

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

Факультет инфокоммуникационных технологий

Направление подготовки 11.03.02

Практическая работа №4

Выполнил:

Дощенников Никита Андреевич

Группа: К3221

Проверила:

Татьяна Евгеньевна Войтюк

Санкт-Петербург
2025

Цель работы

Освоить создание и использование обычных и материализованных представлений, пользовательских функций и хранимых процедур.

Задачи, решаемые при выполнении работы

Создать обычные и материализованные представления, протестировать их работу, изучить влияние изменений в базовых таблицах, разработать пользовательские функции, создать и вызвать хранимые процедуры, проверить корректность их выполнения.

Исходные данные

СУБД PostgreSQL, база данных `appdb`, схема “EmployeesDepartments”, таблица “EMPLOYEES” с полями сотрудника.

Выполнение работы

Задание 1. Создание представления с помощью графического интерфейса

Я открыл окно создания представления в БД `appdb` в схеме “EmployeesDepartments”. (Рис. 1)

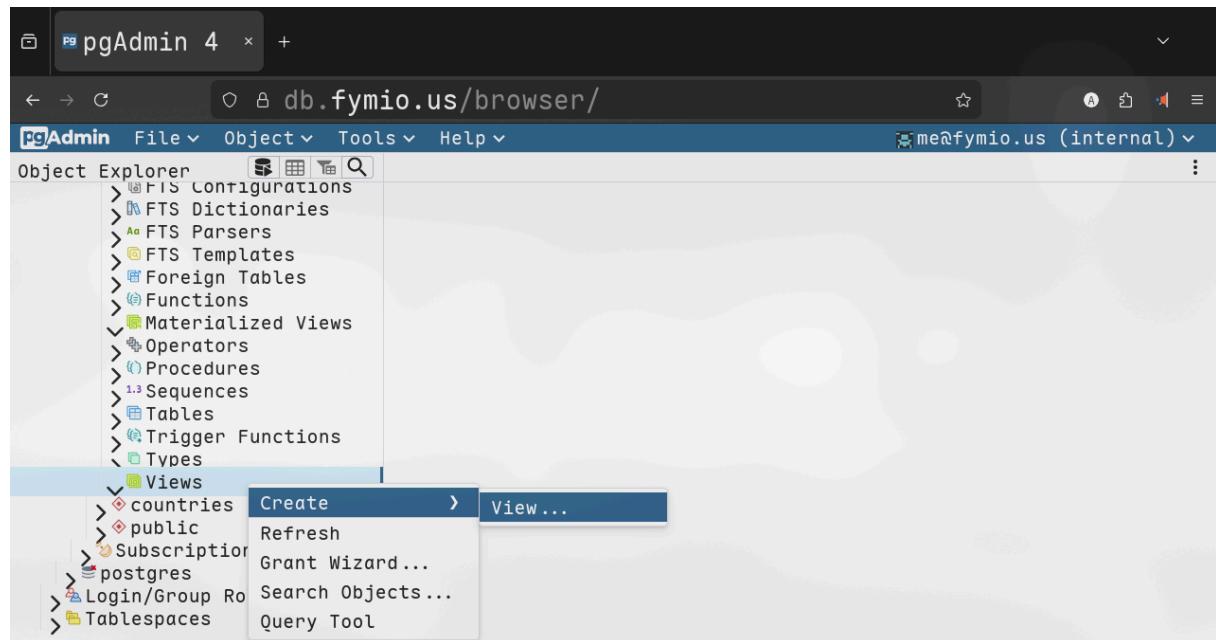


Рис. 1: Создание представления.

Ввел наименование представления `v_emp_active` на вкладке General. (Рис. 2)

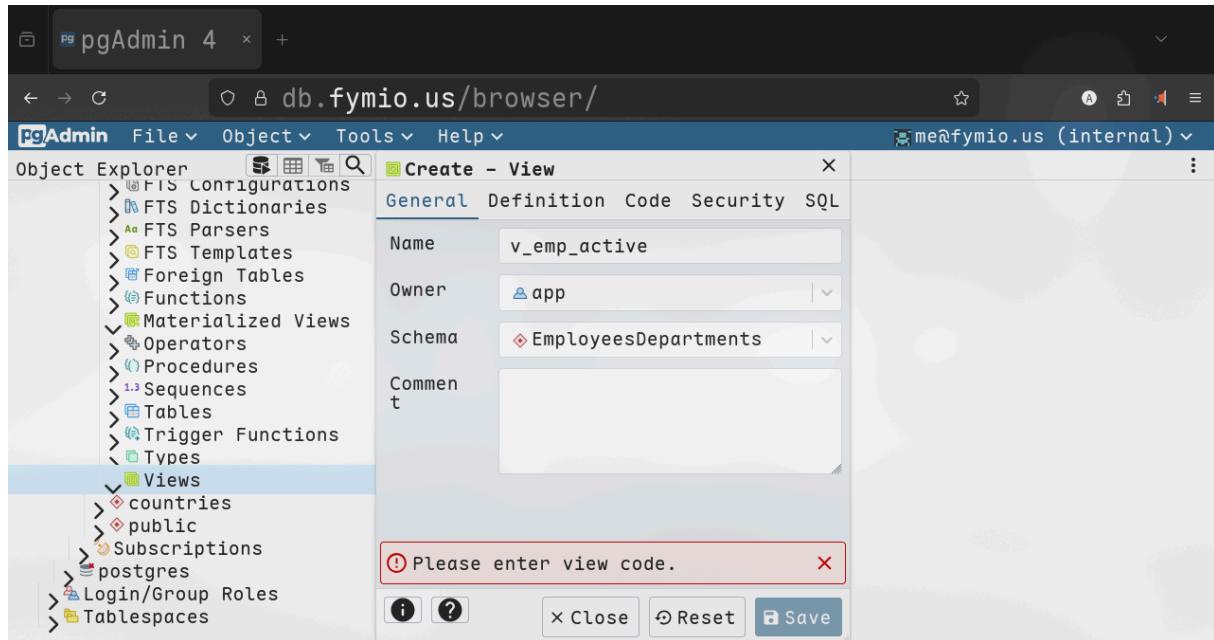


Рис. 2: Задание наименования представления.

