НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря Сікорського»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ
Кафедра системного програмування та спеціалізованих комп'ютерних систем

Розрахунково-графічна робота

з дисципліни «**Бази даних та засоби управління**»

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

Виконав студент групи : КВ-23
ПІБ : Буц Аліса Сергіївна
Перевірив:

Київ 2024

Варіант 47

“Система обліку виконавців та виступів на фестивалях”

Метою роботи є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

Загальне завдання роботи полягає у наступному:

1. Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

Деталізоване завдання:

1. Забезпечити можливість уведення/редагування/вилучення даних у таблицях бази даних з можливістю контролю відповідності типів даних атрибутів таблиць (рядків, чисел, дати/часу). Для контролю пропонується два варіанти: контроль при введенні (валідація даних) та перехоплення помилок (try..except) від сервера PostgreSQL при виконанні відповідної команди SQL. Особливу увагу варто звернути на дані таблиць, що мають зв’язок 1:N. При цьому з боку батьківської таблиці необхідно контролювати **вилучення** рядків за умови наявності даних у підлеглій таблиці. З точки зору підлеглої таблиці варто контролювати наявність відповідного рядка у батьківській таблиці при виконанні **внесення** нових даних. Унеможливити виведення програмою системних помилок на екрані шляхом їх перехоплення і адекватної обробки. Внесення даних виконується користувачем у консольному вікні програми.
2. Забезпечити можливість автоматичної генерації великої кількості даних у таблицях за допомогою вбудованих у PostgreSQL функцій роботи з псевдовипадковими числами. Дані мають бути згенерованими **не мовою програмування, а відповідним SQL-запитом**!

Вимоги до звіту у форматі PDF (у електронній формі)

Загальні вимоги

- титульний аркуш (включно з посиланням студента в Телеграм), завдання, URL репозиторію з вихідним кодом та відповіді на вимоги до звітування щодо пунктів 1-4 деталізованого завдання (див. нижче);
- діаграму сутність-зв’язок та структуру бази даних з лабораторної роботи №1, а також короткий опис бази даних;
- схему меню користувача з описом функціональності кожного пункту;
- назву мови програмування та бібліотек, що були використані;

Вимоги до пункту №1 деталізованого завдання:

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

Вимоги до пункту №2 деталізованого завдання:

- копії екрану (ілюстрації) з фрагментами згенерованих даних таблиць;
- копії SQL-запитів, що ілюструють генерацію при визначених вхідних параметрах.

Вимоги до пункту №3 деталізованого завдання:

- ілюстрації уведення пошукового запиту та результатів виконання запитів;
- копії SQL-запитів, що ілюструють пошук із зазначеними початковими параметрами.

Вимоги до пункту №4 деталізованого завдання:

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

Вимоги до звіту у форматі PDF (у електронній формі)

Загальні вимоги

- титульний аркуш (включно з посиланням студента в Телеграм), завдання, URL репозиторію з вихідним кодом та відповіді на вимоги до звітування щодо пунктів 1-4 деталізованого завдання (див. нижче);
- діаграму сутність-зв’язок та структуру бази даних з лабораторної роботи №1, а також короткий опис бази даних;
- схему меню користувача з описом функціональності кожного пункту;
- назву мови програмування та бібліотек, що були використані;

Вимоги до пункту №1 деталізованого завдання:

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

Вимоги до пункту №2 деталізованого завдання:

- копії екрану (ілюстрації) з фрагментами згенерованих даних таблиць;
- копії SQL-запитів, що ілюструють генерацію при визначених вхідних параметрах.

Вимоги до пункту №3 деталізованого завдання:

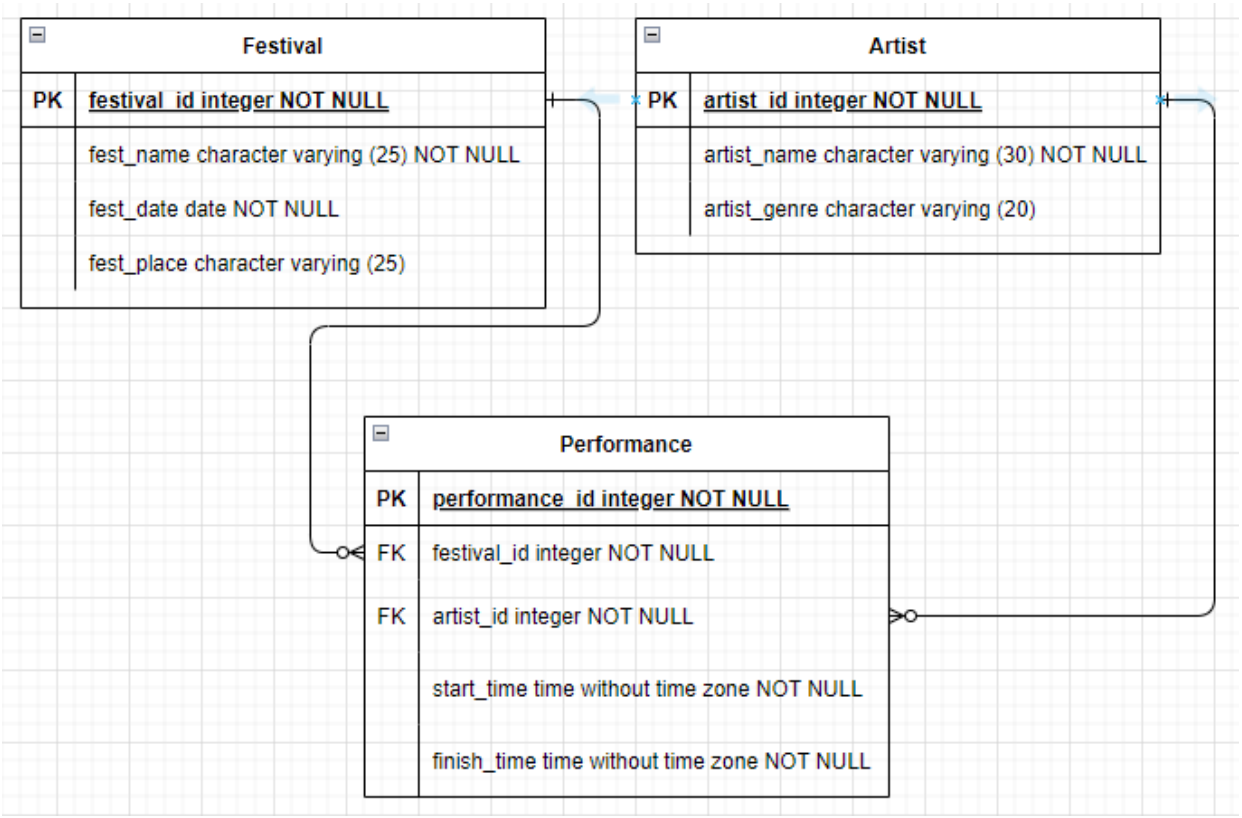
- ілюстрації уведення пошукового запиту та результатів виконання запитів;
- копії SQL-запитів, що ілюструють пошук із зазначеними початковими параметрами.

Вимоги до пункту №4 деталізованого завдання:

Модель “Система обліку виконавців та виступів на фестивалях” має такі сутності та зв’язки :

1. Сутність “festival” (Фестиваль) з атрибутами “festival_id” (primary key) “fest_name”, “fest_date”, “fest_place”.
2. Сутність “artist” (Виконавець) з атрибутами “artist_id” (primary key), “artist_name”, “artist_genre”
3. Сутність “performance” (Виступ) з атрибутами “performance_id” (primary key), “festival_id” (foreign key), “artist_id”(foreign key), “start_time”, “finish_time”

Логічна модель “Система обліку виконавців та виступів на фестивалях”



Середовище для налаштування, підключення та розробки бази даних

Мова програмування : Python 3.11 Модуль “psycopg2” був використаний для підключення до сервера бази даних

Схема меню користувача

Опис функціональності кожного пункту

1. Add New Artist

2. Add New Performance

3. Add New Festival

Перші 3 опції меню відповідають за додавання нового запису в 1 з таблиць на вибір.

4. Show Artists

5. Show Performances

6. Show Festivals

Наступні 3 опції меню відповідають за показ всіх даних з 1 таблиці на вибір.

7. Update Artist

8. Update Performance

9. Update Festival
Опції з "Update" дозволяють оновлювати дані для кожної з таблиць на вибір.

10. Remove Artist

11. Remove Performance

12. Remove Festival
"Remove" відповідає за видалення даних з таблиці на вибір по id.

13. Create Data By Random
Створення випадкових даних для всіх таблиць.

14. Delete All Data
Видалення всіх даних

15. View Analytics
Перегляд аналітики

16. Exit
Завершення роботи програми

Пункт 1 :

Уведення/редагування/вилучення даних у таблицях бази даних

Додавання нового виконавця та нового фестивалю:

```
Choose an action : 1
Input artist_id: 1
Input name: Rammstein
Input genre: Rock
Successfully Add An Artist.
```

```
Choose an action : 3
Input festival id: 1
Input festival name: FairyCore
Input festival date: 2025-01-01
Input festival place: Kyiv
Successfully Added A Festival
```

Перевірка присутності нових даних :

```
Choose an action : 4
Artists:
artist_id: 1, artist_name: Rammstein,  artist_genre: Rock
```

```
Choose an action : 6
Festivals:
ID: 1, Name: FairyCore, Date: 2025-01-01, Place: Kyiv
```

Редагування :

```
Choose an action : 7
Input artist_id: 1
Input name: NotRammstein
Input genre: notRock
Successfully Update An Artist
```

```
Choose an action : 9
Input festival id: 1
Input festival name: CoreFairy
Input festival date: 2025-01-02
Input festival place: Lviv
Successfully Updated A Festival
```

