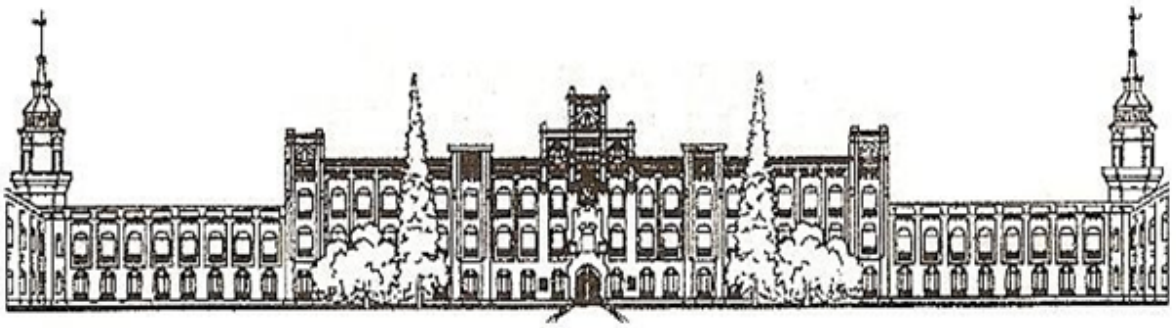


Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут» ім. Ігоря Сікорського

Розрахунково-графічна робота

з дисципліни «Бази даних та засоби управління»

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

Виконав студент групи: КВ-32

Іваха Богдан Миколайович

Київ 2025

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).

Обрана предметна галузь: Бібліотечна система управління книгами.

Репозиторій Github

Телеграм: @bohdanivakha

Опис бази даних

1. **Автор**
   Призначення: зберігає дані про авторів, яким належать книги.
   Атрибути: Прізвище, ім’я, поштова скринька (email), ідентифікатор (id).
2. **Книга**
   Призначення: містить інформацію про книги, що доступні у бібліотеці.
   Атрибути: Назва, рік видання, автор, кількість сторінок,ідентифікатор (id).
3. **Читач**
   Призначення: зберігає відомості про користувачів бібліотеки.
   Атрибути: Прізвище, ім’я,поштова скринька(email), ідентифікатор (id).
4. **Журнал видачі книг**
   Призначення: реєструє факти отримання книг читачами, містить інформацію про дату видачі та повернення.
   Атрибути: Читач, Книга, дата видачі, дата повернення, ідентифікатор (id).

Графічний файл розробленої моделі «сутність-зв’язок»

Для побудови ER-діаграми використовується нотація «Crow’s Foot» («Пташина лапка»).

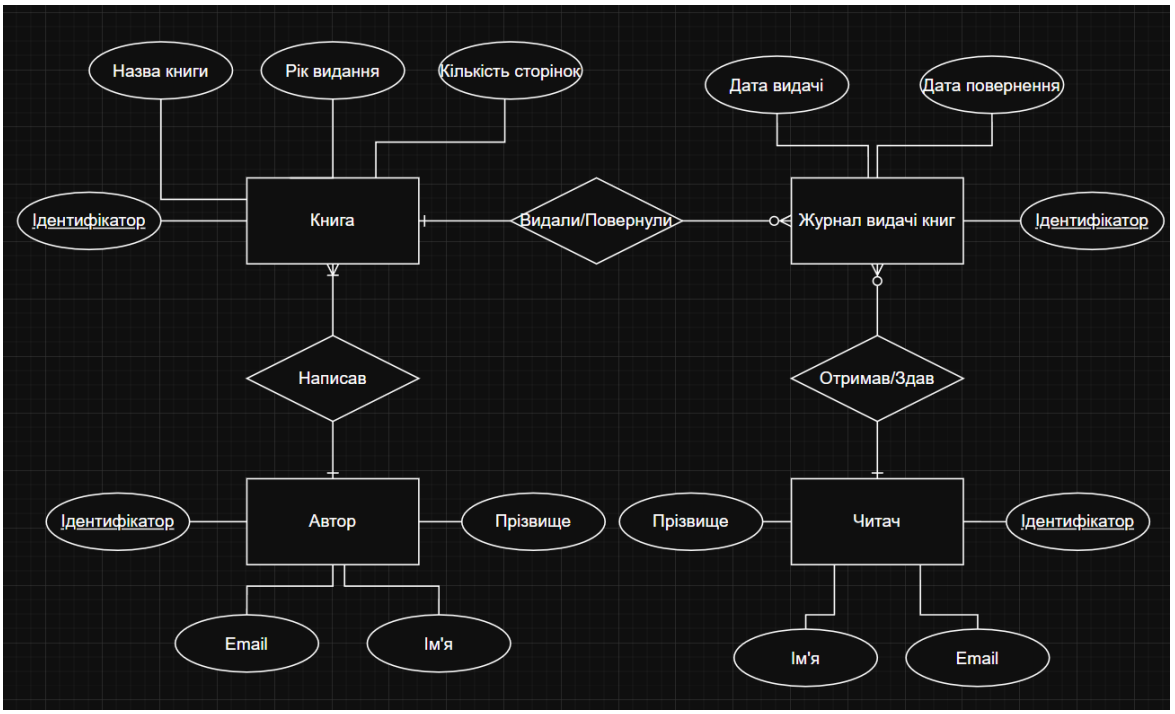


Рисунок 1 - модель сутність-зв’язок

-

-

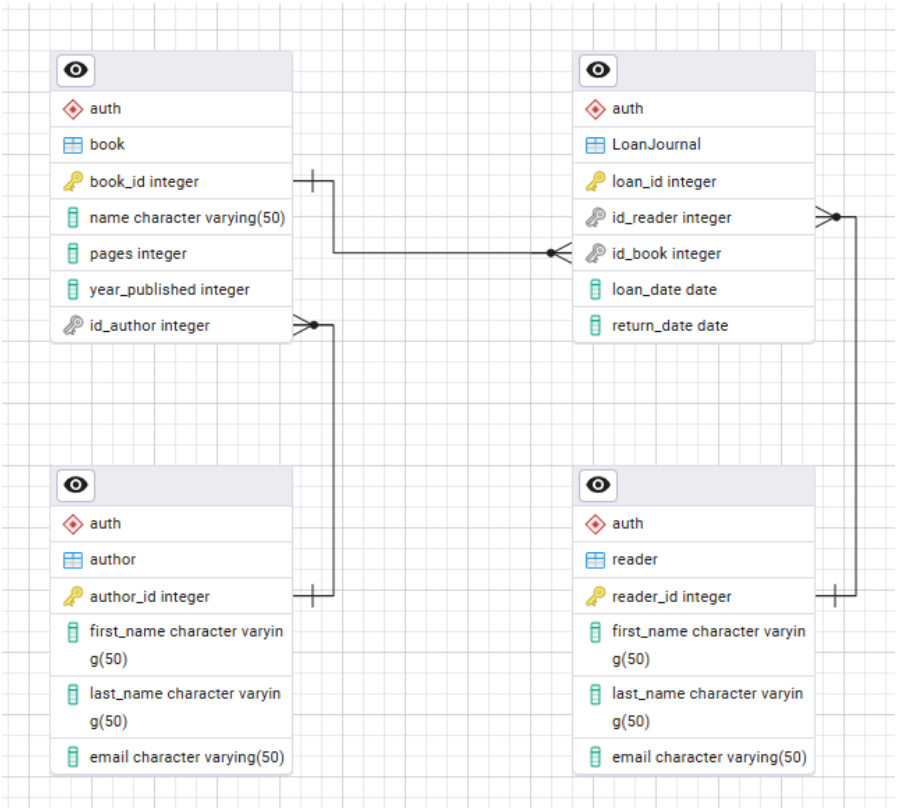


Рисунок 2 - Структура бази даних

Структура схеми меню:

- **=== ГОЛОВНЕ МЕНЮ ===**
  - **1. Перегляд даних**
    - 1.1. Переглянути таблицю "Автори"
    - 1.2. Переглянути таблицю "Книги"
    - 1.3. Переглянути таблицю "Читачі"
    - 1.4. Переглянути таблицю "Журнал видачі книг"
    - 1.5. Повернутися до головного меню
  - **2. Додавання запису**
    - 2.1. Додати нового автора
    - 2.2. Додати нову книгу
    - 2.3. Додати нового читача
    - 2.4. Зареєструвати видачу книги (журнал)
    - 2.5. Повернутися до головного меню
  - **3. Редагування запису**
    - 3.1. Редагувати дані автора
    - 3.2. Редагувати дані книги
    - 3.3. Редагувати дані читача
    - 3.4. Редагувати запис журналу видачі
    - 3.5. Повернутися до головного меню
  - **4. Видалення запису**
    - 4.1. Видалити автора
    - 4.2. Видалити книгу
    - 4.3. Видалити читача
    - 4.4. Видалити запис журналу видачі
    - 4.5. Повернутися до головного меню
  - **5. Генерація випадкових даних**
    - 5.1. Згенерувати випадкових авторів
    - 5.2. Згенерувати випадкові книги
    - 5.3. Згенерувати випадкових читачів
    - 5.4. Згенерувати випадкові записи журналу видачі
    - 5.5. Повернутися до головного меню
  - **6. Пошук за кількома параметрами**
    - 6.1. Пошук книг за автором і роком видання
    - 6.2. Пошук читачів, які отримували певні книги
    - 6.3. Пошук книг, виданих у певному періоді
    - 6.4. Повернутися до головного меню
  - **7. Аналітичні (складні) запити**
    - 7.1. Кількість виданих книг по кожному автору
    - 7.2. Топ-10 найактивніших читачів
    - 7.3. Середній час користування книгою
    - 7.4. Повернутися до головного меню
  - **0. Вихід із програми**

Схема меню з описом функціональності:

=== ГОЛОВНЕ МЕНЮ ===

1. Перегляд даних

Дозволяє користувачу переглядати дані з таблиць бази даних.

- **Підменю:**
  - **1.1. Переглянути таблицю "Автори"**
    - Виводить список усіх авторів, використовуючи запит **SELECT ... FROM author ORDER BY author_id**.
  - **1.2. Переглянути таблицю "Книги"**
    - Виводить список книг, автоматично підтягуючи повне ім'я автора за допомогою **SELECT** та **LEFT JOIN**.
  - **1.3. Переглянути таблицю "Читачі"**
    - Виводить список усіх читачів, використовуючи запит **SELECT ... FROM reader ORDER BY reader_id**.
  - **1.4. Переглянути таблицю "Журнал видачі книг"**
    - Виводить історію видач, підтягуючи назви книг та імена читачів за допомогою двох команд **JOIN**. Результат відсортовано за loan_id (**ORDER BY**).
  - **1.5. Повернутися до головного меню**

2. Додавання запису

Внесення нових даних у базу з повною валідацією.

- **Підменю:**
  - **2.1. Додати нового автора**
    - Запитує ім'я, прізвище та email і додає нового автора в базу за допомогою **INSERT INTO ... VALUES**.
    - **Обробка помилок (від БД):** UniqueViolation (якщо такий email вже існує).
  - **2.2. Додати нову книгу**
    - Запитує назву, рік, сторінки та id_author. Після валідації додає книгу через **INSERT INTO ... VALUES**.
    - **Валідація (в Контролері):** Перевіряє, чи існує автор із вказаним id_author (через **SELECT ... WHERE**).
  - **2.3. Додати нового читача**

        - Запитує ім'я, прізвище та email і додає нового читача через **INSERT INTO ... VALUES**.
        - **Обробка помилок (від БД):** UniqueViolation (якщо такий email вже існує).
    - **2.4. Зареєструвати видачу книги (журнал)**
        - Запитує id_book, id_reader, дату видачі та повернення. Після валідації створює запис через **INSERT INTO ... VALUES**.
        - **Валідація (в Контролері):**
            1. Перевіряє, чи існує книга з таким id_book (**SELECT ... WHERE**).
            2. Перевіряє, чи існує читач з таким id_reader (**SELECT ... WHERE**).
            3. Забороняє видачу, якщо рік видачі < рік публікації книги.
    - **2.5. Повернутися до головного меню**

3. Редагування запису

Зміна існуючих записів з повною валідацією.

- **Підменю:**
    - **3.1. Редагувати дані автора**
        - Запитує author_id. Після валідації та вводу нових даних оновлює запис через **UPDATE ... SET ... WHERE**.
        - **Валідація (в Контролері):** Перевіряє, чи існує автор з таким ID (**SELECT ... WHERE**).
    - **3.2. Редагувати дані книги**
        - Запитує book_id. Після валідації ID книги та нового id_author, оновлює запис через **UPDATE ... SET ... WHERE**.
        - **Валідація (в Контролері):**
            1. Перевіряє, чи існує книга з таким ID.
            2. Перевіряє, чи існує новий автор (id_author).
    - **3.3. Редагувати дані читача**
        - Запитує reader_id. Після валідації ID та вводу нових даних оновлює запис через **UPDATE ... SET ... WHERE**.

