

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра системного програмування та спеціалізованих комп’ютерних систем

**Лабораторна робота № 2**

з дисципліни

**«Бази даних і засоби управління»**

Виконав: студент III курсу

ФПМ групи КВ-02

Костюков Сергій Васильович

Перевірив: Павловський В. І.

Київ-2022

***Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL***

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

**Логічна модель предметної галузі «Продаж квитків кіно»**

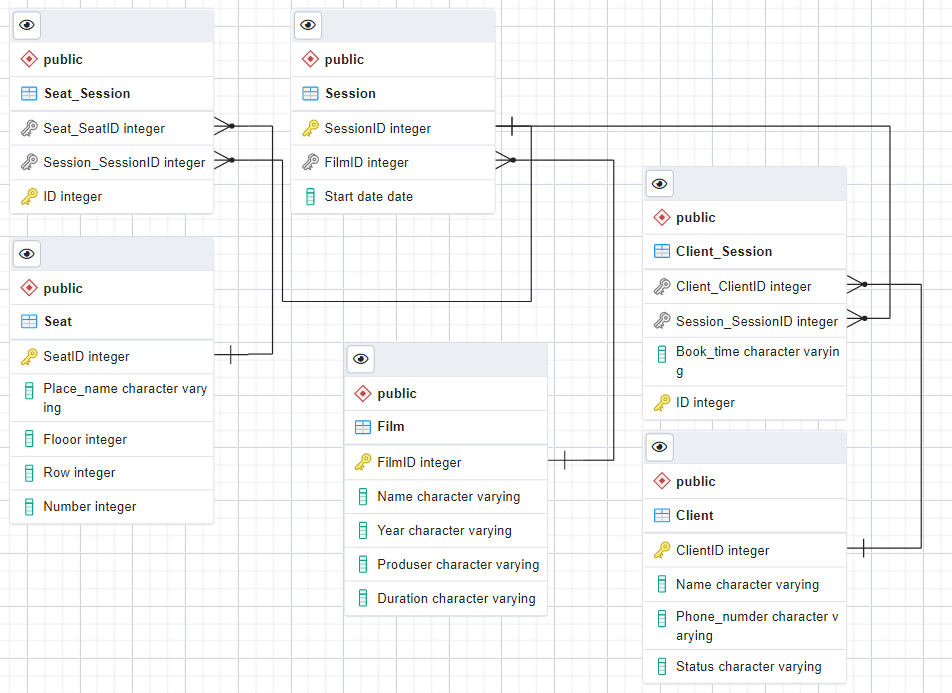


Рисунок 1 – Логічна модель предметної галузі «Продаж квитків кіно»

**Середовище розробки та налаштування програмної системи**

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

**Шаблон проектування**

MVC(модель-подання-контролер) - Шаблон проектування, який використаний у програмі.

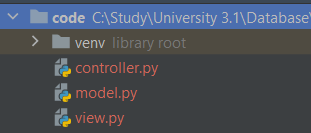
Model – представляє клас, що описує логіку використовуваних даних.

View – в нашому випадку консольний інтерфейс з яким буде взаємодіяти наш користувач.

Controller – представляє клас, що забезпечує зв'язок між користувачем і системою, поданням і сховищем даних.

**Структура програми та її опис**





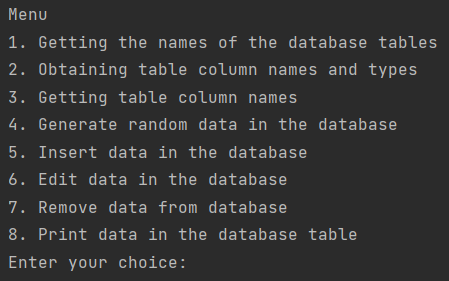
Програма умовно поділена на 3 модулі: файл controller.py, файл model.py, файл view.py. Класи, як видно з їх назв, повністю відповідають використаному патерну MVC.

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

У файлі controller.py описаний інтерфейс взаємодії з користувачем, запит бажаної дії, виконання пошуку, тощо.

У файлі view.py описаний клас, що виводить результати виконання тієї чи іншої дії на екран консолі.

