EA black and white drawing of a building

Description automatically generated

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського» ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра системного програмування та спеціалізованих комп’ютерних систем

**Розрахунково-графічна робота**

з дисципліни **Бази даних і засоби управління**

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

Виконав:

студент ІІI курсу

групи КВ-21

Саюк В.А.

Перевірив:

Київ – 2024

***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).

**Опис предметної області**

Предметна область – *онлайн-платформа для зберігання та пошуку наукових публікацій.*

Вона охоплює набір засобів і інструментів, які спрямовані на ефективне управління та доступ до наукових досліджень і статей. Вона дозволяє зберігати різноманітні типи публікацій, включаючи журнальні статті, тези конференцій, наукові звіти та дисертації, надаючи можливість їх швидкого пошуку за ключовими словами, авторами або темами. Ця система спрощує доступ до актуальних наукових знань, підтримуючи дослідників, викладачів та студентів у пошуку необхідних матеріалів. Завдяки аналітичним інструментам платформи користувачі можуть відстежувати популярність публікацій, цитування та останні тенденції в певних галузях науки, що сприяє розвитку дослідницької діяльності та наукової співпраці.

**Опис сутностей**

Для побудови бази даних для *онлайн-платформи зберігання та пошуку наукових публікацій*, були виділені такі сутності:

Collection (Збірник)

- Атрибути:

- *Collection\_id*: Ідентифікатор збірника (унікальний ключ для кожного збірника).

- *name*: Назва збірника, яка визначає його унікальність або зміст (наприклад, "Збірник наукових праць").

- *type*: Формат видання збірника, що може включати підручники, журнальні статті, матеріали конференцій тощо.

- *view*: Статус збірника, який вказує на географічну спрямованість: всеукраїнський (національний) або міжнародний.

**Призначення:**

Збірник представляє собою організовану колекцію наукових публікацій, підручників або інших навчальних матеріалів. Він дозволяє структурувати та систематизувати публікації за певними критеріями, що полегшує доступ до необхідної інформації.

Edition (Видання)

- Атрибути:

- *Edition\_id*: Унікальний ідентифікатор видання.

- *name*: Назва видання, що може вказувати на тему або серію публікацій (наприклад, "Науковий вісник" або "Журнал з фізики").

- *branch*: Галузь або науковий напрямок, до якого належить видання (наприклад, фізика, біологія, історія тощо).

- *number\_of\_pages*: Кількість сторінок у виданні, що може вказувати на обсяг матеріалу.

- *languages*: Мови, якими видання доступне (наприклад, українська, англійська, німецька).

**Призначення:**

Видання є основною одиницею публікації наукових або освітніх матеріалів. Його мета полягає у наданні інформації та досліджень у певній галузі, допомагаючи розповсюджувати знання серед наукової спільноти або студентів.

Author (Автор)

- Атрибути:

- *Author\_ID*: Унікальний ідентифікатор автора.

- *name*: Ім'я автора.

- *surname*: Прізвище автора.

**- Призначення:**

Автор є ключовою фігурою у створенні наукових публікацій або навчальних матеріалів. Він несе відповідальність за зміст публікації, її якість та наукову значущість. В системі обліку автор дозволяє відстежувати внесок конкретної особи у певні публікації та спрощує пошук робіт за іменем чи прізвищем автора.

**Опис зв’язків між сутностями**

Ці три сутності(Author,Edition, Collection) взаємопов'язані через тернарний зв'язок, що представляє відносини *"багато до багатьох" (N:M).*

Це означає, що один і той самий автор може бути залучений до створення численних публікацій, які видаються в різних збірниках. Водночас одна публікація може мати кілька авторів і бути частиною одного збірника.

Такий тип зв'язку дозволяє гнучко керувати даними про авторів, видання та збірники, адже публікація може бути результатом колективної роботи кількох науковців, а також може входити до різних збірників або випусків. Така модель також спрощує роботу з багатоспрямованими дослідженнями та міждисциплінарними проєктами, де участь різних авторів та вихід у кількох збірниках є нормою.

