

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих комп’ютерних систем**

**Розрахунково-графічна робота**

з дисципліни

**«Бази даних та засоби управління»**

**Тема: «Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL»**

Виконав: студент 3 курсу

ФПМ групи КВ-13

Абраменко Д. О.

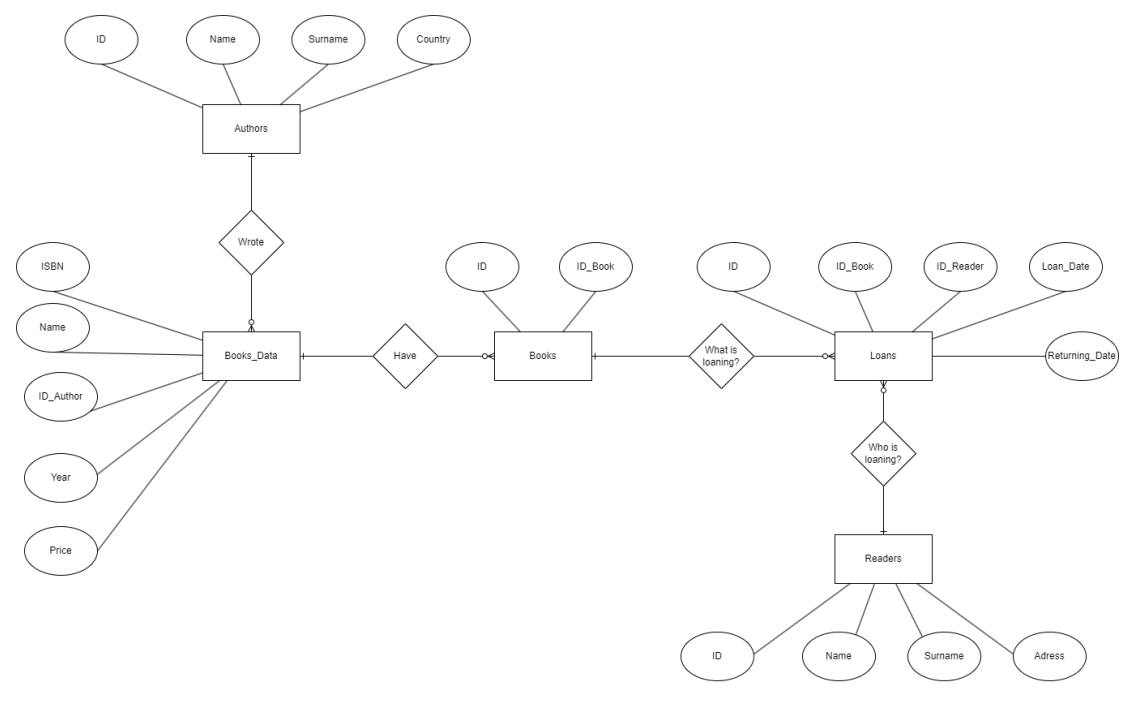
Київ – 2023

*Telegram:* [*https://t.me/SaigeNick*](https://t.me/SaigeNick) *Git: https://github.com/SaigeNick/BD\_Library  
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).

**Концептуальна модель предметної області "Бібліотека"**



В концептуальній моделі предметної області "Бібліотека" виділяються наступні сутності та зв'язки між ними:

Сутність “Authors” з атрибутами: ID, Surname, Name, Country.

Сутність "Books\_Data" з атрибутами : ID, Name, ID\_Author, Year, Price;

Сутність “Books” з атрибутами: ID, ID\_Book.

Сутність “Loans” з атрибутами: ID, ID\_Book, ID\_Reader, Loan\_Date,

Returning\_Date.

Сутність “Readers” з атрибутами: ID, Name, Surname, Adress.

Між сутностями “Loans” та "Books" зв’язок R(1:N) тому, що одну книгу

може бути видано багато разів, і кожній видачі відповідає одна книга.

Між сутностями “Books” та "Books\_Data" зв’язок R(1:N) тому, що кожна

книга може мати разні дані, але тільки один номер.

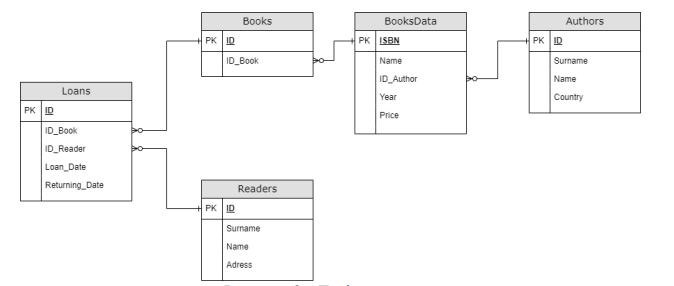
Між сутностями “Books\_Data” та "Authors" зв’язок R(1:N) тому, що кожен

автор може мати декілька книг, але у книги може бути один автор.

Між сутностями “Loans” та "Readers" зв’язок R(1:N) тому, що кожній

видачі відповідає один читач, але читач може мати багато видач.

**Схема бази даних у графічному вигляді**



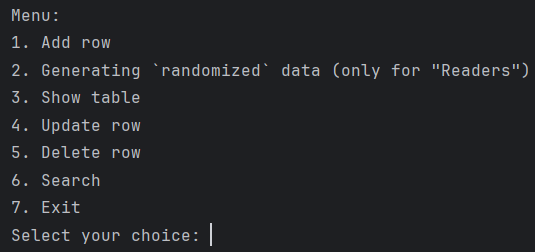
*Для лабораторної роботи знадобилися:*

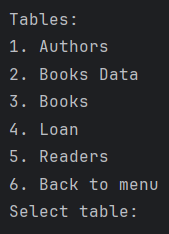
Середовище для відлагодження SQL-запитів до бази даних – PgAdmin4.

Мова програмування – Python 3.10.

Середовище розробки програмного забезпечення – PyCharm Community Edition.

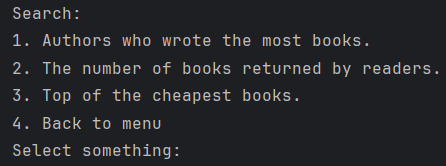
Бібліотека взаємодії з PostgreSQL - psycopg2.





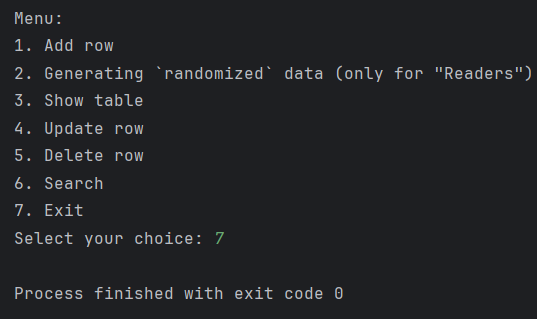
Меню складається з 7 пунктів:

1. **Add row** служить для додавання рядка в таблицю. Після вибору цього пункту, потрібно обрати таблицю, для якої буде виконана ця операція, після чого, треба ввести дані для кожного атрибуту таблиці, щоб додати новий рядок.
2. **Generating `randomized` data**. Для цього пункту було обрано таблицю “Readers”. Цей пункт створений для додавання «рандомізованих» даних. Користувачу потрібно ввести число полів, скільки треба додати у таблицю.
3. **Show table** служить для показу таблиці. Перед виведенням, користувач обирає, яку саме таблицю потрібно вивести. Після цього на екрані виводяться всі поля обраної таблиці БД.
4. **Update row** використовується для редагування полів по id у таблицях. Спочатку потрібно обрати, для якої таблиці буде відбуватися, після чого потрібно ввести id поля, яке потрібно змінити. Залишається ввести нові дані для кожного атрибуту таблиці.
5. **Delete row** служить для видалення рядку по id у таблицях. Спочатку потрібно обрати, для якої таблиці буде відбуватися видалення рядка, після чого користувач вводить id рядка, який потрібно видалити.
6. **Search** створений для пошуку за атрибутами з декількох таблиць. Пропонується 4 варіанти вибору:



Роботу цих запитів буде розглянуто у завданні №3.