**Структура меню програми**

****

Перший пункт пропонує отримання імен таблиць бази даних

Другий пункт пропонує отримання імен та типів стовпчиків таблиці

Третій пункт пропонує отримання імен стовпчиків таблиці

Четвертий пункт пропонує генерування даних в таблиці

П’ятий пункт пропонує вставку даних в таблицю

Шостий пункт пропонує оновлення даних в таблиці

Сьомий пункт пропонує видалення даних з таблиці

Восьмий пункт пропонує отримання даних таблиці

**Фрагменти програм внесення, редагування та вилучення даних у БД**

Фрагмент програми для внесення даних

def insert\_data(self, table\_name, values):  
 line = ''  
 columns = '("'  
 for key in values:  
 if values[key]:  
 line += '%(' + key + ')s,'  
 columns += key + '","'  
 columns = columns[:-3] + '")'  
 self.\_\_cursor.execute(  
 sql.SQL('INSERT INTO {} {} VALUES (' + line[:-1] + ')').format(sql.Identifier(table\_name),  
 sql.SQL(columns)),  
 values)  
 self.\_\_context.commit()

Фрагмент програми для редагування даних

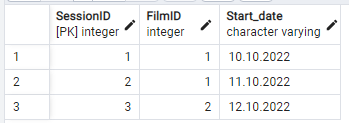
def change\_data(self, table\_name, values):  
 line = ''  
 condition = values.pop('condition')  
 for key in values:  
 if values[key]:  
 line += key + '=%(' + key + ')s,'  
 self.\_\_cursor.execute(  
 sql.SQL('UPDATE {} SET ' + line[:-1] + ' WHERE {} ').format(sql.Identifier(table\_name), sql.SQL(condition)),  
 values)  
 self.\_\_context.commit()

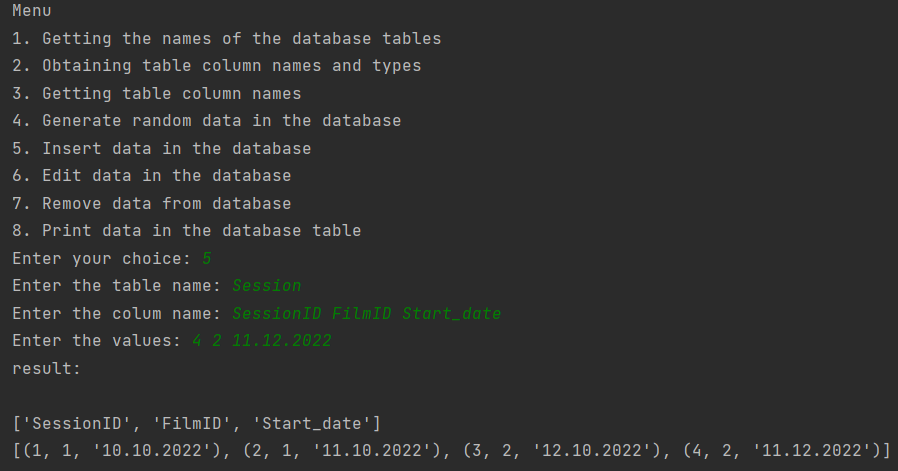
Фрагмент програми для видалення даних

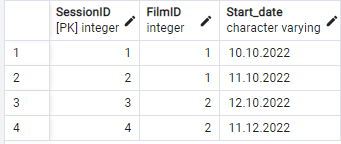
def delete\_data(self, table\_name, value, cond):  
 self.\_\_cursor.execute(  
 sql.SQL('DELETE FROM {} WHERE {} = {}').format(sql.Identifier(table\_name), sql.Identifier(value),  
 sql.SQL(cond)))  
 self.\_\_context.commit()

Дані фрагменти програми, які наведені вище, відповідають за функціонал додавання даних, редагування та вилучення даних у базі даних.  
Взаємодія відбувається через клас Model, який займається підключенням до БД, а самі функції знаходяться у файлі Controller.

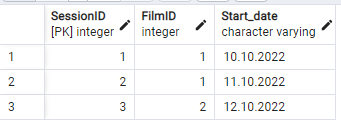
**Скріншоти результатів виконання операцій вставки запису в таблицю**