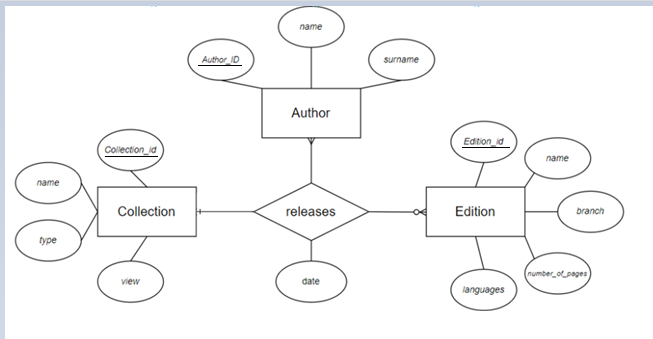


Рисунок 1 – ER-діаграма, побудована за нотацією “Вороняча лапка”

**A screenshot of a computer

Description automatically generated**

Рисунок.2 Схема бази даних

**Програма**

Вигляд початкового меню:

A screenshot of a computer program

Description automatically generated

*Під час запуску відображається початкове меню в якому пропонується обрати таблицю для подальшої роботи (1-5) або запити (6).*

Обравши таблицю, відображаються наступне меню:

A screenshot of a computer

Description automatically generated

*Для таблиць доступні додавання (1), перегляд (2), оновлення (3), вилучення (4) та генерування даних (5). Опція виходу (6).*

Додамо автора та видання:

A screenshot of a computer

Description automatically generated

Результат вставки в дочірню таблицю даних яких не існує в батьківських:

*Додам дані в таблицю Author\_Collection\_Edition з неіснуючим ID*

A white background with black text

Description automatically generated

*У випадку ж додавання існуючого помилка не виникає:*

A screenshot of a computer

Description automatically generated

Перейдемо до генерації даних:

*Генерація 3 випадкових збірників:*

A screenshot of a computer

Description automatically generated

*Додам запис в Author\_Collection\_Edition та Author\_Collection\_Edition\_ED:*

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Тепер перевіримо програму на вилучення з батьківської таблиці даних:

*Було реалізовано каскадне видалення. Якщо запис про автора видаляється з таблиці, то будуть видалені і відповідні записи за таблиці Author\_Collection\_Edition*

**Приклад:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer code

Description automatically generated

*Тепер видалю запис з Author і перегляну, чи залишиться запис у Author\_collection\_edition:*

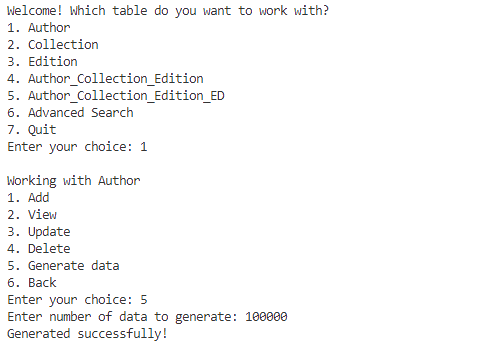
A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

Генерація 100 000 даних у таблицю Author:



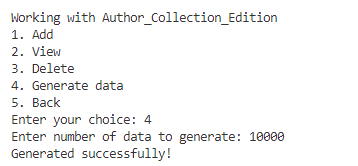
A screenshot of a computer

Description automatically generated

Згенеруємо дані і для таблиці Edition та Author\_Collection\_Edition:

A screenshot of a computer program

Description automatically generated



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

SQL запити, які використовувались для генерації:

if table\_name == "Author":