На вкладке Code я ввел его определение. (Рис. 3)

```
SELECT "EMPLOYEE_ID",
       "FIRST_NAME",
       "LAST_NAME",
       "EMAIL",
       "JOB_ID",
       "SALARY",
       "DEPARTMENT_ID"
  FROM "EmployeesDepartments"."EMPLOYEES"
 WHERE "SALARY" > 5000;
```

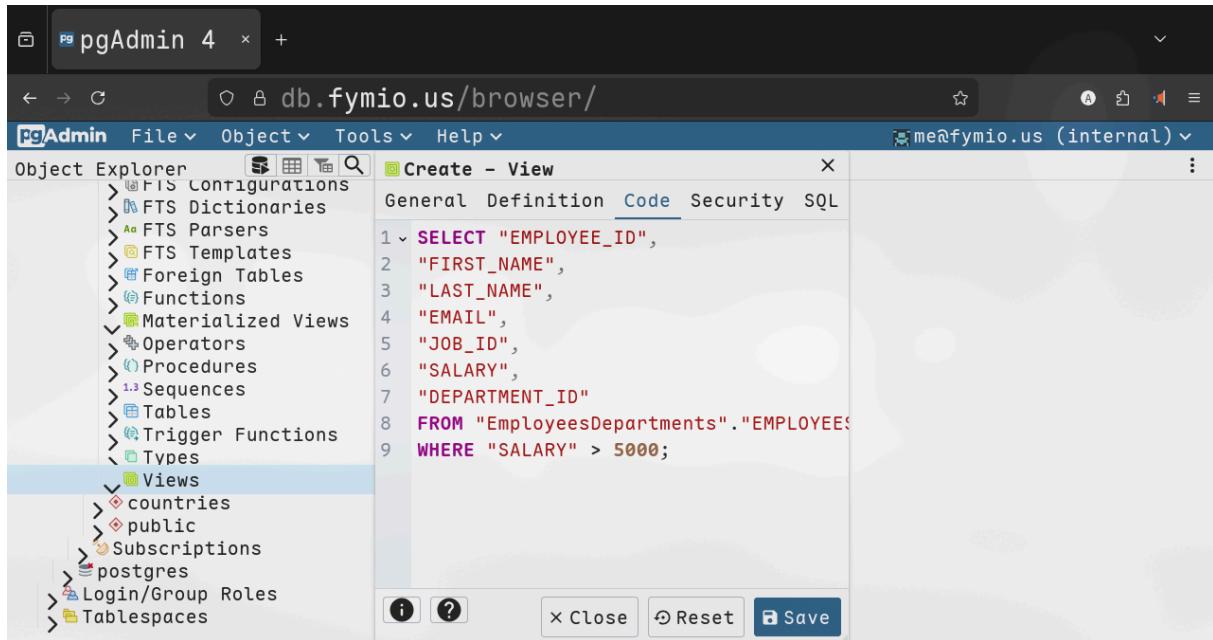


Рис. 3: Создание тела представления `v_emp_active`.

Затем я протестировал созданное представление командой ниже.
(Рис. 4)

```
SELECT * FROM "EmployeesDepartments".v_emp_active
```

Object Explorer		Query History	Data Output																								
Servers(1)	Web PostgreSQL	1 SELECT * FROM "EmployeesDepartments".v_emp_active	Showing rows: 1 to 20 Page No: 1 of 1																								
appdb	Databases(2)		<table border="1"> <thead> <tr> <th>EMPLOYEE_ID</th><th>FIRST_NAME</th><th>LAST_NAME</th><th>EMAIL</th></tr> </thead> <tbody> <tr><td>1</td><td>Steven</td><td>King</td><td>king@</td></tr> <tr><td>2</td><td>Neena</td><td>Kochhar</td><td>kochha</td></tr> <tr><td>3</td><td>Lex</td><td>De Haan</td><td>dehaar</td></tr> <tr><td>4</td><td>103</td><td></td><td>Successfully run. Total query runtime: 433 msec. 20 rows affected.</td></tr> <tr><td>5</td><td>104</td><td></td><td>Total rows: 20 Query complete 00:00:00.433</td></tr> </tbody> </table>	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	1	Steven	King	king@	2	Neena	Kochhar	kochha	3	Lex	De Haan	dehaar	4	103		Successfully run. Total query runtime: 433 msec. 20 rows affected.	5	104		Total rows: 20 Query complete 00:00:00.433
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL																								
1	Steven	King	king@																								
2	Neena	Kochhar	kochha																								
3	Lex	De Haan	dehaar																								
4	103		Successfully run. Total query runtime: 433 msec. 20 rows affected.																								
5	104		Total rows: 20 Query complete 00:00:00.433																								

Рис. 4: Тестирование представления `v_emp_active`.

Задание 2. Создание представления на основе представления и изменение базовых таблиц представлений.

Аналогичным образом в схеме “EmployeesDepartments” я создал еще одно представление, которое имеет название `V_EMP_ACTIVE_INFO` и предназначено для отбора сотрудников из отдела 80 с заработной платой выше 5000. Для этого я определил DDL оператор `CREATE` для создания нового представления. (Рис. 5)

```
CREATE OR REPLACE VIEW
"EmployeesDepartments"."V_EMP_ACTIVE_INFO" AS
SELECT "EMPLOYEE_ID",
       "FIRST_NAME" || ' ' || "LAST_NAME" AS FULL_NAME,
       "EMAIL",
       "JOB_ID",
       "SALARY",
       "DEPARTMENT_ID"
FROM "EmployeesDepartments".v_emp_active
WHERE "DEPARTMENT_ID" = 80;
```