Перевірка успішності редагування :

```
Choose an action : 4
Artists:
artist_id: 1, artist_name: NotRammstein,  artist_genre: notRock
```

```
Choose an action : 6
Festivals:
ID: 1, Name: CoreFairy, Date: 2025-01-02, Place: Lviv
```

Видалення даних та перевірка їх відсутності :

```
Choose an action : 10
Input artist_id: 1
successfully Deleted An Artist
```

```
Choose an action : 4
Artists:
```

```
Choose an action : 12
Input festival id: 1
Successfully Deleted A Festival
```

```
Choose an action : 6
Festivals:
```

При введенні неіснуючих даних виводиться відповідна помилка

```
Choose an action : 8
Input Performance ID: 1
Performance With This ID Does Not Exist
```

Помилка при спробі видалити дані з батьківської таблиці

```
Choose an action : 12
Input festival id: 1
Festival 1 cannot be deleted as it is associated with performances.
Festival Not Deleted
```

Пункт 2 :

Результат створення випадково згенерованих даних для всіх таблиць

```
Choose an action : 13
Input Number Of Generations: 6
Successfully Created Artist Sequence
6 Artists Created successfully!
Successfully Generated Festival Sequence
6 Festivals Successfully Generated
Successfully Created Performance Sequence
6 Performances Successfully Created
```

```
Choose an action : 4
Artists:
artist_id: 1, artist_name: My dear Coffin,  artist_genre: heavy metal
artist_id: 2, artist_name: Hellfire Angel,  artist_genre: alternative rock
artist_id: 3, artist_name: Black Angel,  artist_genre: heavy metal
artist_id: 4, artist_name: Black Death,  artist_genre: industrial metal
artist_id: 5, artist_name: Hellfire Death,  artist_genre: heavy metal
artist_id: 6, artist_name: My dear Death,  artist_genre: alternative rock
```

```
Choose an action : 5
Performance:
Performance ID: 1, Start time: 04:30:54, Finish time: 20:26:15, Festival ID: 4, Artis ID: 1
Performance ID: 2, Start time: 01:09:01, Finish time: 12:12:20, Festival ID: 2, Artis ID: 1
Performance ID: 3, Start time: 12:47:40, Finish time: 09:38:15, Festival ID: 2, Artis ID: 4
Performance ID: 4, Start time: 12:36:09, Finish time: 13:12:27, Festival ID: 1, Artis ID: 3
Performance ID: 5, Start time: 07:08:35, Finish time: 05:49:19, Festival ID: 3, Artis ID: 2
Performance ID: 6, Start time: 15:22:42, Finish time: 06:44:29, Festival ID: 1, Artis ID: 6
```

```
Choose an action : 6
Festivals:
ID: 1, Name: Heart-Shaped Box, Date: 2025-03-11, Place: New York
ID: 2, Name: Bleach, Date: 2025-04-13, Place: Berlin
ID: 3, Name: Meteora, Date: 2025-09-15, Place: New York
ID: 4, Name: Nevermind, Date: 2025-07-17, Place: Berlin
ID: 5, Name: Bleach, Date: 2025-11-23, Place: London
ID: 6, Name: Cherry Waves, Date: 2025-11-30, Place: London
```

Відповідні SQL запити для випадкового заповнення таблиць

```
    c.execute("""
    INSERT INTO "artist" ("artist_id", "artist_name", "artist_genre")
    SELECT
        nextval('artist_id_seq'),
        -- Combine two random words for artist_name
        (array['Hellfire', 'Cursed', 'My dear', 'Black'])[floor(random() * 4) + 1] || ' ' ||
        (array['Death', 'Angel', 'String', 'Coffin'])[floor(random() * 4) + 1],
        (array['rock', 'heavy metal', 'industrial metal', 'alternative rock'])[floor(random() * 4) + 1]
    FROM generate_series(1, %s);
    """, (number_of_operations,))
```

```
c.execute("""
    INSERT INTO "performance" ("performance_id", "festival_id", "artist_id", "start_time", "finish_time")
    SELECT
        nextval('performance_id_seq'),
        -- Random festival_id
        floor(random() * (SELECT max("festival_id") FROM "festival") + 1),
        --  Random artist_id
        floor(random() * (SELECT max("artist_id") FROM "artist") + 1),
        TO_CHAR('00:00:00'::time + (random() * interval '23 hours'), 'HH24:MI:SS')::time as start_time,
        TO_CHAR('00:00:00'::time + (random() * interval '23 hours') + interval '2 hours', 'HH24:MI:SS')::time as finish_time
    FROM generate_series(1, %s);
""", (number_of_operations,))
```

```
    c.execute("""
    INSERT INTO "festival" ("festival_id", "fest_name", "fest_date", "fest_place")
    SELECT
        nextval('festival_festival_id_seq'),  -- Використовуємо послідовність для генерації нового id
        (array['Faint', 'Meteora', 'Bleach', 'Nevermind', 'Cherry Waves', 'Heart-Shaped Box'])[floor(random() * 6) + 1],
        CURRENT_DATE + (floor(random() * 365) * interval '1 day'),
        (array['London', 'Kyiv', 'Paris', 'Berlin', 'New York'])[floor(random() * 5) + 1]
    FROM generate_series(1, %s);
    """, (number_of_operations,))
```

