Методические указания

по лабораторной работе №3

на тему: «Защита базы данных»

по дисциплине: Информационная безопасность баз данных

Санкт-Петербург

2024

Цель работы

Получение навыков созданию примитивных систем мониторинга, разграничения доступа и шифрования средствами СУБД.

Задание

1. Задачи по мониторингу БД:

* Создайте таблицу-лог, отдельную от ваших основных сущностей БД.
* Создайте для каждой основной таблицы в вашей БД триггер, который срабатывает при любых изменениях в БД (вставка новых данных, изменение существующих записей, удаление кортежей из таблицы). При срабатывании триггер должен вносить в таблицу-лог информацию о том, когда было произведено изменение, со стороны какой роли поступил запрос, какие кортежи поменялись, старые и новые значения.
* Продемонстрируйте работу системы логирования для различных операций и отношений.

1. Задачи по шифрованию данных.

* Создайте таблицу с секретными данными, отдельно от ваших основных сущностей. Например, это может быть таблица с токенами или ключами доступа, для каждого класса-пользователей.
* Зашифруйте содержимое данной таблицы, в качестве алгоритма шифрования используйте любой симметричный алгоритм шифрования. Ключ шифрования для данной таблицы не должен храниться в ИС. Ключ шифрования может быть получен из индивидуального пароля для дешифрования суперпользователя (пароль не связан с паролем для входа в СУБД). Индивидуальный пароль суперпользователя и ключ шифрования может быть связан через одностороннюю функцию. Например, пусть индивидуальный пароль комбинация «!stroNgpsw31234», считаем от данного пароля детерминированную хэш-функцию (например, sha-256), полученный хэш-используем как ключ шифрования/дешифрования для симметричного алгоритма шифрования таблицы с секретными данными (например, для AES-256)
* Демонстрируем, что даже обладая полными правами администратора, но без знания индивидуального пароля невозможно получить содержимое таблицы с секретными данными

1. Задачи по разграничению доступа в БД:

* Создайте в СУБД как минимум 2 роли (суперпользователь не считается) для каждого из классов потребителей информации;
* С помощью внутренних инструментов СУБД для каждой роли определите набор привилегий по отношению к таблицам вашей БД. Руководствуйтесь принципом минимальных привилегий, если определенному классу потребителей не нужен доступ к определенным таблицам/атрибутам (список задач БД, составленный в рамках 1 ЛР), то доступ к этим таблицам/атрибутам не предоставляется. Разграничиваем доступ к представлениям, созданным в 1 ЛР, а также таблицам логирования (таблицы логирования может просматривать только суперпользователь)
* Продемонстрируйте работу вашей системы разграничения доступа. Зайдите за каждую из ролей и покажите доступные со стороны каждой роли отношения.

Пример отчета по лабораторной работе №3.

**Задачи по мониторингу операций в БД:**

Создадим триггерную функцию *logging()*, которая будет срабатывать при изменениях данных и сохранять тип операции, время, пользователя и выполненные изменения.

*CREATE OR REPLACE FUNCTION logging() RETURNS TRIGGER AS $logging$*

*BEGIN*

*IF (TG\_OP = 'DELETE') THEN*

*INSERT INTO public.main\_log (operation\_type, operation\_date, user\_operator, changed\_data) VALUES ('D', now(), current\_user, row\_to\_json(OLD));*

*ELSIF (TG\_OP = 'UPDATE') THEN*

*INSERT INTO public.main\_log (operation\_type, operation\_date, user\_operator, changed\_data) VALUES ('U', now(), current\_user, row\_to\_json(NEW));*

*ELSIF (TG\_OP = 'INSERT') THEN*

*INSERT INTO public.main\_log (operation\_type, operation\_date, user\_operator, changed\_data) VALUES ('I', now(), current\_user, row\_to\_json(NEW));*

*END IF;*

*RETURN NULL;*

*END;*

*$logging$ LANGUAGE plpgsql;*

Создадим таблицу-лог «main\_log», состоящую из следующих атрибутов:

*CREATE TABLE public.main\_log*

*(*

*log\_item\_id integer NOT NULL,*

*operation\_type text,*

*operation\_date text,*

*user\_operator text,*

*changed\_data text,*

*PRIMARY KEY (log\_item\_id)*

*);*

Сделаем атрибут «log\_item\_id» автогенерируемым идентификатором кортежа в таблице:

*ALTER TABLE IF EXISTS public.main\_log*

*ALTER COLUMN log\_item\_id ADD GENERATED BY DEFAULT AS IDENTITY;*

Создаем для каждой таблицы («classes», «groups», «specialty», «subject», «teachers») в БД триггер, который вызывает триггерную функцию *logging(),* по следующему шаблону:

*CREATE TRIGGER logging\_exec*

*AFTER INSERT OR UPDATE OR DELETE ON classes*

*FOR EACH ROW EXECUTE FUNCTION logging();*

Продемонстрируем работу созданной системы логирования для случайных таблиц.

Например, вставим данные таблицу classes и выведем содержимое лога «main\_log».

