

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 ІII курсу

групи КВ-01

Соліна Т. А.

Перевірив:

Павловський В. І.

Київ – 2022

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. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі. 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).

Деталізоване завдання:

Забезпечити можливість уведення/редагування/вилучення даних у таблицях бази даних з можливістю контролю відповідності типів даних атрибутів таблиць (рядків, чисел, дати/часу). Для контролю пропонується два варіанти: контроль при введенні (валідація даних) та перехоплення помилок (try..except) від сервера PostgreSQL при виконанні відповідної команди SQL. Особливу увагу варто звернути на дані таблиць, що мають зв’язок 1:N. При цьому з боку батьківської таблиці необхідно контролювати вилучення рядків за умови наявності даних у підлеглій таблиці. З точки зору підлеглої таблиці варто контролювати наявність відповідного рядка у батьківській таблиці при виконанні внесення нових даних. Унеможливити виведення програмою системних помилок на екрані шляхом їх перехоплення і адекватної обробки. Внесення даних виконується користувачем у консольному вікні програми.

Забезпечити можливість автоматичної генерації великої кількості даних у таблицях за допомогою вбудованих у PostgreSQL функцій роботи з псевдовипадковими числами. Дані мають бути згенерованими не мовою програмування, а відповідним SQL-запитом!

Для реалізації пошуку необхідно підготувати 3 запити, що включають дані з декількох таблиць і фільтрують рядки за 3-4 атрибутами цих таблиць. Забезпечити можливість уведення конкретних значень констант для фільтрації з клавіатури користувачем. Крім того, після виведення даних необхідно вивести час виконання запиту у мілісекундах. Перевірити швидкодію роботи запитів на попередньо згенерованих даних.

Програмний код організувати згідно шаблону Model-View-Controller(MVC). Приклад організації коду згідно шаблону доступний за даним посиланням. При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати лише мову SQL.

**Логічна модель предметної області «Кінотеатр»**

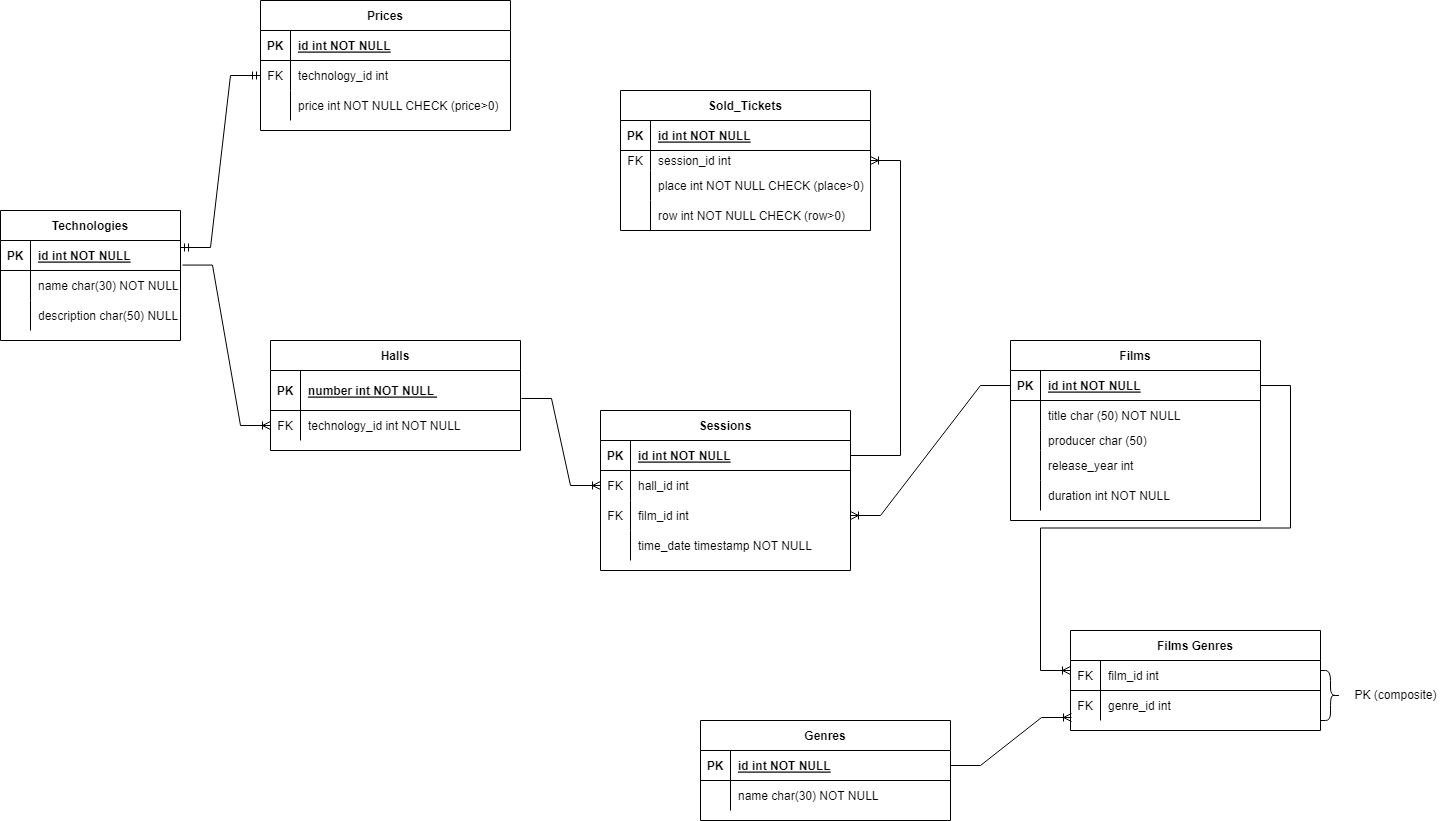


Рисунок 1. Схема бази даних, побудовано у draw.io.

**Середовище та компоненти розробки**

Для розробки використовувалась мова програмування Python, середовище розробки PyCharm, а також стороння бібліотека, що надає API для доступу до PostgreSQL – psycopg2.

**Шаблон проектування**

MVC - Шаблон проектування, який використаний у програмі.

Model – представляє клас, що описує логіку використовуваних даних. Згідно компоненту моделі, у моїй програмі відповідають всі компоненти які знаходять у файлі model.py.

View – в нашому випадку консольний інтерфейс з яким буде взаємодіяти наш користувач. Згідно компоненту представлення, то їй відповідають такі компоненти, згідно яким користувач бачить необхідні дані, що є представленням даних у вигляді консольного інтерфейсу.

