

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”*

Виконав:

студент ІII курсу

групи КВ-01

Рябенко Б. Ю.

Перевірив:

Павловський В. І.

Київ – 2022

*Метою роботи* є 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.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

*Деталізоване завдання*:

1. 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.
2. 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**!
3. 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.
4. Програмний код організувати згідно шаблону Model-View-Controller(MVC). При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати **лише мову SQL** (без ORM).

**Логічна модель бази даних**

Нижче (Рисунок 1) наведено логічну модель бази даних:

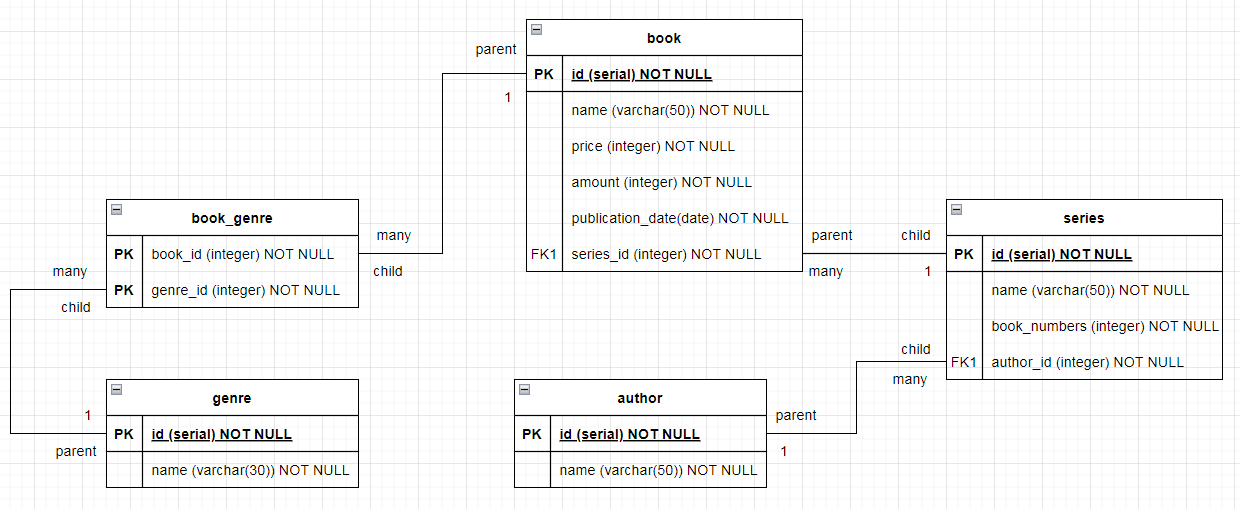


Рисунок 1 – Логічна модель бази даних

**Середовище розробки та налаштування підключення до бази даних**

Для розробки використовувалась мова програмування Python, середовище розробки PyCharm Community Edition, 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 папці Models.

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 вводяться користувачем дані і обробляє їх. І в залежності від результатів обробки відправляє користувачеві певний висновок, наприклад, у вигляді подання.

**Структура програми та її опис**

На рисунку 2 відображено деревовидну структуру програми:

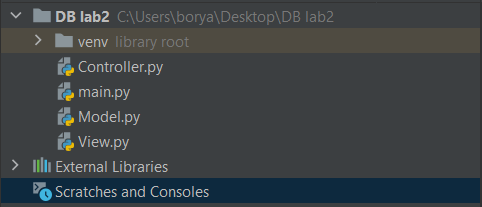


Рисунок 2 – структура програми

Програма поділена на 4 файли: main.py, Controller.py, Model.py, View.py.

У файлі View.py описані функції, що виводять дані у консоль.

У файлі Model.py описаний клас, який реалізує методи управління базою даних

У файлі Controller.py описаний клас, який використовує функції та методи файлів View.py та Model.py.

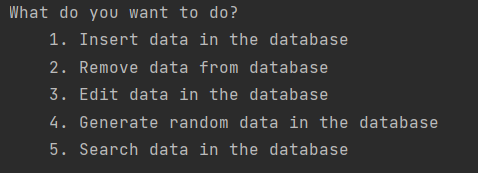
**Структура меню програми**

Рисунок 3 – Початкове меню програми

Перший пункт пропонує внесення даних. Користувач повинен вибрати до якої таблиці вносити дані, а потім ввести дані, які будуть додані до таблиці.

Другий пункт пропонує видалення даних. Користувач повинен вибрати з якої таблиці видалити дані, а потім ввести номер рядку.

