ОТЧЕТ

О выполнении долгосрочного домашнего задания по дисциплине «Основы построения защищенных баз данных»

Выполнил: А.О. Студзинский

Москва

2023 г.

1. **Описание предметной области (ПО).**

**Предметная область**: Интернет-магазин, специализирующийся на продаже билетов в театр.

1. **Ограничения и правила:**

* Связанные таблицы не могут быть созданы без зависимых (foreign key)
* Доступ вне роли запрещен
* Ведется аудит доступа

1. **Сущности**

Table ticket {

ticket\_id integer [primary key]

event integer

holl integer

zone integer

date\_of\_purchase timestamp

}

Table event {

event\_id integer [primary key]

event\_date timestamp

group integer

place integer

}

Table group {

group\_id integer [primary key]

group\_name varchar

music\_style varchar

est timestamp

}

Table musician {

musician\_id integer [primary key]

group integer

executor\_name varchar

tool\_type varchar

BIO varchar

}

Table holl {

holl\_id integer [primary key]

holl\_name varchar

}

Table zone {

zone\_id integer [primary key]

cost integer

description varchar

}

Table place {

place\_id integer [primary key]

description varchar

}

Table users {

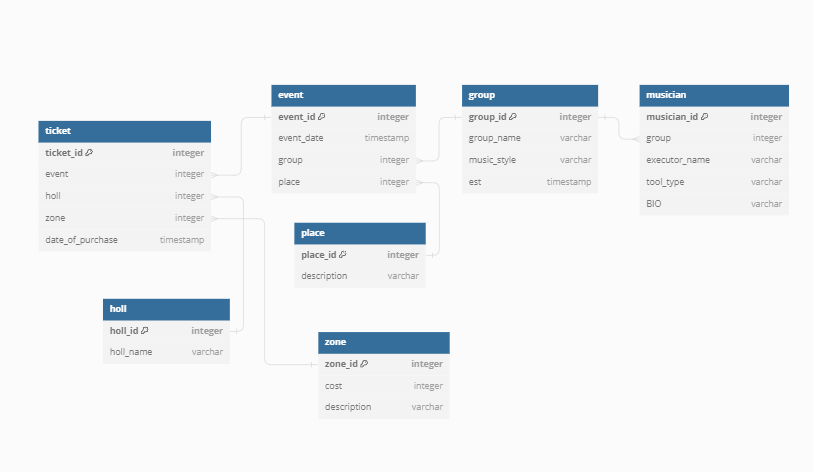
id integer [primary key]

login varchar

password varchar

}

1. **TR - диаграмма.** В этом разделе приводится TR-диаграмма, соответствующая выбранной предметной области и ER-диаграмме из п.4. TR-диаграмма строится для заданной преподавателем версии СУБД.



1. **Ролевая модель безопасности БД.** Содержит перечень защищаемых объектов БД, описание введенных ролей, правила назначения ролей пользователям БД, матрицы доступа к данным БД в зависимости от роли и/или пользователя. В отчете должны быть приведены скрипты создания соответствующих ролей, назначения привилегий и назначения ролей пользователям.

**Роли и их права:**

* 1. Администратор

**Роль**: Администратор магазина.

**Права**:

* Создание, изменение и удаление записей в таблицах: ticket, holl, even, group, place, musician, zone, audit\_log, users.
* Управление структурой базы данных, включая создание и удаление таблиц.
* Назначение и управление ролями.
  1. Клиент:

**Роль**: Покупатели.

**Права**:

* Просмотр записей в таблицах: ticket, holl, even, group, place, musician, zone
  1. Менеджер:

**Роль:** Менеджер.

**Права:**

* Создание, изменение и удаление записей в таблицах: ticket, even, group, musician
* Просмотр записей в таблицах: holl, place, musician

Таблица прав доступа к таблицам:

|  |  |  |
| --- | --- | --- |
| **Роль** | **Таблица** | **Права доступа** |
| Администратор | ticket, holl, even, group, place, musician, zone, audit\_log, users. | Чтение, Запись, Изменение, Удаление, Обновление |
| Менеджер | ticket, even, group, musician | Чтение, Запись, Изменение, Удаление, Обновление |
| Менеджер | holl, place, musician | Чтение |
| Пользователь | ticket, holl, even, group, place, musician, zone | Чтение |

1. **Тщательный контроль доступа (RLS).** Для выбранной таблицы базы данных реализовать контроль доступа на уровне строк с использованием представлений. Для каждой введенной в п.6 роли и/или пользователя уровень доступа (подмножество выводимых строк) должен быть различен.

Сделаем RLS для первой таблицы так, чтобы пользователи могли видеть информацию лишь о себе, т.е. – по своим именам.

Создаем пользователей

Create role sec with login nosuperuser nocreatedb nocreaterole INHERIT noreplication connection limit -1;

Grant Client to sec;

1. **Политика аудита.** Должна быть описана политика аудирования событий уровня таблицы и уровня базы данных, приведены скрипты задания этой политики в СУБД, а также результаты работы этой политики (выдержки из журнала аудита).

