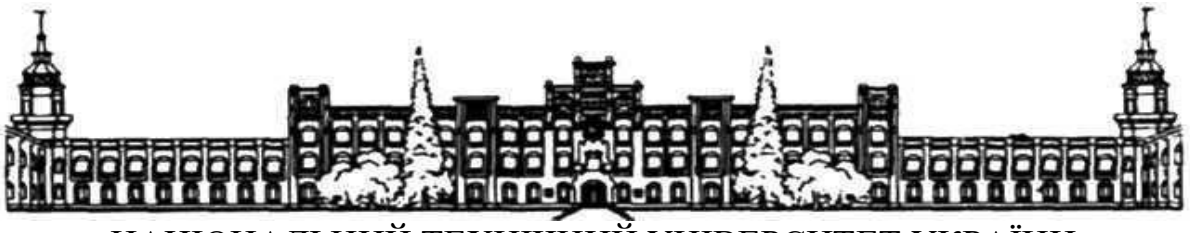


НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ
Кафедра системного програмування та спеціалізованих комп'ютерних систем

РГР

з дисципліни
«Бази даних і засоби управління»
Тема: «Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL»

Виконав: студент III курсу
ФПМ групи КВ-13
Петраш Антон Степанович

Київ – 2023

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

Концептуальна модель предметної області "Система обліку виконавців та виступів на фестивалях"

В концептуальній моделі "Система обліку виконавців та виступів на фестивалях" є наступні сутності та зв'язки:

1. Сутність "Performer" з атрибутами "Artist_ID", "name", "surname", "genre"

2. Сутність “Performance” з атрибутами “Performance_ID”, “Festival_ID”, “Artist_ID”, “Start_time”, “Finish_time”
3. Сутність “Festival” з атрибутами “Festival_ID”, “Fest_name”, “Price”, “City”

Зв’язок між виконавцем і виступом - один до багатьох, тому що один виконавець може проводити багато виступів, але один виступ проводить один виконавець. Зв’язок між Фестивалем і виступом - один до багатьох, адже на одному фестивалі може бути багато виступів і кожен виступ належить одному фестивалю.

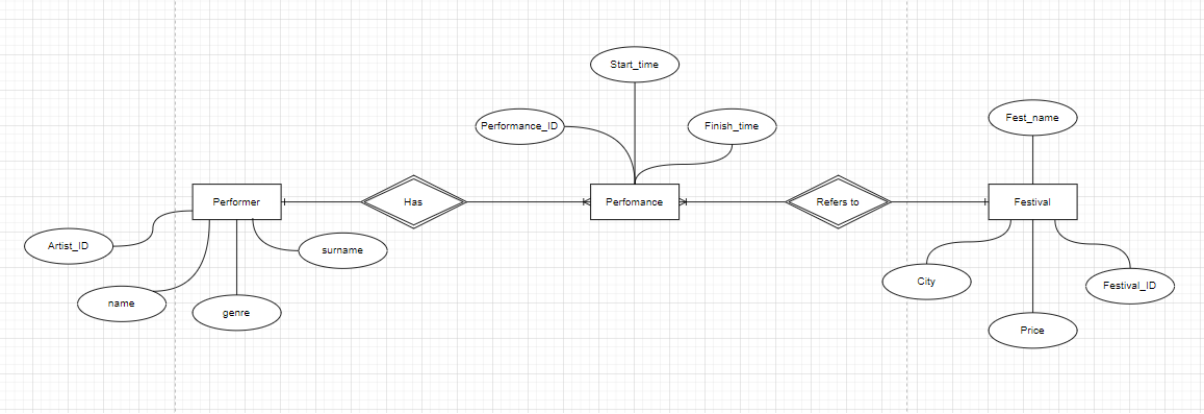


Рис.1 концептуальна модель “Система обліку виконавців та виступів на фестивалях”

Логічна модель “Система обліку виконавців та виступів на фестивалях”