Controller – представляє клас, що забезпечує зв'язок між користувачем і системою, поданням і сховищем даних. Він отримує введені користувачем дані і обробляє їх. І в залежності від результатів обробки відправляє користувачеві певний висновок.

**Структура програми та її опис**

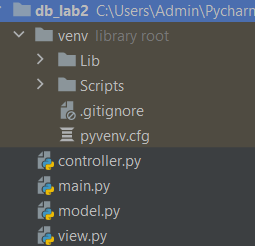


Рисунок 2. Структура програми.

Програма умовно поділена на 4 модулі: main.py, model.py, view.py, controller.py. Класи, як видно з їх назв, повністю відповідають використаному патерну MVC.

У файлі model.py описано клас Model, що займається регулювання підключення до бази даних, та виконанням низькорівневих запитів до неї.

У файлах view.py описано клас View, що виводить результати виконання тієї чи іншої дії.

У файлі controller.py описано клас Controller, які оброблюються завдяки взаємодії з користувачем, запит бажаної дії, виконання пошуку, тощо.

У файлі main.py створюється екземпляр класу Controller, якому користувач передає вказівки щодо виконання програми.

**Структура меню програми**

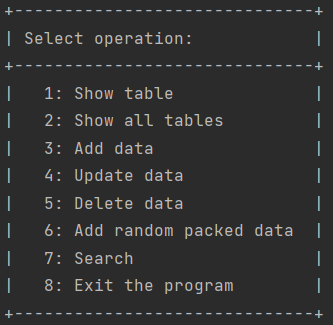


Рисунок 3. Меню для вибору операції над БД.

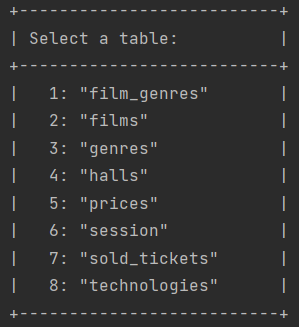


Рисунок 4. Меню для вибору таблиці, над якою проводитиметься обрана операція.

**Меню операцій складається з восьми пунктів (Рисунок 3)**

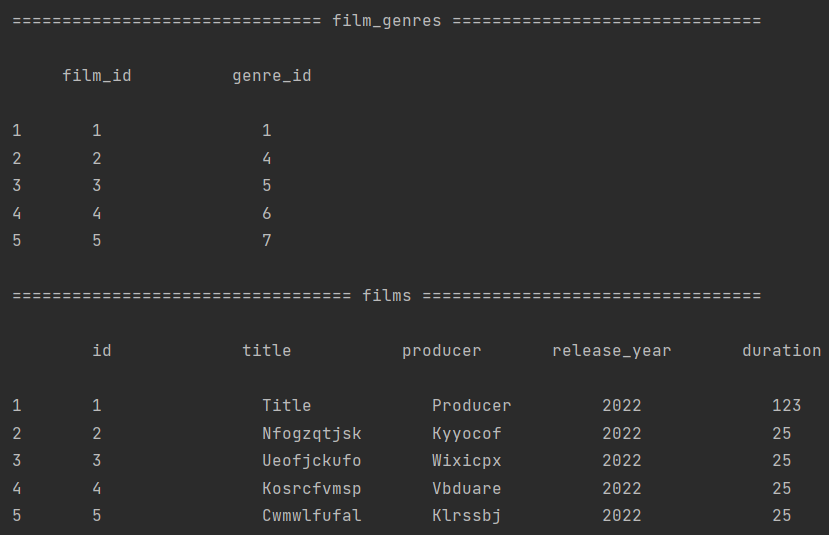
1. Форматоване виведення таблиці у файл table.txt.
2. Форматоване виведення всіх таблиць у файл table.txt.
3. Додавання нового рядку даних до таблиці.
4. Оновлення рядку даних у таблиці.
5. Видалення рядку даних з таблиці.
6. Додавання нових випадкових даних до кожної таблиці БД.
7. Пошук усіх рядків з усіх таблиць, які мають певні параметри, введені користувачем.
8. Завершення програми.

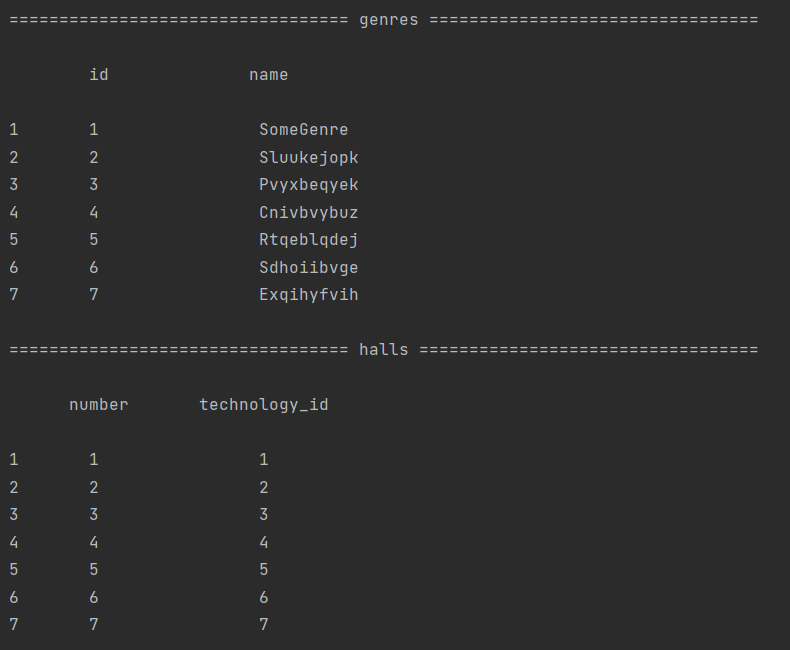
**Меню таблиць складається з 8 пунктів (Рисунок 4)**

