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*.

```
dbsecurity=# grant delete on aircraft to user1 with grant option;
GRANT
dbsecurity=# set role user1;
SET
dbsecurity=> grant delete on aircraft to user2;
GRANT
```

```
dbsecurity=# create user user1 with password 'password';
CREATE ROLE
dbsecurity=# create user user2 with password 'password';
CREATE ROLE
```

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

```
dbsecurity=# \dp aircraft
                                 Access privileges
 Schema |   Name   | Type  |     Access privileges     | Column privileges | Policies
--------+----------+-------+---------------------------+-------------------+----------
 public | aircraft | table | postgres=arwdDxt/postgres+|                   |
        |          |       | user1=d*/postgres        +|                   |
        |          |       | user2=d/user1             |                   |
(1 row)
```

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

```
dbsecurity=> revoke delete on aircraft from user2;
REVOKE
dbsecurity=> \dp aircraft
                                 Access privileges
 Schema |   Name   | Type  |     Access privileges     | Column privileges | Policies
--------+----------+-------+---------------------------+-------------------+----------
 public | aircraft | table | postgres=arwdDxt/postgres+|                   |
        |          |       | user1=d*/postgres         |                   |
(1 row)
```

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

```
                                Access privileges
 Schema |   Name   | Type  |     Access privileges     | Column privileges | Policies
--------+----------+-------+---------------------------+-------------------+----------
 public | aircraft | table | postgres=arwdDxt/postgres+|                   |
        |          |       | user1=d*/postgres        +|                   |
        |          |       | user2=d/user1             |                   |
(1 row)

dbsecurity=# revoke delete on aircraft from user1;
ERROR:  dependent privileges exist
HINT:  Use CASCADE to revoke them too.
dbsecurity=# revoke delete on aircraft from user1 cascade;
REVOKE
dbsecurity=# \dp aircraft
                                Access privileges
 Schema |   Name   | Type  |     Access privileges     | Column privileges | Policies
--------+----------+-------+---------------------------+-------------------+----------
 public | aircraft | table | postgres=arwdDxt/postgres |                   |
(1 row)
```

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.

```
dbsecurity=# create schema subschema_u1 authorization user1;
CREATE SCHEMA
dbsecurity=# create schema subschema_u2 authorization user2;
CREATE SCHEMA
dbsecurity=# create table subschema_u1.aircraft ( name text, val int);
CREATE TABLE
dbsecurity=# create table subschema_u2.aircraft ( name text, val int);
CREATE TABLE
dbsecurity=# \dn
     List of schemas
     Name     |  Owner
--------------+----------
 public       | postgres
 subschema_u1 | user1
 subschema_u2 | user2
(3 rows)
```

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.

```
dbsecurity=# create view a2a as select a.name,a.val,a2.val2 from aircraft a, aircraft2 a2
 where a.name=a2.name;
CREATE VIEW
dbsecurity=# select * from a2a;
  name   | val | val2
---------+-----+-------
 flanker |  10 |  9000
 fulcrum |  13 |  1300
 havoc   |   9 | 10000
(3 rows)
```

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.

```
dbsecurity=# alter table rowlvl enable row level security;
ALTER TABLE
dbsecurity=# create policy lvl_pol on rowlvl for select to user1 using (created < (now()
- interval '1 hour'));
CREATE POLICY
dbsecurity=# select * from rowlvl;
 name  | val |          created
-------+-----+----------------------------
 item1 |  10 | 2021-04-10 16:12:38.436557
(1 row)

dbsecurity=# set role user1;
SET
dbsecurity=> select * from rowlvl;
 name | val | created
------+-----+---------
(0 rows)

dbsecurity=> select now()
dbsecurity-> ;
              now
-------------------------------
 2021-04-10 16:22:57.116796+03
(1 row)

dbsecurity=> select * from rowlvl;
 name  | val |          created
-------+-----+----------------------------
 item1 |  10 | 2021-04-10 16:12:38.436557
(1 row)

dbsecurity=> select now();
              now
-------------------------------
 2021-04-10 17:13:51.344459+03
(1 row)
```