Пункт 3 :

Запит №1: Найпопулярніші виконавці

Запит :

```
c.execute("""
        SELECT "artist_id", "artist_name", "num_performances"
        FROM (
            SELECT pr."artist_id", pr."artist_name",
            COUNT(*) AS num_performances,
            DENSE_RANK() OVER (ORDER BY COUNT(*) DESC) AS rnk
            FROM "performance" p
            JOIN "artist" pr ON p."artist_id" = pr."artist_id"
            GROUP BY pr."artist_id", pr."artist_name"
        ) ranked
        WHERE rnk = 1;
""")
```

Результат :

```
--------------------------------------------------------------------------------
Найпопулярніші виконавці:
ID: 1, ім'я: My dear Coffin, кількість виступів: 3
ID: 11, ім'я: Cursed Death, кількість виступів: 3
--------------------------------------------------------------------------------
```

Запит №2: Інформація про останні 10 доданих виступів :

Запит :

```
c.execute("""
            SELECT
                p."performance_id",
                p."festival_id",
                p."artist_id",
                pr."artist_name",
                pr."artist_genre",
                p."start_time",
                p."finish_time"
            FROM
                "performance" p
            JOIN
                "artist" pr ON p."artist_id" = pr."artist_id"
            ORDER BY
                p."performance_id" DESC -- Sort by the most recently added
            LIMIT 10;  -- Return only the 10 most recent performances
""")
```

Результат :

```
--------------------------------------------------------------------------------
Інформація про останні 10 доданих виступів
Виступ № 21, № фестивалю: 11, артист: My dear Death (rock), початок: 12:51:59, кінець: 00:44:05
Виступ № 20, № фестивалю: 3, артист: Black Angel (heavy metal), початок: 14:23:24, кінець: 18:25:47
Виступ № 19, № фестивалю: 5, артист: Cursed Death (heavy metal), початок: 11:54:32, кінець: 04:59:09
Виступ № 18, № фестивалю: 6, артист: Hellfire String (rock), початок: 03:27:52, кінець: 08:18:42
Виступ № 17, № фестивалю: 20, артист: My dear Coffin (heavy metal), початок: 20:34:39, кінець: 11:01:18
Виступ № 16, № фестивалю: 17, артист: Black String (alternative rock), початок: 21:57:45, кінець: 15:11:07
Виступ № 15, № фестивалю: 13, артист: Hellfire Angel (industrial metal), початок: 10:43:50, кінець: 18:39:46
Виступ № 14, № фестивалю: 3, артист: Hellfire Death (rock), початок: 21:51:04, кінець: 10:10:45
Виступ № 13, № фестивалю: 11, артист: Hellfire Coffin (alternative rock), початок: 12:03:32, кінець: 11:37:27
Виступ № 12, № фестивалю: 3, артист: My dear Coffin (heavy metal), початок: 17:41:26, кінець: 03:59:34
--------------------------------------------------------------------------------
```

Запит №3: Найпопулярніший жанр

Запит :

```
c.execute("""
        WITH GenreRank AS (
            SELECT
                pr."artist_genre" AS popular_genre,
                COUNT(*) AS num_performances,
                DENSE_RANK() OVER (ORDER BY COUNT(*) DESC) AS rnk
            FROM
                "performance" p
            JOIN
                "artist" pr ON p."artist_id" = pr."artist_id"
            GROUP BY
                pr."artist_genre"
        )
        SELECT
            popular_genre,
            num_performances
        FROM
            GenreRank
        WHERE
            rnk = 1;
""")
```

Результат :

```
--------------------------------------------------------------------------------
Найчастіше використані жанри:
Жанр: heavy metal, Кількість виступів: 10
--------------------------------------------------------------------------------
```

Код модулів “Model”

model.py

```
import psycopg2

class Model:  2 usages
    def __init__(self):
        self.conn = psycopg2.connect(
            dbname='postgres',
            user='postgres',
            password='admin',
            host='localhost',
            port=5432
        )
        self.create_tables()

    def create_tables(self):  1 usage
        c = self.conn.cursor()
        # Check for tables
        c.execute("SELECT EXISTS (SELECT 1 FROM information_schema.tables WHERE table_name = 'performance')")
        performance_table_exists = c.fetchone()[0]

        c.execute("SELECT EXISTS (SELECT 1 FROM information_schema.tables WHERE table_name = 'festival')")
        festival_table_exists = c.fetchone()[0]

        c.execute("SELECT EXISTS (SELECT 1 FROM information_schema.tables WHERE table_name = 'artist')")
        artist_table_exists = c.fetchone()[0]

        if not performance_table_exists:
            c.execute('''
                        CREATE TABLE performance (
                            performance_id SERIAL PRIMARY KEY,
                            festival_id INTEGER NOT NULL,
                            artist_id INTEGER NOT NULL,
                            start_time TIME,
                            finish_time TIME
                        )
                    ''')
    if not festival_table_exists:
        c.execute('''
                    CREATE TABLE "festival" (
                        "festival_id" SERIAL PRIMARY KEY,
                        "fest_name" TEXT NOT NULL,
                        "fest_date" DATE NOT NULL,
                        "fest_place" TEXT NOT NULL
                    )
                ''')
    if not artist_table_exists:
        c.execute('''
                    CREATE TABLE "artist" (
                        "artist_id" SERIAL PRIMARY KEY,
                        "artist_name" TEXT NOT NULL,
                        "artist_genre" TEXT
                    )
                ''')

    self.conn.commit()
```

