1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 3**

**Организация уровневого доступа к данным**

по дисциплине «Системы управления базами данных»

1. Выполнила
2. студентка гр. 4851003/90801 Кулеева А.Г.

1. Руководитель
2. ассистент Зубков Е.А.
3. Санкт-Петербург
4. 2023

# **Цель**

Получение навыков обработки событий и гранулирования доступа при работе с СУБД.

# Задание

Для выполнения работы используется схема данных из работы 1. Если вы уже выполнили работу 1, то реализованное в ней разграничение прав (GRANT и созданные политики) лучше удалить в начале работы, для устранения конфликтов. Или, что предпочтительно, создать отдельную схему данных для последующего сравнения реализаций.

1. Реализовать заданное ограничение целостности.
2. Реализовать представления в соответствии с требованиями по разграничению прав из работы 1. Выполнить все условия разграничения. При разработке представлений учитывать обе функции безопасности, которые они выполняют.
3. Дать соответствующим пользователям права только в отношении созданных представлений.
4. Для прав на модификацию данных обеспечить и протестировать обновляемость представлений.
5. Сделать выводы в отношении применения стандартных способов разграничения прав доступа и построения разграничения на основе представлений по итогам данной работы и работы 1.
6. Обеспечить аудит действий пользователя по манипулированию данными с использованием вспомогательной таблицы для логгирования (в отношении базовой таблицы по выбору студента).
7. Оценить ограничения созданного лог-контента и его операционные характеристики: размер, скорость поиска и т.д.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 ― Вариант задания

# Ход работы

## Создание базы

В соответствии с выданным вариантом (рисунок 1) была реализована логическая схема данных (рисунок 2).

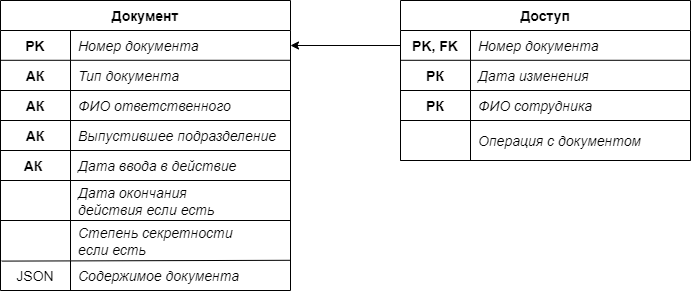


Рисунок 2 ― Реализованные связи

Создадим базу данных для выполнения работы (рисунок 3). Данная база будет состоять из двух таблиц (рисунки 4-5). Код реализации таблиц представлен в Приложении. Также стоит отметить, что для каждой таблицы была создана своя переменная перечислимого типа.

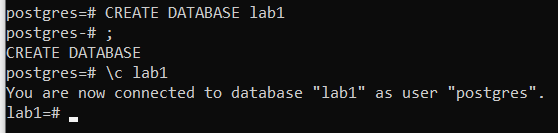


Рисунок 3 ― Требуемые привилегии

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 ― Схема для отношения 1

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 ― Схема для отношения 2

В соответствии с вариантом была создана матрица доступа (рисунок 6).



Рисунок 6 ― Ролевая матрица доступа

## Ограничение целостности

Первой операцией с документом всегда должна быть операция «Создание», а последней – «Архивирование» (аналог удаления).

Для данного ограничения создадим триггер, который будет проверять вводимые значения. Триггер будет связан с указанной таблицей, представлением или сторонней таблицей и будет выполнять заданную функцию при определённых операциях с этой таблицей.

CREATE OR REPLACE FUNCTION oper\_check() RETURNS TRIGGER

AS $$

BEGIN

IF left(NEW.operation, 1) = ‘c’ AND

right(NEW.operation, 1) = ‘d’ THEN

RETURN NEW;

ELSE

RAISE EXCEPTION 'Первой операцией с документом всегда должна быть операция «Создание», а последней – «Архивирование» (аналог удаления)';

END IF;

END

$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS oper\_check ON change\_log;

CREATE TRIGGER oper\_check

BEFORE INSERT OR UPDATE

ON change\_log

FOR EACH ROW

EXECUTE PROCEDURE oper\_check();

update change\_log set operation = 'cribksud' where operation = 'r';

На рисунке 7 представлен результат вставки корректного и некорректного значений. Единственная проблема в том, что PostgreSQL не поддерживает русский язык и исключение печатается в виде непонятного набора символов.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 7 — Результат создания триггера

## Создание представлений

Представления — это таблицы, чье содержание выбирается или получается из других таблиц. Они работают в запросах и операторах DML точно также как и основные таблицы, но не содержат никаких собственных данных. Представления подобны окнам, через которые можно просматривать информацию, которая фактически хранится в базовой таблице. Представление — это фактически запрос, который выполняется всякий раз, когда представление становится темой команды. Вывод запроса при этом в каждый момент становится содержанием представления.

