

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ

ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського» ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра системного програмування та спеціалізованих комп’ютерних систем

**Лабораторна робота №2**

з дисципліни **Бази даних і засоби управління**

# на тему: “Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL”

Виконала: студентка 3 курсу

групи КВ-92

Невінчаний А. В.

Перевірив: Павловський В.І.

Київ – 2021

# Мета роботи

Здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

***Загальне завдання* роботи полягає у наступному:**

1. Реалізувати функції перегляду, 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;

1. Програмний код виконати згідно шаблону MVC (модель-подання- контролер).

# Інформація про програму

Посилання на репозиторій у GitHub з вихідним кодом програми та звітом: <https://github.com/nutasanchik/DB_Lab2>

Використана мова програмування: Python 3.10

Використані бібліотеки: psycopg2 (для зв’язку з СУБД), time (для виміру часу запиту пошуку для завдання 3), sys (для реалізації консольного інтерфейсу).

# Інформація про обрану базу даних

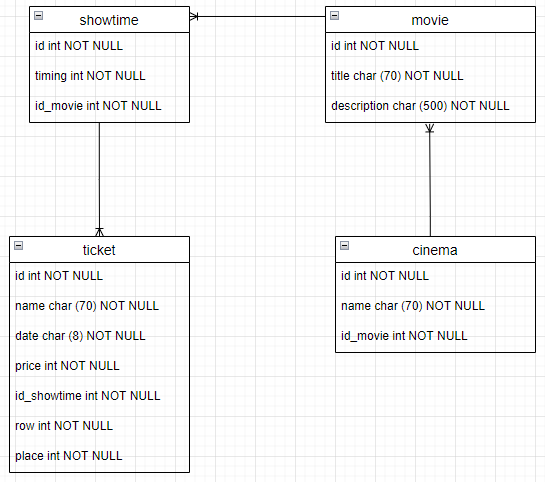


Рисунок 1. Схема бази даних Таблиця 1. Опис структури БД

|  |  |  |
| --- | --- | --- |
| **Відношення** | **Атрибут** | **Тип** |
| Відношення «movie»  Містить інформація про фільм | id – персональний номер фільму title – назва фільму  description – короткий опис фільму | Числовий  Текстовий (70)  Текстовий (500) |
| Відношення «showtime»  Містить інформацію про розклад сеансів | id – персональний номер сеансу timing – тривалість сеансу  id\_movie – номер фільму на даний сеанс | Числовий Числовий  Числовий |
| Відношення «cinema»  Містить інформацію про кінотеатр | id – персональний номер кінотеатру name – назва кінотеатру  id\_movie – персональний номер сеансу | Числовий  Текстовий (70) Числовий |
| Відношення «ticket»  Містить інформацію про придбаний квиток | id – персональний номер квитка name – ім’я глядача  date – дата перегляду фільму price – вартість квитка  id\_showtime – персональний номер сеансу row – номер ряду у залі  place – номер місця у залі | Числовий  Текстовий (70)  Текстовий (8) Числовий  Числовий Числовий  Числовий |

# Короткий опис структури БД «Кінотеатр»

Модель «сутність-зв’язок» галузі продажу квитків у кіно. Модель має чотири сутності: movie, showtime, cinema, ticket.

Сутність movie – описує фільм, на який був куплений квиток. Кожен фільм має власний id, name та description.

Сутність showtime – описує час сеансу. Атрибутами даної сутності є id, timing та id\_movie.

Сутність cinema – описує кінотеатр. Кожен кінотеатр має власний id, name, id\_movie.

Сутність ticket – описує квиток, куплений глядачем. Кожен квиток має власний id, name, date, price, id\_showtime, row, place.

# Реалізація пункту 1 завдання до ЛР

print\_table – виводить вміст заданої таблиці у вікно терміналу. У разі некоректності введених даних, у вікно терміналу виводиться помилка. Можливі аргументи: Cinema, Movie, Showtime, Ticket.

delete\_record – видаляє запис з вказаним первинним ключем. У разі некоректності введених даних, у вікно терміналу виводиться помилка. Аргументи: table\_name, key\_name, key\_value .

update\_record – змінює поля, за заданим первинним ключем. У разі некоректності введених даних, у вікно терміналу виводиться помилка. Можливі аргументи:

Cinema id(int) name(str) id\_movie(int) Movie id(int) title(str) description(str) Showtime id(int) timing(int) id\_movie(int)

Ticket id(int) name(str) date(int) price(int) id\_showtime(int) row(int) place(int) insert\_record – вставляє новий рядок у задану таблицю. У разі

некоректності введених даних, у вікно терміналу виводиться помилка. Можливі аргументи:

Cinema id(int) name(str) id\_movie(int) Movie id(int) title(str) description(str)

Showtime id(int) timing(int) id\_movie(int)

Ticket id(int) name(str) date(int) price(int) id\_showtime(int) row(int) place(int)

# Реалізація пункту 2 завдання до ЛР

generate\_randomly –здійснює генерування n псевдорандомізованих записів у заданій таблиці. У разі некоректності введених даних, у вікно терміналу виводиться помилка. Аргументи: table\_name, число записів, що мають бути створені.