Третій пункт пропонує редагування даних. Користувач повинен вибрати яку таблицю редагувати та номер рядку яка буде редагуватися, після цього користувач може вводити дані.

Четвертий пункт пропонує заповнення даних випадковим чином. Користувач повинен вибрати яку таблицю заповнювати, а потім скільки всього потрібно даних згенерувати.

П’ятий пункт пропонує пошук за атрибутами з декількох таблиць. Користувач має вибрати, який запит він хоче виконувати, а тоді вже здійснювати відповідний пошук. Після введення необхідних атрибутів він побачить результати пошуку та час самого пошуку.

**Лістинги програм**

Лістинг фрагменту програми для внесення даних

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

line = '('

columns = '('

**for** key **in** range(len(values)):

**if** values[key]:

line += column[key] + ','

columns += f"'{values[key]}'" + ','

columns = columns[:-**1**] + ')'

line = line[:-**1**] + ')'

**try**:

self.\_\_cursor.execute(

sql.SQL('INSERT INTO {} {} VALUES {}').format(sql.Identifier(table\_name),

sql.SQL(line), sql.SQL(columns)))

self.\_\_context.commit()

**except** **Exception** **as** e:

print(e)

Лістинг фрагменту програми для видалення даних

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

string = f"{parameter}={cond}"

self.\_\_cursor.execute(

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

self.\_\_context.commit()

Лістинг фрагменту програми для зміни даних

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

string = f"{id\_name} = '{cond}'"

columns = ''

**for** key **in** range(len(values)):

**if** values[key]:

columns += f"{column[key]} = '{values[key]}'" + ','

columns = columns[:-**1**] + ''

**try**:

self.\_\_cursor.execute(

sql.SQL('UPDATE {} SET {} WHERE {}').format(sql.Identifier(table\_name),

sql.SQL(columns), sql.SQL(string)))

self.\_\_context.commit()

**except** **Exception** **as** e:

print(e)

Лістинг фрагменту програми для пошуку даних у таблицях

**def** **find\_data**(self, num):

**if** num == **1**:

string = f"SELECT \* FROM book WHERE amount > {int(input('Write min. amount: '))}"

self.\_\_cursor.execute(string)

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

**elif** num == **2**:

string = f"SELECT \* FROM author WHERE name = '{input('Write name: ')}'"

self.\_\_cursor.execute(string)

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

**elif** num == **3**:

string = f"SELECT \* FROM genre WHERE name = '{input('Write name: ')}'"

self.\_\_cursor.execute(string)

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

Лістинг фрагменту програми для генерації випадкових даних у таблицях

**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** **if** "id" **in** self.get\_table\_data(table\_name)[**0**] **else** **0**, len(types)):

t = types[i]

name = t[**0**]

kind = 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** kind == 'integer':

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

**elif** kind == 'character varying':

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

**elif** kind == 'date':

select\_subquery += """ date(timestamp '2014-01-10' + random() \*(timestamp '2020-01-20' - timestamp '2014-01-10'))"""

**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()

Для генерування випадкових даних спочатку, для кожного типу даних в обраній таблиці, за спеціальним шаблоном генерується рядок, який буде генерувати набір даних певного типу. Потім рядки для кожного стовпчика поєднуються, і на виході маємо готовий SQL-запит для генерації даних.

Дані фрагменти програми, які наведені вище, відповідають за додавання, видалення, зміну, генерування, та пошук даних у базі даних. Ці методи реалізовані у файлі Model.py, які потім використовує файл Controller.py.