*1. Где хранятся триггеры?*

*Триггеры хранятся в системной таблице pg\_trigger базы данных PostgreSQL. Информация о триггерах, их функциях и условиях находится в этой таблице.*

*2. Как посмотреть к каким таблицам они прикреплены?*

*Для просмотра триггеров, прикрепленных к таблицам, можно использовать запрос к таблице pg\_trigger.*

SELECT tgname, tgrelid::regclass

FROM pg\_trigger;

*Этот запрос вернет имена триггеров и таблиц, к которым они прикреплены.*

*3. Как посмотреть код триггера?*

*Код триггера можно получить из системной функции pg\_get\_triggerdef. Пример:*

SELECT pg\_get\_triggerdef(oid)

FROM pg\_trigger;

*Здесь oid - это идентификатор объекта триггера, который можно получить, например, из запроса в предыдущем ответе.*

*4. Как посмотреть, что настроено в системе?*

*Для просмотра информации о триггерах, таблицах и других объектах базы данных можно использовать различные запросы к системным таблицам PostgreSQL.*

*Для просмотра всех триггеров:*

SELECT \* FROM pg\_trigger;

*Для просмотра всех таблиц:*

SELECT \* FROM pg\_tables WHERE schemaname = 'public';

*Для получения дополнительной информации о таблицах:*

SELECT \* FROM information\_schema.tables WHERE table\_schema = 'public';

*5. Какие объекты базы данных хранят настройки?*

*Настройки базы данных в PostgreSQL хранятся в нескольких системных объектах:*

* *`pg\_settings`: Таблица, содержащая текущие параметры конфигурации PostgreSQL. Запрос к ней выглядит так:*

SELECT name, setting FROM pg\_settings;

* *Конфигурационные файлы: postgresql.conf, который предоставляет глобальные параметры, а также в файлах pg\_hba.conf и pg\_ident.conf, которые управляют аутентификацией.*
* *Функции и представления, которые предоставляют информацию о текущей конфигурации. Например, current\_setting()*
* *Таблица `pg\_database`: В таблице pg\_database содержится информация о каждой базе данных, включая параметры, связанные с этой базой данных.*
* *Таблица `pg\_user`: Содержит информацию о пользователях, включая их роли и права.*

**SQL-код реализации политики аудита с использованием триггеров и функций**

CREATE TABLE audit\_log (

event\_time timestamptz,

event\_type text,

user\_name text,

object\_schema text,

object\_name text,

statement text,

affected\_data jsonb

);

CREATE OR REPLACE FUNCTION audit\_if\_modified\_func() RETURNS TRIGGER AS $$

DECLARE

old\_data jsonb;

new\_data jsonb;

BEGIN

old\_data = row\_to\_json(OLD);

new\_data = row\_to\_json(NEW);

IF (TG\_OP = 'INSERT') THEN

INSERT INTO audit\_log (event\_time, event\_type, user\_name, object\_schema, object\_name, statement, affected\_data)

VALUES (now(), 'INSERT', current\_user, TG\_TABLE\_SCHEMA, TG\_TABLE\_NAME, 'INSERT', new\_data);

ELSIF (TG\_OP = 'UPDATE') THEN

INSERT INTO audit\_log (event\_time, event\_type, user\_name, object\_schema, object\_name, statement, affected\_data)

VALUES (now(), 'UPDATE', current\_user, TG\_TABLE\_SCHEMA, TG\_TABLE\_NAME, 'UPDATE', jsonb\_build\_object('old', old\_data, 'new', new\_data));

ELSIF (TG\_OP = 'DELETE') THEN

INSERT INTO audit\_log (event\_time, event\_type, user\_name, object\_schema, object\_name, statement, affected\_data)

VALUES (now(), 'DELETE', current\_user, TG\_TABLE\_SCHEMA, TG\_TABLE\_NAME, 'DELETE', old\_data);

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER audit\_if\_modified\_trigger\_ticket

AFTER INSERT OR UPDATE OR DELETE ON ticket

FOR EACH ROW EXECUTE FUNCTION audit\_if\_modified\_func();

CREATE TRIGGER audit\_if\_modified\_trigger\_event

AFTER INSERT OR UPDATE OR DELETE ON event

FOR EACH ROW EXECUTE FUNCTION audit\_if\_modified\_func();

CREATE TRIGGER audit\_if\_modified\_trigger\_group

AFTER INSERT OR UPDATE OR DELETE ON group

FOR EACH ROW EXECUTE FUNCTION audit\_if\_modified\_func();

CREATE TRIGGER audit\_if\_modified\_trigger\_musician

AFTER INSERT OR UPDATE OR DELETE ON musician

FOR EACH ROW EXECUTE FUNCTION audit\_if\_modified\_func();

CREATE TRIGGER audit\_if\_modified\_trigger\_holl

AFTER INSERT OR UPDATE OR DELETE ON holl

FOR EACH ROW EXECUTE FUNCTION audit\_if\_modified\_func();

CREATE TRIGGER audit\_if\_modified\_trigger\_zone

AFTER INSERT OR UPDATE OR DELETE ON zone

FOR EACH ROW EXECUTE FUNCTION audit\_if\_modified\_func();

CREATE TRIGGER audit\_if\_modified\_trigger\_place

AFTER INSERT OR UPDATE OR DELETE ON place

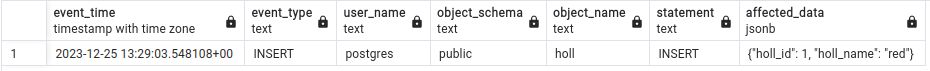
FOR EACH ROW EXECUTE FUNCTION audit\_if\_modified\_func();