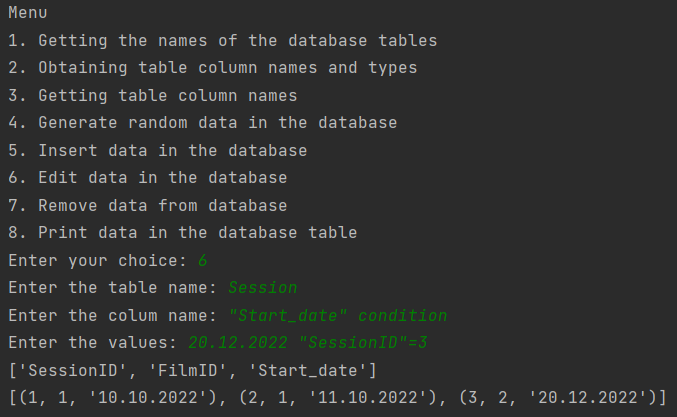
****

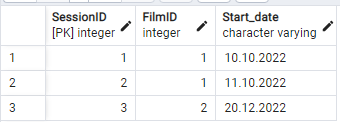
****

****

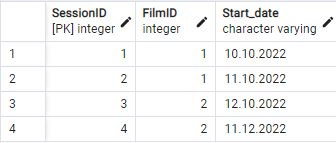
**Скріншоти результатів виконання операції редагування таблиці**

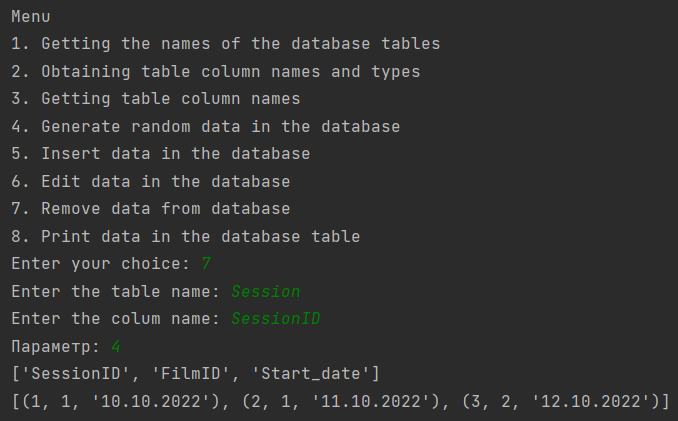
****

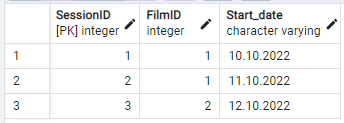
****

****

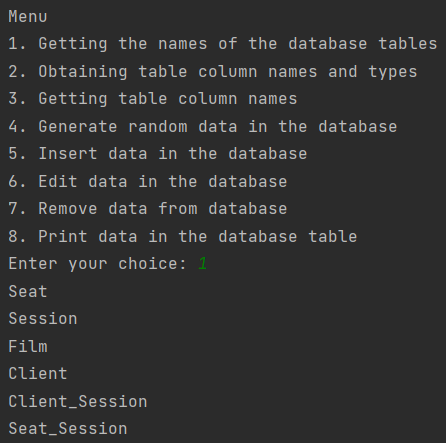
**Скріншоти результатів виконання операції видалення**

****

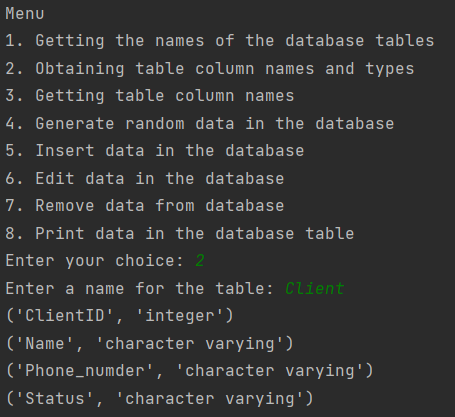
****

****

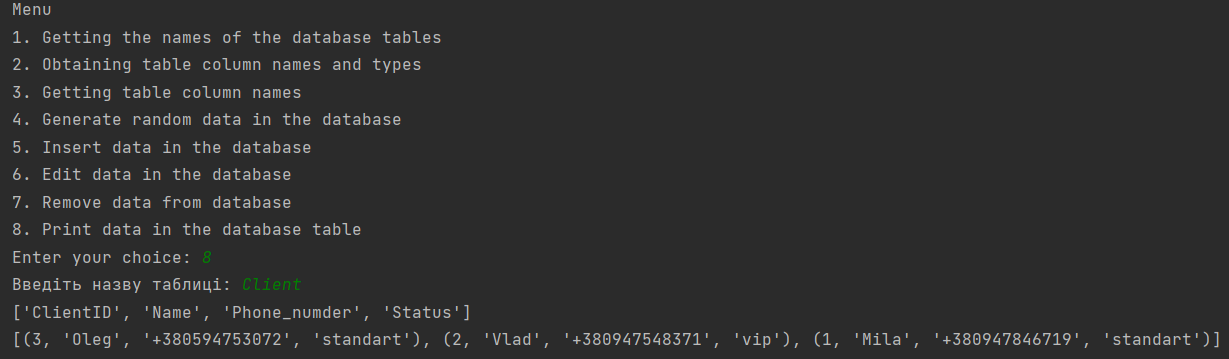
**Результат отримання назв таблиць бази даних**

