МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**Факультет безопасности информационных технологий**

**Направление подготовки: 10.03.01 Информационная безопасность**

**Образовательная программа: "Информационная безопасность / Information security"**

**Дисциплина:**

**«*Информационная безопасность баз данных*»**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙРАБОТЕ № 4**

***«Контроль доступа и системы аудита»***

**Выполнил студент:**

группа/поток 1.3

Бардышев Артём Антонович/\_\_\_\_\_\_\_\_\_\_

*Подпись*

**Проверил:**

Карманова Наталья Андреевна/\_\_\_\_\_\_\_\_\_\_

*Подпись*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Отметка о выполнении (один из вариантов:*

*отлично, хорошо, удовлетворительно, зачтено)*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата*

Санкт-Петербург

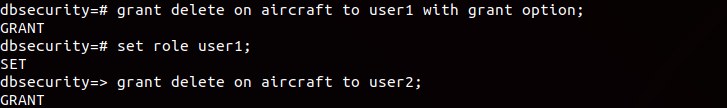
2025г.

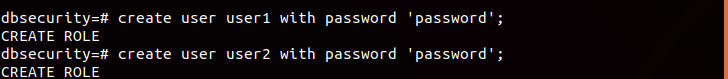
**Контроль доступа и системы аудита**

**Существует множество подходов к реализации системы контроля доступа внутри базы данных. В данной практической работе описаны некоторые из них.**

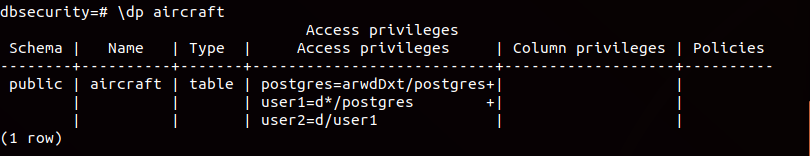
**Во-первых, PostgreSQL имеет возможность устанавливать права на запросы *select/update/delete/ insert* с помощью команд *grant/revoke*. Также PostgreSQL позволяет делегировать управление правами пользователям, а те могут делиться собственными правами друг с другом, используя опцию «*with grant option*».**

**Например, мы можем создать пользователя *user1* и дать ему право на удаление строк в таблице и возможность передать это право другому пользователю. После этого *user1* от своего имени может выдать разрешение на удаление строк другому пользователю *user2*.**

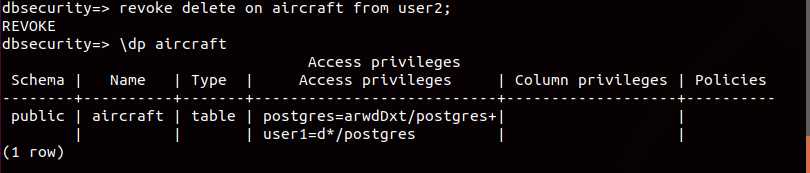
****

****

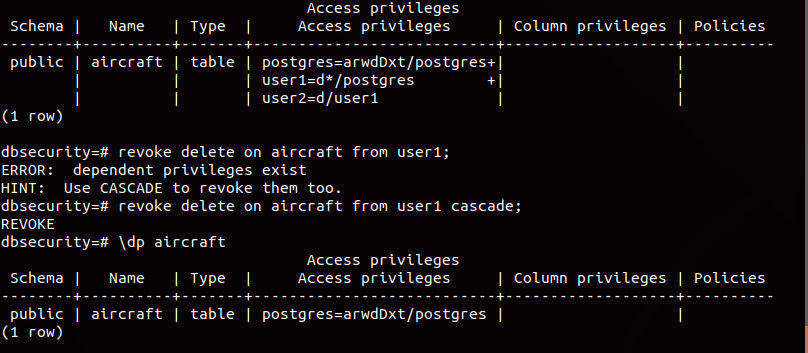
**Чтобы просмотреть настройки прав, установленные для таблиц на текущий момент, надо ввести встроенную короткую команду *\dp*.**