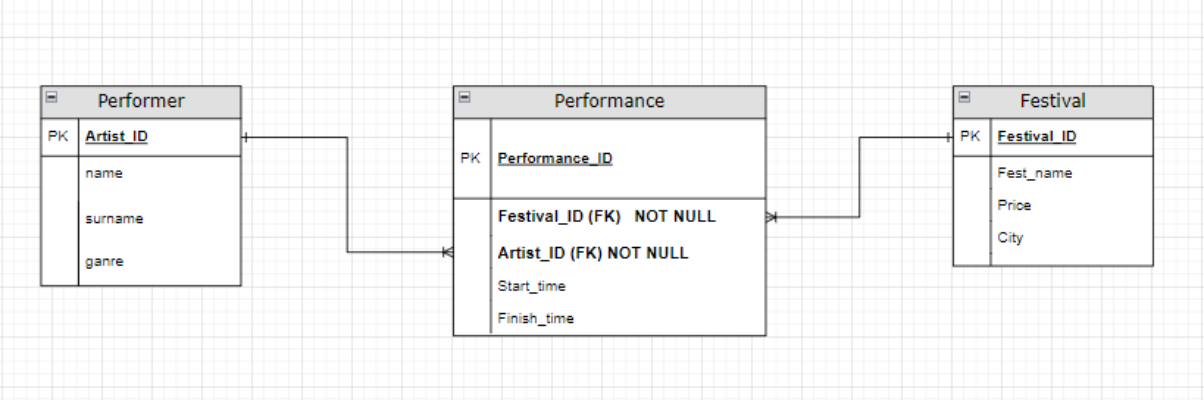
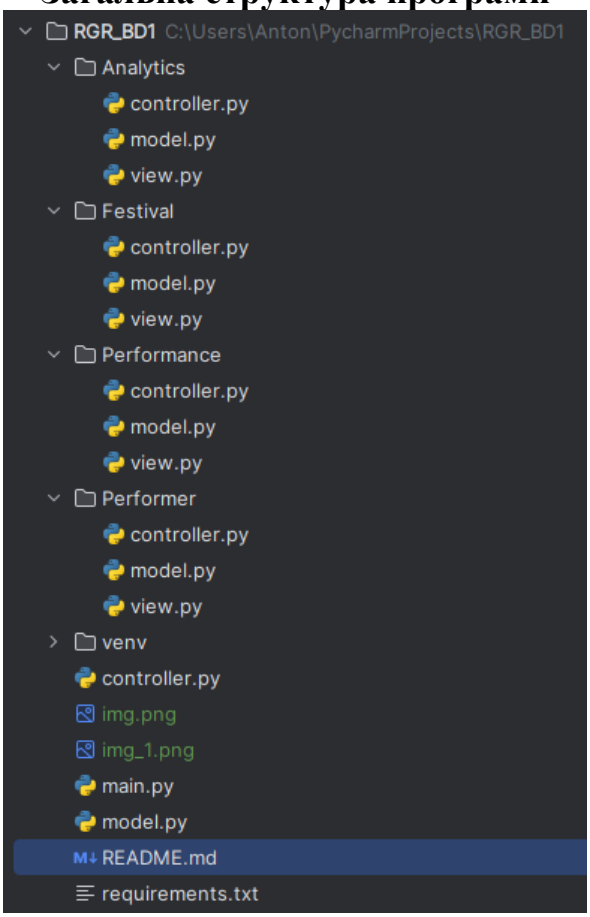


Рис.1 логічна модель “Система обліку виконавців та виступів на фестивалях”

Середовище для налаштування, підключення та розробки бази даних
Мова програмування : Python 3.11
Модуль “psycopg2” був використаний для підключення до сервера бази даних

Загальна структура програми



Структура складається з чотирьох модулів, кожен з яких відповідає за свою частину програми.

Меню програми

```
Main Menu:
1. Add New Performance
2. Show Performances
3. Renewal Performance
4. Remove Performance
5. Add New Festival
6. Show Festivals
7. Renewal Festival
8. Remove Festival
9. Add New Performer
10. Show Performers
11. Renewal Performer
12. Remove Performer
13. Create Data By Random
14. Delete All Data
15. View Analytics
16. Exit
Choose an action :
```

Реалізація додавання, зміни та видалення даних

Створення нового рядка

```
Choose an action : 5
Input Festival id: 102
Input Festival name: All Star
Input Festival price: 240
Input Festival city: Denmark
Successfully Added A Festival
```

```
Choose an action : 6
Festivals:
ID: 102, Name: All Star, Price: 240, City: Denmark
```

Видалення рядка

```
Choose an action : 8
Input Festival id: 102
Successfully Deleted A Festival
```

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

Редагування рядка

```
Choose an action : 7
Input Festival id: 100
Input Festival name: Dust 2
Input Festival price: 120
Input Festival city: Lisabon
Successfully Updated A Festival
```

```
Festivals:
ID: 100, Name: Dust 2, Price: 120, City: Lisabon
```

При введенні неіснуючого ID видається відповідна помилка

```
Choose an action : 8
Input Festival id: 123
It Does`nt Exist A Festival With This ID
```

Очищення всіх таблиць

```
Choose an action : 14
Confirm The Action. Type Yes or No: Yes
All Performances Data Successfully Deleted
Successfully Deleted All Festival`s Data
All Performers Data Deleted
```