        - **Валідація (в Контролері):** Перевіряє, чи існує читач з таким ID.
    - **3.4. Редагувати запис журналу видачі**
        - Запитує loan_id. Після валідації ID запису, нових id_book та id_reader, а також перевірки бізнес-логіки, оновлює запис через **UPDATE ... SET ... WHERE**.
        - **Валідація (в Контролері):**
            1. Перевіряє, чи існує запис журналу з таким ID.
            2. Перевіряє існування нових id_book та id_reader.
            3. Перевіряє, що рік видачі >= рік публікації книги.
    - **3.5. Повернутися до головного меню**

4. Видалення запису

Безпечне видалення даних з валідацією та перевіркою обмежень.

- **Підменю:**
    - **4.1. Видалити автора**
        - Запитує author_id. Після валідації викликає видалення запису через **DELETE FROM ... WHERE**.
        - **Валідація (в Контролері):** Перевіряє, чи існує автор з таким ID (**SELECT ... WHERE**).
        - **Обробка помилок (від БД):** ON DELETE RESTRICT не дозволить видалити автора, якщо у нього є книги.
    - **4.2. Видалити книгу**
        - Запитує book_id. Після валідації викликає видалення запису через **DELETE FROM ... WHERE**.
        - **Валідація (в Контролері):** Перевіряє, чи існує книга з таким ID.
        - **Обробка помилок (від БД):** ON DELETE RESTRICT не дозволить видалити книгу, якщо вона є в журналі.
    - **4.3. Видалити читача**
        - Запитує reader_id. Після валідації викликає видалення запису через **DELETE FROM ... WHERE**.
        - **Валідація (в Контролері):** Перевіряє, чи існує читач з таким ID.
        - **Обробка помилок (від БД):** ON DELETE RESTRICT не дозволить видалити читача, якщо він є в журналі.

    - **4.4. Видалити запис журналу видачі**
        - Запитує loan_id. Після валідації викликає видалення запису через **DELETE FROM ... WHERE**. (Помилок зв'язності не очікується).
        - **Валідація (в Контролері):** Перевіряє, чи існує запис з таким ID.
    - **4.5. Повернутися до головного меню**

5. Генерація випадкових даних

Наповнення бази тестовими даними за допомогою SQL.

- **Підменю:**
    - **5.1. Згенерувати випадкових авторів**
        - Запитує кількість. Генерує англійські імена/прізвища та логічні email за допомогою **INSERT INTO ... SELECT ... FROM unnest() CROSS JOIN unnest() ORDER BY random() LIMIT %s**.
    - **5.2. Згенерувати випадкові книги**
        - Запитує кількість. Генерує назви, роки та призначає випадкового існуючого автора. Використовує **INSERT INTO ... SELECT ... WITH (array_agg) ... FROM generate_series()**.
    - **5.3. Згенерувати випадкових читачів**
        - Запитує кількість. Генерує англійські імена/прізвища та логічні email, використовуючи ту саму логіку, що й 5.1 (**INSERT ... CROSS JOIN ... LIMIT**).
    - **5.4. Згенерувати випадкові записи журналу видачі**
        - Запитує кількість. Створює записи, обираючи випадкових існуючих читачів та книги, використовуючи логіку WITH та array_agg (як у 5.2).
    - **5.5. Повернутися до головного меню**

6. Пошук за кількома параметрами

Виконання складних SELECT запитів з фільтрацією та вимірюванням часу.

- **Підменю:**

- **6.1. Пошук книг за автором і роком видання**
  - Фільтрує книги за прізвищем автора та діапазоном років, використовуючи **SELECT ... JOIN ... WHERE ... LIKE ... AND ... BETWEEN**.
- **6.2. Пошук читачів, які отримували певні книги**
  - Знаходить читачів за назвою книги, використовуючи **SELECT DISTINCT ... JOIN (x2) ... WHERE ... LIKE**.
- **6.3. Пошук книг, виданих у певному періоді**
  - Фільтрує записи журналу за діапазоном дат видачі, використовуючи **SELECT ... JOIN (x2) ... WHERE ... BETWEEN**.
- **6.4. Повернутися до головного меню**

7. Аналітичні (складні) запити

Отримання статистики (використовуючи GROUP BY, COUNT, AVG).

- **Підменю:**
  - **7.1. Кількість виданих книг по кожному автору**
    - Рахує, скільки разів брали книги кожного автора, за допомогою **SELECT**, **COUNT()**, **LEFT JOIN** та **GROUP BY**.
  - **7.2. Топ-10 найактивніших читачів**
    - Складає рейтинг читачів за кількістю взятих книг, використовуючи **SELECT**, **COUNT()**, **JOIN**, **GROUP BY**, **ORDER BY DESC**, **LIMIT 10**.
  - **7.3. Середній час користування книгою**
    - Обчислює середню кількість днів, на яку читачі беруть книги, за допомогою **SELECT AVG(...) FROM ... WHERE**.
  - **7.4. Повернутися до головного меню**

0. Вихід із програми

Завершує роботу програми та коректно закриває з'єднання з базою даних.

Технологічні характеристики проєкту

- **Мова програмування:** Python 3.11
- **Середовище розробки:** Visual Studio Code
- **Система управління базами даних (СУБД):** PostgreSQL
- **Бібліотеки Python:**
    - **Psycopg 2** (для взаємодії з БД)
    - **time** (для вимірювання швидкодії запитів)
- **Архітектурний шаблон:** Model–View–Controller (MVC)

Деталізоване завдання №1

1) Невдале видалення "батька" (наприклад, **Автора**), поки у нього є "діти" (тобто **Книги**).

Маємо:

```
=== ГОЛОВНЕ МЕНЮ БІБЛІОТЕКИ ===

1. Перегляд даних
2. Додавання запису
3. Редагування запису
4. Видалення запису
5. Генерація випадкових даних
6. Пошук за параметрами
7. Аналітичні запити
0. Вихід із програми
Виберіть опцію: 1

--- VIEW MENU ---
1.1. Переглянути таблицю 'Автори'
1.2. Переглянути таблицю 'Книги'
1.3. Переглянути таблицю 'Читачі'
1.4. Переглянути таблицю 'Журнал видачі'
1.5. Повернутися до головного меню
Ваш вибір: 1.1

author_id | last_name | first_name | email
--------------------------------------------------------------------------------
1 | Petrenko | Ivan | ivan.petrenko@gmail.com
2 | Kovalenko | Mykola | mykola.k@gmail.com
3 | Shevchenko | Olena | olena.shev@gmail.com
4 | Sergienko | Sergiy | serg.serg@gmail.com
--------------------------------------------------------------------------------
```

Вміст “батьківської” таблиці Автор до видалення

```
--- VIEW MENU ---
1.1. Переглянути таблицю 'Автори'
1.2. Переглянути таблицю 'Книги'
1.3. Переглянути таблицю 'Читачі'
1.4. Переглянути таблицю 'Журнал видачі'
1.5. Повернутися до головного меню
Ваш вибір: 1.2

book_id | name | year_published | pages | author_name
--------------------------------------------------------------------------------
1 | Programming Basics | 2020 | 250 | Petrenko Ivan
2 | Modern Art | 2018 | 150 | Kovalenko Mykola
3 | Physics Fundamentals | 2001 | 300 | Shevchenko Olena
4 | C++ | 2010 | 200 | Petrenko Ivan
--------------------------------------------------------------------------------
```

Вміст “дочірньої” таблиці Книга до видалення

Спробуємо видалити автора, в якого є книги:

```
=== ГОЛОВНЕ МЕНЮ БІБЛІОТЕКИ ===

1. Перегляд даних
2. Додавання запису
3. Редагування запису
4. Видалення запису
5. Генерація випадкових даних
6. Пошук за параметрами
7. Аналітичні запити
0. Вихід із програми
Виберіть опцію: 4

--- DELETE MENU ---
4.1. Видалити автора
4.2. Видалити книгу
4.3. Видалити читача
4.4. Видалити запис журналу видачі
4.5. Повернутися до головного меню
Ваш вибір: 4.1

author_id | last_name | first_name | email
--------------------------------------------------------------------------------
1 | Petrenko | Ivan | ivan.petrenko@gmail.com
2 | Kovalenko | Mykola | mykola.k@gmail.com
3 | Shevchenko | Olena | olena.shev@gmail.com
4 | Sergienko | Sergiy | serg.serg@gmail.com
--------------------------------------------------------------------------------
Введіть ID для автора для видалення: 1
Ви впевнені, що хочете видалити автора ID=1? (y/n): y
Помилка: Порушення зв'язності даних. (Наприклад, не можна видалити сутність, на яку посилається журнал).

--- DELETE MENU ---
4.1. Видалити автора
4.2. Видалити книгу
4.3. Видалити читача
4.4. Видалити запис журналу видачі
4.5. Повернутися до головного меню
Ваш вибір:
```

Програма успішно перехопила помилку цілісності даних, яку згенерувала база даних.

2) Успіше видалення "батька", **Автора**, коли у нього немає “дітей” - **книг.**

Вміст таблиць author і book ДО успішного видалення:

```
--- VIEW MENU ---
1.1. Переглянути таблицю 'Автори'
1.2. Переглянути таблицю 'Книги'
1.3. Переглянути таблицю 'Читачі'
1.4. Переглянути таблицю 'Журнал видачі'
1.5. Повернутися до головного меню
Ваш вибір: 1.1

author_id | last_name | first_name | email
--------------------------------------------------------------------------------
1 | Petrenko | Ivan | ivan.petrenko@gmail.com
2 | Kovalenko | Mykola | mykola.k@gmail.com
3 | Shevchenko | Olena | olena.shev@gmail.com
4 | Sergienko | Sergiy | serg.serg@gmail.com
--------------------------------------------------------------------------------

--- VIEW MENU ---
1.1. Переглянути таблицю 'Автори'
1.2. Переглянути таблицю 'Книги'
1.3. Переглянути таблицю 'Читачі'
1.4. Переглянути таблицю 'Журнал видачі'
1.5. Повернутися до головного меню
Ваш вибір: 1.2

book_id | name | year_published | pages | author_name
--------------------------------------------------------------------------------
1 | Programming Basics | 2020 | 250 | Petrenko Ivan
2 | Modern Art | 2018 | 150 | Kovalenko Mykola
3 | Physics Fundamentals | 2001 | 300 | Shevchenko Olena
4 | C++ | 2010 | 200 | Petrenko Ivan
--------------------------------------------------------------------------------
```

Бачимо з таблиць, що у автора 4 (id) немає написаних книг.

```
=== ГОЛОВНЕ МЕНЮ БІБЛІОТЕКИ ===

1. Перегляд даних
2. Додавання запису
3. Редагування запису
4. Видалення запису
5. Генерація випадкових даних
6. Пошук за параметрами
7. Аналітичні запити
0. Вихід із програми
Виберіть опцію: 4

--- DELETE MENU ---
4.1. Видалити автора
4.2. Видалити книгу
4.3. Видалити читача
4.4. Видалити запис журналу видачі
4.5. Повернутися до головного меню
Ваш вибір: 4.1
```

```
Ваш вибір: 4.1

author_id | last_name | first_name | email
--------------------------------------------------------------------------------
1 | Petrenko | Ivan | ivan.petrenko@gmail.com
2 | Kovalenko | Mykola | mykola.k@gmail.com
3 | Shevchenko | Olena | olena.shev@gmail.com
4 | Sergienko | Sergiy | serg.serg@gmail.com
--------------------------------------------------------------------------------
Введіть ID для автора для видалення: 4
Ви впевнені, що хочете видалити автора ID=4? (y/n): y
Автор видалений.
```

Переглянемо таблиці після видалення:

```
--- VIEW MENU ---
1.1. Переглянути таблицю 'Автори'
1.2. Переглянути таблицю 'Книги'
1.3. Переглянути таблицю 'Читачі'
1.4. Переглянути таблицю 'Журнал видачі'
1.5. Повернутися до головного меню
Ваш вибір: 1.1

author_id | last_name | first_name | email
--------------------------------------------------------------------------
1 | Petrenko | Ivan | ivan.petrenko@gmail.com
2 | Kovalenko | Mykola | mykola.k@gmail.com
3 | Shevchenko | Olena | olena.shev@gmail.com
--------------------------------------------------------------------------
```

```
--- VIEW MENU ---
1.1. Переглянути таблицю 'Автори'
1.2. Переглянути таблицю 'Книги'
1.3. Переглянути таблицю 'Читачі'
1.4. Переглянути таблицю 'Журнал видачі'
1.5. Повернутися до головного меню
Ваш вибір: 1.2

book_id | name | year_published | pages | author_name
--------------------------------------------------------------------------------
1 | Programming Basics | 2020 | 250 | Petrenko Ivan
2 | Modern Art | 2018 | 150 | Kovalenko Mykola
3 | Physics Fundamentals | 2001 | 300 | Shevchenko Olena
4 | C++ | 2010 | 200 | Petrenko Ivan
--------------------------------------------------------------------------------
```

Бачимо, що автора Sergienko Sergiy (id 4) більше немає, а вміст таблиці з книгами не змінився **- видалення успішне.**

Причина помилки полягає у спрацьовуванні **обмеження зовнішнього ключа** (foreign key constraint) на рівні СУБД PostgreSQL.