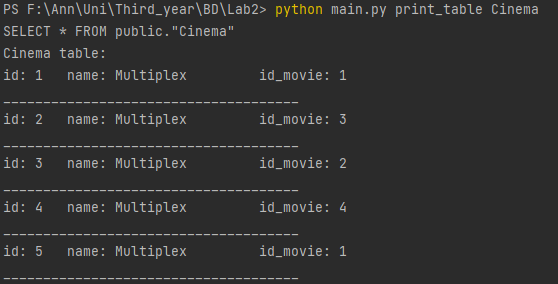
# Реалізація пункту 3 завдання до ЛР

search\_records –реалізує пошук за 1 та більше атрибутами з заданих 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. У разі некоректності введених даних, у вікно терміналу виводиться помилка. Початково потрібно вказати аргументи: table1\_name table2\_name table1\_key table2\_key aбо table1\_name table2\_name table3\_name table1\_key table2\_key table3\_key table13\_key або table1\_name table2\_name table3\_name table4\_name table1\_key table2\_key table3\_key table13\_key table4\_key table24\_key де table13\_key, table24\_key – це зовнішні ключі, що зв’язують 1 та 3 таблицю, або 2 та 4. 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 або four.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

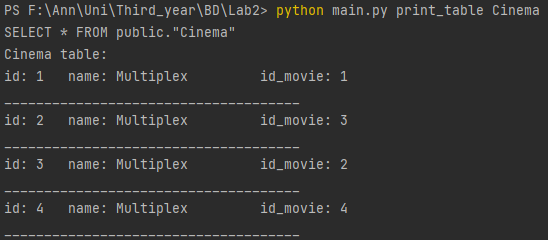
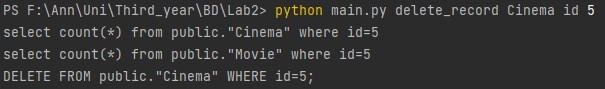
# Завдання 1

Видалення:

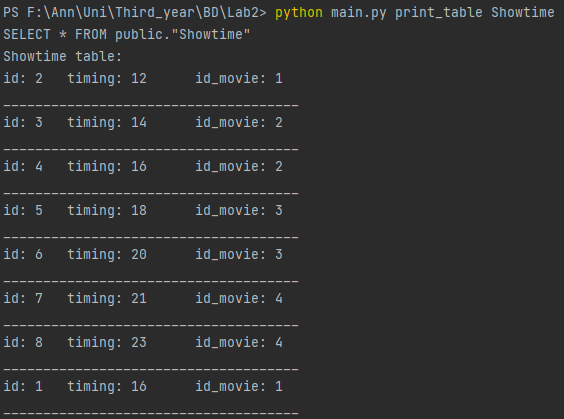
Таблиця Cinema до видалення запису 5:



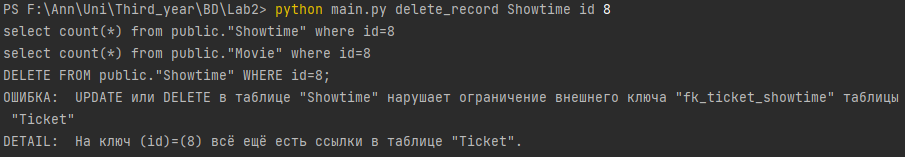
Таблиця Cinema після видалення запису 5:



Таблиця Showtime до видалення запису 8:

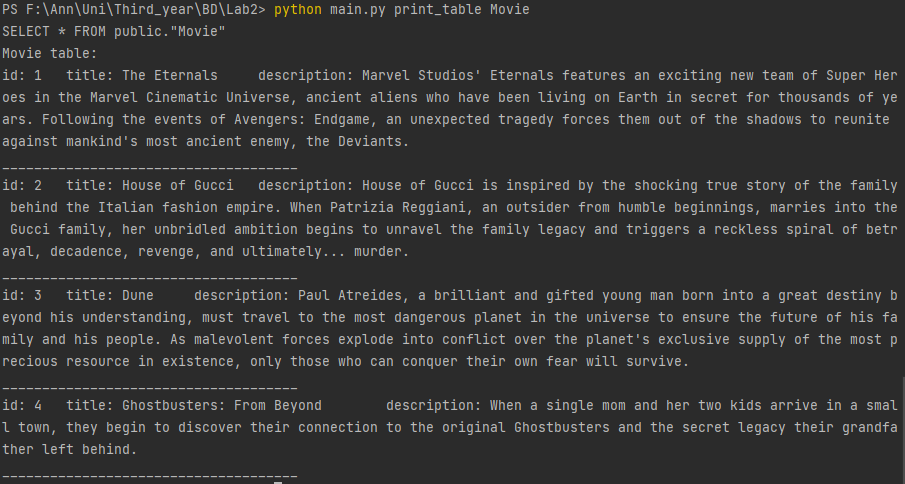


Таблиця Showtime після спроби видалення запису 8:

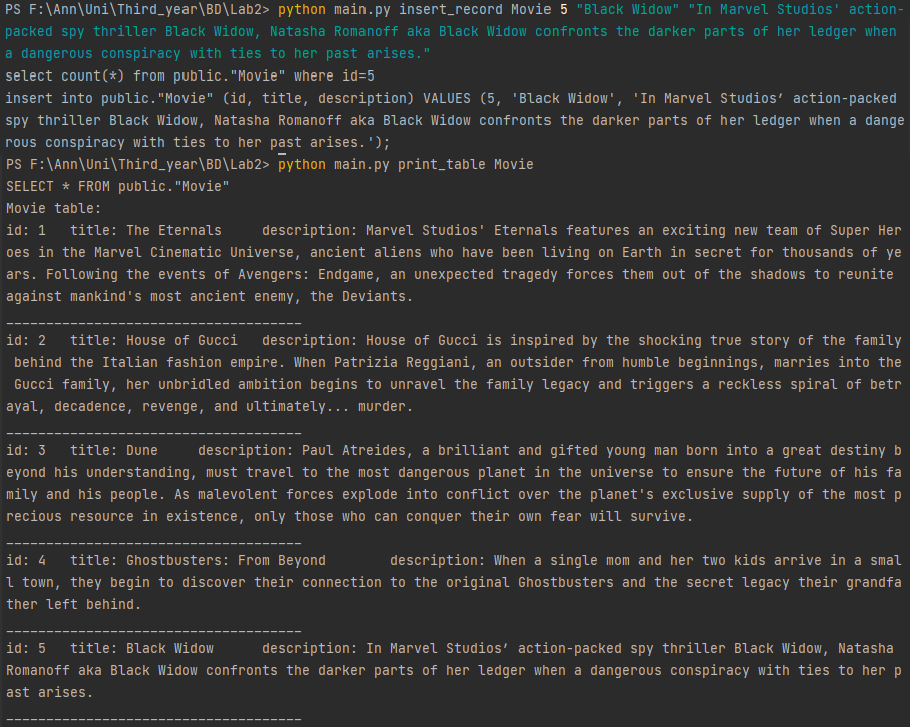


Вставка:

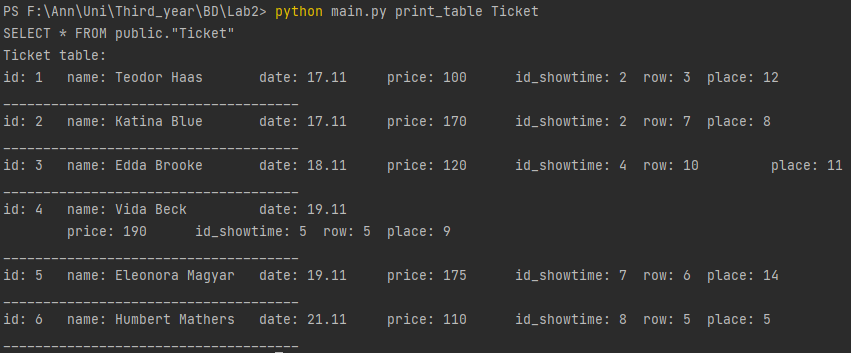
Таблиця Movie до вставки запису 5:



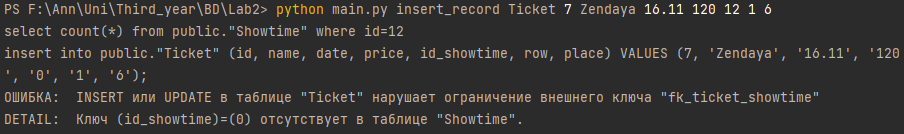
Таблиця Movie після вставки запису 7:



Таблиця Ticket до вставки запису 7:

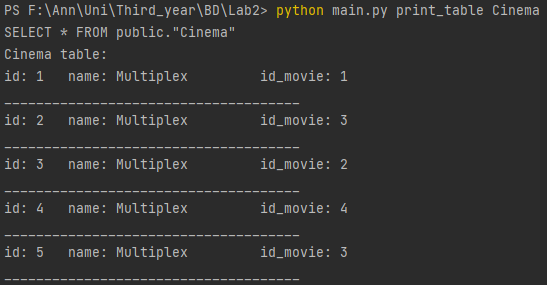


Таблиця Ticket після спроби вставки запису 7:

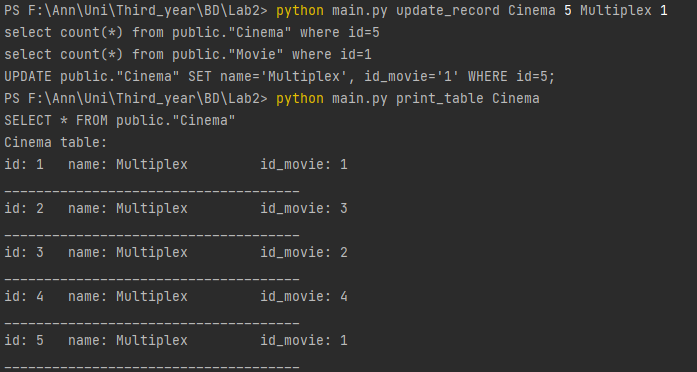


Зміна:

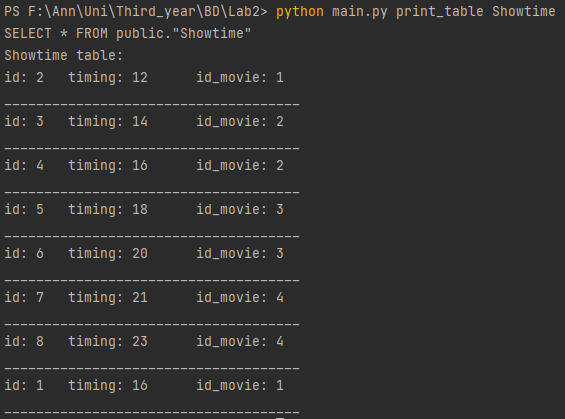
Таблиця Cinema до зміни запису 5:



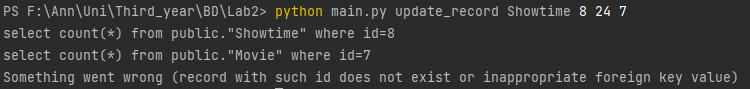
Таблиця Cinema після зміни запису 5:



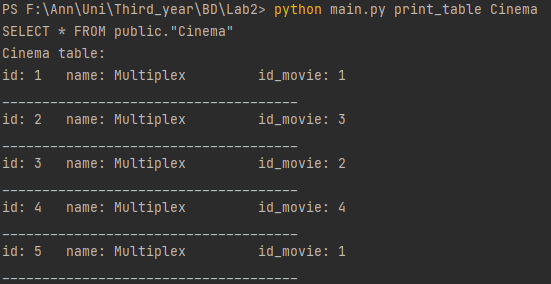
Таблиця Showtime до зміни запису 8:



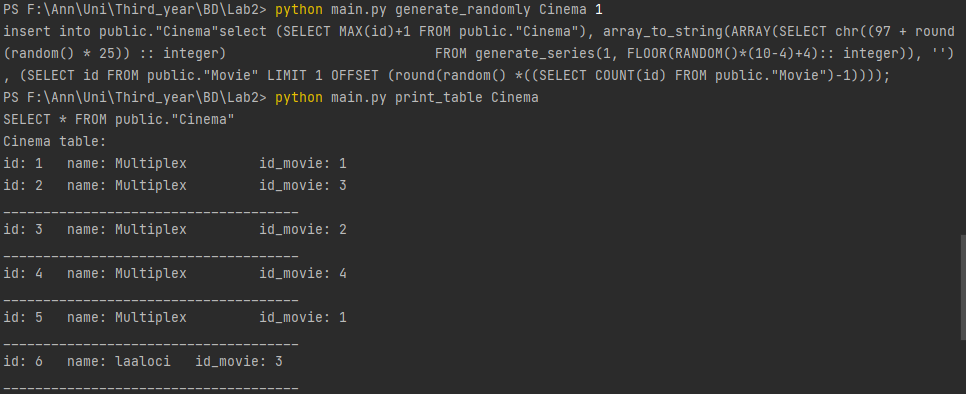
Таблиця Showtime після спроби зміни запису 8:



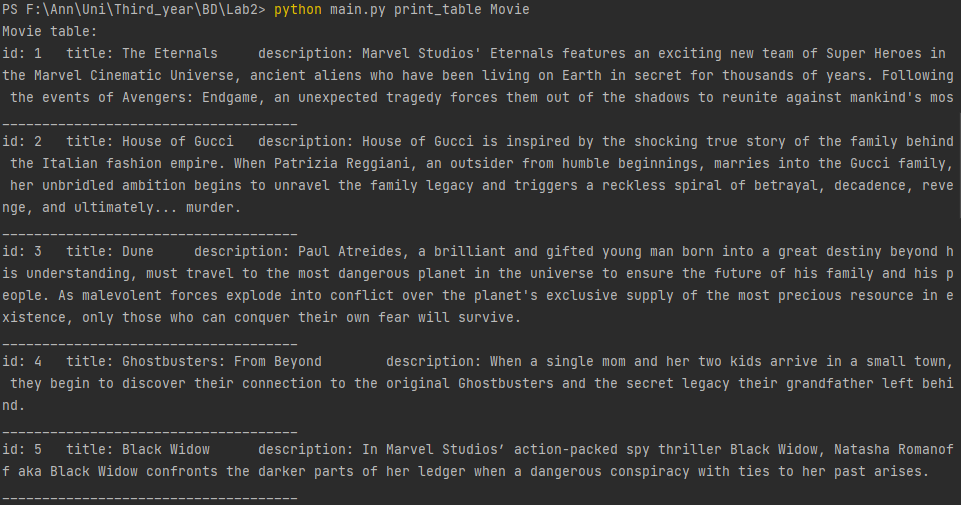
**Завдання 2** Рандомізація даних: Cinema до змін