                # Insert random authors

                for \_ in range(num):

                    name = ''.join(random.choice(string.ascii\_letters) for \_ in range(5))

                    surname = ''.join(random.choice(string.ascii\_letters) for \_ in range(7))

                    c.execute('INSERT INTO "Author" ("name","surname") VALUES (%s,%s)', (name, surname))

            elif table\_name == "Collection":

                for \_ in range(num):

                    col\_name = ''.join(random.choice(string.ascii\_letters) for \_ in range(8))

                    col\_type = random.choice(["Literature", "Fiction", "Poetry", "Science"])

                    col\_view = random.choice(["Paper", "Electronic"])

                    c.execute('INSERT INTO "Collection" ("name","type","view") VALUES (%s,%s,%s)', (col\_name, col\_type, col\_view))

            elif table\_name == "Edition":

                for \_ in range(num):

                    ed\_name = ''.join(random.choice(string.ascii\_letters) for \_ in range(6))

                    branch = random.choice(["Fiction", "Poetry", "Non-Fiction", "Science"])

                    pages = random.randint(50, 1000)

                    languages = random.choice(["English", "Ukrainian", "French"])

                    c.execute('INSERT INTO "Edition" ("name","branch","number\_of\_pages","languages") VALUES (%s,%s,%s,%s)', (ed\_name, branch, pages, languages))

            elif table\_name == "Author\_Collection\_Edition":

                # For linking table, we need existing authors and editions

                c.execute('SELECT "author\_id" FROM "Author"')

                authors = [row[0] for row in c.fetchall()]

                c.execute('SELECT "edition\_id" FROM "Edition"')

                editions = [row[0] for row in c.fetchall()]

                if not authors or not editions:

                    self.view.show\_message("No authors or editions to link. Generate them first.")

                    return

                for \_ in range(num):

                    aid = random.choice(authors)

                    eid = random.choice(editions)

                    try:

                        c.execute('INSERT INTO "Author\_Collection\_Edition" ("author\_id","edition\_id") VALUES (%s,%s)', (aid, eid))

                    except psycopg2.errors.UniqueViolation:

                        self.conn.rollback()  # ignore duplicates

                        continue

            elif table\_name == "Author\_Collection\_Edition\_ED":

                # We need existing editions and collections

                c.execute('SELECT "edition\_id" FROM "Edition"')

                editions = [row[0] for row in c.fetchall()]

                c.execute('SELECT "collection\_id" FROM "Collection"')

                collections = [row[0] for row in c.fetchall()]

                if not editions or not collections:

                    self.view.show\_message("No editions or collections to link. Generate them first.")

                    return

                for \_ in range(num):

                    eid = random.choice(editions)

                    cid = random.choice(collections)

                    # Random date

                    year = random.randint(1800, 2020)

                    month = random.randint(1,12)

                    day = random.randint(1,28)

                    date\_str = f"{year}-{month:02d}-{day:02d}"

                    try:

                        c.execute('INSERT INTO "Author\_Collection\_Edition\_ED" ("edition\_id","collection\_id","date") VALUES (%s,%s,%s)', (eid, cid, date\_str))

                    except psycopg2.errors.UniqueViolation:

                        self.conn.rollback()

                        continue

Перейдемо до SQL запиту.

Я вирішив реалізувати запит, який виконуватиме пошук за певною комбінацією літер в прізвищі автора:

def advanced\_search(self, surname\_pattern, min\_pages, max\_pages, start\_date, end\_date):

        # Example advanced query

        # Joins across multiple tables and applies filters on numeric range, string pattern, and date range

        c = self.conn.cursor()

        sql = """

            SELECT a.name, a.surname, ed.name as edition\_name, ed.number\_of\_pages, aced.date, c.name as collection\_name

            FROM "Author" a

            JOIN "Author\_Collection\_Edition" ace ON a.author\_id = ace.author\_id

            JOIN "Edition" ed ON ace.edition\_id = ed.edition\_id

            JOIN "Author\_Collection\_Edition\_ED" aced ON ed.edition\_id = aced.edition\_id

            JOIN "Collection" c ON aced.collection\_id = c.collection\_id

            WHERE a.surname ILIKE %s

              AND ed.number\_of\_pages BETWEEN %s AND %s

              AND aced.date BETWEEN %s AND %s

            ORDER BY a.surname, ed.number\_of\_pages;