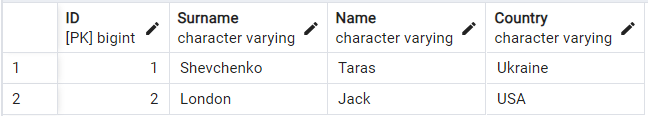
1. **Exit** служить для виходу з програми.



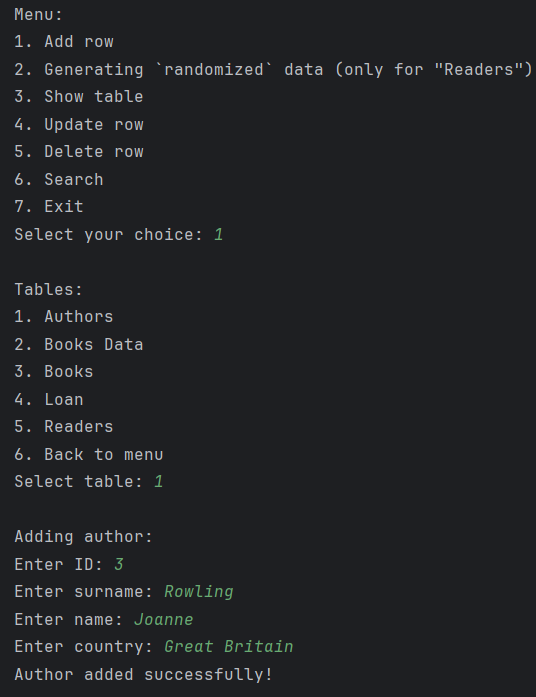
**Завдання 1**

**Додавання рядка**

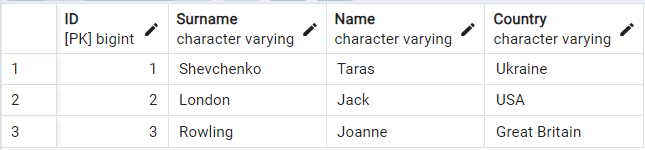
***Таблиця “Authors” до:***

******

***Процес додавання рядка:***

******

***Таблиця “Authors” після:***

******

***Лістинг add:***

**def** add\_author(self, id, surname, name, country):

c = self.conn.cursor()

c.execute('INSERT INTO public."Authors"("ID", "Surname", "Name", "Country") VALUES(%s, %s, %s, %s);', (id, surname, name, country))

self.conn.commit()

**def** add\_book\_data(self, isbn, id\_author, name, year, price):

c = self.conn.cursor()

c.execute('INSERT INTO public."Books\_Data"("ISBN", "ID\_Author", "Name", "Year", "Price") VALUES(%s, %s, %s, %s, %s);', (isbn, id\_author, name, year, price))

self.conn.commit()

**def** add\_book(self, id, id\_book):

c = self.conn.cursor()

c.execute('INSERT INTO public."Books"("ID", "ID\_Book") VALUES(%s, %s);', (id, id\_book))

self.conn.commit()

**def** add\_loan(self, id, id\_book, id\_user, loan\_date, return\_date):

c = self.conn.cursor()

c.execute('INSERT INTO public."Loan"("ID", "ID\_Book", "ID\_User", "Loan\_Date", "Return\_Date") VALUES(%s, %s, %s, %s, %s);', (id, id\_book, id\_user, loan\_date, return\_date))

self.conn.commit()

**def** add\_reader(self, id, surname, name, adress):

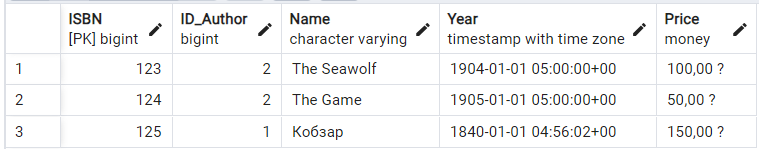
c = self.conn.cursor()

c.execute('INSERT INTO public."Readers"("ID", "Surname", "Name", "Adress") VALUES(%s, %s, %s, %s);', (id, surname, name, adress))

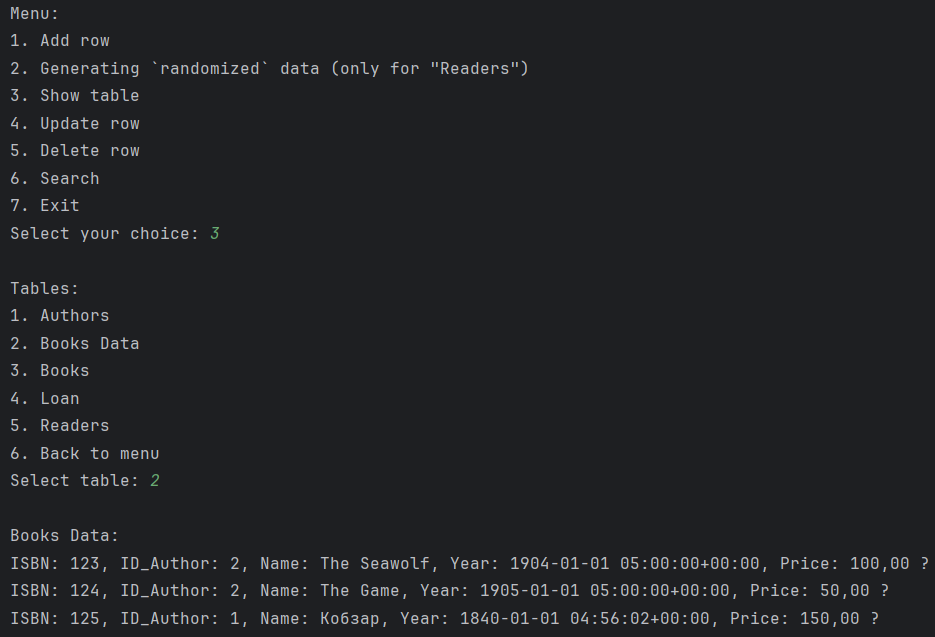
self.conn.commit()