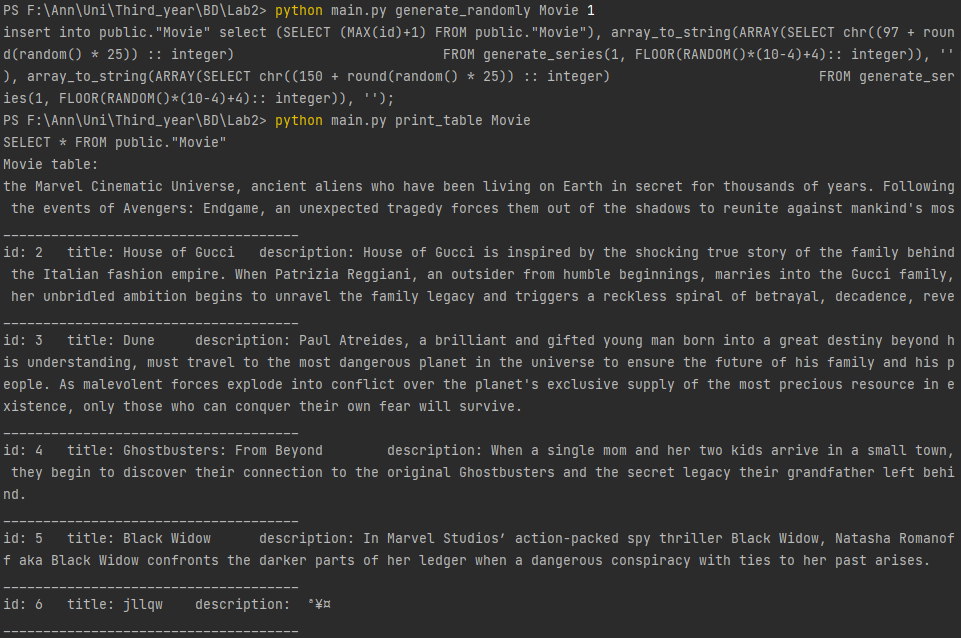
Cinema після рандомізації даних



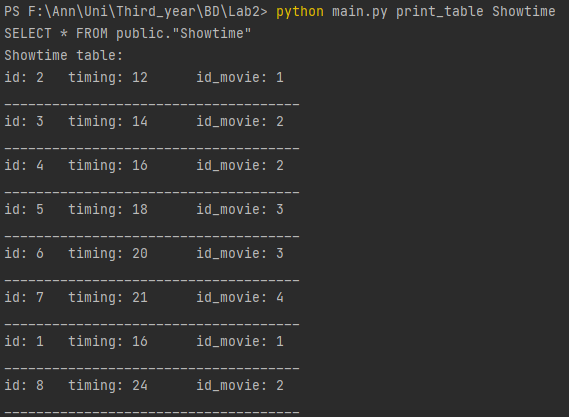
Movie до змін



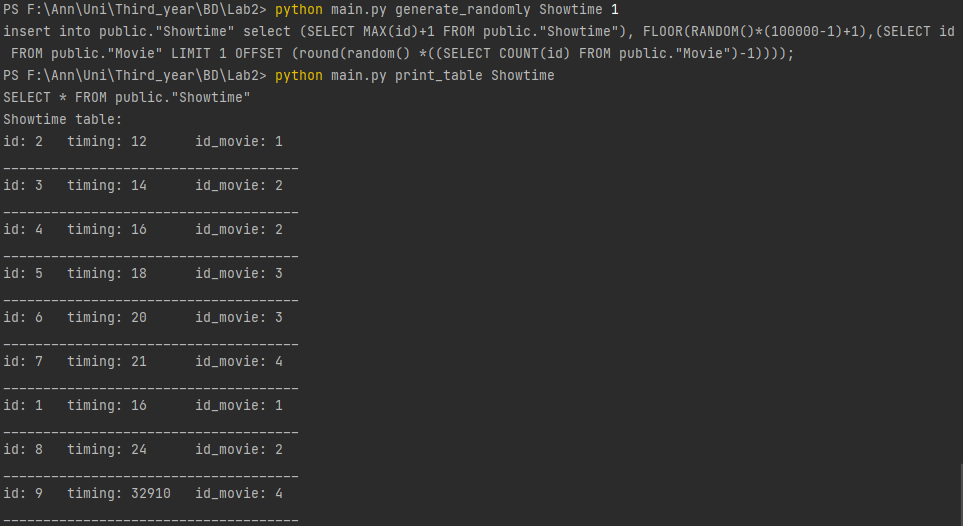
Movie після рандомізації даних



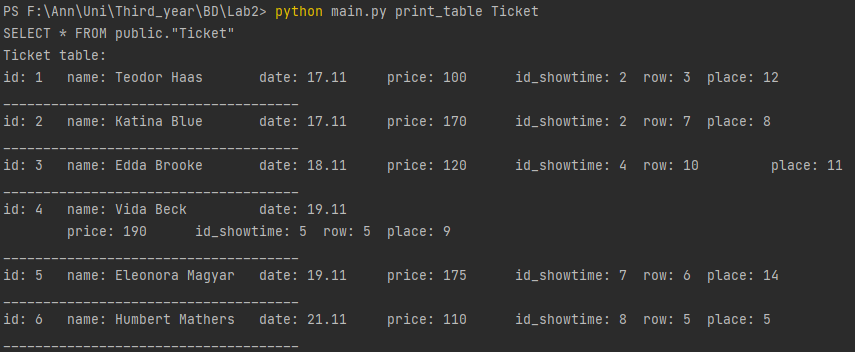
Showtime до змін