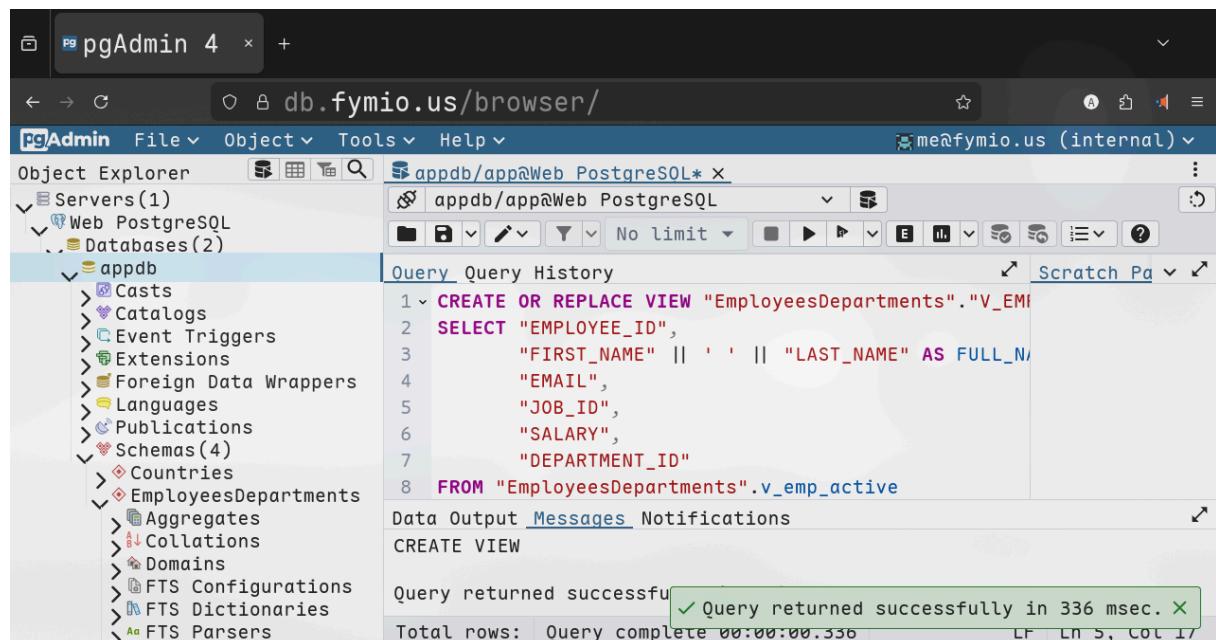


Рис. 5: Создание представления `V_EMP_ACTIVE_INFO`.

Затем, я протестировал созданное представление (Рис. 6)

```
SELECT * FROM "EmployeesDepartments"."V_EMP_ACTIVE_INFO"
```

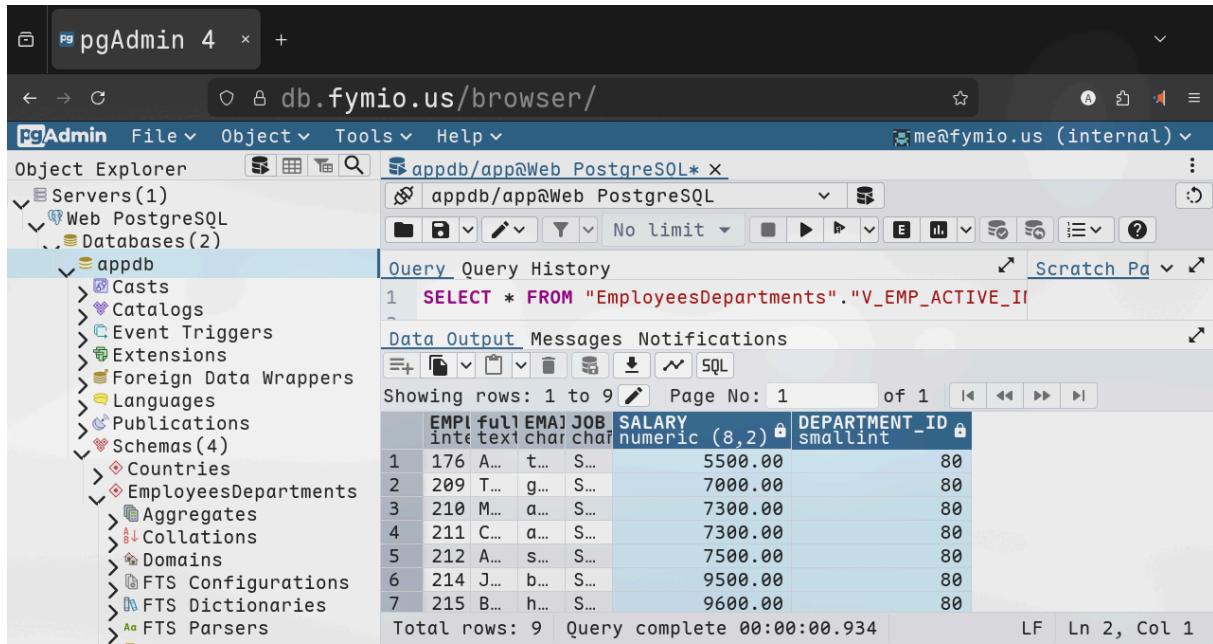


Рис. 6: Тестирование созданного представления.

После этого я провел дальнейшее тестирование представления `V_EMP_ACTIVE_INFO`. Для этого я добавил одну запись непосредственно в таблицу “EMPLOYEES”, а вторую запись добавил через представление `v_emp_active` и проверил, что представление `V_EMP_ACTIVE_INFO` отображает корректные данные. (Рис. 7)

```
INSERT INTO "EmployeesDepartments"."EMPLOYEES"
("FIRST_NAME", "LAST_NAME", "EMAIL", "JOB_ID", "SALARY",
"DEPARTMENT_ID")
VALUES('Nik', 'Nikov', 'Ni_Nik@itmo.ru', 'SA_REP', 6000, 80 );

INSERT INTO "EmployeesDepartments".v_emp_active
("FIRST_NAME", "LAST_NAME", "EMAIL", "JOB_ID", "SALARY",
"DEPARTMENT_ID")
VALUES('Sem', 'Semov', 'Se_Sem@itmo.ru', 'AD_VP', 22000, 80 );

SELECT * FROM "EmployeesDepartments"."V_EMP_ACTIVE_INFO"
```

```

1 ✓ INSERT INTO "EmployeesDepartments"."EMPLOYEES"
2 ("FIRST_NAME", "LAST_NAME", "EMAIL", "JOB_ID", "SAL)
3 VALUES('NIK', 'Nikov', 'Ni_Nik@itmo.ru', 'SA_REP',
4
5 ✓ INSERT INTO "EmployeesDepartments".v_emp_active
6 ("FIRST_NAME", "LAST_NAME", "EMAIL", "JOB_ID", "SAL)
7 VALUES('Sem', 'Semov', 'Se_Sem@itmo.ru', 'AD_VP', 2
8
9 SELECT * FROM "EmployeesDepartments".V_EMP_ACTIVE_INFO

```

Data Output Messages Notifications
Successfully run. Total query runtime: 379 msec. 11 rows affected.
Showing rows 1 to 11 Total rows: 11 Query complete 00:00:00.379

Рис. 7: Тестируем представление V_EMP_ACTIVE_INFO.

Затем, я попробовал внести изменения в таблицу “EMPLOYEES” с помощью кода ниже (Рис. 8)

```

ALTER TABLE "EmployeesDepartments"."EMPLOYEES"
ALTER COLUMN "FIRST_NAME" TYPE varchar(100);

```

```

1 ✓ ALTER TABLE "EmployeesDepartments"."EMPLOYEES"
2 ALTER COLUMN "FIRST_NAME" TYPE varchar(100);


```

Error: rule _RETURN on view "EmployeesDepartments".v_emp_active depends on column "FIRST_NAME" cannot alter type of a column used by a view or rule

Error: cannot alter type of a column used by a view or rule
SQL state: 0A000
Detail: rule _RETURN on view "EmployeesDepartments".v_emp_active depends on column "FIRST_NAME"

Total rows: 11 Query complete 00:00:00.671

Рис. 8: Попытка изменения таблицы “EMPLOYEES”.

