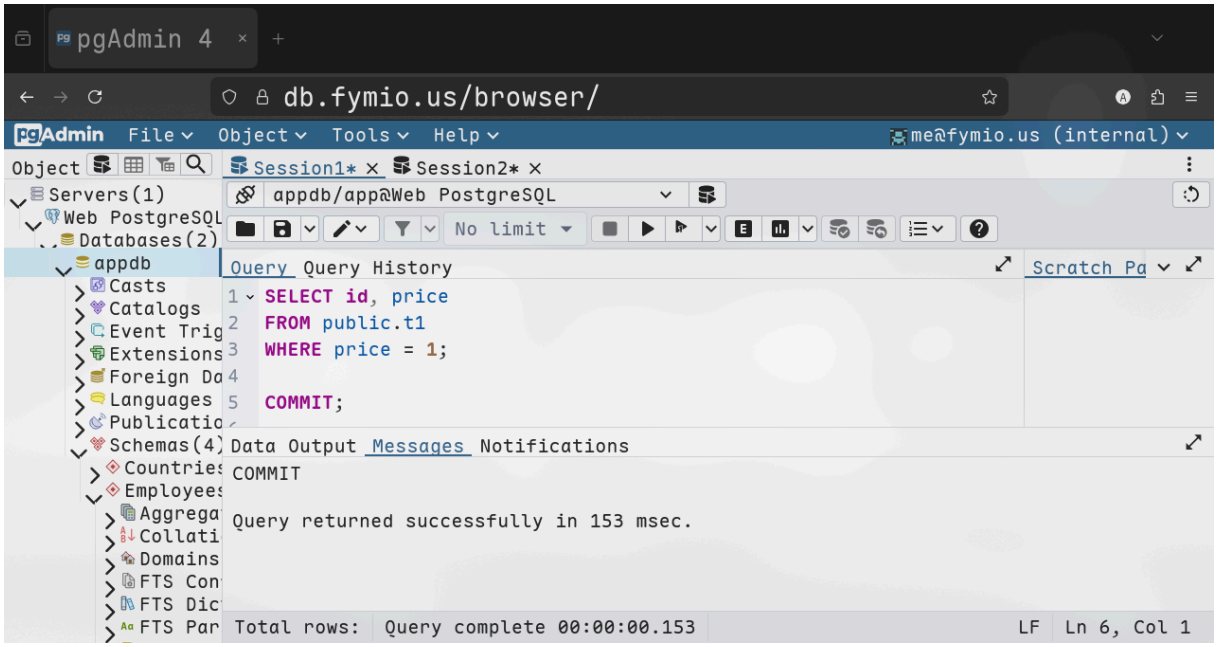

Рис. 48: Результат выполнения Скрипт. 39.

Задание 5. Уровень изоляции SERIALIZABLE

Я обновил строку в “Session1” и прочитал ее при помощи Скрипт. 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”.

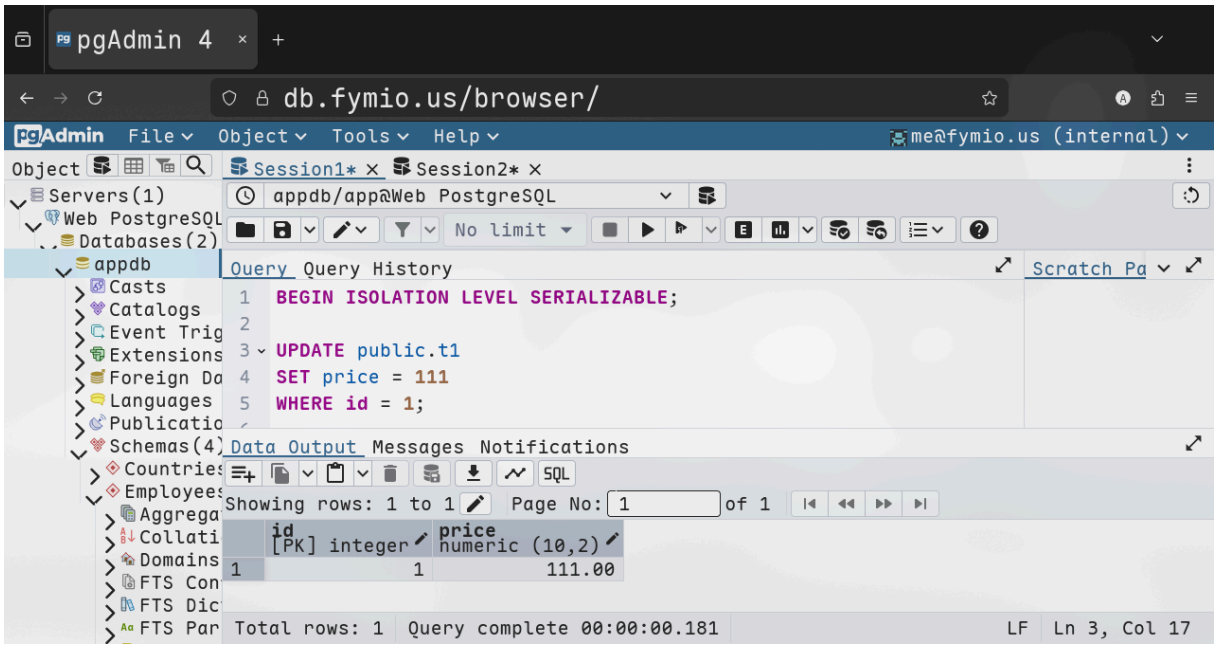

Рис. 49: Результат выполнения Скрипт. 40

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

Затем из “Session2” я попытался обновить ту же строку при помощи Скрипт. 41. (Рис. 50)

```
BEGIN ISOLATION LEVEL SERIALIZABLE;

UPDATE public.t1
SET price = 222
WHERE id = 1;

COMMIT;
```