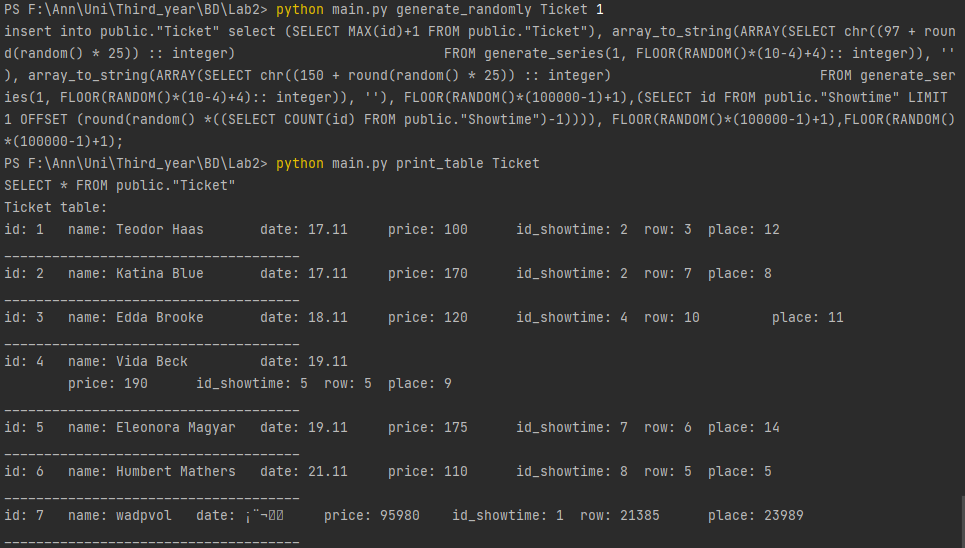
Showtime після рандомізації даних



Ticket до змін



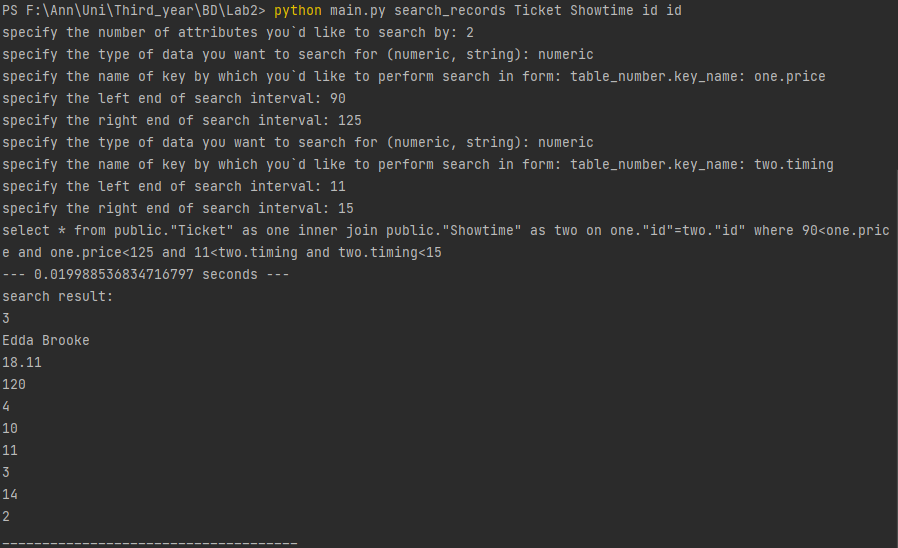
Ticket після рандомізації даних

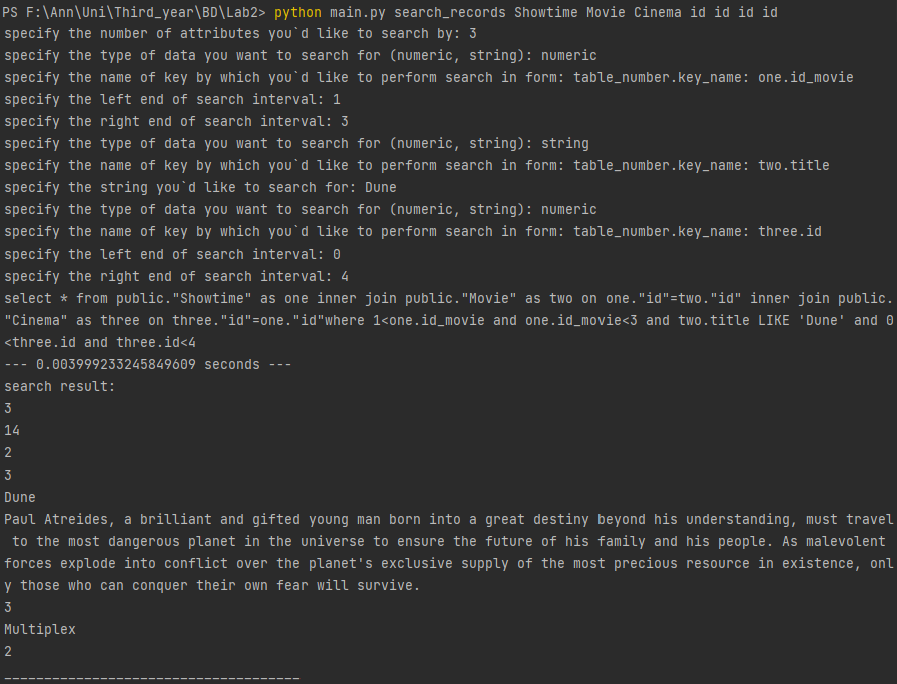


# Завдання 3

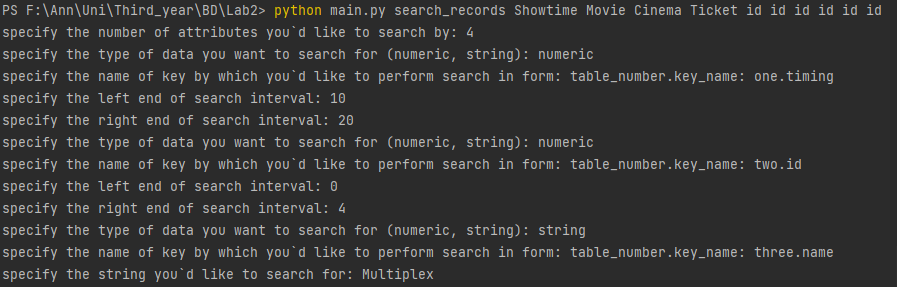
Пошук:

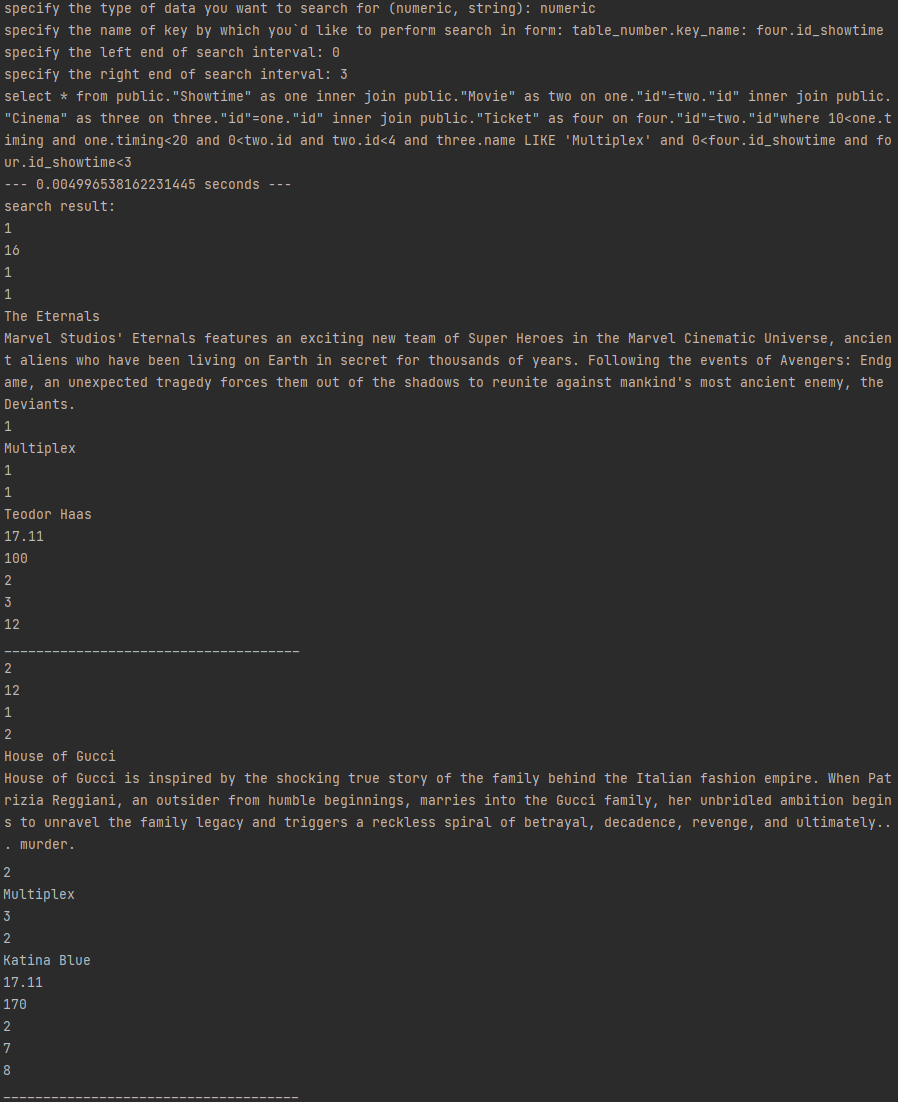
Пошук за двома таблицями



Пошук за трьома таблицями

Пошук за чотирма таблицями





# Завдання 4

Код програми:

Model.py

import psycopg2 as ps

class Model:

def init (self):

self.conn = None try:

self.conn = ps.connect( dbname="Cinema\_1", user='postgres', password="12345", host='127.0.0.1',

port="5432",

)

except(Exception, ps.DatabaseError) as error:

print("[INFO] 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\_movie(self) -> None:

return self.get(f"SELECT \* FROM public.\"Movie\"")

def print\_showtime(self) -> None:

return self.get(f"SELECT \* FROM public.\"Showtime\"")

def print\_ticket(self) -> None:

return self.get(f"SELECT \* FROM public.\"Ticket\"")

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\_cinema(self, key\_value: int, name: str, id\_movie: int) -> None:

self.request(f"UPDATE public.\"Cinema\" SET name=\'{name}\', id\_movie=\'{id\_movie}\' WHERE id={key\_value};")

def update\_data\_movie(self, key\_value: int, title: str, description: str) -

> None:

self.request(f"UPDATE public.\"Movie\" SET title=\'{title}\', description=\'{description}\' "

f"WHERE id={key\_value};")

def update\_data\_showtime(self, key\_value: int, timing: int, id\_movie: int)