CREATE TRIGGER audit\_if\_modified\_trigger\_users

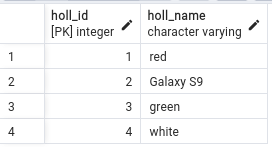
AFTER INSERT OR UPDATE OR DELETE ON users

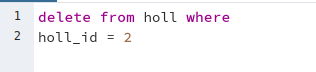
FOR EACH ROW EXECUTE FUNCTION audit\_if\_modified\_func();

**Пример записи в таблице логов**

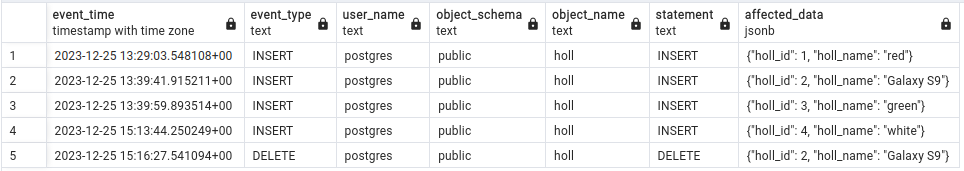


**Пример использования настроенной политики аудита**





**Вид таблицы логов после выполнения этих действий**



**Использование стандартных средств аудита.**

Для реализации **аудита уровня базы данных**, используем стандартные средства PostgreSQL. Для настройки надо отредактировать конфигурационный файл.

ddz=# show config\_file;

config\_file

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

/etc/postgresql/16/main/postgresql.conf

(1 row)

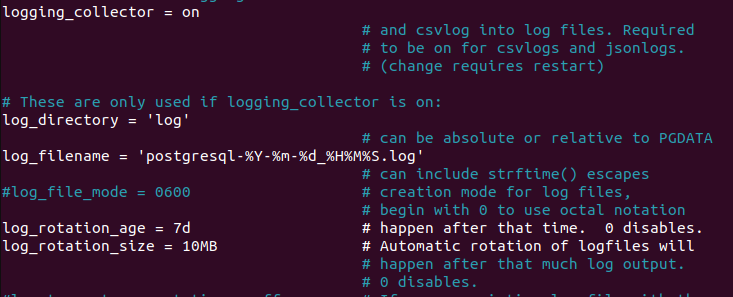
ddz=# show data\_directory;

data\_directory

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

/var/lib/postgresql/16/main

Изменения в конфигурационном файле



sudo service postgresql restart

sudo systemctl restart postgresql

**SQL код реализации политики аудита уровня базы данных**

-- Создание таблицы для логов событий уровня базы данных

CREATE TABLE ddl\_logs (

id serial primary key,

ddl\_date timestamptz,

ddl\_tag text,

object\_name text,

username text

);

-- Создание триггерной функции для событий уровня базы данных кроме

CREATE OR REPLACE FUNCTION log\_ddl() RETURNS event\_trigger AS $$

DECLARE

r RECORD;

BEGIN

IF tg\_tag <> 'DROP TABLE' THEN

r := pg\_event\_trigger\_ddl\_commands();

INSERT INTO ddl\_logs (ddl\_date, ddl\_tag, object\_name, username) VALUES (statement\_timestamp(), tg\_tag, r.object\_identity,current\_user);

END IF;

END;

$$ LANGUAGE plpgsql;

-- Создание триггерной функции события DROP

CREATE OR REPLACE FUNCTION log\_ddl\_drop() RETURNS event\_trigger AS $$

DECLARE

r RECORD;

BEGIN

FOR r IN SELECT \* FROM pg\_event\_trigger\_dropped\_objects()

LOOP

INSERT INTO ddl\_logs (ddl\_date, ddl\_tag, object\_name,username) VALUES (statement\_timestamp(), tg\_tag, r.object\_identity,current\_user);

return;

END LOOP;

END;

$$ LANGUAGE plpgsql;

-- Создание триггеров событий уровня базы данных

CREATE EVENT TRIGGER log\_ddl\_info ON ddl\_command\_end EXECUTE PROCEDURE log\_ddl();

CREATE EVENT TRIGGER log\_ddl\_drop\_info ON sql\_drop EXECUTE PROCEDURE log\_ddl\_drop();

--Проверка

CREATE TABLE testing\_ddl(id integer not null, data\_str text);

ALTER TABLE testing\_ddl add column another text;

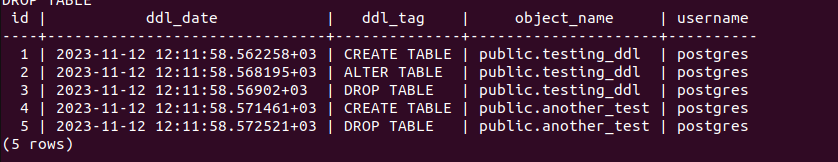
DROP TABLE testing\_ddl;

CREATE TABLE another\_test(id integer);

DROP TABLE another\_test;