        """

        c.execute(sql, (f"%{surname\_pattern}%", min\_pages, max\_pages, start\_date, end\_date))

        rows = c.fetchall()

        return rows

Для цього було згенеровано дані для кожної таблиці, після чого вводячи уточнюючі дані для пошукового запиту отримую відповідні дані, які мають записи в кожній з таблиць:

A screenshot of a computer

Description automatically generated

**Код програми**

**model.py**

import psycopg2

from view import View

import random

import string

class Model:

    def \_\_init\_\_(self):

        self.conn = psycopg2.connect(

            dbname='science\_pub',

            user='postgres',

            password='1111',

            host='localhost',

            port='5432'

        )

        self.view = View()

        # Create tables if not exist

        self.create\_tables()

    def create\_tables(self):

        c = self.conn.cursor()

        # Check and create Author table

        c.execute("""

            CREATE TABLE IF NOT EXISTS "Author" (

                "Author\_ID" SERIAL PRIMARY KEY,

                "name" VARCHAR NOT NULL,

                "surname" VARCHAR NOT NULL

            );

        """)

        # Check and create Collection table

        c.execute("""

            CREATE TABLE IF NOT EXISTS "Collection" (

                "Collection\_ID" SERIAL PRIMARY KEY,

                "name" VARCHAR NOT NULL,

                "type" VARCHAR NOT NULL,

                "view" VARCHAR NOT NULL

            );

        """)

        # Check and create Edition table

        c.execute("""

            CREATE TABLE IF NOT EXISTS "Edition" (

                "Edition\_ID" SERIAL PRIMARY KEY,

                "name" VARCHAR NOT NULL,

                "branch" VARCHAR NOT NULL,

                "number\_of\_pages" INT NOT NULL,

                "languages" VARCHAR NOT NULL

            );

        """)

        # Check and create Author\_Collection\_Edition table

        c.execute("""

            CREATE TABLE IF NOT EXISTS "Author\_Collection\_Edition" (

                "Author\_ID" INT NOT NULL,

                "Edition\_ID" INT NOT NULL,

                PRIMARY KEY ("Author\_ID", "Edition\_ID"),

                FOREIGN KEY ("Author\_ID") REFERENCES "Author" ("Author\_ID") ON DELETE CASCADE,

                FOREIGN KEY ("Edition\_ID") REFERENCES "Edition" ("Edition\_ID") ON DELETE CASCADE

            );

        """)

        # Check and create Author\_Collection\_Edition\_ED table

        c.execute("""

            CREATE TABLE IF NOT EXISTS "Author\_Collection\_Edition\_ED" (

                "Edition\_ID" INT NOT NULL,

                "Collection\_ID" INT NOT NULL,

                "Date" DATE,

                PRIMARY KEY ("Edition\_ID", "Collection\_ID"),

                FOREIGN KEY ("Edition\_ID") REFERENCES "Edition" ("Edition\_ID") ON DELETE CASCADE,

                FOREIGN KEY ("Collection\_ID") REFERENCES "Collection" ("Collection\_ID") ON DELETE CASCADE

            );

        """)

        self.conn.commit()

    def get\_all(self, table\_name):

        c = self.conn.cursor()

        c.execute(f'SELECT \* FROM "{table\_name}"')

        return c.fetchall()

    def add\_data(self, table\_name, data, pk\_columns=None):

        try:

            c = self.conn.cursor()

            columns = ', '.join(data.keys())

            placeholders = ', '.join(['%s'] \* len(data))

            sql = f'INSERT INTO "{table\_name}" ({columns}) VALUES ({placeholders});'

            c.execute(sql, list(data.values()))

            self.conn.commit()

            self.view.show\_message("Added successfully!")