**Результати роботи програми**

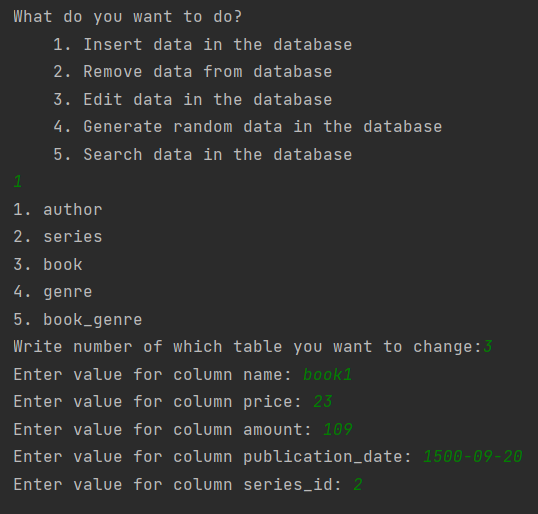
****

Рисунок 4 – Введення даних

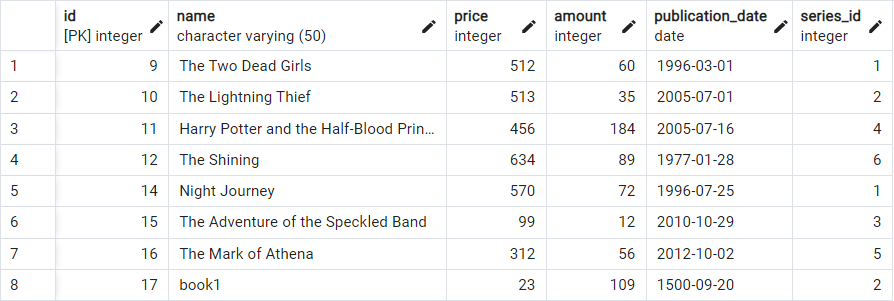
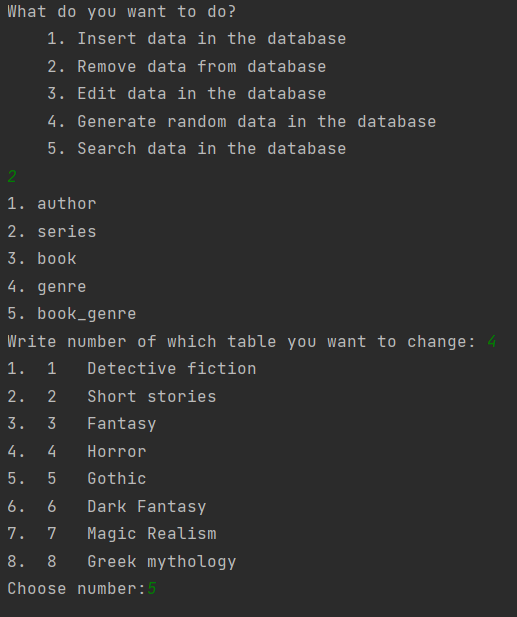


Рисунок 5 – Таблиця ‘book’ після внесення до неї даних

Рисунок 6 – Видалення даних

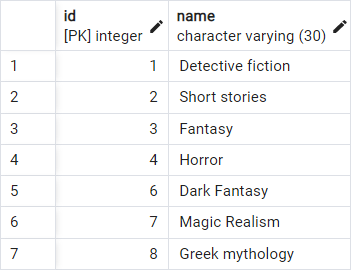


Рисунок 7 – Таблиця ‘genre’ після видалення даних

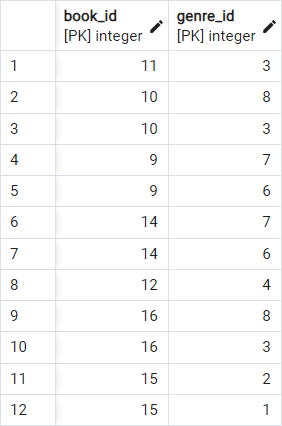
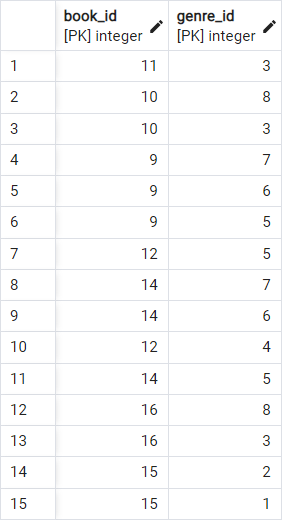


Рисунок 8 – Таблиця ‘genre\_book’ до та після видалення даних у таблиці ‘genre’