****

**Результат отримання назв таблиць бази даних**

****

**Результат отримання вмісту таблиці бази даних**

****

**Текст програми**

Файл controller.py

**import** view

**import** model

**import** time

is\_end = 0

model = model.DbModel()

**while** is\_end == 0:

view.hello()

choice = input("Enter your choice: ")

match choice:

case "1":

mas = model.get\_table\_names()

view.show(mas)

time.sleep(2)

case "2":

table = input("Enter a name for the table: ")

mas = model.get\_column\_types(table)

view.show(mas)

time.sleep(2)

case "3":

table = input("Enter a name for the table: ")

mas = model.get\_column\_names(table)

view.show(mas)

time.sleep(2)

case "4":

table = input("Enter the table name: ")

count = input("Enter count: ")

model.generate\_data(table, count)

mas = model.get\_table\_data(table)

view.show(mas)

time.sleep(2)

case "5":

table = input("Enter the table name: ")

columns = input("Enter the colum name: ").split(' ')

val = input("Enter the values: ").split(' ')

values = {key: value **for** (key, value) **in** zip(columns, val)}

model.insert\_data(table, values)

**print**("result:\n")

mas = model.get\_table\_data(table)

view.show(mas)

time.sleep(2)

case "6":

table = input("Enter the table name: ")

columns = input("Enter the colum name: ").split(' ')

val = input("Enter the values: ").split(' ')

values = {key: value **for** (key, value) **in** zip(columns, val)}

model.change\_data(table, values)

mas = model.get\_table\_data(table)

view.show(mas)

time.sleep(2)

case "7":

table = input("Enter the table name: ")

column = input("Enter the colum name: ")

param = input("Параметр: ")

model.delete\_data(table, column, param)

mas = model.get\_table\_data(table)

view.show(mas)

time.sleep(2)

case "8":

table = input("Enter the table name: ")

mas = model.get\_table\_data(table)

view.show(mas)

time.sleep(2)

case \_:

is\_end = 1

**print**("End")

Файл model.py

**import** psycopg2

**from** psycopg2 **import** sql

**class** DbModel:

**def** \_\_init\_\_(self):

self.host = "localhost"

self.database = "cinema\_tickets"

self.user = "postgres"

self.password = "123"

**try**:

self.\_\_context = psycopg2.connect(host=self.host, database=self.database, user=self.user,

password=self.password)

self.\_\_cursor = self.\_\_context.cursor()

self.\_\_table\_names = None

**except** Exception **as** \_ex:

**print**("[INFO] Error while working with PostgreSQL", \_ex)

**def** \_\_del\_\_(self):

self.\_\_cursor.close()

self.\_\_context.close()

**def** clear\_transaction(self):

self.\_\_context.rollback()

**def** get\_table\_names(self):

**if** self.\_\_table\_names **is** None:

self.\_\_cursor.execute("""SELECT table\_name

                             FROM information\_schema.tables

                             WHERE table\_schema = 'public'""")

self.\_\_table\_names = [table[0] **for** table **in** self.\_\_cursor]

**return** self.\_\_table\_names

**def** get\_column\_types(self, table\_name):

self.\_\_cursor.execute("""SELECT column\_name, data\_type

            FROM information\_schema.columns

           WHERE table\_schema = 'public' AND table\_name = %s

           ORDER BY table\_schema, table\_name""", (table\_name,))

**return** self.\_\_cursor.fetchall()

**def** get\_column\_names(self, table\_name):

self.\_\_cursor.execute("""

            SELECT column\_name FROM information\_schema.columns

            WHERE table\_schema = 'public' AND table\_name = %s

            ORDER BY table\_schema, table\_name""", (table\_name,))

**return** [x[0] **for** x **in** self.\_\_cursor.fetchall()]

**def** get\_foreign\_key\_info(self, table\_name):

self.\_\_cursor.execute("""