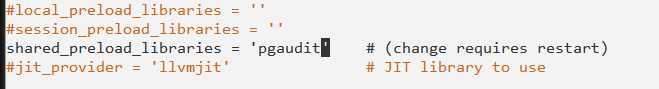
select \* from ddl\_logs;

**Пример вида таблицы ddl\_logs после выполнения проверочных действий из скрипта**



**Настройка pgaudit**

1. В файле **/etc/postgresql/16/main/postgresql.conf**



1. После изменения **/etc/postgresql/16/main/postgresql.conf** необходимо перезапустить postgres



1. В базе данных



1. В файле **/etc/postgresql/16/main/postgresql.conf** в самом низу допишем собственноручно



1. После изменения **/etc/postgresql/16/main/postgresql.conf** необходимо перезапустить postgres



1. **Политика тщательного аудита (FGA).** Должна быть описана политика аудирования событий уровня строк выбранной слушателем таблицы, приведены соответствующие скрипты задания этой политики в СУБД, а также результаты работы этой политики (выдержки из журнала аудита).

Рассмотрена в предыдущем пункте.

1. **Контроль целостности.** Необходимо реализовать проверку правил (ограничений) предметной области с помощью декларативных (CONSTRAINT) и процедурных (триггеры уровня строки/таблицы) средств контроля целостности.

**Контроль цены на билеты:**

CREATE OR REPLACE FUNCTION check\_price()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.cost <= 0 THEN

RAISE EXCEPTION 'Цена должна быть больше 0';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_price\_trigger

BEFORE INSERT OR UPDATE ON zone

FOR EACH ROW EXECUTE FUNCTION check\_price();

**Контроль возможности изменения названия групп. Это предотвращает изменение названия группы, если есть связанные музыканты:**

CREATE OR REPLACE FUNCTION check\_name\_change\_trigger()

RETURNS TRIGGER AS $$

BEGIN

IF EXISTS (SELECT 1 FROM group WHERE group\_id = NEW.group \_id) THEN

RAISE EXCEPTION 'Изменение названия группы невозможно: есть связанные музыканты';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_name\_change\_trigger

BEFORE UPDATE ON group

FOR EACH ROW EXECUTE FUNCTION check\_name\_change\_trigger();

**Контроль минимальной длины названия хола. Это гарантирует, что название хола имеет минимальную длину 3 символа при добавлении или изменении:**

CREATE OR REPLACE FUNCTION check\_holl\_name\_length\_trigger()

RETURNS TRIGGER AS $$

BEGIN

IF LENGTH(NEW.holl\_name) < 3 THEN

RAISE EXCEPTION 'Минимальная длина названия зоны - 3 символа';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_holl\_name\_length\_trigger

BEFORE INSERT OR UPDATE ON holl

FOR EACH ROW EXECUTE FUNCTION check\_holl\_name\_length\_trigger();

1. **Шифрование данных.** Необходимо настроить и реализовать две технологии шифрования данных в БД:

- Выполним второй пункт шифрования, а именно – шифрование симметричным ключом, таким образом, чтобы при извлечении данные были видны незашифрованными.

Создадим таблицу с пользователями и их ключами

create table keys(

name text,

key text);

INSERT INTO KEYS(NAME, KEY) VALUES('lamelo', 'ball'),('steve', 'king'),('postgres', 'admin');

Теперь введём специальную сущность – таблицу для проверки работоспособности данного пункта, в ней будут заключены имена пользователей и номера их договоров на оплату.

create table contracts(

name text,

contract text

);

Создадим необходимое функции pgp\_sym\_encrypt расширение, а также триггер и функцию для зашифрования при вставке.

*create trigger tr\_con\_ins*

*planes-# before*

*planes-# insert on contracts*

*planes-# for each row*

*planes-# execute function con\_inst();*

*CREATE EXTENSION pgcrypto;*

*create or replace function con\_inst() returns trigger*

*as $con\_ins$*

*planes$# begin*

*planes$# IF current\_user NOT IN (SELECT name FROM keys) THEN*

*planes$# RAISE EXCEPTION 'Current user dont have key';*

*planes$# end if;*

*planes$# NEW.contract := pgp\_sym\_encrypt(NEW.contract,(select*

*key from keys where name = current\_user));*

*planes$# return NEW;*

*planes$# end;*

*planes$# $con\_ins$ language plpgsql*

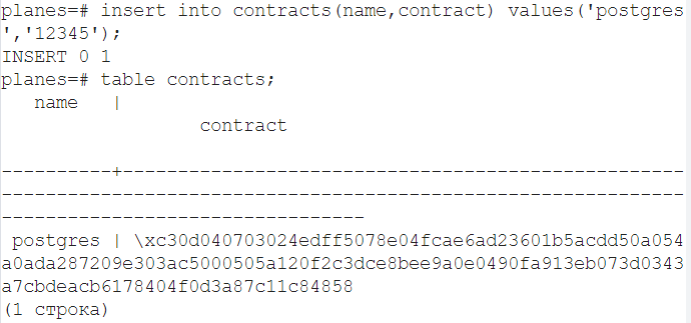
*planes-# ;*

Проверим работоспособность решения.

i*nsert into contracts(name,contract) values('postgres','12345');*

*INSERT 0 1*

*planes=# select \* from contracts;*



Займемся ограничением доступа к таблице keys. Выдадим первоначально право на select, затем добавим RLS на все тот же select по имени пользователя в поле name этой таблицы.