****

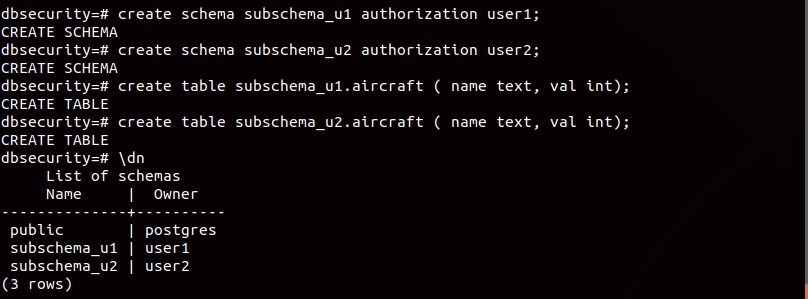
**Чтобы отозвать у пользователя выданные права, нужно использовать команду *revoke*.**

****

**Фактически, мы можем удалить выданные права со всеми случаями делегирования одной командой, применив опцию «*cascade*».**

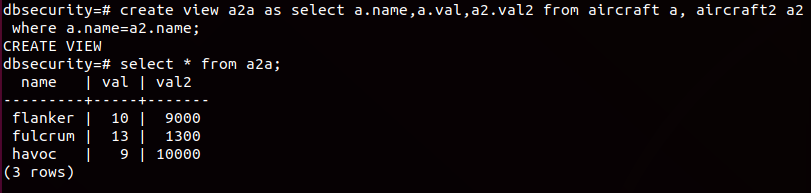
****

**Еще одним способом управления доступом пользователей к объектам внутри базы данных являются подсхемы. Подсхемы позволяют выделить отдельные наборы таблиц для каждого пользователя. Последовательность команд для создания подсхем представлена ​​на рисунке ниже.**

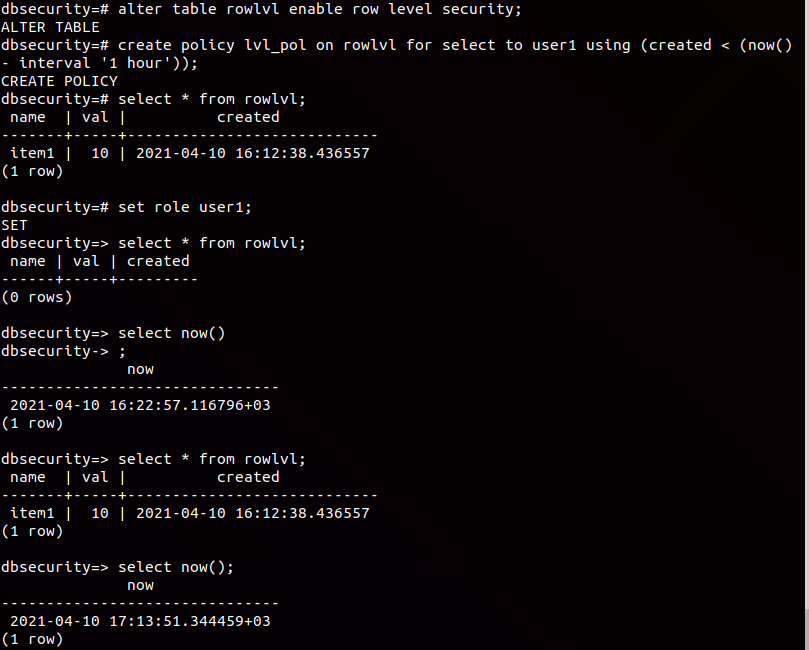
****

**Пользователь *user1* из примера может видеть только таблицу из своей подсхемы.**

**Если необходимо управление доступом на уровне столбцов таблицы, то в PostgresSQL можно использовать представления. Пример команды для создания представления с целью управления доступом можно найти на рисунке ниже.**

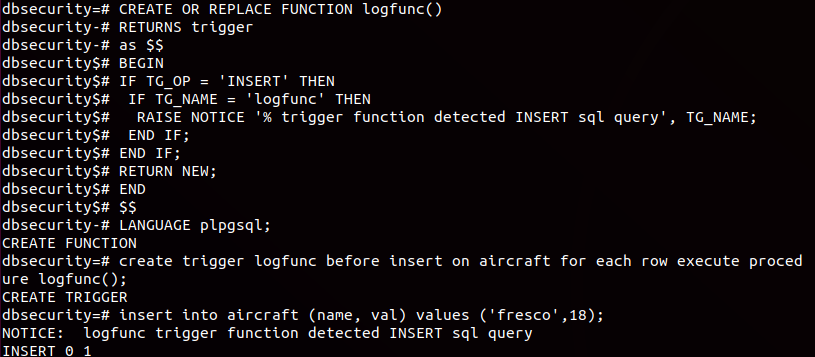
****

**Следующим уровнем детализации при разграничении доступа к объектам внутри базы данных является управление доступом на уровне строк. Функция управления доступом активируется путем ввода команды «*enable row level security*», а также с помощью применения политики для вывода только соответствующих ей строк. Пример такой настройки представлен на рисунке ниже.**