        except psycopg2.errors.ForeignKeyViolation as e:

            self.conn.rollback()

            self.view.show\_message(f"ERROR: Foreign key violation.\n{e}")

        except psycopg2.errors.UniqueViolation as e:

            self.conn.rollback()

            self.view.show\_message(f"ERROR: Duplicate key.\n{e}")

    def update\_data(self, table\_name, data, condition\_column, condition\_value):

        try:

            if not table\_name or not data or not condition\_column or condition\_value is None:

                self.view.show\_message("Insufficient information to update data.")

                return

            c = self.conn.cursor()

            set\_clause = ", ".join([f"\"{key}\" = %s" for key in data.keys()])

            sql = f'UPDATE "{table\_name}" SET {set\_clause} WHERE "{condition\_column}" = %s;'

            values = list(data.values())

            values.append(condition\_value)

            c.execute(sql, values)

            self.conn.commit()

            self.view.show\_message("Updated successfully!")

        except psycopg2.Error as e:

            self.conn.rollback()

            self.view.show\_message(str(e))

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

        try:

            c = self.conn.cursor()

            sql = f'DELETE FROM "{table\_name}" WHERE "{cond}" = %s;'

            c.execute(sql, (val,))

            self.conn.commit()

            self.view.show\_message("Deleted successfully!")

        except psycopg2.Error as e:

            self.conn.rollback()

            self.view.show\_message(str(e))

    def delete\_data\_composite(self, table\_name, conditions):

        # conditions: list of tuples (cond, val)

        try:

            c = self.conn.cursor()

            where\_clause = ' AND '.join([f'"{cond}" = %s' for cond, \_ in conditions])

            values = [val for \_, val in conditions]

            sql = f'DELETE FROM "{table\_name}" WHERE {where\_clause};'

            c.execute(sql, values)

            self.conn.commit()

            self.view.show\_message("Deleted successfully!")

        except psycopg2.Error as e:

            self.conn.rollback()

            self.view.show\_message(str(e))

    def generate\_data(self, table\_name, num):

        # Example random data generation

        c = self.conn.cursor()

        try:

            if table\_name == "Author":

                # Insert random authors

                for \_ in range(num):

                    name = ''.join(random.choice(string.ascii\_letters) for \_ in range(5))

                    surname = ''.join(random.choice(string.ascii\_letters) for \_ in range(7))

                    c.execute('INSERT INTO "Author" ("name","surname") VALUES (%s,%s)', (name, surname))

            elif table\_name == "Collection":

                for \_ in range(num):

                    col\_name = ''.join(random.choice(string.ascii\_letters) for \_ in range(8))

                    col\_type = random.choice(["Literature", "Fiction", "Poetry", "Science"])

                    col\_view = random.choice(["Paper", "Electronic"])

                    c.execute('INSERT INTO "Collection" ("name","type","view") VALUES (%s,%s,%s)', (col\_name, col\_type, col\_view))

            elif table\_name == "Edition":

                for \_ in range(num):

                    ed\_name = ''.join(random.choice(string.ascii\_letters) for \_ in range(6))

                    branch = random.choice(["Fiction", "Poetry", "Non-Fiction", "Science"])

                    pages = random.randint(50, 1000)

                    languages = random.choice(["English", "Ukrainian", "French"])

                    c.execute('INSERT INTO "Edition" ("name","branch","number\_of\_pages","languages") VALUES (%s,%s,%s,%s)', (ed\_name, branch, pages, languages))

            elif table\_name == "Author\_Collection\_Edition":

                # For linking table, we need existing authors and editions

                c.execute('SELECT "Author\_ID" FROM "Author"')

                authors = [row[0] for row in c.fetchall()]

                c.execute('SELECT "Edition\_ID" FROM "Edition"')

                editions = [row[0] for row in c.fetchall()]

                if not authors or not editions:

                    self.view.show\_message("No authors or editions to link. Generate them first.")

                    return

                for \_ in range(num):

                    aid = random.choice(authors)

                    eid = random.choice(editions)

                    try:

                        c.execute('INSERT INTO "Author\_Collection\_Edition" ("Author\_ID","Edition\_ID") VALUES (%s,%s)', (aid, eid))

                    except psycopg2.errors.UniqueViolation:

                        self.conn.rollback()  # ignore duplicates

                        continue

            elif table\_name == "Author\_Collection\_Edition\_ED":

                # We need existing editions and collections

                c.execute('SELECT "Edition\_ID" FROM "Edition"')

                editions = [row[0] for row in c.fetchall()]

                c.execute('SELECT "Collection\_ID" FROM "Collection"')

                collections = [row[0] for row in c.fetchall()]

                if not editions or not collections:

                    self.view.show\_message("No editions or collections to link. Generate them first.")

                    return

                for \_ in range(num):

                    eid = random.choice(editions)

                    cid = random.choice(collections)

                    # Random date

                    year = random.randint(1800, 2020)

                    month = random.randint(1,12)

                    day = random.randint(1,28)

                    date\_str = f"{year}-{month:02d}-{day:02d}"

                    try:

                        c.execute('INSERT INTO "Author\_Collection\_Edition\_ED" ("Edition\_ID","Collection\_ID","Date") VALUES (%s,%s,%s)', (eid, cid, date\_str))

                    except psycopg2.errors.UniqueViolation:

                        self.conn.rollback()

                        continue

            self.conn.commit()

            self.view.show\_message("Generated successfully!")

        except psycopg2.Error as e:

            self.conn.rollback()

            self.view.show\_message(str(e))

    def advanced\_search(self, surname\_pattern, min\_pages, max\_pages, start\_date, end\_date):

        # Example advanced query

        # Joins across multiple tables and applies filters on numeric range, string pattern, and date range

        c = self.conn.cursor()

        sql = """

            SELECT a."name", a."surname", ed."name" as edition\_name, ed."number\_of\_pages", aced."Date", c."name" as collection\_name

            FROM "Author" a

            JOIN "Author\_Collection\_Edition" ace ON a."Author\_ID" = ace."Author\_ID"

            JOIN "Edition" ed ON ace."Edition\_ID" = ed."Edition\_ID"

            JOIN "Author\_Collection\_Edition\_ED" aced ON ed."Edition\_ID" = aced."Edition\_ID"

            JOIN "Collection" c ON aced."Collection\_ID" = c."Collection\_ID"

            WHERE a."surname" ILIKE %s

              AND ed."number\_of\_pages" BETWEEN %s AND %s

              AND aced."Date" BETWEEN %s AND %s

            ORDER BY a."surname", ed."number\_of\_pages";

        """

        c.execute(sql, (f"%{surname\_pattern}%", min\_pages, max\_pages, start\_date, end\_date))

        rows = c.fetchall()

        return rows

**view.py**

from tabulate import tabulate

class View:

    def show\_author(self, data):

        print("Authors:")

        headers = ["Author\_ID", "Name", "Surname"]

        print(tabulate(data, headers, tablefmt="pretty"))

    def show\_collection(self, data):

        print("Collections:")

        headers = ["Collection\_ID", "Name", "Type", "View"]

        print(tabulate(data, headers, tablefmt="pretty"))

    def show\_edition(self, data):

        print("Editions:")

        headers = ["Edition\_ID", "Name", "Branch", "Number\_of\_pages", "Languages"]

        print(tabulate(data, headers, tablefmt="pretty"))

    def show\_author\_collection\_edition(self, data):

        print("Author\_Collection\_Edition:")

        headers = ["Author\_ID", "Edition\_ID"]

        print(tabulate(data, headers, tablefmt="pretty"))

    def show\_author\_collection\_edition\_ed(self, data):

        print("Author\_Collection\_Edition\_ED:")

        headers = ["Edition\_ID", "Collection\_ID", "Date"]

        print(tabulate(data, headers, tablefmt="pretty"))

    def show\_message(self, message):

        print(message)

    def show\_main\_menu(self):

        self.show\_message("\nWelcome! Which table do you want to work with?")