**Перегляд таблиці**

***Таблиця “Books\_Data”:***

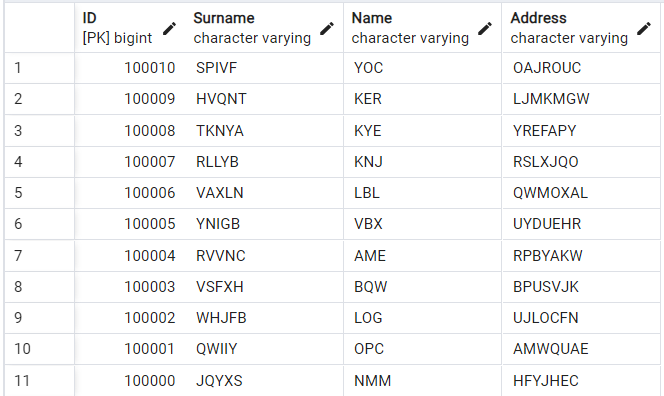
******

***Процес перегляду таблиці:***

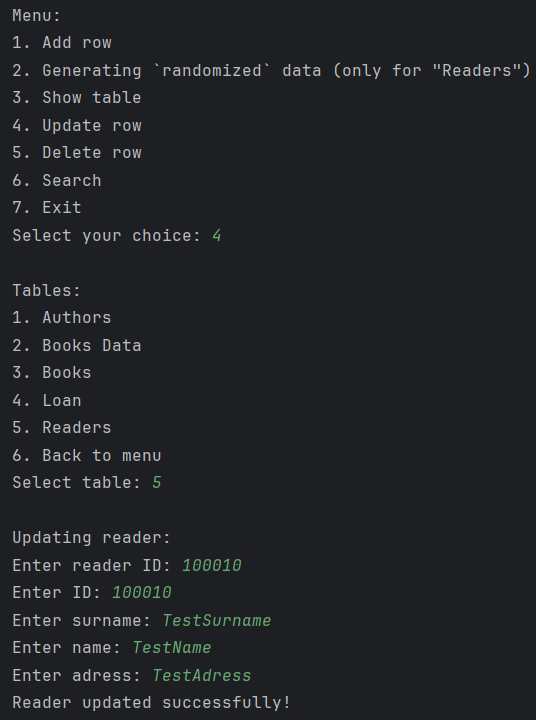
******

**Оновлення рядка**

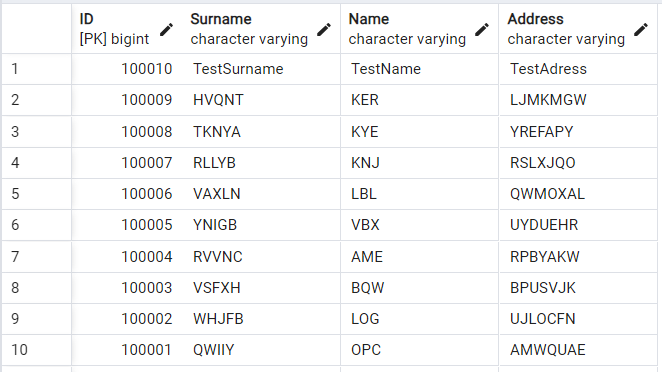
***Таблиця “Readers” до:***

******

***Процес оновлення рядка:***

******

***Таблиця “Readers” після:***

******

***Лістинг update:***

**def** update\_author(self, author\_id, surname, name, country, id):

c = self.conn.cursor()

c.execute('UPDATE public."Authors" SET "ID"=%s, "Surname"=%s, "Name"=%s, "Country"=%s WHERE "ID"=%s', (author\_id, surname, name, country, id))

self.conn.commit()

**def** update\_book\_data(self, isbn, id\_author, name, year, price, id):

c = self.conn.cursor()

c.execute('UPDATE public."Books\_Data" SET "ISBN"=%s, "ID\_Author"=%s, "Name"=%s, "Year"=%s, "Price"=%s WHERE "ISBN"=%s', (isbn, id\_author, name, year, price, id))

self.conn.commit()

**def** update\_book(self, book\_id, id\_book, id):

c = self.conn.cursor()

c.execute('UPDATE public."Books" SET "ID"=%s, "ID\_Book"=%s WHERE "ID"=%s', (book\_id, id\_book, id))

self.conn.commit()

**def** update\_loan(self, loan\_id, id\_book, id\_user, loan\_date, return\_date, id):

c = self.conn.cursor()

c.execute('UPDATE public."Loan" SET "ID"=%s, "ID\_Book"=%s, "ID\_User"=%s, "Loan\_Date"=%s, "Return\_Date"=%s WHERE "ID"=%s', (loan\_id, id\_book, id\_user, loan\_date, return\_date, id))

self.conn.commit()

**def** update\_reader(self, reader\_id, surname, name, adress, id):

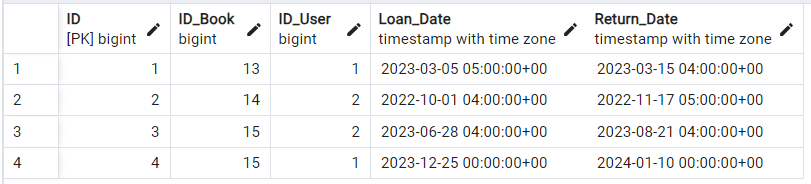
c = self.conn.cursor()

c.execute('UPDATE public."Readers" SET "ID"=%s, "Surname"=%s, "Name"=%s, "Address"=%s WHERE "ID"=%s', (reader\_id, surname, name, adress, id))

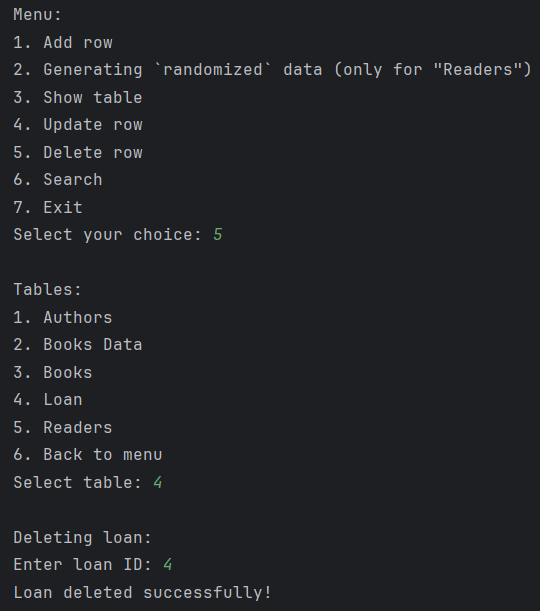
self.conn.commit()

**Видалення рядка**

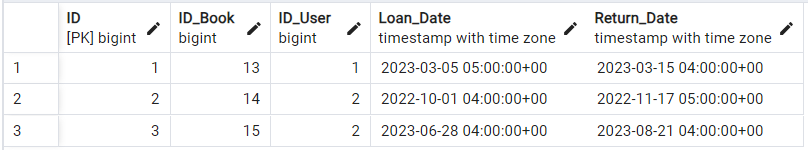
***Таблиця “Loan” до:***

******

***Процес видалення рядка:***

******

***Таблиця “Loan” після:***

******

***Лістинг delete:***

**def** delete\_author(self, id):

c = self.conn.cursor()

c.execute('DELETE FROM public."Authors" WHERE "ID"=%s', (id,))

self.conn.commit()

**def** delete\_book\_data(self, isbn):

c = self.conn.cursor()

c.execute('DELETE FROM public."Books\_Data" WHERE "ISBN"=%s', (isbn,))

self.conn.commit()

**def** delete\_book(self, id):

c = self.conn.cursor()

c.execute('DELETE FROM public."Books" WHERE "ID"=%s', (id,))

self.conn.commit()

**def** delete\_loan(self, id):

c = self.conn.cursor()

c.execute('DELETE FROM public."Loan" WHERE "ID"=%s', (id,))

self.conn.commit()

**def** delete\_reader(self, id):

c = self.conn.cursor()

c.execute('DELETE FROM public."Readers" WHERE "ID"=%s', (id,))

self.conn.commit()

**Завдання 2**

**Генерування «рандомізованих» даних**

Запит, що був використаний для генерування «рандомізованих» даних для таблиці “Readers”:

**INSERT** **INTO** "Readers" ("ID", "Surname", "Name", "Address")

**SELECT**

row\_number() OVER () + (**SELECT** COALESCE(MAX("ID"), 0) **FROM** "Readers"),

chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int),

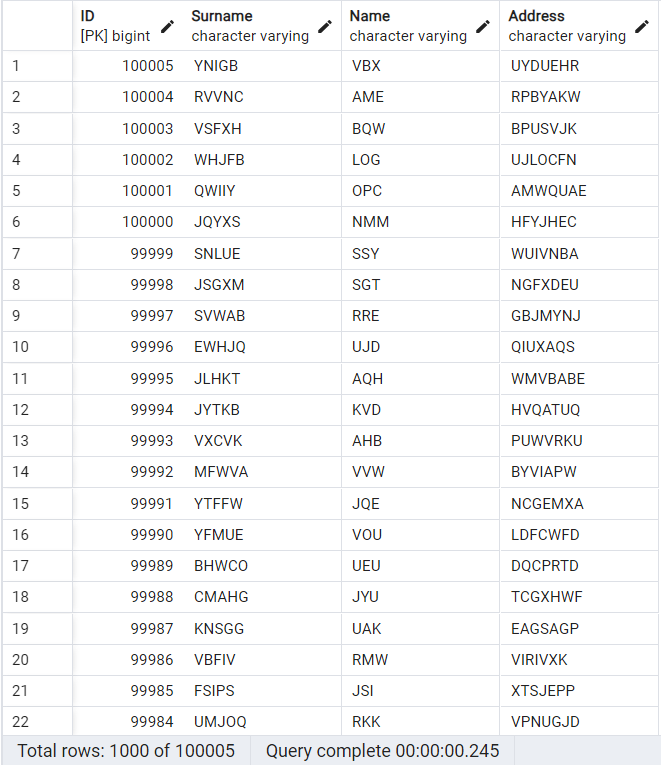
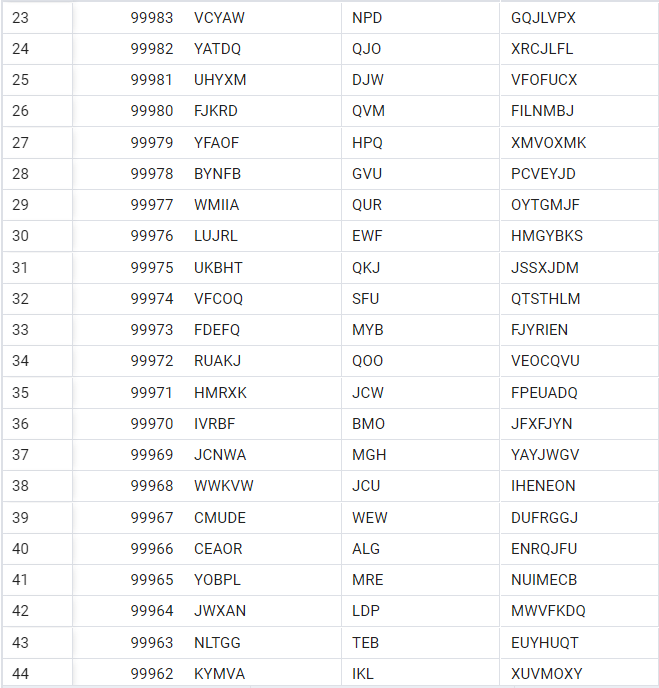
chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int),

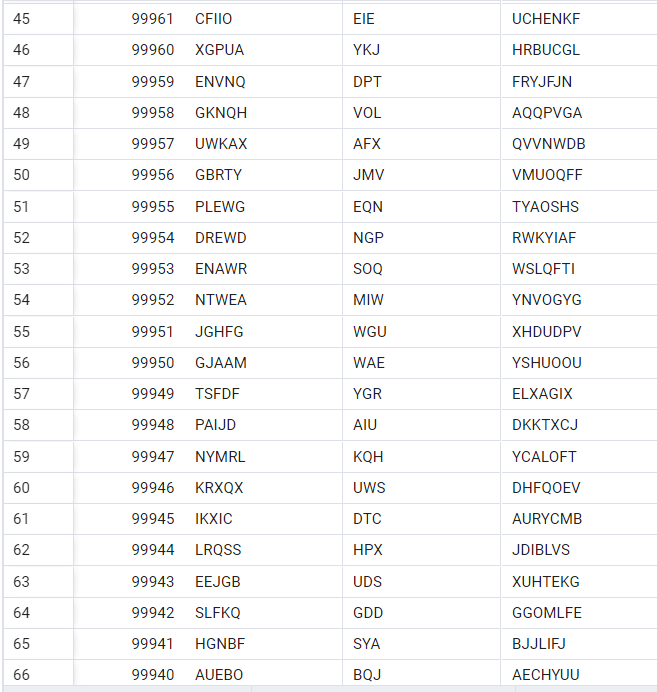
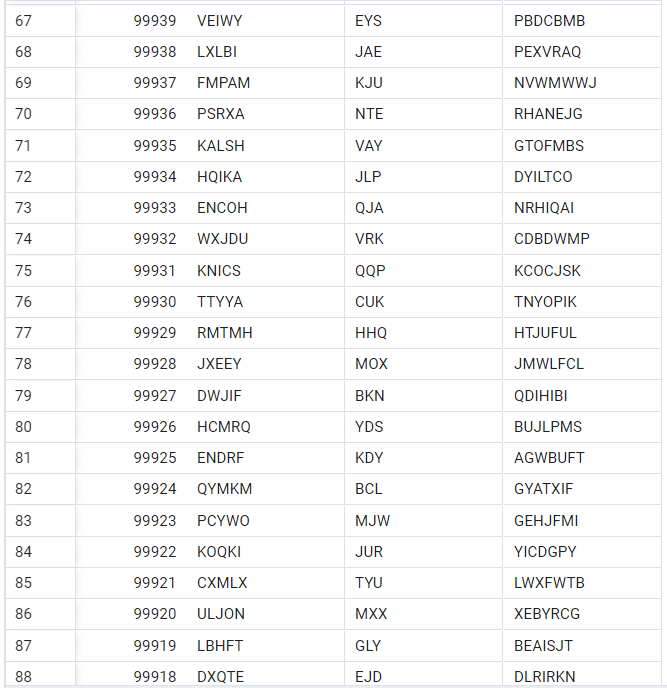
chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int)

**FROM** generate\_series(1, %s);

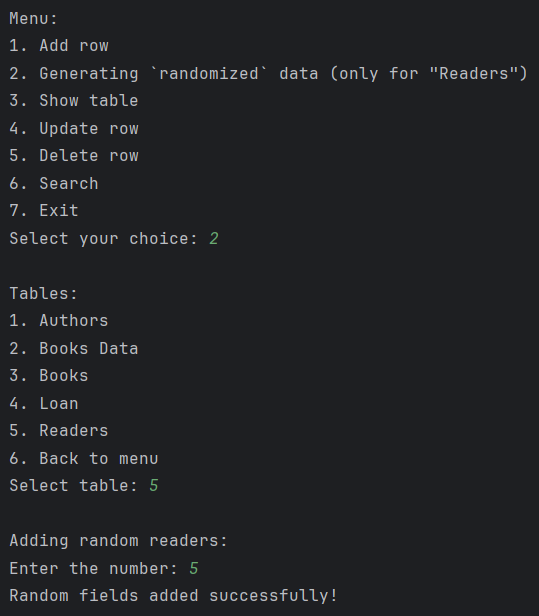
Примітка: %s – це введене користувачем число бажаних рядків для додавання в таблицю “Readers”. Протестуємо:

***Таблиця “Readers” до:***

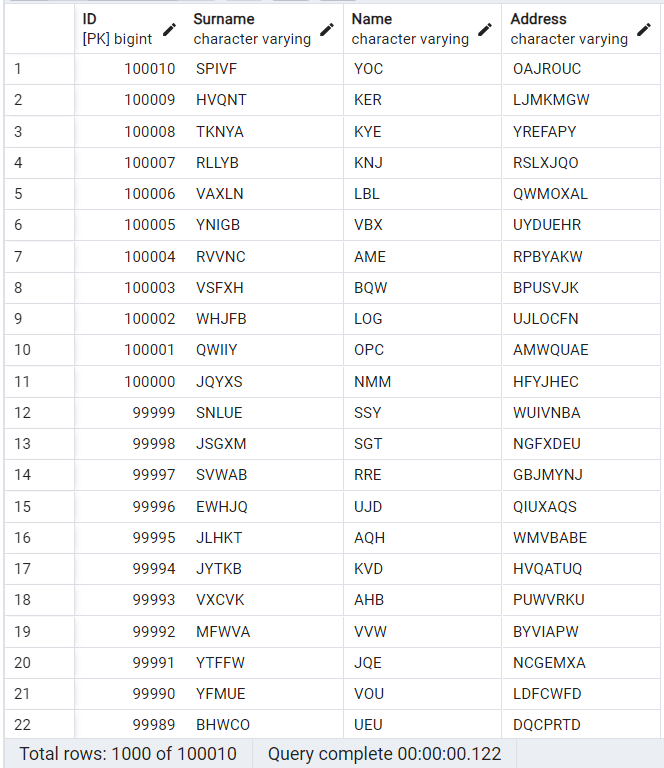
*** ***

*** ***

***Процес додавання «рандомізованих» даних:***

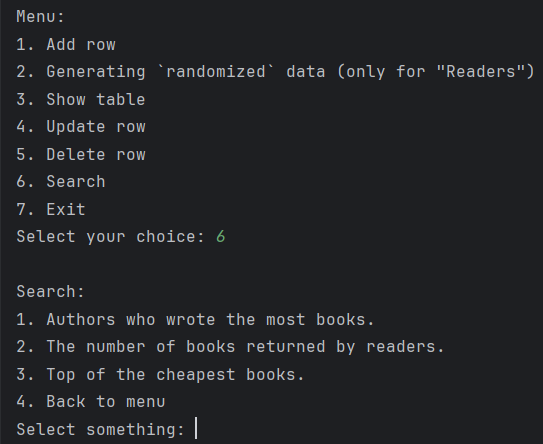
******

***Таблиця “Readers” після:***

****

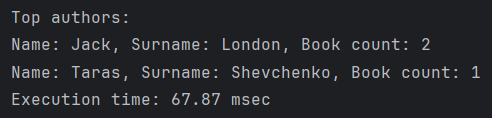
**Завдання 3**

**Пошук даних**

****

***Запит №1***

Authors who wrote the most books. Цей запит створений для отримання авторів, які написали найбільше книг. На виході отримуємо список авторів, сортований за зменшенням написаних книг, з їхніми іменами, прізвищами та відповідно числом книг. Також було засічено час виконання цього запиту; він становив 67.87 мілісекунд.



SQL запит виглядає наступним чином:

**SELECT** "Authors"."Name", "Authors"."Surname", COUNT("Books\_Data"."ISBN") **AS** BookCount

**FROM** "Authors"

**JOIN** "Books\_Data" **ON** "Authors"."ID" = "Books\_Data"."ID\_Author"

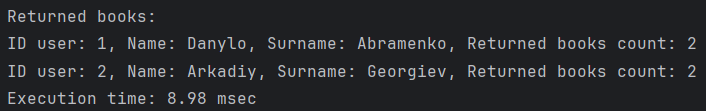
**GROUP** **BY** "Authors"."Name", "Authors"."Surname"

**ORDER** **BY** BookCount **DESC**

**LIMIT** 3;

***Запит №2***

The number of books returned by readers. Цей запит створений для отримання кількості книг, яку повернули читачі. На виході отримуємо список читачів по порядку з їхніми id, іменами, прізвищами та числом повернутих книг. Також було засічено час виконання цього запиту; він становив 8.98 мілісекунд.



SQL запит виглядає наступним чином:

**SELECT** "ID\_User", "Readers"."Name" **AS** UserName, "Readers"."Surname" **AS** UserSurname, COUNT("ID\_Book") **AS** ReturnedBooksCount

**FROM** "Loan"

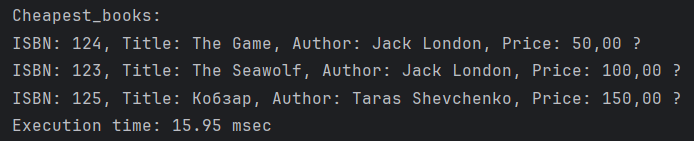
**JOIN** "Readers" **ON** "Loan"."ID\_User" = "Readers"."ID"

**GROUP** **BY** "ID\_User", UserName, UserSurname

**ORDER** **BY** "ID\_User";

***Запит №3***

Top of the cheapest books. Цей запит створений для отримання списку найдешевших книг. На виході отримуємо ТОП із 5 книг з їхніми ISBN, назвами, авторами та цінами. Також було засічено час виконання цього запиту; він становив 15.95 мілісекунд.



SQL запит виглядає наступним чином:

**SELECT** "Books\_Data"."ISBN" **AS** BookID, "Books\_Data"."Name" **AS** BookName, CONCAT("Authors"."Name", ' ', "Authors"."Surname") **AS** AuthorFullName, "Books\_Data"."Price"

**FROM** "Books\_Data"

**JOIN** "Authors" **ON** "Books\_Data"."ID\_Author" = "Authors"."ID"

**ORDER** **BY** "Books\_Data"."Price"

**LIMIT** 5;

**Завдання 4**

**Шаблон MVC**

**MVC** визначає архітектурний шаблон програмування, який включає три основні компоненти: Модель (Model), Вид (View) та Контролер (Controller). Цей шаблон дозволяє розділити логічні частини програми, щоб полегшити розробку, управління та розуміння коду.

Основні компоненти шаблону MVC:

**Model** – представляє клас, що описує логіку використовуваних даних. Клас реалізований у файлі model.py, у ньому відбуваються найважчі процеси (вставка, видалення, оновлення, пошук, рандомізація даних, звернення до бази даних) і після виконаної події відправляє результат до View.

**View** – це консольний інтерфейс, з яким взаємодіє користувач. Відповідає за введення/виведення даних. У програмі це реалізовано за допомогою файлу view.py (клас View та клас Menu).

**Controller** – забезпечує зв'язок між користувачем і системою, поданням і сховищем даних. Він отримує введені користувачем дані і обробляє їх. У програмі це реалізовано у файлі controller.py

Користуючись шаблоном MVC, розробники можуть розділити програмний код на логічно зв'язані компоненти, що полегшує розуміння, тестування та зміну програми. Це особливо корисно для великих проектів, де структурованість і підтримка коду грають важливу роль.

**Код програми**

**main.py**

**from** controller **import** Controller

**if** \_\_name\_\_ == "\_\_main\_\_":

controller = Controller()

controller.run()

**model.py**

**import** psycopg2

**class** Model:

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

self.conn = psycopg2.connect(

dbname='library',

user='postgres',

password='1111',

host='localhost',

port=3000

)

**def** add\_author(self, id, surname, name, country):

c = self.conn.cursor()

c.execute('INSERT INTO public."Authors"("ID", "Surname", "Name", "Country") VALUES(%s, %s, %s, %s);', (id, surname, name, country))

self.conn.commit()

**def** add\_book\_data(self, isbn, id\_author, name, year, price):

c = self.conn.cursor()

c.execute('INSERT INTO public."Books\_Data"("ISBN", "ID\_Author", "Name", "Year", "Price") VALUES(%s, %s, %s, %s, %s);', (isbn, id\_author, name, year, price))

self.conn.commit()

**def** add\_book(self, id, id\_book):

c = self.conn.cursor()

c.execute('INSERT INTO public."Books"("ID", "ID\_Book") VALUES(%s, %s);', (id, id\_book))

self.conn.commit()

**def** add\_loan(self, id, id\_book, id\_user, loan\_date, return\_date):

c = self.conn.cursor()

c.execute('INSERT INTO public."Loan"("ID", "ID\_Book", "ID\_User", "Loan\_Date", "Return\_Date") VALUES(%s, %s, %s, %s, %s);', (id, id\_book, id\_user, loan\_date, return\_date))

self.conn.commit()

**def** add\_reader(self, id, surname, name, adress):

c = self.conn.cursor()

c.execute('INSERT INTO public."Readers"("ID", "Surname", "Name", "Address") VALUES(%s, %s, %s, %s);', (id, surname, name, adress))

self.conn.commit()

**def** get\_authors(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM public."Authors";')

**return** c.fetchall()

**def** get\_books\_data(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM public."Books\_Data";')

**return** c.fetchall()

**def** get\_books(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM public."Books";')

**return** c.fetchall()

**def** get\_loans(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM public."Loan";')

**return** c.fetchall()

**def** get\_readers(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM public."Readers";')

**return** c.fetchall()

**def** update\_author(self, author\_id, surname, name, country, id):

c = self.conn.cursor()

c.execute('UPDATE public."Authors" SET "ID"=%s, "Surname"=%s, "Name"=%s, "Country"=%s WHERE "ID"=%s', (author\_id, surname, name, country, id))

self.conn.commit()