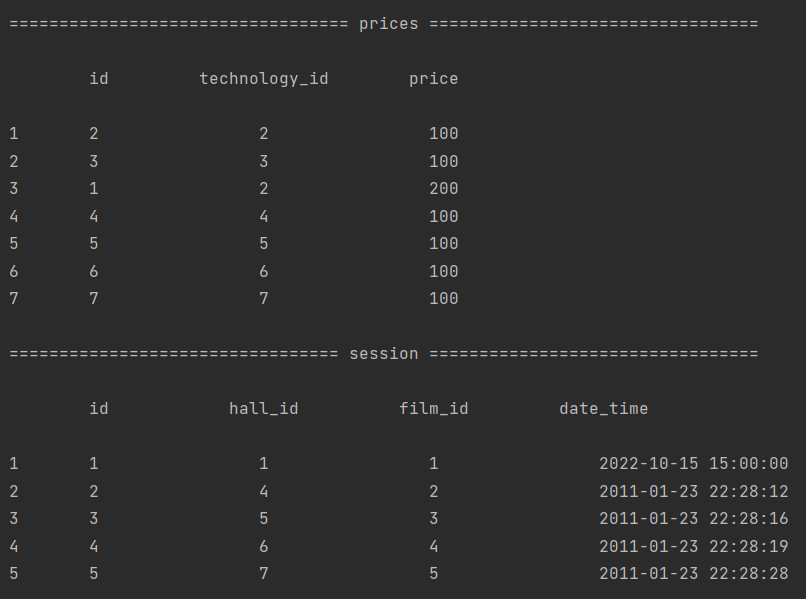
1. Виконати обрану операцію над таблицею “film\_genres”.
2. Виконати обрану операцію над таблицею “films”.
3. Виконати обрану операцію над таблицею “genres”.
4. Виконати обрану операцію над таблицею “halls”.
5. Виконати обрану операцію над таблицею “prices”.
6. Виконати обрану операцію над таблицею “session”
7. Виконати обрану операцію над таблицею “sold\_tickets”
8. Виконати обрану операцію над таблицею “technologies”