Я получил ошибку (Рис. 8), которая говорит о том, что мы не можем менять тип столбца, используемого представлением.

Задание 3. Создание материализованного представления

Я пересоздал существующее представление с использованием директивы `MATERIALIZED`. (Рис. 9)

```
DROP VIEW IF EXISTS "EmployeesDepartments".v_emp_active
CASCADE;

CREATE MATERIALIZED VIEW "EmployeesDepartments".v_emp_active
AS
SELECT "EMPLOYEE_ID",
       "FIRST_NAME",
       "LAST_NAME",
       "EMAIL",
       "JOB_ID",
       "SALARY",
       "DEPARTMENT_ID"
  FROM "EmployeesDepartments"."EMPLOYEES"
 WHERE "SALARY" > 5000;
```

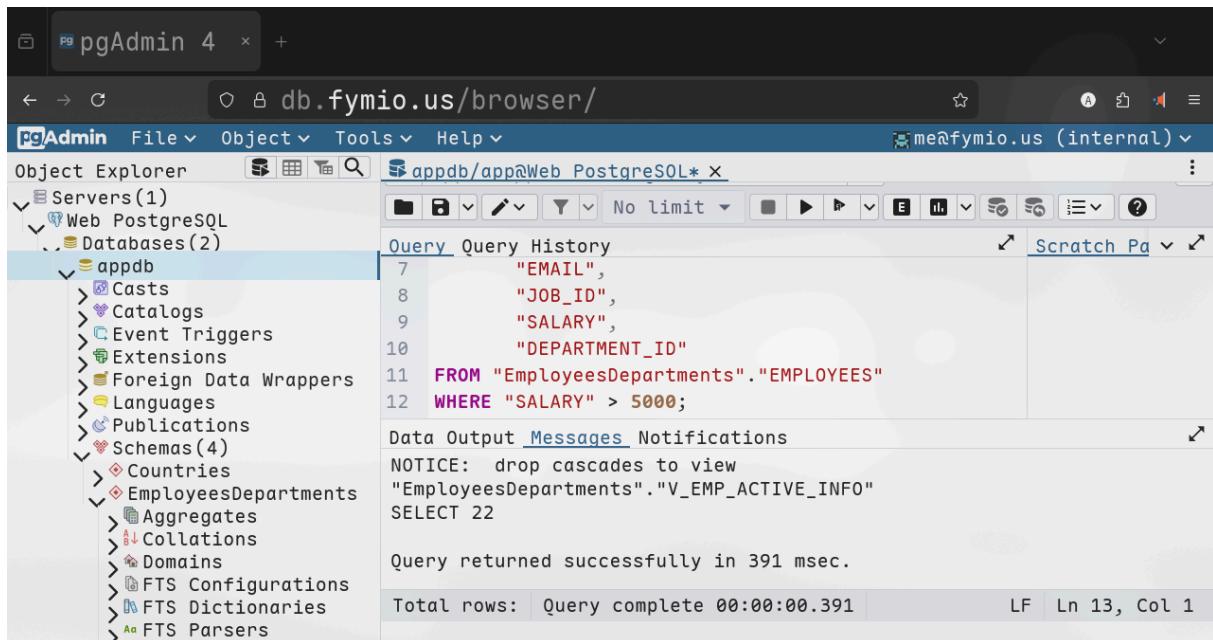


Рис. 9: Создание материализованного представления.

Затем я проверил новое представление (Рис. 10):

```
SELECT * FROM "EmployeesDepartments".v_emp_active WHERE
LAST_NAME = "Ivanov";
```

```

pgAdmin 4
db.fymio.us/browser/
Object Explorer
Servers(1)
Web PostgreSQL
Databases(2)
appdb
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas(4)
Countries
EmployeesDepartments
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
Query History
SELECT *
FROM "EmployeesDepartments".v_emp_active
WHERE "LAST_NAME" = 'Petrov';
Data Output Messages Notifications
Showing rows: 1 to 1 Page No: 1 of 1
EMPLOYEE_ID FIRST_NAME LAST_NAME EMAIL
integer character varying (20) character varying (25) character
1 2 Petr Petrov Pe_Ptr
Total rows: 1 Query complete 00:00:00.270
Successfully run. Total query runtime: 270 msec. 1 rows affected.

```

Рис. 10: Проверка представления `v_emp_active`.

Затем, я обновил имя сотрудника с фамилией “Petrov” в таблице “Employees”, установив ему имя “Vlad”. (Рис. 11)

```

UPDATE "EmployeesDepartments"."EMPLOYEES"
SET "FIRST_NAME" = 'Vlad'
WHERE "LAST_NAME" = 'Petrov';

```

```

pgAdmin 4
db.fymio.us/browser/
Object Explorer
Servers(1)
Web PostgreSQL
Databases(2)
appdb
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas(4)
Countries
EmployeesDepartments
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
Query History
UPDATE "EmployeesDepartments"."EMPLOYEES"
SET "FIRST_NAME" = 'Vlad'
WHERE "LAST_NAME" = 'Petrov';
Data Output Messages Notifications
UPDATE 1
Query returned successfully in 357 msec.

```

Рис. 11: Обновление имени сотрудника.

Проверим, что данные обновились в таблице “EMPLOYEES”. (Рис. 12)

```
SELECT * FROM "EmployeesDepartments"."EMPLOYEES" WHERE  
"LAST_NAME" = 'Petrov';
```

The screenshot shows the pgAdmin 4 interface. In the Object Explorer, under 'Servers(1) > Web PostgreSQL > Databases(2) > appdb > Schemas(4) > EmployeesDepartments', the 'Query' tab is selected. The query window contains:

```
1 < SELECT * FROM "EmployeesDepartments"."EMPLOYEES"  
2 WHERE "LAST_NAME" = 'Petrov';
```

The 'Data Output' tab shows the results of the query:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL
1	2	Vlad	Petrov

Total rows: 1 Query complete 00:00:00.602

Рис. 12: Проверка обновления данных в таблице.