-> None:

self.request(f"UPDATE public.\"Showtime\" SET timing=\'{timing}\', id\_movie=\'{id\_movie}\' WHERE id={key\_value};")

def update\_data\_ticket(self, key\_value: int, name: str, date: str, price: int, id\_showtime: int, row: int,

place: int) -> None: self.request(f"UPDATE public.\"Ticket\" SET name=\'{name}\',

date=\'{date}\', price=\'{price}\', "

f"id\_showtime=\'{id\_showtime}\', row=\'{row}\', place=\'{place}\' WHERE id={key\_value};")

def insert\_data\_cinema(self, key\_value: int, name: str, id\_movie: int) -> None:

self.request(f"insert into public.\"Cinema\" (id, name, id\_movie) " f"VALUES ({key\_value}, \'{name}\', \'{id\_movie}\');")

def insert\_data\_movie(self, key\_value: int, title: str, description: str) -

> None:

self.request(f"insert into public.\"Movie\" (id, title, description) " f"VALUES ({key\_value}, \'{title}\', \'{description}\');")

def insert\_data\_showtime(self, key\_value: int, timing: int, id\_movie: int)

-> None:

self.request(f"insert into public.\"Showtime\" (id, timing, id\_movie) " f"VALUES ({key\_value}, \'{timing}\', \'{id\_movie}\');")

def insert\_data\_ticket(self, key\_value: int, name: str, date: str, price: int, id\_showtime: int, row: int,

place: int) -> None:

self.request(f"insert into public.\"Ticket\" (id, name, date, price, id\_showtime, row, place) "

f"VALUES ({key\_value}, \'{name}\', \'{date}\',

\'{price}\', \'{id\_showtime}\', \'{row}\', \'{place}\');")

def cinema\_data\_generator(self, times: int) -> None: for i in range(times):

self.request("insert into public.\"Cinema\""

"select (SELECT MAX(id)+1 FROM public.\"Cinema\"), " "array\_to\_string(ARRAY(SELECT chr((97 + round(random()

\* 25)) :: integer) \ integer)), ''), "

FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4)::

"(SELECT id FROM public.\"Movie\" LIMIT 1 OFFSET " "(round(random() \*((SELECT COUNT(id) FROM

public.\"Movie\")-1))));")

def movie\_data\_generator(self, times: int) -> None: for i in range(times):

self.request("insert into public.\"Movie\" "

"select (SELECT (MAX(id)+1) FROM public.\"Movie\"), " "array\_to\_string(ARRAY(SELECT chr((97 + round(random()

\* 25)) :: integer) \ integer)), ''), "

\* 25)) :: integer) \ integer)), ''); ")

FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4):: "array\_to\_string(ARRAY(SELECT chr((97 + round(random() FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4)::

def showtime\_data\_generator(self, times: int) -> None: for i in range(times):

self.request("insert into public.\"Showtime\" "

"select (SELECT MAX(id)+1 FROM public.\"Showtime\"), " "FLOOR(RANDOM()\*(100000-1)+1),"

"(SELECT id FROM public.\"Movie\" LIMIT 1 OFFSET " "(round(random() \*((SELECT COUNT(id) FROM

public.\"Movie\")-1))));")

def ticket\_data\_generator(self, times: int) -> None: for i in range(times):

self.request("insert into public.\"Ticket\" "

"select (SELECT MAX(id)+1 FROM public.\"Ticket\"), " "array\_to\_string(ARRAY(SELECT chr((97 + round(random()

\* 25)) :: integer) \ integer)), ''), "

FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4)::

"array\_to\_string(ARRAY(SELECT chr((150 + round(random() \* 25)) :: integer) \

FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4)::

integer)), ''), "

"FLOOR(RANDOM()\*(100000-1)+1),"

"(SELECT id FROM public.\"Showtime\" LIMIT 1 OFFSET " "(round(random() \*((SELECT COUNT(id) FROM

public.\"Showtime\")-1)))), "

"FLOOR(RANDOM()\*(100000-1)+1),"

"FLOOR(RANDOM()\*(100000-1)+1);")

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,

table13\_key,

table1\_key, table2\_key, table3\_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}")

def search\_data\_all\_tables(self, table1\_name: str, table2\_name: str, table3\_name: str, table4\_name: str,

table1\_key, table2\_key, table3\_key, table13\_key, table4\_key, table24\_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}\" inner join public.\"{table4\_name}\" as four "

f"on four.\"{table4\_key}\"=two.\"{table24\_key}\"" f"where {search}")

Програмна частина model.py слугує головною частиною, оскільки має доступ до бази даних. Бібліотека Python – psycopg2 надає дану можливість.

Короткий опис функцій model.py

request : робить запит до БД, у разі успішного виконання повертає True, у разі невдачі – False.

get : усі дані що було взято з запитів SELECT, у разі невдачі – False.

get\_el : повертає тільки перший запис, у разі невдачі – False. count : повертає кількість усіх записів таблиці.

find : відповідно заданій умові шукає записи у таблиці і повертає їх кількість, у разі невдачі – False.

max, min : повертають максимальне і мінімальне значення зазначеного ключа у таблиці.

print\_(cinema/movie/showtime/ticket): виводять на екран відповідні таблиці. delete\_data : видаляє відповідний запис у заданій таблиці. update\_data\_(cinema/movie/showtime/ticket) : оновлює відповідний запис у

заданій таблиці.

insert\_data\_(cinema/movie/showtime/ticket) : додає відповідний запис у задану таблицю.

(cinema/movie/showtime/ticket)\_data\_generator : додає рандомізований запис у задану таблицю.

search\_data\_(two/three/all)\_tables : здійснює пошук по двум/трьом/чотирьом заданим таблицям.