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.

```
dbsecurity=# CREATE OR REPLACE FUNCTION logfunc()
dbsecurity-# RETURNS trigger
dbsecurity-# as $$
dbsecurity$# BEGIN
dbsecurity$# IF TG_OP = 'INSERT' THEN
dbsecurity$#  IF TG_NAME = 'logfunc' THEN
dbsecurity$#   RAISE NOTICE '% trigger function detected INSERT sql query', TG_NAME;
dbsecurity$#  END IF;
dbsecurity$# END IF;
dbsecurity$# RETURN NEW;
dbsecurity$# END
dbsecurity$# $$
dbsecurity-# LANGUAGE plpgsql;
CREATE FUNCTION
dbsecurity=# create trigger logfunc before insert on aircraft for each row execute proced
ure logfunc();
CREATE TRIGGER
dbsecurity=# insert into aircraft (name, val) values ('fresco',18);
NOTICE:  logfunc trigger function detected INSERT sql query
INSERT 0 1
```

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
);
```

```
postgres=# CREATE DATABASE access_lab;
CREATE DATABASE
postgres=# \c access_lab
You are now connected to database "access_lab" as user "postgres".
access_lab=#
access_lab=# CREATE TABLE employees (
access_lab(#     id SERIAL PRIMARY KEY,
access_lab(#     name TEXT,
access_lab(#     position TEXT,
access_lab(#     salary INTEGER
access_lab(# );
CREATE TABLE
access_lab=#
access_lab=# CREATE TABLE departments (
access_lab(#     id SERIAL PRIMARY KEY,
access_lab(#     name TEXT,
access_lab(#     location TEXT
access_lab(# );
CREATE TABLE
```

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;

```
access_lab=# CREATE USER user1 WITH PASSWORD 'pass1';
CREATE ROLE
access_lab=# CREATE USER user2 WITH PASSWORD 'pass2';
CREATE ROLE
access_lab=# CREATE USER user3 WITH PASSWORD 'pass3';
CREATE ROLE
```

```
access_lab=# -- user1: полный доступ
access_lab=# GRANT ALL ON employees, departments TO user1;
GRANT
access_lab=#
access_lab=# -- user2: select, insert, update
access_lab=# GRANT SELECT, INSERT, UPDATE ON employees, departments TO user2;
GRANT
access_lab=#
access_lab=# -- user3: delete + право делегировать
access_lab=# GRANT DELETE ON employees, departments TO user3 WITH GRANT OPTION;
GRANT
```

3) **Делегирование прав**

CREATE USER user4 WITH PASSWORD 'pass4';

GRANT DELETE ON employees, departments TO user4;

Проверка прав:

\dp employees

\dp departments

```
access_lab=# CREATE USER user4 WITH PASSWORD 'pass4';
CREATE ROLE
access_lab=# GRANT DELETE ON employees, departments TO user4;
GRANT
```

```
access_lab=# \dp employees
                                Access privileges
 Schema |   Name    | Type  |     Access privileges      | Column privileges | Policies
--------+-----------+-------+----------------------------+-------------------+----------
 public | employees | table | postgres=arwdDxtm/postgres+|                   |
        |           |       | user1=arwdDxtm/postgres   +|                   |
        |           |       | user2=arw/postgres        +|                   |
        |           |       | user3=d*/postgres         +|                   |
        |           |       | user4=d/postgres           |                   |
(1 row)


access_lab=# \dp departments
                                Access privileges
 Schema |    Name     | Type  |     Access privileges      | Column privileges | Policies
--------+-------------+-------+----------------------------+-------------------+----------
 public | departments | table | postgres=arwdDxtm/postgres+|                   |
        |             |       | user1=arwdDxtm/postgres   +|                   |
        |             |       | user2=arw/postgres        +|                   |
        |             |       | user3=d*/postgres         +|                   |
        |             |       | user4=d/postgres           |                   |
(1 row)
```

4) **Отмена всех прав**

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

```
access_lab=# REVOKE ALL ON employees, departments FROM user1, user2, user3, user4 CASCADE;
REVOKE
```

```
access_lab=# \dp employees
                                Access privileges
 Schema |   Name    | Type  |     Access privileges      | Column privileges | Policies
--------+-----------+-------+----------------------------+-------------------+----------
 public | employees | table | postgres=arwdDxtm/postgres |                   |
(1 row)


access_lab=# \dp departments
                                Access privileges
 Schema |    Name     | Type  |     Access privileges      | Column privileges | Policies
--------+-------------+-------+----------------------------+-------------------+----------
 public | departments | table | postgres=arwdDxtm/postgres |                   |
(1 row)
```

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);