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

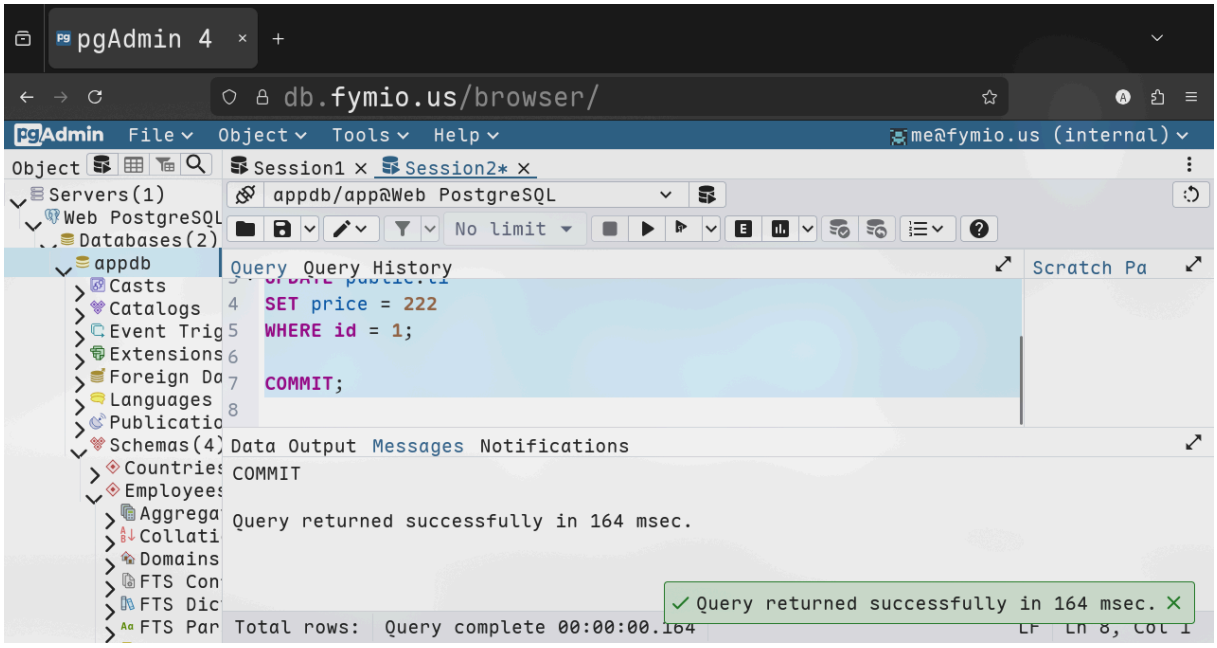

Рис. 50: Результат выполнения Скрипт. 41.

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

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

COMMIT;
```

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

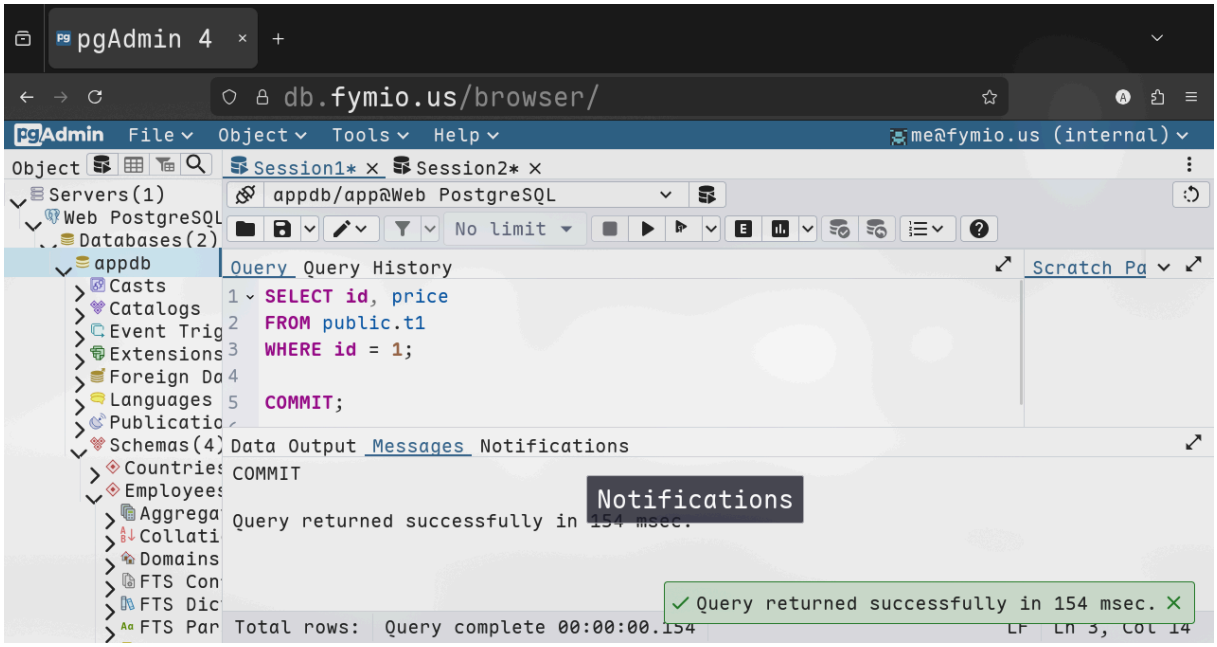

Рис. 51: Результат выполнения Скрипт. 42

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.