Користувач намагався виконати операцію DELETE для "батьківського" запису з таблиці author (наприклад, з author_id=1).

СУБД PostgreSQL перевірила, чи існують "дочірні" записи в таблиці book, які посилаються на цей author_id. Оскільки такі записи були знайдені, а зв'язок було створено з правилом **ON DELETE RESTRICT**, база даних згенерувала **помилку порушення цілісності** (ForeignKeyViolation), заборонивши операцію DELETE.

```
self.cursor.execute("""
    CREATE TABLE IF NOT EXISTS book (
        book_id SERIAL PRIMARY KEY,
        name TEXT NOT NULL,
        year_published INT,
        pages INT,
        id_author INT,
        CONSTRAINT fk_author
            FOREIGN KEY(id_author)
            REFERENCES author(author_id)
            ON DELETE RESTRICT
    );
""")
```

Як видно з лістингів, ця помилка була коректно **перехоплена** в шарі **Моделі** (в методі _execute_dml) за допомогою блоку try...except psycopg2.errors.ForeignKeyViolation. Програма негайно відкотила транзакцію (rollback) і повернула Контролеру відповідь про невдачу.

```
def _execute_dml(self, query, params=None):
    try:
        self.cursor.execute(query, params)
        self.conn.commit()
        return (True, None)

    except psycopg2.errors.UniqueViolation as e:
        self.conn.rollback()
        return (False, "Порушення унікальності. Можливо, такий email вже існує.")

    except psycopg2.errors.ForeignKeyViolation as e:
        self.conn.rollback()
        return (False, "Порушення зв'язності даних. (Наприклад, не можна видалити сутність, на яку посилається журнал).")

    except psycopg2.errors.NotNullViolation as e:
        self.conn.rollback()
        return (False, f"Не заповнене обов'язкове поле.")

    except (psycopg2.Error, psycopg2.DataError) as e:
        self.conn.rollback()
        return (False, f"Загальна помилка SQL: {e}")
```

```
def handle_delete_menu(self):
    while True:
        choice = self.view.show_submenu('delete')

        if choice == '4.1':
            self.view.show_data(self.model.get_authors())
            author_id = self.view.get_id("автора для видалення")

            if not self.model.get_entity_by_id("author", author_id):
                self.view.show_message(f"Автора з ID {author_id} не існує.", is_error=True)
                continue

            if self.view.get_confirmation(f"видалити автора ID={author_id}"):
                response = self.model.delete_author(author_id)
                self._handle_dml_response(response, "Автор видалений.")

        elif choice == '4.2':
```

```
def _handle_dml_response(self, response_tuple, success_message):
    success, error_message = response_tuple
    if success:
        self.view.show_message(success_message)
    else:
        self.view.show_message(error_message, is_error=True)
```

В результаті, Контролер передав цю інформацію Поданню, і користувач побачив зрозуміле повідомлення (Помилка: Порушення зв'язності даних...) замість аварійного завершення (crash) програми.

```python
def show_message(self, message, is_error=False):
    if is_error:
        print(f"Помилка: {message}")
    else:
        print(f"{message}")
```

Тепер **додамо** запис в дочірню таблицю:

Переглянемо таблиці авторів і книг ДО додавання книги:

```
--- VIEW MENU ---
1.1. Переглянути таблицю 'Автори'
1.2. Переглянути таблицю 'Книги'
1.3. Переглянути таблицю 'Читачі'
1.4. Переглянути таблицю 'Журнал видачі'
1.5. Повернутися до головного меню
Ваш вибір: 1.1

author_id | last_name | first_name | email
--------------------------------------------------------------------------------
1 | Petrenko | Ivan | ivan.petrenko@gmail.com
2 | Kovalenko | Mykola | mykola.k@gmail.com
3 | Shevchenko | Olena | olena.shev@gmail.com
--------------------------------------------------------------------------------

--- VIEW MENU ---
1.1. Переглянути таблицю 'Автори'
1.2. Переглянути таблицю 'Книги'
1.3. Переглянути таблицю 'Читачі'
1.4. Переглянути таблицю 'Журнал видачі'
1.5. Повернутися до головного меню
Ваш вибір: 1.2

book_id | name | year_published | pages | author_name
--------------------------------------------------------------------------------
1 | Programming Basics | 2020 | 250 | Petrenko Ivan
2 | Modern Art | 2018 | 150 | Kovalenko Mykola
3 | Physics Fundamentals | 2001 | 300 | Shevchenko Olena
4 | C++ | 2010 | 200 | Petrenko Ivan
--------------------------------------------------------------------------------
```

Приклад невдалого додавання книги:

```
=== ГОЛОВНЕ МЕНЮ БІБЛІОТЕКИ ===

1. Перегляд даних
2. Додавання запису
3. Редагування запису
4. Видалення запису
5. Генерація випадкових даних
6. Пошук за параметрами
7. Аналітичні запити
0. Вихід із програми
Виберіть опцію: 2

--- ADD MENU ---
2.1. Додати нового автора
2.2. Додати нову книгу
2.3. Додати нового читача
2.4. Зареєструвати видачу книги
2.5. Повернутися до головного меню
Ваш вибір: 2.2
Порада: перегляньте ID авторів (меню 1.1) перед додаванням книги.

author_id | last_name | first_name | email
--------------------------------------------------------------------------------
1 | Petrenko | Ivan | ivan.petrenko@gmail.com
2 | Kovalenko | Mykola | mykola.k@gmail.com
3 | Shevchenko | Olena | olena.shev@gmail.com
--------------------------------------------------------------------------------

--- Введіть дані книги ---
Назва (name): База Даних
Рік видання (year_published): 2017
Кількість сторінок (pages): 222
ID Автора (id_author): 55
Помилка: Автора з ID 55 не існує.
```

В результаті виникла помилка, адже книга не може бути додана, бо автора з ID 55 не існує (в батьківські таблиці немає відповідного існуючого зовнішнього ключа (автора)).

Приклад успішного додавання книги:

```
--- ADD MENU ---
2.1. Додати нового автора
2.2. Додати нову книгу
2.3. Додати нового читача
2.4. Зареєструвати видачу книги
2.5. Повернутися до головного меню
Ваш вибір: 2.2
Порада: перегляньте ID авторів (меню 1.1) перед додаванням книги.

author_id | last_name | first_name | email
--------------------------------------------------------------------------------
1 | Petrenko | Ivan | ivan.petrenkc@gmail.com
2 | Kovalenko | Mykola | mykola.k@gmail.com
3 | Shevchenko | Olena | olena.shev@gmail.com
--------------------------------------------------------------------------------

--- Введіть дані книги ---
Назва (name): Бази Даних
Рік видання (year_published): 2017
Кількість сторінок (pages): 222
ID Автора (id_author): 3
Книгу успішно додано.
```

Переглянему таблиці авторів і книг:

```
--- VIEW MENU ---
1.1. Переглянути таблицю 'Автори'
1.2. Переглянути таблицю 'Книги'
1.3. Переглянути таблицю 'Читачі'
1.4. Переглянути таблицю 'Журнал видачі'
1.5. Повернутися до головного меню
Ваш вибір: 1.1

author_id | last_name | first_name | email
--------------------------------------------------------------------------------
1 | Petrenko | Ivan | ivan.petrenkc@gmail.com
2 | Kovalenko | Mykola | mykola.k@gmail.com
3 | Shevchenko | Olena | olena.shev@gmail.com
--------------------------------------------------------------------------------

--- VIEW MENU ---
1.1. Переглянути таблицю 'Автори'
1.2. Переглянути таблицю 'Книги'
1.3. Переглянути таблицю 'Читачі'
1.4. Переглянути таблицю 'Журнал видачі'
1.5. Повернутися до головного меню
Ваш вибір: 1.2

book_id | name | year_published | pages | author_name
--------------------------------------------------------------------------------
1 | Programming Basics | 2020 | 250 | Petrenko Ivan
2 | Modern Art | 2018 | 150 | Kovalenko Mykola
3 | Physics Fundamentals | 2001 | 300 | Shevchenko Olena
4 | C++ | 2010 | 200 | Petrenko Ivan
5 | Бази Даних | 2017 | 222 | Shevchenko Olena
--------------------------------------------------------------------------------
```

Бачимо, що в таблиці із книгами з’явилась книга “Бази Даних”, автор якої Shevchenko Olena (id 3)

Причиною **відмови** в операції став контроль при введенні на рівні Контролера.

Користувач намагався додати "дочірній" запис у таблицю book зі значенням id_author=55.

Контролер спершу виконав перевірку. Він звернувся до Моделі (get_entity_by_id), яка перевірила, чи існує запис з author_id=55 у "батьківській" таблиці author.

```
elif choice == '2.2':
    self.view.show_message("Порада: перегляньте ID авторів (меню 1.1) перед додаванням книги.")
    self.view.show_data(self.model.get_authors())
    details = self.view.get_book_details()

    author_id = details['id_author']
    if not self.model.get_entity_by_id("author", author_id):
        self.view.show_message(f"Автора з ID {author_id} не існує.", is_error=True)
        continue

    response = self.model.add_book(details['name'], details['year_published'], details['pages'], author_id)
    self._handle_dml_response(response, "Книгу успішно додано.")

elif choice == '2.3':
```

Оскільки такого запису не існує (Модель повернула None/[]), Контролер **зупинив операцію ще до її початку**.

```
# Методи для валідації
def get_entity_by_id(self, entity_name, entity_id):
    allowed_entities = {
        'author': ('author', 'author_id'),
        'book': ('book', 'book_id'),
        'reader': ('reader', 'reader_id'),
        'LoanJournal': ('"LoanJournal"', 'loan_id')
    }

    if entity_name not in allowed_entities:
        return None

    table_name, id_column = allowed_entities[entity_name]
    query = f"SELECT 1 FROM {table_name} WHERE {id_column} = %s"

    result = self._execute_query(query, (entity_id,), fetch=True)
    return result
```

```
def _execute_query(self, query, params=None, fetch=True):
    try:
        self.cursor.execute(query, params)
        self.conn.commit()
        if fetch:
            return self.cursor.fetchall()
        return None
    except (psycopg2.Error, psycopg2.DataError) as e:
        print(f"Помилка читання (SELECT): {e}")
        self.conn.rollback()
        return None
```

Як видно з лістингу Контролера (if not self.model.get_entity_by_id(...)), програма **попередила** потенційну помилку порушення цілісності (ForeignKeyViolation) і негайно вивела користувачеві зрозуміле повідомлення про помилку валідації (Помилка: Автора з ID 55 не існує.) замість аварійного завершення.

Деталізоване завдання №2

Згенеруємо по 15000 записів до кожної таблиці.

```
=== ГОЛОВНЕ МЕНЮ БІБЛІОТЕКИ ===

1. Перегляд даних
2. Додавання запису
3. Редагування запису
4. Видалення запису
5. Генерація випадкових даних
6. Пошук за параметрами
7. Аналітичні запити
0. Вихід із програми
Виберіть опцію: 5

--- GENERATE MENU ---
5.1. Згенерувати випадкових авторів
5.2. Згенерувати випадкові книги
5.3. Згенерувати випадкових читачів
5.4. Згенерувати випадкові записи журналу
5.5. Повернутися до головного меню
Ваш вибір: 5.1
Введіть кількість записів для генерації: 15000
Генерація 15000 записів...
Успішно згенеровано 15000 записів за 0.0823 сек.
```

```
--- GENERATE MENU ---
5.1. Згенерувати випадкових авторів
5.2. Згенерувати випадкові книги
5.3. Згенерувати випадкових читачів
5.4. Згенерувати випадкові записи журналу
5.5. Повернутися до головного меню
Ваш вибір: 5.2
Введіть кількість записів для генерації: 15000
Генерація 15000 записів...
Успішно згенеровано 15000 записів за 0.0907 сек.
```

```
--- GENERATE MENU ---
5.1. Згенерувати випадкових авторів
5.2. Згенерувати випадкові книги
5.3. Згенерувати випадкових читачів
5.4. Згенерувати випадкові записи журналу
5.5. Повернутися до головного меню
Ваш вибір: 5.3
Введіть кількість записів для генерації: 15000
Генерація 15000 записів...
Успішно згенеровано 15000 записів за 0.0828 сек.
```

```
--- GENERATE MENU ---
5.1. Згенерувати випадкових авторів
5.2. Згенерувати випадкові книги
5.3. Згенерувати випадкових читачів
5.4. Згенерувати випадкові записи журналу
5.5. Повернутися до головного меню
Ваш вибір: 5.4
Введіть кількість записів для генерації: 15000
Генерація 15000 записів...
Успішно згенеровано 15000 записів за 0.1376 сек.
```