```
access_lab=# CREATE SCHEMA auth_user1 AUTHORIZATION user1;
CREATE SCHEMA
access_lab=# CREATE SCHEMA auth_user2 AUTHORIZATION user2;
CREATE SCHEMA
access_lab=#
access_lab=# -- Таблицы в схемах (можно создать что-то вроде)
access_lab=# CREATE TABLE auth_user1.table1 (id SERIAL PRIMARY KEY, info TEXT);
CREATE TABLE
access_lab=# CREATE TABLE auth_user2.table2 (id SERIAL PRIMARY KEY, detail TEXT);
CREATE TABLE
```

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;

```
access_lab=# CREATE TABLE auth_user1.table1 (id SERIAL PRIMARY KEY, info TEXT);
CREATE TABLE
access_lab=# CREATE TABLE auth_user2.table2 (id SERIAL PRIMARY KEY, detail TEXT);
CREATE TABLE
access_lab=# CREATE VIEW employee_dept_view AS
access_lab-# SELECT e.name, e.position, d.name AS department
access_lab-# FROM employees e
access_lab-# JOIN departments d ON e.id = d.id;
CREATE VIEW
access_lab=#
access_lab=# -- Ограничим доступ:
access_lab=# REVOKE ALL ON employee_dept_view FROM PUBLIC;
REVOKE
access_lab=# GRANT SELECT ON employee_dept_view TO user2;
GRANT
```

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;

```
access_lab=# ALTER TABLE employees ENABLE ROW LEVEL SECURITY;
ALTER TABLE
access_lab=#
access_lab=# CREATE POLICY employee_policy ON employees
access_lab-# USING (current_user = 'user2');
CREATE POLICY
access_lab=#
access_lab=# GRANT SELECT ON employees TO user2;
GRANT
```

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
);
```

```
access_lab=# CREATE TABLE audit_log (
access_lab(#     id SERIAL PRIMARY KEY,
access_lab(#     action TEXT,
access_lab(#     table_name TEXT,
access_lab(#     user_name TEXT,
access_lab(#     action_time TIMESTAMP DEFAULT now(),
access_lab(#     old_data JSONB,
access_lab(#     new_data JSONB
access_lab(# );
CREATE TABLE
```

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;
```

```
access_lab=# CREATE OR REPLACE FUNCTION log_changes()
access_lab-# RETURNS TRIGGER AS $$
access_lab$# BEGIN
access_lab$#     IF TG_OP = 'INSERT' THEN
access_lab$#         INSERT INTO audit_log(action, table_name, user_name, new_data)
access_lab$#         VALUES ('INSERT', TG_TABLE_NAME, current_user, row_to_json(NEW));
access_lab$#         RETURN NEW;
access_lab$#     ELSIF TG_OP = 'UPDATE' THEN
access_lab$#         INSERT INTO audit_log(action, table_name, user_name, old_data, new_data)
access_lab$#         VALUES ('UPDATE', TG_TABLE_NAME, current_user, row_to_json(OLD), row_to_json(NEW));
access_lab$#         RETURN NEW;
access_lab$#     ELSIF TG_OP = 'DELETE' THEN
access_lab$#         INSERT INTO audit_log(action, table_name, user_name, old_data)
access_lab$#         VALUES ('DELETE', TG_TABLE_NAME, current_user, row_to_json(OLD));
access_lab$#         RETURN OLD;
access_lab$#     END IF;
access_lab$# END;
access_lab$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
```

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();
```

```
access_lab=# CREATE TRIGGER emp_audit
access_lab-# AFTER INSERT OR UPDATE OR DELETE ON employees
access_lab-# FOR EACH ROW EXECUTE FUNCTION log_changes();
CREATE TRIGGER
access_lab=#
access_lab=# CREATE TRIGGER dept_audit
access_lab-# AFTER INSERT OR UPDATE OR DELETE ON departments
access_lab-# FOR EACH ROW EXECUTE FUNCTION log_changes();
CREATE TRIGGER
```

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.