****

**Наконец, давайте посмотрим, как мы можем реализовать систему аудита на основе встроенных функций базы данных PostgreSQL. Основная идея – запускать обработчик для каждого действия в базе данных. Хорошим способом для реализации системы аудита является использование функции триггера, которая позволяет обработать каждый оператор SQL во время его выполнения.**

**Нам нужно создать функцию триггера и сопоставить ее с некоторой таблицей событий, например, для оператора *insert*. В приведенном ниже примере создается функция, которая проверяет, является ли тип запроса вставкой, и вызывает триггер, который вставляет уведомление в таблицу событий о том, какое точное значение и в какую таблицу было вставлено.**

****

**Задачи для практической работы:**

**1. Подготовьте таблицы для выполнения перечисленных ниже задач. Достаточно 2-3 таблиц для п. 1-5 ниже.**

**2. Выдайте права 3 пользователям. Пользователь User1 должен иметь полный доступ к таблице. User2 должен иметь право на вставку, select-запросы и обновление значений в таблицах. User3 должен иметь право на удаление строк из таблиц, а также возможность делегировать свои права любому пользователю.**

**3. Предоставьте право на удаление от пользователя User3 пользователю User4 и проверьте все выданные права.**

**4. Отмените все предоставленные выше права.**

**5. Создайте подсхему авторизации для User1 и User2 с различным набором таблиц (служебное слово AUTHORIZATION у команды CREATE SCHEMA).**

**6. Создайте представление как объединенный набор столбцов из разных таблиц. С помощью команды grant ограничьте доступ к представлению.**

**7. Настройте безопасность на уровне строк (RLS), политика должна быть создана на основе текущего пользователя, и протестируйте ее.**

**8. Создайте триггер для регистрации вставки, обновления и удаления содержимого в определенных таблицах.**

ХОД РАБОТЫ:

1) **Подготовка таблиц**

CREATE DATABASE access\_lab;

\c access\_lab

CREATE TABLE employees (

id SERIAL PRIMARY KEY,

name TEXT,

position TEXT,

salary INTEGER

);

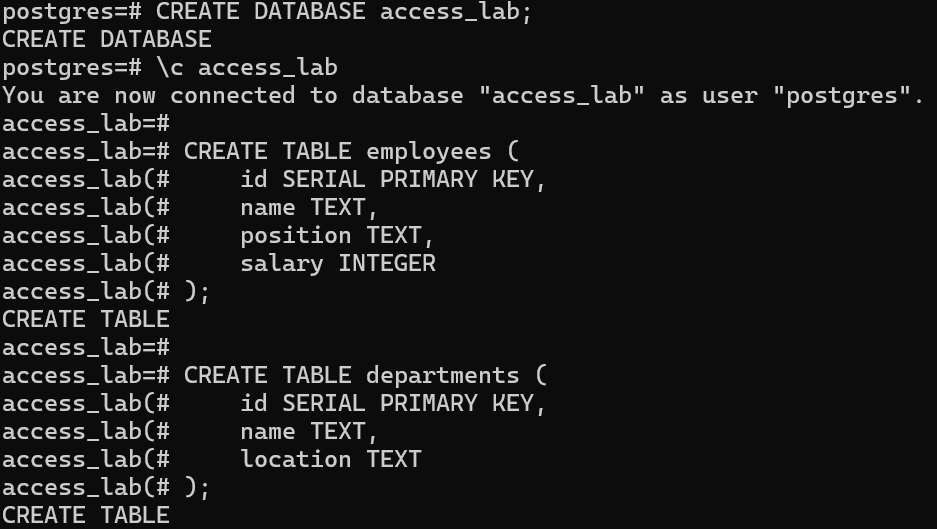
CREATE TABLE departments (

id SERIAL PRIMARY KEY,

name TEXT,

location TEXT

);