Artist : model.py

```python
class ModelArtist:  2 usages
    def __init__(self, db_model):
        self.conn = db_model.conn

    def add_artist(self, artist_id, artist_name, artist_genre):  1 usage (1 dynamic)
        c = self.conn.cursor()
        try:
            c.execute('INSERT INTO "artist" ("artist_id" ,"artist_name", "artist_genre") VALUES (%s, %s, %s)', (artist_id, artist_name, artist_genre))
            self.conn.commit()
            return True  # Returns True if the update was successful
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Adding An Artist: {str(e)}")
            return False   # Returns False if insertion fails

    def get_all_artists(self):  1 usage (1 dynamic)
        c = self.conn.cursor()
        c.execute('SELECT * FROM "artist"')
        return c.fetchall()

    def update_artist(self, artist_id, artist_name, artist_genre):  1 usage (1 dynamic)
        c = self.conn.cursor()
        try:
            c.execute('UPDATE "artist" SET "artist_name" = %s, "artist_genre" = %s WHERE "artist_id" = %s', (artist_name, artist_genre, artist_id))
            self.conn.commit()
            return True  # Returns True if the update was successful
        except Exception as e:
            self.conn.rollback()
            print(f"Error With An Artist Updating: {str(e)}")
            return False   # Returns False if insertion fails

    def delete_artist(self, artist_id):  1 usage (1 dynamic)
        c = self.conn.cursor()
        try:
            c.execute('DELETE FROM "artist" WHERE "artist_id"=%s', (artist_id,))
            self.conn.commit()
            return True  # Returns True if the update was successful
        except Exception as e:
            self.conn.rollback()
            print(f"Error With An Artist Deleting: {str(e)}")
            return False  # Returns False if insertion fails

    def check_artist_existence(self, artist_id):  2 usages (2 dynamic)
        c = self.conn.cursor()
        c.execute('SELECT 1 FROM "artist" WHERE "artist_id" = %s', (artist_id,))
        return c.fetchone() is not None

    def create_artist_sequence(self):  1 usage (1 dynamic)
        # Check for the existence of a sequence and ensure it starts with the correct value
        c = self.conn.cursor()
        c.execute("""
            DO $$
            DECLARE
                max_id INT;
            BEGIN
                -- Find the maximum existing artist_id
                SELECT COALESCE(MAX(artist_id), 0) INTO max_id FROM artist;
```

```
            -- Check if the sequence exists
            IF NOT EXISTS (
                SELECT 1
                FROM pg_sequences
                WHERE schemaname = 'public' AND sequencename = 'artist_id_seq'
            ) THEN
                -- Create a new sequence starting after the max_id
                EXECUTE 'CREATE SEQUENCE artist_id_seq START WITH ' || (max_id + 1);
            ELSE
                -- Reset the existing sequence to start after the max_id
                EXECUTE 'ALTER SEQUENCE artist_id_seq RESTART WITH ' || (max_id + 1);
            END IF;
        END $$;
    """)
    self.conn.commit()
```

```
def generate_rand_artist_data(self, number_of_operations):  1 usage (1 dynamic)
    c = self.conn.cursor()
    try:
        # Insert data
        c.execute("""
        INSERT INTO "artist" ("artist_id", "artist_name", "artist_genre")
        SELECT
            nextval('artist_id_seq'),
            -- Combine two random words for artist_name
            (array['Hellfire', 'Cursed', 'My dear', 'Black'])[floor(random() * 4) + 1] || ' ' ||
            (array['Death', 'Angel', 'String', 'Coffin'])[floor(random() * 4) + 1],
            (array['rock', 'heavy metal', 'industrial metal', 'alternative rock'])[floor(random() * 4) + 1]
        FROM generate_series(1, %s);
        """, (number_of_operations,))
        self.conn.commit()
        return True  # Returns True if the insertion was successful
    except Exception as e:
        self.conn.rollback()
        print(f"Error With An Artist Adding: {str(e)}")
        return False  # Returns False if insertion fails

def truncate_artist_table(self):  1 usage (1 dynamic)
    c = self.conn.cursor()
    try:
        # Insert data
        c.execute("""DELETE FROM "artist" """)
        self.conn.commit()
        return True  # Returns True if the update was successful
    except Exception as e:
        self.conn.rollback()
        print(f"Error With An Artist`s Data Deleting: {str(e)}")
        return False   # Returns False if insertion fails
```

