

**Національний технічний університет України
“Київський політехнічний інститут” імені Ігоря
Сікорського**

Факультет прикладної математики

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

ЛАБОРАТОРНА РОБОТА №2

з дисципліни

“Бази Даних Та Засоби Управління”

**ТЕМА: “Створення додатку бази даних, орієнтованого на взаємодію з
СУБД PostgreSQL”**

Група: KB-03

Виконав: Донченко І. Ю.

Оцінка:

Київ – 2022

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).

Вимоги до звіту у форматі PDF (у електронній формі)

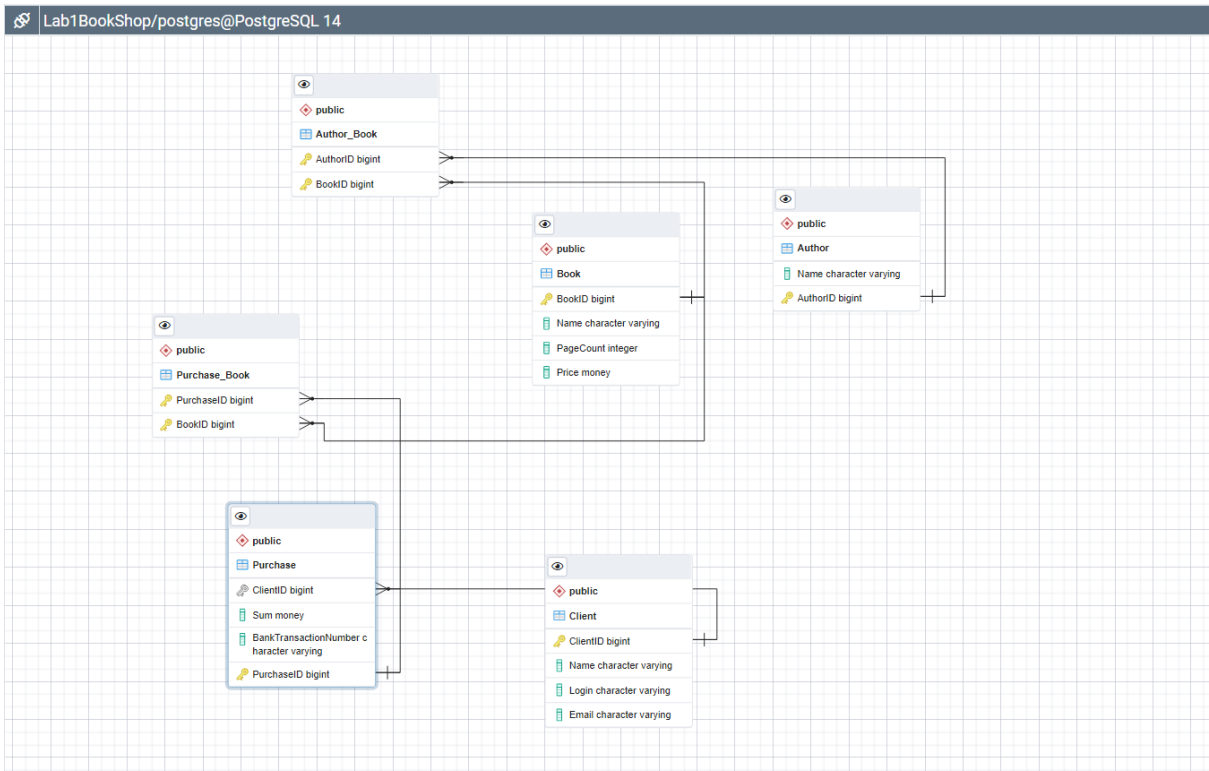
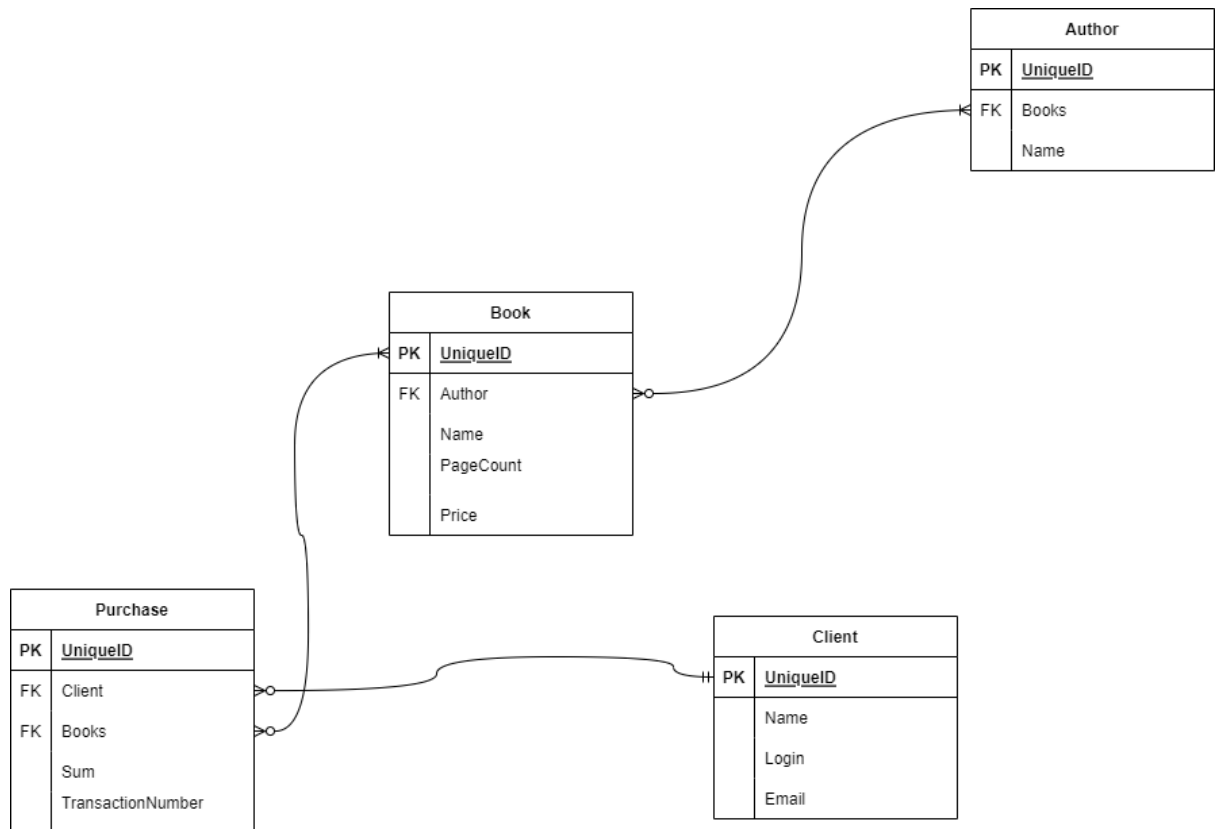
Загальні вимоги