2) **Выдача прав пользователям**

**CREATE USER user1 WITH PASSWORD 'pass1';**

**CREATE USER user2 WITH PASSWORD 'pass2';**

**CREATE USER user3 WITH PASSWORD 'pass3';**

**-- user1: полный доступ**

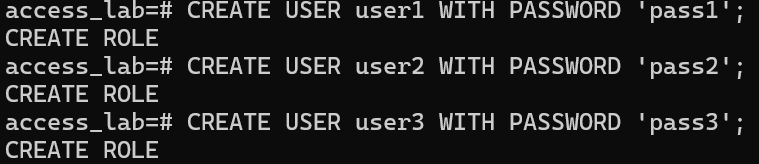
**GRANT ALL ON employees, departments TO user1;**

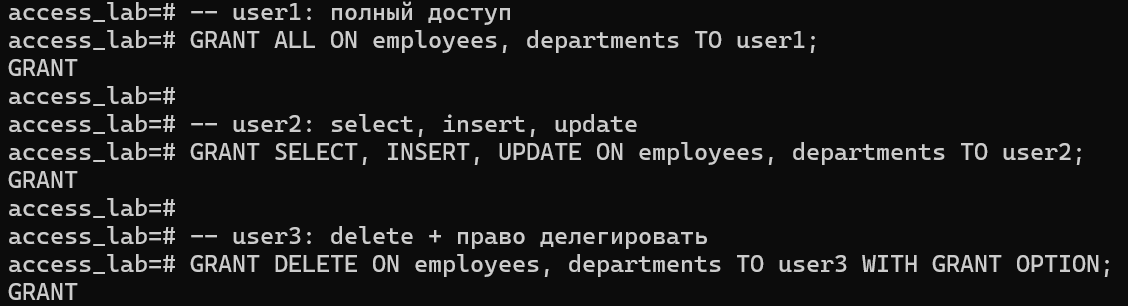
**-- user2: select, insert, update**

**GRANT SELECT, INSERT, UPDATE ON employees, departments TO user2;**

**-- user3: delete + право делегировать**

**GRANT DELETE ON employees, departments TO user3 WITH GRANT OPTION;**





3) **Делегирование прав**

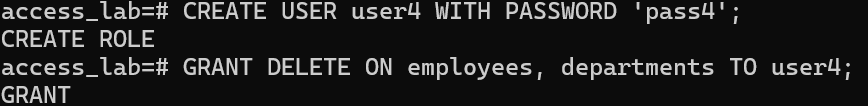
CREATE USER user4 WITH PASSWORD 'pass4';

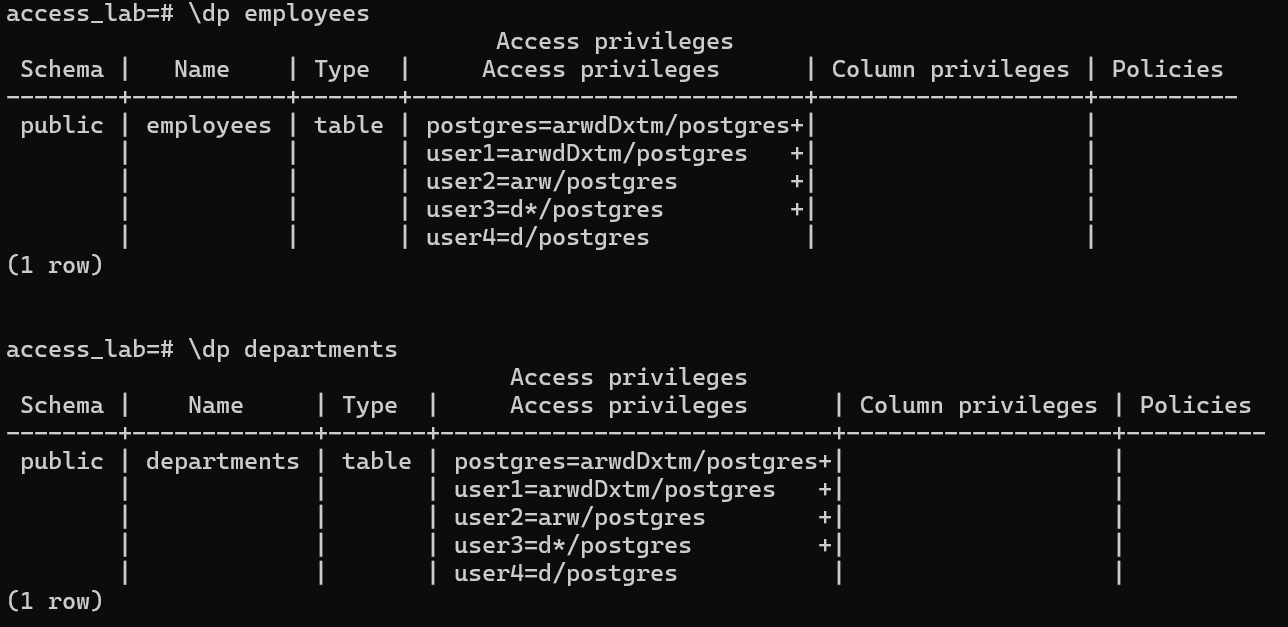
GRANT DELETE ON employees, departments TO user4;