Festival : model.py

```python
class ModelFestival:  2 usages
    def __init__(self, db_model):
        self.conn = db_model.conn

    def add_festival(self, festival_id, fest_name, fest_date, fest_place):  1 usage (1 dynamic)
        c = self.conn.cursor()
        try:
            c.execute('INSERT INTO "festival" ("festival_id", "fest_name", "fest_date", "fest_place") VALUES (%s, %s, %s, %s)',
                      (festival_id, fest_name, fest_date, fest_place))
            self.conn.commit()
            return True  # Returns True if the operation is successful
        except Exception as e:
            self.conn.rollback()
            print(f"Помилка при додаванні фестивалю: {str(e)}")
            return False  # Returns False if the operation failed

    def get_all_festivals(self):  1 usage (1 dynamic)
        c = self.conn.cursor()
        c.execute('SELECT * FROM "festival"')
        return c.fetchall()

    def update_festival(self, festival_id, fest_name, fest_date, fest_place):  1 usage (1 dynamic)
        c = self.conn.cursor()
        try:
            c.execute('UPDATE "festival" SET "fest_name" = %s, "fest_date" = %s, "fest_place" = %s WHERE "festival_id"=%s',
                      (fest_name, fest_date, fest_place, festival_id))
            self.conn.commit()
            return True   # Returns True if the operation is successful
        except Exception as e:
            self.conn.rollback()
            print(f"Помилка при оновленні фестивалю: {str(e)}")
            return False  # Returns False if the operation failed

    def delete_festival(self, festival_id):  1 usage (1 dynamic)
        c = self.conn.cursor()
        try:
            c.execute("DELETE FROM festival WHERE festival_id = %s", (festival_id,))
            self.conn.commit()
            print(f"Festival {festival_id} successfully deleted.")
        except Exception as e:
            self.conn.rollback()
            if "violates foreign key constraint" in str(e):
                print(f"Festival {festival_id} cannot be deleted as it is associated with performances.")
            else:
                print(f"Error deleting festival: {str(e)}")

    def check_festival_existence(self, festival_id):  2 usages (2 dynamic)
        c = self.conn.cursor()
        c.execute('SELECT 1 FROM "festival" WHERE "festival_id" = %s', (festival_id,))
        return bool(c.fetchone())

    def create_festival_sequence(self):  1 usage (1 dynamic)
        # Creating or updating the sequence for festival_id
        c = self.conn.cursor()
        c.execute("""
            DO $$
            DECLARE
                max_id INT;
            BEGIN
                -- Знаходимо максимальний festival_id
                SELECT COALESCE(MAX(festival_id), 0) INTO max_id FROM festival;
```

```
            -- Перевіряємо чи існує послідовність
            IF NOT EXISTS (
                SELECT 1 FROM pg_sequences WHERE schemaname = 'public' AND sequencename = 'festival_festival_id_seq'
            ) THEN
                -- Створюємо нову послідовність для festival_id, починаючи з max_id + 1
                EXECUTE 'CREATE SEQUENCE festival_festival_id_seq START WITH ' || (max_id + 1);
            ELSE
                -- Оновлюємо існуючу послідовність, щоб вона починалась з max_id + 1
                EXECUTE 'ALTER SEQUENCE festival_festival_id_seq RESTART WITH ' || (max_id + 1);
            END IF;
        END $$;
    """)
    self.conn.commit()
```

```
def generate_rand_festival_data(self, number_of_operations):  1 usage (1 dynamic)
    c = self.conn.cursor()
    try:
        # Inserting data into the festival table
        c.execute("""
            INSERT INTO "festival" ("festival_id", "fest_name", "fest_date", "fest_place")
            SELECT
                nextval('festival_festival_id_seq'),  -- Використовуємо послідовність для генерації нового id
                (array['Faint', 'Meteora', 'Bleach', 'Nevermind', 'Cherry Waves', 'Heart-Shaped Box'])[floor(random() * 6) + 1],
                CURRENT_DATE + (floor(random() * 365) * interval '1 day'),
                (array['London', 'Kyiv', 'Paris', 'Berlin', 'New York'])[floor(random() * 5) + 1]
            FROM generate_series(1, %s);
            """, (number_of_operations,))
        self.conn.commit()
        return True  # If insertion is successful
    except Exception as e:
        self.conn.rollback()
        print(f"Помилка при додаванні даних фестивалю: {str(e)}")
        return False  # If insertion failed

def truncate_festival_table(self):  1 usage (1 dynamic)
    c = self.conn.cursor()
    try:
        c.execute("DELETE FROM festival")   # Clear the festival table
        self.conn.commit()
        return True
    except Exception as e:
        self.conn.rollback()
        print(f"Помилка при видаленні всіх даних фестивалю: {str(e)}")
        return False
```