Результат Створення Рандомізованих Даних

```
Performance ID: 4, Festival ID: 8, Artis ID: 3, Start time: 2023-02-08, Finish time: 2023-08-04
Performance ID: 5, Festival ID: 2, Artis ID: 5, Start time: 2023-03-19, Finish time: 2023-10-29
Performance ID: 7, Festival ID: 11, Artis ID: 16, Start time: 2023-07-19, Finish time: 2023-09-11
Performance ID: 8, Festival ID: 4, Artis ID: 8, Start time: 2023-04-16, Finish time: 2023-05-23
Performance ID: 9, Festival ID: 5, Artis ID: 2, Start time: 2023-08-23, Finish time: 2023-12-18
Performance ID: 10, Festival ID: 6, Artis ID: 12, Start time: 2023-02-01, Finish time: 2023-10-12
Performance ID: 11, Festival ID: 2, Artis ID: 5, Start time: 2023-05-02, Finish time: 2023-07-18
Performance ID: 13, Festival ID: 1, Artis ID: 8, Start time: 2023-01-05, Finish time: 2023-06-16
Performance ID: 14, Festival ID: 11, Artis ID: 9, Start time: 2023-04-17, Finish time: 2023-06-14
Performance ID: 16, Festival ID: 14, Artis ID: 7, Start time: 2023-06-12, Finish time: 2023-09-18
```

```
ID: 1, Name: Instrumental, Price: 431, City: Lviv
ID: 2, Name: Loud, Price: 447, City: Lviv
ID: 3, Name: Box, Price: 874, City: Lviv
ID: 4, Name: Fire, Price: 528, City: London
ID: 5, Name: Box, Price: 738, City: Berlin
ID: 6, Name: Dance, Price: 189, City: Berlin
ID: 7, Name: Box, Price: 486, City: Paris
ID: 8, Name: Music, Price: 483, City: London
ID: 9, Name: Loud, Price: 247, City: Stockholm
ID: 10, Name: Box, Price: 701, City: Paris
ID: 11, Name: Fire, Price: 613, City: London
ID: 12, Name: Music, Price: 988, City: Paris
ID: 13, Name: Music, Price: 344, City: Paris
ID: 14, Name: Music, Price: 242, City: Berlin
ID: 15, Name: Instrumental, Price: 650, City: Berlin
ID: 16, Name: Music, Price: 953, City: Stockholm
```

```
Artist_ID: 1, Artist name: John, Artist surname: Vachovski, genre: Pop
Artist_ID: 2, Artist name: Ivan, Artist surname: Potter, genre: All
Artist_ID: 3, Artist name: Ivan, Artist surname: Potter, genre: All
Artist_ID: 4, Artist name: John, Artist surname: Muller, genre: Pop
Artist_ID: 5, Artist name: Ivan, Artist surname: Lenon, genre: Pop
Artist_ID: 6, Artist name: Harry, Artist surname: Potter, genre: Jazz
Artist_ID: 7, Artist name: John, Artist surname: Muller, genre: Pop
Artist_ID: 8, Artist name: Harry, Artist surname: Lenon, genre: Jazz
Artist_ID: 9, Artist name: Ivan, Artist surname: Lenon, genre: Rock
Artist_ID: 10, Artist name: Ivan, Artist surname: Muller, genre: Jazz
Artist_ID: 11, Artist name: Mike, Artist surname: Muller, genre: Pop
Artist_ID: 12, Artist name: Mike, Artist surname: Potter, genre: All
Artist_ID: 13, Artist name: Ivan, Artist surname: Lenon, genre: Pop
Artist_ID: 14, Artist name: Harry, Artist surname: Muller, genre: All
Artist_ID: 15, Artist name: John, Artist surname: Lenon, genre: Jazz
Artist_ID: 16, Artist name: Ivan, Artist surname: Lenon, genre: Pop
```