Проверка прав:

\dp employees

\dp departments





4) **Отмена всех прав**

REVOKE ALL ON employees, departments FROM user1, user2, user3, user4 CASCADE;





5) **Создание подсхем (schemas)**

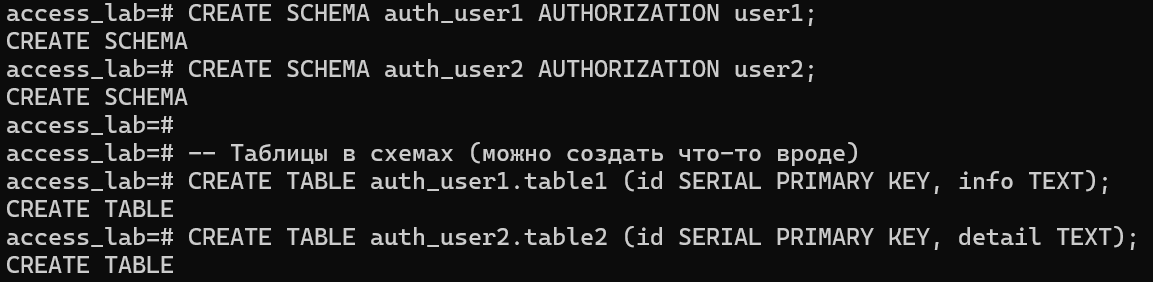
**CREATE SCHEMA auth\_user1 AUTHORIZATION user1;**

**CREATE SCHEMA auth\_user2 AUTHORIZATION user2;**

**-- Таблицы в схемах (можно создать что-то вроде)**

**CREATE TABLE auth\_user1.table1 (id SERIAL PRIMARY KEY, info TEXT);**

**CREATE TABLE auth\_user2.table2 (id SERIAL PRIMARY KEY, detail TEXT);**



6) **Создание представления (view)**

CREATE VIEW employee\_dept\_view AS

SELECT e.name, e.position, d.name AS department

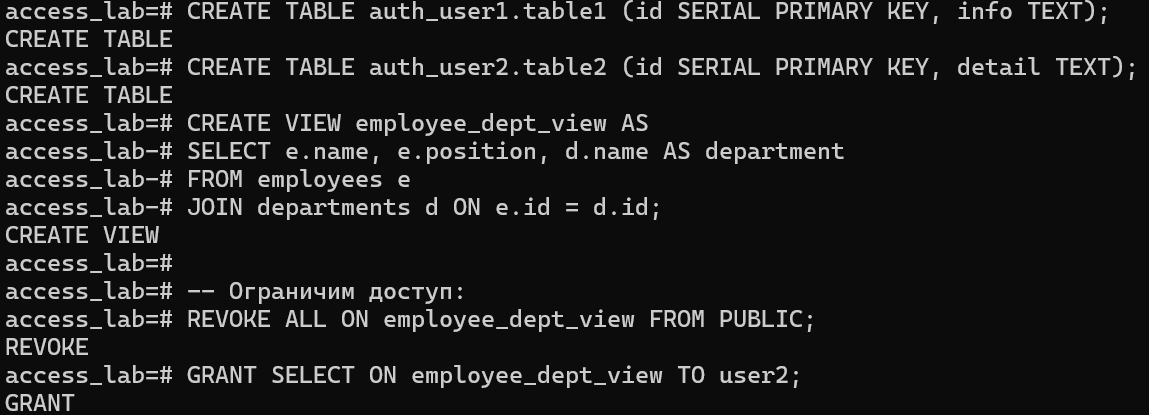
FROM employees e

JOIN departments d ON e.id = d.id;