Переглянемо нові згенеровані записи в таблицях:

```
--- VIEW MENU ---
1.1. Переглянути таблицю 'Автори'
1.2. Переглянути таблицю 'Книги'
1.3. Переглянути таблицю 'Читачі'
1.4. Переглянути таблицю 'Журнал видачі'
1.5. Повернутися до головного меню
Ваш вибір: 1.1
```

```
14965 | Murakami | Virginia | author.14961@authors.com
14966 | Murakami | Agatha | author.14962@authors.com
14967 | Woolf | George | author.14963@authors.com
14968 | Wilde | Gabriel | author.14964@authors.com
14969 | Atwood | Haruki | author.14965@authors.com
14970 | Austen | Virginia | author.14966@authors.com
14971 | Woolf | Oscar | author.14967@authors.com
14972 | King | Stephen | author.14968@authors.com
14973 | Hemingway | Ernest | author.14969@authors.com
14974 | Wilde | Haruki | author.14970@authors.com
14975 | Murakami | Jane | author.14971@authors.com
14976 | Wilde | Stephen | author.14972@authors.com
14977 | King | Virginia | author.14973@authors.com
14978 | Rowling | Stephen | author.14974@authors.com
14979 | Garcia Marquez | Stephen | author.14975@authors.com
14980 | Wilde | George | author.14976@authors.com
14981 | King | Stephen | author.14977@authors.com
14982 | Christie | Jane | author.14978@authors.com
14983 | Atwood | Margaret | author.14979@authors.com
14984 | King | Leo | author.14980@authors.com
14985 | Tolstoy | Agatha | author.14981@authors.com
14986 | Austen | J.K. | author.14982@authors.com
14987 | Hemingway | J.K. | author.14983@authors.com
14988 | Murakami | Haruki | author.14984@authors.com
14989 | Hemingway | Stephen | author.14985@authors.com
14990 | Woolf | Stephen | author.14986@authors.com
14991 | Woolf | Gabriel | author.14987@authors.com
14992 | Murakami | Agatha | author.14988@authors.com
14993 | Wilde | Gabriel | author.14989@authors.com
14994 | Christie | Stephen | author.14990@authors.com
14995 | Austen | Gabriel | author.14991@authors.com
14996 | Orwell | Ernest | author.14992@authors.com
14997 | Wilde | Margaret | author.14993@authors.com
14998 | Woolf | Oscar | author.14994@authors.com
14999 | Austen | Leo | author.14995@authors.com
15000 | Woolf | Oscar | author.14996@authors.com
15001 | Wilde | Margaret | author.14997@authors.com
15002 | Garcia Marquez | Jane | author.14998@authors.com
15003 | Rowling | Haruki | author.14999@authors.com
15004 | Wilde | Virginia | author.15000@authors.com
--------------------------------------------------------------------------------
```

```
--- VIEW MENU ---
1.1. Переглянути таблицю 'Автори'
1.2. Переглянути таблицю 'Книги'
1.3. Переглянути таблицю 'Читачі'
1.4. Переглянути таблицю 'Журнал видачі'
1.5. Повернутися до головного меню
Ваш вибір: 1.2
```

```
14972 | Згенерована Книга №14967 | 1960 | 661 | Hemingway George
14973 | Згенерована Книга №14968 | 2003 | 219 | Christie Haruki
14974 | Згенерована Книга №14969 | 1983 | 144 | Hemingway Gabriel
14975 | Згенерована Книга №14970 | 1970 | 214 | Murakami Oscar
14976 | Згенерована Книга №14971 | 1981 | 430 | Wilde Leo
14977 | Згенерована Книга №14972 | 2002 | 503 | Murakami Stephen
14978 | Згенерована Книга №14973 | 2011 | 300 | Wilde Oscar
14979 | Згенерована Книга №14974 | 2017 | 152 | Hemingway George
14980 | Згенерована Книга №14975 | 1967 | 383 | Rowling J.K.
14981 | Згенерована Книга №14976 | 1954 | 594 | Woolf Gabriel
14982 | Згенерована Книга №14977 | 1952 | 529 | Christie Jane
14983 | Згенерована Книга №14978 | 1959 | 789 | Tolstoy Stephen
14984 | Згенерована Книга №14979 | 1950 | 449 | Orwell Haruki
14985 | Згенерована Книга №14980 | 2023 | 322 | Garcia Marquez Ernest
14986 | Згенерована Книга №14981 | 1989 | 492 | Christie Margaret
14987 | Згенерована Книга №14982 | 1969 | 782 | Christie Margaret
14988 | Згенерована Книга №14983 | 1958 | 564 | Woolf Stephen
14989 | Згенерована Книга №14984 | 1993 | 165 | King Agatha
14990 | Згенерована Книга №14985 | 1993 | 101 | Rowling J.K.
14991 | Згенерована Книга №14986 | 1990 | 334 | Wilde Ernest
14992 | Згенерована Книга №14987 | 1998 | 727 | Austen Margaret
14993 | Згенерована Книга №14988 | 2018 | 398 | Rowling Jane
14994 | Згенерована Книга №14989 | 1990 | 427 | Garcia Marquez Margaret
14995 | Згенерована Книга №14990 | 1985 | 191 | Hemingway Leo
14996 | Згенерована Книга №14991 | 1977 | 138 | King Jane
14997 | Згенерована Книга №14992 | 1995 | 197 | Tolstoy Leo
14998 | Згенерована Книга №14993 | 1988 | 673 | Austen Oscar
14999 | Згенерована Книга №14994 | 1999 | 229 | Woolf Oscar
15000 | Згенерована Книга №14995 | 1964 | 166 | Austen Ernest
15001 | Згенерована Книга №14996 | 1983 | 127 | Wilde Gabriel
15002 | Згенерована Книга №14997 | 1979 | 274 | Woolf J.K.
15003 | Згенерована Книга №14998 | 1951 | 232 | Atwood Jane
15004 | Згенерована Книга №14999 | 2003 | 750 | Tolstoy Stephen
15005 | Згенерована Книга №15000 | 1964 | 753 | Christie Virginia
--------------------------------------------------------------------------------
```

```
--- VIEW MENU ---
1.1. Переглянути таблицю 'Автори'
1.2. Переглянути таблицю 'Книги'
1.3. Переглянути таблицю 'Читачі'
1.4. Переглянути таблицю 'Журнал видачі'
1.5. Повернутися до головного меню
Ваш вибір: 1.3
```

```
14967 | Williams | Jennifer | reader.14967@library.ua
14968 | Jones | Michael | reader.14968@library.ua
14969 | Garcia | Sarah | reader.14969@library.ua
14970 | Rodriguez | Alice | reader.14970@library.ua
14971 | Nelson | Mary | reader.14971@library.ua
14972 | Wilson | Ann | reader.14972@library.ua
14973 | Davis | Peter | reader.14973@library.ua
14974 | Walker | Peter | reader.14974@library.ua
14975 | Garcia | Chris | reader.14975@library.ua
14976 | Brown | James | reader.14976@library.ua
14977 | Smith | Tom | reader.14977@library.ua
14978 | Miller | Sarah | reader.14978@library.ua
14979 | Nelson | David | reader.14979@library.ua
14980 | Hall | Ann | reader.14980@library.ua
14981 | Miller | Emily | reader.14981@library.ua
14982 | Jones | Sarah | reader.14982@library.ua
14983 | Nelson | John | reader.14983@library.ua
14984 | Garcia | Chris | reader.14984@library.ua
14985 | Wilson | Sarah | reader.14985@library.ua
14986 | Rodriguez | Mary | reader.14986@library.ua
14987 | Garcia | Mary | reader.14987@library.ua
14988 | Johnson | Bob | reader.14988@library.ua
14989 | Hall | Chris | reader.14989@library.ua
14990 | Walker | Mary | reader.14990@library.ua
14991 | White | Ann | reader.14991@library.ua
14992 | Hall | Sarah | reader.14992@library.ua
14993 | Miller | Emily | reader.14993@library.ua
14994 | Rodriguez | John | reader.14994@library.ua
14995 | Smith | Mary | reader.14995@library.ua
14996 | Hall | Ann | reader.14996@library.ua
14997 | Jones | Mary | reader.14997@library.ua
14998 | Walker | Ann | reader.14998@library.ua
14999 | Miller | Peter | reader.14999@library.ua
15000 | Brown | Ann | reader.15000@library.ua
--------------------------------------------------------------------------------
```

```
--- VIEW MENU ---
1.1. Переглянути таблицю 'Автори'
1.2. Переглянути таблицю 'Книги'
1.3. Переглянути таблицю 'Читачі'
1.4. Переглянути таблицю 'Журнал видачі'
1.5. Повернутися до головного меню
Ваш вибір: 1.4
```

```
14967 | Згенерована Книга №11754 | Johnson Alice | 2023-02-08 | 2023-03-30
14968 | Згенерована Книга №4996 | Jones Sarah | 2020-04-10 | None
14969 | Згенерована Книга №10440 | White James | 2023-07-06 | 2023-08-03
14970 | Згенерована Книга №3894 | Garcia Mary | 2023-02-10 | 2023-05-08
14971 | Згенерована Книга №7864 | Miller Alice | 2022-11-26 | 2022-12-10
14972 | Згенерована Книга №2073 | Williams Emily | 2021-01-30 | None
14973 | Згенерована Книга №6005 | Johnson Jennifer | 2023-01-08 | 2023-04-06
14974 | Згенерована Книга №7205 | Jones Emily | 2021-11-14 | 2021-12-01
14975 | Згенерована Книга №13823 | Jones David | 2023-06-19 | 2023-08-08
14976 | Згенерована Книга №13431 | White Ann | 2022-08-06 | 2022-09-05
14977 | Згенерована Книга №7028 | Johnson Chris | 2023-07-08 | 2023-09-21
14978 | Згенерована Книга №13795 | Rodriguez Mary | 2022-07-28 | 2022-08-03
14979 | Згенерована Книга №2121 | Walker Sarah | 2020-08-22 | 2020-09-25
14980 | Згенерована Книга №14990 | Williams Mary | 2023-02-10 | 2023-04-30
14981 | Згенерована Книга №1523 | Nelson Tom | 2022-10-04 | None
14982 | Згенерована Книга №11015 | Miller Daniel | 2020-06-12 | 2020-07-14
14983 | Згенерована Книга №11520 | Lee Mary | 2021-09-02 | 2021-11-06
14984 | Згенерована Книга №10024 | Brown John | 2021-05-25 | 2021-08-06
14985 | Згенерована Книга №8583 | White John | 2023-07-12 | 2023-10-07
14986 | Згенерована Книга №11334 | Jones Alice | 2022-05-26 | 2022-08-08
14987 | Згенерована Книга №11008 | Walker Emily | 2021-03-03 | 2021-04-09
14988 | Згенерована Книга №10377 | Miller Jennifer | 2023-08-19 | 2023-10-27
14989 | Згенерована Книга №3532 | Hall Sarah | 2023-05-19 | 2023-06-27
14990 | Згенерована Книга №4932 | Miller Bob | 2020-11-26 | 2021-01-27
14991 | Згенерована Книга №6305 | Garcia Tom | 2023-08-04 | 2023-10-10
14992 | Згенерована Книга №6436 | Lee Sarah | 2022-12-29 | 2023-02-01
14993 | Згенерована Книга №14604 | Smith Michael | 2021-08-25 | 2021-10-08
14994 | Згенерована Книга №4142 | Brown Ann | 2020-05-13 | 2020-05-31
14995 | Згенерована Книга №4857 | Williams David | 2023-04-04 | 2023-04-14
14996 | Згенерована Книга №6094 | Miller David | 2021-06-16 | 2021-08-02
14997 | Згенерована Книга №6344 | White Peter | 2020-01-26 | 2020-04-02
14998 | Згенерована Книга №14925 | Wilson Sarah | 2023-05-15 | 2023-07-27
14999 | Згенерована Книга №10780 | Miller Jennifer | 2021-05-04 | 2021-06-19
15000 | Згенерована Книга №13792 | Davis Mary | 2020-01-12 | None
--------------------------------------------------------------------------------
```

Згенеровано по 15000 записів для кожної таблиці разом за

0,0823сек + 0,0907сек + 0,0828сек + 0,1376сек = 0,3934 сек.