**def** update\_book\_data(self, isbn, id\_author, name, year, price, id):

c = self.conn.cursor()

c.execute('UPDATE public."Books\_Data" SET "ISBN"=%s, "ID\_Author"=%s, "Name"=%s, "Year"=%s, "Price"=%s WHERE "ISBN"=%s', (isbn, id\_author, name, year, price, id))

self.conn.commit()

**def** update\_book(self, book\_id, id\_book, id):

c = self.conn.cursor()

c.execute('UPDATE public."Books" SET "ID"=%s, "ID\_Book"=%s WHERE "ID"=%s', (book\_id, id\_book, id))

self.conn.commit()

**def** update\_loan(self, loan\_id, id\_book, id\_user, loan\_date, return\_date, id):

c = self.conn.cursor()

c.execute('UPDATE public."Loan" SET "ID"=%s, "ID\_Book"=%s, "ID\_User"=%s, "Loan\_Date"=%s, "Return\_Date"=%s WHERE "ID"=%s', (loan\_id, id\_book, id\_user, loan\_date, return\_date, id))

self.conn.commit()

**def** update\_reader(self, reader\_id, surname, name, adress, id):

c = self.conn.cursor()

c.execute('UPDATE public."Readers" SET "ID"=%s, "Surname"=%s, "Name"=%s, "Address"=%s WHERE "ID"=%s', (reader\_id, surname, name, adress, id))

self.conn.commit()

**def** delete\_author(self, id):

c = self.conn.cursor()

c.execute('DELETE FROM public."Authors" WHERE "ID"=%s', (id,))

self.conn.commit()

**def** delete\_book\_data(self, isbn):

c = self.conn.cursor()

c.execute('DELETE FROM public."Books\_Data" WHERE "ISBN"=%s', (isbn,))

self.conn.commit()

**def** delete\_book(self, id):

c = self.conn.cursor()

c.execute('DELETE FROM public."Books" WHERE "ID"=%s', (id,))

self.conn.commit()

**def** delete\_loan(self, id):

c = self.conn.cursor()

c.execute('DELETE FROM public."Loan" WHERE "ID"=%s', (id,))

self.conn.commit()

**def** delete\_reader(self, id):

c = self.conn.cursor()

c.execute('DELETE FROM public."Readers" WHERE "ID"=%s', (id,))

self.conn.commit()

**def** get\_top\_authors(self):

c = self.conn.cursor()

c.execute('SELECT "Authors"."Name", "Authors"."Surname", COUNT("Books\_Data"."ISBN") AS BookCount '

'FROM "Authors" '

'JOIN "Books\_Data" ON "Authors"."ID" = "Books\_Data"."ID\_Author" '

'GROUP BY "Authors"."Name", "Authors"."Surname" '

'ORDER BY BookCount DESC '

'LIMIT 3;')

**return** c.fetchall()

**def** get\_returned\_books(self):

c = self.conn.cursor()

c.execute('''

SELECT "ID\_User", "Readers"."Name" AS UserName, "Readers"."Surname" AS UserSurname, COUNT("ID\_Book") AS ReturnedBooksCount

FROM "Loan"

JOIN "Readers" ON "Loan"."ID\_User" = "Readers"."ID"

GROUP BY "ID\_User", UserName, UserSurname

ORDER BY "ID\_User";

''')

**return** c.fetchall()

**def** get\_cheapest\_books(self):

c = self.conn.cursor()

c.execute('''

SELECT "Books\_Data"."ISBN" AS BookID, "Books\_Data"."Name" AS BookName, CONCAT("Authors"."Name", ' ', "Authors"."Surname") AS AuthorFullName, "Books\_Data"."Price"

FROM "Books\_Data"

JOIN "Authors" ON "Books\_Data"."ID\_Author" = "Authors"."ID"

ORDER BY "Books\_Data"."Price"

LIMIT 5;

''')

**return** c.fetchall()

**def** add\_random\_fields(self, number):

c = self.conn.cursor()

c.execute("""

INSERT INTO "Readers" ("ID", "Surname", "Name", "Address")

SELECT

row\_number() OVER () + (SELECT COALESCE(MAX("ID"), 0) FROM "Readers"),

chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int),

chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int),

chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int)

FROM generate\_series(1, %s);

""", (number,))

self.conn.commit()

**controller.py**

**import** time

**from** model **import** Model

**from** view **import** View

**class** Controller:

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

self.model = Model()

self.view = View()

**def** run(self):

**while** True:

choice = self.view.show\_menu()

**if** choice == '7':

**break**

**if** choice == '6':

self.process\_search\_option()

**elif** choice **in** ['1', '2', '3', '4', '5']:

self.process\_menu\_choice(choice)

**else**:

self.view.show\_message("Wrong choice. Try again.")

**def** process\_menu\_choice(self, choice):

**while** True:

table = self.view.show\_tables()

**if** table == '6':

**break**

**if** choice == '1':

self.process\_add\_option(table)

**elif** choice == '2':

self.process\_add\_random\_option(table)

**elif** choice == '3':

self.process\_view\_option(table)

**elif** choice == '4':

self.process\_update\_option(table)

**elif** choice == '5':

self.process\_delete\_option(table)

**else**:

self.view.show\_message("Wrong choice. Try again.")

**def** process\_add\_option(self, table):

**if** table == '1':

self.view.show\_message("\nAdding author:")

self.add\_author()

**elif** table == '2':

self.view.show\_message("\nAdding book data:")

self.add\_book\_data()

**elif** table == '3':

self.view.show\_message("\nAdding book:")

self.add\_book()

**elif** table == '4':

self.view.show\_message("\nAdding loan:")

self.add\_loan()

**elif** table == '5':

self.view.show\_message("\nAdding reader:")

self.add\_reader()

**else**:

self.view.show\_message("Wrong choice. Try again.")

**def** process\_add\_random\_option(self, table):

**if** table == '5':

self.view.show\_message("\nAdding random readers:")

self.add\_random\_fields()

**else**:

self.view.show\_message("Wrong choice. Try again.")

**def** process\_view\_option(self, table):

**if** table == '1':

self.show\_authors()

**elif** table == '2':

self.show\_books\_data()

**elif** table == '3':

self.show\_books()

**elif** table == '4':

self.show\_loans()

**elif** table == '5':

self.show\_readers()

**elif** table == '6':

self.view.show\_menu()

**else**:

self.view.show\_message("Wrong choice. Try again.")

**def** process\_update\_option(self, table):

**if** table == '1':

self.view.show\_message("\nUpdating author:")

self.update\_author()

**elif** table == '2':

self.view.show\_message("\nUpdating book data:")

self.update\_book\_data()

**elif** table == '3':

self.view.show\_message("\nUpdating book:")

self.update\_book()

**elif** table == '4':

self.view.show\_message("\nUpdating loan:")

self.update\_loan()

**elif** table == '5':

self.view.show\_message("\nUpdating reader:")

self.update\_reader()

**else**:

self.view.show\_message("Wrong choice. Try again.")

**def** process\_delete\_option(self, table):

**if** table == '1':

self.view.show\_message("\nDeleting author:")

self.delete\_author()

**elif** table == '2':

self.view.show\_message("\nDeleting book data:")

self.delete\_book\_data()

**elif** table == '3':

self.view.show\_message("\nDeleting book:")

self.delete\_book()

**elif** table == '4':

self.view.show\_message("\nDeleting loan:")

self.delete\_loan()

**elif** table == '5':

self.view.show\_message("\nDeleting reader:")

self.delete\_reader()

**else**:

self.view.show\_message("Wrong choice. Try again.")