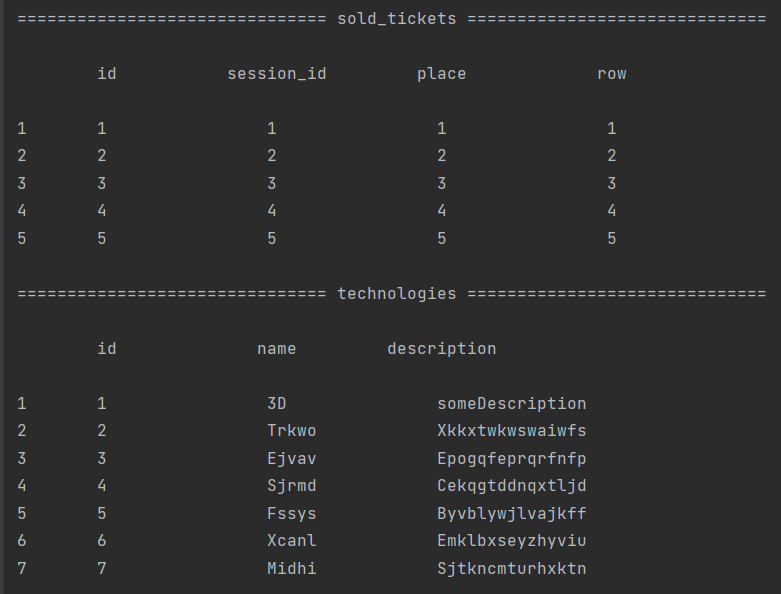
**Результати та виконання операцій**

**Результат операції Show (для всіх таблиць)**

****

****

****

****

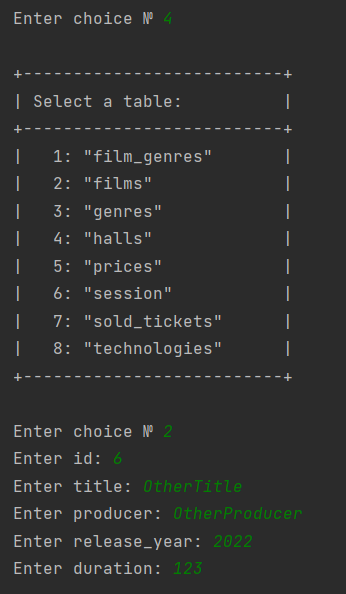
**Операція Add для таблиці “films”**

****

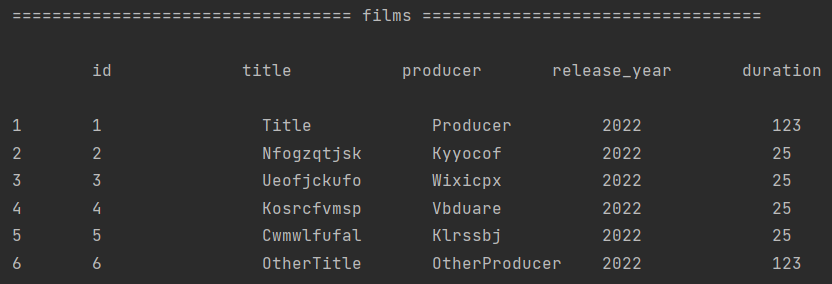
**Результат**

****

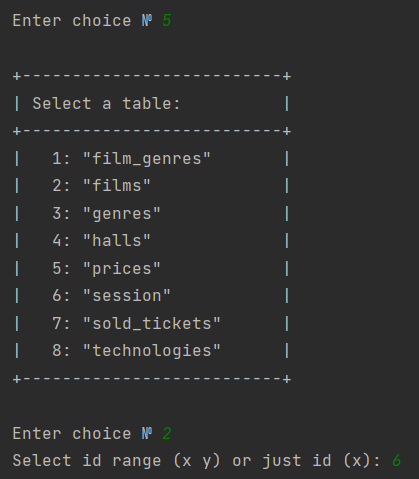
**Оновлення даних у таблиці “films”**

****

**Результат**

****

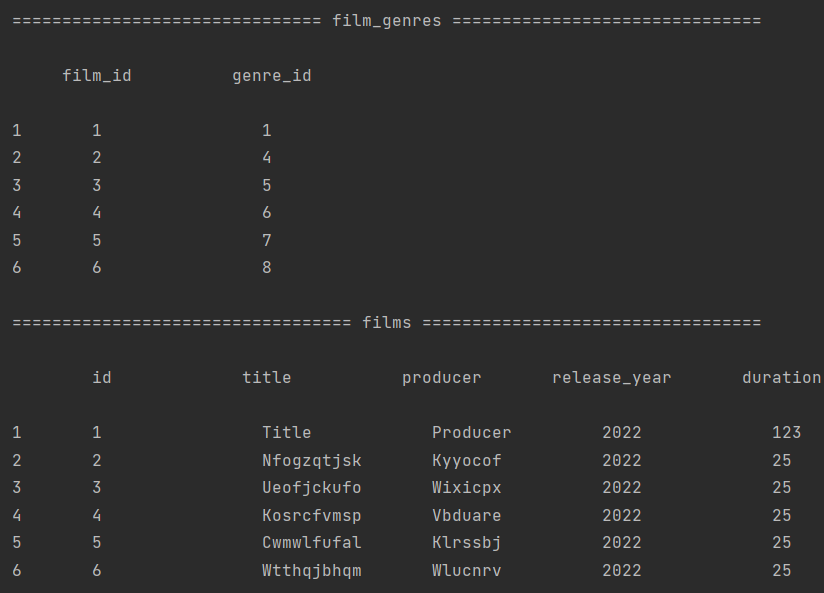
**Видалення фільму з id = 6**

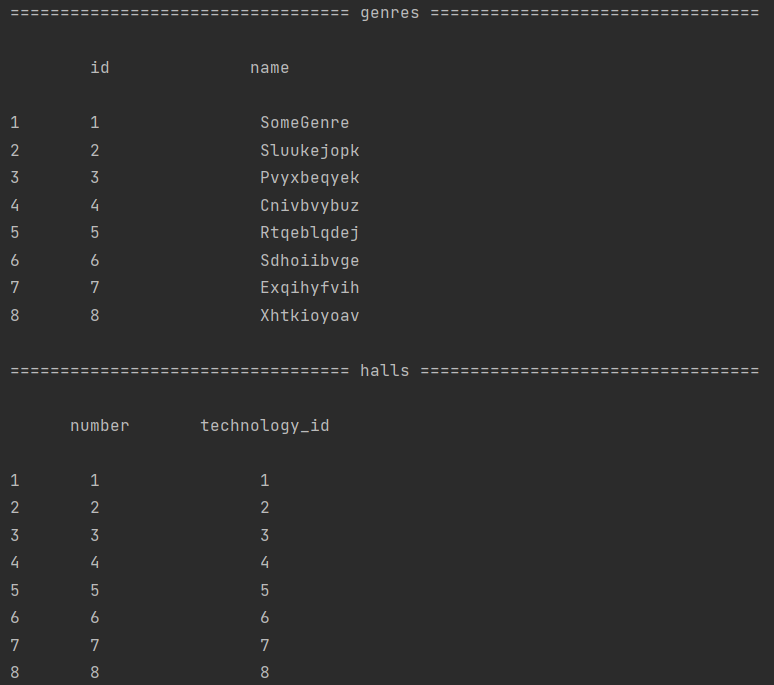
****

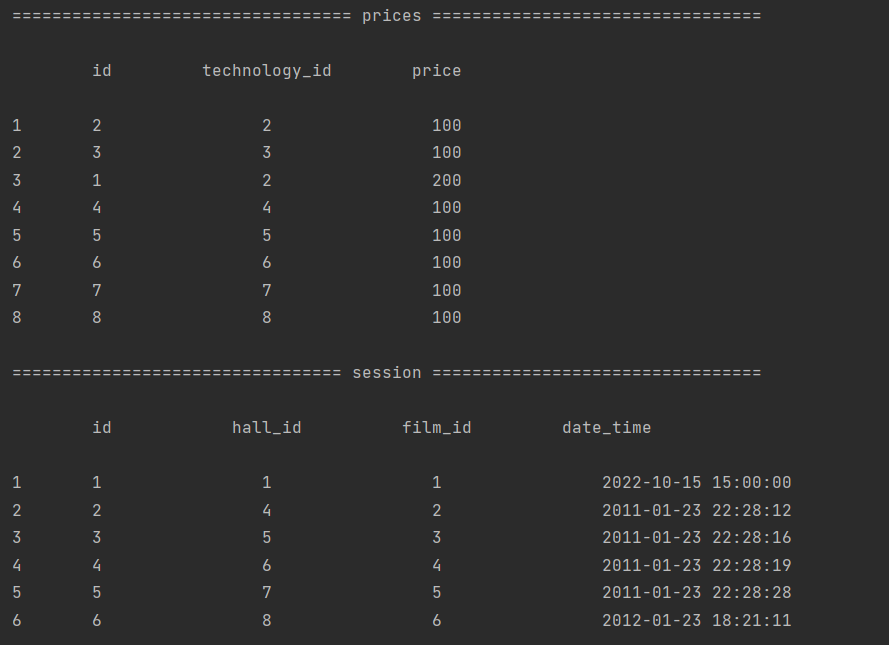
**Результат**

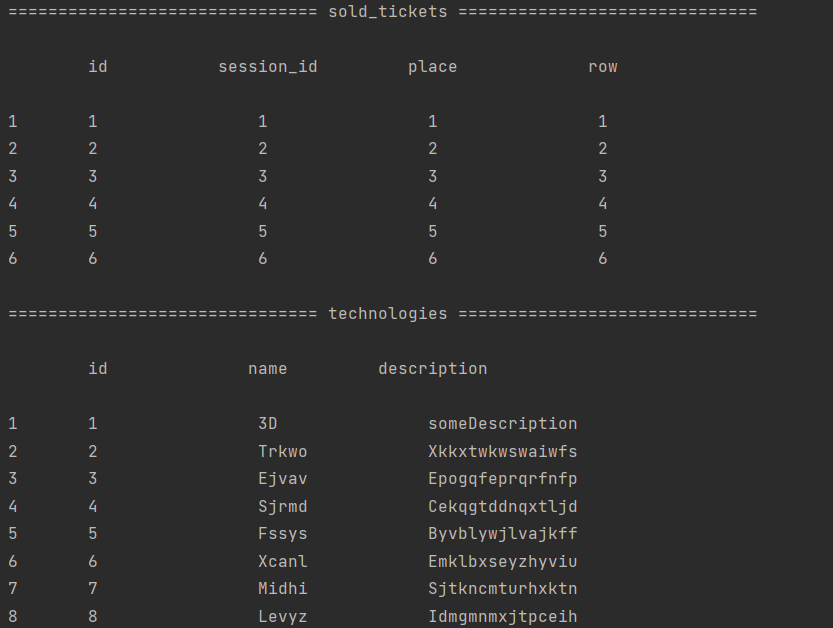
****

**Додавання випадкових даних до кожної таблиці (останні рядки)**

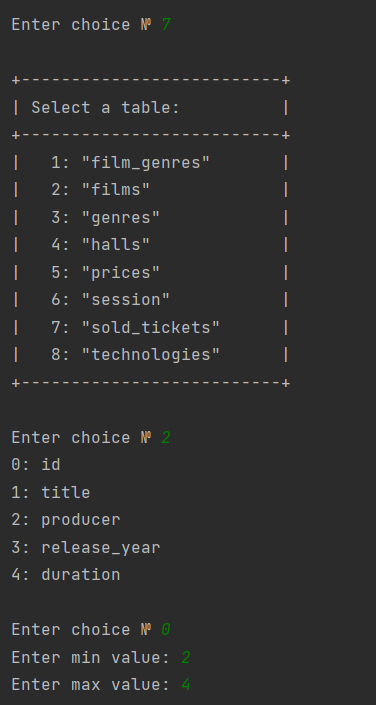