*GRANT SELECT ON TABLE public.keys TO users;*

*alter table keys enable row level security;*

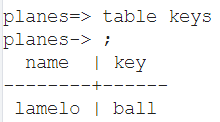
*create policy view\_only\_self on keys for select to users using (name = current\_user);*

*Проверим работоспособность*

*set role lamelo;*

*SET*

*table keys*



Добавим обработку случая обновления данных

*create or replace function con\_upt() returns trigger as $*

*con\_upt$*

*begin*

*IF current\_user NOT IN (select name from keys) then*

*RAISE EXCEPTION 'Current user doesnt have key';*

*end if;*

*IF current\_user != NEW.name then*

*RAISE EXCEPTION 'You cannot update data not yours';*

*end if;*

*NEW.contract := pgp\_sym\_encrypt(NEW.contract::text, (sele*

*ct key from keys where name=current\_user));*

*return NEW;*

*end;*

*$con\_upt$ language plpgsql;*

*create trigger tr\_con\_upt*

*before*

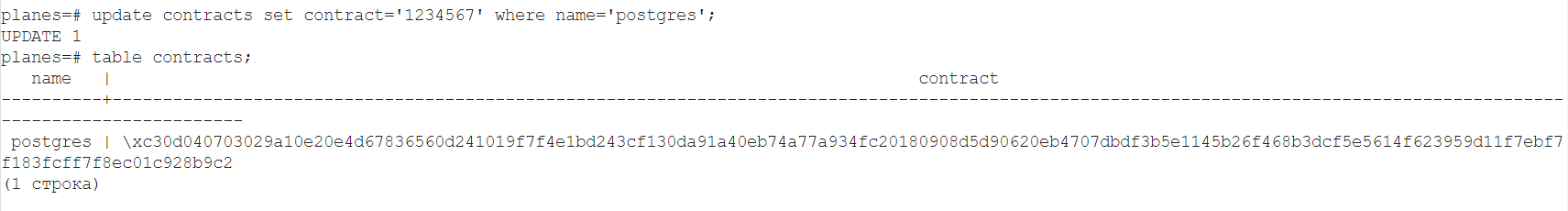
*update on contracts*

*for each row*

*execute function con\_upt();*

*GRANT SELECT, INSERT, UPDATE ON TABLE public.contracts TO users;*

Проверим работоспособность



И, наконец, сделаем процедуру для просмотра содержимого в открытом виде, естественно, только для своего пользователя.

*create or replace function sel\_con() returns text as $$*

*planes$# declare*

*planes$# name\_p text;*

*planes$# contract\_p text;*

*planes$# begin*

*planes$# if current\_user not in (select name from keys) then*

*planes$# raise exception 'current user doesnt have key';*

*planes$# end if;*

*planes$# if current\_user not in (select name from contracts) then*

*planes$# raise exception 'you dont have any data in contracts';*

*planes$# end if;*

*planes$# select name, pgp\_sym\_decrypt(contract::bytea,(select key from ke*

*ys where name=current\_user)) into name\_p, contract\_p from contracts where*

*name=current\_user;*

*planes$# return format('%s : %s', name\_p, contract\_p);*

*planes$# end;*

*planes$# $$ language plpgsql;*

*insert into contracts(name,contract) values('postgres*

*',pgp\_sym\_encrypt('12345', (select key from keys where name=cu*

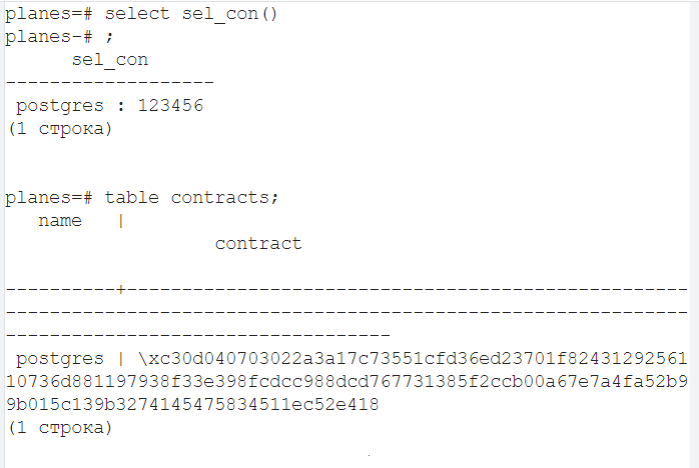
*rrent\_user)));*

*select pgp\_sym\_decrypt((select contract from contract*

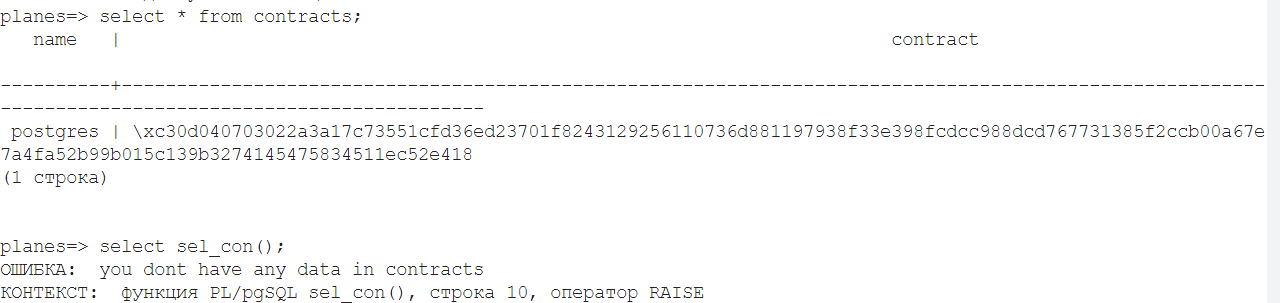
*s where name=current\_user), (select key from keys where name=c*