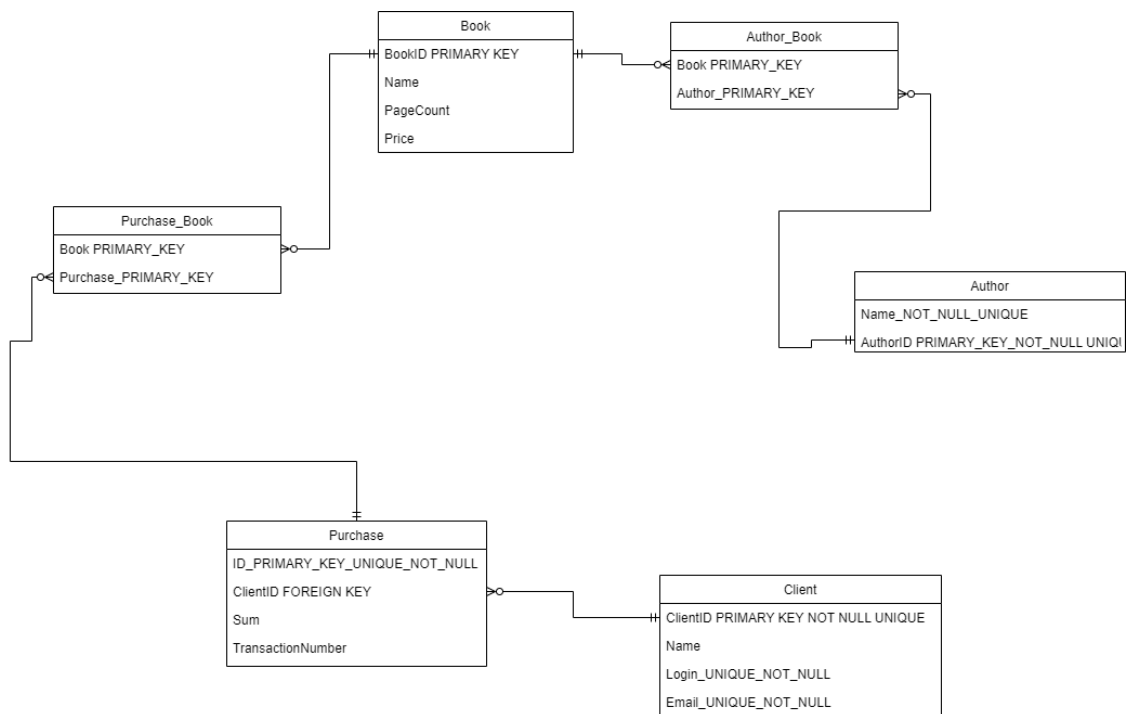
- титульний аркуш, завдання, URL репозиторію з вихідним кодом та відповіді на вимоги до звітування щодо пунктів 1-4 деталізованого завдання (див. нижче);

<https://github.com/diyt2f/DBLabs1-3>

- діаграму сутність-зв'язок та структуру бази даних з лабораторної роботи №1, а також короткий опис бази даних;

Є чотири об'єкти: книга, автор, покупка, і клієнт, кожному з яких відповідає таблиця, є дві додаткові таблиці: автор_книга і покупка_книга. Книга може мати багато авторів, і покупка може мати багато книг, покупка має лише одного клієнта. Є ціна покупки, і ціна книг, вони зберігаються окремо, оскільки ціна книги може змінитися, а ціна покупки в такому випадку має залишитися такою самою.





- схему меню користувача з описом функціональності кожного пункту;

Розглянемо функцію `open_menu()` меню: бачимо, що спочатку потрібно ввести слово команди `read`, `add`, `update`, `search`, `generate`, потім в залежності від команди ввести `book`, `author`, `purchase`, `client` (іноді доступні і інші наприклад `author_book`), потім аргументи. Щоб прочитати данні про книгу із `ID == 15`, вводимо `read book 15`. Приклади інших запитів: `update book 15`, `generate books 1000`, `add client`, `add author_book`, `search book 10 25`, `search book Test`.

```

def open_menu():
    while 1:
        i = input('Input: ')
        i = i.upper()
        i = i.split()

        if i[0] == 'READ':
            View.display('Reading...')

            if i[1] == "AUTHOR":