Відповідні запити які використовуються для заповнення таблиць рандомними даними:

```
            c.execute("""
            INSERT INTO "Festival" ("Festival_ID", "Fest_name", "Price", "City")
            SELECT
                nextval('festival_id_seq'),
                (array['Fire', 'Dance', 'Loud', 'Music', 'Instrumental', 'Box'])[floor(random() * 6) + 1],
                random() * 1000 ,
                (array['London', 'Lviv', 'Paris', 'Berlin', 'Stockholm'])[floor(random() * 5) + 1]
            FROM generate_series(1, %s);
            """, (number_of_operations,))
```

```
c.execute("""
INSERT INTO "Perfomance" ("Preformance_ID", "Festival_ID", "Artist_ID", "Start_time", "Finish_time")
select * from (
SELECT
       nextval('perfomance_id_seq'),
       floor(random() * (SELECT max("Festival_ID") FROM "Festival") + 1),
       floor(random() * (SELECT max("Artist_ID") FROM "Performer") + 1),
       ('2023-01-01'::date + floor(random() * 3) * interval '1 day' + floor(random() * 12) * interval '1 month' + floor(random() * 31) * interval '1 day') as begin1,
       ('2023-01-01'::date + floor(random() * 3) * interval '1 day' + floor(random() * 12) * interval '1 month' + floor(random() * 31) * interval '1 day') as end1
   FROM generate_series(1, %s)) as t
WHERE begin1 < end1
    """, (number_of_operations,))
```

```
c.execute("""
INSERT INTO "Performer" ("Artist_ID" ,"name", "surname", "genre")
SELECT
    nextval('artist_id_seq'),
    (array['Ivan', 'Mike', 'John', 'Harry'])[floor(random() * 4) + 1],
    (array['Lenon', 'Muller', 'Vachovski', 'Potter'])[floor(random() * 4) + 1],
    (array['Rock', 'Jazz', 'All', 'Pop'])[floor(random() * 4) + 1]
FROM generate_series(1, %s);
""", (number_of_operations,))
self.conn.commit()
```

Виконання пункту 3

Запит №1: Найпопулярніші виконавці:

Запит:

```
c.execute("""
        SELECT "Artist_ID", "name", "surname", "num_performances"
        FROM (
            SELECT pr."Artist_ID", pr."name", pr."surname",
            COUNT(*) AS num_performances,
        DENSE_RANK() OVER (ORDER BY COUNT(*) DESC) AS rnk
        FROM "Perfomance" p
        JOIN "Performer" pr ON p."Artist_ID" = pr."Artist_ID"
        GROUP BY pr."Artist_ID", pr."name", pr."surname"
        ) ranked
        WHERE rnk = 1;

        """)
```

Результат:

```
--------------------------------------------------------------------------------
Найпопулярніші виконавці:
ID: 220, ім'я та прізвище John Muller, кількість виступів: 3
ID: 479, ім'я та прізвище Ivan Lenon, кількість виступів: 3
ID: 126, ім'я та прізвище John Lenon, кількість виступів: 3
ID: 37, ім'я та прізвище Harry Muller, кількість виступів: 3
ID: 276, ім'я та прізвище Ivan Vachovski, кількість виступів: 3
ID: 320, ім'я та прізвище Ivan Potter, кількість виступів: 3
ID: 265, ім'я та прізвище Harry Lenon, кількість виступів: 3
--------------------------------------------------------------------------------
```

Запит №2: Кількість виступів які відбулися за останній місяць:

Запит:

```
SELECT
    p."Preformance_ID",
    p."Festival_ID",
    p."Artist_ID",
    pr."name" AS artist_name,
    pr."surname" AS artist_surname,
    p."Start_time",
    p."Finish_time"
FROM
    "Perfomance" p
JOIN
    "Performer" pr ON p."Artist_ID" = pr."Artist_ID"
WHERE
    p."Start_time" >= CURRENT_DATE - INTERVAL '1 month';
```

Результат:

```
--------------------------------------------------------------------------
Кількість виступів за останній місяць:
Виступ № 25, № фестивалю: 7, артист:Harry Vachovski ,початок та кінець: 2023-11-28 -> 2023-12-18
Виступ № 35, № фестивалю: 233, артист:Harry Lenon ,початок та кінець: 2023-11-23 -> 2023-12-25
Виступ № 306, № фестивалю: 95, артист:Ivan Muller ,початок та кінець: 2023-11-29 -> 2023-12-04
Виступ № 399, № фестивалю: 422, артист:Mike Lenon ,початок та кінець: 2023-12-17 -> 2024-01-01
Виступ № 429, № фестивалю: 356, артист:Harry Vachovski ,початок та кінець: 2023-11-23 -> 2023-11-27
Виступ № 437, № фестивалю: 245, артист:Ivan Vachovski ,початок та кінець: 2023-12-20 -> 2023-12-27
Виступ № 482, № фестивалю: 295, артист:Ivan Muller ,початок та кінець: 2023-12-11 -> 2023-12-17
--------------------------------------------------------------------------
```

Запит №3: Жанр який користується найбільшим попитом у виконавців:

Запит:

```sql
WITH GenreRank AS (
    SELECT
        pr."genre" AS popular_genre,
        COUNT(*) AS num_performances,
        DENSE_RANK() OVER (ORDER BY COUNT(*) DESC) AS rnk
    FROM
        "Perfomance" p
    JOIN
        "Performer" pr ON p."Artist_ID" = pr."Artist_ID"
    GROUP BY
        pr."genre"
)
SELECT
    popular_genre,
    num_performances
FROM
    GenreRank
WHERE
    rnk = 1;
```