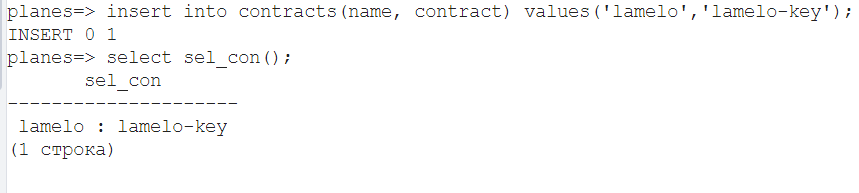
*urrent\_user));*

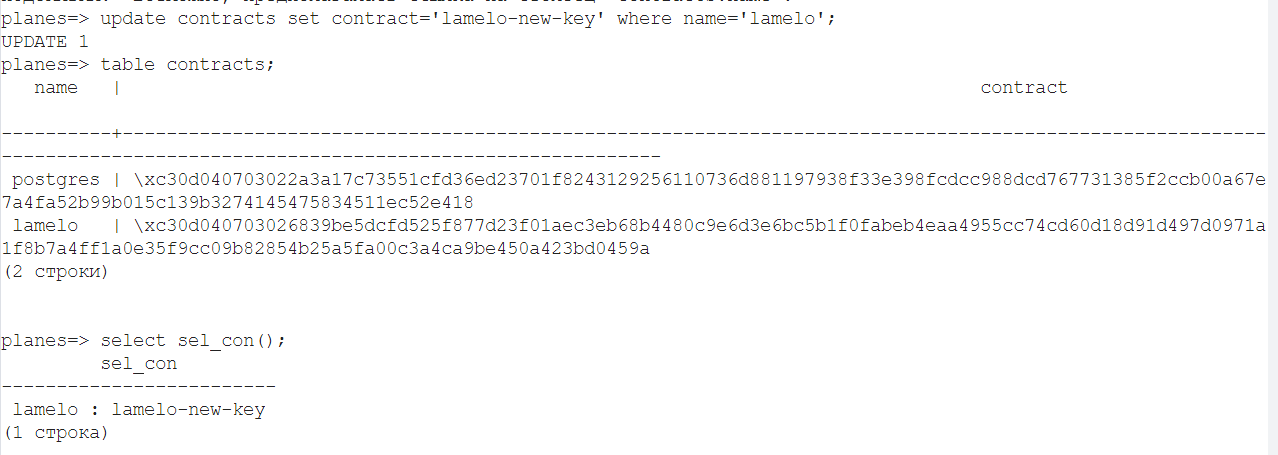
Проверим работоспособность. Видим, что при вызове функций данные видны, тогда как через обычный select – скрыты.



При этом для другого пользователя данные функцией видны не будут – он их не добавлял.

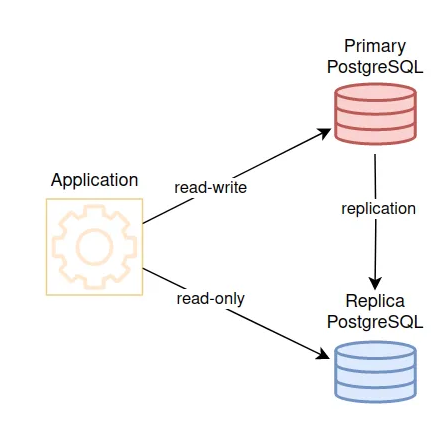






1. **Настройка политики резервного копирования фалов данных, конфигурационных файлов и файлов журналов транзакций.** Требуется настроить политику резервирования путем создания репликации сервера в режиме «master-slave».

Настроим с использование docker-compose:

Схема:

Настройка PostgreSQL включает в себя два файла: 00\_init.sql SQL-скрипт инициализации и docker-compose.yaml описание докер-сервисов основного и реплики.

Структура каталога:



Скрипт инициализации используется для создания слота репликации и пользователя, который будет использовать этот слот. Этот файл монтируется в каталог контейнера /docker-entrypoint-initdb.d, который является каталогом для сценариев инициализации.