****

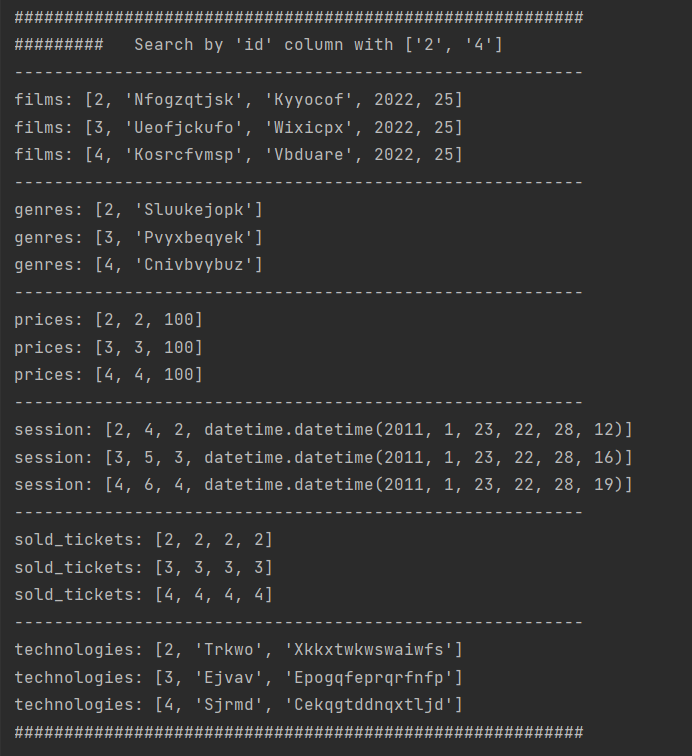
****

****

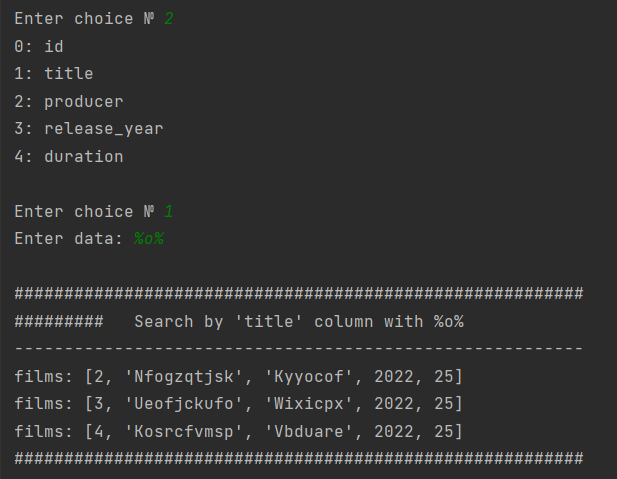
****

**Пошук та результати за параметром id = [2, 4]**

****

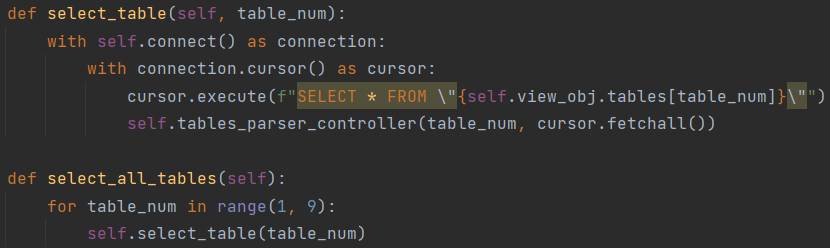
****

**Пошук за параметром %o%**

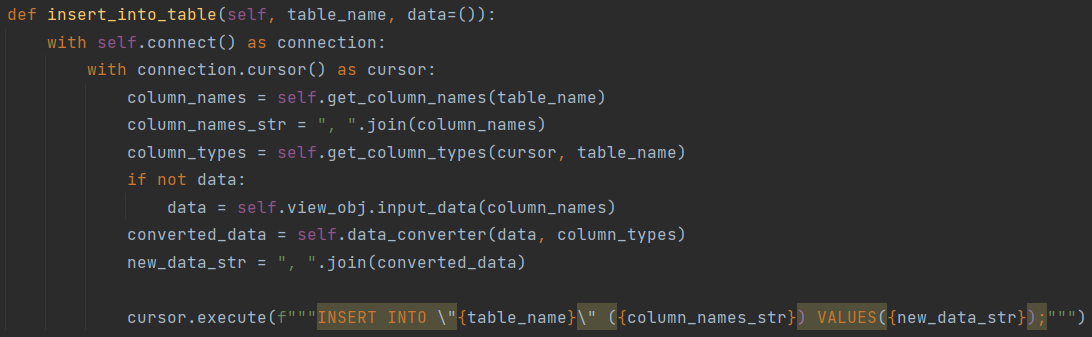
****

**Ілюстрації програмного коду модуля “model.py”**

**Вибір даних з таблиць, та передання їх до контролеру парсерів, де буде викликана відповідна функція з модуля view.py та дані будуть виведені у консоль.**

****

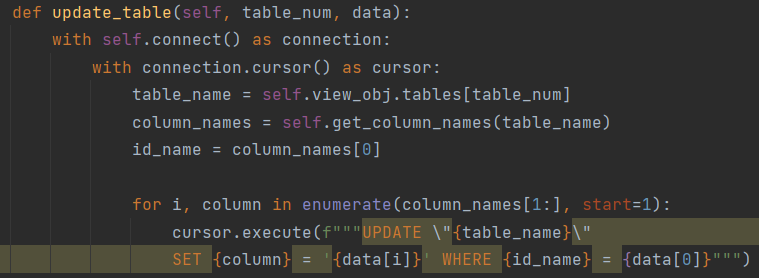
**Функція додавання нових даних у таблицю.**

****

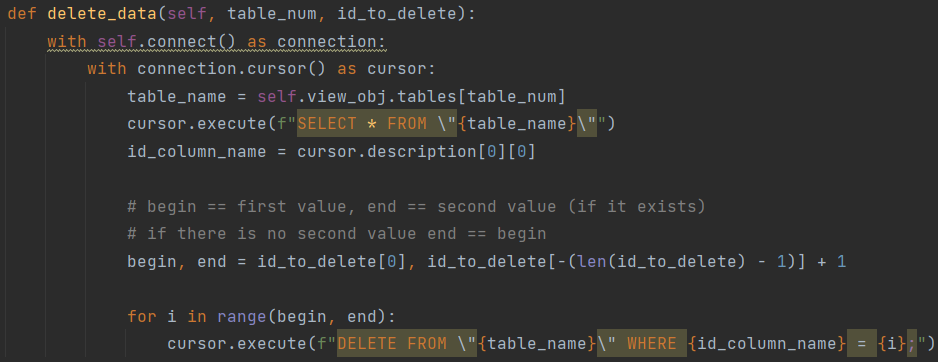
**Вставка пакетних «рандомізованих» даних має таку реалізацію.**