Performance : model.py

```python
class ModelPerformance:  2 usages
    def __init__(self, db_model):
        self.conn = db_model.conn

    def add_performance(self, performance_id, festival_id, artist_id, start_time, finish_time):  1 usage (1 dynamic)
        c = self.conn.cursor()
        try:
            # Check if client_id and room_number match parent tables
            c.execute('SELECT 1 FROM "festival" WHERE "festival_id" = %s', (festival_id,))
            festival_exists = c.fetchone()

            c.execute('SELECT 1 FROM "artist" WHERE "artist_id" = %s', (artist_id,))
            artist_exists = c.fetchone()

            if not festival_exists or not artist_exists:
                # Return an exception notification and throw an error
                return False  # Or throw an exception to process it further
            else:
                # All checks have passed, insert into booking_ticket
                c.execute(
                    'INSERT INTO "performance" ("performance_id", '
                    '"start_time", "finish_time", "festival_id", "artist_id") VALUES (%s, %s, %s, %s, %s)',
                    (performance_id, start_time, finish_time, festival_id, artist_id,))
                self.conn.commit()
                return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Adding A Performance: {str(e)}")
            return False

    def get_all_performance(self):  1 usage (1 dynamic)
        c = self.conn.cursor()
        c.execute('SELECT * FROM "performance"')
        return c.fetchall()

    def update_performance(self, performance_id, festival_id, artist_id, start_time, finish_time):  1 usage (1 dynamic)
        c = self.conn.cursor()
        try:
            # Attempting to update a record
            c.execute('UPDATE "performance" SET "start_time" = %s, '
                      '"finish_time" = %s, "festival_id" = %s, "artist_id" = %s  WHERE  "performance_id" = %s',
                      (festival_id, artist_id, start_time, finish_time, performance_id))
            self.conn.commit()
            return True  # Returns True if the update was successful
        except Exception as e:
            # Handling an error if the update failed
            self.conn.rollback()
            print(f"Error With Updating A Performance: {str(e)}")
            return False   # Returns False if insertion fails

    def delete_performance(self, performance_id):  1 usage (1 dynamic)
        c = self.conn.cursor()
        try:
            # Attempting to update a record
            c.execute('DELETE FROM "performance" WHERE "performance_id" = %s', (performance_id,))
            self.conn.commit()
            return True  # Returns True if the update was successful
        except Exception as e:
            # Handling an error in case the deletion failed
```

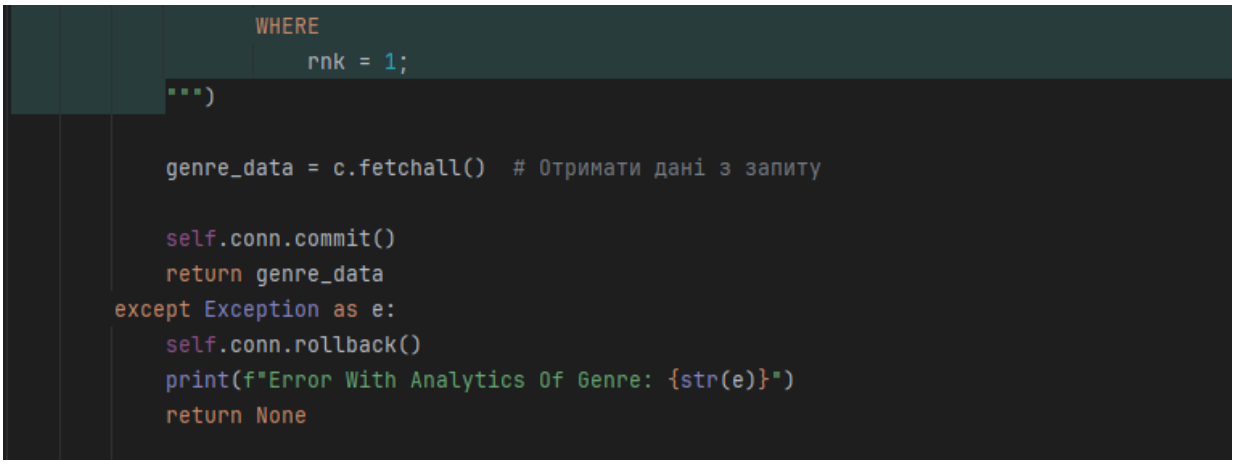
```
        # Handling an error in case the deletion failed
        self.conn.rollback()
        print(f"Error With Deleting A Performance: {str(e)}")
        return False   # Returns False if insertion fails

def check_performance_existence(self, performance_id):  2 usages (2 dynamic)
    c = self.conn.cursor()
    c.execute('SELECT 1 FROM "performance" WHERE "performance_id" = %s', (performance_id,))
    return bool(c.fetchone())

def create_performance_sequence(self):  1 usage (1 dynamic)
    c = self.conn.cursor()
    c.execute("""
        DO $$
        DECLARE
            max_id INT;
        BEGIN
            -- Find the maximum existing performance_id
           SELECT COALESCE(MAX(performance_id), 0) INTO max_id FROM performance;

           -- Check if the sequence exists
           IF NOT EXISTS (
               SELECT 1
               FROM pg_sequences
               WHERE schemaname = 'public' AND sequencename = 'performance_id_seq'
           ) THEN
               -- Create a new sequence starting after the max_id
               EXECUTE 'CREATE SEQUENCE performance_id_seq START WITH ' || (max_id + 1);
           ELSE
               -- Reset the existing sequence to start after the max_id
               EXECUTE 'ALTER SEQUENCE performance_id_seq RESTART WITH ' || (max_id + 1);
           END IF;
       END $$;
    """)
    self.conn.commit()

def generate_rand_performance_data(self, number_of_operations):  1 usage (1 dynamic)
    c = self.conn.cursor()
    try:
        c.execute("""
            INSERT INTO "performance" ("performance_id", "festival_id", "artist_id", "start_time", "finish_time")
            SELECT
                nextval('performance_id_seq'),
                -- Random festival_id
                floor(random() * (SELECT max("festival_id") FROM "festival") + 1),
                --  Random artist_id
                floor(random() * (SELECT max("artist_id") FROM "artist") + 1),
                TO_CHAR('00:00:00'::time + (random() * interval '23 hours'), 'HH24:MI:SS')::time as start_time,
                TO_CHAR('00:00:00'::time + (random() * interval '23 hours') + interval '2 hours', 'HH24:MI:SS')::time as finish_time
            FROM generate_series(1, %s);
        """, (number_of_operations,))
        self.conn.commit()
        return True
    except Exception as e:
```

```
            self.conn.rollback()
            print(f"Error With Performance Adding: {str(e)}")
            return False

    def truncate_performance_table(self):  1 usage (1 dynamic)
        c = self.conn.cursor()
        try:
            # Insert data
            c.execute("""DELETE FROM "performance" """)
            self.conn.commit()
            return True  # Returns True if the insertion was successful
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Deleting A Performance Data: {str(e)}")
            return False  # Returns False if insertion fails
```