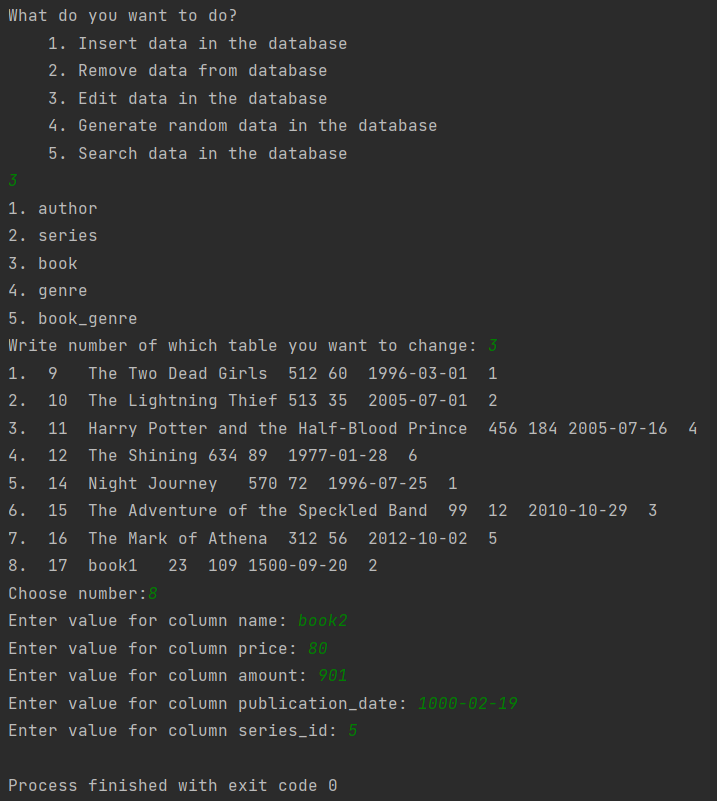
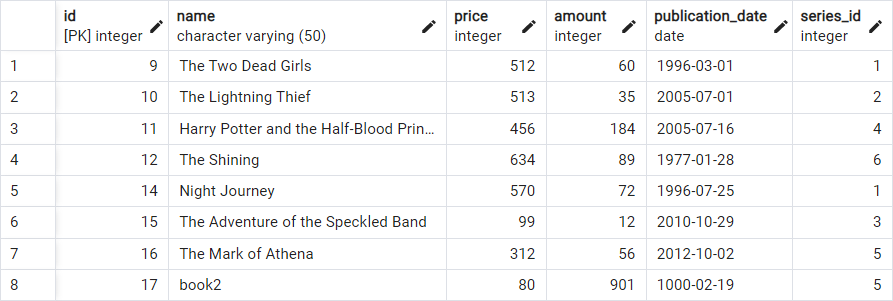


Рисунок 9 – Зміна даних



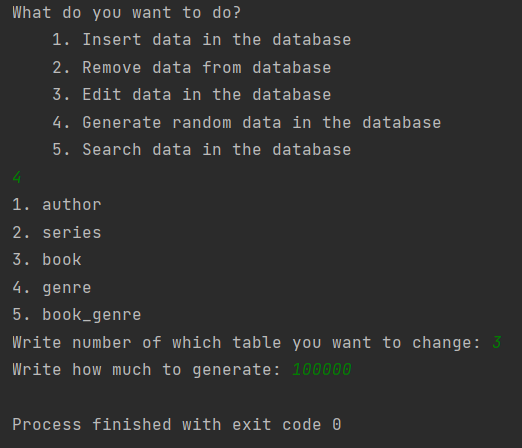
Рисунок 10 – Таблиця ‘book’ після зміни даних