****

**Оновлення даних рядка таблиці.**

****

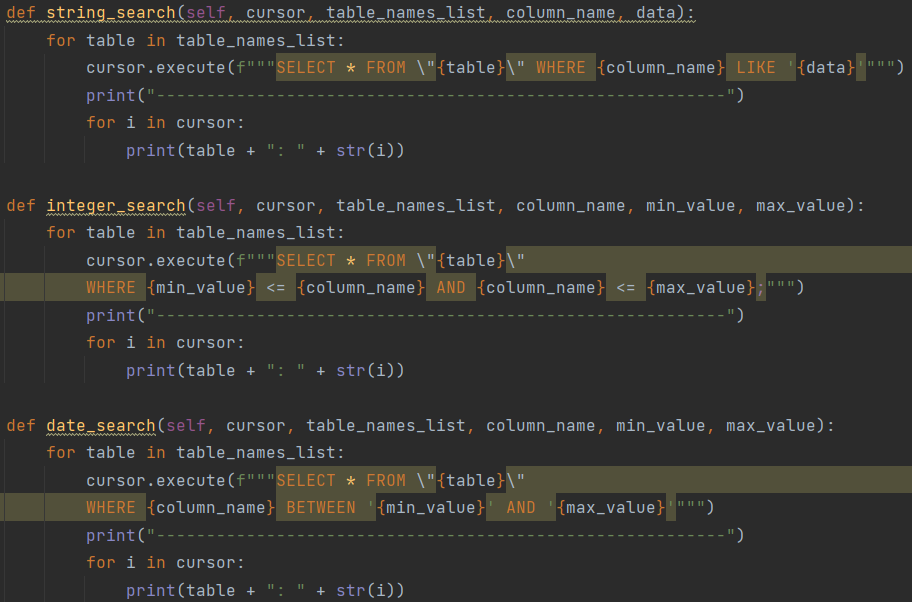
**Видалення даних з таблиці**

****

При зчитуванні id рядку для видалення, було реалізовано можливість видалення одразу великої їх кількості (у діапазоні від Х до У).

При цьому було оброблено ситуацію, коли користувач випадково вводить більше 2 значень, їх не враховуємо, беруться тільки перші значення. Проте, можна видалити й тільки один рядок, тоді значення begin == end.

**Пошук для рядкових, числових та часових форматів.**



**Код програми**

**main.py**

from controller import Controller  
  
  
def main():  
 obj = Controller()  
 obj.menu()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

**model.py**

from view import View  
  
from datetime import datetime  
  
import psycopg2  
import psycopg2.extras  
from psycopg2.extras import DictCursor  
  
  
class Model:  
 def \_\_init\_\_(self):  
 self.view\_obj = View()  
  
 def connect(self):  
 return psycopg2.connect(dbname='cinema', user='postgres', password='Sqlpr0v0d0k', host='localhost')  
  
 def select\_table(self, table\_num):  
 with self.connect() as connection:  
 with connection.cursor() as cursor:  
 cursor.execute(f"SELECT \* FROM \"{self.view\_obj.tables[table\_num]}\"")  
 self.tables\_parser\_controller(table\_num, cursor.fetchall())  
  
 def select\_all\_tables(self):  
 for table\_num in range(1, 9):  
 self.select\_table(table\_num)  
  
 def tables\_parser\_controller(self, table\_num, data):  
 if table\_num == 1:  
 self.view\_obj.film\_genres\_parser(data)  
 elif table\_num == 2:  
 self.view\_obj.films\_parser(data)  
 elif table\_num == 3:  
 self.view\_obj.genres\_parser(data)  
 elif table\_num == 4:  
 self.view\_obj.halls\_parser(data)  
 elif table\_num == 5:  
 self.view\_obj.prices\_parser(data)  
 elif table\_num == 6:  
 self.view\_obj.session\_parser(data)  
 elif table\_num == 7:  
 self.view\_obj.sold\_tickets\_parser(data)  
 elif table\_num == 8:  
 self.view\_obj.technologies\_parser(data)  
  
 def get\_column\_names(self, table\_name):  
 with self.connect() as connection:  
 with connection.cursor() as cursor:  
 cursor.execute(f"SELECT \* FROM \"{table\_name}\"")  
 return [item[0] for item in cursor.description]  
  
 def get\_column\_types(self, cursor, table\_name):  
 cursor.execute(f"""SELECT data\_type FROM information\_schema.columns  
 WHERE table\_schema = 'public' AND table\_name = '{table\_name}'""")  
 return [i[0] for i in cursor.fetchall()]  
  
 def data\_converter(self, data, column\_types):  
 return [value  
 if \_type == 'integer' else "'" + str(value) + "'"  
 for value, \_type in zip(data, column\_types)]  
  
 def insert\_into\_table(self, table\_name, data=()):  
 with self.connect() as connection:  
 with connection.cursor() as cursor:  
 column\_names = self.get\_column\_names(table\_name)  
 column\_names\_str = ", ".join(column\_names)  
 column\_types = self.get\_column\_types(cursor, table\_name)  
 if not data:  
 data = self.view\_obj.input\_data(column\_names)  
 converted\_data = self.data\_converter(data, column\_types)  
 new\_data\_str = ", ".join(converted\_data)  
  
 cursor.execute(f"""INSERT INTO \"{table\_name}\" ({column\_names\_str}) VALUES({new\_data\_str});""")  
  
 def get\_last\_entry\_from\_column(self, table\_name, column\_name, cursor):  
 cursor.execute(f"""SELECT {column\_name} FROM \"{table\_name}\"  
 ORDER BY {column\_name} DESC LIMIT 1""")  
 return cursor.fetchone()[0]  
  
 def generate\_random\_data(self, cursor, length):  
 uppercase\_letter = "chr(ascii('A') + (random() \* 25)::int)"  
 lowercase\_letter = "chr(ascii('a') + (random() \* 25)::int)"  
 cursor.execute(f"""SELECT ({uppercase\_letter}{(" || " + lowercase\_letter) \* (length - 1)})""")  
 return cursor.fetchone()[0]  
  
 def insert\_random\_data\_packet(self):  
 with self.connect() as connection:  
 with connection.cursor() as cursor:  
 film\_id = self.get\_last\_entry\_from\_column('films', 'id', cursor) + 1  
 title = self.generate\_random\_data(cursor, 10)  
 producer = self.generate\_random\_data(cursor, 7)  
 self.insert\_into\_table('films', (str(film\_id),   
 title,   
 producer,   
 '2022',   
 '25'))  
  