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

        self.show\_message("2. Collection")

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

        self.show\_message("4. Author\_Collection\_Edition")

        self.show\_message("5. Author\_Collection\_Edition\_ED")

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

        self.show\_message("7. Quit")

        return input("Enter your choice: ")

    def show\_table\_menu(self, table\_name):

        self.show\_message(f"\nWorking with {table\_name}")

        self.show\_message("1. Add")

        self.show\_message("2. View")

        if table\_name != "Author\_Collection\_Edition" and table\_name != "Author\_Collection\_Edition\_ED":

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

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

            self.show\_message("5. Generate data")

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

        else:

            # Linking tables might be add/view/delete/generate only

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

            self.show\_message("4. Generate data")

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

        return input("Enter your choice: ")

    def get\_data\_input(self, table\_name):

        if table\_name == "Author":

            name = input("Enter author's name: ")

            surname = input("Enter author's surname: ")

            return {"name": name, "surname": surname}

        elif table\_name == "Collection":

            cname = input("Enter collection's name: ")

            ctype = input("Enter collection's type: ")

            cview = input("Enter collection's view: ")

            return {"name": cname, "type": ctype, "view": cview}

        elif table\_name == "Edition":

            ename = input("Enter edition's name: ")

            branch = input("Enter edition's branch: ")

            pages = int(input("Enter number\_of\_pages: "))

            langs = input("Enter languages: ")

            return {"name": ename, "branch": branch, "number\_of\_pages": pages, "languages": langs}

        elif table\_name == "Author\_Collection\_Edition":

            aid = int(input("Enter Author\_ID: "))

            eid = int(input("Enter Edition\_ID: "))

            return {"Author\_ID": aid, "Edition\_ID": eid}

        elif table\_name == "Author\_Collection\_Edition\_ED":

            eid = int(input("Enter Edition\_ID: "))

            cid = int(input("Enter Collection\_ID: "))

            date = input("Enter Date (YYYY-MM-DD): ")

            return {"Edition\_ID": eid, "Collection\_ID": cid, "Date": date}

    def get\_update\_input(self, table\_name, pk\_val):

        if table\_name == "Author":

            name = input("Enter new author's name: ")

            surname = input("Enter new author's surname: ")

            return {"name": name, "surname": surname}

        elif table\_name == "Collection":

            cname = input("Enter new collection's name: ")

            ctype = input("Enter new collection's type: ")

            cview = input("Enter new collection's view: ")

            return {"name": cname, "type": ctype, "view": cview}

        elif table\_name == "Edition":

            ename = input("Enter new edition's name: ")

            branch = input("Enter new branch: ")

            pages = int(input("Enter new number\_of\_pages: "))

            langs = input("Enter new languages: ")

            return {"name": ename, "branch": branch, "number\_of\_pages": pages, "languages": langs}

    def get\_pk(self, table\_name):

        if table\_name == "Author":

            val = int(input("Enter Author\_ID: "))

            return "Author\_ID", val

        elif table\_name == "Collection":

            val = int(input("Enter Collection\_ID: "))

            return "Collection\_ID", val

        elif table\_name == "Edition":

            val = int(input("Enter Edition\_ID: "))

            return "Edition\_ID", val

    def get\_pk\_composite(self, table\_name):

        if table\_name == "Author\_Collection\_Edition":

            aid = int(input("Enter Author\_ID: "))

            eid = int(input("Enter Edition\_ID: "))

            return [("Author\_ID", aid), ("Edition\_ID", eid)]

        elif table\_name == "Author\_Collection\_Edition\_ED":

            eid = int(input("Enter Edition\_ID: "))

            cid = int(input("Enter Collection\_ID: "))

            return [("Edition\_ID", eid), ("Collection\_ID", cid)]

    def get\_num(self):

        return int(input("Enter number of data to generate: "))

    def advanced\_search\_input(self):

