

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»**

Зображення, що містить текст, у приміщенні

Автоматично згенерований опис

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**

**Зображення, що містить текст

Автоматично згенерований 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**

Зображення, що містить текст

Автоматично згенерований опис

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 aбо 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 використовються 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**

Зображення, що містить текст

Автоматично згенерований опис Рисунок3 – Таблиця session до видалення запису 5

**Зображення, що містить текст

Автоматично згенерований опис**

Рисунок4 - Таблиця session після видалення запису 5

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

Зображення, що містить текст

Автоматично згенерований описРисунок5 – Таблиця movie

**Зображення, що містить текст

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

**Зображення, що містить текст

Автоматично згенерований опис**

Рисунок7 – Таблиця session до вставлення запису 10

**Зображення, що містить текст

Автоматично згенерований опис**

Рисунок8 – Таблиця session після вставлення запису 10

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

Зображення, що містить текст

Автоматично згенерований опис

Рисунок9 – Спроба додати запис до таблиці session з неіснуючим зовнішнім ключем movie\_id = 80

Зображення, що містить текст

Автоматично згенерований опис

Рисунок10 – Таблиця seat до зміни запису 322

Зображення, що містить текст

Автоматично згенерований опис

Рисунок11 – Таблиця seat після зміни запису 322

Зображення, що містить текст

Автоматично згенерований опис

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

Зображення, що містить текст

Автоматично згенерований опис

Рисунок12 – Таблиця cinema до вставки 6 рандомізованих значень

Зображення, що містить текст

Автоматично згенерований опис

Рисунок13 – Генерація 6 рандомізованих значень таблиці cinema

Зображення, що містить текст

Автоматично згенерований опис

Рисунок14 – Таблиця cinema після вставки 6 рандомізованих значень

Зображення, що містить текст

Автоматично згенерований опис

Рисунок15 – Таблиця movie до вставки 6 рандомізованих значень

Зображення, що містить текст

Автоматично згенерований опис

Рисунок16 – Генерація 6 рандомізованих значень таблиці movie

Зображення, що містить текст

Автоматично згенерований опис

Рисунок17 – Таблиця movie після вставки 6 рандомізованих значень

Зображення, що містить текст

Автоматично згенерований опис

Рисунок18 – Таблиця session до вставки 6 рандомізованих значень

Зображення, що містить текст

Автоматично згенерований опис

Рисунок19 – Генерація 6 рандомізованих значень таблиці session

Зображення, що містить текст

Автоматично згенерований опис

Рисунок20 – Таблиця session після вставки 6 рандомізованих значень

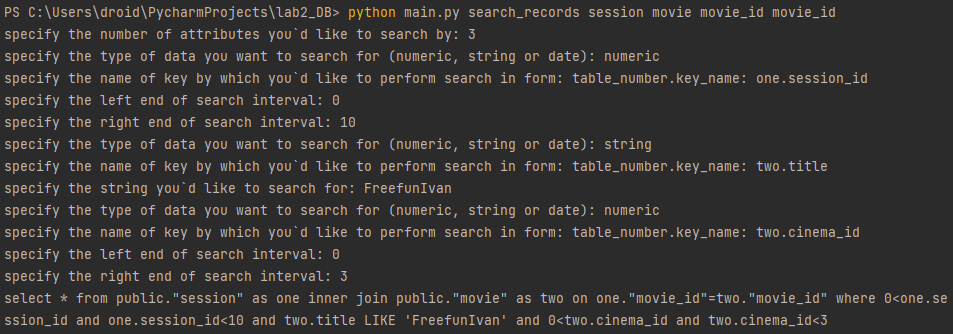


Рисунок21 – Пошук за трьома атрибутами з двох таблиць (session, movie)

Зображення, що містить текст

Автоматично згенерований опис

Рисунок22 – Результат пошуку, зображеного на рис.21

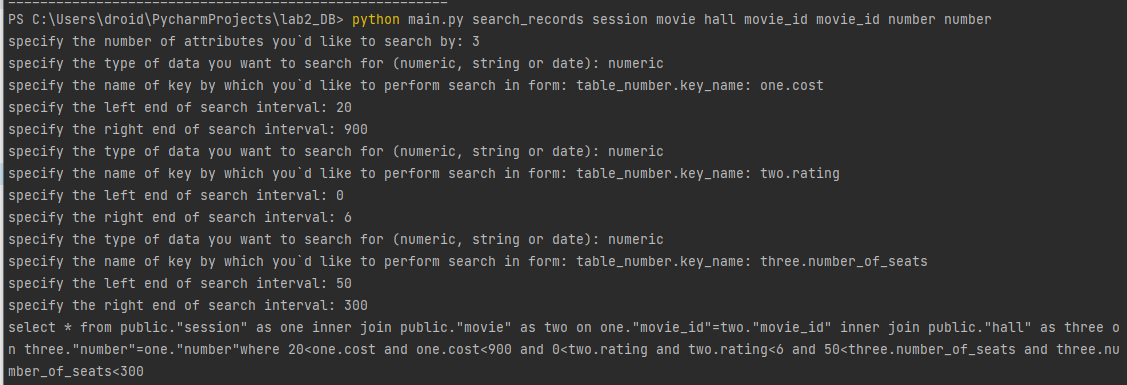


Рисунок23 – Пошук за трьома атрибутами з трьох таблиць (session, movie, hall)

Зображення, що містить текст

Автоматично згенерований опис

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}")