-- Ограничим доступ:

REVOKE ALL ON employee\_dept\_view FROM PUBLIC;

GRANT SELECT ON employee\_dept\_view TO user2;



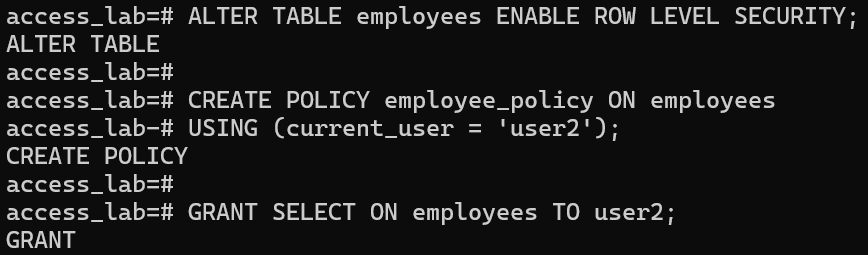
7) **Безопасность на уровне строк (RLS)**

ALTER TABLE employees ENABLE ROW LEVEL SECURITY;

CREATE POLICY employee\_policy ON employees

USING (current\_user = 'user2');

GRANT SELECT ON employees TO user2;



8) Создание триггера для аудита

Создаём таблицу для логов:

CREATE TABLE audit\_log (

id SERIAL PRIMARY KEY,

action TEXT,

table\_name TEXT,

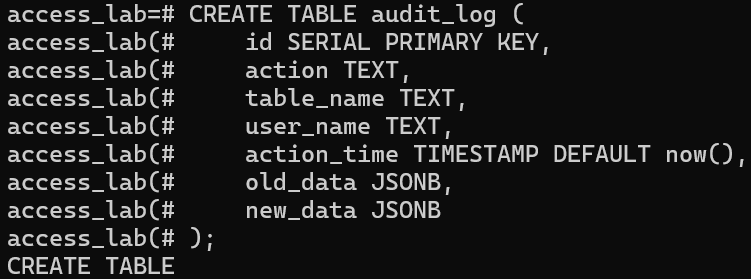
user\_name TEXT,

action\_time TIMESTAMP DEFAULT now(),

old\_data JSONB,

new\_data JSONB

);



Функция триггера:

CREATE OR REPLACE FUNCTION log\_changes()

RETURNS TRIGGER AS $$

BEGIN

IF TG\_OP = 'INSERT' THEN

INSERT INTO audit\_log(action, table\_name, user\_name, new\_data)

VALUES ('INSERT', TG\_TABLE\_NAME, current\_user, row\_to\_json(NEW));

RETURN NEW;

ELSIF TG\_OP = 'UPDATE' THEN

INSERT INTO audit\_log(action, table\_name, user\_name, old\_data, new\_data)

VALUES ('UPDATE', TG\_TABLE\_NAME, current\_user, row\_to\_json(OLD), row\_to\_json(NEW));

RETURN NEW;

ELSIF TG\_OP = 'DELETE' THEN

INSERT INTO audit\_log(action, table\_name, user\_name, old\_data)