 genre\_id = self.get\_last\_entry\_from\_column('genres', 'id', cursor) + 1  
 genre\_name = self.generate\_random\_data(cursor, 10)  
 self.insert\_into\_table('genres', (str(genre\_id),   
 genre\_name))  
  
 self.insert\_into\_table('film\_genres', (str(film\_id),   
 str(genre\_id)))  
  
 technology\_id = self.get\_last\_entry\_from\_column('technologies', 'id', cursor) + 1  
 technology\_name = self.generate\_random\_data(cursor, 5)  
 description = self.generate\_random\_data(cursor, 15)  
 self.insert\_into\_table('technologies', (str(technology\_id),   
 technology\_name,   
 description))  
  
 hall\_number = self.get\_last\_entry\_from\_column('halls', 'number', cursor) + 1  
 self.insert\_into\_table('halls', (str(hall\_number),   
 str(technology\_id)))  
  
 price\_id = self.get\_last\_entry\_from\_column('prices', 'id', cursor) + 1  
 self.insert\_into\_table('prices', (str(price\_id),   
 str(technology\_id),   
 '100'))  
  
 session\_id = self.get\_last\_entry\_from\_column('session', 'id', cursor) + 1  
 date\_time = datetime.now().strftime("%y-%m-%d %H:%M:%S")  
 self.insert\_into\_table('session', (str(session\_id),   
 str(hall\_number),   
 str(film\_id),   
 date\_time))  
  
 ticket\_id = self.get\_last\_entry\_from\_column('sold\_tickets', 'id', cursor) + 1  
 self.insert\_into\_table('sold\_tickets', (str(ticket\_id),   
 str(session\_id),   
 str(ticket\_id),   
 str(ticket\_id)))  
  
 def update\_table(self, table\_num, data):  
 with self.connect() as connection:  
 with connection.cursor() as cursor:  
 table\_name = self.view\_obj.tables[table\_num]  
 column\_names = self.get\_column\_names(table\_name)  
 id\_name = column\_names[0]  
  
 for i, column in enumerate(column\_names[1:], start=1):  
 cursor.execute(f"""UPDATE \"{table\_name}\"  
 SET {column} = '{data[i]}' WHERE {id\_name} = {data[0]}""")  
  
 def delete\_data(self, table\_num, id\_to\_delete):  
 with self.connect() as connection:  
 with connection.cursor() as cursor:  
 table\_name = self.view\_obj.tables[table\_num]  
 cursor.execute(f"SELECT \* FROM \"{table\_name}\"")  
 id\_column\_name = cursor.description[0][0]  
  
 # begin == first value, end == second value (if it exists)  
 # if there is no second value end == begin  
 begin, end = id\_to\_delete[0], id\_to\_delete[-(len(id\_to\_delete) - 1)] + 1  
  
 for i in range(begin, end):  
 cursor.execute(f"DELETE FROM \"{table\_name}\" WHERE {id\_column\_name} = {i};")  
  
 def get\_column\_names\_to\_view(self, table\_num):  
 with self.connect() as connection:  
 with connection.cursor() as cursor:  
 table\_name = self.view\_obj.tables[table\_num]  
 return self.get\_column\_names(table\_name)  
  
  
 def string\_search(self, cursor, table\_names\_list, column\_name, data):  
 for table in table\_names\_list:  
 cursor.execute(f"""SELECT \* FROM \"{table}\" WHERE {column\_name} LIKE '{data}'""")  
 print("---------------------------------------------------------")  
 for i in cursor:  
 print(table + ": " + str(i))  
  
 def integer\_search(self, cursor, table\_names\_list, column\_name, min\_value, max\_value):  
 for table in table\_names\_list:  
 cursor.execute(f"""SELECT \* FROM \"{table}\"  
 WHERE {min\_value} <= {column\_name} AND {column\_name} <= {max\_value};""")  
 print("---------------------------------------------------------")  
 for i in cursor:  
 print(table + ": " + str(i))  
  
 def date\_search(self, cursor, table\_names\_list, column\_name, min\_value, max\_value):  
 for table in table\_names\_list:  
 cursor.execute(f"""SELECT \* FROM \"{table}\"  
 WHERE {column\_name} BETWEEN '{min\_value}' AND '{max\_value}'""")  
 print("---------------------------------------------------------")  
 for i in cursor:  
 print(table + ": " + str(i))  
  
 def search\_controller(self, cursor, table\_names\_list, column\_name, data\_type, data):  
 print("\n#########################################################")  
 print(f"#########\tSearch by '{column\_name}' column with {data}")  
 if data\_type == 'integer':  
 self.integer\_search(cursor, table\_names\_list, column\_name, \*data)  
 elif data\_type == 'character varying':  
 self.string\_search(cursor, table\_names\_list, column\_name, data)  
 elif data\_type == 'timestamp without time zone':  
 self.date\_search(cursor, table\_names\_list, column\_name, \*data)  
 print("#########################################################\n\n")  
  
  
 def search\_into\_table(self, table\_num, search\_param):  
 with self.connect() as connection:  
 with connection.cursor(cursor\_factory=DictCursor) as cursor:  
 table\_name = self.view\_obj.tables[table\_num]  
 column\_names = self.get\_column\_names(table\_name)  
 column = column\_names[search\_param]  
  
 cursor.execute(f"""SELECT TABLE\_NAME FROM INFORMATION\_SCHEMA.COLUMNS   
 WHERE COLUMN\_NAME = '{column}'""")  
 table\_names\_list = [item[0] for item in cursor.fetchall()]  
  
 cursor.execute(f"""SELECT \* FROM INFORMATION\_SCHEMA.COLUMNS  
 WHERE TABLE\_NAME = '{table\_name}' AND COLUMN\_NAME = '{column}'""")  
 data\_type = cursor.fetchone()['data\_type']  
  
 data\_to\_search = self.view\_obj.input\_data\_to\_search(data\_type)  
 self.search\_controller(cursor, table\_names\_list, column, data\_type, data\_to\_search)

**view.py**

class View:  
 \_\_operations\_menu = """  
+------------------------------+  
| Select operation: |  
+------------------------------+  
| 1: Show table |  
| 2: Show all tables |  
| 3: Add data |  
| 4: Update data |  
| 5: Delete data |  
| 6: Add random packed data |  
| 7: Search |  
| 8: Exit the program |  
+------------------------------+  
"""  
  