**def** process\_search\_option(self):

option = self.view.show\_search()

**if** option == '1':

start\_time = time.time()

self.show\_top\_authors()

end\_time = time.time()

elapsed\_time = (end\_time - start\_time) \* 1000

**print**(f"Execution time: {elapsed\_time:.2f} msec")

**elif** option == '2':

start\_time = time.time()

self.show\_returned\_books()

end\_time = time.time()

elapsed\_time = (end\_time - start\_time) \* 1000

**print**(f"Execution time: {elapsed\_time:.2f} msec")

**elif** option == '3':

start\_time = time.time()

self.show\_cheapest\_books()

end\_time = time.time()

elapsed\_time = (end\_time - start\_time) \* 1000

**print**(f"Execution time: {elapsed\_time:.2f} msec")

**else**:

self.view.show\_menu()

**def** add\_author(self):

**try**:

id, surname, name, country = self.view.get\_author\_input()

self.model.add\_author(id, surname, name, country)

self.view.show\_message("Author added successfully!")

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

self.view.show\_message(f"Something went wrong: {e}")

**def** add\_book\_data(self):

**try**:

isbn, id\_author, name, year, price = self.view.get\_book\_data\_input()

self.model.add\_book\_data(isbn, id\_author, name, year, price)

self.view.show\_message("Book Data added successfully!")

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

self.view.show\_message(f"Something went wrong: {e}")

**def** add\_book(self):

**try**:

id, id\_book = self.view.get\_book\_input()

self.model.add\_book(id, id\_book)

self.view.show\_message("Book added successfully!")

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

self.view.show\_message(f"Something went wrong: {e}")

**def** add\_loan(self):

**try**:

id, id\_book, id\_user, loan\_date, return\_date = self.view.get\_loan\_input()

self.model.add\_loan(id, id\_book, id\_user, loan\_date, return\_date)

self.view.show\_message("Loan added successfully!")

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

self.view.show\_message(f"Something went wrong: {e}")

**def** add\_reader(self):

**try**:

id, surname, name, adress = self.view.get\_reader\_input()

self.model.add\_reader(id, surname, name, adress)

self.view.show\_message("Reader added successfully!")

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

self.view.show\_message(f"Something went wrong: {e}")

**def** show\_authors(self):

**try**:

authors = self.model.get\_authors()

self.view.show\_authors(authors)

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

self.view.show\_message(f"Something went wrong: {e}")

**def** show\_books\_data(self):

**try**:

books\_data = self.model.get\_books\_data()

self.view.show\_books\_data(books\_data)

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

self.view.show\_message(f"Something went wrong: {e}")

**def** show\_books(self):

**try**:

books = self.model.get\_books()

self.view.show\_books(books)

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

self.view.show\_message(f"Something went wrong: {e}")

**def** show\_loans(self):

**try**:

loans = self.model.get\_loans()

self.view.show\_loans(loans)

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

self.view.show\_message(f"Something went wrong: {e}")

**def** show\_readers(self):

**try**:

readers = self.model.get\_readers()

self.view.show\_readers(readers)

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

self.view.show\_message(f"Something went wrong: {e}")

**def** update\_author(self):

**try**:

id = self.view.get\_author\_id()

author\_id, surname, name, country = self.view.get\_author\_input()

self.model.update\_author(author\_id, surname, name, country, id)

self.view.show\_message("Author updated successfully!")

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

self.view.show\_message(f"Something went wrong: {e}")

**def** update\_book\_data(self):

**try**:

id = self.view.get\_book\_data\_id()

isbn, id\_author, name, year, price = self.view.get\_book\_data\_input()

self.model.update\_book\_data(isbn, id\_author, name, year, price, id)

self.view.show\_message("Book Data updated successfully!")

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

self.view.show\_message(f"Something went wrong: {e}")

**def** update\_book(self):

**try**:

id = self.view.get\_book\_id()

book\_id, id\_book = self.view.get\_book\_input()

self.model.update\_book(book\_id, id\_book, id)

self.view.show\_message("Book updated successfully!")

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

self.view.show\_message(f"Something went wrong: {e}")

**def** update\_loan(self):

**try**:

id = self.view.get\_loan\_id()

loan\_id, id\_book, id\_user, loan\_date, return\_date = self.view.get\_loan\_input()

self.model.update\_loan(loan\_id, id\_book, id\_user, loan\_date, return\_date, id)

self.view.show\_message("Loan updated successfully!")

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

self.view.show\_message(f"Something went wrong: {e}")

**def** update\_reader(self):

**try**:

id = self.view.get\_reader\_id()

reader\_id, surname, name, adress = self.view.get\_reader\_input()

self.model.update\_reader(reader\_id, surname, name, adress, id)

self.view.show\_message("Reader updated successfully!")

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

self.view.show\_message(f"Something went wrong: {e}")

**def** delete\_author(self):

**try**:

id = self.view.get\_author\_id()

self.model.delete\_author(id)

self.view.show\_message("Author deleted successfully!")

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

self.view.show\_message(f"Something went wrong: {e}")

**def** delete\_book\_data(self):

**try**:

isbn = self.view.get\_book\_data\_id()

self.model.delete\_book\_data(isbn)

self.view.show\_message("Book Data deleted successfully!")

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

self.view.show\_message(f"Something went wrong: {e}")

**def** delete\_book(self):

**try**:

id = self.view.get\_book\_id()

self.model.delete\_book(id)

self.view.show\_message("Book deleted successfully!")

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

self.view.show\_message(f"Something went wrong: {e}")

**def** delete\_loan(self):

**try**:

id = self.view.get\_loan\_id()

self.model.delete\_loan(id)

self.view.show\_message("Loan deleted successfully!")

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

self.view.show\_message(f"Something went wrong: {e}")

**def** delete\_reader(self):

**try**:

id = self.view.get\_reader\_id()

self.model.delete\_reader(id)

self.view.show\_message("Reader deleted successfully!")

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

self.view.show\_message(f"Something went wrong: {e}")

**def** show\_top\_authors(self):

**try**:

top\_authors = self.model.get\_top\_authors()

self.view.show\_top\_authors(top\_authors)

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

self.view.show\_message(f"Something went wrong: {e}")

**def** show\_returned\_books(self):

**try**:

returned\_books = self.model.get\_returned\_books()

self.view.show\_returned\_books(returned\_books)

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

self.view.show\_message(f"Something went wrong: {e}")

**def** show\_cheapest\_books(self):

**try**:

cheapest\_books = self.model.get\_cheapest\_books()

self.view.show\_cheapest\_books(cheapest\_books)

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

self.view.show\_message(f"Something went wrong: {e}")

**def** add\_random\_fields(self):

**try**:

number = self.view.get\_number()

self.model.add\_random\_fields(number)

self.view.show\_message("Random fields added successfully!")

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

self.view.show\_message(f"Something went wrong: {e}")

**view.py**

**from** datetime **import** datetime

**class** View:

**def** show\_menu(self):

self.show\_message("\nMenu:")

self.show\_message("1. Add row")

self.show\_message('2. Generating `randomized` data (only for "Readers")')

self.show\_message("3. Show table")

self.show\_message("4. Update row")

self.show\_message("5. Delete row")

self.show\_message("6. Search")

self.show\_message("7. Exit")

choice = input("Select your choice: ")

**return** choice

**def** show\_tables(self):

self.show\_message("\nTables:")

self.show\_message("1. Authors")

self.show\_message("2. Books Data")

self.show\_message("3. Books")

self.show\_message("4. Loan")

self.show\_message("5. Readers")

self.show\_message("6. Back to menu")

table = input("Select table: ")

**return** table

**def** show\_search(self):

self.show\_message("\nSearch:")

self.show\_message("1. Authors who wrote the most books.")

self.show\_message("2. The number of books returned by readers.")

self.show\_message("3. Top of the cheapest books.")

self.show\_message("4. Back to menu")

choice = input("Select something: ")

**return** choice

**def** show\_authors(self, authors):

**print**("\nAuthors:")

**for** author **in** authors:

**print**(f"ID: {author[0]}, Surname: {author[1]}, Name: {author[2]}, Country: {author[3]}")

**def** show\_books\_data(self, books\_data):

**print**("\nBooks Data:")

**for** book\_data **in** books\_data:

**print**(f"ISBN: {book\_data[0]}, ID\_Author: {book\_data[1]}, Name: {book\_data[2]}, Year: {book\_data[3]}, Price: {book\_data[4]}")

**def** show\_books(self, books):

**print**("\nBooks:")

**for** book **in** books:

**print**(f"ID: {book[0]}, ID\_Book: {book[1]}")

**def** show\_loans(self, loans):

**print**("\nLoans:")

**for** loan **in** loans:

**print**(f"ID: {loan[0]}, ID\_Book: {loan[1]}, ID\_User: {loan[2]}, Loan\_Date: {loan[3]}, Return\_Date: {loan[4]}")

**def** show\_readers(self, readers):

**print**("\nReaders:")

**for** reader **in** readers:

**print**(f"ID: {reader[0]}, Surname: {reader[1]}, Name: {reader[2]}, Adress: {reader[3]}")

**def** show\_top\_authors(self, rows):

**print**("\nTop authors:")

**for** row **in** rows:

**print**(

f"Name: {row[0]}, Surname: {row[1]}, Book count: {row[2]}")

**def** show\_returned\_books(self, rows):

**print**("\nReturned books:")

**for** row **in** rows:

**print**(

f"ID user: {row[0]}, Name: {row[1]}, Surname: {row[2]}, Returned books count: {row[3]}")

**def** show\_cheapest\_books(self, rows):

**print**("\nCheapest\_books:")

**for** row **in** rows:

**print**(

f"ISBN: {row[0]}, Title: {row[1]}, Author: {row[2]}, Price: {row[3]}")

**def** get\_author\_input(self):

**while** True:

**try**:

id = input("Enter ID: ")

**if** id.strip():

id = int(id)

**break**

**else**:

**print**("ID cannot be empty.")

**except** ValueError:

**print**("It must be a number.")

**while** True:

**try**:

surname = input("Enter surname: ")

**if** surname.strip():

**break**

**else**:

**print**("Surname cannot be empty.")

**except** ValueError:

**print**("It must be a string.")

**while** True:

**try**:

name = input("Enter name: ")

**if** name.strip():

**break**

**else**:

**print**("Name cannot be empty.")

**except** ValueError:

**print**("It must be a string.")

**while** True:

**try**:

country = input("Enter country: ")

**if** country.strip():

**break**

**else**:

**print**("Country cannot be empty.")

**except** ValueError:

**print**("It must be a string.")

**return** id, surname, name, country

**def** get\_book\_data\_input(self):

**while** True:

**try**:

isbn = input("Enter ISBN: ")

**if** isbn.strip():

isbn = int(isbn)

**break**

**else**:

**print**("ISBN cannot be empty.")

**except** ValueError:

**print**("It must be a number.")

**while** True:

**try**:

id\_author = input("Enter ID author: ")

**if** id\_author.strip():

id\_author = int(id\_author)

**break**

**else**:

**print**("ID author cannot be empty.")

**except** ValueError:

**print**("It must be a number.")

**while** True:

**try**:

name = input("Enter name: ")

**if** name.strip():

**break**

**else**:

**print**("Name cannot be empty.")

**except** ValueError:

**print**("It must be a string.")

**while** True:

**try**:

year\_input = input("Enter year (YYYY-MM-DD HH:MM:SS+TZ): ")

**if** year\_input.strip():

year = datetime.strptime(year\_input, "%Y-%m-%d %H:%M:%S%z")

**break**

**else**:

**print**("Year cannot be empty.")

**except** ValueError:

**print**("Invalid year format. Please use YYYY-MM-DD HH:MM:SS+TZ.")

**while** True:

**try**:

price\_input = input("Enter price: ")

**if** price\_input.strip():

price = float(price\_input)

**break**

**else**:

**print**("Price cannot be empty.")

**except** ValueError:

**print**("Price must be a valid number.")

**return** isbn, id\_author, name, year, price

**def** get\_book\_input(self):

**while** True:

**try**:

id = input("Enter ID: ")

**if** id.strip():

id = int(id)

**break**

**else**:

**print**("ID cannot be empty.")

**except** ValueError:

**print**("It must be a number.")

**while** True:

**try**:

id\_book = input("Enter ID book: ")

**if** id\_book.strip():

id\_book = int(id\_book)

**break**

**else**:

**print**("ID book cannot be empty.")

**except** ValueError:

**print**("It must be a number.")

**return** id, id\_book

**def** get\_loan\_input(self):

**while** True:

**try**:

id = input("Enter ID: ")

**if** id.strip():

id = int(id)

**break**

**else**:

**print**("ID cannot be empty.")

**except** ValueError:

**print**("It must be a number.")

**while** True:

**try**:

id\_book = input("Enter ID book: ")

**if** id\_book.strip():

id\_book = int(id\_book)

**break**

**else**:

**print**("ID book cannot be empty.")

**except** ValueError:

**print**("It must be a number.")

**while** True:

**try**:

id\_user = input("Enter ID user: ")

**if** id\_user.strip():

id\_user = int(id\_user)

**break**

**else**:

**print**("ID user cannot be empty.")

**except** ValueError:

**print**("It must be a number.")

**while** True:

**try**:

loan\_date\_input = input("Enter loan date (YYYY-MM-DD HH:MM:SS+TZ): ")

**if** loan\_date\_input.strip():

loan\_date = datetime.strptime(loan\_date\_input, "%Y-%m-%d %H:%M:%S%z")

**break**

**else**:

**print**("Loan date cannot be empty.")

**except** ValueError:

**print**("Invalid year format. Please use YYYY-MM-DD HH:MM:SS+TZ.")

**while** True:

**try**:

return\_date\_input = input("Enter return date (YYYY-MM-DD HH:MM:SS+TZ): ")

**if** return\_date\_input.strip():

return\_date = datetime.strptime(return\_date\_input, "%Y-%m-%d %H:%M:%S%z")

**break**

**else**:

**print**("Return date cannot be empty.")

**except** ValueError:

**print**("Invalid year format. Please use YYYY-MM-DD HH:MM:SS+TZ.")

**return** id, id\_book, id\_user, loan\_date, return\_date

**def** get\_reader\_input(self):

**while** True:

**try**:

id = input("Enter ID: ")

**if** id.strip():

id = int(id)

**break**

**else**:

**print**("ID cannot be empty.")

**except** ValueError:

**print**("It must be a number.")

**while** True:

**try**:

surname = input("Enter surname: ")

**if** surname.strip():

**break**

**else**:

**print**("Surname cannot be empty.")

**except** ValueError:

**print**("It must be a string.")

**while** True:

**try**:

name = input("Enter name: ")

**if** name.strip():

**break**

**else**:

**print**("Name cannot be empty.")

**except** ValueError:

**print**("It must be a string.")

**while** True:

**try**:

adress = input("Enter adress: ")

**if** adress.strip():

**break**

**else**:

**print**("Adress cannot be empty.")

**except** ValueError:

**print**("It must be a string.")

**return** id, surname, name, adress

**def** get\_author\_id(self):

**while** True:

**try**:

id = int(input("Enter author ID: "))

**break**

**except** ValueError:

**print**("It must be a number.")

**return** id

**def** get\_book\_data\_id(self):

**while** True:

**try**:

isbn = int(input("Enter ISBN: "))

**break**

**except** ValueError:

**print**("It must be a number.")

**return** isbn

**def** get\_book\_id(self):

**while** True:

**try**:

id = int(input("Enter book ID: "))

**break**

**except** ValueError:

**print**("It must be a number.")

**return** id

**def** get\_loan\_id(self):

**while** True:

**try**:

id = int(input("Enter loan ID: "))

**break**

**except** ValueError:

**print**("It must be a number.")

**return** id

**def** get\_reader\_id(self):

**while** True:

**try**:

id = int(input("Enter reader ID: "))

**break**

**except** ValueError:

**print**("It must be a number.")

**return** id

**def** show\_message(self, message):

**print**(message)

**def** get\_number(self):

**while** True:

**try**:

number = int(input("Enter the number: "))

**break**

**except** ValueError:

**print**("It must be a number.")

**return** number