Скрипт:

*CREATE USER replicator WITH REPLICATION ENCRYPTED PASSWORD 'replicator\_password';  
SELECT pg\_create\_physical\_replication\_slot('replication\_slot');*

В файле описаны две службы:

1) Основной экземпляр PostgreSQL с определенными POSTGRES\_HOST\_AUTH\_METHOD(для насройки доступа)

2) Реплика с флагами запуска -R и --slot=replication\_slot

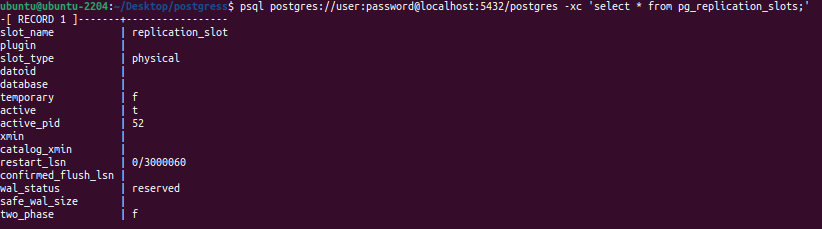
1)-R - запуск реплики в режиме постоянного восстановления.

2) —slot=replication\_slot задает использование слота репликации, который был создан в 00\_init.sqlскрипте.

Docker-compose:

*version: '3.8'  
x-postgres-common:  
&postgres-common  
image: postgres:14-alpine  
user: postgres  
restart: always  
healthcheck:  
test: 'pg\_isready -U user --dbname=postgres'  
interval: 10s  
timeout: 5s  
retries: 5  
  
services:  
postgres\_primary:  
<<: \*postgres-common  
ports:  
- 5432:5432  
environment:  
POSTGRES\_USER: user  
POSTGRES\_DB: postgres  
POSTGRES\_PASSWORD: password  
POSTGRES\_HOST\_AUTH\_METHOD: "scram-sha-256\nhost replication all 0.0.0.0/0 md5"  
POSTGRES\_INITDB\_ARGS: "--auth-host=scram-sha-256"  
command: |  
postgres   
-c wal\_level=replica   
-c hot\_standby=on   
-c max\_wal\_senders=10   
-c max\_replication\_slots=10   
-c hot\_standby\_feedback=on  
volumes:  
- ./00\_init.sql:/docker-entrypoint-initdb.d/00\_init.sql  
  
postgres\_replica:  
<<: \*postgres-common  
ports:  
- 5433:5432  
environment:  
PGUSER: replicator  
PGPASSWORD: replicator\_password  
command: |  
bash -c "  
until pg\_basebackup --pgdata=/var/lib/postgresql/data -R --slot=replication\_slot --host=postgres\_primary --port=5432  
do  
echo 'Waiting for primary to connect...'  
sleep 1s  
done  
echo 'Backup done, starting replica...'  
chmod 0700 /var/lib/postgresql/data  
postgres  
"  
depends\_on:  
- postgres\_primary*

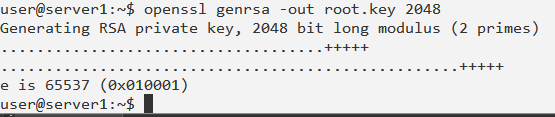
**Тестирование:**

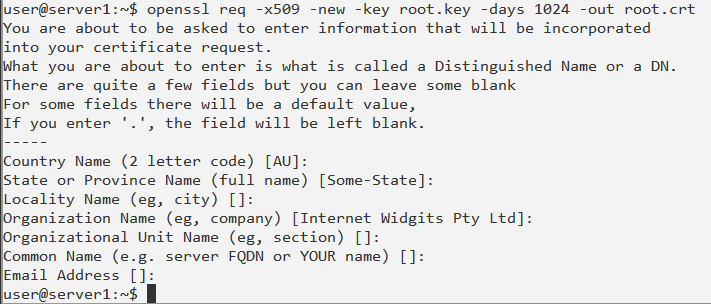
****

1. **Настройка внешней аутентификации.** Необходимо настроить взаимную аутентификацию пользователей и/или рабочих мест и сервера по протоколу SSL и Kerberos. В отчете необходимо привести последовательность действий по настройке аутентификации, а также результаты ее работы (листинги сетевого перехвата, подтверждающие использование протокола SSL или Kerberos для аутентификации пользователя СУБД).

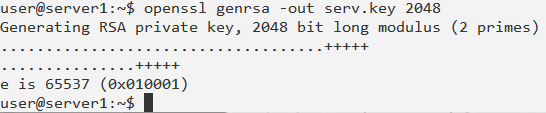
**На сервере**

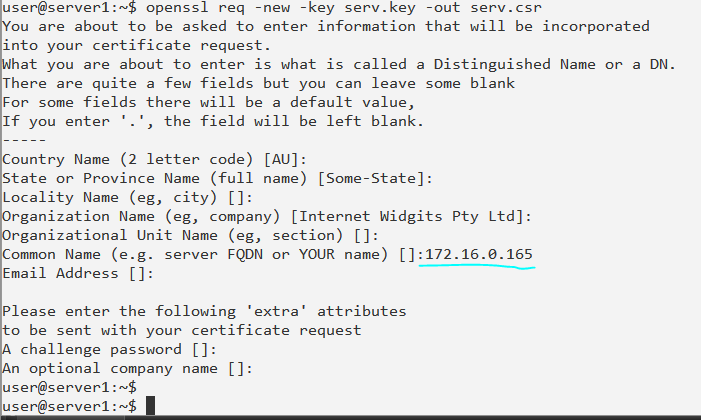
Создадим закрытый ключ и корневой сертификат