Рисунок 11 – Генерація випадкових значень

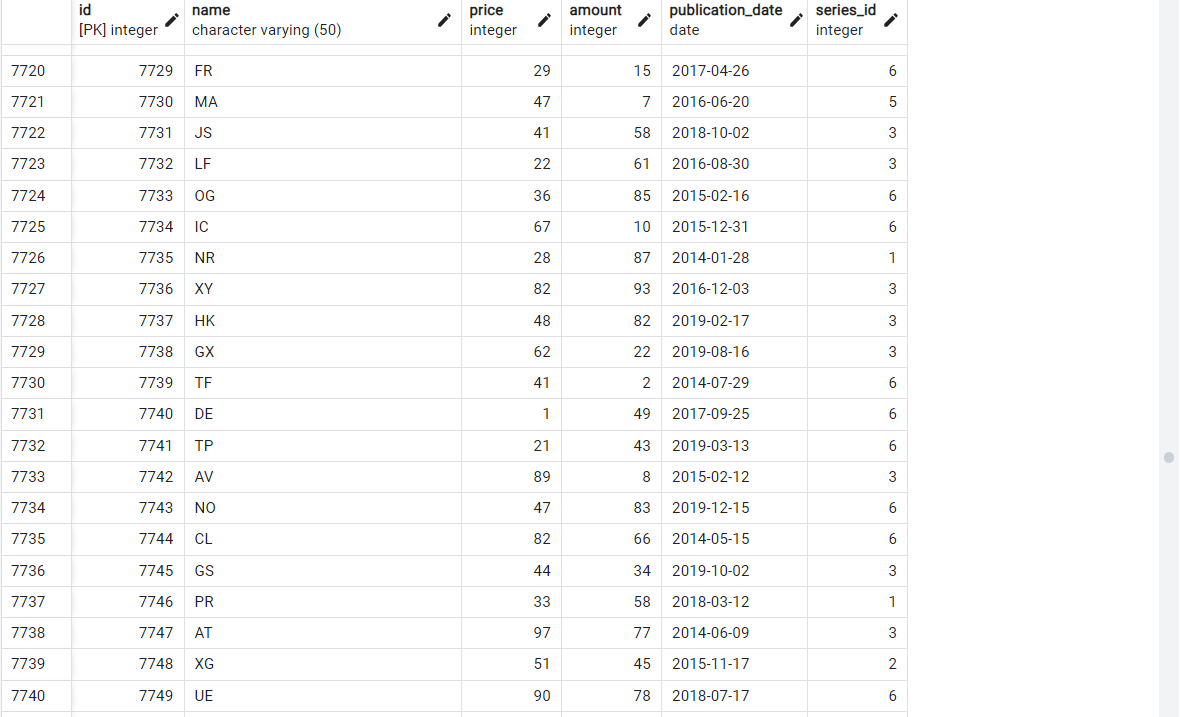


Рисунок 12 – таблиця ‘book’ після генерації випадкових значень

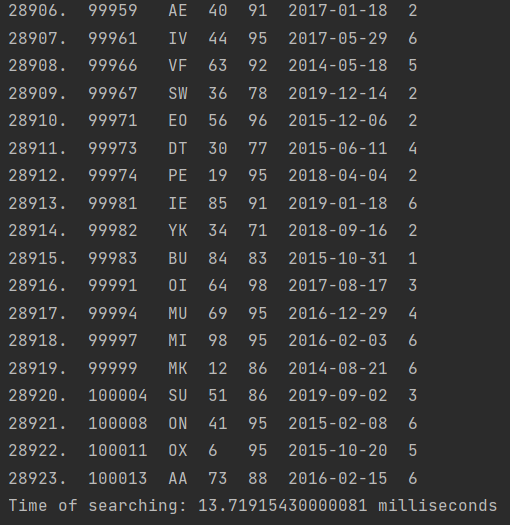
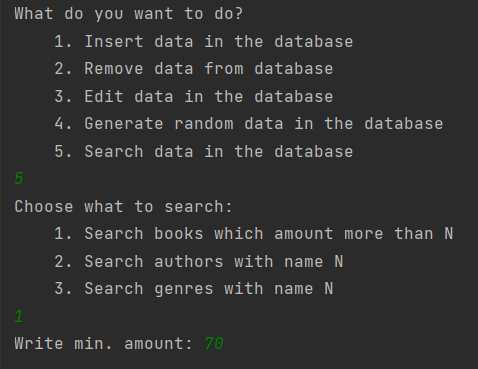


Рисунок 13 – Пошук даних

**Текст програми**

***Main.py***

**from** **Controller** **import** \*

a = Controller()

***View.py***

**def** **tables\_print**(tables):

**for** i **in** range(len(tables)):

print(f"{i+1}.", tables[i])

**return** tables[int(input("Write number of which table you want to change: "))-**1**]

**def** **table\_print**(rows):

number = **1**

**for** line **in** rows:

string = f"{number}.**\t**"

**for** elem **in** range(len(line)):

string += str(line[elem]) + "**\t**"

print(string)

number += **1**

**return** number

**def** **menu**():

print("""What do you want to do?

1. Insert data in the database

2. Remove data from database

3. Edit data in the database

4. Generate random data in the database

5. Search data in the database""")

n = int(input())

**if** **1** <= n <= **5**:

**return** n

**else**:

print("Wrong parameter")

**def** **find\_data\_menu**():

print("""Choose what to search:

1. Search books which amount more than N

2. Search authors with name N

3. Search genres with name N""")

***Model.py***

**import** **psycopg2**

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

**class** **Model**:

**def** **\_\_init\_\_**(self, dbname, user\_name, password, host):

**try**:

self.\_\_context = psycopg2.connect(host=host, user=user\_name, password=password, database=dbname)

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

self.\_\_table\_names = **None**

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

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

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

**if** self.\_\_context:

self.\_\_cursor.close()

self.\_\_context.close()

**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\_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 {} ORDER BY {} ASC').format(sql.Identifier(table\_name), sql.SQL(id\_column)))

**except** **Exception** **as** e:

**return** str(e)

**return** [col.name **for** col **in** cursor.description], 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** **insert\_data**(self, table\_name, values, column):

line = '('

columns = '('

**for** key **in** range(len(values)):

**if** values[key]:

line += column[key] + ','

columns += f"'{values[key]}'" + ','

columns = columns[:-**1**] + ')'

line = line[:-**1**] + ')'

**try**:

self.\_\_cursor.execute(

sql.SQL('INSERT INTO {} {} VALUES {}').format(sql.Identifier(table\_name),

sql.SQL(line), sql.SQL(columns)))

self.\_\_context.commit()

**except** **Exception** **as** e:

print(e)

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

string = f"{parameter}={cond}"

self.\_\_cursor.execute(

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

self.\_\_context.commit()

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

string = f"{id\_name} = '{cond}'"

columns = ''

**for** key **in** range(len(values)):

**if** values[key]:

columns += f"{column[key]} = '{values[key]}'" + ','

columns = columns[:-**1**] + ''

**try**:

self.\_\_cursor.execute(

sql.SQL('UPDATE {} SET {} WHERE {}').format(sql.Identifier(table\_name),

sql.SQL(columns), sql.SQL(string)))

self.\_\_context.commit()

**except** **Exception** **as** e:

print(e)

**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** **if** "id" **in** self.get\_table\_data(table\_name)[**0**] **else** **0**, len(types)):

t = types[i]

name = t[**0**]

kind = 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** kind == 'integer':

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

**elif** kind == 'character varying':

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

**elif** kind == 'date':

select\_subquery += """ date(timestamp '2014-01-10' + random() \*

(timestamp '2020-01-20' - timestamp '2014-01-10'))"""

**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** **find\_data**(self, num):

**if** num == **1**:

string = f"SELECT \* FROM book WHERE amount > {int(input('Write min. amount: '))}"

self.\_\_cursor.execute(string)

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

**elif** num == **2**:

string = f"SELECT \* FROM author WHERE name = '{input('Write name: ')}'"

self.\_\_cursor.execute(string)

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

**elif** num == **3**:

string = f"SELECT \* FROM genre WHERE name = '{input('Write name: ')}'"

self.\_\_cursor.execute(string)

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

***Controller.py***

**from** **View** **import** \*

**from** **Model** **import** \*

**import** **time**

**def** **insert**(model):

table = tables\_print(model.get\_table\_names())

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

**if** 'id' **in** table\_data[**0**]:

table\_data[**0**].remove('id')

values = []

**for** i **in** range(len(table\_data[**0**])):

values.append(input(f"Enter value for column {table\_data[0][i]}: "))

model.insert\_data(table, values, table\_data[**0**])

**def** **delete**(model):

table = tables\_print(model.get\_table\_names())

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

table\_print(table\_data[**1**])

model.delete\_data(table, table\_data[**0**][**0**], table\_data[**1**][int(input('Choose number:')) - **1**][**0**])

**def** **change**(model):

table = tables\_print(model.get\_table\_names())

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

table\_print(table\_data[**1**])

id\_name = table\_data[**0**][**0**]

num = table\_data[**1**][int(input('Choose number:')) - **1**][**0**]

**if** 'id' **in** table\_data[**0**]:

table\_data[**0**].remove('id')

values = []

**for** i **in** range(len(table\_data[**0**])):

values.append(input(f"Enter value for column {table\_data[0][i]}: "))

model.change\_data(table, values, table\_data[**0**], num, id\_name)

**def** **random**(model):

table = tables\_print(model.get\_table\_names())

model.generate\_data(table, int(input("Write how much to generate: ")))

**def** **find**(model):

find\_data\_menu()

n = int(input())

data = model.find\_data(n) **if** **1** <= n <= **3** **else** print("Wrong parameter")

table\_print(data)

**class** **Controller**:

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

self.\_\_operate = menu()

**if** self.\_\_operate:

self.operation()

**def** **operation**(self):

model = Model("book\_shop", "postgres", "qwerty", "127.0.0.1")

**if** self.\_\_operate == **1**:

insert(model)

**elif** self.\_\_operate == **2**:

delete(model)

**elif** self.\_\_operate == **3**:

change(model)

**elif** self.\_\_operate == **4**:

random(model)

**elif** self.\_\_operate == **5**:

t1 = time.perf\_counter()

find(model)

t2 = time.perf\_counter()

print(f"Time of searching: {t2-t1} milliseconds")