```

```
        result = Model.get_author(int(i[2]))

        View.display_author(result)

    elif i[1] == "BOOK":

        result = Model.get_book(int(i[2]))

        View.display_book(result)

    elif i[1] == "PURCHASE":

        result = Model.get_purchase(int(i[2]))

        View.display_purchase(result)

    elif i[1] == "CLIENT":

        result = Model.get_client(int(i[2]))

        View.display_client(result)

    elif i[1] == "BOOKS_BY_PURCHASE_ID":

        result = Model.get_books_by_purchase_id(int(i[2]))

        View.display_books(result)

    elif i[1] == "AUTHORS_BY_BOOK_ID":

        result = Model.get_authors_by_book_id(int(i[2]))

        View.display_authors(result)

elif i[0] == 'ADD':

    View.display('Adding...')

    if i[1] == "AUTHOR":

        View.display('author_id/author_name')

        input1 = View.input_many()

        View.display(input1)

        Model.add_author(input1[0], input1[1])

    elif i[1] == "BOOK":

        View.display('book_id/book_name/page_count/book_price')

        input1 = View.input_many()
```

```

        Model.add_book(input1[0], input1[1], input1[2],
input1[3])

        elif i[1] == "PURCHASE":

View.display('purchase_id/client_id/sum/transaction_number')

            input1 = View.input_many()

            Model.add_purchase(input1[0], input1[1], input1[2],
input1[3])

        elif i[1] == "CLIENT":

            View.display('client_id/name/login/email')

            input1 = View.input_many()

            Model.add_client(input1[0], input1[1], input1[2],
input1[3])

        elif i[1] == "AUTHOR_BOOK":

            View.display('author_id/book_id')

            input1 = View.input_many()

            Model.add_author_book_pair(input1[0], input1[1])

        elif i[1] == "PURCHASE_BOOK":

            View.display('purchase_id/book_id')

            input1 = View.input_many()

            Model.add_purchase_book_pair(input1[0], input1[1])

    elif i[0] == 'UPDATE':

        View.display('Updating...')

        if i[1] == "AUTHOR":

            View.display('author_id/author_name')

            input1 = View.input_many()

            View.display(input1)

            Model.update_author(input1[0], input1[1])

```

```
        elif i[1] == "BOOK":

            View.display('book_id/book_name/page_count/book_price')

            input1 = View.input_many()

            Model.update_book(input1[0], input1[1], input1[2],
input1[3])

        elif i[1] == "PURCHASE":

View.display('purchase_id/client_id/sum/transaction_number')

            input1 = View.input_many()

            Model.update_purchase(input1[0], input1[1], input1[2],
input1[3])

        elif i[1] == "CLIENT":

            View.display('client_id/name/login/email')

            input1 = View.input_many()

            Model.update_client(input1[0], input1[1], input1[2],
input1[3])

    elif i[0] == 'DELETE':

        View.display('Deleting...')

        if i[1] == "AUTHOR":

            Model.delete_author(int(i[2]))

        elif i[1] == "BOOK":

            Model.delete_book(int(i[2]))

        elif i[1] == "PURCHASE":

            Model.delete_purchase(int(i[2]))

        elif i[1] == "CLIENT":

            Model.delete_client(int(i[2]))

        elif i[1] == "PURCHASE_BOOK":

Model.delete_purchase_book_pairs_by_purchase_id(int(i[2]))
```

```
        elif i[1] == "AUTHOR_BOOK":

            Model.delete_author_book_pairs_by_book_id(int(i[2]))

elif i[0] == "GENERATE":

    View.display("Generating...")

    if i[1] == "AUTHORS":

        Model.add_random_authors(int(i[2]))

    elif i[1] == "BOOKS":

        Model.add_random_books(int(i[2]))

    elif i[1] == "PURCHASES":

        Model.add_random_purchases(int(i[2]))

    elif i[1] == "CLIENTS":

        Model.add_random_clients(int(i[2]))

    elif i[1] == "AUTHOR_BOOK_PAIRS":

        Model.add_random_author_book_pairs(int(i[2]))

    elif i[1] == "PURCHASE_BOOK_PAIRS":

        Model.add_random_purchase_book_pairs(int(i[2]))

elif i[0] == 'SEARCH':

    View.display('Searching...')

    if i[1] == "CLIENT":

        result = Model.search_client(i[2])

        View.display(result)

    elif i[1] == "AUTHOR":

        result = Model.search_author(i[2])

        View.display(result)

    elif i[1] == "PURCHASE":

        result = Model.search_purchase(i[2])

        View.display(result)
```



```
elif i[1] == "BOOK":  
  
    if len(i) == 3:  
  
        result = Model.search_book_by_name(i[2])  
  
    elif len(i) == 4:  
  
        result = Model.search_book(int(i[2]), int(i[3]))  
  
View.display_books1(result)
```

- назву мови програмування та бібліотек, що були використані;

Python, Psychopg2

Вимоги до пункту №1 деталізованого завдання:

- лістинги та скріншоти результатів виконання операції вилучення запису батьківської таблиці та виведення вмісту дочірньої таблиці після цього вилучення, а якщо воно неможливе, то результат перехоплення помилки з виведенням повідомлення про неможливість такого видалення за наявності залежних даних. Причини помилок мають бути пояснені;

The image displays four screenshots of a Python script running in a Windows Command Prompt. The script interacts with a database to perform various operations on different tables. The operations shown are:

- Author Operations:** Reading, updating, and deleting author records. The script uses a delimiter to separate the author ID from the operation details.
- Book Operations:** Reading, updating, and deleting book records. The script uses a delimiter to separate the book ID from the operation details.
- Client Operations:** Reading, updating, and deleting client records. The script uses a delimiter to separate the client ID from the operation details.
- Purchase Operations:** Reading, updating, and deleting purchase records. The script uses a delimiter to separate the purchase ID from the operation details.

The script uses a simple text-based interface for input and output, displaying the results of each operation. The output shows the current state of the database after each operation, including the ID, name, and other relevant details of the record being manipulated.