Лістинги SQL-запитів для генерації даних

Кількість записів: s=15000

```
INSERT INTO author (first_name, last_name, email)
SELECT
    (array[
        'Stephen', 'George', 'Jane', 'Haruki', 'Agatha', 'Ernest',
        'Virginia', 'Oscar', 'Leo', 'Margaret', 'J.K.', 'Gabriel'
    ])[floor(random() * 12 + 1)] AS first_name,

    (array[
        'King', 'Orwell', 'Austen', 'Murakami', 'Christie', 'Hemingway',
        'Woolf', 'Wilde', 'Tolstoy', 'Atwood', 'Rowling', 'Garcia Marquez'
    ])[floor(random() * 12 + 1)] AS last_name,

    'author.' || i::text || '@authors.com' AS email
FROM
    generate_series(1, %s) AS s(i);
```

Цей запит створює вказану кількість авторів. Він використовує generate_series(1, %s) як "SQL-цикл", який виконується %s разів. На кожному кроці циклу (i), він вибирає випадкове ім'я та прізвище з наданих масивів. Для гарантії унікальності пошти (щоб уникнути помилки UniqueViolation), він "склеює" номер кроку i з рядком, створюючи пошту на кшталт 'author.5@authors.com'.

```
INSERT INTO reader (first_name, last_name, email)
SELECT
    (array[
        'John', 'Ann', 'Bob', 'Alice', 'Peter', 'Mary', 'David',
        'Michael', 'Sarah', 'Chris', 'Emily', 'James', 'Jennifer', 'Daniel', 'Tom'
    ])[floor(random() * 15 + 1)] AS first_name,

    (array[
        'Smith', 'Nelson', 'Wilson', 'Brown', 'Davis', 'Miller', 'Johnson',
        'Williams', 'Jones', 'Garcia', 'Rodriguez', 'Lee', 'Walker', 'Hall', 'White'
    ])[floor(random() * 15 + 1)] AS last_name,

    'reader.' || i::text || '@library.ua' AS email
FROM
    generate_series(1, %s) AS s(i);
```

Цей запит працює за **точно такою ж логікою**, як і generate_authors, але використовує інші масиви для імен/прізвищ та інший домен для пошти ('@library.ua'), також гарантуючи унікальність за допомогою номера з generate_series.

```sql
INSERT INTO book (name, year_published, pages, id_author)
WITH authors AS (
    SELECT array_agg(author_id) AS ids FROM author
)
SELECT
    'Згенерована Книга №' || s.id,
    floor(random() * (2024 - 1950 + 1) + 1950)::int,
    floor(random() * (800 - 100 + 1) + 100)::int,
    a.ids[floor(random() * array_length(a.ids, 1) + 1 + (s.id * 0))]
FROM
    generate_series(1, %s) AS s(id),
    authors a
WHERE
    a.ids IS NOT NULL;
```

Цей запит спочатку виконує "підготовчий крок" WITH authors AS ..., де збирає ID всіх існуючих авторів в один масив. Потім він запускає generate_series для створення книг. Кожна книга отримує унікальну назву ('Згенерована Книга №' || s.id) та гарантовано існуючий id_author, який випадково обирається з попередньо зібраного масиву. Це запобігає помилці ForeignKeyViolation.

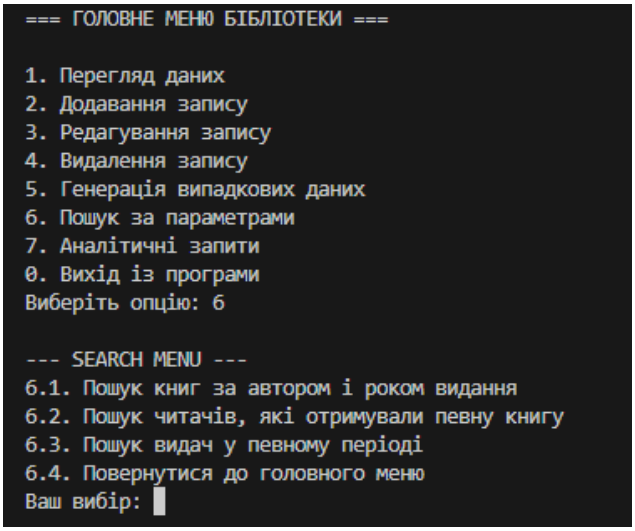
```sql
INSERT INTO "LoanJournal" (id_book, id_reader, loan_date, return_date)
WITH
    books AS (
        SELECT array_agg(book_id) AS ids FROM book
    ),
    readers AS (
        SELECT array_agg(reader_id) AS ids FROM reader
    ),
    GeneratedData AS (
        SELECT
            b.ids[floor(random() * array_length(b.ids, 1) + 1 + (s.id * 0))] AS b_id,
            r.ids[floor(random() * array_length(r.ids, 1) + 1 + (s.id * 0))] AS r_id,
            (timestamp '2020-01-01' + random() * (timestamp '2023-11-01' - timestamp '2020-01-01')) AS i_date
        FROM
            generate_series(1, %s) s(id),
            books b,
            readers r
        WHERE
            b.ids IS NOT NULL AND r.ids IS NOT NULL
    )
SELECT
    b_id,
    r_id,
    i_date::date,
    CASE WHEN random() > 0.2
         THEN (i_date + (floor(random() * 85 + 5) || ' days')::interval)::date
         ELSE NULL
    END
FROM GeneratedData;
```

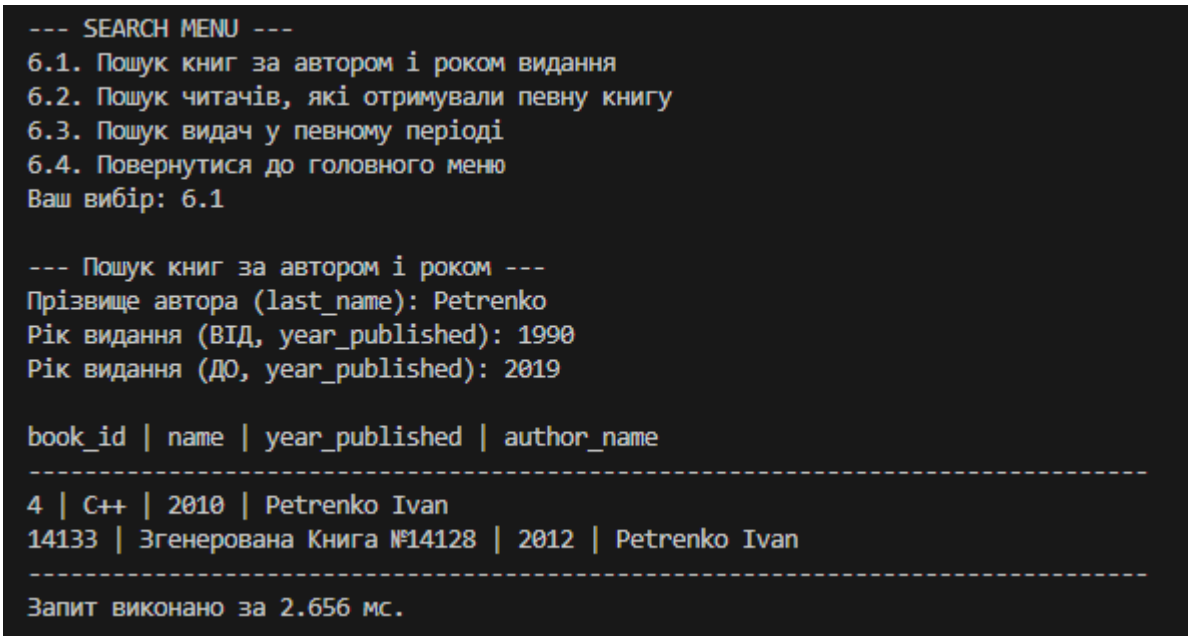
Цей запит використовує ту ж техніку, але збирає **два масиви**: ID всіх книг (WITH books AS...) та ID всіх читачів (WITH readers AS...). Потім він

у циклі generate_series створює записи журналу, випадково обираючи "безпечні" ID з обох масивів. Він також використовує CASE WHEN random() > 0.2 ... ELSE NULL, щоб у 20% випадків залишити дату повернення NULL, імітуючи книги, які ще не повернули.

Деталізоване завдання №3

Виконаємо пошукові запити:





У результаті пошуку за прізвищем Petrenko і за діапазоном року видання (від і до) бачимо, що знайшлось 2 книжки в цьому діапазоні років.

```
--- SEARCH MENU ---
6.1. Пошук книг за автором і роком видання
6.2. Пошук читачів, які отримували певну книгу
6.3. Пошук видач у певному періоді
6.4. Повернутися до головного меню
Ваш вибір: 6.2

--- Пошук читачів за назвою книги ---
Назва книги (name, можна частину): Modern Art

reader_id | last_name | first_name | email | book_name
-------------------------------------------------------------------------------
10250 | Davis | David | reader.10250@library.ua | Modern Art
-------------------------------------------------------------------------------
Запит виконано за 23.786 мс.
```

У результаті пошуку за назвою книги, бачимо, що за весь час її брав лише один читач.

Для перевірки коректності **пошуку читачів за книгою**, здійснимо ще один запит із частковою назвою “Згенерована Книга №1000”.

Очікуємо, що такий запит поверне **всі факти видачі книг**, назви яких *починаються* з “Згенерована Книга №1000…” (наприклад, "Згенерована Книга №1000", "Згенерована Книга №10001", "Згенерована Книга №10002" і т.д.), **разом із даними читачів, які їх брали**.

Якщо один читач брав декілька книг, що відповідають шаблону, він з'явиться у списку декілька разів — по одному разу для кожної взятої книги.

```
--- SEARCH MENU ---
6.1. Пошук книг за автором і роком видання
6.2. Пошук читачів, які отримували певну книгу
6.3. Пошук видач у певному періоді
6.4. Повернутися до головного меню
Ваш вибір: 6.2

--- Пошук читачів за назвою книги ---
Назва книги (name, можна частину): Згенерована Книга №1000

reader_id | last_name | first_name | email | book_name
-------------------------------------------------------------------------------
7222 | Brown | Bob | reader.7222@library.ua | Згенерована Книга №10009
6826 | Brown | Chris | reader.6826@library.ua | Згенерована Книга №10002
5835 | Brown | David | reader.5835@library.ua | Згенерована Книга №10009
14854 | Davis | Daniel | reader.14854@library.ua | Згенерована Книга №10008
231 | Davis | Jennifer | reader.231@library.ua | Згенерована Книга №10004
9531 | Hall | Sarah | reader.9531@library.ua | Згенерована Книга №10003
307 | Lee | Daniel | reader.307@library.ua | Згенерована Книга №10002
13873 | Lee | Daniel | reader.13873@library.ua | Згенерована Книга №10008
739 | Walker | John | reader.739@library.ua | Згенерована Книга №1000
10899 | White | John | reader.10899@library.ua | Згенерована Книга №10001
-------------------------------------------------------------------------------
Запит виконано за 38.162 мс.
```

Виконаємо пошук видач у певному періоді:

```
--- SEARCH MENU ---
6.1. Пошук книг за автором і роком видання
6.2. Пошук читачів, які отримували певну книгу
6.3. Пошук видач у певному періоді
6.4. Повернутися до головного меню
Ваш вибір: 6.3
```

```
--- Пошук видач за періодом ---
Дата (ВІД, loan_date), РРРР-ММ-ДД: 2020-05-07
Дата (ДО, loan_date), РРРР-ММ-ДД: 2020-05-10

loan_id | name | reader | loan_date | return_date
--------------------------------------------------------------------------------
7800 | Згенерована Книга №7786 | Garcia Sarah | 2020-05-07 | 2020-06-09
4199 | Згенерована Книга №13350 | Rodriguez Ann | 2020-05-07 | 2020-07-05
6231 | Згенерована Книга №5219 | White Michael | 2020-05-07 | 2020-06-09
13816 | Згенерована Книга №13834 | Hall David | 2020-05-07 | 2020-07-01
13028 | Згенерована Книга №6089 | Rodriguez Chris | 2020-05-07 | 2020-07-26
9346 | Згенерована Книга №10973 | Davis David | 2020-05-07 | 2020-05-26
14857 | Згенерована Книга №8203 | Walker John | 2020-05-07 | 2020-07-06
2605 | Згенерована Книга №1727 | Brown Chris | 2020-05-07 | 2020-05-15
8016 | Згенерована Книга №10082 | Lee James | 2020-05-07 | 2020-05-15
7952 | Згенерована Книга №12764 | White Michael | 2020-05-08 | 2020-06-02
797 | Згенерована Книга №9827 | Hall David | 2020-05-08 | 2020-06-01
2590 | Згенерована Книга №4730 | Garcia Jennifer | 2020-05-08 | None
5161 | Згенерована Книга №766 | Davis Jennifer | 2020-05-08 | 2020-06-16
219 | Згенерована Книга №13076 | Walker Jennifer | 2020-05-08 | 2020-07-27
9082 | Згенерована Книга №2023 | Jones Alice | 2020-05-08 | 2020-07-08
10110 | Згенерована Книга №3114 | Walker Peter | 2020-05-08 | 2020-07-15
10424 | Згенерована Книга №4210 | Johnson Bob | 2020-05-08 | 2020-07-04
11152 | Згенерована Книга №848 | Garcia Peter | 2020-05-08 | 2020-06-08
13968 | Згенерована Книга №8480 | Nelson Daniel | 2020-05-08 | 2020-07-13
13995 | Згенерована Книга №6053 | Williams Jennifer | 2020-05-08 | 2020-06-11
14664 | Згенерована Книга №12360 | Lee Sarah | 2020-05-08 | None
10289 | Згенерована Книга №4273 | Davis Mary | 2020-05-09 | 2020-07-08
4749 | Згенерована Книга №5466 | Hall John | 2020-05-09 | None
439 | Згенерована Книга №6750 | Brown James | 2020-05-09 | 2020-07-31
7398 | Згенерована Книга №433 | White Bob | 2020-05-09 | 2020-05-25
11293 | Згенерована Книга №14625 | Davis Sarah | 2020-05-09 | None
1295 | Згенерована Книга №10545 | Davis Michael | 2020-05-09 | 2020-06-29
2381 | Згенерована Книга №5231 | Davis Tom | 2020-05-09 | 2020-05-29
6393 | Згенерована Книга №6440 | Miller Michael | 2020-05-10 | 2020-07-03
3105 | Згенерована Книга №11949 | White Michael | 2020-05-10 | 2020-07-27
12847 | Згенерована Книга №877 | Williams Bob | 2020-05-10 | 2020-08-03
1332 | Згенерована Книга №211 | Jones David | 2020-05-10 | 2020-05-18
2936 | Згенерована Книга №9230 | Wilson Daniel | 2020-05-10 | None
13635 | Згенерована Книга №1912 | White Sarah | 2020-05-10 | None
10786 | Згенерована Книга №14197 | Brown Sarah | 2020-05-10 | 2020-07-26
11007 | Згенерована Книга №231 | Smith Peter | 2020-05-10 | None
5146 | Згенерована Книга №5161 | Rodriguez Tom | 2020-05-10 | 2020-05-26
11512 | Згенерована Книга №2525 | White Michael | 2020-05-10 | 2020-08-06
5493 | Згенерована Книга №9140 | Johnson Mary | 2020-05-10 | 2020-06-30
--------------------------------------------------------------------------------
Запит виконано за 1.496 мс.
```

В результаті отримали **список усіх записів журналу**, дата видачі (loan_date) яких потрапляє у **вказаний користувачем часовий діапазон**.