        surname\_pattern = input("Enter surname pattern (e.g. part of surname): ")

        min\_pages = int(input("Enter minimum number\_of\_pages: "))

        max\_pages = int(input("Enter maximum number\_of\_pages: "))

        start\_date = input("Enter start date (YYYY-MM-DD): ")

        end\_date = input("Enter end date (YYYY-MM-DD): ")

        return surname\_pattern, min\_pages, max\_pages, start\_date, end\_date

    def show\_advanced\_search\_results(self, data):

        print("Advanced Search Results:")

        headers = ["Author Name", "Author Surname", "Edition Name", "Pages", "Date", "Collection Name"]

        print(tabulate(data, headers, tablefmt="pretty"))

**controller.py**

from model import Model

from view import View

class Controller:

    def \_\_init\_\_(self):

        self.model = Model()

        self.view = View()

    def run(self):

        while True:

            choice = self.view.show\_main\_menu()

            if choice == '1':

                self.handle\_table("Author")

            elif choice == '2':

                self.handle\_table("Collection")

            elif choice == '3':

                self.handle\_table("Edition")

            elif choice == '4':

                self.handle\_table("Author\_Collection\_Edition")

            elif choice == '5':

                self.handle\_table("Author\_Collection\_Edition\_ED")

            elif choice == '6':

                self.advanced\_search()

            elif choice == '7':

                break

    def handle\_table(self, table\_name):

        while True:

            choice = self.view.show\_table\_menu(table\_name)

            if table\_name in ["Author", "Collection", "Edition"]:

                if choice == '1':

                    self.add(table\_name)

                elif choice == '2':

                    self.view\_table(table\_name)

                elif choice == '3':

                    self.update(table\_name)

                elif choice == '4':

                    self.delete(table\_name)

                elif choice == '5':

                    self.generate\_data(table\_name)

                elif choice == '6':

                    break

            else:

                # For linking tables (Author\_Collection\_Edition, Author\_Collection\_Edition\_ED)

                if choice == '1':

                    self.add(table\_name)

                elif choice == '2':

                    self.view\_table(table\_name)

                elif choice == '3':

                    self.delete(table\_name, composite=True)

                elif choice == '4':

                    self.generate\_data(table\_name)

                elif choice == '5':

                    break

    def add(self, table\_name):

        data = self.view.get\_data\_input(table\_name)

        self.model.add\_data(table\_name, data)

    def view\_table(self, table\_name):

        data = self.model.get\_all(table\_name)

        if table\_name == "Author":

            self.view.show\_author(data)

        elif table\_name == "Collection":

            self.view.show\_collection(data)

        elif table\_name == "Edition":

            self.view.show\_edition(data)

        elif table\_name == "Author\_Collection\_Edition":

            self.view.show\_author\_collection\_edition(data)

        elif table\_name == "Author\_Collection\_Edition\_ED":

            self.view.show\_author\_collection\_edition\_ed(data)

    def update(self, table\_name):

        pk, val = self.view.get\_pk(table\_name)

        new\_data = self.view.get\_update\_input(table\_name, val)

        self.model.update\_data(table\_name, new\_data, pk, val)

    def delete(self, table\_name, composite=False):

        if composite:

            conditions = self.view.get\_pk\_composite(table\_name)

            self.model.delete\_data\_composite(table\_name, conditions)

        else:

            pk, val = self.view.get\_pk(table\_name)

            self.model.delete\_data(table\_name, pk, val)

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

        num = self.view.get\_num()

        self.model.generate\_data(table\_name, num)

    def advanced\_search(self):

        surname\_pattern, min\_pages, max\_pages, start\_date, end\_date = self.view.advanced\_search\_input()

        data = self.model.advanced\_search(surname\_pattern, min\_pages, max\_pages, start\_date, end\_date)

        self.view.show\_advanced\_search\_results(data)

**main.py**

from controller import Controller

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

    controller = Controller()

    controller.run()