- лістинги та скріншоти результатів виконання операції вставки запису в дочірню таблицю та виведення повідомлення про її неможливість, якщо в батьківські таблиці немає відповідного запису.

```
Command Prompt - py 1.py
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>py 1.py
Input: add author_book
Adding...
author_id/book_id
Input (delimiter=="/"): 7/4
Input: add author_book
Adding...
author_id/book_id
Input (delimiter=="/"): 7/1
Can't add author book pair.
Input: add purchase_book
Adding...
purchase_id/book_id
Input (delimiter=="/"): 3/1
Can't add purchase book pair.
Input: add purchase_book
Adding...
purchase_id/book_id
Input (delimiter=="/"): 1/8
Input:
```

Вимоги до пункту №2 деталізованого завдання:

- копії екрану (ілюстрації) з фрагментами згенерованих даних таблиць;

	Name character varying 	AuthorID [PK] bigint 
20	KSSSF XRI[Z	26822540
21	HKEJU MCUUZ	39675507
22	WNKZF HKCCR	40478770
23	HOIRH NUBJD	54708620
24	GHYPE NIYTK	59134876
25	HRZLI JT[N[76261063

	BookID [PK] bigint 	Name character varying 	PageCount integer 	Price money 
8	59333210	RVITF	490	\$90.00
9	64110334	YLGGN	747	\$129.00
10	64571624	IPGSZ	889	\$26.00
11	73208930	NROUV	99	\$77.00
12	81900898	ERBTU	659	\$22.00
13	98708607	JJXRV	875	\$45.00

- копії SQL-запитів, що ілюструють генерацію при визначених вхідних параметрах.

```
def add_random_authors(n):

    conn, cur = Controller.connect()

    for i in range(0, n):

        try:

            cur.execute("""INSERT INTO "Author" (\\"AuthorID\\", \\"Name\\")

                VALUES (

                    (random() * 99999999)::int,

                    (SELECT (chr(ascii('B') + (random() * 25)::int)) ||

                        chr(ascii('B') + (random() * 25)::int)) ||

                        chr(ascii('B') + (random() * 25)::int)) ||

                        chr(ascii('B') + (random() * 25)::int)) ||

                        chr(ascii('B') + (random() * 25)::int)) ||

                        chr(ascii(' ')) ||

                        chr(ascii('B') + (random() * 25)::int)) ||

                        chr(ascii('B') + (random() * 25)::int)) ||

                        chr(ascii('B') + (random() * 25)::int)) ||

                        chr(ascii('B') + (random() * 25)::int)) ||

                        chr(ascii('B') + (random() * 25)::int))

                    ));""")

        except:

            i = i - 1

    conn.commit()

    cur.close()

    conn.close()
```

```

def add_random_books(n):

    conn, cur = Controller.connect()

    for i in range(0, n):

        try:

            cur.execute("""INSERT INTO "Book" (\\"BookID\\", \\"Name\\",
\\"PageCount\\", \\"Price\\")

                        VALUES (

                            (random() * 999999999)::int,

                            (SELECT (chr(ascii('B') + (random() * 25)::int) ||

                                chr(ascii('B') + (random() * 25)::int) ||

                                chr(ascii('B') + (random() * 25)::int) ||

                                chr(ascii('B') + (random() * 25)::int) ||

                                chr(ascii('B') + (random() * 25)::int)

                                )),

                            (random() * 999)::int,

                            (random() * 250)::int

                        ) """)

        except:

            i = i - 1

    conn.commit()

    cur.close()

    conn.close()

#purchase_id, client_id, sum, transaction_number

def add_random_purchases(n):

    conn, cur = Controller.connect()

    for i in range(0, n):

```



```

        cur.execute("""INSERT INTO \"Purchase\" (\"PurchaseID\",
\"ClientID\", \"Sum\", \"BankTransactionNumber\")

        VALUES (

            (random() * 99999999)::int,

            (SELECT \"ClientID\" FROM \"Client\" ORDER BY
RANDOM() LIMIT 1),

            (random() * 2500)::int,

            (SELECT (chr(ascii('U')) ||

chr(ascii('A')) ||

chr(ascii('0') + (random() * 9)::int) ||

chr(ascii('0') + (random() * 9)::int) ||

chr(ascii('0') + (random() * 9)::int) ||

chr(ascii('0') + (random() * 9)::int) ||

chr(ascii('0') + (random() * 9)::int) ||

chr(ascii('0') + (random() * 9)::int) ||

chr(ascii('0') + (random() * 9)::int) ||

chr(ascii('0') + (random() * 9)::int)))""")

    conn.commit()

    cur.close()

    conn.close()

#client_id, name, login, email
def add_random_clients(n):

    conn, cur = Controller.connect()

    for i in range(0, n):

        cur.execute("""INSERT INTO \"Client\" (\"ClientID\", \"Name\",
\"Login\", \"Email\")

        VALUES (

```

```
(random() * 999999999)::int,

    (SELECT (chr(ascii('B') + (random() * 25)::int)

||

    chr(ascii('B') + (random() * 25)::int) ||

    chr(ascii('B') + (random() * 25)::int) ||

    chr(ascii('B') + (random() * 25)::int) ||

    chr(ascii(' ') ) ||

    chr(ascii('B') + (random() * 25)::int) ||

    chr(ascii('B') + (random() * 25)::int) ||

    chr(ascii('B') + (random() * 25)::int))),

    (SELECT (chr(ascii('B') + (random() * 25)::int)

||

    chr(ascii('B') + (random() * 25)::int) ||

    chr(ascii('B') + (random() * 25)::int) ||

    chr(ascii('B') + (random() * 25)::int))),

    (SELECT (chr(ascii('B') + (random() * 25)::int)

||

    chr(ascii('B') + (random() * 25)::int) ||

    chr(ascii('B') + (random() * 25)::int) ||

    chr(ascii('B') + (random() * 25)::int) ||

    chr(ascii('@')) ||

    chr(ascii('g')) ||

    chr(ascii('m')) ||

    chr(ascii('a')) ||

    chr(ascii('i')) ||

    chr(ascii('l')) ||
```

```

        chr(ascii('.')) ||

        chr(ascii('c')) ||

        chr(ascii('o')) ||

        chr(ascii('m'))))) """)

conn.commit()

cur.close()

conn.close()

def add_random_author_book_pairs(n):

    conn, cur = Controller.connect()

    for i in range(0, n):

        try:

            cur.execute("""INSERT INTO "Author_Book" (\\"AuthorID\\",
\\"BookID\\")

                        VALUES (

                                (SELECT \\"AuthorID\\" FROM \\"Author\\" ORDER BY
RANDOM() LIMIT 1),

                                (SELECT \\"BookID\\" FROM \\"Book\\" ORDER BY
RANDOM() LIMIT 1)

                        ) """)

        except:

            i = i - 1

    conn.commit()

    cur.close()

    conn.close()

def add_random_purchase_book_pairs(n):

    conn, cur = Controller.connect()