Кожен запис у списку містить повну інформацію (завдяки JOIN): ID видачі, назву книги, ім'я читача та самі дати. Також було виведено час виконання цього пошукового запиту в мілісекундах.

Перевіримо **аналітичні запити**, що використовують GROUP BY...

```
=== ГОЛОВНЕ МЕНЮ БІБЛІОТЕКИ ===

1. Перегляд даних
2. Додавання запису
3. Редагування запису
4. Видалення запису
5. Генерація випадкових даних
6. Пошук за параметрами
7. Аналітичні запити
0. Вихід із програми
Виберіть опцію: 7

--- ANALYTICS MENU ---
7.1. Кількість виданих книг по кожному автору
7.2. Топ-10 найактивніших читачів
7.3. Середній час користування книгою
7.4. Повернутися до головного меню
Ваш вибір:
```

```
--- ANALYTICS MENU ---
7.1. Кількість виданих книг по кожному автору
7.2. Топ-10 найактивніших читачів
7.3. Середній час користування книгою
7.4. Повернутися до головного меню
Ваш вибір: 7.1
Порада: Введіть прізвище (або його частину) для фільтрації.
       Або залиште поле порожнім, щоб побачити всіх авторів.
Введіть прізвище [Enter для всіх]:: Petrenko

author_id | last_name | first_name | book_count
--------------------------------------------------------------------------------
1 | Petrenko | Ivan | 3
--------------------------------------------------------------------------------
Запит виконано за 12.913 мс.
```

Вводимо прізвище Petrenko і бачимо, що в даного автора є 3 книги.

Якщо натиснемо Enter, то отримаємо весь список авторів, в якому показано скільки у кожного автора написано книг у порядку спадання:

```
--- ANALYTICS MENU ---
7.1. Кількість виданих книг по кожному автору
7.2. Топ-10 найактивніших читачів
7.3. Середній час користування книгою
7.4. Повернутися до головного меню
Ваш вибір: 7.1
Порада: Введіть прізвище (або його частину) для фільтрації.
       Або залиште поле порожнім, щоб побачити всіх авторів.
Введіть прізвище [Enter для всіх]:: 
```

Натискаємо Enter:

```
8180 | Woolf | Stephen | 0
7454 | Woolf | Gabriel | 0
14818 | Woolf | Leo | 0
1317 | Woolf | Agatha | 0
12229 | Woolf | Oscar | 0
7251 | Woolf | Agatha | 0
8840 | Woolf | Margaret | 0
12861 | Woolf | Leo | 0
5712 | Woolf | Margaret | 0
11065 | Woolf | George | 0
9932 | Woolf | Margaret | 0
4923 | Woolf | Agatha | 0
7135 | Woolf | Gabriel | 0
1181 | Woolf | Gabriel | 0
3872 | Woolf | Virginia | 0
3548 | Woolf | Oscar | 0
7076 | Woolf | Oscar | 0
9541 | Woolf | Leo | 0
1121 | Woolf | Agatha | 0
14692 | Woolf | J.K. | 0
12860 | Woolf | Margaret | 0
4203 | Woolf | Jane | 0
2054 | Woolf | Stephen | 0
11822 | Woolf | Agatha | 0
14667 | Woolf | Stephen | 0
11818 | Woolf | Virginia | 0
13929 | Woolf | Virginia | 0
13028 | Woolf | Agatha | 0
4801 | Woolf | Stephen | 0
--------------------------------------------------------------------------------
Запит виконано за 37.868 мс.
```

Знайдемо статистику топ-10 найактивніших читачів:

```
--- ANALYTICS MENU ---
7.1. Кількість виданих книг по кожному автору
7.2. Топ-10 найактивніших читачів
7.3. Середній час користування книгою
7.4. Повернутися до головного меню
Ваш вибір: 7.2

reader_id | last_name | first_name | loan_count
--------------------------------------------------------------------------------
6807 | Jones | Tom | 7
12098 | Smith | Michael | 7
837 | Brown | Michael | 6
6914 | Walker | Sarah | 6
3707 | Williams | David | 6
13806 | Rodriguez | Peter | 6
2424 | Brown | Tom | 5
13322 | White | Sarah | 5
7788 | Garcia | Jennifer | 5
4725 | Rodriguez | Peter | 5
--------------------------------------------------------------------------------
Запит виконано за 10.166 мс.
```

В результаті бачимо **топ-10 найактивніших читачів** у бібліотеці.

Список відсортовано за **загальною кількістю виданих їм книг** (loan_count) у порядку спадання. Для кожного читача у списку відображається його reader_id, прізвище та ім'я, що дозволяє уникнути плутанини між однофамільцями. А також в кінці показує час, за який виконано запит.

Знайдемо середній час користування книгою:

```
--- ANALYTICS MENU ---
7.1. Кількість виданих книг по кожному автору
7.2. Топ-10 найактивніших читачів
7.3. Середній час користування книгою
7.4. Повернутися до головного меню
Ваш вибір: 7.3

avg_duration_days
-------------------------------------------------------------------------------
46.9052089391044280
-------------------------------------------------------------------------------
Запит виконано за 1.610 мс.
```

В результаті бачимо **єдине значення** — середній час (в днях), на який читачі беруть книги.

Цей показник (avg_duration_days) обчислюється як **середня кількість днів** між датою видачі (loan_date) та датою повернення (return_date) по всім записам журналу, де книга вже була повернута (тобто return_date **не** NULL).

SQL-запит для пошуку книг за прізвищем автора і діапазоном років видання

```
SELECT b.book_id, b.name, b.year_published, a.last_name || ' ' || a.first_name AS author_name
FROM book b
JOIN author a ON b.id_author = a.author_id
WHERE a.last_name LIKE %s AND b.year_published BETWEEN %s AND %s
```

Цей запит знаходить **книги**, об'єднуючи (JOIN) таблиці book та author. Він **фільтрує** (WHERE) результати за трьома критеріями, введеними користувачем: прізвищем автора (з ILIKE %s для пошуку за шаблоном) та діапазоном років видання (>= %s та <= %s).

SQL-запит для пошуку читачів за певною назвою книги.

```
SELECT
    r.reader_id,
    r.last_name,
    r.first_name,
    r.email,
    b.name AS book_name

FROM reader r
JOIN "LoanJournal" l ON r.reader_id = l.id_reader
JOIN book b ON l.id_book = b.book_id
WHERE
    b.name ILIKE %s
ORDER BY
    r.last_name, r.first_name, b.name;
```

Цей запит знаходить **читачів**, які брали книги з певною назвою. Він використовує JOIN для зв'язку трьох таблиць (reader, LoanJournal, book). **Фільтрація** (WHERE) відбувається за шаблоном назви книги (ILIKE %s), що дозволяє шукати за частковою назвою (наприклад, "Книга №1000%").

SQL-запит для пошуку записів видачі книг за датами видачі

```
SELECT l.loan_id, b.name, r.last_name || ' ' || r.first_name AS reader, l.loan_date, l.return_date
FROM "LoanJournal" l
JOIN book b ON l.id_book = b.book_id
JOIN reader r ON l.id_reader = r.reader_id
WHERE l.loan_date BETWEEN %s AND %s
ORDER BY l.loan_date
```

Цей запит знаходить **записи про видачу** (LoanJournal), які потрапляють у вказаний користувачем діапазон дат. Він об'єднує (JOIN) три таблиці, щоб показати зрозумілі назви книги та імена читачів замість ID. **Фільтрація** (WHERE) відбувається за датою видачі, яка має бути між початковою (>= %s) та кінцевою (<= %s) датами.

SQL-запит для пошуку кількісті книг за автором

```
SELECT
    a.author_id,
    a.last_name,
    a.first_name,
    COUNT(b.book_id) AS book_count
FROM author a
LEFT JOIN book b ON a.author_id = b.id_author
WHERE a.last_name ILIKE %s
GROUP BY a.author_id, a.last_name, a.first_name
ORDER BY book_count DESC, a.last_name;
```

Цей запит підраховує (COUNT) загальну кількість книг для кожного автора. Він використовує LEFT JOIN, щоб включити у звіт навіть тих авторів, у яких 0 книг. Ключовим є GROUP BY, який збирає всі книги одного автора в один рядок.

Запит містить **статичний** фільтр WHERE a.last_name ILIKE %s. Логіка двох режимів роботи реалізована в **Контролері**, який готує параметр для цього запиту:

1. **Для фільтрації** (якщо користувач ввів King), Контролер надсилає шаблон King%.
2. **Для повного звіту** (якщо користувач натиснув Enter), Контролер надсилає шаблон % (символ, що означає "будь-який текст"), в результаті чого ILIKE % повертає *всіх* авторів."

SQL-запит для пошуку Топ-10 читачів

```
SELECT
    r.reader_id,
    r.last_name,
    r.first_name,
    COUNT(l.loan_id) AS loan_count
FROM reader r
JOIN "LoanJournal" l ON r.reader_id = l.id_reader
GROUP BY r.reader_id, r.last_name, r.first_name
ORDER BY loan_count DESC
LIMIT 10;
```

Цей запит знаходить 10 найактивніших читачів. Він об'єднує (JOIN) читачів та журнал видачі, **групує** (GROUP BY) записи за кожним унікальним читачем і **підраховує** (COUNT) кількість його видач. Потім він

сортує (ORDER BY ... DESC) результат за спаданням кількості та **обмежує** (LIMIT 10) вивід першими 10 рядками.

SQL-запит для пошуку середнього часу користування книгою:

```
SELECT AVG(return_date - loan_date) AS avg_duration_days
FROM "LoanJournal"
WHERE return_date IS NOT NULL AND return_date >= loan_date
```

Цей запит обчислює одне значення: середню кількість днів, на яку читачі беруть книги. Він використовує агрегатну функцію AVG() для обчислення середнього арифметичного від **різниці дат** (return_date - loan_date). Ключовий фільтр WHERE return_date IS NOT NULL гарантує, що у розрахунок потрапляють **лише ті книги, які вже були повернуті**. TRUNC(..., 2) використовується для форматування результату до двох знаків після коми.

Деталізоване завдання №4

Програмний код модуля "Model"

Модуль model.py реалізує шар **Model** у шаблоні MVC. Він єдиний у програмі, хто "знає" про існування бази даних, мову SQL та бібліотеку psycopg2. Він повністю інкапсулює всю логіку роботи з даними, надаючи Контролеру простий та зрозумілий "API" для доступу до них.