Чтобы получить актуальную информацию из материализованного представления, я обновил его с помощью команды
REFRESH MATERIALIZED VIEW . (Рис. 13)

```
REFRESH MATERIALIZED VIEW "EmployeesDepartments".v_emp_active;  
SELECT * FROM "EmployeesDepartments".v_emp_active  
WHERE "LAST_NAME" = 'Petrov';
```

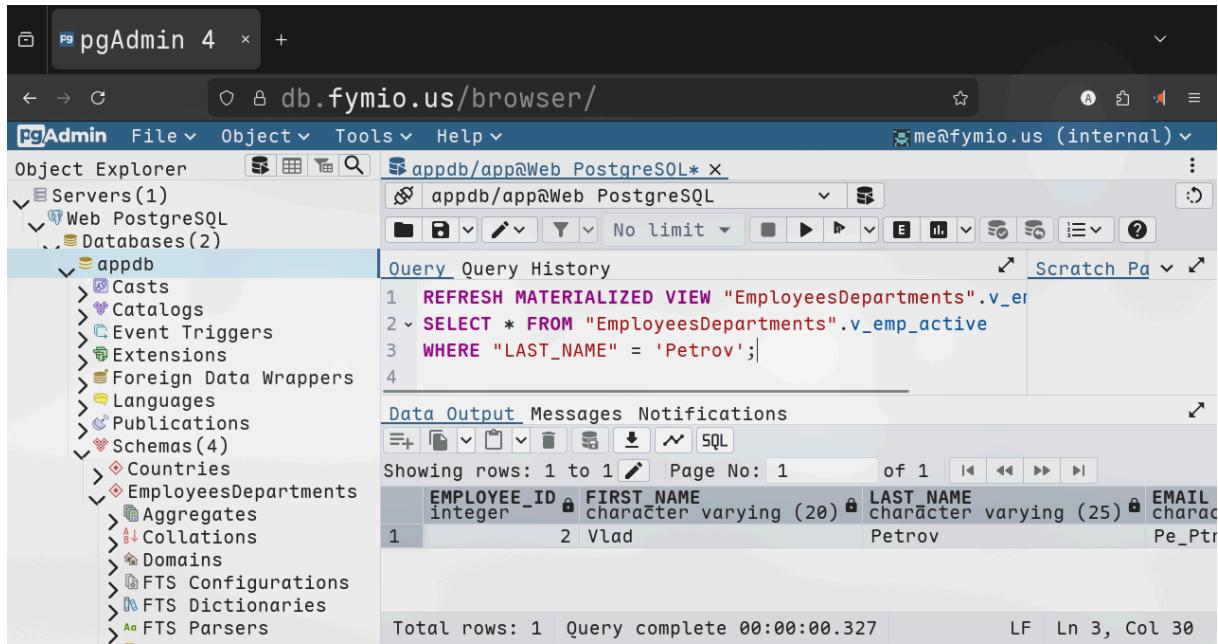


Рис. 13: Обновление представления.

Обычное представление не хранит данные, а каждый раз вычисляет результат заново, поэтому оно всегда показывает актуальную информацию. Материализованное представление хранит готовые данные, работает быстрее, но может содержать устаревшие значения и требует обновления. Обычное представление используют, когда важна актуальность данных, а материализованное — когда запрос тяжёлый, данные меняются редко и нужна высокая скорость чтения. (Рис. 14)

Задание 4. Создание скалярной пользовательской функции

В окне Query Tool я создал функцию, которая по идентификатору сотрудника, возвращает его полное имя, т.е. имя, соединенное с фамилией, через пробел. Если сотрудник не найден, то функция должна вернуть значение `NULL`.

```

CREATE OR REPLACE FUNCTION
"EmployeesDepartments".get_employee_full_name(p_emp_id INT)
RETURNS TEXT AS
$$
DECLARE
    v_full_name TEXT;
BEGIN
    SELECT "FIRST_NAME" || ' ' || "LAST_NAME"
    INTO v_full_name
    
```

```

    FROM      "EmployeesDepartments"."EMPLOYEES"
    WHERE     "EMPLOYEE_ID" = p_emp_id;

    RETURN v_full_name;
END;
$$ LANGUAGE plpgsql;

```

The screenshot shows the pgAdmin 4 interface. On the left, the Object Explorer tree view displays the database structure under 'appdb'. A selected node is 'EmployeesDepartments'. In the main pane, a query editor window titled 'Query' contains the SQL code for creating a scalar function:

```

CREATE FUNCTION v_full_name(p_emp_id integer)
RETURNS TEXT
AS $$
DECLARE
    v_full_name TEXT;
BEGIN
    SELECT "FIRST_NAME" || ' ' || "LAST_NAME"
    INTO v_full_name
    FROM "EmployeesDepartments"."EMPLOYEES"
    WHERE "EMPLOYEE_ID" = p_emp_id;
    RETURN v_full_name;
END;
$$ LANGUAGE plpgsql;

```

Below the code, the status bar indicates: 'Query returned successfully in 732 msec.' and 'Total rows: 1 Query complete 00:00:00.732'.

Рис. 14: Создание скалярной функции.

Затем я протестировал работоспособность скалярной функции, передав в нее известный мне идентификатор сотрудника. (Рис. 15)

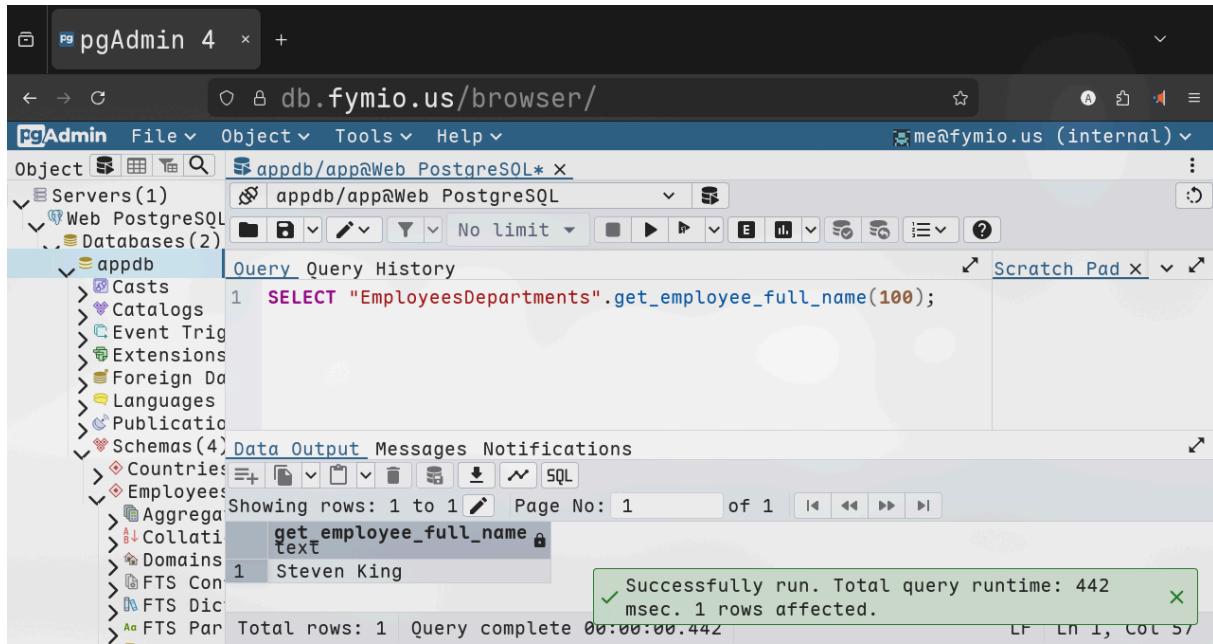


Рис. 15: Тестирование скалярной функции.