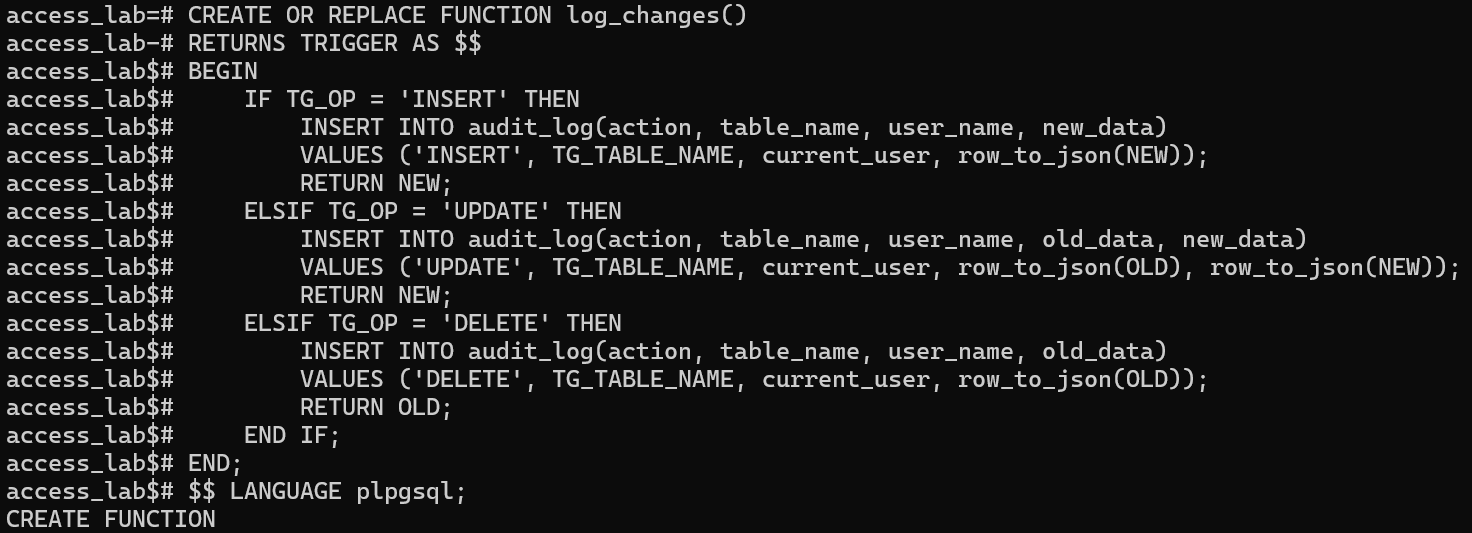
VALUES ('DELETE', TG\_TABLE\_NAME, current\_user, row\_to\_json(OLD));

RETURN OLD;

END IF;

END;

$$ LANGUAGE plpgsql;



Добавим триггеры к таблицам:

CREATE TRIGGER emp\_audit

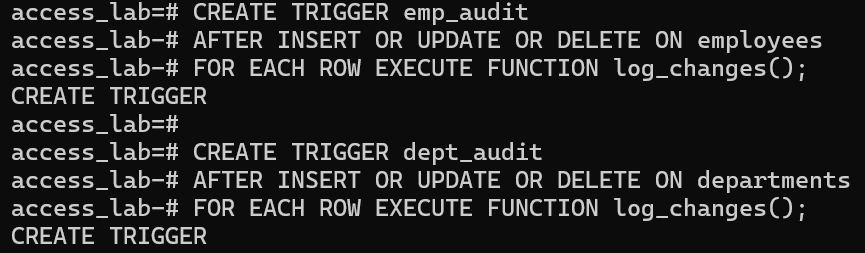
AFTER INSERT OR UPDATE OR DELETE ON employees

FOR EACH ROW EXECUTE FUNCTION log\_changes();

CREATE TRIGGER dept\_audit

AFTER INSERT OR UPDATE OR DELETE ON departments

FOR EACH ROW EXECUTE FUNCTION log\_changes();



Вывод:

В ходе выполнения лабораторной работы были изучены и практически реализованы основные механизмы разграничения доступа в СУБД PostgreSQL. Мы создали несколько таблиц и пользователей с различными уровнями прав, научились управлять доступом к данным с помощью команд GRANT и REVOKE, а также делегировать права другим пользователям.

Особое внимание было уделено созданию подсхем, представлений и реализации механизмов безопасности на уровне строк (RLS), что позволяет гибко управлять доступом вплоть до отдельных строк таблиц. Кроме того, была реализована система аудита с помощью триггеров, позволяющая отслеживать действия пользователей в базе данных, включая вставку, обновление и удаление данных.

Полученные знания и практические навыки являются важными для обеспечения безопасности и целостности данных в многопользовательских системах и могут быть успешно применены при разработке и сопровождении корпоративных информационных систем.