Результат:

```
--------------------------------------------------------------------------------
Найчастіше використані жанри :
Жанр: Jazz, Кількість: 79
--------------------------------------------------------------------------------
```

Код модулів програми:
model.py

```
import psycopg2

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

    1 usage  Anton Petrash
    def create_tables(self):
        c = self.conn.cursor()
        # Check for tables
        c.execute("SELECT EXISTS (SELECT 1 FROM information_schema.tables WHERE t
        Performance_table_exists = c.fetchone()[0]

        c.execute("SELECT EXISTS (SELECT 1 FROM information_schema.tables WHERE t
        Festival_table_exists = c.fetchone()[0]

        c.execute("SELECT EXISTS (SELECT 1 FROM information_schema.tables WHERE t
        Performer_table_exists = c.fetchone()[0]

        if not Performance_table_exists:
            c.execute('''
                        CREATE TABLE Perfomance (
                            Preformance_ID SERIAL PRIMARY KEY,
                            Festival_ID INTEGER NOT NULL,
                            Artist_ID INTEGER NOT NULL,
                            Start_time DATE NOT NULL,
                            Finish_time DATE NOT NULL
                        )
                    ''')
        if not Festival_table_exists:
                    c.execute('''
                                CREATE TABLE "Festival" (
                                    "Festival_ID" SERIAL PRIMARY KEY,
                                    "Fest_name" TEXT NOT NULL,
                                    "Price" INTEGER NOT NULL,
                                    "City" TEXT NOT NULL
                                )
                            ''')
                if not Performer_table_exists:
                    c.execute('''
                                CREATE TABLE "Performer" (
                                    "Artist_ID" SERIAL PRIMARY KEY,
                                    "name" TEXT NOT NULL
                                    "surname" TEXT NOT NULL
                                    "genre" TEXT
                                )
                            ''')

                self.conn.commit()
```

Performer\model.py

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

    1 usage (1 dynamic)  Anton Petrash
    def add_Performer(self, Artist_ID, name, surname, genre):
        c = self.conn.cursor()
        try:
            c.execute('INSERT INTO "Performer" ("Artist_ID" ,"name", "surname", "genre") VALUES (%s, %s, %s, %s)', (Artist_ID, name, surname, 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 A Performer: {str(e)}")
            return False   # Returns False if insertion fails

    1 usage (1 dynamic)  Anton Petrash
    def get_all_Performers(self):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Performer"')
        return c.fetchall()

    1 usage (1 dynamic)  Anton Petrash
    def update_Performer(self, Artist_ID, name, surname, genre):
        c = self.conn.cursor()
        try:
            c.execute('UPDATE "Performer" SET "name"=%s, "surname"=%s, "genre"=%s WHERE "Artist_ID"=%s', (name, surname, 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 A Performer Updating: {str(e)}")
            return False   # Returns False if insertion fails

    1 usage (1 dynamic)  Anton Petrash
    def delete_Performer(self, Artist_ID):
        c = self.conn.cursor()
        try:
            c.execute('DELETE FROM "Performer" 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

    2 usages (2 dynamic)  Anton Petrash
    def check_Performer_existence(self, Artist_ID):
        c = self.conn.cursor()
        c.execute('SELECT 1 FROM "Performer" WHERE "Artist_ID" = %s', (Artist_ID,))
        return c.fetchone() is not None

    1 usage (1 dynamic)  Anton Petrash
    def create_Performer_sequence(self):
        # Check for the existence of a sequence
        c = self.conn.cursor()
        c.execute("""
            DO $$
            BEGIN
                IF NOT EXISTS (SELECT 1 FROM pg_sequences WHERE schemaname = 'public' AND sequencename = 'artist_id_seq') THEN
                    CREATE SEQUENCE artist_id_seq;
                ELSE
                    DROP SEQUENCE artist_id_seq;
                    CREATE SEQUENCE artist_id_seq;
                END IF;
            END $$;
        """)
        self.conn.commit()
    1 usage (1 dynamic)  Anton Petrash
    def generate_rand_Performer_data(self, number_of_operations):
        c = self.conn.cursor()
        try:
            # Insert data
            c.execute("""
            INSERT INTO "Performer" ("Artist_ID" ,"name", "surname", "genre")
            SELECT
                nextval('artist_id_seq'),
```

```
            (array['Ivan', 'Mike', 'John', 'Harry'])[floor(random() * 4) + 1],
            (array['Lenon', 'Muller', 'Vachovski', 'Potter'])[floor(random() * 4) + 1],
            (array['Rock', 'Jazz', 'All', 'Pop'])[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 A Performer Adding: {str(e)}")
        return False   # Returns False if insertion fails

