ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего профессионального образования
Национальный исследовательский университет
«Высшая школа экономики»

Московский институт электроники и математики

Департамент прикладной математики

Образовательная программа «Компьютерная безопасность»

Отчёт по практической работе по дисциплине «СУБД»

Домашняя практическая работа № 2
«Администрирование баз данных»

Вариант 5 «Фильмотека»

Выполнил:
Студент группы *СКБ182*
Шаплавский Леонид Павлович

Проверил:
Профессор, Белов А.В.

Москва-2022

Постановка задачи

Модель «Фильмотека» должна содержать информацию о кинофильмах, актерах, киностудиях и контрактах между актерами и киностудиями

Использовать следующие справочники и классификаторы: - справочник жанров кинофильмов
- амплуа актера (комик, травести, универсал, статист и т.п.) - справочник стран

Отчеты:

1. Получить список всех актеров, снимающихся на заданной киностудии;
2. Получить список кинофильмов, в которых заданный актер не снялся ни разу;
3. Получитьпары(ФИОактера,Nоконтракта),занятыхвфильмах,выпущенных

   на заданной киностудии;

4. Получить список киностудий, в которых были сняты фильмы по заданной

   тематике и в заданном году;

5. Получить список актеров, у которых имеется контракт с киностудией,

   расположенной в том же городе, в каком проживают эти актеры;

6. Получить список кинофильмов, в которых снимаются те же актеры, что и в заданном кинофильме. Вывести название фильма, количество совпадающих

   актеров;

7. Получитьсписокизтрехсамыхмалочисленныхжанров,зарегистрированных

   в фильмотеке (т.е. фильмов с таким жанром меньше всех остальных), с

   указанием процента фильмов от общего числа;

8. Получить список из трех самых плодовитых режиссеров за указанное

   десятилетие. Вывести ФИО, количество фильмов, название и дату выхода последнего фильма (за указанный срок).

Задачи:

Задача 1. Создать базу данных, спроектированную в ходе выполнения предыдущей практической работы, в любой SQL среде (предпочтительно использование MS

SQL). Скрипты, создающие БД и ее объекты, должны быть представлены в отчете для последующего тестирования.

Задача 2. Предусмотреть ограничения целостности данных в базе, а также триггеры, пересчитывающие значение вычислимого атрибута, где это необходимо (исходя из бизнес-правил задания).

Задача 3. Импортировать и экспортировать данные в созданную базу с использованием средств служб DTS и языка SQL:

- с помощью SQL-команды (например, BULK INSERT или OPENROWSET(BULK...) для MS SQL; COPY для Postgres SQL или прочее);
- с помощью мастера импорта и экспорта.
  Данные для экспорта/импорта должны быть представлены в текстовом формате .txt и формате .xls

Задача 4. Сформировать запросы к построенной базе данных информационной системы в соответствии с выбранной моделью и заданием No2 предыдущей практической работы. Предпочтительно оформить их в виде представлений или хранимых процедур (смотря что будет целесообразнее в конкретном случае). В отчете для каждого запроса (где это возможно по заданию) приготовить демонстрацию всех возможных событий, а именно:

- результат запроса – пустая таблица, т.е. нет данных, подходящих под условия;
- результат запроса представляет собой одну строку / значение, т.е. под все условия походит только одна запись;
- результат запроса представляет собой набор значений, т.е. под все условия подходят не менее двух записей.

Тексты представлений или хранимых процедур должны быть представлены в отчете для проверки их работоспособности.

Задача5. Создать хранимую процедуру по внесению новой записи сразу в два отношения, связанные внешним ключом.

Например, при наличии таблицы с пользовательскими данными, и отдельной таблицы с телефонами пользователей, можно создать хранимую процедуру, которая принимает входные параметры – данные пользователя и телефон. Результатом выполнения процедуры будет – внесение новой записи в обе таблицы, если пользователь новый, и внесение записи в таблицу телефонов, если пользователь уже существует, а такого телефона за ним не закреплено. Если и этот пользователь, и этот телефон (закрепленный за этим пользователем) уже хранятся в базе, хранимая процедура может вывести сообщение об этом.

Задача 6. Создать триггер, обрабатывающий изменение и удаление записи в любом справочнике. Рекомендуемое поведение – сохранение старой записи в отдельной таблице с указанием данных инициатора изменений.

Задача7. Создать набор пользователей БД, разработанный в ходе предыдущей практической работы, и разграничение прав доступа к объектам БД для разных пользователей. В отчете приготовить демонстрацию обращения пользователей к допустимым и недопустимым для них объектам.

Задача 8. Создать представления для каждой группы пользователей (не менее одного на группу), которые будут содержать достаточную информацию для ответов на предполагаемые запросы группы.

Задача9. Настроить шифрование любого атрибута любой таблицы. Создать представление, возвращающее данные в расшифрованном виде. Предусмотреть ограниченный доступ к этому представлению.

Задача 10. Создать резервную копию БД, удалить ее и восстановить БД по резервной копии. В отчете приготовить демонстрацию всех перечисленных действий.

Задача 11. Подготовить и загрузить отчет о выполнении практической работы в Smart LMS.

Выбранная среда реализации работы

PostgreSQL
Python 3
PyCharm
DBeaver

Создание БД и настройка ограничений целостности

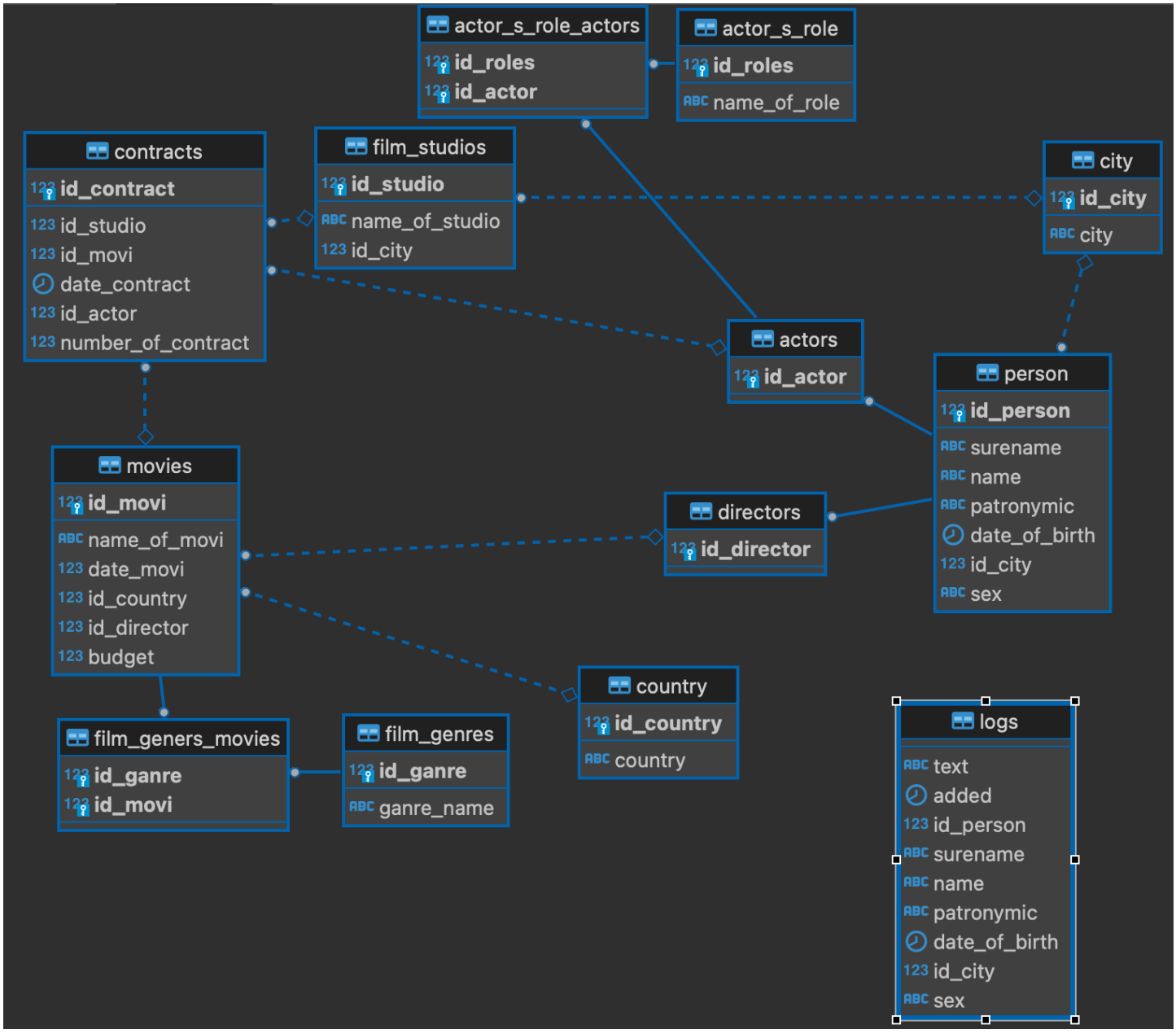


Рис. 1. Логическая схема демонстрационной базы

Код создания базы данных, базовых объектов размещен и настройки ограничения целостности в приложении 1.

Для ограничения целостности практически все переменные должны быть заполнены (NOT NULL) и была сделана проверка, чтобы значение было больше нуля.

Импорт и экспорт данных

С помощью Python были созданы данные (приложение 2) и записаны в файл с расширение .txt и с помощью SQL запросов COPY TO и COPY FROM были экспортированы и импортированы данные в БД приложении 1.

Формирование запросов к БД на языке SQL

1 запрос

```
select * from (select id_actor, name_of_studio from
((actors inner join contracts using (id_actor))
inner join film_studios using (id_studio))) as step1
inner join person on (step1.id_actor = id_person)
where step1.name_of_studio = 'Emelda'
```

2 запрос

```
with step1 as (select * from ((movies inner join contracts using (id_movi))
inner join actors using (id_actor)))
select step1.id_actor, step1.name_of_movi from step1, actors
where step1.id_actor != actors.id_actor
```

3 запрос

```
select * from (select id_actor, name_of_studio, number_of_contract from
((actors inner join contracts using (id_actor))
inner join film_studios using (id_studio))) as step1
inner join person on (step1.id_actor = id_person)
where step1.name_of_studio = 'Emelda' as step2
```

4 запрос

```
select name_of_studio from ((((film_geners_movies
inner join film_genres using (id_ganre))
inner join movies using (id_movi))
inner join contracts using (id_movi))
inner join film_studios using (id_studio))
where (ganre_name = 'Horror') and (date_movi = 1997)
```

5 запрос

```
select * from ((((actors inner join contracts using (id_actor))
inner join film_studios using (id_studio)))
inner join city using(id_city)) as step1
inner join person on (step1.id_actor = id_person)
where step1.id_city = person.id_city
```

Создание хранимой процедуры

Были выбраны соотношения город и пользователь. При вызове запроса проверяется был ли создан город с таким Id и названием и проверяется был ли создан пользователь с таким ID.

```
create or replace procedure insert_data( city_name varchar, city_id int, person_id int,
last_name varchar, name_person varchar, mid_name varchar, date_per date, sex_per varchar)
language plpgsql
as $$
begin
      begin
            if (exists (select * from city where id_city = city_id) or exists
```

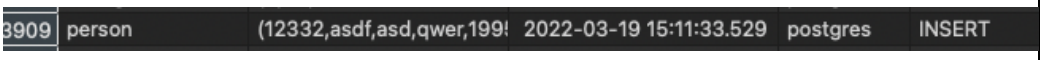
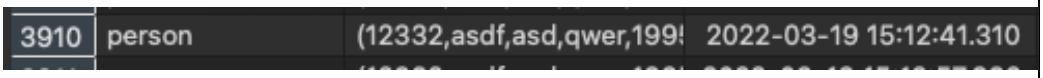
```
            (select * from city where city = city_name)) then
            raise notice 'Запись с таким city_id уже есть';

            else insert into city values (city_id, city_name);
            raise notice 'Занесена запись в таблицу city';
            end if;
      end;

      begin
            if exists (select * from person where id_person = person_id) then
            raise notice 'Запись с таким person_id уже есть';

            else insert into person  values (person_id, last_name,
      name_person, mid_name, date_per, city_id, sex_per);
            raise notice 'Занесена запись в таблицу person';
            end if;
      end;

END;
$$;
```

Запрос	Демонстрация результата	Комментарий
`CALL insert_data('Mosccc', 11234, 112234, 'Shaplavskiy', 'Leonid', 'Pavlovich', '12-29-1999', 'men') ;`	Занесена запись в таблицу city Занесена запись в таблицу person	Успешная запись города и пользователя
`CALL insert_data('Mosccc', 11234, 112234, 'Shaplavskiy', 'Leonid', 'Pavlovich', '12-29-1999', 'men') ;`	Запись с таким city_id уже есть Запись с таким person_id уже есть	Так как уже существует такие записи - не возможно записать в таблицу
`CALL insert_data('Mosccc', 11234, 212234, 'Shaplavskiy', 'Leonid', 'Pavlovich', '12-29-1999', 'men') ;`	Запись с таким city_id уже есть Занесена запись в таблицу person	Записан только пользователь, поскольку пользователя с таким ID ещё не было
`CALL insert_data('Mosccc', 1234, 212234, 'Shaplavskiy', 'Leonid', 'Pavlovich', '12-29-1999', 'men') ;`	Запись с таким city_id уже есть Запись с таким person_id уже есть	Не возможно записать город поскольку город с таким названием уже есть в таблице

Создание триггера обработки изменений в справочнике

```
CREATE OR REPLACE FUNCTION add_to_log() RETURNS TRIGGER AS $$

    BEGIN
        IF  TG_OP = 'INSERT' or TG_OP = 'UPDATE' THEN
            INSERT INTO logs(tablename,text,added, "user", op_type) values
(TG_TABLE_NAME, NEW,NOW(), user, TG_OP);
```

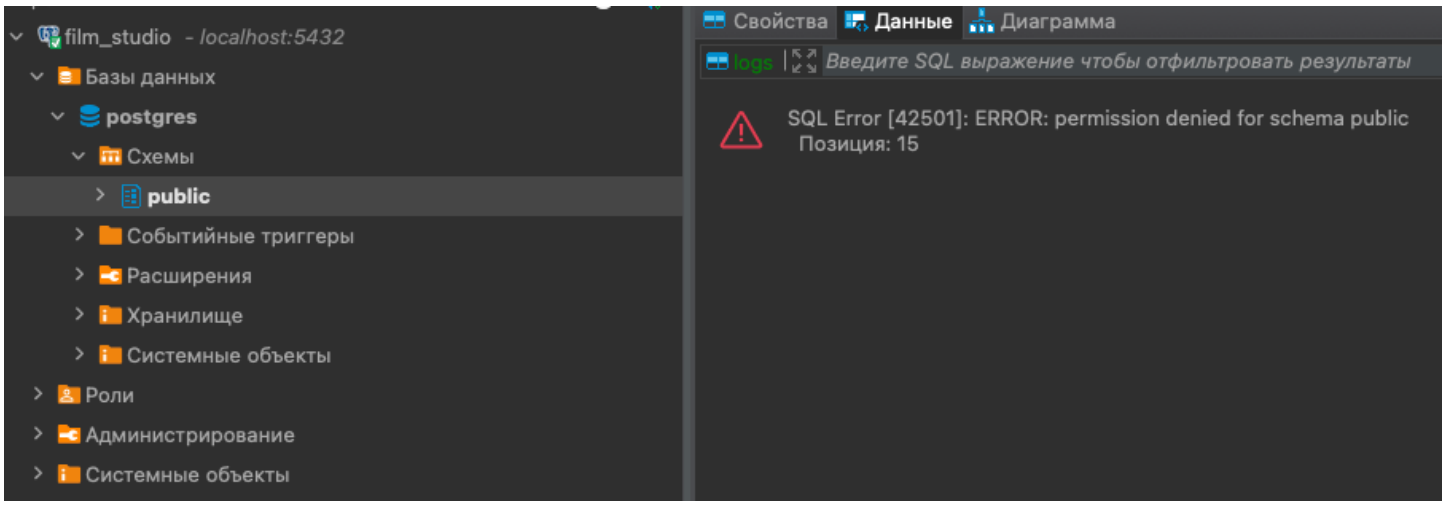
```
            RETURN NEW;
        ELSIF TG_OP = 'DELETE' THEN
            INSERT INTO logs(tablename,text,added, "user", op_type) values
(TG_TABLE_NAME, OLD,NOW(), user, TG_OP);
            RETURN OLD;
        END IF;
    END;
    $$ LANGUAGE plpgsql;

      CREATE TRIGGER user_person
    AFTER INSERT OR UPDATE OR DELETE ON person FOR EACH ROW EXECUTE PROCEDURE add_to_log
();

    CREATE TRIGGER user_city
    AFTER INSERT OR UPDATE OR DELETE ON city FOR EACH ROW EXECUTE PROCEDURE add_to_log
();

    CREATE TRIGGER user_actors
    AFTER INSERT OR UPDATE OR DELETE ON actors for EACH ROW EXECUTE PROCEDURE add_to_log
();

    CREATE TRIGGER user_directors
    AFTER INSERT OR UPDATE OR DELETE ON directors for EACH ROW EXECUTE PROCEDURE
add_to_log ();

    CREATE TRIGGER user_country
    AFTER INSERT OR UPDATE OR DELETE ON country for EACH ROW EXECUTE PROCEDURE add_to_log
();

    CREATE TRIGGER user_film_genres
    AFTER INSERT OR UPDATE OR DELETE ON film_genres for EACH ROW EXECUTE PROCEDURE
add_to_log ();

    CREATE TRIGGER user_film_geners_movies
    AFTER INSERT OR UPDATE OR DELETE ON film_geners_movies for EACH ROW EXECUTE PROCEDURE
add_to_log ();

    CREATE TRIGGER user_movies
    AFTER INSERT OR UPDATE OR DELETE ON movies for EACH ROW EXECUTE PROCEDURE add_to_log
();

    CREATE TRIGGER user_contracts
    AFTER INSERT OR UPDATE OR DELETE ON contracts for EACH ROW EXECUTE PROCEDURE
add_to_log ();

    CREATE TRIGGER user_film_studios
    AFTER INSERT OR UPDATE OR DELETE ON film_studios for EACH ROW EXECUTE PROCEDURE
add_to_log ();

    CREATE TRIGGER user_actor_s_role_actors
    AFTER INSERT OR UPDATE OR DELETE ON actor_s_role_actors for EACH ROW EXECUTE
PROCEDURE add_to_log ();

    CREATE TRIGGER actor_s_role
    AFTER INSERT OR UPDATE OR DELETE ON actor_s_role for EACH ROW EXECUTE PROCEDURE
add_to_log ();
```

Запрос	Демонстрация результата	Комментарий
`insert into person (id_person, surename, "name", patronymic, date_of_birth, id_city, sex) values (12332, 'asdf', 'asd', 'qwer', '05-05-1995', 1, 'fmael')`		Добавление пользователя
`delete from person where id_person = 12332`		Удаление пользователя
`UPDATE person set id_person = 589456 where id_person = 12332`		Изменение пользователя

Создание пользователей и РПД

```
create group film_studio_group with login;
grant select, update, insert on contracts to film_studio_group;
grant select, update, insert on actors to film_studio_group;
grant select, update, insert on actor_s_role to film_studio_group;


create group actors_group with login;
grant select on film_studios to actors_group;
grant select on city to actors_group;
grant select on movies to actors_group;
grant select on directors to actors_group;

create group user_group with login;
grant select on country to user_group;
grant select on movies to user_group;
grant select on directors to user_group;

create user film_studio_user with password '1234' login;
    grant film_studio_group to film_studio_user;

create user actors_user with password '1234' login;
    grant actors_group to actors_user;
```

```
create user user_user with password '1234' login;
    grant user_group to user_user;
```



Создание представлений для пользователей

```
create or replace view Contracts_and_Actors as
      select id_actor, id_movi, id_contract, number_of_contract, date_contract, id_studio
from (((actors inner join contracts using (id_actor))
inner join film_studios using (id_studio))
inner join person on (id_actor = person.id_person))


create or replace view Studio_and_City as
select id_studio, name_of_studio, city from
(city inner join film_studios using (id_city))


create or replace view Movi_and_Country as
select * from (country inner join movies using (id_country))
```

Настройка шифрования атрибута

```
create extension if not exists pgcrypto;
CREATE FUNCTION encrypt_value_supplier() RETURNS TRIGGER AS $$
    BEGIN
        NEW.sex = PGP_SYM_ENCRYPT(NEW.sex ,'AES_KEY');
        RETURN NEW;
    END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER encrypt_row_supplier
    BEFORE INSERT on person
    FOR EACH ROW
    EXECUTE PROCEDURE encrypt_value_supplier();
```

id_person	surename	name	patronymic	date_of_birth	id_city	sex
0	Ramos	Donella	Tijuana	2001-06-09	42	\xc30d0407030237b3f23118484db177d2360
1	Mann	Shella	Maryanna	1979-03-01	9	\xc30d04070302324ec851da8d13186ad2360
2	Hardy	Luther	Rusty	2001-03-14	61	\xc30d040703023c46c54ba641e45e7ad237(
3	Maxwell	Moises	Omega	1981-03-28	43	\xc30d04070302117d0baa37cf80d974d2360
4	Calderon	Elliott	Armando	1980-02-12	7	\xc30d040703027ff0509fea0108b36cd2370
5	Webb	Arron	Otto	1973-06-10	33	\xc30d0407030215c16730f7e5f88d74d2360
6	Travis	Esteban	Quinton	1979-10-26	50	\xc30d04070302667dcc62d65f14956ed237(
7	Miranda	Annamae	Annalee	1959-04-16	83	\xc30d040703021ea586823d8716f57cd2360
8	Melton	Dwana	Geoffrey	1981-11-14	92	\xc30d040703021fd29c01efc9756f67d23501

Резервное копирование данных

pg_dump postgres > /Users/leoto/Desktop/work/DB_LAB_2/post.sql

psql -U postgres < post.sql

Приложение 1.

```
import psycopg2
import config

try:

    connection = psycopg2.connect(user=config.user,
                                  password=config.password,
                                  host=config.host,
                                  port=config.port,
                                  database=config.database)
    connection.autocommit = True
    cursor = connection.cursor()
    cursor.execute(config.drop)  # удаление базы данных
    cursor.execute(config.create)    #создание базы данных
    cursor.execute(config.imp)       #импортирование данных
    cursor.execute(config.export)    #экспартирование данных

except Exception as error:
    print("Ошибка при работе с PostgreSQL", error)

finally:
    if connection:
        cursor.close()
        connection.close()
        print("Соединение с PostgreSQL закрыто")
```

```
user="postgres"
password=""
host="localhost"
port="5432"
database="postgres"

create = """CREATE TABLE Actor_s_role
(
   ID_roles             INTEGER NOT NULL ,
   name_of_role         VARCHAR(20) NOT NULL
);


ALTER TABLE Actor_s_role
   ADD CONSTRAINT  XPKActor_s_role PRIMARY KEY (ID_roles);

CREATE TABLE Actors
(
   ID_actor             INTEGER NOT NULL
);



ALTER TABLE Actors
   ADD CONSTRAINT  XPKActors PRIMARY KEY (ID_actor);

CREATE TABLE Actor_s_role_Actors
(
   ID_roles             INTEGER NOT NULL ,
   ID_actor             INTEGER NOT NULL
);


ALTER TABLE Actor_s_role_Actors
   ADD CONSTRAINT  XPKActor_s_role_Actors PRIMARY KEY (ID_roles,ID_actor);
```

```
CREATE TABLE City
(
   ID_city              INTEGER NOT NULL ,
   City                 VARCHAR(30) NOT NULL
);


ALTER TABLE City
   ADD CONSTRAINT  XPKCity PRIMARY KEY (ID_city);

CREATE TABLE Country
(
   ID_country           INTEGER NOT NULL ,
   Country              VARCHAR(30) NOT NULL
);


ALTER TABLE Country
   ADD CONSTRAINT  XPKCountry PRIMARY KEY (ID_country);

CREATE TABLE Film_studios
(
   ID_studio            INTEGER NOT NULL ,
   name_of_studio       VARCHAR(100) NOT NULL ,
   ID_city              INTEGER NOT NULL
);



ALTER TABLE Film_studios
   ADD CONSTRAINT  XPKFilm_studios PRIMARY KEY (ID_studio);

CREATE TABLE Movies
(
   ID_movi              INTEGER NOT NULL ,
   Name_of_movi         VARCHAR(30) NOT NULL ,
   date_movi            INTEGER NOT NULL ,
   ID_country           INTEGER NOT NULL ,
   ID_director          INTEGER NOT NULL ,
   budget               INTEGER NOT NULL CHECK (budget > 0)
);


ALTER TABLE Movies
   ADD CONSTRAINT  XPKMovies PRIMARY KEY (ID_movi);

CREATE TABLE Contracts
(
   ID_contract          INTEGER NOT NULL ,
   ID_studio            INTEGER NOT NULL ,
   ID_movi              INTEGER NOT NULL ,
   date_contract        DATE NOT NULL ,
   ID_actor             INTEGER NOT NULL ,
   Number_of_contract   INTEGER NOT NULL UNIQUE CHECK (Number_of_contract > 0)
);



ALTER TABLE Contracts
   ADD CONSTRAINT  XPKContracts PRIMARY KEY (ID_contract);

CREATE TABLE Directors
(
   ID_director          INTEGER NOT NULL
);
```

```
ALTER TABLE Directors
   ADD CONSTRAINT  XPKDirectors PRIMARY KEY (ID_director);

CREATE TABLE Film_genres
(
   ID_ganre             INTEGER NOT NULL ,
   ganre_name           VARCHAR(30) NOT NULL
);



ALTER TABLE Film_genres
   ADD CONSTRAINT  XPKFilm_genres PRIMARY KEY (ID_ganre);

CREATE TABLE Film_geners_Movies
(
   ID_ganre             INTEGER NOT NULL ,
   ID_movi              INTEGER NOT NULL
);



ALTER TABLE Film_geners_Movies
   ADD CONSTRAINT  XPKFilm_geners_Movies PRIMARY KEY (ID_ganre,ID_movi);

CREATE TABLE Person
(
   ID_person            INTEGER NOT NULL ,
   Surename             VARCHAR(30) NOT NULL ,
   Name                 VARCHAR(30) NOT NULL ,
   Patronymic           VARCHAR(30) NULL ,
   Date_of_Birth        DATE NOT NULL ,
   ID_city              INTEGER NOT NULL ,
   Sex                  text NOT NULL
);

CREATE TABLE logs
(
    "text" text,
    "added" timestamp without time zone,
    ID_person            INTEGER NULL ,
    Surename             VARCHAR(30) NULL ,
   Name                 VARCHAR(30) NULL ,
   Patronymic           VARCHAR(30) NULL ,
   Date_of_Birth        DATE NULL ,
   ID_city              INTEGER NULL ,
   Sex                  text NULL
);



ALTER TABLE Person
   ADD CONSTRAINT  XPKPerson PRIMARY KEY (ID_person);


ALTER TABLE Actors
   ADD CONSTRAINT R_13 FOREIGN KEY (ID_actor) REFERENCES Person (ID_person) ON DELETE
CASCADE;

ALTER TABLE Actor_s_role_Actors
   ADD CONSTRAINT R_2 FOREIGN KEY (ID_roles) REFERENCES Actor_s_role (ID_roles);

ALTER TABLE Actor_s_role_Actors
```

```
   ADD CONSTRAINT R_3 FOREIGN KEY (ID_actor) REFERENCES Actors (ID_actor);

ALTER TABLE Film_studios
   ADD CONSTRAINT R_16 FOREIGN KEY (ID_city) REFERENCES City (ID_city);

ALTER TABLE Movies
   ADD CONSTRAINT R_8 FOREIGN KEY (ID_director) REFERENCES Directors (ID_director);

ALTER TABLE Movies
   ADD CONSTRAINT R_9 FOREIGN KEY (ID_country) REFERENCES Country (ID_country);

ALTER TABLE Contracts
   ADD CONSTRAINT R_4 FOREIGN KEY (ID_studio) REFERENCES Film_studios (ID_studio);

ALTER TABLE Contracts
   ADD CONSTRAINT R_5 FOREIGN KEY (ID_actor) REFERENCES Actors (ID_actor);

ALTER TABLE Contracts
   ADD CONSTRAINT R_7 FOREIGN KEY (ID_movi) REFERENCES Movies (ID_movi);

ALTER TABLE Directors
   ADD CONSTRAINT R_14 FOREIGN KEY (ID_director) REFERENCES Person (ID_person) ON DELETE
CASCADE;

ALTER TABLE Film_geners_Movies
   ADD CONSTRAINT R_11 FOREIGN KEY (ID_ganre) REFERENCES Film_genres (ID_ganre);

ALTER TABLE Film_geners_Movies
   ADD CONSTRAINT R_12 FOREIGN KEY (ID_movi) REFERENCES Movies (ID_movi);

ALTER TABLE Person
   ADD CONSTRAINT R_17 FOREIGN KEY (ID_city) REFERENCES City (ID_city);
"""

drop = """
DROP SCHEMA public CASCADE;
CREATE SCHEMA public;
"""

imp = """
COPY city FROM '/Users/leoto/Desktop/work/DB_LAB_2/csv/city_table.txt' DELIMITER'|';
COPY person FROM '/Users/leoto/Desktop/work/DB_LAB_2/csv/person_table.txt' DELIMITER '|'
;
COPY country FROM '/Users/leoto/Desktop/work/DB_LAB_2/csv/country_table.txt' DELIMITER
'|' ;
COPY film_genres FROM '/Users/leoto/Desktop/work/DB_LAB_2/csv/film_gan_table.txt'
DELIMITER '|' ;
COPY film_studios FROM '/Users/leoto/Desktop/work/DB_LAB_2/csv/studio_table.txt'
DELIMITER '|' ;
COPY actors FROM '/Users/leoto/Desktop/work/DB_LAB_2/csv/actor.txt' DELIMITER '|' ;
COPY directors FROM '/Users/leoto/Desktop/work/DB_LAB_2/csv/directors_table.txt'
DELIMITER '|' ;
COPY actor_s_role FROM '/Users/leoto/Desktop/work/DB_LAB_2/csv/rol.txt' DELIMITER '|' ;
COPY actor_s_role_actors FROM
'/Users/leoto/Desktop/work/DB_LAB_2/csv/actor_s_role_actors_table.txt' DELIMITER '|' ;
COPY movies FROM '/Users/leoto/Desktop/work/DB_LAB_2/csv/movi_table.txt' DELIMITER '|' ;
COPY contracts FROM '/Users/leoto/Desktop/work/DB_LAB_2/csv/contract_table.txt' DELIMITER
'|' ;
COPY film_geners_movies FROM
'/Users/leoto/Desktop/work/DB_LAB_2/csv/film_geners_movies_table.txt' DELIMITER '|' ;
"""
```

```
export = """
COPY city TO '/Users/leoto/Desktop/work/DB_LAB_2/csv1/city_table.txt' DELIMITER',' ;
COPY person TO '/Users/leoto/Desktop/work/DB_LAB_2/csv1/person_table.txt' DELIMITER ',' ;
COPY country TO '/Users/leoto/Desktop/work/DB_LAB_2/csv1/country_table.txt' DELIMITER ','
;
COPY film_genres TO '/Users/leoto/Desktop/work/DB_LAB_2/csv1/film_gan_table.txt'
DELIMITER ',' ;
COPY film_studios TO '/Users/leoto/Desktop/work/DB_LAB_2/csv1/studio_table.txt' DELIMITER
',' ;
COPY actors TO '/Users/leoto/Desktop/work/DB_LAB_2/csv1/actor.txt' DELIMITER ',' ;
COPY directors TO '/Users/leoto/Desktop/work/DB_LAB_2/csv1/directors_table.txt' DELIMITER
',' ;
COPY actor_s_role TO '/Users/leoto/Desktop/work/DB_LAB_2/csv1/rol.txt' DELIMITER ',' ;
COPY actor_s_role_actors TO
'/Users/leoto/Desktop/work/DB_LAB_2/csv1/actor_s_role_actors_table.txt' DELIMITER ',' ;
COPY movies TO '/Users/leoto/Desktop/work/DB_LAB_2/csv1/movi_table.txt' DELIMITER ',' ;
COPY contracts TO '/Users/leoto/Desktop/work/DB_LAB_2/csv1/contract_table.txt' DELIMITER
',' ;
COPY film_geners_movies TO
'/Users/leoto/Desktop/work/DB_LAB_2/csv1/film_geners_movies_tablev.txt' DELIMITER ',' ;

"""
```

Приложение 2

```
import csv
import mimesis
import random


def city(numb):
    array = []
    array2 = []
    for i in range(numb):
        array.append(mimesis.Address().city())
    b = list(set(array))
    for j in range(len(b)):
        array2.append([str(j), b[j]])
    return array2


def generate_person(city, num):
    g = mimesis.Person('en')
    array = []
    for i in range(num):
        a = mimesis.Datetime().datetime(start=1950, end=2005)
        city_id = random.randrange(1, len(city) - 1)
        array.append([str(i), g.last_name(), g.name(), g.name(),  str(a.month) + '-' +
str(a.day) + '-' + str(a.year),
                      str(city_id), g.gender(iso5218=False, symbol=False)])
    return array


def country(numb):
    array = []
    array2 = []
    for i in range(numb):
        array.append(mimesis.Address().country())
    b = list(set(array))
    for j in range(len(b)):
        array2.append([str(j), b[j]])
    return array2


def film_ganer():
    ganer = ['Action', 'Comedy', 'Drama', 'Fantasy', 'Horror', 'Mystery', 'Romance',
'Thriller', 'Western']
    array = []
    for i in range(len(ganer)):
        array.append([str(i), ganer[i]])
    return array


def studio(num, city):
    g = mimesis.Person('en')
    array = []
    for i in range(num):
        city_id = random.randrange(0, len(city) - 1)
        array.append([str(i), g.name(), str(city_id)])
    return array


def actors(person):
    array = []
    for i in range(0, (len(person) // 2)):
        array.append([str(i)])
    return array
```

```
def directors(person):
    array = []
    for i in range((len(person) // 2), len(person)-1):
        array.append([str(i)])
    return array


def actor_s_role():
    role = ['Background Actor', 'Series regular', 'Recurring', 'Guest star', 'Co-star/day
player', 'Cameo']
    array = []
    for i in range(len(role)):
        array.append([str(i), role[i]])
    return array


def actor_s_role_actors(actor, role):
    array = []
    for i in range(1, len(actor)):
        role_id = random.randrange(0, len(role) - 1)
        array.append([role_id, actor[i][0]])
    return array


def movi(num, country, diretors):
    g = mimesis.Person('en')
    array = []
    for i in range(num):
        a = mimesis.Datetime().datetime(start=1950, end=2022)
        direcors_id = random.randrange(1, len(diretors) - 1)
        contry_id = random.randrange(0, len(country) - 1)
        budget = random.randrange(0, 1000000000)
        array.append(
            [str(i), g.name(), int(a.year), str(contry_id),
str(diretors[direcors_id][0]),
             str(budget)])
    return array


def contracts(actors, studio, movis, num):
    array = []
    for i in range(num):
        studio_id = random.randrange(0, len(studio) - 1)
        movi_id = random.randrange(1, len(movis) - 1)
        a = mimesis.Datetime().datetime(start=1950, end=2022)
        actors_id = random.randrange(0, len(actors) - 1)
        budget = random.randrange(0, 100000000)
        array.append(
            [str(i), str(studio_id),str(movis[movi_id][0]), str(a.month) + '-' +
str(a.day) + '-' + str(a.year), str(actors_id),
             str(budget)])
    return array


def film_geners_movies(movi, ganer):
    array = []
    for i in range(0, len(movi)-1):
        ganer_id = random.randrange(0, len(ganer) - 1)
        array.append([str(ganer_id), str(i)])
    return array


def pr_all():
    print(cit)
    print(person)
```

```
    print(country_array)
    print(film_gan)
    print(studio_table)
    print(actor)
    print(directors_table)
    print(rol)
    print(actor_s_role_actors_table)
    print(movi_table)
    print(contract_table)
    print(film_geners_movies_table)


def write_to_csv(path, array):
    with open(path, 'w') as f:
        writer = csv.writer(f, quoting=csv.QUOTE_NONNUMERIC)
        for row in array:
            writer.writerow(row)

def write_to_txt(path, array):
    f = open(path, 'w')
    for row in array:
        line = ''
        for i in range(len(row)):
            line = line + str(row[i])
            if i != len(row)-1:
                line += '|'
        f.write(line + '\n')




num_of_people = 1000
cit = city(num_of_people // 10)
write_to_txt('./csv/city_table.txt', cit)
person = generate_person(cit, num_of_people)
write_to_txt('./csv/person_table.txt', person)
country_array = country(num_of_people // 20)
write_to_txt('./csv/country_table.txt', country_array)
film_gan = film_ganer()
write_to_txt('./csv/film_gan_table.txt', film_gan)
studio_table = studio(50, cit)
write_to_txt('./csv/studio_table.txt', studio_table)
actor = actors(person)
write_to_txt('./csv/actor.txt', actor)
directors_table = directors(person)
write_to_txt('./csv/directors_table.txt', directors_table)
rol = actor_s_role()
write_to_txt('./csv/rol.txt', rol)
actor_s_role_actors_table = actor_s_role_actors(actor, rol)
write_to_txt('./csv/actor_s_role_actors_table.txt', actor_s_role_actors_table)
movi_table = movi(num_of_people // 10, country_array, directors_table)
write_to_txt('./csv/movi_table.txt', movi_table)
contract_table = contracts(actor, studio_table, movi_table, len(movi_table) * 10)
write_to_txt('./csv/contract_table.txt', contract_table)
film_geners_movies_table = film_geners_movies(movi_table, film_gan)
write_to_txt('./csv/film_geners_movies_table.txt', film_geners_movies_table)
```