 \_\_tables\_menu = """  
+--------------------------+  
| Select a table: |  
+--------------------------+  
| 1: "film\_genres" |  
| 2: "films" |  
| 3: "genres" |  
| 4: "halls" |  
| 5: "prices" |  
| 6: "session" |  
| 7: "sold\_tickets" |  
| 8: "technologies" |  
+--------------------------+  
"""  
  
 def \_\_init\_\_(self):  
 self.tables = {  
 1: 'film\_genres',  
 2: 'films',  
 3: 'genres',  
 4: 'halls',  
 5: 'prices',  
 6: 'session',  
 7: 'sold\_tickets',  
 8: 'technologies',  
 # 9: 'Exit the program',  
 }  
  
 def input\_choice(self, func, end):  
 choice = int(input("Enter choice № "))  
 if end < choice < 1:  
 print(f"Enter the number from 1 to {end}")  
 return func()  
 return choice  
  
 def operations\_menu(self):  
 print(self.\_\_operations\_menu)  
 return self.input\_choice(self.operations\_menu, 7)  
  
 def tables\_menu(self):  
 print(self.\_\_tables\_menu)  
 return self.input\_choice(self.operations\_menu, 8)  
  
 @staticmethod  
 def print\_table\_data(data, indents):  
 for i, item in enumerate(data, start=1):  
 print(str(i).ljust(8, " "), end="")  
 for j, indent in enumerate(indents):  
 print(str(item[j]).ljust(indent, " "), end="")  
 print()  
  
 def film\_genres\_parser(self, data):  
 print("\n" + " film\_genres ".center(75, "=") + "\n")  
 print("film\_id".center(17, " ") + "genre\_id".center(17, " ") + "\n")  
  
 indents = (17, 0)  
 self.print\_table\_data(data, indents)  
  
 def films\_parser(self, data):  
 print("\n" + " films ".center(75, "=") + "\n")  
 print("id".center(17, " ")  
 + "title".center(17, " ")  
 + "producer".center(17, " ")  
 + "release\_year".center(17, " ")  
 + "duration".center(17, " ") + "\n")  
  
 indents = (17, 17, 17, 17, 0)  
 self.print\_table\_data(data, indents)  
  
 def genres\_parser(self, data):  
 print("\n" + " genres ".center(75, "=") + "\n")  
 print("id".center(17, " ") + "name".center(17, " ") + "\n")  
  
 indents = (17, 0)  
 self.print\_table\_data(data, indents)  
  
 def halls\_parser(self, data):  
 print("\n" + " halls ".center(75, "=") + "\n")  
 print("number".center(17, " ") + "technology\_id".center(17, " ") + "\n")  
  
 indents = (17, 0)  
 self.print\_table\_data(data, indents)  
  
 def prices\_parser(self, data):  
 print("\n" + " prices ".center(75, "=") + "\n")  
 print("id".center(17, " ")  
 + "technology\_id".center(17, " ")  
 + "price".center(17, " ") + "\n")  
  
 indents = (17, 17, 0)  
 self.print\_table\_data(data, indents)  
  
 def session\_parser(self, data):  
 print("\n" + " session ".center(75, "=") + "\n")  
 print("id".center(17, " ")  
 + "hall\_id".center(17, " ")  
 + "film\_id".center(17, " ")  
 + "date\_time".center(17, " ") + "\n")  
  
 indents = (17, 17, 17, 0)  
 self.print\_table\_data(data, indents)  
  
 def sold\_tickets\_parser(self, data):  
 print("\n" + " sold\_tickets ".center(75, "=") + "\n")  
 print("id".center(17, " ")  
 + "session\_id".center(17, " ")  
 + "place".center(17, " ")  
 + "row".center(17, " ") + "\n")  
  
 indents = (17, 17, 17, 0)  
 self.print\_table\_data(data, indents)  
  
 def technologies\_parser(self, data):  
 print("\n" + " technologies ".center(75, "=") + "\n")  
 print("id".center(17, " ")  
 + "name".center(17, " ")  
 + "description".center(17, " ") + "\n")  
  
 indents = (17, 17, 0)  
 self.print\_table\_data(data, indents)  
  
 def input\_data(self, column\_names):  
 return [input(f"Enter {column}: ") for column in column\_names]  
  
 def delete(self):  
 id\_range = input("Select id range (x y) or just id (x): ").split()  
 return [int(i) for i in id\_range]  
  
 def input\_data\_to\_search(self, data\_type):  
 if data\_type == 'integer':  
 min\_value = input("Enter min value: ")  
 max\_value = input("Enter max value: ")  
 return [min\_value, max\_value]  
 elif data\_type == 'character varying':  
 return input("Enter data: ")  
 elif data\_type == 'timestamp without time zone':  
 min\_value = input("Enter YYYY-MM-DD HH:MM:SS min date: ")  
 max\_value = input("Enter YYYY-MM-DD HH:MM:SS max date: ")  
 return [min\_value, max\_value]  
 else:  
 print(f"Cannot find data with '{data\_type}' type")  
  
 def search(self, column\_names):  
 for i, column in enumerate(column\_names):  
 print(str(i) + ": " + column)  
  
 return int(input("\nEnter choice № "))

**controller.py**

from model import Model  
from view import View  
  
  
class Controller:  
 def \_\_init\_\_(self):  
 self.view\_obj = View()  
 self.model\_obj = Model()  
  
 def menu(self):  
 table\_num = 0  
 while True:  
 operation\_num = self.view\_obj.operations\_menu()  
 if operation\_num == 8:  
 break  
 if operation\_num != 2 and operation\_num != 6:  
 table\_num = self.view\_obj.tables\_menu()  
  
 if operation\_num == 1:  
 self.model\_obj.select\_table(table\_num)  
 elif operation\_num == 2:  
 self.model\_obj.select\_all\_tables()  
 elif operation\_num == 3:  
 table\_name = self.view\_obj.tables[table\_num]  
 self.model\_obj.insert\_into\_table(table\_name)  
 elif operation\_num == 4:  
 table\_name = self.view\_obj.tables[table\_num]  
 column\_names = self.model\_obj.get\_column\_names(table\_name)  
 data = self.view\_obj.input\_data(column\_names)  
 self.model\_obj.update\_table(table\_num, data)  
 elif operation\_num == 5:  
 id\_num = self.view\_obj.delete()  
 self.model\_obj.delete\_data(table\_num, id\_num)  
 elif operation\_num == 6:  
 self.model\_obj.insert\_random\_data\_packet()  
 elif operation\_num == 7:  
 column\_names = self.model\_obj.get\_column\_names\_to\_view(table\_num)  
 search\_param = self.view\_obj.search(column\_names)  
 self.model\_obj.search\_into\_table(table\_num, search\_param)