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

Festival/model.py

```
class ModelFestival:
    Anton Petrash
    def __init__(self, db_model):
        self.conn = db_model.conn

    1 usage (1 dynamic)  Anton Petrash
    def add_Festival(self, Festival_ID, Fest_name, Price, City):
        c = self.conn.cursor()
        try:
            c.execute('INSERT INTO "Festival" ("Festival_ID", "Fest_name", "Price", "City") VALUES (%s, %s, %s, %s)',
                      (Festival_ID, Fest_name, Price, City))
            self.conn.commit()
            return True  # Returns True if the update was successful
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Adding A Festival: {str(e)}")
            return False  # Returns False if insertion fails

    1 usage (1 dynamic)  Anton Petrash
    def get_all_Festivals(self):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Festival"')
        return c.fetchall()

    1 usage (1 dynamic)  Anton Petrash
    def update_Festival(self, Festival_ID, Fest_name, Price, City):
        c = self.conn.cursor()
        try:
            c.execute('UPDATE "Festival" SET "Fest_name"=%s, "Price"=%s, "City"=%s WHERE "Festival_ID"=%s',
                      (Fest_name, Price, City, Festival_ID))
            self.conn.commit()
            return True  # Returns True if the update was successful
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Updating A Festival: {str(e)}")
            return False   # Returns False if insertion fails

    1 usage (1 dynamic)  Anton Petrash
    def delete_Festival(self, Festival_ID):
```

```
    c = self.conn.cursor()
    try:
        c.execute('DELETE FROM "Festival" WHERE "Festival_ID"=%s', (Festival_ID,))
        self.conn.commit()
        return True  # Returns True if the update was successful
    except Exception as e:
        self.conn.rollback()
        print(f"Error With Deleting A Festival Violates A Foreign Key Constraint: {str(e)}")
        return False   # Returns False if insertion fails

2 usages (2 dynamic)  Anton Petrash
def check_Festival_existence(self, Festival_ID):
    c = self.conn.cursor()
    c.execute('SELECT 1 FROM "Festival" WHERE "Festival_ID" = %s', (Festival_ID,))
    return bool(c.fetchone())

1 usage (1 dynamic)  Anton Petrash
def create_Festival_sequence(self):
    # Check for the existence of a sequence
    c = self.conn.cursor()
    c.execute("""
    DO $$
       BEGIN
           IF NOT EXISTS (SELECT 1 FROM pg_sequences WHERE schemaname = 'public' AND sequencename = 'festival_id_seq') THEN
               CREATE SEQUENCE festival_id_seq;
           ELSE
               DROP SEQUENCE festival_id_seq;
               CREATE SEQUENCE festival_id_seq;
           END IF;
       END $$;
    """)
    self.conn.commit()

1 usage (1 dynamic)  Anton Petrash
def generate_rand_Festival_data(self, number_of_operations):
    c = self.conn.cursor()
    try:
        # Insert data
        c.execute("""
            INSERT INTO "Festival" ("Festival_ID", "Fest_name", "Price", "City")
            SELECT
                nextval('festival_id_seq'),
                (array['Fire', 'Dance', 'Loud', 'Music', 'Instrumental', 'Box'])[floor(random() * 6) + 1],
                random() * 1000 ,
                (array['London', 'Lviv', 'Paris', 'Berlin', 'Stockholm'])[floor(random() * 5) + 1]
            FROM generate_series(1, %s);
            """, (number_of_operations,))
            self.conn.commit()
            return True  # Returns True if the update was successful
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Adding The Festivals: {str(e)}")
            return False   # Returns False if insertion fails


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

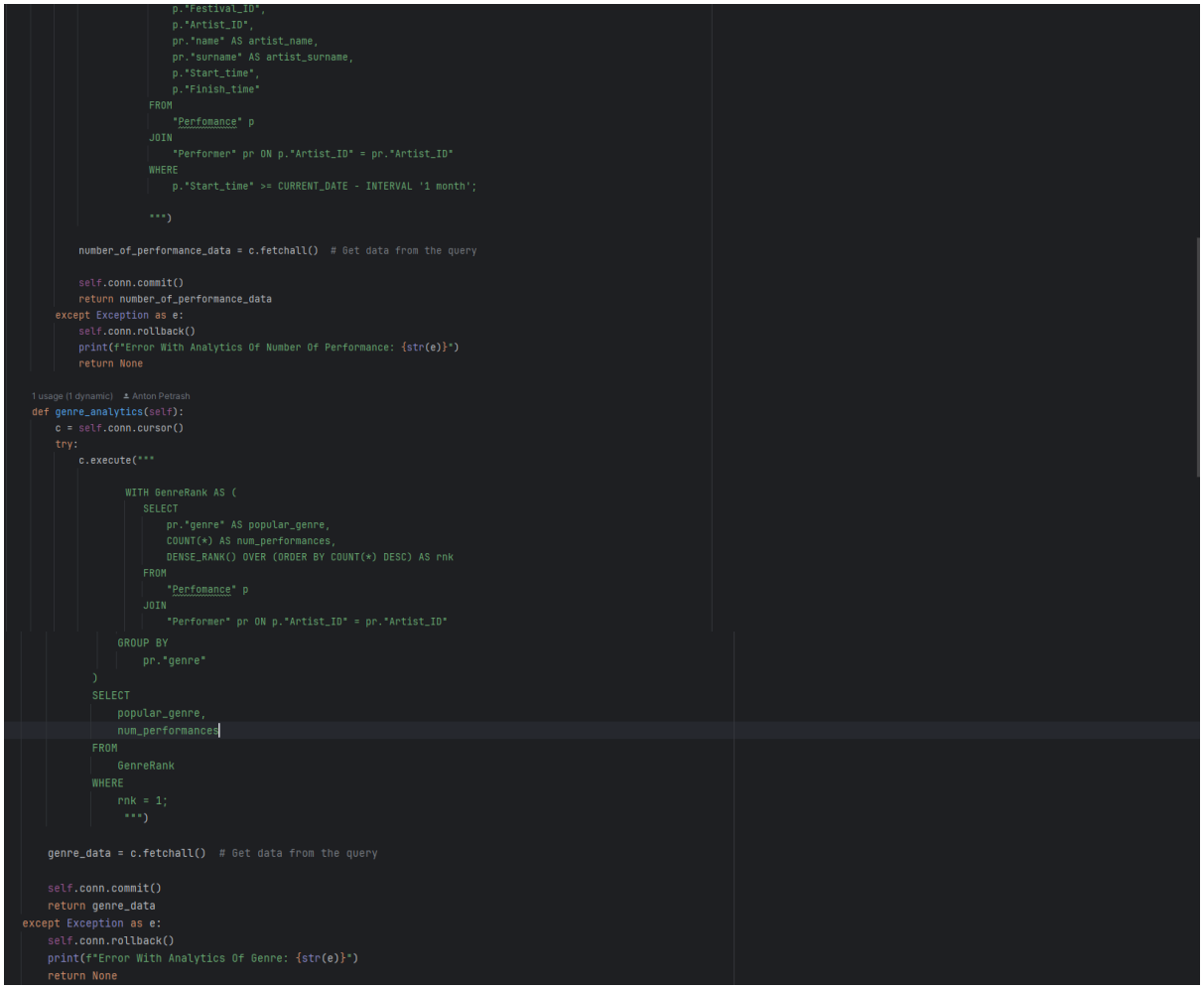
Performance/model.py