Основні групи функцій модуля:

- **__init__ та create_tables**:
  - __init__: Встановлює з'єднання з базою даних PostgreSQL. Налаштовує autocommit=False для ручного керування транзакціями та DictCursor для зручного доступу до даних за назвами колонок. При успішному з'єднанні викликає create_tables.

```
def __init__(self):
        try:
```

```
            self.conn = psycopg2.connect(
                dbname='postgres',
                user='postgres',
                password='1111',
                host='localhost',
                options='-c search_path=auth,public',
                port=5432
            )
            self.conn.autocommit = False
            self.cursor =
self.conn.cursor(cursor_factory=psycopg2.extras.DictCursor)
            print("З'єднання з БД успішно встановлено.")
            self.create_tables()
        except psycopg2.Error as e:
            print(f"Помилка підключення до БД: {e}")
            exit(1)
```

- create_tables: Виконує набір SQL-команд CREATE TABLE IF NOT EXISTS для створення всієї структури бази даних (author, reader, book, "LoanJournal"), включно з обмеженнями PRIMARY KEY, FOREIGN KEY (з ON DELETE RESTRICT) та UNIQUE.

```
def create_tables(self):
        try:
            self.cursor.execute("""
                CREATE TABLE IF NOT EXISTS author (
                    author_id SERIAL PRIMARY KEY,
                    last_name TEXT NOT NULL,
                    first_name TEXT NOT NULL,
                    email TEXT UNIQUE
                );
            """)
            self.cursor.execute("""
                CREATE TABLE IF NOT EXISTS reader (
                    reader_id SERIAL PRIMARY KEY,
                    last_name TEXT NOT NULL,
                    first_name TEXT NOT NULL,
                    email TEXT UNIQUE
                );
            """)
            self.cursor.execute("""
                CREATE TABLE IF NOT EXISTS book (
```

```
                    book_id SERIAL PRIMARY KEY,
                    name TEXT NOT NULL,
                    year_published INT,
                    pages INT,
                    id_author INT,
                    CONSTRAINT fk_author
                        FOREIGN KEY(id_author)
                        REFERENCES author(author_id)
                        ON DELETE RESTRICT
                );
            """)
            self.cursor.execute("""
                CREATE TABLE IF NOT EXISTS "LoanJournal" (
                    loan_id SERIAL PRIMARY KEY,
                    id_book INT NOT NULL,
                    id_reader INT NOT NULL,
                    loan_date DATE NOT NULL DEFAULT CURRENT_DATE,
                    return_date DATE,
                    CONSTRAINT fk_book
                        FOREIGN KEY(id_book)
                        REFERENCES book(book_id)
                        ON DELETE RESTRICT,
                    CONSTRAINT fk_reader
                        FOREIGN KEY(id_reader)
                        REFERENCES reader(reader_id)
                        ON DELETE RESTRICT,
                    CHECK (return_date IS NULL OR return_date >=
loan_date)
                );
            """)
            self.conn.commit()
            print("Таблиці успішно перевірені/створені.")
        except psycopg2.Error as e:
            print(f"Помилка при створенні таблиць: {e}")
            self.conn.rollback()
```

- **Приватні "виконавці" (_execute_query та _execute_dml)**:
    - _execute_query: Централізований метод для **всіх SELECT**-запитів (включно з пошуком та аналітикою). Він виконує запит, commit (оскільки autocommit вимкнено) та повертає результат (fetchall). Обробляє помилки читання.

```
def _execute_query(self, query, params=None, fetch=True):
        try:
            self.cursor.execute(query, params)
            self.conn.commit()
            if fetch:
                return self.cursor.fetchall()
            return None
        except (psycopg2.Error, psycopg2.DataError) as e:
            print(f"Помилка читання (SELECT): {e}")
            self.conn.rollback()
            return None
```

- _execute_dml: Централізований метод для **всіх INSERT, UPDATE, DELETE**-запитів (DML). Це ключовий метод для обробки помилок цілісності: він має try...except блоки для перехоплення специфічних помилок PostgreSQL (UniqueViolation, ForeignKeyViolation тощо), виконує rollback() транзакції у разі невдачі та повертає Контролеру кортеж (Успіх, Повідомлення_про_помилку).

```
def _execute_dml(self, query, params=None):
        try:
            self.cursor.execute(query, params)
            self.conn.commit()
            return (True, None)

        except psycopg2.errors.UniqueViolation as e:
            self.conn.rollback()
            return (False, "Порушення унікальності. Можливо, такий
email вже існує.")

        except psycopg2.errors.ForeignKeyViolation as e:
            self.conn.rollback()
            return (False, "Порушення зв'язності даних. (Наприклад, не
можна видалити сутність, на яку посилається журнал).")

        except psycopg2.errors.NotNullViolation as e:
            self.conn.rollback()
            return (False, f"Не заповнене обов'язкове поле.")

        except (psycopg2.Error, psycopg2.DataError) as e:
            self.conn.rollback()
```

```
            return (False, f"Загальна помилка SQL: {e}")
```

- **Методи CRUD (Create, Read, Update, Delete)**:

Методи `Read` (Читання)

Ця група методів відповідає за **отримання** (SELECT) даних з бази даних. Вони викликаються Контролером, коли користувач обирає **Меню 1 (Перегляд)**. Усі вони використовують _execute_query як безпечний та централізований спосіб виконання SELECT-запитів.

- **get_authors() та get_readers()**: Це найпростіші запити. Вони виконують SELECT * FROM ... ORDER BY ..._id для отримання повного списку всіх авторів або читачів, відсортованих за їхніми ID для послідовного відображення.

```
def get_authors(self):
        return self._execute_query("SELECT * FROM author ORDER BY
author_id", fetch=True)
def get_readers(self):
        return self._execute_query("SELECT * FROM reader ORDER BY
reader_id", fetch=True)
```

- **get_books() та get_loans()**: Ці методи є складнішими, оскільки вони використовують **LEFT JOIN** (для get_books) або **JOIN** (для get_loans). Це робиться для того, щоб надати користувачеві *зрозумілу* інформацію.
    - Замість того, щоб get_books повертав незрозумілий id_author, він об'єднує таблицю book з author і повертає author_name.
    - Аналогічно, get_loans об'єднує журнал видачі з таблицями book та reader, щоб у звіті відображалися назва книги та ім'я читача, а не їхні ID.

```
 def get_books(self):
        query = """
            SELECT b.book_id, b.name, b.year_published, b.pages,
a.last_name || ' ' || a.first_name AS author_name
            FROM book b LEFT JOIN author a ON b.id_author = a.author_id
            ORDER BY b.book_id
        """
        return self._execute_query(query, fetch=True)
```

```
def get_loans(self):
        query = """
            SELECT l.loan_id, b.name AS book_title, r.last_name || ' '
|| r.first_name AS reader_name, l.loan_date, l.return_date
            FROM "LoanJournal" l
            JOIN book b ON l.id_book = b.book_id
            JOIN reader r ON l.id_reader = r.reader_id
            ORDER BY l.loan_id ASC
        """
        return self._execute_query(query, fetch=True)
```

Методи Create (Створення)

Ця група методів відповідає за **додавання** (INSERT) нових записів у базу даних. Вони викликаються Контролером з **Меню 2 (Додавання)**.

- add_author(last_name, first_name, email)
- add_book(name, year_published, pages, id_author)
- add_reader(last_name, first_name, email)
- add_loan(id_book, id_reader, loan_date, return_date)

Усі ці методи просто формують SQL-запит INSERT INTO ... VALUES (%s, %s, ...) і передають його разом із параметрами (отриманими від Контролера) у приватний метод **_execute_dml**.

Вони не обробляють логіку самі. Їхня єдина відповідальність — передати дані "виконавцю" (_execute_dml), який уже виконає commit у разі успіху або rollback у разі помилки. Якщо _execute_dml перехопить помилку (наприклад, UniqueViolation, якщо користувач спробує додати email, що вже існує), цей метод поверне Контролеру кортеж (False, "Порушення унікальності...").

```
def add_author(self, last_name, first_name, email):
        query = "INSERT INTO author (last_name, first_name, email)
VALUES (%s, %s, %s)"
        return self._execute_dml(query, (last_name, first_name, email))

    def add_reader(self, last_name, first_name, email):
        query = "INSERT INTO reader (last_name, first_name, email)
VALUES (%s, %s, %s)"
        return self._execute_dml(query, (last_name, first_name, email))
```

```
    def add_book(self, name, year_published, pages, id_author):
        query = "INSERT INTO book (name, year_published, pages,
id_author) VALUES (%s, %s, %s, %s)"
        return self._execute_dml(query, (name, year_published, pages,
id_author))

    def add_loan(self, id_book, id_reader, loan_date, return_date):
        query = 'INSERT INTO "LoanJournal" (id_book, id_reader,
loan_date, return_date) VALUES (%s, %s, %s, %s)'
        return self._execute_dml(query, (id_book, id_reader, loan_date,
return_date))
```

Методи Update (Оновлення)

Ця група методів відповідає за **модифікацію** (UPDATE) існуючих записів. Вони викликаються Контролером з **Меню 3 (Редагування)**.

- update_author(author_id, last_name, first_name, email)
- update_book(book_id, name, year_published, pages, id_author)
- update_reader(reader_id, last_name, first_name, email)
- update_loan(loan_id, id_book, id_reader, loan_date, return_date)

Ці методи схожі на add_..., але їхні SQL-запити використовують ключове слово UPDATE ... SET ... та **WHERE ..._id = %s**. Це гарантує, що оновиться лише той запис, ID якого було передано.

Вони також повністю покладаються на _execute_dml для виконання транзакції та обробки помилок, таких як UniqueViolation (якщо email оновили на той, що вже існує) або ForeignKeyViolation (якщо id_author у книзі оновили на неіснуючий).

```
def update_author(self, author_id, last_name, first_name, email):
        query = "UPDATE author SET last_name = %s, first_name = %s,
email = %s WHERE author_id = %s"
        return self._execute_dml(query, (last_name, first_name, email,
author_id))

    def update_reader(self, reader_id, last_name, first_name, email):
        query = "UPDATE reader SET last_name = %s, first_name = %s,
email = %s WHERE reader_id = %s"
        return self._execute_dml(query, (last_name, first_name, email,
reader_id))
```

```
    def update_book(self, book_id, name, year_published, pages,
id_author):
        query = "UPDATE book SET name = %s, year_published = %s, pages
= %s, id_author = %s WHERE book_id = %s"
        return self._execute_dml(query, (name, year_published, pages,
id_author, book_id))

    def update_loan(self, loan_id, id_book, id_reader, loan_date,
return_date):
        query = 'UPDATE "LoanJournal" SET id_book = %s, id_reader = %s,
loan_date = %s, return_date = %s WHERE loan_id = %s'
        return self._execute_dml(query, (id_book, id_reader, loan_date,
return_date, loan_id))
```

Методи Delete (Видалення)

Ця група методів відповідає за **видалення** (DELETE) записів. Вони викликаються Контролером з **Меню 4 (Видалення)**.

- delete_author(author_id)
- delete_book(book_id)
- delete_reader(reader_id)
- delete_loan(loan_id)

Кожен метод формує простий SQL-запит DELETE FROM ... WHERE ..._id = %s і передає його у **_execute_dml**.

Найважливіша логіка тут прихована в _execute_dml та схемі БД. Якщо Контролер викликає delete_author(1), а в базі даних є книги, що посилаються на цього автора (з id_author = 1), то обмеження ON DELETE RESTRICT у CREATE TABLE спрацює. PostgreSQL згенерує помилку, _execute_dml її **перехопить** (except psycopg2.errors.ForeignKeyViolation), виконає self.conn.rollback() і поверне Контролеру зрозуміле повідомлення (False, "Порушення зв'язності даних...") замість "падіння" програми.

```
    def delete_author(self, author_id):
            query = "DELETE FROM author WHERE author_id = %s"
            return self._execute_dml(query, (author_id,))

        def delete_book(self, book_id):
            query = "DELETE FROM book WHERE book_id = %s"
```

```
            return self._execute_dml(query, (book_id,))

        def delete_reader(self, reader_id):
            query = "DELETE FROM reader WHERE reader_id = %s"
            return self._execute_dml(query, (reader_id,))

        def delete_loan(self, loan_id):
            query = 'DELETE FROM "LoanJournal" WHERE loan_id = %s'
            return self._execute_dml(query, (loan_id,))
```

- **Методи валідації**:
    - get_entity_by_id: Оптимізований запит SELECT 1, який Контролер використовує для перевірки існування ID *перед* тим, як виконати операцію (напр., перевірити, чи існує автор, перш ніж додати книгу).

```
def get_entity_by_id(self, entity_name, entity_id):
        allowed_entities = {
            'author': ('author', 'author_id'),
            'book': ('book', 'book_id'),
            'reader': ('reader', 'reader_id'),
            'LoanJournal': ('"LoanJournal"', 'loan_id')
        }

        if entity_name not in allowed_entities:
            return None

        table_name, id_column = allowed_entities[entity_name]
        query = f"SELECT 1 FROM {table_name} WHERE {id_column} = %s"

        result = self._execute_query(query, (entity_id,), fetch=True)
        return result
```

    - get_book_validation_details: Допоміжний метод, що повертає рік видання книги для валідації в Контролері.

```
def get_book_validation_details(self, book_id):
        query = "SELECT year_published, name FROM book WHERE book_id =
%s"
        result = self._execute_query(query, (book_id,), fetch=True)
```

```
        if result:
            return result[0]
        return None
```

- **Методи генерації (generate_...)**:

Ці методи викликаються Контролером з **Меню 5 (Генерація)**. Їхня мета — швидко наповнити базу даних тисячами тестових записів.

Вони **не** використовують Python для створення даних у циклі. Замість цього, вони виконують **єдиний, складний SQL-запит** INSERT INTO ... SELECT ..., який змушує сервер PostgreSQL виконати всю роботу. Це **надзвичайно ефективно** і відповідає вимогам про генерацію даних засобами SQL.

- **generate_authors(count) та generate_readers(count)**:

Використовують generate_series(1, %s) для створення SQL-"циклу" на count ітерацій.

Щоб уникнути помилки UniqueViolation при створенні 15 000 однакових email, вони "склеюють" (||) номер ітерації з циклу (i::text) до пошти. Це гарантує унікальний email (напр., author.1@..., author.2@...).