Analytics : model.py

```
class ModelAnalytics:  2 usages
    def __init__(self, db_model):
        self.conn = db_model.conn

    def popular_artist(self):  1 usage (1 dynamic)
        c = self.conn.cursor()
        try:
            c.execute("""
                    SELECT "artist_id", "artist_name", "num_performances"
                    FROM (
                        SELECT pr."artist_id", pr."artist_name",
                        COUNT(*) AS num_performances,
                        DENSE_RANK() OVER (ORDER BY COUNT(*) DESC) AS rnk
                        FROM "performance" p
                        JOIN "artist" pr ON p."artist_id" = pr."artist_id"
                        GROUP BY pr."artist_id", pr."artist_name"
                    ) ranked
                    WHERE rnk = 1;
                """)

            popular_artist_data = c.fetchall() # Fetch the data from the query

            self.conn.commit()
            return popular_artist_data
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Analytics Of Popular Artist: {str(e)}")
            return None
```

```python
def number_of_performance(self):  1 usage (1 dynamic)
    c = self.conn.cursor()
    try:
        c.execute("""
                    SELECT
                        p."performance_id",
                        p."festival_id",
                        p."artist_id",
                        pr."artist_name",
                        pr."artist_genre",
                        p."start_time",
                        p."finish_time"
                    FROM
                        "performance" p
                    JOIN
                        "artist" pr ON p."artist_id" = pr."artist_id"
                    ORDER BY
                        p."performance_id" DESC -- Sort by the most recently added
                    LIMIT 10;  -- Return only the 10 most recent performances
        """)

        number_of_performance_data = c.fetchall()  # Отримати дані з запиту

        self.conn.commit()
        return number_of_performance_data
    except Exception as e:
        self.conn.rollback()
        print(f"Error With Analytics Of Number Of Performance: {str(e)}")
        return None

def genre_analytics(self):  1 usage (1 dynamic)
    c = self.conn.cursor()
    try:
        c.execute("""
                WITH GenreRank AS (
                    SELECT
                        pr."artist_genre" AS popular_genre,
                        COUNT(*) AS num_performances,
                        DENSE_RANK() OVER (ORDER BY COUNT(*) DESC) AS rnk
                    FROM
                        "performance" p
                    JOIN
                        "artist" pr ON p."artist_id" = pr."artist_id"
                    GROUP BY
                        pr."artist_genre"
                )
                SELECT
                    popular_genre,
                    num_performances
                FROM
                    GenreRank
```



```
            WHERE
                rnk = 1;
        """)

        genre_data = c.fetchall()  # Отримати дані з запиту

        self.conn.commit()
        return genre_data
    except Exception as e:
        self.conn.rollback()
        print(f"Error With Analytics Of Genre: {str(e)}")
        return None
```

Опис роботи модулів :

Performance – Робота з даними із таблиці performance

Artist– Робота з даними із таблиці artist

Festival – Робота з даними із таблиці festival

Analytics – Виклик запитів для пункту №3

Посилання на репозиторій : https://github.com/ButsAlice/BD.git

Телеграм : @feyrebel