Задание 5. Создание собственной скалярной функции

Я создал собственную функцию (Рис. 16)

```

CREATE OR REPLACE FUNCTION
"EmployeesDepartments".get_salary_by_mode(p_emp_id INT, p_mode
TEXT)
RETURNS NUMERIC AS
$$
DECLARE
    v_salary NUMERIC;
BEGIN
    SELECT "SALARY" INTO v_salary
    FROM "EmployeesDepartments"."EMPLOYEES"
    WHERE "EMPLOYEE_ID" = p_emp_id;

    IF v_salary IS NULL THEN
        RETURN NULL;
    END IF;

    IF upper(p_mode) = 'ROUND' THEN
        RETURN round(v_salary);
    ELSIF upper(p_mode) = 'EXACT' THEN
        RETURN v_salary;
    ELSE
        RETURN v_salary;
    END IF;

```

```

END IF;
END;
$$ LANGUAGE plpgsql;

```

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

- Servers:** appdb/app@Web PostgreSQL
- Databases:** appdb
- Query History:**

```

CREATE OR REPLACE FUNCTION "EmployeesDepartments".get_salary_by_mode()
RETURNS NUMERIC AS
$$
DECLARE
    v_salary NUMERIC;
BEGIN
    SELECT "SALARY" INTO v_salary
    FROM "EmployeesDepartments"."EMPLOYEES"
    WHERE "EMPLOYEE_ID" = p_emp_id;
    RETURN v_salary;
END;
$$ LANGUAGE plpgsql;

```
- Scratch Pad:** Contains a message: "Query returned successfully in 238 msec."

Рис. 16: Создание функции.

Точный режим. Находим сотрудника по имени и запросим точную зарплату. (Рис. 17)

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

- Servers:** appdb/app@Web PostgreSQL
- Databases:** appdb
- Query History:**

```

SELECT "EmployeesDepartments".get_salary_by_mode(
    (SELECT "EMPLOYEE_ID" FROM "EmployeesDepartments"."EMPLOYEES"
     WHERE "NAME" = 'EXACT');
)

```
- Result Grid:**

	get_salary_by_mode
1	6000.00
- Scratch Pad:** Contains a message: "Successfully run. Total query runtime: 204 msec. 1 rows affected."

Рис. 17: Демонстрация точного режима.

Округленный режим. Округляет зарплату до целого числа. (Рис. 18)

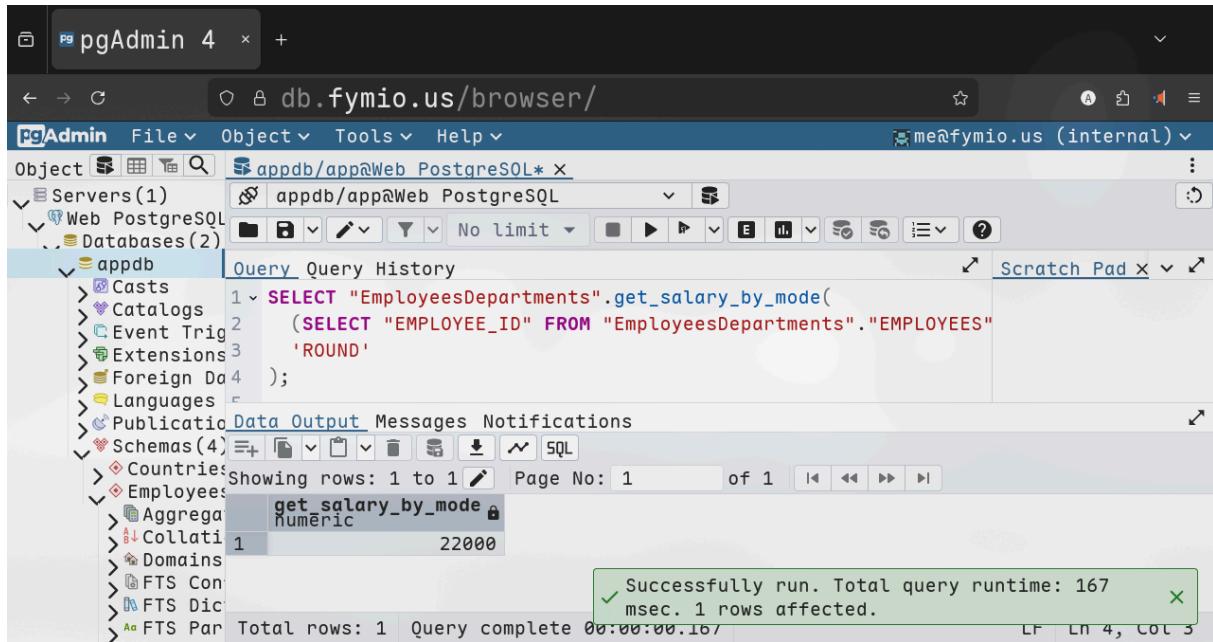


Рис. 18: Демонстрация округленного режима.

Задание 6. Создание хранимой процедуры

Я создал хранимую процедуру, предназначенную для увеличения заработной платы выбранного сотрудника на указанный процент. (Рис. 19)

```

CREATE OR REPLACE PROCEDURE
"EmployeesDepartments".raise_salary(
    p_employee_id INTEGER,
    p_percent NUMERIC
)
LANGUAGE plpgsql
AS $$

BEGIN
    -- Увеличиваем зарплату сотрудника
    UPDATE "EmployeesDepartments"."EMPLOYEES"
    SET "SALARY" = "SALARY" * (1 + p_percent / 100)
    WHERE "EMPLOYEE_ID" = p_employee_id;
END;
$$;

```

The screenshot shows the pgAdmin 4 interface. In the left sidebar, under 'appdb/app@Web PostgreSQL', there is a 'Query' tab where the following SQL code is being typed:

```

SET SALARY = SALARY +
  WHERE "EMPLOYEE_ID" = 100;
END;
$$;

```

The 'Execute script' button (F5) is highlighted. Below the query, a message says 'Query returned successfully in 180 msec.' and at the bottom, it says 'Total rows: 0 Query complete 00:00:00.180'.

Рис. 19: Создание процедуры `raise_salary`.