Итак, создадим две роли: ответственный и неответственный. Создадим представления, то есть результат запроса SELECT к определённым данным. Поскольку представление — это, по сути, такая же таблица, то сделаем GRANT через эту таблицу, а не через основную (рисунок 8).

Изображение выглядит как текст

Автоматически созданное описаниеИзображение выглядит как текст

Автоматически созданное описание

Рисунок 8 — Представления

Создадим пользователей и проверим наличие доступа для различных ролей (рисунки 9-10).

Изображение выглядит как текст

Автоматически созданное описаниеИзображение выглядит как текст

Автоматически созданное описание

Рисунок 9 — Проверка доступа для ответственного пользователя

Изображение выглядит как текст

Автоматически созданное описаниеРисунок 10 — Проверка доступа для неответственного пользователя

## Аудит действий пользователя

Требуется создать дополнительную таблицу, в которую будут сохраняться лог-сообщения обо всех операциях, которые совершает тот или иной пользователь в отношении основных таблиц. Для этого был создан дополнительный триггер, срабатывающий при попытке пользователя получить доступ к данным.

drop table if exists logging\_table;

create table logging\_table (

username text not null,

tablename text not null,

operation text not null,

mtime timestamp not null,

old\_record varchar(200) not null,

new\_record varchar(200) not null);

create or replace function log\_update()

returns trigger as $log\_update$

begin

insert into logging\_table (username, tablename, operation, mtime, old\_record, new\_record)

values (current\_user, TG\_TABLE\_NAME, TG\_OP, now(), old::varchar(200), new::varchar(200));

return null;

end;

$log\_update$ language plpgsql;

drop trigger if exists logging\_trigger on document;

create trigger logging\_trigger

after insert or update or delete on document

for each row execute procedure log\_update();

После создания данного триггера и сопутствующей функции пользователи потеряли право выполнять свои запросы, поскольку у них нет доступа к таблице логов. Для этого дадим им права следующей командой:

GRANT INSERT ON logging\_table TO respons;

Проверим сохранение данных в таблицу (рисунок 11).

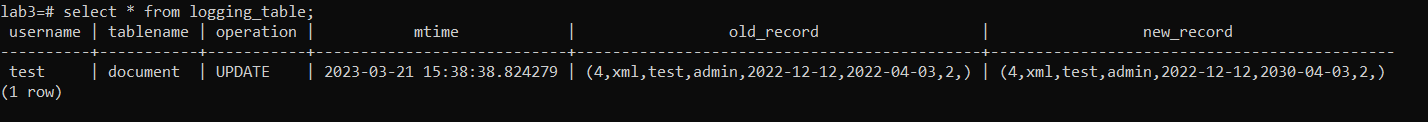


Рисунок 11 — Логирование действий пользователя

# Вывод

В ходе выполнения данной лабораторной работы были получены навыки создания схемы данных и разграничения доступа на основе представлений при работе с СУБД. Были созданы две таблицы и пользователи с различными привилегиями. После тестирования разработанной модели все условия задания оказались выполненными, то есть, итоговая матрица доступа не отличается от теоретической. Однако создание представлений может быть более эффективно с точки зрения безопасности, поскольку тогда пользователь обращается не к исходной таблице напрямую, а к промежуточному значению. Следовательно, пользователь может не знать, что администратор скрыл от него некоторые столбцы.

Также был изучен механизм триггеров. Данный механизм таже помогает ограничить действия пользователя, установить ограничения на ввод некорректных данных.