           SELECT kcu.column\_name, ccu.table\_name AS

                  foreign\_table\_name,

                  ccu.column\_name AS foreign\_column\_name

           FROM information\_schema.table\_constraints AS tc

              JOIN information\_schema.key\_column\_usage AS kcu

                 ON tc.constraint\_name = kcu.constraint\_name

                 AND tc.table\_schema = kcu.table\_schema

              JOIN information\_schema.constraint\_column\_usage AS ccu

                 ON ccu.constraint\_name = tc.constraint\_name

                 AND ccu.table\_schema = tc.table\_schema

           WHERE tc.constraint\_type = 'FOREIGN KEY' AND

                          tc.table\_name=%s;""", (table\_name,))

**return** self.\_\_cursor.fetchall()

**def** get\_table\_data(self, table\_name):

id\_column = self.get\_column\_types(table\_name)[0][0]

cursor = self.\_\_cursor

**try**:

cursor.execute(

sql.SQL('SELECT \* FROM {}').format(sql.Identifier(table\_name), sql.SQL(id\_column)))

**except** Exception **as** \_ex:

**print**("[INFO] Error while working with PostgreSQL", \_ex)

**return** ([col.name **for** col **in** cursor.description], cursor.fetchall())

**def** insert\_data(self, table\_name, values):

line = ''

columns = '("'

**for** key **in** values:

**if** values[key]:

line += '%(' + key + ')s,'

columns += key + '","'

columns = columns[:-3] + '")'

self.\_\_cursor.execute(

sql.SQL('INSERT INTO {} {} VALUES (' + line[:-1] + ')').format(sql.Identifier(table\_name),

sql.SQL(columns)),

values)

self.\_\_context.commit()

**def** generate\_data(self, table\_name, count):

types = self.get\_column\_types(table\_name)

fk\_array = self.get\_foreign\_key\_info(table\_name)

select\_subquery = ""

insert\_query = 'INSERT INTO "' + table\_name + '" ("'

**for** i **in** range(1, len(types)):

t = types[i]

name = t[0]

type = t[1]

fk = [x **for** x **in** fk\_array **if** x[0] == name]

**if** fk:

select\_subquery += ('(SELECT "{}" FROM "{}" ORDER BY RANDOM(), ser LIMIT 1)'.format(fk[0][2], fk[0][1]))

**elif** type == 'integer':

select\_subquery += 'trunc(random()\*100)::INT'

**elif** type == 'character varying':

select\_subquery += 'chr(trunc(65 + random()\*25)::INT) || chr(trunc(65 + random()\*25)::INT)'

**elif** type == 'date':

select\_subquery += """ date(timestamp '2014.01.10' +

                    random() \*

                    (timestamp '2020.01.20' - timestamp '2014.01.10'))"""

**elif** type == 'time without time zone':

select\_subquery += "time '00:00:00' + DATE\_TRUNC('second',RANDOM() \* time '24:00:00')"

**else**:

**continue**

insert\_query += name

**if** i != len(types) - 1:

select\_subquery += ','

insert\_query += '","'

**else**:

insert\_query += '") '

self.\_\_cursor.execute(

insert\_query + "SELECT " + select\_subquery +

" FROM generate\_series(1," + str(count) + ") as ser")

self.\_\_context.commit()

**def** change\_data(self, table\_name, values):

line = ''

condition = values.pop('condition')

**for** key **in** values:

**if** values[key]:

line += key + '=%(' + key + ')s,'

self.\_\_cursor.execute(

sql.SQL('UPDATE {} SET ' + line[:-1] + ' WHERE {} ').format(sql.Identifier(table\_name), sql.SQL(condition)),

values)

self.\_\_context.commit()

**def** delete\_data(self, table\_name, value, cond):

self.\_\_cursor.execute(

sql.SQL('DELETE FROM {} WHERE {} = {}').format(sql.Identifier(table\_name), sql.Identifier(value),

sql.SQL(cond)))

self.\_\_context.commit()

Файл view.py

**def** hello():

**print**("Menu\n"

"1. Getting the names of the database tables\n"

"2. Obtaining table column names and types\n"

"3. Getting table column names\n"

"4. Generate random data in the database\n"

"5. Insert data in the database\n"

"6. Edit data in the database\n"

"7. Remove data from database\n"

"8. Print data in the database table")

**def** show(mas):

**for** element **in** mas:

**print**(element)