Затем я проверил работу созданной хранимой процедуры. Для этого я проверил зарплату сотрудника с идентификатором 100. (Рис. 20)

```

SELECT "SALARY" FROM "EmployeesDepartment"."EMPLOYEES" WHERE
"EMPLOYEE_ID" = 100;

```

The screenshot shows the pgAdmin 4 interface. In the left sidebar, under 'appdb/app@Web PostgreSQL', there is a 'Query' tab with the following query:

```

1 SELECT "SALARY" FROM "EmployeesDepartment"."EMPLOYEES"
2 WHERE "EMPLOYEE_ID" = 100;

```

The results pane shows a single row with the value '24000.00'. A message at the bottom right says 'Successfully run. Total query runtime: 940 msec. 1 rows affected.'

Рис. 20: Зарплата “номера 100”.

Затем я вызвал процедуру, которая повысила заработную плату сотруднику на 10%. (Рис. 21)

```
CALL "EmployeesDepartments".raise_salary(100, 10);
```

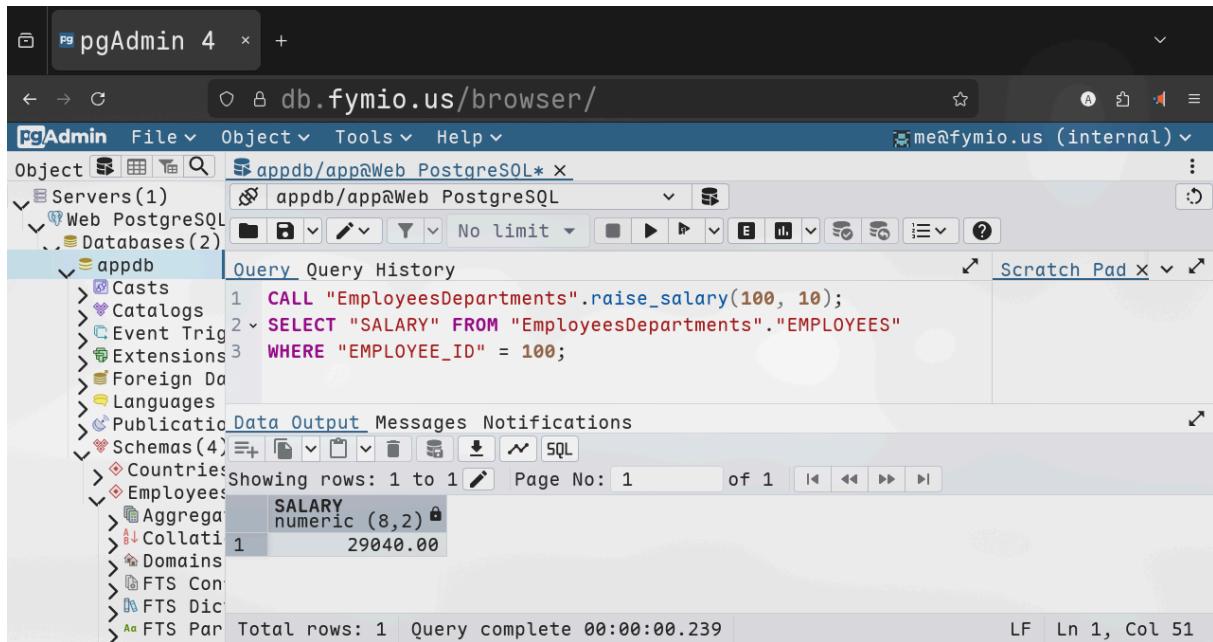


Рис. 21: Вызов процедуры `raise_salary`.

В процедуру не обязательно передавать параметры в той последовательности, в которой они были объявлены. К параметрам процедуры можно обращаться по имени. (Рис. 22)

```
SELECT * FROM "EmployeesDepartments"."EMPLOYEES" WHERE
"EMPLOYEE_ID" = 101;

CALL "EmployeesDepartments".raise_salary(p_percent:=10,
p_employee_id:=101);

SELECT * FROM "EmployeesDepartments"."EMPLOYEES" WHERE
"EMPLOYEE_ID" = 101;
```