```python
class ModelPerformance:
    Anton Petrash
    def __init__(self, db_model):
        self.conn = db_model.conn

    1 usage (1 dynamic)  Anton Petrash
    def add_Performance(self, Performance_ID, Festival_ID, Artist_ID, Start_time, Finish_time):
        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 "Performer" WHERE "Artist_ID" = %s', (Artist_ID,))
            Performer_exists = c.fetchone()

            if not Festival_exists or not Performer_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 "Perfomance" ("Preformance_ID", "Festival_ID", "Artist_ID", '
                    '"Start_time", "Finish_time") VALUES (%s, %s, %s, %s, %s)',
                    (Performance_ID, Festival_ID, Artist_ID, Start_time, Finish_time))
                self.conn.commit()
                return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Adding A Performance: {str(e)}")
            return False

    1 usage (1 dynamic)  Anton Petrash
    def get_all_Performance(self):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Perfomance"')
        return c.fetchall()
def update_Performance(self, Performance_ID, Festival_ID, Artist_ID, Start_time, Finish_time):
    c = self.conn.cursor()
    try:
        # Attempting to update a record
        c.execute('UPDATE "Perfomance" SET "Festival_ID"=%s, "Artist_ID"=%s, "Start_time"=%s, '
                  '"Finish_time"=%s WHERE "Preformance_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

1 usage (1 dynamic)  Anton Petrash
def delete_Performance(self, Performance_ID):
    c = self.conn.cursor()
    try:
        # Attempting to update a record
        c.execute('DELETE FROM "Perfomance" WHERE "Preformance_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
        self.conn.rollback()
        print(f"Error With Deleting A Performance: {str(e)}")
        return False   # Returns False if insertion fails

2 usages (2 dynamic)  Anton Petrash
def check_Performance_existence(self, Performance_ID):
    c = self.conn.cursor()
    c.execute('SELECT 1 FROM "Perfomance" WHERE "Preformance_ID" = %s', (Performance_ID,))
    return bool(c.fetchone())