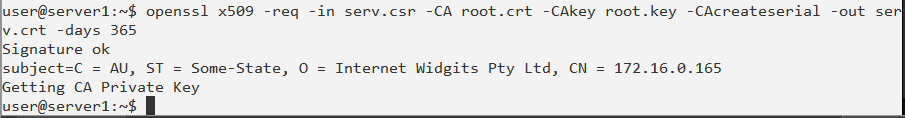


Создадим для сервера ключ и запрос на сертификат

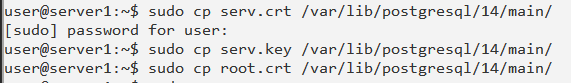




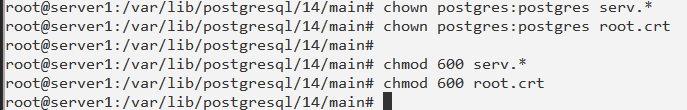
Далее подпишем сертификат сервера



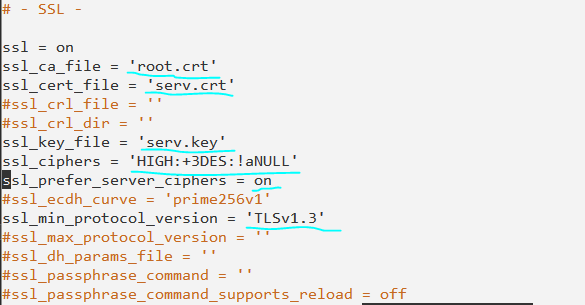
Скопируем корневой сертификат, ключ и сертификат сервера в /var/lib/postgresql/16/main



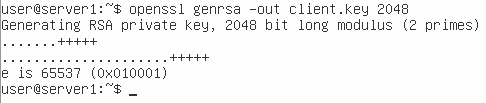
Назначим права и маски согласно требованиям postgres

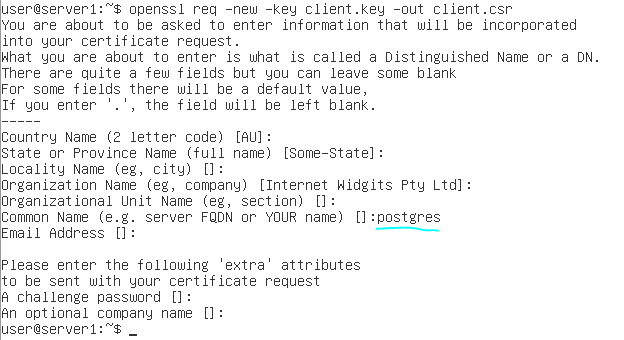


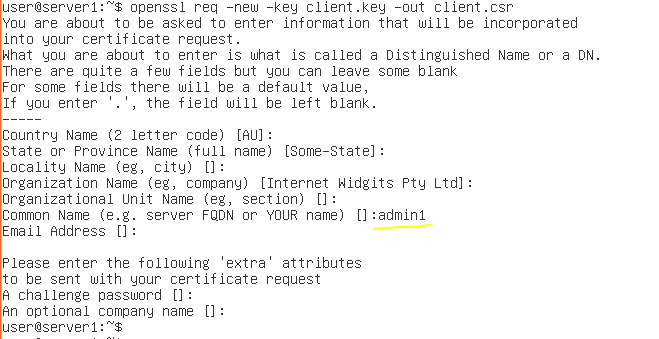
Настроим файл postgresql.conf



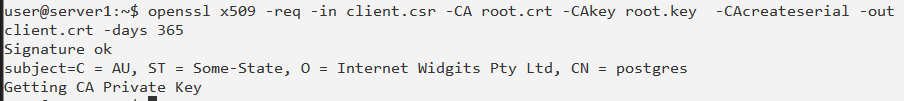
Затем

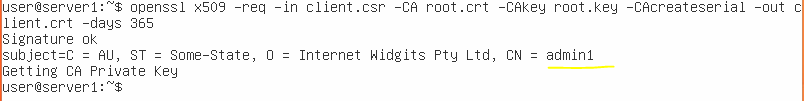






Подпишем сертификат





**На клиенте**

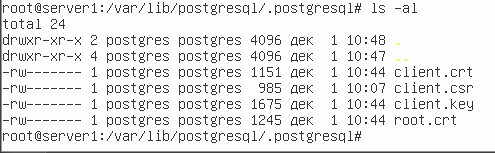
Скопируем ключи и сертификаты клиента, и корневой сертификат с сервера



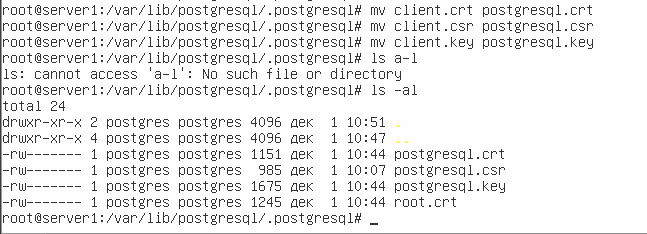
Создадим согласно документации скрытую папку



И вставим в нее то, что скачали выше, в итоге должно получиться так



Далее переименуем



1. **Настройка шифрования трафика**. Доступ клиента к серверу должен быть организован по каналу, защищенному с применением технологии SSL/TLS.

Рассмотрена в предыдущем пункте