```

```

for i in range(0, n):

    try:

        cur.execute("""INSERT INTO "Purchase_Book" (\\"PurchaseID\\",
\\"BookID\\")

                        VALUES (

                                (SELECT \\"PurchaseID\\" FROM \\"Purchase\\" ORDER
BY RANDOM() LIMIT 1),

                                (SELECT \\"BookID\\" FROM \\"Book\\" ORDER BY
RANDOM() LIMIT 1)

                        ) """)

    except:

        i = i - 1

conn.commit()

cur.close()

conn.close()

```

Вимоги до пункту №3 деталізованого завдання:

- ілюстрації уведення пошукового запиту та результатів виконання запитів;

```
Command Prompt - py 1.py
Searching...
SELECT * FROM "Book" WHERE UPPER("Name") LIKE 'X101X';
READ:
Input: search book 101 500
Searching...
READ:
Book is (64110334, 'VLGGN', 747, '$129.00')
Book is (45234037, 'LYZVM', 292, '$96.00')
Book is (59333210, 'RVITF', 490, '$90.00')
Book is (56990548, 'GVHTT', 204, '$187.00')
Input: search book 10 230
Searching...
READ:
Book is (10399316, 'TJBL0', 525, '$61.00')
Book is (73208930, 'NRQV', 99, '$77.00')
Book is (01900899, 'ERBTU', 659, '$22.00')
Book is (64571624, 'IPGSZ', 889, '$26.00')
Book is (64110334, 'VLGGN', 747, '$129.00')
Book is (45234037, 'LYZVM', 292, '$96.00')
Book is (08708607, 'J3XRV', 875, '$45.00')
Book is (59333210, 'RVITF', 490, '$90.00')
Book is (56990548, 'GVHTT', 204, '$187.00')
Input:

Command Prompt - py 1.py
SELECT * FROM "Book" WHERE UPPER("Name") LIKE 'X1X';
READ:
Book is (7, 'UFLMI', 6, '$7.00')
Book is (64571624, 'IPGSZ', 889, '$26.00')
Book is (59333210, 'RVITF', 490, '$90.00')
Input: search book p
Searching...
SELECT * FROM "Book" WHERE UPPER("Name") LIKE 'XpX';
READ:
Book is (64571624, 'IPGSZ', 889, '$26.00')
Input: search book z
Searching...
SELECT * FROM "Book" WHERE UPPER("Name") LIKE 'XzX';
READ:
Book is (4, 'D0ZLV', 3, '$4.00')
Book is (64571624, 'IPGSZ', 889, '$26.00')
Book is (45234037, 'LYZVM', 292, '$96.00')
Input: search book x
Searching...
SELECT * FROM "Book" WHERE UPPER("Name") LIKE 'X0X';
READ:
Book is (98708607, 'J3XRV', 875, '$45.00')
Input:

C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>py 1.py
Input: search author z
Searching...
SELECT * FROM "Author" WHERE UPPER("Name") LIKE 'ZSX';
[('VWHS', 5), ('SEBLD', 6), ('DR[SF', 9), ('YUSXS', 88605), ('VBSSY YD[PU', 6290355), ('KSSSF XRI[Z', 26822540)]
Input: search author d
Searching...
SELECT * FROM "Author" WHERE UPPER("Name") LIKE 'ZDX';
[('SEBLD', 6), ('VWHS', 5), ('DR[SF', 9), ('Updated2', 15), ('JDINT', 60705), ('HOIRN MUBJD', 54708620), ('DQZIV CRKFU', 19153), ('VBSSY YD[PU', 6290355)]
Input: search author x
Searching...
SELECT * FROM "Author" WHERE UPPER("Name") LIKE 'Z0X';
[('VWHS', 5), ('SEBLD', 6), ('DR[SF', 9), ('Updated2', 15), ('JDINT', 60705), ('HOIRN MUBJD', 54708620), ('DQZIV CRKFU', 19153), ('VBSSY YD[PU', 6290355)]
Input:

C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>
C:\Users\Ivan\Desktop\DBLabs\DB_GitHub\DBLabs1-3\Lab2>py 1.py
Input: search purchase 5454
Searching...
SELECT * FROM "Purchase" WHERE UPPER("BankTransactionNumber") LIKE 'X3454X';
[(184808067, '$244.00', 'UA53754541', 41749667)]
Input: search purchase 244
Searching...
SELECT * FROM "Purchase" WHERE UPPER("BankTransactionNumber") LIKE 'X244X';
[(975715920, '$318.00', 'UA2440410', 82387074), (41387575, '$2,351.00', 'UA82445311', 12453151), (470662964, '$905.00', 'UA24483328', 42778512), (872715707, '$1,100.00', 'UA24473076', 11959882), (184808067, '$558.00', 'UA06224454', 75324707), (27211204, '$1,634.00', 'UA23324495', 26346153), (546030986, '$2,233.00', 'UA44244190', 70297600), (91163519, '$376.00', 'UA04244027', 1866023), (750271553, '$738.00', 'UA86244543', 3807880), (11164770, '$520.00', 'UA72447440', 87437338), (627374318, '$1,464.00', 'UA32443634', 96009459)]
Input: search purchase ua5375
Searching...
SELECT * FROM "Purchase" WHERE UPPER("BankTransactionNumber") LIKE 'XUA5375X';
[(184808067, '$244.00', 'UA53754541', 41749667)]
Input:
```