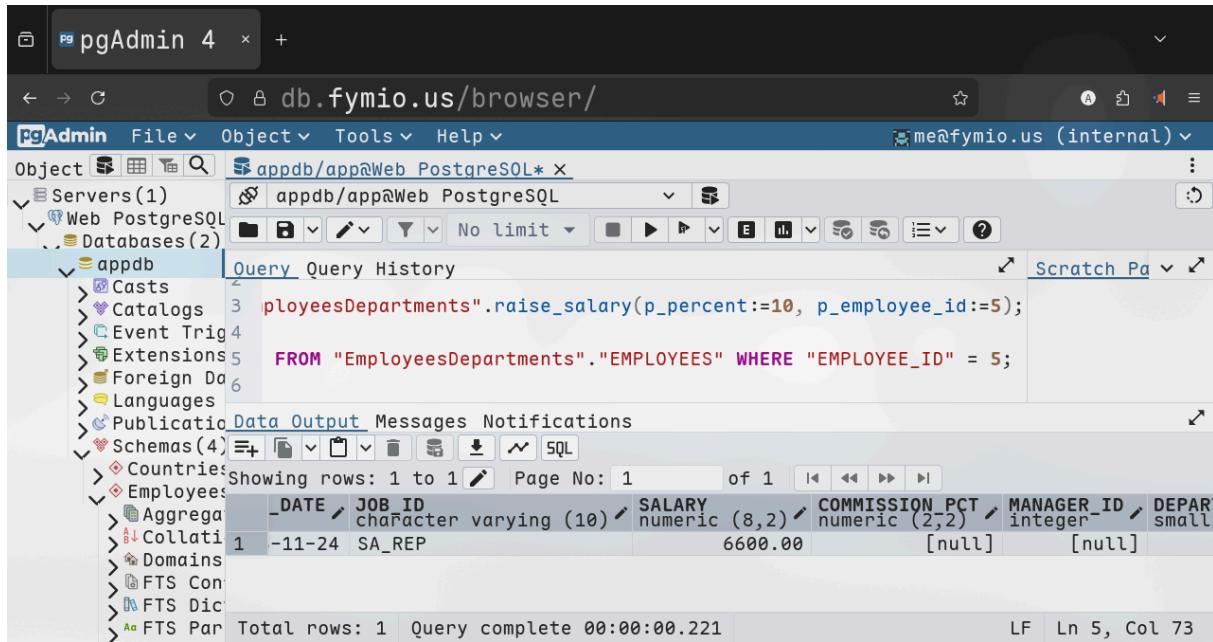


Рис. 22: Обращение к параметрам процедуры `raise_salary` по имени

Задание 7. Создание собственной хранимой процедуры

Я создал процедуру. (Рис. 23)

```

CREATE OR REPLACE PROCEDURE
"EmployeesDepartments".manage_employee(p_emp_id INT, p_mode
TEXT)
LANGUAGE plpgsql AS
$$
BEGIN
    IF NOT EXISTS(
        SELECT 1
        FROM "EmployeesDepartments"."EMPLOYEES"
        WHERE "EMPLOYEE_ID" = p_emp_id
    ) THEN
        RETURN;
    END IF;

    IF upper(p_mode) = 'INCREASE' THEN
        UPDATE "EmployeesDepartments"."EMPLOYEES"
        SET "SALARY" = round("SALARY" * 1.10)
        WHERE "EMPLOYEE_ID" = p_emp_id;
    ELSIF upper(p_mode) = 'BONUS' THEN
        UPDATE "EmployeesDepartments"."EMPLOYEES"
        SET "SALARY" = "SALARY" + 1000
        WHERE "EMPLOYEE_ID" = p_emp_id;
    END IF;

```

```

ELSE
    RETURN;
END IF;
END;
$$;

```

```

pgAdmin 4  + db.fymio.us/browser/
PgAdmin File Object Tools Help
Object Servers(1) Web PostgreSQL Databases(2)
appdb Query RETURN;
Casts 22 END IF;
Catalogs 23
Event Trig 24 END;
Extensions 25 $$;
Foreign Da 26
Languages 27
Publicatio Data Output Messages Notifications
Schemas(4) CREATE PROCEDURE
Countries
Employees Query returned successfully in 350 msec.
Aggrega
Collati
Domains
FTS Con
FTS Dic
FTS Par Total rows: Query complete 00:00:00.350
Query returned successfully in 350 msec. X
LF Ln Zb, Col 1

```

Рис. 23: Создание процедуры.

Режим `INCREASE`. Увеличение на 10% (Рис. 24)

```

SELECT "EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME", "SALARY"
FROM "EmployeesDepartments"."EMPLOYEES"
WHERE "EMPLOYEE_ID" = 100;

CALL "EmployeesDepartments".manage_employee(100, 'INCREASE');

SELECT "EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME", "SALARY"
FROM "EmployeesDepartments"."EMPLOYEES"
WHERE "EMPLOYEE_ID" = 100;

```

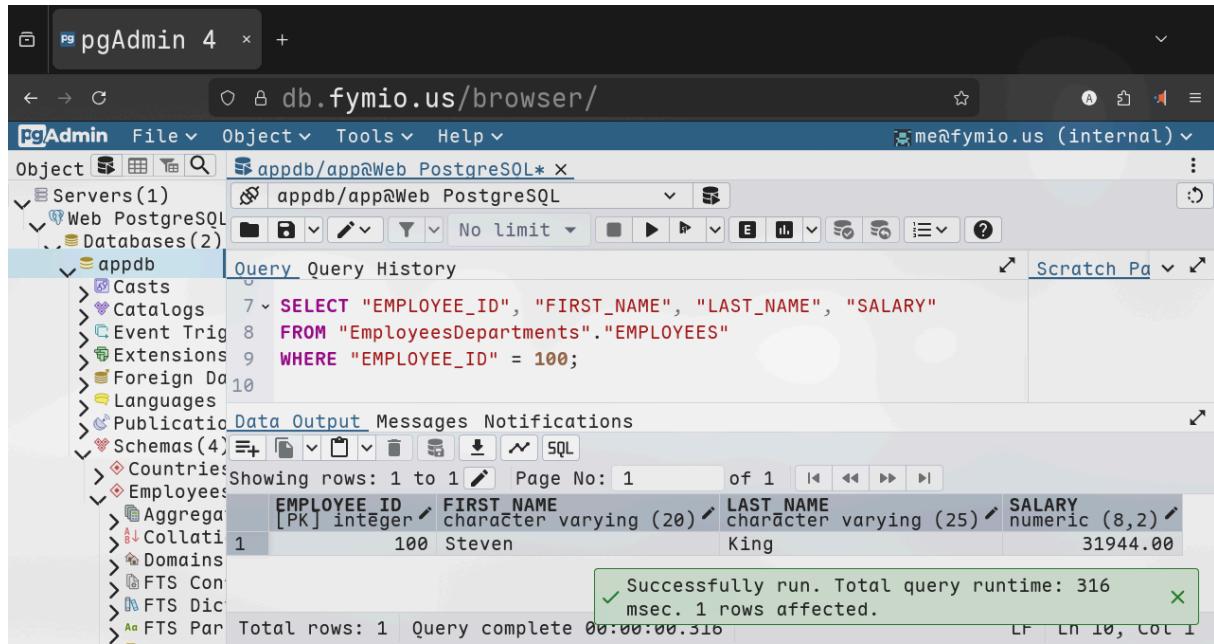


Рис. 24: Режим INCREASE .

Режим BONUS . Прибавление 1000 (Рис. 25)

```

SELECT "EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME", "SALARY"
FROM "EmployeesDepartments"."EMPLOYEES"
WHERE "EMPLOYEE_ID" = 100;

CALL "EmployeesDepartments".manage_employee(100, 'BONUS');

SELECT "EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME", "SALARY"
FROM "EmployeesDepartments"."EMPLOYEES"
WHERE "EMPLOYEE_ID" = 100;

```

```

5 CALL "EmployeesDepartments".manage_employee(100, 'BONUS');
6
7 SELECT "EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME", "SALARY"
8 FROM "EmployeesDepartments"."EMPLOYEES"
9 WHERE "EMPLOYEE_ID" = 100;

```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
1	100	Steven	King	32944.00

Successfully run. Total query runtime: 804 msec. 1 rows affected.

Рис. 25: Режим BONUS .

Выводы и анализ результатов работы

В ходе выполнения работы я последовательно создал обычные и материализованные представления, пользовательские функции и хранимые процедуры, а также проверил их работу на практических примерах. Была получена ясная разница между видами представлений: обычное всегда возвращает актуальные данные, так как вычисляет запрос при каждом обращении, а материализованное хранит результат и требует обновления, но работает быстрее. Это проявилось в том, что изменения в таблице “Employees” не отражались в материализованном представлении до выполнения команды обновления.

В процессе работы я освоил создание представлений как через интерфейс pgadmin, так и через sql, а также убедился, что изменения структуры таблиц, от которых зависят представления, могут быть заблокированы. Были созданы две скалярные функции — одна для получения полного имени сотрудника, другая для возврата зарплаты в разных режимах.

Также была реализована собственная процедура с ветвлением, выполняющая разные действия в зависимости от режима.