1 usage (1 dynamic)  Anton Petrash
def create_Performance_sequence(self):
    c = self.conn.cursor()
    c.execute("""
```

```
        DO $$
        BEGIN
            IF NOT EXISTS (SELECT 1 FROM pg_sequences WHERE schemaname = 'public' AND sequencename = 'perfomance_id_seq') THEN
                CREATE SEQUENCE perfomance_id_seq;
            ELSE
                DROP SEQUENCE perfomance_id_seq;
                CREATE SEQUENCE perfomance_id_seq;
            END IF;
        END $$;
        """)
    self.conn.commit()

1 usage (1 dynamic)  Anton Petrash
def generate_rand_Performance_data(self, number_of_operations):
    c = self.conn.cursor()
    try:
        c.execute("""
        INSERT INTO "Perfomance" ("Preformance_ID", "Festival_ID", "Artist_ID", "Start_time", "Finish_time")
        select * from (
        SELECT
               nextval('perfomance_id_seq'),
               floor(random() * (SELECT max("Festival_ID") FROM "Festival") + 1),
               floor(random() * (SELECT max("Artist_ID") FROM "Performer") + 1),
               ('2023-01-01'::date + floor(random() * 3) * interval '1 day' + floor(random() * 12) * interval '1 month' + floor(random() * 31) * interval '1 day') as begin1,
               ('2023-01-01'::date + floor(random() * 3) * interval '1 day' + floor(random() * 12) * interval '1 month' + floor(random() * 31) * interval '1 day') as end1
           FROM generate_series(1, %s)) as t
        WHERE begin1 < end1
             """, (number_of_operations,))

        self.conn.commit()
        return True
    except Exception as e:
        self.conn.rollback()
        print(f"Error With Creating A Performance: {str(e)}")
        return False


1 usage (1 dynamic)  Anton Petrash
def truncate_Performance_table(self):
  c = self.conn.cursor()
  try:
      # Insert data
      c.execute("""DELETE FROM "Perfomance" """)
      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:
    Anton Petrash
    def __init__(self, db_model):
        self.conn = db_model.conn

    1 usage (1 dynamic)  Anton Petrash
    def popular_artist(self):
        c = self.conn.cursor()
        try:
            c.execute("""
                    SELECT "Artist_ID", "name", "surname", "num_performances"
                    FROM (
                        SELECT pr."Artist_ID", pr."name", pr."surname",
                        COUNT(*) AS num_performances,
                    DENSE_RANK() OVER (ORDER BY COUNT(*) DESC) AS rnk
                    FROM "Perfomance" p
                    JOIN "Performer" pr ON p."Artist_ID" = pr."Artist_ID"
                    GROUP BY pr."Artist_ID", pr."name", pr."surname"
                    ) ranked
                    WHERE rnk = 1;

                    """)

            popular_artist_data = c.fetchall()  # Get 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

    1 usage (1 dynamic)  Anton Petrash
    def number_of_performance(self):
        c = self.conn.cursor()
        try:
            c.execute("""
                        SELECT
                            p."Preformance_ID",
```



```
                    p."Festival_ID",
                    p."Artist_ID",
                    pr."name" AS artist_name,
                    pr."surname" AS artist_surname,
                    p."Start_time",
                    p."Finish_time"
                FROM
                    "Perfomance" p
                JOIN
                    "Performer" pr ON p."Artist_ID" = pr."Artist_ID"
                WHERE
                    p."Start_time" >= CURRENT_DATE - INTERVAL '1 month';

                ***)

        number_of_performance_data = c.fetchall()  # Get data from the query

        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

1 usage (1 dynamic)  Anton Petrash
def genre_analytics(self):
    c = self.conn.cursor()
    try:
        c.execute("""

              WITH GenreRank AS (
                SELECT
                    pr."genre" AS popular_genre,
                    COUNT(*) AS num_performances,
                    DENSE_RANK() OVER (ORDER BY COUNT(*) DESC) AS rnk
                FROM
                    "Perfomance" p
                JOIN
                    "Performer" pr ON p."Artist_ID" = pr."Artist_ID"
            GROUP BY
                pr."genre"
        )
        SELECT
            popular_genre,
            num_performances
        FROM
            GenreRank
        WHERE
            rnk = 1;
             """)

    genre_data = c.fetchall()  # Get data from the query

    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 Perfomance
Performer – Робота з даними із таблиці Performer
Festival – Робота з даними із таблиці Festival
Analytics – Виклик запитів для пункту №3

Мій телеграм: https://t.me/Redivius
Мій ГітХаб: https://github.com/AntonPetrash/RGR_BD.git