Імена та прізвища вибираються випадково з array[...] на кожній ітерації.

```
def generate_authors(self, count):
        query = """
            INSERT INTO author (first_name, last_name, email)
            SELECT
                (array[
                    'Stephen', 'George', 'Jane', 'Haruki', 'Agatha',
'Ernest',
                    'Virginia', 'Oscar', 'Leo', 'Margaret', 'J.K.',
'Gabriel'
                ])[floor(random() * 12 + 1)] AS first_name,

                (array[
                    'King', 'Orwell', 'Austen', 'Murakami', 'Christie',
'Hemingway',
                    'Woolf', 'Wilde', 'Tolstoy', 'Atwood', 'Rowling',
'Garcia Marquez'
                ])[floor(random() * 12 + 1)] AS last_name,
```

```
                'author.' || i::text || '@authors.com' AS email
            FROM
                generate_series(1, %s) AS s(i);
        """
        return self._execute_dml(query, (count,))
    def generate_readers(self, count):
        query = """
            INSERT INTO reader (first_name, last_name, email)
            SELECT
                (array[
                    'John', 'Ann', 'Bob', 'Alice', 'Peter', 'Mary',
'David',
                    'Michael', 'Sarah', 'Chris', 'Emily', 'James',
'Jennifer', 'Daniel', 'Tom'
                ])[floor(random() * 15 + 1)] AS first_name,

                (array[
                    'Smith', 'Nelson', 'Wilson', 'Brown', 'Davis',
'Miller', 'Johnson',
                    'Williams', 'Jones', 'Garcia', 'Rodriguez', 'Lee',
'Walker', 'Hall', 'White'
                ])[floor(random() * 15 + 1)] AS last_name,

                'reader.' || i::text || '@library.ua' AS email
            FROM
                generate_series(1, %s) AS s(i);
        """
        return self._execute_dml(query, (count,))
```

- **generate_books(count) та generate_loans(count)**:

Також використовують generate_series(1, %s) для створення SQL-"циклу" на count ітерацій.

Щоб уникнути помилки ForeignKeyViolation (спроби створити книгу з неіснуючим id_author), запит спочатку використовує WITH ... AS (SELECT array_agg(...) ...) для збору **всіх реально існуючих** ID авторів (або книг/читачів) в один масив.

На кожній ітерації циклу запит вибирає випадковий, але **гарантовано існуючий** ID з цього попередньо зібраного масиву. generate_loans є найскладнішим, оскільки він збирає *два* таких масиви (для id_book та id_reader).

```
    def generate_books(self, count):
            query = """
                INSERT INTO book (name, year_published, pages,
id_author)
                WITH authors AS (
                    SELECT array_agg(author_id) AS ids FROM author
                )
                SELECT
                    'Згенерована Книга №' || s.id,
                    floor(random() * (2024 - 1950 + 1) + 1950)::int,
                    floor(random() * (800 - 100 + 1) + 100)::int,
                    a.ids[floor(random() * array_length(a.ids, 1) + 1
+ (s.id * 0))]
                FROM
                    generate_series(1, %s) AS s(id),
                    authors a
                WHERE
                    a.ids IS NOT NULL;
            """
            return self._execute_dml(query, (count,))

        def generate_loans(self, count):
            query = """
                INSERT INTO "LoanJournal" (id_book, id_reader,
loan_date, return_date)
                WITH
                    books AS (
                        SELECT array_agg(book_id) AS ids FROM book
                    ),
                    readers AS (
                        SELECT array_agg(reader_id) AS ids FROM
reader
                    ),
                    GeneratedData AS (
                        SELECT
                            b.ids[floor(random() *
array_length(b.ids, 1) + 1 + (s.id * 0))] AS b_id,
```

```
                            r.ids[floor(random() *
array_length(r.ids, 1) + 1 + (s.id * 0))] AS r_id,
                            (timestamp '2020-01-01' + random() *
(timestamp '2023-11-01' - timestamp '2020-01-01')) AS i_date
                        FROM
                            generate_series(1, %s) s(id),
                            books b,
                            readers r
                        WHERE
                            b.ids IS NOT NULL AND r.ids IS NOT NULL
                    )
                SELECT
                    b_id,
                    r_id,
                    i_date::date,
                    CASE WHEN random() > 0.2
                         THEN (i_date + (floor(random() * 85 + 5) ||
' days')::interval)::date
                         ELSE NULL
                    END
                FROM GeneratedData;
            """
            return self._execute_dml(query, (count,))
```

- **Методи пошуку та аналітики (search_..., get_...)**:

Ці методи викликаються з **Меню 6 (Пошук)** та **Меню 7 (Аналітика)**. Вони є реалізацією складних SELECT-запитів, які фільтрують, об'єднують та агрегують дані.

Методи пошуку (Меню 6)

Основна мета — **фільтрація** (WHERE).

- **search_books_by_author_year(...)**: Використовує JOIN для зв'язку book та author і фільтрує результат за трьома критеріями (WHERE a.last_name ILIKE %s AND b.year_published >= %s AND b.year_published <= %s).

```
def search_books_by_author_year(self, author_last_name, start_year,
end_year):
```

```
        query = """
            SELECT b.book_id, b.name, b.year_published, a.last_name ||
' ' || a.first_name AS author_name
            FROM book b
            JOIN author a ON b.id_author = a.author_id
            WHERE a.last_name LIKE %s AND b.year_published BETWEEN %s
AND %s
        """
        return self._execute_query(query, (f"%{author_last_name}%",
start_year, end_year), fetch=True)
```

- **search_readers_by_book_title(...)**: Використовує JOIN для зв'язку трьох таблиць (reader, LoanJournal, book). Фільтрує записи, де назва книги відповідає шаблону ILIKE %s (напр., Згенерована Книга №1000%), який готує Контролер.

```
def search_readers_by_book_title(self, book_name_pattern):
        query = """
            SELECT
                r.reader_id,
                r.last_name,
                r.first_name,
                r.email,
                b.name AS book_name

            FROM reader r
            JOIN "LoanJournal" l ON r.reader_id = l.id_reader
            JOIN book b ON l.id_book = b.book_id
            WHERE
                b.name ILIKE %s
            ORDER BY
                r.last_name, r.first_name, b.name;
        """
        return self._execute_query(query, (book_name_pattern,))
```

- **search_loans_by_date_range(...)**: Простий фільтр, який знаходить записи журналу, де loan_date знаходиться BETWEEN %s AND %s.

```
def search_loans_by_date_range(self, start_date, end_date):
        query = """
            SELECT l.loan_id, b.name, r.last_name || ' ' ||
r.first_name AS reader, l.loan_date, l.return_date
            FROM "LoanJournal" l
```

```
            JOIN book b ON l.id_book = b.book_id
            JOIN reader r ON l.id_reader = r.reader_id
            WHERE l.loan_date BETWEEN %s AND %s
            ORDER BY l.loan_date
        """
        return self._execute_query(query, (start_date, end_date),
fetch=True)
```

Методи аналітики (Меню 7)

Основна мета — **агрегація** (GROUP BY, COUNT, AVG).

- **get_books_per_author(last_name_pattern)**: Це запит з **"розумною" фільтрацією**.

Він використовує COUNT(b.book_id) та GROUP BY a.author_id, щоб підрахувати кількість книг *для кожного* автора.

LEFT JOIN використовується, щоб у звіт потрапили навіть ті автори, у яких 0 книг.

Метод завжди очікує параметр last_name_pattern. **Контролер** вирішує, що надіслати: якщо користувач ввів King, Контролер надсилає King%; якщо користувач натиснув Enter, Контролер надсилає % (шаблон, що означає "будь-який символ"), що ефективно повертає всіх авторів.

```
def get_books_per_author(self, last_name_pattern):
        query = """
            SELECT
                a.author_id,
                a.last_name,
                a.first_name,
                COUNT(b.book_id) AS book_count
            FROM author a
            LEFT JOIN book b ON a.author_id = b.id_author
            WHERE a.last_name ILIKE %s
            GROUP BY a.author_id, a.last_name, a.first_name
            ORDER BY book_count DESC, a.last_name;
        """
        return self._execute_query(query, (last_name_pattern,))
```

- **get_top_10_readers()**: Класичний "Топ-N" запит. Він підраховує (COUNT) видачі для кожного читача (GROUP BY r.reader_id), сортує їх за спаданням (ORDER BY ... DESC) і повертає лише перші 10 (LIMIT 10).

```
def get_top_10_readers(self):
        query = """
            SELECT
                r.reader_id,
                r.last_name,
                r.first_name,
                COUNT(l.loan_id) AS loan_count
            FROM reader r
            JOIN "LoanJournal" l ON r.reader_id = l.id_reader
            GROUP BY r.reader_id, r.last_name, r.first_name
            ORDER BY loan_count DESC
            LIMIT 10;
        """
        return self._execute_query(query, fetch=True)
```

- **get_avg_loan_duration()**: Обчислює одне значення. Він використовує AVG(return_date - loan_date) для розрахунку середньої кількості днів. Найважливіша частина — WHERE return_date IS NOT NULL, що гарантує, що у розрахунок не потраплять книги, які ще на руках у читачів. TRUNC використовується для форматування результату.

```
def get_avg_loan_duration(self):
        query = """
            SELECT AVG(return_date - loan_date) AS avg_duration_days
            FROM "LoanJournal"
            WHERE return_date IS NOT NULL AND return_date >= loan_date
        """
        return self._execute_query(query, fetch=True)
```

- **close**:
    - Цей метод відповідає за **коректне закриття з'єднання** з базою даних. Він викликається Контролером один раз при завершенні роботи програми (у блоці finally, що гарантує його виконання навіть у разі помилки).

```
def close(self):
        if self.cursor:
            self.cursor.close()
        if self.conn:
            self.conn.close()
            print("З'єднання з БД закрито.")
```

Висновок:

Під час виконання цієї розрахунково-графічної роботи було успішно розроблено та реалізовано консольний додаток для взаємодії з базою даних бібліотеки на СУБД **PostgreSQL**. Мета роботи полягала у здобутті практичних навичок програмування, що було реалізовано шляхом розробки повнофункціонального консольного додатку.

Вся архітектура програми була побудована згідно з шаблоном **MVC** (Model-View-Controller), що дозволило чітко розділити відповідальність:

- **model.py**: Інкапсулює всю логіку роботи з базою даних, з'єднання (через бібліотеку **psycopg2** без ORM) та виконання виключно SQL-запитів.
- **view.py**: Відповідає за весь консольний інтерфейс, відображення меню, отримання вводу від користувача та валідацію типів даних.
- **controller.py**: Виступає як "мозок" програми, який керує потоком, обробляє логіку меню, викликає методи Моделі та передає дані для відображення у Подання.

У ході роботи було реалізовано наступний функціонал:

1. **Повний CRUD-функціонал**: Створено меню для перегляду (SELECT), додавання (INSERT), редагування (UPDATE) та видалення (DELETE) даних у всіх таблицях.
2. **Контроль цілісності даних**: Це ключове завдання було вирішено двома шляхами:
    - **На рівні БД:** Завдяки обмеженню ON DELETE RESTRICT у схемі БД та перехопленню помилки psycopg2.errors.ForeignKeyViolation в model.py, програма коректно блокує видалення "батьківських" записів (напр., автора), на які існують посилання.

    - **На рівні Контролера:** Перед додаванням "дочірніх" записів (напр., книги), Контролер виконує валідацію — перевіряє існування id_author за допомогою методу model.get_entity_by_id().
    - Також реалізовано перехоплення UniqueViolation (для email) та інших помилок psycopg2, що запобігає "падінню" програми.
3. **Масова генерація даних**: Відповідно до завдання, реалізовано модуль генерації, який виконує SQL-запити (INSERT INTO ... SELECT ...), а не генерує дані мовою Python. Використання **generate_series** дозволило створити будь-яку кількість записів (протестовано на 15000+). Проблеми цілісності при генерації було вирішено засобами SQL:
    - **Унікальність (**UNIQUE**)**: Гарантована через конкатенацію з номером ітерації ('author.' || i::text || ...).
    - **Зовнішні ключі (**FOREIGN KEY**)**: Гарантовані через збір існуючих ID в масив (WITH ... AS (SELECT array_agg(...) ...)), з якого потім обиралося випадкове, але "безпечне" значення.
4. **Комплексний пошук та аналітика**: Реалізовано 3 пошукові запити (Меню 6) та 3 аналітичні запити (Меню 7).
    - Запити використовують JOIN для зв'язку кількох таблиць, WHERE для фільтрації за введеними користувачем параметрами (діапазони дат, шаблони ILIKE) та GROUP BY з агрегатними функціями (COUNT, AVG) для аналітики.
    - Реалізовано гнучкий пошук, де Контролер готує параметр (напр., King% для фільтра або % для всіх записів), який передається у статичний SQL-запит в Моделі.
    - Усі ці запити вимірюють час свого виконання у мілісекундах.

Таким чином, розроблений додаток повністю відповідає всім поставленим у завданні вимогам, демонструючи вміння керувати транзакціями, обробляти помилки цілісності, оптимізувати SQL-запити для генерації та пошуку, і застосовувати шаблон MVC.