- копії SQL-запитів, що ілюструють пошук із зазначеними початковими параметрами.

```
def search_client(name):

    conn, cur = Controller.connect()

    s = "SELECT * FROM \"Client\" WHERE UPPER(\"Name\") LIKE '%" + name
    + "%' OR UPPER(\"Login\") LIKE '%" + name + "%' OR UPPER(\"Email\")
    LIKE '%" + name + "%' ;"

    print(s)

    try:

        cur.execute(s)

    except:

        print("Can't search book.")

    return cur.fetchall()

def search_author(name):

    conn, cur = Controller.connect()

    s = "SELECT * FROM \"Author\" WHERE UPPER(\"Name\") LIKE '%" + name
    + "%';"
```

```

print(s)

try:

    cur.execute(s)

except:

    print("Can't search book.")

return cur.fetchall()

def search_purchase(name):

    conn, cur = Controller.connect()

    s = "SELECT * FROM \"Purchase\" WHERE
UPPER(\"BankTransactionNumber\") LIKE '%" + name + "%'";

    print(s)

    try:

        cur.execute(s)

    except:

        print("Can't search book.")

    return cur.fetchall()

def search_book(price_min, price_max):

    conn, cur = Controller.connect()

    try:

        cur.execute("SELECT * FROM \"Book\" WHERE %s <=
(\"Price\"::numeric::int) and (\"Price\"::numeric::int) <= %s OR %s <=
\"PageCount\" and \"PageCount\" <= %s OR %s <= \"BookID\" and
\"BookID\" <= %s;", (price_min, price_max, price_min, price_max,
price_min, price_max))

    except:

        print("Can't search book.")

    return cur.fetchall()

```

```

def search_book_by_name(name):

    conn, cur = Controller.connect()

    s = "SELECT * FROM \"Book\" WHERE UPPER(\"Name\") LIKE '%" + name +
"%'";

    print(s)

    try:

        cur.execute(s)

    except:

        print("Can't search book.")

    return cur.fetchall()

```

Вимоги до пункту №4 деталізованого завдання:

- ілюстрації програмного коду модуля “Model”, згідно із шаблоном MVC. Надати короткий опис функцій модуля.

Функції поділяються на [add_, get_, update_, delete_, add_random_, search_] і [_author, _book, _client, _purchase]. Відповідно перше слово означає дію, друге слово об’єкт над якою дія виконується. Спочатку розміщені функції add_, get_, update_, delete_ для кожного об’єкта, потім add_random_, і в самому кінці search_.

```

import psycopg2

import Controller

def add_author(author_id, author_name):

    conn, cur = Controller.connect()

    try:

        cur.execute("INSERT INTO \"Author\" (\"AuthorID\", \"Name\")
VALUES (%s, %s)", (author_id, author_name))

    except:

```

```

        print("Can't add author.")

    conn.commit()

    cur.close()

    conn.close()

def get_author(author_id):

    conn, cur = Controller.connect()

    try:

        cur.execute("SELECT \"AuthorID\", \"Name\" FROM \"Author\" WHERE \"AuthorID\"='%s'", (author_id,))

    except:

        print("Can't get author.")

    return cur.fetchall()

def update_author(author_id, author_name):

    conn, cur = Controller.connect()

    try:

        cur.execute("UPDATE \"Author\" SET \"Name\"=%s WHERE \"AuthorID\"=%s", (author_name, author_id))

    except:

        print("Can't update author.")

    conn.commit()

    cur.close()

    conn.close()

def delete_author(author_id):

    conn, cur = Controller.connect()

    try:

```



```
        cur.execute("DELETE FROM \"Author\" WHERE \"AuthorID\"='%s'",
(author_id,))
```

```
    except:
```

```
        print("Can't delete author.")
```

```
    conn.commit()
```

```
    cur.close()
```

```
    conn.close()
```

```
def add_book(book_id, book_name, page_count, book_price):
```

```
    conn, cur = Controller.connect()
```

```
    try:
```

```
        cur.execute("INSERT INTO \"Book\" (\"BookID\", \"Name\",
\"PageCount\", \"Price\") VALUES (%s, %s, %s, %s);", (book_id,
book_name, page_count, book_price))
```

```
    except:
```

```
        print("Can't add book.")
```

```
    conn.commit()
```

```
    cur.close()
```

```
    conn.close()
```

```
def get_book(book_id):
```

```
    conn, cur = Controller.connect()
```

```
    try:
```

```
        cur.execute("SELECT \"BookID\", \"Name\", \"PageCount\", \"Price\"
FROM \"Book\" WHERE \"BookID\"='%s'", (book_id,))
```

```
    except:
```

```
        print("Can't get book.")
```

```

        return cur.fetchall()

def update_book(book_id, book_name, page_count, book_price):

    conn, cur = Controller.connect()

    try:

        cur.execute("UPDATE \"Book\" SET \"Name\"=%s, \"PageCount\"=%s, \"Price\"=%s WHERE \"BookID\"=%s", (book_name, page_count, book_price, book_id))

    except:

        print("Can't update book.")

    conn.commit()

    cur.close()

    conn.close()

def delete_book(book_id):

    conn, cur = Controller.connect()

    try:

        cur.execute("DELETE FROM \"Book\" WHERE \"BookID\"='%s'", (book_id,))

    except:

        print("Can't delete book.")

    conn.commit()

    cur.close()

    conn.close()

def add_author_book_pair(author_id, book_id):

    conn, cur = Controller.connect()

    try:

```

```

        cur.execute("INSERT INTO \"Author_Book\" (\"AuthorID\",
\"BookID\") VALUES (%s, %s)", (author_id, book_id))

    except:

        print("Can't add author book pair.")

    conn.commit()

    cur.close()

    conn.close()

def get_authors_by_book_id(book_id):

    conn, cur = Controller.connect()

    try:

        cur.execute("SELECT \"AuthorID\" FROM \"Author_Book\" WHERE
\"BookID\"='%s'", (book_id,))

    except:

        print("Can't get authors by book id.")

    return cur.fetchall()

def delete_author_book_pairs_by_book_id(book_id):

    conn, cur = Controller.connect()

    try:

        cur.execute("DELETE FROM \"Author_Book\" WHERE
\"BookID\"='%s'", (book_id,))

    except:

        print("Can't delete author_book pair.")

    conn.commit()

    cur.close()

    conn.close()

```

```
def add_purchase_book_pair(purchase_id, book_id):

    conn, cur = Controller.connect()

    try:

        cur.execute("INSERT INTO \"Purchase_Book\" (\"PurchaseID\",
\"BookID\") VALUES (%s, %s)", (purchase_id, book_id))

    except:

        print("Can't add purchase book pair.")

    conn.commit()

    cur.close()

    conn.close()


def get_books_by_purchase_id(purchase_id):

    conn, cur = Controller.connect()

    try:

        cur.execute("SELECT \"BookID\" FROM \"Purchase_Book\" WHERE
\"PurchaseID\"='%s'", (purchase_id,))

    except:

        print("Can't get books by purchase id.")

    return cur.fetchall()


def delete_purchase_book_pairs_by_purchase_id(purchase_id):

    conn, cur = Controller.connect()

    try:

        cur.execute("DELETE FROM \"Purchase_Book\" WHERE
\"PurchaseID\"='%s'", (purchase_id,))

    except:

        print("Can't delete purchase_book pair.")

    conn.commit()
```

```

cur.close()

conn.close()

def add_purchase(purchase_id, client_id, sum, transaction_number):

    conn, cur = Controller.connect()

    try:

        cur.execute("INSERT INTO \"Purchase\" (\"ClientID\", \"Sum\",
\"BankTransactionNumber\", \"PurchaseID\") VALUES (%s, %s, %s, %s);",
(client_id, sum, transaction_number, purchase_id))

    except:

        print("Can't add purchase.")

    conn.commit()

    cur.close()

    conn.close()

def get_purchase(purchase_id):

    conn, cur = Controller.connect()

    try:

        cur.execute("SELECT \"ClientID\", \"Sum\",
\"BankTransactionNumber\", \"PurchaseID\" FROM \"Purchase\" WHERE
\"PurchaseID\"='%s'", (purchase_id,))

    except:

        print("Can't get purchase")

    return cur.fetchall()

def update_purchase(purchase_id, client_id, sum, transaction_number):

    conn, cur = Controller.connect()

    try:

```

```

        cur.execute("UPDATE \"Purchase\" SET \"ClientID\"=%s,
\"Sum\"=%s, \"BankTransactionNumber\"=%s WHERE \"PurchaseID\"=%s",
(client_id, sum, transaction_number, purchase_id))

    except:

        print("Can't update purchase.")

    conn.commit()

    cur.close()

    conn.close()

def delete_purchase(purchase_id):

    conn, cur = Controller.connect()

    try:

        cur.execute("DELETE FROM \"Purchase\" WHERE
\"PurchaseID\"='%s'", (purchase_id,))

    except:

        print("Can't delete purchase.")

    conn.commit()

    cur.close()

    conn.close()

def add_client(client_id, name, login, email):

    conn, cur = Controller.connect()

    try:

        cur.execute("INSERT INTO \"Client\" (\"ClientID\", \"Name\",
\"Login\", \"Email\") VALUES (%s, %s, %s, %s);", (client_id, name,
login, email))

    except:

        print("Can't add client.")

    conn.commit()

```

```

cur.close()

conn.close()

def get_client(client_id):

    conn, cur = Controller.connect()

    try:

        cur.execute("SELECT \"ClientID\", \"Name\", \"Login\",
\"Email\" FROM \"Client\" WHERE \"ClientID\"='%s'", (client_id,))

    except:

        print("Can't get client.")

    return cur.fetchall()

def update_client(client_id, name, login, email):

    conn, cur = Controller.connect()

    try:

        cur.execute("UPDATE \"Client\" SET \"Name\"=%s, \"Login\"=%s,
\"Email\"=%s WHERE \"ClientID\"=%s", (name, login, email, client_id))

    except:

        print("Can't update client.")

    conn.commit()

    cur.close()

    conn.close()

def delete_client(client_id):

    conn, cur = Controller.connect()

    try:

        cur.execute("DELETE FROM \"Client\" WHERE \"ClientID\"='%s'",
(client_id,))

```

```

except:

    print("Can't delete client.")

conn.commit()

cur.close()

conn.close()

def add_random_authors(n):

    conn, cur = Controller.connect()

    for i in range(0, n):

        try:

            cur.execute("""INSERT INTO "Author"(\\"AuthorID\\", \\"Name\\")

                VALUES (

                    (random() * 999999999)::int,

                    (SELECT (chr(ascii('B') + (random() * 25)::int) ||

                        chr(ascii('B') + (random() * 25)::int) ||

                        chr(ascii('B') + (random() * 25)::int) ||

                        chr(ascii('B') + (random() * 25)::int) ||

                        chr(ascii('B') + (random() * 25)::int) ||

                        chr(ascii(' '))) ||

                        chr(ascii('B') + (random() * 25)::int) ||

                        chr(ascii('B') + (random() * 25)::int) ||

                        chr(ascii('B') + (random() * 25)::int) ||

                        chr(ascii('B') + (random() * 25)::int) ||

                        chr(ascii('B') + (random() * 25)::int)

                    )))""")

        except:

            i = i - 1

```



```

conn.commit()

cur.close()

conn.close()

def add_random_books(n):

    conn, cur = Controller.connect()

    for i in range(0, n):

        try:

            cur.execute("""INSERT INTO "Book" (\\"BookID\\", \\"Name\\",
\\"PageCount\\", \\"Price\\")

                        VALUES (

                            (random() * 999999999)::int,

                            (SELECT (chr(ascii('B') + (random() * 25)::int) ||

                                chr(ascii('B') + (random() * 25)::int) ||

                                chr(ascii('B') + (random() * 25)::int) ||

                                chr(ascii('B') + (random() * 25)::int) ||

                                chr(ascii('B') + (random() * 25)::int)

                                )) ,

                            (random() * 999)::int,

                            (random() * 250)::int

                        ) """)

        except:

            i = i - 1

    conn.commit()

    cur.close()

    conn.close()

```

```

#purchase_id, client_id, sum, transaction_number

def add_random_purchases(n):

    conn, cur = Controller.connect()

    for i in range(0, n):

        cur.execute("""INSERT INTO \"Purchase\" (\"PurchaseID\",
\"ClientID\", \"Sum\", \"BankTransactionNumber\")

                        VALUES (

                                (random() * 999999999)::int,

                                (SELECT \"ClientID\" FROM \"Client\" ORDER BY
RANDOM() LIMIT 1),

                                (random() * 2500)::int,

                                (SELECT (chr(ascii('U')) ||

chr(ascii('A')) ||

chr(ascii('0') + (random() * 9)::int) ||

chr(ascii('0') + (random() * 9)::int) ||

chr(ascii('0') + (random() * 9)::int) ||

chr(ascii('0') + (random() * 9)::int) ||

chr(ascii('0') + (random() * 9)::int) ||

chr(ascii('0') + (random() * 9)::int) ||

chr(ascii('0') + (random() * 9)::int) ||

chr(ascii('0') + (random() * 9)::int)))""")

        conn.commit()

        cur.close()

        conn.close()

#client_id, name, login, email

def add_random_clients(n):

    conn, cur = Controller.connect()

```

```

for i in range(0, n):

    cur.execute("""INSERT INTO \"Client\" (\"ClientID\", \"Name\",
\"Login\", \"Email\")

                VALUES (

                    (random() * 999999999)::int,

                    (SELECT (chr(ascii('B') + (random() * 25)::int)
||

                    chr(ascii('B') + (random() * 25)::int) ||

                    chr(ascii('B') + (random() * 25)::int) ||

                    chr(ascii('B') + (random() * 25)::int) ||

                    chr(ascii(' ')) ||

                    chr(ascii('B') + (random() * 25)::int) ||

                    chr(ascii('B') + (random() * 25)::int) ||

                    chr(ascii('B') + (random() * 25)::int))),

                    (SELECT (chr(ascii('B') + (random() * 25)::int)
||

                    chr(ascii('B') + (random() * 25)::int) ||

                    chr(ascii('B') + (random() * 25)::int) ||

                    chr(ascii('B') + (random() * 25)::int))),

                    (SELECT (chr(ascii('B') + (random() * 25)::int)
||

                    chr(ascii('B') + (random() * 25)::int) ||

                    chr(ascii('B') + (random() * 25)::int) ||

                    chr(ascii('B') + (random() * 25)::int) ||

                    chr(ascii('@')) ||

                    chr(ascii('g')) ||

```

```

        chr(ascii('m')) ||
        chr(ascii('a')) ||
        chr(ascii('i')) ||
        chr(ascii('l')) ||
        chr(ascii('.')) ||
        chr(ascii('c')) ||
        chr(ascii('o')) ||
        chr(ascii('m'))))) """)

conn.commit()

cur.close()

conn.close()

def add_random_author_book_pairs(n):

    conn, cur = Controller.connect()

    for i in range(0, n):

        try:

            cur.execute("""INSERT INTO "Author_Book" (\\"AuthorID\\",
\\"BookID\\")

                        VALUES (

                                (SELECT \\"AuthorID\\" FROM \\"Author\\" ORDER BY
RANDOM() LIMIT 1),

                                (SELECT \\"BookID\\" FROM \\"Book\\" ORDER BY
RANDOM() LIMIT 1)

                        ) """)

        except:

            i = i - 1

    conn.commit()

    cur.close()

```

```

conn.close()

def add_random_purchase_book_pairs(n):

    conn, cur = Controller.connect()

    for i in range(0, n):

        try:

            cur.execute("""INSERT INTO "Purchase_Book" (\\"PurchaseID\\",
\\"BookID\\")

                        VALUES (

                                (SELECT \\"PurchaseID\\" FROM \\"Purchase\\" ORDER
BY RANDOM() LIMIT 1),

                                (SELECT \\"BookID\\" FROM \\"Book\\" ORDER BY
RANDOM() LIMIT 1)

                        ) """)

        except:

            i = i - 1

    conn.commit()

    cur.close()

    conn.close()

def search_client(name):

    conn, cur = Controller.connect()

    s = "SELECT * FROM \\"Client\\" WHERE UPPER(\\"Name\\") LIKE '%" + name
+ "%' OR UPPER(\\"Login\\") LIKE '%" + name + "%' OR UPPER(\\"Email\\")
LIKE '%" + name + "%' ;"

    print(s)

    try:

        cur.execute(s)

    except:

```

```
        print("Can't search book.")

    return cur.fetchall()

def search_author(name):

    conn, cur = Controller.connect()

    s = "SELECT * FROM \"Author\" WHERE UPPER(\"Name\") LIKE '%" + name
+ "%';"

    print(s)

    try:

        cur.execute(s)

    except:

        print("Can't search book.")

    return cur.fetchall()

def search_purchase(name):

    conn, cur = Controller.connect()

    s = "SELECT * FROM \"Purchase\" WHERE
UPPER(\"BankTransactionNumber\") LIKE '%" + name + "%';"

    print(s)

    try:

        cur.execute(s)

    except:

        print("Can't search book.")

    return cur.fetchall()

def search_book(price_min, price_max):

    conn, cur = Controller.connect()

    try:
```

```

        cur.execute("SELECT * FROM \"Book\" WHERE %s <=
(\"Price\"::numeric::int) and (\"Price\"::numeric::int) <= %s OR %s <=
\"PageCount\" and \"PageCount\" <= %s OR %s <= \"BookID\" and
\"BookID\" <= %s;", (price_min, price_max, price_min, price_max,
price_min, price_max))

    except:

        print("Can't search book.")

    return cur.fetchall()

def search_book_by_name(name):

    conn, cur = Controller.connect()

    s = "SELECT * FROM \"Book\" WHERE UPPER(\"Name\") LIKE '%" + name +
"%'";

    print(s)

    try:

        cur.execute(s)

    except:

        print("Can't search book.")

    return cur.fetchall()

```