ПРИЛОЖЕНИЕ 1

CREATE DATABASE lab3;

\c lab3

CREATE TYPE sec\_lvl AS ENUM ('1', '2');

CREATE TABLE document (

num\_doc BIGSERIAL NOT NULL PRIMARY KEY,

type\_doc VARCHAR(50) NOT NULL UNIQUE,

fio\_respons VARCHAR(100) NOT NULL UNIQUE,

department VARCHAR(50) NOT NULL UNIQUE,

date\_start DATE NOT NULL UNIQUE,

date\_end DATE,

secret sec\_lvl,

content json);

CREATE TABLE change\_log (

num\_doc BIGSERIAL NOT NULL UNIQUE REFERENCES document (num\_doc),

date\_change DATE NOT NULL UNIQUE,

fio VARCHAR(100) NOT NULL UNIQUE,

operation text,

PRIMARY KEY (num\_doc, date\_change, fio));

//////////////////////////////////////////////////////

INSERT INTO document (

type\_doc, fio\_respons, department, date\_start, date\_end, secret)

VALUES (‘docx’, ‘Ivanov’, ‘IT’, ’2022-01-01’, ’2025-01-01’, ‘2’);

INSERT INTO document (

type\_doc, fio\_respons, department, date\_start, date\_end, secret)

VALUES (‘pdf’, ‘Smirnov’, ‘HR’, ’2020-03-01’, ’2024-01-01’, ‘1’);

INSERT INTO document (

type\_doc, fio\_respons, department, date\_start, date\_end, secret)

VALUES (‘txt’, ‘Kuleeva’, ‘IC’, ’2022-11-11’, ’2022-02-02’, ‘2’);

INSERT INTO document (

type\_doc, fio\_respons, department, date\_start, date\_end, secret)

VALUES (‘xml’, ‘test’, ‘admin’, ’2022-12-12’, ’2022-03-03’, ‘2’);

INSERT INTO document (

type\_doc, fio\_respons, department, date\_start, date\_end, secret)

VALUES (‘c’, ‘weak’, ‘finance’, ’2000-03-01’, ’2004-01-01’, ‘1’);

-----------------------------

INSERT INTO change\_log (

date\_change, fio, operation)

VALUES (‘2022-05-01’, ‘Kuleeva’, ‘u’);

INSERT INTO change\_log (

date\_change, fio, operation)

VALUES (‘2021-05-01’, ‘Smirnov’, ‘r’);

INSERT INTO change\_log (

date\_change, fio, operation)

VALUES (‘2022-05-05’, ‘Ivanov’, ‘d’);

INSERT INTO change\_log (

date\_change, fio, operation)

VALUES (‘2022-06-06’, ‘test’, ‘d’);

INSERT INTO change\_log (

date\_change, fio, operation)

VALUES (‘2002-06-06’, ‘weak’, ‘crud’);

///////////////////////////////////////////////////////

CREATE ROLE respons;

CREATE VIEW selector\_resp\_doc AS

SELECT num\_doc, type\_doc, fio\_respons, department, date\_start, date\_end, secret

FROM document

WHERE (fio\_respons = current\_user);

GRANT SELECT ON TABLE selector\_resp\_doc TO respons;

GRANT INSERT ON TABLE selector\_resp\_doc TO respons;

GRANT UPDATE ("date\_end") ON TABLE selector\_resp\_doc TO respons;

CREATE VIEW selector\_resp\_oper AS

SELECT operation FROM change\_log

WHERE (fio = current\_user);

GRANT SELECT ON TABLE selector\_resp\_oper TO respons;

GRANT INSERT ON TABLE selector\_resp\_oper TO respons;

GRANT UPDATE ON TABLE selector\_resp\_oper TO respons;

//////////////////////////////////////////////////

CREATE ROLE not\_respons;

CREATE VIEW selector\_not\_resp AS

SELECT num\_doc, type\_doc, fio\_respons, department, date\_start, date\_end, secret

FROM document

WHERE (secret = ‘1’);

GRANT SELECT ON TABLE selector\_not\_resp TO not\_respons;

GRANT INSERT ON TABLE selector\_not\_resp TO not\_respons;

//////////////////////////////////////////////////

CREATE USER test PASSWORD ‘test’;

GRANT respons TO test;

CREATE USER weak PASSWORD ‘weak’;

GRANT not\_respons TO weak;

//////////////////////////////////////////////////

select \* from selector\_resp\_doc;

select \* from document;

select \* from change\_log;

select operation from change\_log;

select \* from selector\_resp\_oper;

update document set date\_end = '2022-04-03' where num\_doc = 2;

update selector\_resp\_doc set date\_end = '2030-04-03' where type\_doc = ‘xml’;

select \* from selector\_not\_resp;

select \* from selector\_resp\_doc;

select \* from document;

select \* from change\_log;

select \* from selector\_resp\_oper;