Вставка данных:

*INSERT INTO public.classes (classes\_id, classes\_start\_time, classes\_end\_time, classroom, building, subject\_id)*

*VALUES ('10'::integer, '8'::text, '9'::text, '232'::text, '1231'::text, '1'::integer);*

Таблица-лог при вставке данных, представлена на Рисунке 1:

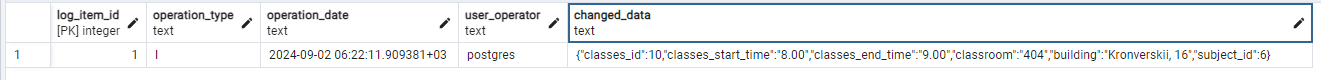


Рисунок 1 – Логирование операции вставки

Изменим данные в таблице «Преподаватели»

*UPDATE public.teachers SET*

*email = 'vvv\_new\_mail@mail.ru'::text WHERE*

*teacher\_id = 3277122;*

Удалим одну из записей из таблицы «Занятия»

*DELETE FROM public.classes*

*WHERE classes\_id IN (2);*

Снова проверим работоспособность таблицы-лога (Результат на Рисунке 2.):

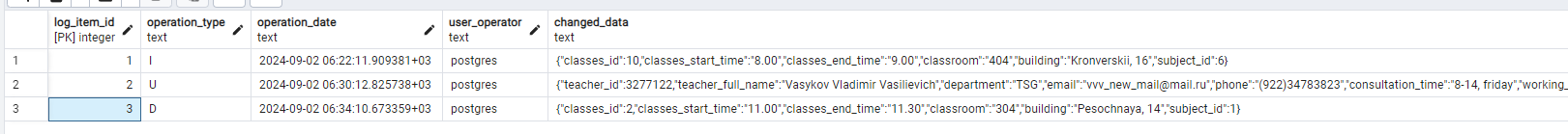


Рисунок 2 – Логирование всех операций

2. **Задачи по шифрованию данных.**

Создадим таблицу с секретными данными. Например, это может быть таблица с токенами или ключами доступа, для каждого пользователей.

*CREATE TABLE public.secret\_data*

*(*

*"ID" integer NOT NULL GENERATED BY DEFAULT AS IDENTITY,*

*username text,*

*secret\_token text,*

*PRIMARY KEY ("ID")*

*);*

Загружаем модуль «pgcrypto», если он не установлен. Заполним таблицу примерами зашифрованных данных. Пусть индивидуальным паролем админа будет «personal\_admin\_pass», посчитаем hash sha-256 от данного пароля, получим «9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08». Полученный хэш будем использовать как симметричный ключ шифрования. Вставим данные в зашифрованном виде:

*INSERT INTO secret\_data (username, secret\_token)*

*VALUES*

*(pgp\_sym\_encrypt('gena\_kozlov', '9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08'),*

*pgp\_sym\_encrypt('token\_4fsdf43fdaf43r2fdsa', '9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08')),*

*(pgp\_sym\_encrypt('vasykov', '9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08'), pgp\_sym\_encrypt('token\_13gdfss4fa232fdsadfsa', '9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08'));*

Зашифрованные значения таблицы «secret\_data» в итоге имеют вид, представленный на Рисунке 3.

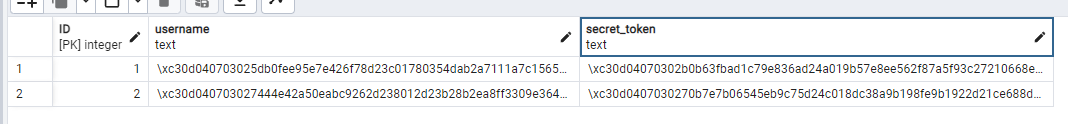


Рисунок 3 – Таблица «secret\_data»

3**. Задачи по разграничению доступа в БД.**

Согласно заданию, необходимо создать как минимум 2 групповые роля для каждого из классов потребителей информации. В первой лабораторной были выделены потребители «Студент» и «Персонал». Также создадим групповую роль «Админ», необходима для 4 ЛР, «Админ» должен иметь возможность просматривать все таблицы и представления в БД.

Код создания групповых ролей:

*CREATE ROLE student\_group\_role WITH*

*NOLOGIN*

*NOSUPERUSER*

*NOCREATEDB*

*NOCREATEROLE*

*INHERIT*

Согласно целям проектирования БД и исходя из принципа представления наименьших привилегий, выделим групповой роли «Студенты» привилегии на просмотр только представлений «compact\_shedule» и «consultations»:

*GRANT SELECT ON TABLE public.compact\_shedule TO student\_group\_role;*

*GRANT SELECT ON TABLE public.consultations TO student\_group\_role;*

Аналогично создадим групповую роль для персонала (*staff\_group\_role*) и предоставим доступ для данной групповой роли только к представлениям «*speciality\_and\_contacts*» и «*speciality\_and\_teachers*»:

*CREATE ROLE staff\_group\_role WITH*

*NOLOGIN*

*NOSUPERUSER*

*NOCREATEDB*

*NOCREATEROLE*

*INHERIT*

*GRANT SELECT ON TABLE public.speciality\_and\_contacts TO staff\_group\_role;*

*GRANT SELECT ON TABLE public.speciality\_and\_teachers TO staff\_group\_role;*

И наконец, создадим и определим привилегии для групповой роди «Админ»:

*CREATE ROLE admin\_group\_role WITH*

*NOLOGIN*

*NOSUPERUSER*

*NOCREATEDB*

*NOCREATEROLE*

*INHERIT*

*GRANT SELECT ON TABLE*

*classes,*

*"groups",*

*main\_log,*

*secret\_data,*

*specialty,*

*subject,*

*teachers,*

*speciality\_and\_contacts,*

*speciality\_and\_teachers,*

*compact\_shedule,*

*consultations*

*TO admin\_group\_role;*

Теперь создадим по одному пользователю для каждой из групповых ролей. Выделим им привилегии группы и продемонстрируем работу предлагаемой системы разграничения доступа на примере доступа к случайным таблицам.

Начнем с пользователя-студента, который будет обладать привилегиями групповой роли «*student\_group\_role*». Создадим индивидуальную роль «Romanov\_student» для конкретного студента с возможностью LOGIN:

CREATE ROLE "Romanov\_student" WITH

LOGIN

NOSUPERUSER

NOCREATEDB

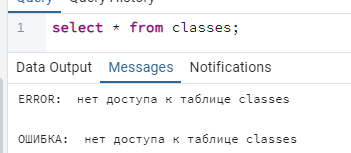
NOCREATEROLE

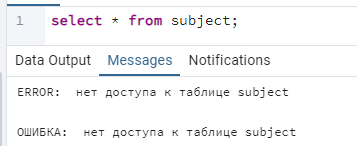
GRANT student\_group\_role TO "Romanov\_student";

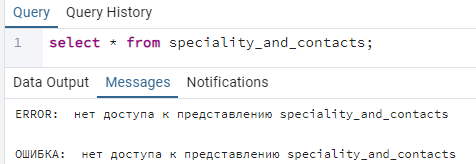
Зайдет со стороны индивидуальной роли "Romanov\_student" и проверим доступность содержимого таблиц.

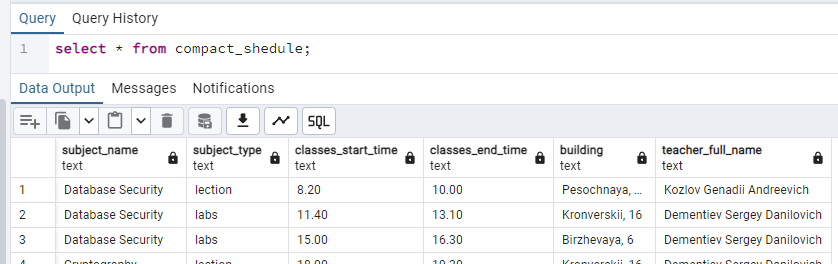
*SET ROLE "Romanov\_student";*

Попытка доступа к случайным таблицам со стороны *"Romanov\_student":*









Доступны только представления «compact\_shedule» и «consultations».

Аналогично, создадим и проверим индивидуальную роль «kozlov\_teacher» для групповой роли «*staff\_group\_role*»:

*CREATE ROLE kozlov\_teacher WITH*

*LOGIN*

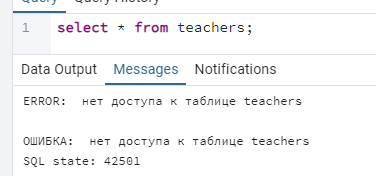
*NOSUPERUSER*

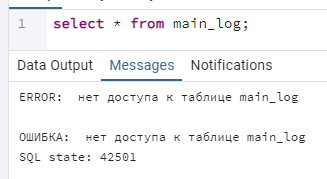
*NOCREATEDB*

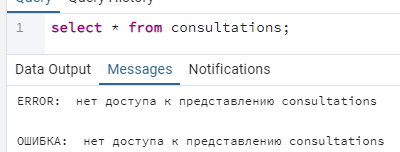
*grant staff\_group\_role to kozlov\_teacher;*

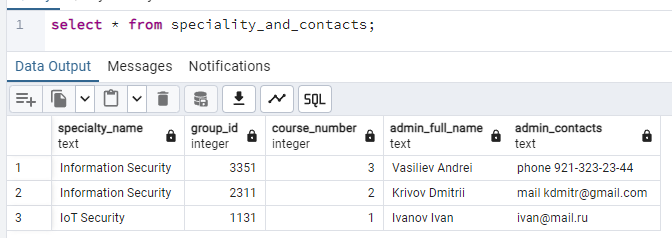
*set role kozlov\_teacher;*

Попытки доступа:









Для индивидуальной роли «Kozlov\_teacher» доступны только представления «*speciality\_and\_contacts*» и «*speciality\_and\_teachers*».

Создадим, индивидуальную роль «*petya\_admin*».

*CREATE ROLE petya\_admin WITH*

*LOGIN*

*NOSUPERUSER*

*NOCREATEDB*

*NOCREATEROLE*

*grant admin\_group\_role TO petya\_admin;*

*set role petya\_admin;*

Попытки доступа:

