МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики
Кафедра системного програмування і спеціалізованих комп’ютерних систем

Лабораторна робота № 2

з дисципліни «Бази даних і засоби управління»

«Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL»

Виконав: Піскун Андрій
Студент групи КВ-93
Перевірив: Павловський В.І.

Київ 2021

Лабораторна робота №2

Проектування бази даних та ознайомлення з базовими операціями СУБД PostgreSQL

Метою роботи є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

Загальне завдання роботи полягає у наступному:

1. Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

Логічна модель учбової предметної області «Кінотеатр»

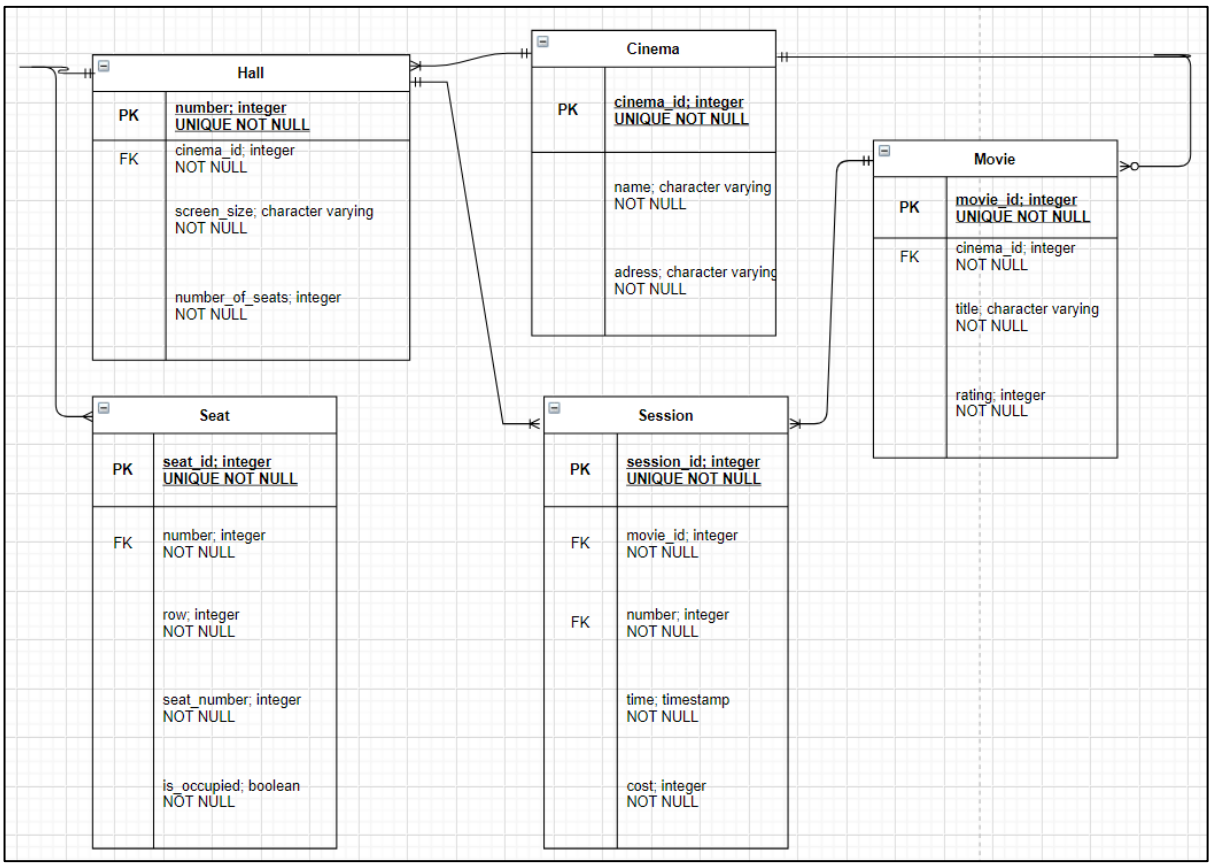

Рисунок0 – логічна модель учбової предметної області «Кінотеатр»

Середовище розробки

Середовище розробки бази даних - PostgreSQL

Середовище розробки програми – PyCharm. Мова програмування - Python 3.8.

Використані бібліотеки: psycopg2 (для зв'язку з СУБД), datetime (для роботи з датою і передачею її у запити до БД), time (для виміру часу запиту пошуку для завдання 3), sys (для реалізації консольного інтерфейсу).

Шаблон проектування

MVC - Шаблон проектування, який використаний у програмі.

Model – клас, що здійснює запит до бази даних.

View – клас, що відповідає за перевірку аргументів та вивід інформації.

Controller – клас, що відповідає за передачу аргументів у клас View на перевірку і, за умови їх коректності, передає ці аргументи у клас Model.

Структура програми та її опис

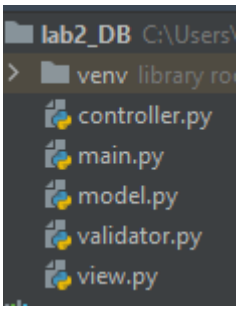

Рисунок1 – структура програми

Точкою входу в програму є main.py. В залежності від обраної команди, вона запускає відповідний метод об'єкту класу Controller, який реалізує передачу аргументів у клас View на перевірку і за умови їх коректності, Controller далі передає ці аргументи у клас Model, що здійснює запит до бази даних. Клас Validator слугує для розширення класу View.

Структура меню програми

```
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py help
print_table - outputs the specified table
        argument (table_name) is required
delete_record - deletes the specified record from table
        arguments (table_name, key_name, key_value) are required
update_record - updates record with specified id in table
        session args (table_name, session_id, movie_id, number(id of hall), time, cost)
        movie args (table_name, movie_id, cinema_id, title, rating)
        hall args (table_name, number(id), cinema_id, screen_size, number_of_seats)
        seat args (table_name, seat_id, number(id of hall), row, seat_number, is_occupied)
        cinema args (table_name, cinema_id, name, adress)
insert_record - inserts record into specified table
        session args (table_name, session_id, movie_id, number(id of hall), time, cost)
        movie args (table_name, movie_id, cinema_id, title, rating)
        hall args (table_name, number(id), cinema_id, screen_size, number_of_seats)
        seat args (table_name, seat_id, number(id of hall), row, seat_number, is_occupied)
        cinema args (table_name, cinema_id, name, adress)
generate_randomly - generates n random records in table
        arguments (table_name, n) are required
search_records - search for records in two or more tables using one or more keys
        arguments (table1_name, table2_name, table1_key, table2_key) are required,
        if you want to perform search in more tables:
        (table1_name, table2_name, table3_name, table1_key, table2_key, table3_key, table13_key)
        (table1_name, table2_name, table3_name, table4_name, table1_key, table2_key, table3_key, table13_key, table4_key, table24_key)
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок2 – Команда help

На рис.2 показано результат виконання команди help. Вона виводить на екран всі доступні команди, їх опис та необхідні аргументи для запуску.

print_table – виводить вміст таблиці, переданої аргументом. Аргументами можуть бути: cinema, movie, hall, session, seat.

delete_record – видаляє запис відповідної таблиці за первинним ключем (необхідно вказати назву ключа).

update_record – змінює всі дані запису таблиці, окрім первинного ключа, за умови наявності такого. На рис.2 показані можливі аргументи.

insert_record – вставляє новий рядок у таблицю, якщо у ній ще немає такого первинного ключа. На рис.2 показані можливі аргументи.

generate_randomly – генерує n всевдорандомізованих записів в певній таблиці, що передається аргументом.

search_records - реалізує пошук за 1 та більше атрибутами з вказаних таблиць (від двох до чотирьох) і виводить у вікно терміналу результат пошуку (або нічого, якщо пошук не дав результатів) та час, за який було проведено запит. На рис.2 показані можливі аргументи.

Фрагменти програм внесення, редагування та вилучення даних у базі даних

Фрагмент програми для внесення даних у таблицю session

```
def insert_session(self, key: str, movie_id: str, number: str, time: str, cost: str):
    if self.v.valid.check_possible_keys('session', 'session_id', key):
        count_s = self.m.find('session', 'session_id', int(key))[0]
    if self.v.valid.check_possible_keys('movie', 'movie_id', movie_id):
        count_m = self.m.find('movie', 'movie_id', int(movie_id))
        m_val = self.v.valid.check_pk(movie_id, count_m)
    if self.v.valid.check_possible_keys('hall', 'number', number):
        count_h = self.m.find('hall', 'number', int(number))
        h_val = self.v.valid.check_pk(number, count_h)

    if (not count_s or count_s == (0,)) and m_val and h_val \
            and self.v.valid.check_possible_keys('session', 'session_id', key) \
            and self.v.valid.check_possible_keys('session', 'cost', cost):
        try:
            arr = [int(x) for x in time.split(sep='.')]
            self.m.insert_data_session(int(key), m_val, h_val,
                                       datetime.datetime(arr[0], arr[1], arr[2],
                                                         arr[3], arr[4], arr[5]),
                                       float(cost))
        except (Exception, Error) as _ex:
            self.v.sql_error(_ex)
    else:
        self.v.insertion_error()
def insert_data_session(self, session_id: int, movie_id: int, number: int, time:
datetime.datetime,
                        cost: float) -> None:
    self.request(f"insert into public.\"session\" (session_id, movie_id, number,
time, cost) "
                 f"VALUES ({session_id}, {movie_id}, {number}, \'{time}\', {cost});")
```

Фрагмент програми для внесення даних у таблицю movie

```
def insert_movie(self, key: str, cinema_id: str, title: str, rating: str):
    if self.v.valid.check_possible_keys('cinema', 'cinema_id', cinema_id):
        count_c = self.m.find('cinema', 'cinema_id', int(cinema_id))
        c_val = self.v.valid.check_pk(int(cinema_id), count_c)
    if self.v.valid.check_possible_keys('movie', 'movie_id', key):
        count_m = self.m.find('movie', 'movie_id', int(key))[0]

    if (not count_m or count_m == (0,)) and c_val and
self.v.valid.check_possible_keys('movie', 'movie_id', key) \
            and self.v.valid.check_possible_keys('movie', 'rating', rating):
        try:
            self.m.insert_data_movie(int(key), c_val, title, float(rating))
        except (Exception, Error) as _ex:
            self.v.sql_error(_ex)
    else:
        self.v.insertion_error()
def insert_data_movie(self, movie_id: int, cinema_id: int, title: str, rating: float)
-> None:
    self.request(f"insert into public.\"movie\" (movie_id, cinema_id, title, rating)
"
                 f"VALUES ({movie_id}, {cinema_id}, \'{title}\', {rating});")
```

Фрагмент програми для внесення даних у таблицю hall

```
def insert_hall(self, key: str, cinema_id: str, screen_size: str, number_of_seats:
str):
```

```
    if self.v.valid.check_possible_keys('cinema', 'cinema_id', cinema_id):
        count_c = self.m.find('cinema', 'cinema_id', int(cinema_id))
        c_val = self.v.valid.check_pk(cinema_id, count_c)
    if self.v.valid.check_possible_keys('hall', 'number', key):
        count_h = self.m.find('hall', 'number', int(key))[0]

    if (not count_h or count_h == (0,)) and c_val \
            and self.v.valid.check_possible_keys('hall', 'number', key):
        try:
            self.m.insert_data_hall(int(key), c_val, screen_size,
int(number_of_seats))
        except (Exception, Error) as _ex:
            self.v.sql_error(_ex)
    else:
        self.v.insertion_error()
def insert_data_hall(self, number: int, cinema_id: int, screen_size: str,
number_of_seats: int) -> None:
    self.request(f"insert into public.\"hall\" (number, cinema_id, screen_size,
number_of_seats) "
                 f"VALUES ({number}, {cinema_id}, \'{screen_size}\',
{number_of_seats});")
```

Фрагмент програми для внесення даних у таблицю cinema

```
def insert_cinema(self, key: str, name: str, adress: str):
    if self.v.valid.check_possible_keys('cinema', 'cinema_id', key):
        count_c = self.m.find('cinema', 'cinema_id', int(key))[0]

    if (not count_c or count_c == (0,)) and
self.v.valid.check_possible_keys('cinema', 'cinema_id', key):
        try:
            self.m.insert_data_cinema(int(key), name, adress)
        except (Exception, Error) as _ex:
            self.v.sql_error(_ex)
    else:
        self.v.insertion_error()
def insert_data_cinema(self, cinema_id: int, name: str, adress: str) -> None:
    self.request(f"insert into public.\"cinema\" (cinema_id, name, adress) "
                 f"VALUES ({cinema_id}, \'{name}\', \'{adress}\');")
```

Фрагмент програми для внесення даних у таблицю seat

```
def insert_seat(self, key: str, number: str, row: str, seat_number: str, is_occupied:
str):
    if self.v.valid.check_possible_keys('hall', 'number', number):
        count_h = self.m.find('hall', 'number', int(number))
        h_val = self.v.valid.check_pk(number, count_h)
    if self.v.valid.check_possible_keys('seat', 'seat_id', key):
        count_s = self.m.find('seat', 'seat_id', int(key))[0]

    if (not count_s or count_s == (0,)) and h_val \
            and self.v.valid.check_possible_keys('seat', 'seat_id', key):
        try:
            self.m.insert_data_seat(int(key), h_val, int(row), int(seat_number),
int(is_occupied))
        except (Exception, Error) as _ex:
            self.v.sql_error(_ex)
    else:
        self.v.insertion_error()
def insert_data_seat(self, seat_id: int, number: int, row: int, seat_number: int,
is_occupied: int) -> None:
    if is_occupied == 1:
```

```
        is_occupied1 = 'true'
    else:
        is_occupied1 = 'false'
    self.request(f"insert into public.\"seat\" (seat_id, number, row, seat_number,
is_occupied) "
                 f"VALUES ({seat_id}, {number}, {row}, {seat_number},
\'{is_occupied1}\');")
```

Фрагмент програми для редагування даних у таблиці session

```
def update_session(self, key: str, movie_id: str, number: str, time: str, cost: str):
    if self.v.valid.check_possible_keys('session', 'session_id', key):
        count_s = self.m.find('session', 'session_id', int(key))
        s_val = self.v.valid.check_pk(key, count_s)
    if self.v.valid.check_possible_keys('movie', 'movie_id', movie_id):
        count_m = self.m.find('movie', 'movie_id', int(movie_id))
        m_val = self.v.valid.check_pk(movie_id, count_m)
    if self.v.valid.check_possible_keys('hall', 'number', number):
        count_h = self.m.find('hall', 'number', int(number))
        h_val = self.v.valid.check_pk(number, count_h)

    if m_val and h_val and s_val and self.v.valid.check_possible_keys('session',
'cost', cost):
        try:
            arr = [int(x) for x in time.split(sep='.')]
            self.m.update_data_session(s_val, m_val, h_val,
                                       datetime.datetime(arr[0], arr[1], arr[2],
arr[3], arr[4], arr[5]),
                                       float(cost))
        except (Exception, Error) as _ex:
            self.v.sql_error(_ex)
    else:
        self.v.updation_error()
def update_data_session(self, key_value: int, movie_id: int, number: int, time:
datetime.datetime,
                        cost: float) -> None:
    self.request(f"UPDATE public.\"session\" SET movie_id={movie_id},
number={number}, time=\'{time}\', "
                 f"cost={cost} WHERE session_id={key_value};")
```

Фрагмент програми для редагування даних у таблиці movie

```
def update_movie(self, key: str, cinema_id: str, title: str, rating: str):
    if self.v.valid.check_possible_keys('cinema', 'cinema_id', cinema_id):
        count_c = self.m.find('cinema', 'cinema_id', int(cinema_id))
        c_val = self.v.valid.check_pk(cinema_id, count_c)
    if self.v.valid.check_possible_keys('movie', 'movie_id', key):
        count_m = self.m.find('movie', 'movie_id', int(key))
        m_val = self.v.valid.check_pk(key, count_m)

    if c_val and m_val and self.v.valid.check_possible_keys('movie', 'rating',
rating):
        try:
            self.m.update_data_movie(m_val, c_val, title, float(rating))
        except (Exception, Error) as _ex:
            self.v.sql_error(_ex)
    else:
        self.v.updation_error()
def update_data_movie(self, key_value: int, cinema_id: int, title: str, rating:
float) -> None:
    self.request(f"UPDATE public.\"movie\" SET cinema_id={cinema_id},
```

```
title=\'{title}\', "
                f"rating={rating} WHERE movie_id={key_value};")
```

Фрагмент програми для редагування даних у таблиці hall

```
def update_hall(self, key: str, cinema_id: str, screen_size: str, number_of_seats:
str):
    if self.v.valid.check_possible_keys('cinema', 'cinema_id', cinema_id):
        count_c = self.m.find('cinema', 'cinema_id', int(cinema_id))
        c_val = self.v.valid.check_pk(cinema_id, count_c)
    if self.v.valid.check_possible_keys('hall', 'number', key):
        count_h = self.m.find('hall', 'number', int(key))
        h_val = self.v.valid.check_pk(key, count_h)

    if c_val and h_val:
        try:
            self.m.update_data_hall(h_val, c_val, screen_size, int(number_of_seats))
        except (Exception, Error) as _ex:
            self.v.sql_error(_ex)
    else:
        self.v.updation_error()
def update_data_hall(self, key_value: int, cinema_id: int, screen_size: str,
number_of_seats: int) -> None:
    self.request(f"UPDATE public.\"hall\" SET cinema_id={cinema_id},
screen_size=\'{screen_size}\', "
                f"number_of_seats={number_of_seats} WHERE number={key_value};")
```

Фрагмент програми для редагування даних у таблиці cinema

```
def update_cinema(self, key: str, name: str, adress: str):
    if self.v.valid.check_possible_keys('cinema', 'cinema_id', key):
        count_c = self.m.find('cinema', 'cinema_id', int(key))
        c_val = self.v.valid.check_pk(key, count_c)

    if c_val:
        try:
            self.m.update_data_cinema(c_val, name, adress)
        except (Exception, Error) as _ex:
            self.v.sql_error(_ex)
    else:
        self.v.updation_error()
def update_data_cinema(self, key_value: int, name: str, adress: str) -> None:
    self.request(f"UPDATE public.\"cinema\" SET name=\'{name}\', "
                f"adress=\'{adress}\' WHERE cinema_id={key_value};")
```

Фрагмент програми для редагування даних у таблиці seat

```
def update_seat(self, key: str, number: str, row: str, seat_number: str, is_occupied:
str):
    if self.v.valid.check_possible_keys('hall', 'number', number):
        count_h = self.m.find('hall', 'number', int(number))
        h_val = self.v.valid.check_pk(number, count_h)
    if self.v.valid.check_possible_keys('seat', 'seat_id', key):
        count_s = self.m.find('seat', 'seat_id', int(key))
        s_val = self.v.valid.check_pk(key, count_s)

    if s_val and h_val :
        try:
            self.m.update_data_seat(s_val, h_val, int(row), int(seat_number),
int(is_occupied))
        except (Exception, Error) as _ex:
            self.v.sql_error(_ex)
    else:
        self.v.insertion_error()
```

```
def update_data_seat(self, key_value: int, number: int, row: int, seat_number: int,
is_occupied: int) -> None:
    if(is_occupied == 1):
        is_occupied1 = 'true'
    else:
        is_occupied1 = 'false'
    self.request(f"UPDATE public.\"seat\" SET number={number}, row={row},
seat_number={seat_number}, "
                 f"is_occupied={is_occupied1} WHERE seat_id={key_value};")
```

Фрагмент програми для видалення даних у таблиці

```
def delete(self, table_name, key_name, value):
    t_name = self.v.valid.check_table_name(table_name)
    k_name = self.v.valid.check_pk_name(table_name, key_name)
    if t_name and k_name:
        count = self.m.find(t_name, k_name, value)
        k_val = self.v.valid.check_pk(value, count)
        if k_val:
            if t_name == 'movie':
                count_s = self.m.find('session', k_name, value)[0]
                if count_s:
                    self.v.cannot_delete()
                else:
                    try:
                        self.m.delete_data(table_name, key_name, k_val)
                    except (Exception, Error) as _ex:
                        self.v.sql_error(_ex)
            elif t_name == 'cinema':
                count_h = self.m.find('hall', k_name, value)[0]
                count_m = self.m.find('movie', k_name, value)[0]
                if count_h or count_m:
                    self.v.cannot_delete()
                else:
                    try:
                        self.m.delete_data(table_name, key_name, k_val)
                    except (Exception, Error) as _ex:
                        self.v.sql_error(_ex)
            elif t_name == 'hall':
                count_seat = self.m.find('seat', k_name, value)[0]
                count_session = self.m.find('session', k_name, value)[0]
                if count_seat or count_session:
                    self.v.cannot_delete()
                else:
                    try:
                        self.m.delete_data(table_name, key_name, k_val)
                    except (Exception, Error) as _ex:
                        self.v.sql_error(_ex)
            else:
                try:
                    self.m.delete_data(table_name, key_name, k_val)
                except (Exception, Error) as _ex:
                    self.v.sql_error(_ex)
        else:
            self.v.deletion_error()
def delete_data(self, table_name: str, key_name: str, key_value) -> None:
    self.request(f"DELETE FROM public.\"{table_name}\" WHERE
{key_name}={key_value};")
```

Фрагменти програми, які наведені вижче, відповідають за функціонал додавання, редагування та вилучення даних у базі даних.

Опишу роботу методів додавання на прикладі таблиці session. Дані прибувають в метод insert_session класу Controller. Оскільки таблиця session є дочірньою таблицею таблиць movie та hall, у неї є два зовнішніх ключа, відповідно movie_id та number. Спочатку перевіряється наявність такого ж первинного ключа в таблиці, щоб запобігти ситуації, коли в таблиці вже є запис з таким первинним ключем. Потім перевіряється наявність зовнішніх ключів у батьківських таблицях. На основі цих умов метод передає всі необхідні дані далі в метод класу Model insert_data_session або видає повідомлення про помилку, відповідно до типу помилки. insert_data_session здійснює запит до бази даних. Аналогічно працюють всі методи додавання запису: якщо у таблиці є батьківська таблиця, то перевіряється наявність в ній зовнішнього ключа, що ми намагаємось додати, іде перевірка на відповідність типів і передача в наступну функцію, що безпосередньо здійснює запит.

Методи оновлення – якщо аргументи введено правильно, запис з таким первичним ключем існує, то змінюємо дані на введені (окрім значення первичного ключа).

Видалення працює таким чином: якщо запис існує, інші таблиці не залежать від цього запису (немає зовнішніх ключів з інших таблиць до запису) та аргументи введено правильно – видаляємо запис з таблиці.

Результати фрагментів роботи програми наведені в розділі «Результати роботи програми», які знаходяться нижче.

Лістинги фрагментів програм з запитами пошуку

Фрагмент програми для пошуку з двох таблиць

```
def search_two(self, table1_name: str, table2_name: str, table1_key: str, table2_key:
str, search: str):
    t1_n = self.v.valid.check_table_name(table1_name)
    t2_n = self.v.valid.check_table_name(table2_name)
    if t1_n and self.v.valid.check_key_names(t1_n, table1_key) and t2_n \
            and self.v.valid.check_key_names(t2_n, table2_key):
        start_time = time.time()
        result = self.m.search_data_two_tables(table1_name, table2_name, table1_key,
table2_key,
                                               search)
        self.v.print_time(start_time)

        self.v.print_search(result)

def search_data_two_tables(self, table1_name: str, table2_name: str, table1_key,
table2_key,
                           search: str):
    return self.get(f"select * from public.\"{table1_name}\" as one inner join
public.\"{table2_name}\" as two "
                    f"on one.\"{table1_key}\"=two.\"{table2_key}\" "
                    f"where {search}")
```

Фрагмент програми для пошуку з трьох таблиць

```
def search_three(self, table1_name: str, table2_name: str, table3_name: str,
                 table1_key: str, table2_key: str, table3_key: str, table13_key: str,
```

```
                search: str):
    t1_n = self.v.valid.check_table_name(table1_name)
    t2_n = self.v.valid.check_table_name(table2_name)
    t3_n = self.v.valid.check_table_name(table3_name)
    if t1_n and self.v.valid.check_key_names(t1_n, table1_key) and
self.v.valid.check_key_names(t1_n, table13_key) \
            and t2_n and self.v.valid.check_key_names(t2_n, table2_key) \
            and t3_n and self.v.valid.check_key_names(t3_n, table3_key) \
            and self.v.valid.check_key_names(t3_n, table13_key):
        start_time = time.time()
        result = self.m.search_data_three_tables(table1_name, table2_name,
table3_name,
                                                 table1_key, table2_key, table3_key,
table13_key,
                                                 search)
        self.v.print_time(start_time)
        self.v.print_search(result)

def search_data_three_tables(self, table1_name: str, table2_name: str, table3_name:
str,
                             table1_key, table2_key, table3_key, table13_key,
                             search: str):
    return self.get(f"select * from public.\"{table1_name}\" as one inner join
public.\"{table2_name}\" as two "
                    f"on one.\"{table1_key}\"=two.\"{table2_key}\" inner join
public.\"{table3_name}\" as three "
                    f"on three.\"{table3_key}\"=one.\"{table13_key}\""
                    f"where {search}")
```

Пошук працює так: за умови введення потрібної кількості аргументів та правильного задання умов пошуку, реалізує пошук за 1 та більше атрибутами з вказаних таблиць (від двох до трьох) і виводить у вікно терміналу результат пошуку (або нічого, якщо пошук не дав результатів) та час, за який було проведено запит. Початково потрібно вказати агрументи: table1_name table2_name table1_key table2_key або table1_name table2_name table3_name table1_key table2_key table3_key table13_key – це зовнішні ключі, що зв’язують 1 та 3 таблицю. Після вказання цієї інформації потрібно буде вказати кількість атрибутів для пошуку, а тип пошуку, ім’я атрибуту (обов’язково з вказанням до якої таблиці з перелічених аргументів він відноситься: one.key_name, two.key_name, three.key_name), та значення (спочатку лівий кінець інтервалу, потім правий для числового пошуку та пошуку за датою, або рядок для пошуку за ключовим словом). Спочатку вказуються всі дані для першого атрибуту, потім для другого і т.д. до введеної кількості атрибутів.

*Важливо: дата з часовою міткою у програмі вказується через крапку у такому форматі: year.month.day.hour.minute.second

Лістинги фрагментів програм генерування випадкових даних в таблицях БД

Лістинг програми для генерування випадкових даних для таблиці session

```
    def session_data_n_rand(self, times: int) -> None:
for i in range(times):
    self.request("insert into public.\"session\""
```

```
                    "select (SELECT MAX(session_id)+1 FROM public.\"session\"), "
                    "(SELECT movie_id FROM public.\"movie\" LIMIT 1 OFFSET "
                    "(round(random() * ((SELECT COUNT(movie_id) FROM
public.\"movie\")-1)))), "
                    "(SELECT number FROM public.\"hall\" LIMIT 1 OFFSET
(round(random() * "
                    "((SELECT COUNT(number) FROM public.\"hall\")-1)))), "
                    "(select timestamp '2018-01-10 10:00:00' + random() * "
                    "(timestamp '2021-01-20 20:00:00' - timestamp '2018-01-10
10:00:00')), "
                    "FLOOR(RANDOM()*(1000-20)+20);")
```

Лістинг програми для генерування випадкових даних для таблиці movie

```
def movie_data_n_rand(self, times: int) -> None:
    for i in range(times):
        self.request("insert into public.\"movie\" select (SELECT (MAX(movie_id)+1)
FROM public.\"movie\"), "
                    "(SELECT cinema_id FROM public.\"cinema\" LIMIT 1 OFFSET "
                    "(round(random() *((SELECT COUNT(cinema_id) FROM
public.\"cinema\")-1)))), "
                    "array_to_string(ARRAY(SELECT chr((97 + round(random() * 25)) ::
integer) "
                    "FROM generate_series(1, FLOOR(RANDOM()*(10-3)+3):: integer)),
''), "
                    "FLOOR(RANDOM()*(11-1)+1);")
```

Лістинг програми для генерування випадкових даних для таблиці hall

```
def hall_data_n_rand(self, times: int) -> None:
    for i in range(times):
        self.request("insert into public.\"hall\" select (SELECT MAX(number)+1 FROM
public.\"hall\"), "
                    "(SELECT cinema_id FROM public.\"cinema\" LIMIT 1 OFFSET "
                    "(round(random() *((SELECT COUNT(cinema_id) FROM
public.\"cinema\")-1)))), "
                    "FLOOR(RANDOM()*(300-80)+80), "
                    "FLOOR(RANDOM()*(350-20)+20);")
```

Лістинг програми для генерування випадкових даних для таблиці cinema

```
def cinema_data_n_rand(self, times: int) -> None:
    for i in range(times):
        self.request("insert into public.\"cinema\" select (SELECT MAX(cinema_id)+1
FROM public.\"cinema\"), "
                    "array_to_string(ARRAY(SELECT chr((97 + round(random() * 25)) ::
integer) "
                    "FROM generate_series(1, FLOOR(RANDOM()*(25-10)+10):: integer)),
''), "
                    "array_to_string(ARRAY(SELECT chr((97 + round(random() * 25)) ::
integer) "
                    "FROM generate_series(1, FLOOR(RANDOM()*(10-4)+4):: integer)),
''); ")
```

Лістинг програми для генерування випадкових даних для таблиці seat

```
def seat_data_n_rand(self, times: int) -> None:
    for i in range(times):
        self.request("insert into public.\"seat\" select (SELECT MAX(seat_id)+1 FROM
public.\"seat\"), "
                    "(SELECT number FROM public.\"hall\" LIMIT 1 OFFSET "
                    "(round(random() *((SELECT COUNT(number) FROM public.\"hall\")-
1)))), "
                    "FLOOR(RANDOM()*(51-1)+1), "
```

```
                "FLOOR(RANDOM()*(350-1)+1), "
                "(round(random())::int)::boolean;")
```

Методи генерування випадкових даних для таблиць реалізують запит до бази даних на вставку всевдорандомізованих даних. В якості первинного ключа використовується максимальне значення вже існуючих записів + 1. В якості зовнішніх ключів використовується випадкове значення із уже існуючих зовнішніх ключів. В якості символьних полів використовюються випадкові символи. В якості чисельних полів використовується випадкове числове значення в певних діапазонах.

Лістинг модуля "Model"

```
import datetime
import psycopg2 as ps


class Model:
    def __init__(self):
        self.conn = None
        try:
            self.conn = ps.connect(
                database="lab1",
                user='postgres',
                password="postgres",
                host='localhost',
                port="5432",
            )
        except(Exception, ps.DatabaseError) as error:
            print("Error while working with Postgresql", error)

    def request(self, req: str):
        try:
            cursor = self.conn.cursor()
            print(req)
            cursor.execute(req)
            self.conn.commit()
            return True
        except(Exception, ps.DatabaseError, ps.ProgrammingError) as error:
            print(error)
            self.conn.rollback()
            return False

    def get(self, req: str):
        try:
            cursor = self.conn.cursor()
            print(req)
            cursor.execute(req)
            self.conn.commit()
            return cursor.fetchall()
        except(Exception, ps.DatabaseError, ps.ProgrammingError) as error:
            print(error)
            self.conn.rollback()
            return False
```

```
    def get_el(self, req: str):
        try:
            cursor = self.conn.cursor()
            print(req)
            cursor.execute(req)
            self.conn.commit()
            return cursor.fetchone()
        except(Exception, ps.DatabaseError, ps.ProgrammingError) as error:
            print(error)
            self.conn.rollback()
            return False

    def count(self, table_name: str):
        return self.get_el(f"select count(*) from public.\"{table_name}\"")

    def find(self, table_name: str, key_name: str, key_value: int):
        return self.get_el(f"select count(*) from public.\"{table_name}\" where
{key_name}={key_value}")

    def max(self, table_name: str, key_name: str):
        return self.get_el(f"select max({key_name}) from public.\"{table_name}\"")

    def min(self, table_name: str, key_name: str):
        return self.get_el(f"select min({key_name}) from public.\"{table_name}\"")

    def print_cinema(self) -> None:
        return self.get(f"SELECT * FROM public.\"cinema\"")

    def print_hall(self) -> None:
        return self.get(f"SELECT * FROM public.\"hall\"")

    def print_movie(self) -> None:
        return self.get(f"SELECT * FROM public.\"movie\"")

    def print_seat(self) -> None:
        return self.get(f"SELECT * FROM public.\"seat\"")

    def print_session(self) -> None:
        return self.get(f"SELECT * FROM public.\"session\"")

    def delete_data(self, table_name: str, key_name: str, key_value) -> None:
        self.request(f"DELETE FROM public.\"{table_name}\" WHERE
{key_name}={key_value};")

    def update_data_session(self, key_value: int, movie_id: int, number: int, time:
datetime.datetime,
                            cost: float) -> None:
        self.request(f"UPDATE public.\"session\" SET movie_id={movie_id},
number={number}, time=\'{time}\', "
                     f"cost={cost} WHERE session_id={key_value};")

    def update_data_movie(self, key_value: int, cinema_id: int, title: str, rating:
float) -> None:
        self.request(f"UPDATE public.\"movie\" SET cinema_id={cinema_id},
title=\'{title}\', "
                     f"rating={rating} WHERE movie_id={key_value};")

    def update_data_hall(self, key_value: int, cinema_id: int, screen_size: str,
number_of_seats: int) -> None:
        self.request(f"UPDATE public.\"hall\" SET cinema_id={cinema_id},
screen_size=\'{screen_size}\', "
```

```
                    f"number_of_seats={number_of_seats} WHERE number={key_value};")

    def update_data_seat(self, key_value: int, number: int, row: int, seat_number:
int, is_occupied: int) -> None:
        if(is_occupied == 1):
            is_occupied1 = 'true'
        else:
            is_occupied1 = 'false'
        self.request(f"UPDATE public.\"seat\" SET number={number}, row={row},
seat_number={seat_number}, "
                    f"is_occupied={is_occupied1} WHERE seat_id={key_value};")

    def update_data_cinema(self, key_value: int, name: str, adress: str) -> None:
        self.request(f"UPDATE public.\"cinema\" SET name=\'{name}\', "
                    f"adress=\'{adress}\' WHERE cinema_id={key_value};")

    def insert_data_session(self, session_id: int, movie_id: int, number: int, time:
datetime.datetime,
                            cost: float) -> None:
        self.request(f"insert into public.\"session\" (session_id, movie_id, number,
time, cost) "
                    f"VALUES ({session_id}, {movie_id}, {number}, \'{time}\',
{cost});")

    def insert_data_movie(self, movie_id: int, cinema_id: int, title: str, rating:
float) -> None:
        self.request(f"insert into public.\"movie\" (movie_id, cinema_id, title,
rating) "
                    f"VALUES ({movie_id}, {cinema_id}, \'{title}\', {rating});")

    def insert_data_hall(self, number: int, cinema_id: int, screen_size: str,
number_of_seats: int) -> None:
        self.request(f"insert into public.\"hall\" (number, cinema_id, screen_size,
number_of_seats) "
                    f"VALUES ({number}, {cinema_id}, \'{screen_size}\',
{number_of_seats});")

    def insert_data_seat(self, seat_id: int, number: int, row: int, seat_number: int,
is_occupied: int) -> None:
        if is_occupied == 1:
            is_occupied1 = 'true'
        else:
            is_occupied1 = 'false'
        self.request(f"insert into public.\"seat\" (seat_id, number, row,
seat_number, is_occupied) "
                    f"VALUES ({seat_id}, {number}, {row}, {seat_number},
\'{is_occupied1}\');")

    def insert_data_cinema(self, cinema_id: int, name: str, adress: str) -> None:
        self.request(f"insert into public.\"cinema\" (cinema_id, name, adress) "
                    f"VALUES ({cinema_id}, \'{name}\', \'{adress}\');")

    def session_data_n_rand(self, times: int) -> None:
        for i in range(times):
            self.request("insert into public.\"session\""
                         "select (SELECT MAX(session_id)+1 FROM public.\"session\"),
"
                         "(SELECT movie_id FROM public.\"movie\" LIMIT 1 OFFSET "
                         "(round(random() * ((SELECT COUNT(movie_id) FROM
public.\"movie\")-1)))), "
                         "(SELECT number FROM public.\"hall\" LIMIT 1 OFFSET
(round(random() * "
```

```
                          "((SELECT COUNT(number) FROM public.\"hall\")-1)))), "
                          "(select timestamp '2018-01-10 10:00:00' + random() * "
                          "(timestamp '2021-01-20 20:00:00' - timestamp '2018-01-10
10:00:00')), "
                          "FLOOR(RANDOM()*(1000-20)+20);")

    def movie_data_n_rand(self, times: int) -> None:
        for i in range(times):
            self.request("insert into public.\"movie\" select (SELECT
(MAX(movie_id)+1) FROM public.\"movie\"), "
                          "(SELECT cinema_id FROM public.\"cinema\" LIMIT 1 OFFSET "
                          "(round(random() *((SELECT COUNT(cinema_id) FROM
public.\"cinema\")-1)))), "
                          "array_to_string(ARRAY(SELECT chr((97 + round(random() *
25)) :: integer) "
                          "FROM generate_series(1, FLOOR(RANDOM()*(10-3)+3)::
integer)), ''), "
                          "FLOOR(RANDOM()*(11-1)+1);")

    def hall_data_n_rand(self, times: int) -> None:
        for i in range(times):
            self.request("insert into public.\"hall\" select (SELECT MAX(number)+1
FROM public.\"hall\"), "
                          "(SELECT cinema_id FROM public.\"cinema\" LIMIT 1 OFFSET "
                          "(round(random() *((SELECT COUNT(cinema_id) FROM
public.\"cinema\")-1)))), "
                          "FLOOR(RANDOM()*(300-80)+80), "
                          "FLOOR(RANDOM()*(350-20)+20);")

    def cinema_data_n_rand(self, times: int) -> None:
        for i in range(times):
            self.request("insert into public.\"cinema\" select (SELECT
MAX(cinema_id)+1 FROM public.\"cinema\"), "
                          "array_to_string(ARRAY(SELECT chr((97 + round(random() *
25)) :: integer) "
                          "FROM generate_series(1, FLOOR(RANDOM()*(25-10)+10)::
integer)), ''), "
                          "array_to_string(ARRAY(SELECT chr((97 + round(random() *
25)) :: integer) "
                          "FROM generate_series(1, FLOOR(RANDOM()*(10-4)+4)::
integer)), ''); ")

    def seat_data_n_rand(self, times: int) -> None:
        for i in range(times):
            self.request("insert into public.\"seat\" select (SELECT MAX(seat_id)+1
FROM public.\"seat\"), "
                          "(SELECT number FROM public.\"hall\" LIMIT 1 OFFSET "
                          "(round(random() *((SELECT COUNT(number) FROM
public.\"hall\")-1)))), "
                          "FLOOR(RANDOM()*(51-1)+1), "
                          "FLOOR(RANDOM()*(350-1)+1), "
                          "(round(random())::int)::boolean;")

    def search_data_two_tables(self, table1_name: str, table2_name: str, table1_key,
table2_key,
                               search: str):
        return self.get(f"select * from public.\"{table1_name}\" as one inner join
public.\"{table2_name}\" as two "
                        f"on one.\"{table1_key}\"=two.\"{table2_key}\" "
                        f"where {search}")

    def search_data_three_tables(self, table1_name: str, table2_name: str,
```

```
table3_name: str,
                                table1_key, table2_key, table3_key, table13_key,
                                search: str):
        return self.get(f"select * from public.\"{table1_name}\" as one inner join
public.\"{table2_name}\" as two "
                        f"on one.\"{table1_key}\"=two.\"{table2_key}\" inner join
public.\"{table3_name}\" as three "
                        f"on three.\"{table3_key}\"=one.\"{table13_key}\""
                        f"where {search}")
```

Конструктор класу з’єднується із сервером і, якщо з’єднання не встановлюється, видає повідомлення про помилку

Метод request здійснює запит, за допомогою cursor, до бази даних і повертає True, якщо запит вдалось зробити. Відповідно – False, якщо не вдалось.

Метод get здійснює запит, за допомогою cursor, до бази даних і повертає дані, що було взято з запитів SELECT, якщо запит вдалось зробити. Повертає False, якщо не вдалось.

Метод get_el здійснює запит, за допомогою cursor, до бази даних і повертає перший запис, якщо запит вдалось зробити. Повертає False, якщо не вдалось.

Метод count повертає кількість усіх записів в таблиці.

Метод find повертає кількість записів таблиці, що відповідають певній умові. Повертає False, якщо не знайшов записів.

Методи max, min повертають відповідно максимальне та мінімальне значення ключа у таблиці.

Методи print_(table_name) здійснюють запит до БД та виводять на екран відповідні таблиці.

Метод delete_data здійснює запит до БД та видаляє відповідний запис в БД.

Методи update_data_(table_name) здійснюють запит до БД на оновлення певного запису відповідної таблиці.

Методи insert_data_(table_name) здійснюють запит до БД на вставлення певного запису в відповідну таблицю.

Методи (table_name)_data_n_rand здійснюють запит до БД на вставлення n-кількості псевдорандомізованих записів в певну тіблицю.

Методи search_data_(number)_tables здійснюють запит на отримання результату пошуку серед number таблиць за рядком, що генерується в контроллері методами: numeric_search, string_search, date_search.

Методи numeric_search, string_search, date_search приймають відповідні параметри пошуку та повертають рядок пошуку за відповідним типом атрибуту.

Результати роботи програми

```
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py print_table session
SELECT * FROM public."session"
session table:
session_id: 2   movie_id: 1     number: 1       time: 2021-12-11 00:00:00       cost: 200
-----------------------------------------------------
session_id: 1   movie_id: 5     number: 2       time: 2020-01-01 12:30:00       cost: 10
-----------------------------------------------------
session_id: 3   movie_id: 5     number: 2       time: 2022-03-28 00:00:00       cost: 199
-----------------------------------------------------
session_id: 4   movie_id: 5     number: 1       time: 2012-09-21 00:00:00       cost: 10
-----------------------------------------------------
session_id: 5   movie_id: 5     number: 1       time: 2021-11-11 00:00:02       cost: 120
-----------------------------------------------------
session_id: 9   movie_id: 1     number: 1       time: 2020-11-02 12:22:00       cost: 109
-----------------------------------------------------
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок3 – Таблиця session до видалення запису 5

```
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py delete_record session session_id 5
select count(*) from public."session" where session_id=5
DELETE FROM public."session" WHERE session_id=5;
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py print_table session
SELECT * FROM public."session"
session table:
session_id: 2   movie_id: 1     number: 1       time: 2021-12-11 00:00:00       cost: 200
-----------------------------------------------------
session_id: 1   movie_id: 5     number: 2       time: 2020-01-01 12:30:00       cost: 10
-----------------------------------------------------
session_id: 3   movie_id: 5     number: 2       time: 2022-03-28 00:00:00       cost: 199
-----------------------------------------------------
session_id: 4   movie_id: 5     number: 1       time: 2012-09-21 00:00:00       cost: 10
-----------------------------------------------------
session_id: 9   movie_id: 1     number: 1       time: 2020-11-02 12:22:00       cost: 109
-----------------------------------------------------
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок4 - Таблиця session після видалення запису 5

Якщо спробувати видалити запис з батьківської таблиці, який пов’язаний із дочірньою таблицею програма видасть помилку.

```
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py print_table movie
SELECT * FROM public."movie"
movie table:
movie_id: 1     cinema_id: 0    title: Harry Potter     rating: 9
----------------------------------------------------------
movie_id: 2     cinema_id: 0    title: Operation `i`    rating: 10
----------------------------------------------------------
movie_id: 3     cinema_id: 1    title: Ivan Vasylievich changes profession      rating: 8
----------------------------------------------------------
movie_id: 4     cinema_id: 2    title: Love and pigeons         rating: 5
----------------------------------------------------------
movie_id: 5     cinema_id: 1    title: FreefunIvan      rating: 2
----------------------------------------------------------
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок5 – Таблиця movie

```
----------------------------------------------------------
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py delete_record movie movie_id 1
select count(*) from public."movie" where movie_id=1
select count(*) from public."session" where movie_id=1
this record is connected with another table, deleting will throw error
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок6 – Спроба видалити запис таблиці movie, який пов’язаний із записом дочірньої таблиці session

```
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py print_table session
SELECT * FROM public."session"
session table:
session_id: 2   movie_id: 1     number: 1       time: 2021-12-11 00:00:00       cost: 200
----------------------------------------------------------
session_id: 1   movie_id: 5     number: 2       time: 2020-01-01 12:30:00       cost: 10
----------------------------------------------------------
session_id: 3   movie_id: 5     number: 2       time: 2022-03-28 00:00:00       cost: 199
----------------------------------------------------------
session_id: 4   movie_id: 5     number: 1       time: 2012-09-21 00:00:00       cost: 10
----------------------------------------------------------
session_id: 9   movie_id: 1     number: 1       time: 2020-11-02 12:22:00       cost: 109
----------------------------------------------------------
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок7 – Таблиця session до вставлення запису 10

```
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py insert_record session 10 2 2 2021.01.02.09.09.00 80
select count(*) from public."session" where session_id=10
select count(*) from public."movie" where movie_id=2
select count(*) from public."hall" where number=2
insert into public."session" (session_id, movie_id, number, time, cost) VALUES (10, 2, 2, '2021-01-02 09:09:00', 80.0);
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py print_table session
SELECT * FROM public."session"
session table:
session_id: 2   movie_id: 1     number: 1       time: 2021-12-11 00:00:00       cost: 200
-----------------------------------------------------
session_id: 1   movie_id: 5     number: 2       time: 2020-01-01 12:30:00       cost: 10
-----------------------------------------------------
session_id: 3   movie_id: 5     number: 2       time: 2022-03-28 00:00:00       cost: 199
-----------------------------------------------------
session_id: 4   movie_id: 5     number: 1       time: 2012-09-21 00:00:00       cost: 10
-----------------------------------------------------
session_id: 9   movie_id: 1     number: 1       time: 2020-11-02 12:22:00       cost: 109
-----------------------------------------------------
session_id: 10  movie_id: 2     number: 2       time: 2021-01-02 09:09:00       cost: 80
-----------------------------------------------------
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок8 – Таблиця session після вставлення запису 10

Якщо спробувати додати запис з неіснуючим зовнішнім ключем – програма видасть помилку.

```
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py insert_record session 11 80 2 2021.01.02.09.09.00 80
select count(*) from public."session" where session_id=11
select count(*) from public."movie" where movie_id=80
select count(*) from public."hall" where number=2
Something went wrong (record with such id exists or inappropriate foreign key values)
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок9 – Спроба додати запис до таблиці session з неіснуючим зовнішнім ключем movie_id = 80

```
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py print_table seat
SELECT * FROM public."seat"
seat table:
seat_id: 212    number: 2       row: 12         seat_number: 21         is_occupied: False
-----------------------------------------------------
seat_id: 10     number: 4       row: 3  seat_number: 31         is_occupied: True
-----------------------------------------------------
seat_id: 1      number: 1       row: 2  seat_number: 11         is_occupied: False
-----------------------------------------------------
seat_id: 4      number: 4       row: 11         seat_number: 22         is_occupied: False
-----------------------------------------------------
seat_id: 322    number: 3       row: 11         seat_number: 336        is_occupied: True
-----------------------------------------------------
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок10 – Таблиця seat до зміни запису 322

```
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py update_record seat 322 1 9 21 0
select count(*) from public."hall" where number=1
select count(*) from public."seat" where seat_id=322
UPDATE public."seat" SET number=1, row=9, seat_number=21, is_occupied=false WHERE seat_id=322;
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py print_table seat
SELECT * FROM public."seat"
seat table:
seat_id: 212    number: 2       row: 12         seat_number: 21         is_occupied: False
-----------------------------------------------------
seat_id: 10     number: 4       row: 3  seat_number: 31         is_occupied: True
-----------------------------------------------------
seat_id: 1      number: 1       row: 2  seat_number: 11         is_occupied: False
-----------------------------------------------------
seat_id: 4      number: 4       row: 11         seat_number: 22         is_occupied: False
-----------------------------------------------------
seat_id: 322    number: 1       row: 9  seat_number: 21         is_occupied: False
-----------------------------------------------------
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок11 – Таблиця seat після зміни запису 322

```
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py print_table cinema
SELECT * FROM public."cinema"
cinema table:
cinema_id: 0    name: Poposha   address: Lykyshkina 9
-----------------------------------------------------
cinema_id: 1    name: Kokosha   address: Voluna 222
-----------------------------------------------------
cinema_id: 3    name: Kyiv_cinema       address: Holybsya99
-----------------------------------------------------
cinema_id: 2    name: Kyiv_cinema2      address: Freska11
-----------------------------------------------------
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py update_record movie 1 8 Kolysha 8
select count(*) from public."cinema" where cinema_id=8
select count(*) from public."movie" where movie_id=1
Something went wrong (record with such id does not exist or inappropriate foreign key value)
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок11 – Таблиця cinema та спроба змінити запис дочірньої таблиці із неіснуючим зовнішнім ключем cinema_id = 8

```
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py print_table cinema
SELECT * FROM public."cinema"
cinema table:
cinema_id: 0    name: Poposha   address: Lykyshkina 9

-------------------------------------------------------
cinema_id: 1    name: Kokosha   address: Voluna 222

-------------------------------------------------------
cinema_id: 3    name: Kyiv_cinema       address: Holybsya99

-------------------------------------------------------
cinema_id: 2    name: Kyiv_cinema2      address: Freska11

-------------------------------------------------------
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок12 – Таблиця cinema до вставки 6 рандомізованих значень

```
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py generate_randomly cinema 6
insert into public."cinema" select (SELECT MAX(cinema_id)+1 FROM public."cinema"), array_to_string(ARRAY
(SELECT chr((97 + round(random() * 25)) :: integer) FROM generate_series(1, FLOOR(RANDOM()*(25-10)+10)::
 integer)), ''), array_to_string(ARRAY(SELECT chr((97 + round(random() * 25)) :: integer) FROM generate_
series(1, FLOOR(RANDOM()*(10-4)+4):: integer)), '');
insert into public."cinema" select (SELECT MAX(cinema_id)+1 FROM public."cinema"), array_to_string(ARRAY
(SELECT chr((97 + round(random() * 25)) :: integer) FROM generate_series(1, FLOOR(RANDOM()*(25-10)+10)::
 integer)), ''), array_to_string(ARRAY(SELECT chr((97 + round(random() * 25)) :: integer) FROM generate_
series(1, FLOOR(RANDOM()*(10-4)+4):: integer)), '');
insert into public."cinema" select (SELECT MAX(cinema_id)+1 FROM public."cinema"), array_to_string(ARRAY
(SELECT chr((97 + round(random() * 25)) :: integer) FROM generate_series(1, FLOOR(RANDOM()*(25-10)+10)::
 integer)), ''), array_to_string(ARRAY(SELECT chr((97 + round(random() * 25)) :: integer) FROM generate_
series(1, FLOOR(RANDOM()*(10-4)+4):: integer)), '');
insert into public."cinema" select (SELECT MAX(cinema_id)+1 FROM public."cinema"), array_to_string(ARRAY
(SELECT chr((97 + round(random() * 25)) :: integer) FROM generate_series(1, FLOOR(RANDOM()*(25-10)+10)::
 integer)), ''), array_to_string(ARRAY(SELECT chr((97 + round(random() * 25)) :: integer) FROM generate_
series(1, FLOOR(RANDOM()*(10-4)+4):: integer)), '');
insert into public."cinema" select (SELECT MAX(cinema_id)+1 FROM public."cinema"), array_to_string(ARRAY
(SELECT chr((97 + round(random() * 25)) :: integer) FROM generate_series(1, FLOOR(RANDOM()*(25-10)+10)::
 integer)), ''), array_to_string(ARRAY(SELECT chr((97 + round(random() * 25)) :: integer) FROM generate_
series(1, FLOOR(RANDOM()*(10-4)+4):: integer)), '');
insert into public."cinema" select (SELECT MAX(cinema_id)+1 FROM public."cinema"), array_to_string(ARRAY
(SELECT chr((97 + round(random() * 25)) :: integer) FROM generate_series(1, FLOOR(RANDOM()*(25-10)+10)::
 integer)), ''), array_to_string(ARRAY(SELECT chr((97 + round(random() * 25)) :: integer) FROM generate_
series(1, FLOOR(RANDOM()*(10-4)+4):: integer)), '');
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок13 – Генерація 6 рандомізованих значень таблиці cinema

```
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py print_table cinema
SELECT * FROM public."cinema"
cinema table:
cinema_id: 0    name: Poposha   address: Lykyshkina 9
------------------------------------------------------
cinema_id: 1    name: Kokosha   address: Voluna 222
------------------------------------------------------
cinema_id: 3    name: Kyiv_cinema       address: Holybsya99
------------------------------------------------------
cinema_id: 2    name: Kyiv_cinema2      address: Freska11
------------------------------------------------------
cinema_id: 4    name: stdiccqxhofbtuaqfd        address: sisinwea
------------------------------------------------------
cinema_id: 5    name: qykdtbokmyxhnqy   address: wiomqhoz
------------------------------------------------------
cinema_id: 6    name: bbyfekmqgxqlwnwklywex     address: astm
------------------------------------------------------
cinema_id: 7    name: iaoemvhdjgfeuzdqxzjjqpr   address: nmqbfj
------------------------------------------------------
cinema_id: 8    name: hmbpqtqbxmxbzfprvssejx    address: ekhgztgbt
------------------------------------------------------
cinema_id: 9    name: hnsrikhcyyceqsknclag      address: dycwqdg
------------------------------------------------------
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок14 – Таблиця cinema після вставки 6 рандомізованих значень

```
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py print_table movie
SELECT * FROM public."movie"
movie table:
movie_id: 1     cinema_id: 0    title: Harry Potter     rating: 9
------------------------------------------------------
movie_id: 2     cinema_id: 0    title: Operation `i`    rating: 10
------------------------------------------------------
movie_id: 3     cinema_id: 1    title: Ivan Vasylievich changes profession      rating: 8
------------------------------------------------------
movie_id: 4     cinema_id: 2    title: Love and pigeons         rating: 5
------------------------------------------------------
movie_id: 5     cinema_id: 1    title: FreefunIvan      rating: 2
------------------------------------------------------
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок15 – Таблиця movie до вставки 6 рандомізованих значень

```
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py generate_randomly movie 6
insert into public."movie" select (SELECT (MAX(movie_id)+1) FROM public."movie"), (SELECT cinema_id FROM
 public."cinema" LIMIT 1 OFFSET (round(random() *((SELECT COUNT(cinema_id) FROM public."cinema")-1)))),
array_to_string(ARRAY(SELECT chr((97 + round(random() * 25)) :: integer) FROM generate_series(1, FLOOR(R
ANDOM()*(10-3)+3):: integer)), ''), FLOOR(RANDOM()*(11-1)+1);
insert into public."movie" select (SELECT (MAX(movie_id)+1) FROM public."movie"), (SELECT cinema_id FROM
 public."cinema" LIMIT 1 OFFSET (round(random() *((SELECT COUNT(cinema_id) FROM public."cinema")-1)))),
array_to_string(ARRAY(SELECT chr((97 + round(random() * 25)) :: integer) FROM generate_series(1, FLOOR(R
ANDOM()*(10-3)+3):: integer)), ''), FLOOR(RANDOM()*(11-1)+1);
insert into public."movie" select (SELECT (MAX(movie_id)+1) FROM public."movie"), (SELECT cinema_id FROM
 public."cinema" LIMIT 1 OFFSET (round(random() *((SELECT COUNT(cinema_id) FROM public."cinema")-1)))),
array_to_string(ARRAY(SELECT chr((97 + round(random() * 25)) :: integer) FROM generate_series(1, FLOOR(R
ANDOM()*(10-3)+3):: integer)), ''), FLOOR(RANDOM()*(11-1)+1);
insert into public."movie" select (SELECT (MAX(movie_id)+1) FROM public."movie"), (SELECT cinema_id FROM
 public."cinema" LIMIT 1 OFFSET (round(random() *((SELECT COUNT(cinema_id) FROM public."cinema")-1)))),
array_to_string(ARRAY(SELECT chr((97 + round(random() * 25)) :: integer) FROM generate_series(1, FLOOR(R
ANDOM()*(10-3)+3):: integer)), ''), FLOOR(RANDOM()*(11-1)+1);
insert into public."movie" select (SELECT (MAX(movie_id)+1) FROM public."movie"), (SELECT cinema_id FROM
 public."cinema" LIMIT 1 OFFSET (round(random() *((SELECT COUNT(cinema_id) FROM public."cinema")-1)))),
array_to_string(ARRAY(SELECT chr((97 + round(random() * 25)) :: integer) FROM generate_series(1, FLOOR(R
ANDOM()*(10-3)+3):: integer)), ''), FLOOR(RANDOM()*(11-1)+1);
insert into public."movie" select (SELECT (MAX(movie_id)+1) FROM public."movie"), (SELECT cinema_id FROM
 public."cinema" LIMIT 1 OFFSET (round(random() *((SELECT COUNT(cinema_id) FROM public."cinema")-1)))),
array_to_string(ARRAY(SELECT chr((97 + round(random() * 25)) :: integer) FROM generate_series(1, FLOOR(R
ANDOM()*(10-3)+3):: integer)), ''), FLOOR(RANDOM()*(11-1)+1);
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок16 – Генерація 6 рандомізованих значень таблиці movie

```
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py print_table movie
SELECT * FROM public."movie"
movie table:
movie_id: 1     cinema_id: 0    title: Harry Potter     rating: 9

-------------------------------------------------------
movie_id: 2     cinema_id: 0    title: Operation `i`    rating: 10

-------------------------------------------------------
movie_id: 3     cinema_id: 1    title: Ivan Vasylievich changes profession      rating: 8

-------------------------------------------------------
movie_id: 4     cinema_id: 2    title: Love and pigeons         rating: 5

-------------------------------------------------------
movie_id: 5     cinema_id: 1    title: FreefunIvan      rating: 2

-------------------------------------------------------
movie_id: 6     cinema_id: 5    title: wfcbkxjws        rating: 8

-------------------------------------------------------
movie_id: 7     cinema_id: 0    title: ucrycnlb         rating: 1

-------------------------------------------------------
movie_id: 8     cinema_id: 1    title: msbg     rating: 4

-------------------------------------------------------
movie_id: 9     cinema_id: 6    title: gpf      rating: 4

-------------------------------------------------------
movie_id: 10    cinema_id: 5    title: xpevkxxb         rating: 1

-------------------------------------------------------
movie_id: 11    cinema_id: 2    title: cbg      rating: 10

-------------------------------------------------------
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок17 – Таблиця movie після вставки 6 рандомізованих значень

```
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py print_table session
SELECT * FROM public."session"
session table:
session_id: 2   movie_id: 1     number: 1       time: 2021-12-11 00:00:00       cost: 200
-------------------------------------------------------
session_id: 1   movie_id: 5     number: 2       time: 2020-01-01 12:30:00       cost: 10
-------------------------------------------------------
session_id: 3   movie_id: 5     number: 2       time: 2022-03-28 00:00:00       cost: 199
-------------------------------------------------------
session_id: 4   movie_id: 5     number: 1       time: 2012-09-21 00:00:00       cost: 10
-------------------------------------------------------
session_id: 9   movie_id: 1     number: 1       time: 2020-11-02 12:22:00       cost: 109
-------------------------------------------------------
session_id: 10  movie_id: 2     number: 2       time: 2021-01-02 09:09:00       cost: 80
-------------------------------------------------------
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок18 – Таблиця session до вставки 6 рандомізованих значень

```
(SELECT number FROM public."hall" LIMIT 1 OFFSET (round(random() * ((SELECT COUNT(number) FROM public."h
all")-1)))), (select timestamp '2018-01-10 10:00:00' + random() * (timestamp '2021-01-20 20:00:00' - tim
estamp '2018-01-10 10:00:00')), FLOOR(RANDOM()*(1000-20)+20);
insert into public."session"select (SELECT MAX(session_id)+1 FROM public."session"), (SELECT movie_id FR
OM public."movie" LIMIT 1 OFFSET (round(random() * ((SELECT COUNT(movie_id) FROM public."movie")-1)))),
(SELECT number FROM public."hall" LIMIT 1 OFFSET (round(random() * ((SELECT COUNT(number) FROM public."h
all")-1)))), (select timestamp '2018-01-10 10:00:00' + random() * (timestamp '2021-01-20 20:00:00' - tim
estamp '2018-01-10 10:00:00')), FLOOR(RANDOM()*(1000-20)+20);
insert into public."session"select (SELECT MAX(session_id)+1 FROM public."session"), (SELECT movie_id FR
OM public."movie" LIMIT 1 OFFSET (round(random() * ((SELECT COUNT(movie_id) FROM public."movie")-1)))),
(SELECT number FROM public."hall" LIMIT 1 OFFSET (round(random() * ((SELECT COUNT(number) FROM public."h
all")-1)))), (select timestamp '2018-01-10 10:00:00' + random() * (timestamp '2021-01-20 20:00:00' - tim
estamp '2018-01-10 10:00:00')), FLOOR(RANDOM()*(1000-20)+20);
insert into public."session"select (SELECT MAX(session_id)+1 FROM public."session"), (SELECT movie_id FR
OM public."movie" LIMIT 1 OFFSET (round(random() * ((SELECT COUNT(movie_id) FROM public."movie")-1)))),
(SELECT number FROM public."hall" LIMIT 1 OFFSET (round(random() * ((SELECT COUNT(number) FROM public."h
all")-1)))), (select timestamp '2018-01-10 10:00:00' + random() * (timestamp '2021-01-20 20:00:00' - tim
estamp '2018-01-10 10:00:00')), FLOOR(RANDOM()*(1000-20)+20);
insert into public."session"select (SELECT MAX(session_id)+1 FROM public."session"), (SELECT movie_id FR
OM public."movie" LIMIT 1 OFFSET (round(random() * ((SELECT COUNT(movie_id) FROM public."movie")-1)))),
(SELECT number FROM public."hall" LIMIT 1 OFFSET (round(random() * ((SELECT COUNT(number) FROM public."h
all")-1)))), (select timestamp '2018-01-10 10:00:00' + random() * (timestamp '2021-01-20 20:00:00' - tim
estamp '2018-01-10 10:00:00')), FLOOR(RANDOM()*(1000-20)+20);
insert into public."session"select (SELECT MAX(session_id)+1 FROM public."session"), (SELECT movie_id FR
OM public."movie" LIMIT 1 OFFSET (round(random() * ((SELECT COUNT(movie_id) FROM public."movie")-1)))),
(SELECT number FROM public."hall" LIMIT 1 OFFSET (round(random() * ((SELECT COUNT(number) FROM public."h
all")-1)))), (select timestamp '2018-01-10 10:00:00' + random() * (timestamp '2021-01-20 20:00:00' - tim
estamp '2018-01-10 10:00:00')), FLOOR(RANDOM()*(1000-20)+20);
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок19 – Генерація 6 рандомізованих значень таблиці session

```
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py print_table session
SELECT * FROM public."session"
session table:
session_id: 2   movie_id: 1     number: 1       time: 2021-12-11 00:00:00       cost: 200
-----------------------------------------------------
session_id: 1   movie_id: 5     number: 2       time: 2020-01-01 12:30:00       cost: 10
-----------------------------------------------------
session_id: 3   movie_id: 5     number: 2       time: 2022-03-28 00:00:00       cost: 199
-----------------------------------------------------
session_id: 4   movie_id: 5     number: 1       time: 2012-09-21 00:00:00       cost: 10
-----------------------------------------------------
session_id: 9   movie_id: 1     number: 1       time: 2020-11-02 12:22:00       cost: 109
-----------------------------------------------------
session_id: 10  movie_id: 2     number: 2       time: 2021-01-02 09:09:00       cost: 80
-----------------------------------------------------
session_id: 11  movie_id: 7     number: 7       time: 2019-12-01 00:22:06.687820        cost: 103
-----------------------------------------------------
session_id: 12  movie_id: 7     number: 4       time: 2019-11-25 04:24:41.949817        cost: 783
-----------------------------------------------------
session_id: 13  movie_id: 7     number: 7       time: 2018-08-24 20:00:17.815274        cost: 251
-----------------------------------------------------
session_id: 14  movie_id: 3     number: 4       time: 2020-12-28 01:16:05.044444        cost: 452
-----------------------------------------------------
session_id: 15  movie_id: 3     number: 6       time: 2018-06-12 01:55:58.032998        cost: 972
-----------------------------------------------------
session_id: 16  movie_id: 9     number: 3       time: 2019-12-12 07:35:16.501171        cost: 821
-----------------------------------------------------
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок20 – Таблиця session після вставки 6 рандомізованих значень

```
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py search_records session movie movie_id movie_id
specify the number of attributes you`d like to search by: 3
specify the type of data you want to search for (numeric, string or date): numeric
specify the name of key by which you`d like to perform search in form: table_number.key_name: one.session_id
specify the left end of search interval: 0
specify the right end of search interval: 10
specify the type of data you want to search for (numeric, string or date): string
specify the name of key by which you`d like to perform search in form: table_number.key_name: two.title
specify the string you`d like to search for: FreefunIvan
specify the type of data you want to search for (numeric, string or date): numeric
specify the name of key by which you`d like to perform search in form: table_number.key_name: two.cinema_id
specify the left end of search interval: 0
specify the right end of search interval: 3
select * from public."session" as one inner join public."movie" as two on one."movie_id"=two."movie_id" where 0<one.se
ssion_id and one.session_id<10 and two.title LIKE 'FreefunIvan' and 0<two.cinema_id and two.cinema_id<3
```

Рисунок21 – Пошук за трьома атрибутами з двох таблиць (session, movie)

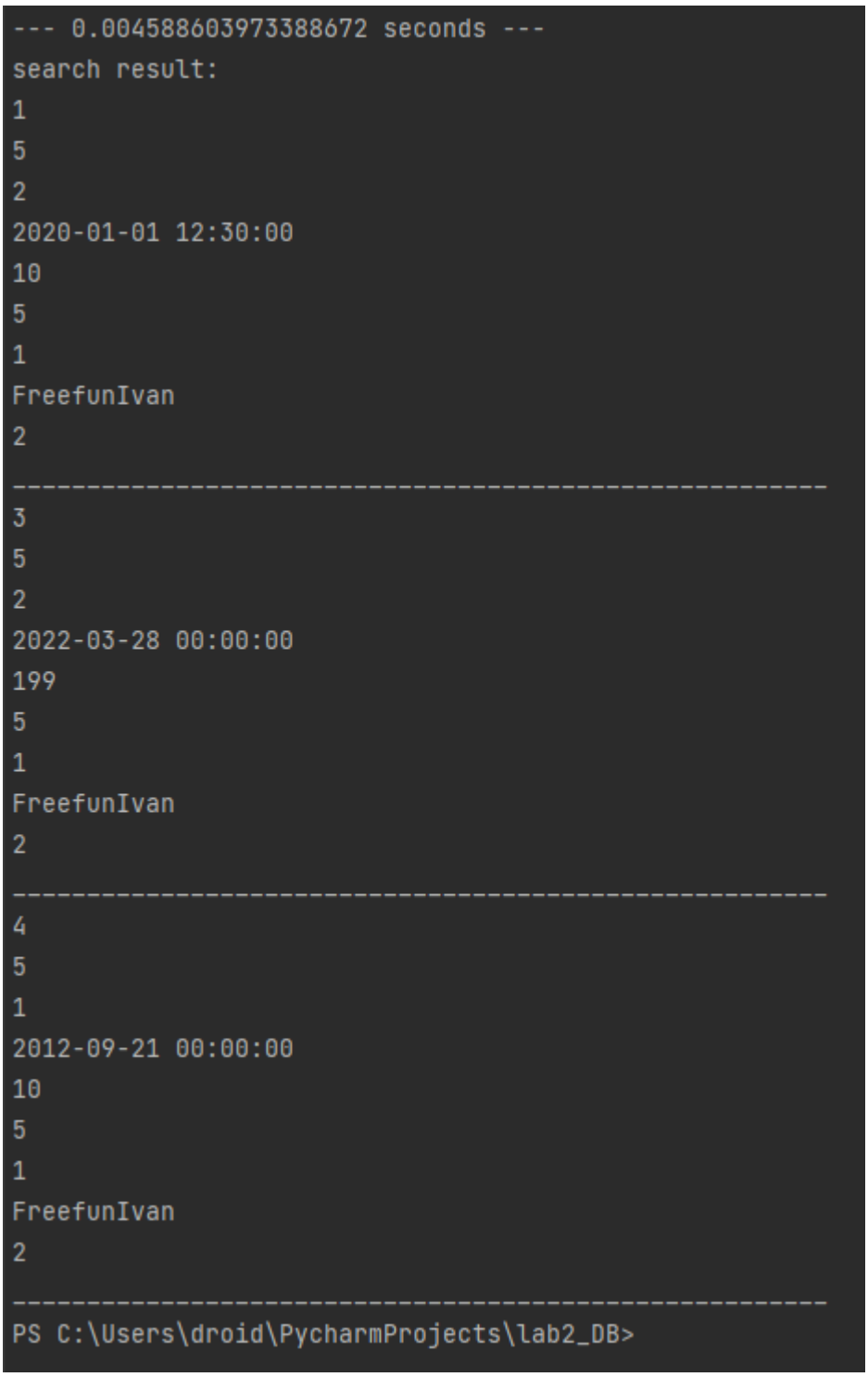

Рисунок22 – Результат пошуку, зображеного на рис.21

```
--------------------------------------------------
PS C:\Users\droid\PycharmProjects\lab2_DB> python main.py search_records session movie hall movie_id movie_id number number
specify the number of attributes you`d like to search by: 3
specify the type of data you want to search for (numeric, string or date): numeric
specify the name of key by which you`d like to perform search in form: table_number.key_name: one.cost
specify the left end of search interval: 20
specify the right end of search interval: 900
specify the type of data you want to search for (numeric, string or date): numeric
specify the name of key by which you`d like to perform search in form: table_number.key_name: two.rating
specify the left end of search interval: 0
specify the right end of search interval: 6
specify the type of data you want to search for (numeric, string or date): numeric
specify the name of key by which you`d like to perform search in form: table_number.key_name: three.number_of_seats
specify the left end of search interval: 50
specify the right end of search interval: 300
select * from public."session" as one inner join public."movie" as two on one."movie_id"=two."movie_id" inner join public."hall" as three o
n three."number"=one."number"where 20<one.cost and one.cost<900 and 0<two.rating and two.rating<6 and 50<three.number_of_seats and three.nu
mber_of_seats<300
```

Рисунок23 – Пошук за трьома атрибутами з трьох таблиць (session, movie, hall)

```
--- 0.0046606063842773344 seconds ---
search result:
12
7
4
2019-11-25 04:24:41.949817
783
7
0
ucrycnlb
1
4
1
130
100
------------------------------------------------------------
16
9
3
2019-12-12 07:35:16.501171
821
9
6
gpf
4
3
2
80
90
------------------------------------------------------------
PS C:\Users\droid\PycharmProjects\lab2_DB>
```

Рисунок24 – Результат пошуку, зображеного на рис.23

Текст програми

main.py

```
import controller as con
import sys

c = con.Controller()

try:
    command = sys.argv[1]
except IndexError:
    c.v.no_command()
else:
    if command == 'print_table':
        try:
            name = sys.argv[2]
        except IndexError:
            c.v.argument_error()
        else:
            c.print(name)

    elif command == 'delete_record':
        try:
            args = {"name": sys.argv[2], "key": sys.argv[3], "val": sys.argv[4]}
        except IndexError:
            c.v.argument_error()
        else:
            c.delete(args["name"], args["key"], args["val"])

    elif command == 'insert_record':
        try:
            args = {"name": sys.argv[2], "key": sys.argv[3]}
            if args["name"] == 'session':
                args["movie_id"], args["number"], args["time"], args["cost"] = \
                    sys.argv[4], sys.argv[5], sys.argv[6], sys.argv[7]
            elif args["name"] == 'movie':
                args["cinema_id"], args["title"], args["rating"] = \
                    sys.argv[4], sys.argv[5], sys.argv[6]
            elif args["name"] == 'hall':
                args["cinema_id"], args["screen_size"], args["number_of_seats"] = \
                    sys.argv[4], sys.argv[5], sys.argv[6]
            elif args["name"] == 'cinema':
                args["name1"], args["adress"] = \
                    sys.argv[4], sys.argv[5]
            elif args["name"] == 'seat':
                args["number"], args["row"], args["seat_number"], args["is_occupied"]
= \
                    sys.argv[4], sys.argv[5], sys.argv[6], sys.argv[7]
            else:
                c.v.wrong_table()
        except IndexError:
            c.v.argument_error()
        else:
            if args["name"] == 'session':
                c.insert_session(args["key"], args["movie_id"], args["number"],
args["time"], args["cost"])
            elif args["name"] == 'movie':
                c.insert_movie(args["key"], args["cinema_id"], args["title"],
args["rating"])
```

```
            elif args["name"] == 'hall':
                c.insert_hall(args["key"], args["cinema_id"], args["screen_size"],
args["number_of_seats"])
            elif args["name"] == 'cinema':
                c.insert_cinema(args["key"], args["name1"], args["adress"])
            elif args["name"] == 'seat':
                c.insert_seat(args["key"], args["number"], args["row"],
args["seat_number"], args["is_occupied"])

    elif command == 'update_record':
        try:
            args = {"name": sys.argv[2], "key": sys.argv[3]}
            if args["name"] == 'session':
                args["movie_id"], args["number"], args["time"], args["cost"] = \
                    sys.argv[4], sys.argv[5], sys.argv[6], sys.argv[7]
            elif args["name"] == 'movie':
                args["cinema_id"], args["title"], args["rating"] = \
                    sys.argv[4], sys.argv[5], sys.argv[6]
            elif args["name"] == 'hall':
                args["cinema_id"], args["screen_size"], args["number_of_seats"] = \
                    sys.argv[4], sys.argv[5], sys.argv[6]
            elif args["name"] == 'seat':
                args["number"], args["row"], args["seat_number"], args["is_occupied"]
= \
                    sys.argv[4], sys.argv[5], sys.argv[6], sys.argv[7]
            elif args["name"] == 'cinema':
                args["name1"], args["adress"] = \
                    sys.argv[4], sys.argv[5]
            else:
                c.v.wrong_table()
        except IndexError:
            c.v.argument_error()
        else:
            if args["name"] == 'session':
                c.update_session(args["key"], args["movie_id"], args["number"],
args["time"], args["cost"])
            elif args["name"] == 'movie':
                c.update_movie(args["key"], args["cinema_id"], args["title"],
args["rating"])
            elif args["name"] == 'hall':
                c.update_hall(args["key"], args["cinema_id"], args["screen_size"],
args["number_of_seats"])
            elif args["name"] == 'seat':
                c.update_seat(args["key"], args["number"], args["row"],
args["seat_number"], args["is_occupied"])
            elif args["name"] == 'cinema':
                c.update_cinema(args["key"], args["name1"], args["adress"])

    elif command == 'generate_randomly':
        try:
            args = {"name": sys.argv[2], "n": int(sys.argv[3])}
        except (IndexError, Exception):
            print(Exception, IndexError)
        else:
            c.generate(args["name"], args["n"])

    elif command == 'search_records':
        if len(sys.argv) in [6, 9]:
            search_num = c.v.get_search_num()
            try:
                search_num = int(search_num)
            except ValueError:
```

```
                c.v.invalid_search_num()
            else:
                if search_num > 0:
                    if len(sys.argv) == 6:
                        args = {"table1_name": sys.argv[2], "table2_name":
sys.argv[3],
                                "key1_name": sys.argv[4], "key2_name": sys.argv[5]}
                        c.search_two(args["table1_name"], args["table2_name"],
args["key1_name"], args["key2_name"],
                                     c.v.proceed_search(search_num))
                    elif len(sys.argv) == 9:
                        args = {"table1_name": sys.argv[2], "table2_name":
sys.argv[3], "table3_name": sys.argv[4],
                                "key1_name": sys.argv[5], "key2_name": sys.argv[6],
"key3_name": sys.argv[7],
                                "key13_name": sys.argv[8]}
                        c.search_three(args["table1_name"], args["table2_name"],
args["table3_name"],
                                       args["key1_name"], args["key2_name"],
args["key3_name"], args["key13_name"],
                                       c.v.proceed_search(search_num))
                else:
                    c.v.invalid_search_num()
        else:
            c.v.argument_error()

    elif command == 'help':
        c.v.print_help()
    else:
        c.v.wrong_command()
```

controller.py

```
from psycopg2 import Error
import model
import view
import datetime
import time


class Controller:
    def __init__(self):
        self.v = view.View()
        self.m = model.Model()

    def print(self, table_name):
        t_name = self.v.valid.check_table_name(table_name)
        if t_name:
            if t_name == 'cinema':
                self.v.print_cinema(self.m.print_cinema())
            elif t_name == 'hall':
                self.v.print_hall(self.m.print_hall())
            elif t_name == 'movie':
                self.v.print_movie(self.m.print_movie())
            elif t_name == 'seat':
                self.v.print_seat(self.m.print_seat())
            elif t_name == 'session':
                self.v.print_session(self.m.print_session())

    def delete(self, table_name, key_name, value):
        t_name = self.v.valid.check_table_name(table_name)
```

```
        k_name = self.v.valid.check_pk_name(table_name, key_name)
        if t_name and k_name:
            count = self.m.find(t_name, k_name, value)
            k_val = self.v.valid.check_pk(value, count)
            if k_val:
                if t_name == 'movie':
                    count_s = self.m.find('session', k_name, value)[0]
                    if count_s:
                        self.v.cannot_delete()
                    else:
                        try:
                            self.m.delete_data(table_name, key_name, k_val)
                        except (Exception, Error) as _ex:
                            self.v.sql_error(_ex)
                elif t_name == 'cinema':
                    count_h = self.m.find('hall', k_name, value)[0]
                    count_m = self.m.find('movie', k_name, value)[0]
                    if count_h or count_m:
                        self.v.cannot_delete()
                    else:
                        try:
                            self.m.delete_data(table_name, key_name, k_val)
                        except (Exception, Error) as _ex:
                            self.v.sql_error(_ex)
                elif t_name == 'hall':
                    count_seat = self.m.find('seat', k_name, value)[0]
                    count_session = self.m.find('session', k_name, value)[0]
                    if count_seat or count_session:
                        self.v.cannot_delete()
                    else:
                        try:
                            self.m.delete_data(table_name, key_name, k_val)
                        except (Exception, Error) as _ex:
                            self.v.sql_error(_ex)
                else:
                    try:
                        self.m.delete_data(table_name, key_name, k_val)
                    except (Exception, Error) as _ex:
                        self.v.sql_error(_ex)
            else:
                self.v.deletion_error()

    def update_session(self, key: str, movie_id: str, number: str, time: str, cost:
str):
        if self.v.valid.check_possible_keys('session', 'session_id', key):
            count_s = self.m.find('session', 'session_id', int(key))
            s_val = self.v.valid.check_pk(key, count_s)
        if self.v.valid.check_possible_keys('movie', 'movie_id', movie_id):
            count_m = self.m.find('movie', 'movie_id', int(movie_id))
            m_val = self.v.valid.check_pk(movie_id, count_m)
        if self.v.valid.check_possible_keys('hall', 'number', number):
            count_h = self.m.find('hall', 'number', int(number))
            h_val = self.v.valid.check_pk(number, count_h)

        if m_val and h_val and s_val and self.v.valid.check_possible_keys('session',
'cost', cost):
            try:
                arr = [int(x) for x in time.split(sep='.')]
                self.m.update_data_session(s_val, m_val, h_val,
                                           datetime.datetime(arr[0], arr[1], arr[2],
arr[3], arr[4], arr[5]),
                                           float(cost))
```

```
            except (Exception, Error) as _ex:
                self.v.sql_error(_ex)
        else:
            self.v.updation_error()

    def update_movie(self, key: str, cinema_id: str, title: str, rating: str):
        if self.v.valid.check_possible_keys('cinema', 'cinema_id', cinema_id):
            count_c = self.m.find('cinema', 'cinema_id', int(cinema_id))
            c_val = self.v.valid.check_pk(cinema_id, count_c)
        if self.v.valid.check_possible_keys('movie', 'movie_id', key):
            count_m = self.m.find('movie', 'movie_id', int(key))
            m_val = self.v.valid.check_pk(key, count_m)

        if c_val and m_val and self.v.valid.check_possible_keys('movie', 'rating',
rating):
            try:
                self.m.update_data_movie(m_val, c_val, title, float(rating))
            except (Exception, Error) as _ex:
                self.v.sql_error(_ex)
        else:
            self.v.updation_error()

    def update_hall(self, key: str, cinema_id: str, screen_size: str,
number_of_seats: str):
        if self.v.valid.check_possible_keys('cinema', 'cinema_id', cinema_id):
            count_c = self.m.find('cinema', 'cinema_id', int(cinema_id))
            c_val = self.v.valid.check_pk(cinema_id, count_c)
        if self.v.valid.check_possible_keys('hall', 'number', key):
            count_h = self.m.find('hall', 'number', int(key))
            h_val = self.v.valid.check_pk(key, count_h)

        if c_val and h_val:
            try:
                self.m.update_data_hall(h_val, c_val, screen_size,
int(number_of_seats))
            except (Exception, Error) as _ex:
                self.v.sql_error(_ex)
        else:
            self.v.updation_error()

    def update_cinema(self, key: str, name: str, adress: str):
        if self.v.valid.check_possible_keys('cinema', 'cinema_id', key):
            count_c = self.m.find('cinema', 'cinema_id', int(key))
            c_val = self.v.valid.check_pk(key, count_c)

        if c_val:
            try:
                self.m.update_data_cinema(c_val, name, adress)
            except (Exception, Error) as _ex:
                self.v.sql_error(_ex)
        else:
            self.v.updation_error()

    def update_seat(self, key: str, number: str, row: str, seat_number: str,
is_occupied: str):
        if self.v.valid.check_possible_keys('hall', 'number', number):
            count_h = self.m.find('hall', 'number', int(number))
            h_val = self.v.valid.check_pk(number, count_h)
        if self.v.valid.check_possible_keys('seat', 'seat_id', key):
            count_s = self.m.find('seat', 'seat_id', int(key))
            s_val = self.v.valid.check_pk(key, count_s)
```

```
        if s_val and h_val :
            try:
                self.m.update_data_seat(s_val, h_val, int(row), int(seat_number),
int(is_occupied))
            except (Exception, Error) as _ex:
                self.v.sql_error(_ex)
        else:
            self.v.insertion_error()

    def insert_session(self, key: str, movie_id: str, number: str, time: str, cost:
str):
        if self.v.valid.check_possible_keys('session', 'session_id', key):
            count_s = self.m.find('session', 'session_id', int(key))[0]
        if self.v.valid.check_possible_keys('movie', 'movie_id', movie_id):
            count_m = self.m.find('movie', 'movie_id', int(movie_id))
            m_val = self.v.valid.check_pk(movie_id, count_m)
        if self.v.valid.check_possible_keys('hall', 'number', number):
            count_h = self.m.find('hall', 'number', int(number))
            h_val = self.v.valid.check_pk(number, count_h)

        if (not count_s or count_s == (0,)) and m_val and h_val \
                and self.v.valid.check_possible_keys('session', 'session_id', key) \
                and self.v.valid.check_possible_keys('session', 'cost', cost):
            try:
                arr = [int(x) for x in time.split(sep='.')]
                self.m.insert_data_session(int(key), m_val, h_val,
                                           datetime.datetime(arr[0], arr[1], arr[2],
                                                             arr[3], arr[4], arr[5]),
                                           float(cost))
            except (Exception, Error) as _ex:
                self.v.sql_error(_ex)
        else:
            self.v.insertion_error()

    def insert_movie(self, key: str, cinema_id: str, title: str, rating: str):
        if self.v.valid.check_possible_keys('cinema', 'cinema_id', cinema_id):
            count_c = self.m.find('cinema', 'cinema_id', int(cinema_id))
            c_val = self.v.valid.check_pk(int(cinema_id), count_c)
        if self.v.valid.check_possible_keys('movie', 'movie_id', key):
            count_m = self.m.find('movie', 'movie_id', int(key))[0]

        if (not count_m or count_m == (0,)) and c_val and
self.v.valid.check_possible_keys('movie', 'movie_id', key) \
                and self.v.valid.check_possible_keys('movie', 'rating', rating):
            try:
                self.m.insert_data_movie(int(key), c_val, title, float(rating))
            except (Exception, Error) as _ex:
                self.v.sql_error(_ex)
        else:
            self.v.insertion_error()

    def insert_hall(self, key: str, cinema_id: str, screen_size: str,
number_of_seats: str):
        if self.v.valid.check_possible_keys('cinema', 'cinema_id', cinema_id):
            count_c = self.m.find('cinema', 'cinema_id', int(cinema_id))
            c_val = self.v.valid.check_pk(cinema_id, count_c)
        if self.v.valid.check_possible_keys('hall', 'number', key):
            count_h = self.m.find('hall', 'number', int(key))[0]

        if (not count_h or count_h == (0,)) and c_val \
                and self.v.valid.check_possible_keys('hall', 'number', key):
            try:
```

```
                self.m.insert_data_hall(int(key), c_val, screen_size,
int(number_of_seats))
            except (Exception, Error) as _ex:
                self.v.sql_error(_ex)
        else:
            self.v.insertion_error()

    def insert_cinema(self, key: str, name: str, adress: str):
        if self.v.valid.check_possible_keys('cinema', 'cinema_id', key):
            count_c = self.m.find('cinema', 'cinema_id', int(key))[0]

        if (not count_c or count_c == (0,)) and
self.v.valid.check_possible_keys('cinema', 'cinema_id', key):
            try:
                self.m.insert_data_cinema(int(key), name, adress)
            except (Exception, Error) as _ex:
                self.v.sql_error(_ex)
        else:
            self.v.insertion_error()

    def insert_seat(self, key: str, number: str, row: str, seat_number: str,
is_occupied: str):
        if self.v.valid.check_possible_keys('hall', 'number', number):
            count_h = self.m.find('hall', 'number', int(number))
            h_val = self.v.valid.check_pk(number, count_h)
        if self.v.valid.check_possible_keys('seat', 'seat_id', key):
            count_s = self.m.find('seat', 'seat_id', int(key))[0]

        if (not count_s or count_s == (0,)) and h_val \
                and self.v.valid.check_possible_keys('seat', 'seat_id', key):
            try:
                self.m.insert_data_seat(int(key), h_val, int(row), int(seat_number),
int(is_occupied))
            except (Exception, Error) as _ex:
                self.v.sql_error(_ex)
        else:
            self.v.insertion_error()

    def generate(self, table_name: str, n: int):
        t_name = self.v.valid.check_table_name(table_name)
        if t_name:
            if t_name == 'session':
                self.m.session_data_n_rand(n)
            elif t_name == 'movie':
                self.m.movie_data_n_rand(n)
            elif t_name == 'hall':
                self.m.hall_data_n_rand(n)
            elif t_name == 'cinema':
                self.m.cinema_data_n_rand(n)
            elif t_name == 'seat':
                self.m.seat_data_n_rand(n)

    def search_two(self, table1_name: str, table2_name: str, table1_key: str,
table2_key: str, search: str):
        t1_n = self.v.valid.check_table_name(table1_name)
        t2_n = self.v.valid.check_table_name(table2_name)
        if t1_n and self.v.valid.check_key_names(t1_n, table1_key) and t2_n \
                and self.v.valid.check_key_names(t2_n, table2_key):
            start_time = time.time()
            result = self.m.search_data_two_tables(table1_name, table2_name,
table1_key, table2_key,
                                                   search)
```

```
            self.v.print_time(start_time)

            self.v.print_search(result)

    def search_three(self, table1_name: str, table2_name: str, table3_name: str,
                     table1_key: str, table2_key: str, table3_key: str, table13_key:
str,
                     search: str):
        t1_n = self.v.valid.check_table_name(table1_name)
        t2_n = self.v.valid.check_table_name(table2_name)
        t3_n = self.v.valid.check_table_name(table3_name)
        if t1_n and self.v.valid.check_key_names(t1_n, table1_key) and
self.v.valid.check_key_names(t1_n, table13_key) \
                and t2_n and self.v.valid.check_key_names(t2_n, table2_key) \
                and t3_n and self.v.valid.check_key_names(t3_n, table3_key) \
                and self.v.valid.check_key_names(t3_n, table13_key):
            start_time = time.time()
            result = self.m.search_data_three_tables(table1_name, table2_name,
table3_name,
                                                     table1_key, table2_key,
table3_key, table13_key,
                                                     search)
            self.v.print_time(start_time)
            self.v.print_search(result)
```

validator.py

```
import datetime


class Validator:
    def __init__(self):
        self.error = ''
        self.er_flag = False

    def check_table_name(self, arg: str):
        if arg in ['cinema', 'hall', 'seat', 'session', 'movie']:
            return arg
        else:
            self.er_flag = True
            self.error = f'table {arg} does not exist in the database'
            print(self.error)
            return False

    def check_pkey_value(self, arg: str, min_val: int, max_val: int):
        try:
            value = int(arg)
        except ValueError:
            self.er_flag = True
            self.error = f'{arg} is not correct primary key value'
            print(self.error)
            return 0
        else:
            if min_val <= value <= max_val:
                return value
            else:
                self.er_flag = True
                self.error = f'{arg} is not existing primary key value'
                print(self.error)
                return 0
```

```
    def check_pk_name(self, table_name: str, key_name: str):
        if table_name == 'cinema' and key_name == 'cinema_id' \
                or table_name == 'hall' and key_name == 'number' \
                or table_name == 'movie' and key_name == 'movie_id' \
                or table_name == 'seat' and key_name == 'seat_id' \
                or table_name == 'session' and key_name == 'session_id':
            return key_name
        else:
            self.er_flag = True
            self.error = f'key {key_name} is not a primary key of table {table_name}'
            print(self.error)
            return False

    def check_pk(self, val, count):
        try:
            value = int(val)
        except ValueError:
            self.er_flag = True
            self.error = f'{val} is not correct primary key value'
            print(self.error)
            return 0
        else:
            if count and not count == (0,):
                return value
            else:
                return 0

    def check_key_names(self, table_name: str, key: str):
        if table_name == 'session' and key in ['session_id', 'movie_id', 'number',
'time', 'cost']:
            return True
        elif table_name == 'movie' and key in ['movie_id', 'cinema_id', 'title',
'rating']:
            return True
        elif table_name == 'hall' and key in ['number', 'screen_size',
'number_of_seats']:
            return True
        elif table_name == 'cinema' and key in ['cinema_id', 'name', 'adress']:
            return True
        elif table_name == 'seat' and key in ['seat_id', 'number', 'row',
'seat_number', 'is_occupied']:
            return True
        else:
            self.er_flag = True
            self.error = f'{key} is not correct name for {table_name} table'
            print(self.error)
            return False

    def check_possible_keys(self, table_name: str, key: str, val):
        if table_name == 'session':
            if key in ['session_id', 'movie_id', 'number']:
                try:
                    value = int(val)
                except ValueError:
                    self.er_flag = True
                    self.error = f'{val} is not correct key value'
                    print(self.error)
                    return False
                else:
                    return True
            elif key == 'time':
                try:
```

```
                arr = [int(x) for x in val.split(sep='.')]
                datetime.datetime(arr[0], arr[1], arr[2], arr[3], arr[4], arr[5])
            except TypeError:
                self.er_flag = True
                self.error = f'{val} is not correct date value'
                print(self.error)
                return False
            else:
                return True
        elif key == 'cost':
            try:
                value = float(val)
            except ValueError:
                self.er_flag = True
                self.error = f'{val} is not correct cost value'
                print(self.error)
                return False
            else:
                return True
        else:
            self.er_flag = True
            self.error = f'{key} is not correct name for session table'
            print(self.error)
            return False
    elif table_name == 'movie':
        if key in ['movie_id', 'cinema_id']:
            try:
                value = int(val)
            except ValueError:
                self.er_flag = True
                self.error = f'{val} is not correct key value'
                print(self.error)
                return False
            else:
                return True
        elif key == 'title':
            return True
        elif key == 'rating':
            try:
                value = float(val)
            except ValueError:
                self.er_flag = True
                self.error = f'{val} is not correct cost value'
                print(self.error)
                return False
            else:
                return True
        else:
            self.er_flag = True
            self.error = f'{key} is not correct name for movie table'
            print(self.error)
            return False
    elif table_name == 'hall':
        if key in ['number', 'cinema_id']:
            try:
                value = int(val)
            except ValueError:
                self.er_flag = True
                self.error = f'{val} is not correct key value'
                print(self.error)
                return False
            else:
```

```
                return True
            elif key == 'screen_size':
                return True
            elif key == 'number_of_seats':
                try:
                    value = int(val)
                except ValueError:
                    self.er_flag = True
                    self.error = f'{val} is not correct number of seats value'
                    print(self.error)
                    return False
                else:
                    return True
            else:
                self.er_flag = True
                self.error = f'{key} is not correct name for hall table'
                print(self.error)
                return False
        elif table_name == 'seat':
            if key in ['seat_id', 'number']:
                try:
                    value = int(val)
                except ValueError:
                    self.er_flag = True
                    self.error = f'{val} is not correct key value'
                    print(self.error)
                    return False
                else:
                    return True
            elif key == 'row':
                try:
                    value = int(val)
                except ValueError:
                    self.er_flag = True
                    self.error = f'{val} is not correct number of seats value'
                    print(self.error)
                    return False
                else:
                    return True
            elif key == 'seat_number':
                try:
                    value = int(val)
                except ValueError:
                    self.er_flag = True
                    self.error = f'{val} is not correct number of seats value'
                    print(self.error)
                    return False
                else:
                    return True
            else:
                self.er_flag = True
                self.error = f'{key} is not correct name for hall table'
                print(self.error)
                return False
        elif table_name == 'cinema':
            if key == 'cinema_id':
                try:
                    value = int(val)
                except ValueError:
                    self.er_flag = True
                    self.error = f'{val} is not correct key value'
                    print(self.error)
```

```
                return False
            else:
                return True
        elif key in ['name', 'adress']:
            return True
        else:
            self.er_flag = True
            self.error = f'{key} is not correct name for cinema table'
            print(self.error)
            return False
```

view.py

```
import datetime
import time
import validator


class View:
    def __init__(self):
        self.valid = validator.Validator()

    def cannot_delete(self) -> None:
        print('this record is connected with another table, deleting will '
              'throw error')

    def sql_error(self, e) -> None:
        print("[INFO] Error while working with Postgresql", e)

    def insertion_error(self) -> None:
        print('Something went wrong (record with such id exists or inappropriate
foreign key values)')

    def updation_error(self) -> None:
        print('Something went wrong (record with such id does not exist or
inappropriate foreign key value)')

    def deletion_error(self) -> None:
        print('record with such id does not exist')

    def invalid_interval(self) -> None:
        print('invalid interval input')

    def print_time(self, start) -> None:
        print("--- %s seconds ---" % (time.time() - start))

    def print_search(self, result):
        print('search result:')
        for row in result:
            for i in range(0, len(row)):
                print(row[i])
            print('____________________________________________________________')

    def print_cinema(self, table):
        print('cinema table:')
        for row in table:
            print('cinema_id:', row[0], '\tname:', row[1], '\taddress:', row[2])
            print('____________________________________________________________')

    def print_hall(self, table):
        print('hall table:')
```

```
        for row in table:
            print('number:', row[0], '\tcinema_id:', row[1], '\tscreen_size:',
row[2], '\tnumber_of_seats:', row[3])
            print('_____________________________________________________')

    def print_movie(self, table):
        print('movie table:')
        for row in table:
            print('movie_id:', row[0], '\tcinema_id:', row[1], '\ttitle:', row[2],
'\trating:', row[3])
            print('_____________________________________________________')

    def print_seat(self, table):
        print('seat table:')
        for row in table:
            print('seat_id:', row[0], '\tnumber:', row[1], '\trow:', row[2],
'\tseat_number:', row[3], '\tis_occupied:', row[4])
            print('_____________________________________________________')

    def print_session(self, table):
        print('session table:')
        for row in table:
            print('session_id:', row[0], '\tmovie_id:', row[1], '\tnumber:', row[2],
'\ttime:', row[3], '\tcost:', row[4])
            print('_____________________________________________________')

    def print_help(self):
        print('print_table - outputs the specified table \n\targument (table_name) is
required')
        print('delete_record - deletes the specified record from table \n'
              '\targuments (table_name, key_name, key_value) are required')
        print('update_record - updates record with specified id in table\n'
              '\tsession args (table_name, session_id, movie_id, number(id of hall),
time, cost)\n'
              '\tmovie args (table_name, movie_id, cinema_id, title, rating)\n'
              '\thall args (table_name, number(id), cinema_id, screen_size,
number_of_seats)\n'
              '\tseat args (table_name, seat_id, number(id of hall), row,
seat_number, is_occupied)\n'
              '\tcinema args (table_name, cinema_id, name, adress)')
        print('insert_record - inserts record into specified table \n'
              '\tsession args (table_name, session_id, movie_id, number(id of hall),
time, cost)\n'
              '\tmovie args (table_name, movie_id, cinema_id, title, rating)\n'
              '\thall args (table_name, number(id), cinema_id, screen_size,
number_of_seats)\n'
              '\tseat args (table_name, seat_id, number(id of hall), row,
seat_number, is_occupied)\n'
              '\tcinema args (table_name, cinema_id, name, adress)')
        print('generate_randomly - generates n random records in table\n'
              '\targuments (table_name, n) are required')
        print('search_records - search for records in two or more tables using one or
more keys \n'
              '\targuments (table1_name, table2_name, table1_key, table2_key) are
required, \n'
              '\tif you want to perform search in more tables: \n'
              '\t(table1_name, table2_name, table3_name, table1_key, table2_key,
table3_key, table13_key) \n'
              '\t(table1_name, table2_name, table3_name, table4_name, table1_key,
table2_key, table3_key, table13_key, '
              'table4_key, table24_key)')
```

```
    def proceed_search(self, search_num):
        search = ''
        for i in range(0, search_num):
            while True:
                search_type = input('specify the type of data you want to search for
'
                                    '(numeric, string or date): ')
                if search_type == 'numeric' or search_type == 'string' or search_type
== 'date':
                    break
            key = input('specify the name of key by which you`d like to perform
search '
                        'in form: table_number.key_name: ')

            if search_type == 'numeric':
                a = input('specify the left end of search interval: ')
                b = input('specify the right end of search interval: ')
                if search == '':
                    search = self.numeric_search(a, b, key)
                else:
                    search += ' and ' + self.numeric_search(a, b, key)

            elif search_type == 'date':
                data = input('specify the left end of search interval '
                             'in form: year.month.day.hour.minute.second: ')
                datb = input('specify the right end of search interval '
                             'in form: year.month.day.hour.minute.second: ')
                if search == '':
                    search = self.date_search(data, datb, key)
                else:
                    search += ' and ' + self.date_search(data, datb, key)

            elif search_type == 'string':
                string = input('specify the string you`d like to search for: ')
                if search == '':
                    search = self.string_search(string, key)
                else:
                    search += ' and ' + self.string_search(string, key)
        return search

    def numeric_search(self, a: str, b: str, key: str):
        try:
            a, b = int(a), int(b)
        except ValueError:
            self.invalid_interval()
        else:
            return f"{a}<{key} and {key}<{b}"

    def date_search(self, a: str, b: str, key: str):
        try:
            arr = [int(x) for x in a.split(sep='.')]
            brr = [int(x) for x in b.split(sep='.')]
        except Exception:
            print(Exception)
            self.invalid_interval()
        else:
            return f"{key} BETWEEN \'{datetime.datetime(arr[0], arr[1], arr[2],
arr[3], arr[4], arr[5])}\' " \
                   f"AND \'{datetime.datetime(brr[0], brr[1], brr[2], brr[3], brr[4],
brr[5])}\'"

    def string_search(self, string: str, key: str):
```

```
        return f"{key} LIKE \'{string}\'"

    def get_search_num(self):
        return input('specify the number of attributes you`d like to search by: ')

    def invalid_search_num(self):
        print('should be number different from 0')

    def argument_error(self):
        print('no required arguments specified')

    def wrong_table(self):
        print('wrong table name')

    def no_command(self):
        print('no command name specified, type help to see possible commands')

    def wrong_command(self):
        print('unknown command name, type help to see possible commands')
```

model.py

```
import datetime
import psycopg2 as ps


class Model:
    def __init__(self):
        self.conn = None
        try:
            self.conn = ps.connect(
                database="lab1",
                user='postgres',
                password="postgres",
                host='localhost',
                port="5432",
            )
        except(Exception, ps.DatabaseError) as error:
            print("Error while working with Postgresql", error)

    def request(self, req: str):
        try:
            cursor = self.conn.cursor()
            print(req)
            cursor.execute(req)
            self.conn.commit()
            return True
        except(Exception, ps.DatabaseError, ps.ProgrammingError) as error:
            print(error)
            self.conn.rollback()
            return False

    def get(self, req: str):
        try:
            cursor = self.conn.cursor()
            print(req)
            cursor.execute(req)
            self.conn.commit()
            return cursor.fetchall()
        except(Exception, ps.DatabaseError, ps.ProgrammingError) as error:
            print(error)
```

```
            self.conn.rollback()
            return False

    def get_el(self, req: str):
        try:
            cursor = self.conn.cursor()
            print(req)
            cursor.execute(req)
            self.conn.commit()
            return cursor.fetchone()
        except(Exception, ps.DatabaseError, ps.ProgrammingError) as error:
            print(error)
            self.conn.rollback()
            return False

    def count(self, table_name: str):
        return self.get_el(f"select count(*) from public.\"{table_name}\"")

    def find(self, table_name: str, key_name: str, key_value: int):
        return self.get_el(f"select count(*) from public.\"{table_name}\" where
{key_name}={key_value}")

    def max(self, table_name: str, key_name: str):
        return self.get_el(f"select max({key_name}) from public.\"{table_name}\"")

    def min(self, table_name: str, key_name: str):
        return self.get_el(f"select min({key_name}) from public.\"{table_name}\"")

    def print_cinema(self) -> None:
        return self.get(f"SELECT * FROM public.\"cinema\"")

    def print_hall(self) -> None:
        return self.get(f"SELECT * FROM public.\"hall\"")

    def print_movie(self) -> None:
        return self.get(f"SELECT * FROM public.\"movie\"")

    def print_seat(self) -> None:
        return self.get(f"SELECT * FROM public.\"seat\"")

    def print_session(self) -> None:
        return self.get(f"SELECT * FROM public.\"session\"")

    def delete_data(self, table_name: str, key_name: str, key_value) -> None:
        self.request(f"DELETE FROM public.\"{table_name}\" WHERE
{key_name}={key_value};")

    def update_data_session(self, key_value: int, movie_id: int, number: int, time:
datetime.datetime,
                            cost: float) -> None:
        self.request(f"UPDATE public.\"session\" SET movie_id={movie_id},
number={number}, time=\'{time}\', "
                     f"cost={cost} WHERE session_id={key_value};")

    def update_data_movie(self, key_value: int, cinema_id: int, title: str, rating:
float) -> None:
        self.request(f"UPDATE public.\"movie\" SET cinema_id={cinema_id},
title=\'{title}\', "
                     f"rating={rating} WHERE movie_id={key_value};")

    def update_data_hall(self, key_value: int, cinema_id: int, screen_size: str,
number_of_seats: int) -> None:
```

```
        self.request(f"UPDATE public.\"hall\" SET cinema_id={cinema_id},
screen_size=\'{screen_size}\', "
                     f"number_of_seats={number_of_seats} WHERE number={key_value};")

    def update_data_seat(self, key_value: int, number: int, row: int, seat_number:
int, is_occupied: int) -> None:
        if(is_occupied == 1):
            is_occupied1 = 'true'
        else:
            is_occupied1 = 'false'
        self.request(f"UPDATE public.\"seat\" SET number={number}, row={row},
seat_number={seat_number}, "
                     f"is_occupied={is_occupied1} WHERE seat_id={key_value};")

    def update_data_cinema(self, key_value: int, name: str, adress: str) -> None:
        self.request(f"UPDATE public.\"cinema\" SET name=\'{name}\', "
                     f"adress=\'{adress}\' WHERE cinema_id={key_value};")

    def insert_data_session(self, session_id: int, movie_id: int, number: int, time:
datetime.datetime,
                            cost: float) -> None:
        self.request(f"insert into public.\"session\" (session_id, movie_id, number,
time, cost) "
                     f"VALUES ({session_id}, {movie_id}, {number}, \'{time}\',
{cost});")

    def insert_data_movie(self, movie_id: int, cinema_id: int, title: str, rating:
float) -> None:
        self.request(f"insert into public.\"movie\" (movie_id, cinema_id, title,
rating) "
                     f"VALUES ({movie_id}, {cinema_id}, \'{title}\', {rating});")

    def insert_data_hall(self, number: int, cinema_id: int, screen_size: str,
number_of_seats: int) -> None:
        self.request(f"insert into public.\"hall\" (number, cinema_id, screen_size,
number_of_seats) "
                     f"VALUES ({number}, {cinema_id}, \'{screen_size}\',
{number_of_seats});")

    def insert_data_seat(self, seat_id: int, number: int, row: int, seat_number: int,
is_occupied: int) -> None:
        if is_occupied == 1:
            is_occupied1 = 'true'
        else:
            is_occupied1 = 'false'
        self.request(f"insert into public.\"seat\" (seat_id, number, row,
seat_number, is_occupied) "
                     f"VALUES ({seat_id}, {number}, {row}, {seat_number},
\'{is_occupied1}\');")

    def insert_data_cinema(self, cinema_id: int, name: str, adress: str) -> None:
        self.request(f"insert into public.\"cinema\" (cinema_id, name, adress) "
                     f"VALUES ({cinema_id}, \'{name}\', \'{adress}\');")

    def session_data_n_rand(self, times: int) -> None:
        for i in range(times):
            self.request("insert into public.\"session\""
                         "select (SELECT MAX(session_id)+1 FROM public.\"session\"),
"
                         "(SELECT movie_id FROM public.\"movie\" LIMIT 1 OFFSET "
                         "(round(random() * ((SELECT COUNT(movie_id) FROM
public.\"movie\")-1)))), "
```

```
                          "(SELECT number FROM public.\"hall\" LIMIT 1 OFFSET
(round(random() * "
                          "((SELECT COUNT(number) FROM public.\"hall\")-1)))), "
                          "(select timestamp '2018-01-10 10:00:00' + random() * "
                          "(timestamp '2021-01-20 20:00:00' - timestamp '2018-01-10
10:00:00')), "
                          "FLOOR(RANDOM()*(1000-20)+20);")

    def movie_data_n_rand(self, times: int) -> None:
        for i in range(times):
            self.request("insert into public.\"movie\" select (SELECT
(MAX(movie_id)+1) FROM public.\"movie\"), "
                          "(SELECT cinema_id FROM public.\"cinema\" LIMIT 1 OFFSET "
                          "(round(random() *((SELECT COUNT(cinema_id) FROM
public.\"cinema\")-1)))), "
                          "array_to_string(ARRAY(SELECT chr((97 + round(random() *
25)) :: integer) "
                          "FROM generate_series(1, FLOOR(RANDOM()*(10-3)+3)::
integer)), ''), "
                          "FLOOR(RANDOM()*(11-1)+1);")

    def hall_data_n_rand(self, times: int) -> None:
        for i in range(times):
            self.request("insert into public.\"hall\" select (SELECT MAX(number)+1
FROM public.\"hall\"), "
                          "(SELECT cinema_id FROM public.\"cinema\" LIMIT 1 OFFSET "
                          "(round(random() *((SELECT COUNT(cinema_id) FROM
public.\"cinema\")-1)))), "
                          "FLOOR(RANDOM()*(300-80)+80), "
                          "FLOOR(RANDOM()*(350-20)+20);")

    def cinema_data_n_rand(self, times: int) -> None:
        for i in range(times):
            self.request("insert into public.\"cinema\" select (SELECT
MAX(cinema_id)+1 FROM public.\"cinema\"), "
                          "array_to_string(ARRAY(SELECT chr((97 + round(random() *
25)) :: integer) "
                          "FROM generate_series(1, FLOOR(RANDOM()*(25-10)+10)::
integer)), ''), "
                          "array_to_string(ARRAY(SELECT chr((97 + round(random() *
25)) :: integer) "
                          "FROM generate_series(1, FLOOR(RANDOM()*(10-4)+4)::
integer)), ''); ")

    def seat_data_n_rand(self, times: int) -> None:
        for i in range(times):
            self.request("insert into public.\"seat\" select (SELECT MAX(seat_id)+1
FROM public.\"seat\"), "
                          "(SELECT number FROM public.\"hall\" LIMIT 1 OFFSET "
                          "(round(random() *((SELECT COUNT(number) FROM
public.\"hall\")-1)))), "
                          "FLOOR(RANDOM()*(51-1)+1), "
                          "FLOOR(RANDOM()*(350-1)+1), "
                          "(round(random())::int)::boolean;")

    def search_data_two_tables(self, table1_name: str, table2_name: str, table1_key,
table2_key,
                               search: str):
        return self.get(f"select * from public.\"{table1_name}\" as one inner join
public.\"{table2_name}\" as two "
                        f"on one.\"{table1_key}\"=two.\"{table2_key}\" "
                        f"where {search}")
```

```
    def search_data_three_tables(self, table1_name: str, table2_name: str,
table3_name: str,
                                 table1_key, table2_key, table3_key, table13_key,
                                 search: str):
        return self.get(f"select * from public.\"{table1_name}\" as one inner join
public.\"{table2_name}\" as two "
                        f"on one.\"{table1_key}\"=two.\"{table2_key}\" inner join
public.\"{table3_name}\" as three "
                        f"on three.\"{table3_key}\"=one.\"{table13_key}\""
                        f"where {search}")
```