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 III курсу

групи КВ-23

Домущі Д.Д.

Перевірив:_____

Київ – 2024

Мета: здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

Посилання на репозиторій: https://github.com/asdqweghj/BDRGR

Телеграм: @cvvhella

Виконання роботи Сутності предметної області

1. Користувач(User),з атрибутами: код користувача, ім’я, прізвище,електронна пошта, номер телефона, дата реєстрації. Призначена для збереження інформації про користувачів платформи, що бронюють спортивні майданчики;

2. Спортивні Заклади (Venues), з атрибутами: код спортивного закладу, назва закладу, адреса, місто, місткість закладу. Призначена для зберігання даних про спортивні заклади, які пропонують можливості для занять спортом;

3. Майданчик/Зал (Facility), з атрибутами: код майданчика, код закладу, назва майданчика, тип майданчика. Призначена для зберігання інформації про майданчики, доступні для бронювання в спортивних закладах;

4. Бронювання (Booking), з атрибутами: код бронювання, код користувача, код майданчика, дата, час початку, час завершення, статус. Призначена для: управління бронюваннями користувачів на майданчиках у спортивних закладах;

5. Оплата (Payment), з атрибутами: код оплати, код бронювання, сума, дата оплати, статус. Призначена для зберігання інформації про оплати користувачів за бронювання майданчиків;

Зв’язки між сутностями предметної області

Сутність "Користувач" (User) має зв’язок 1:N по відношенню до сутності "Бронювання" (Booking), оскільки один користувач може створити кілька бронювань.

Сутність "Спортивний заклад" (Venue) має зв’язок 1:N по відношенню до сутності "Майданчик/Зал" (Facility), оскільки один спортивний заклад може містити кілька майданчиків або зон для занять.

Сутність "Майданчик/Зал" (Facility) має зв’язок 1:N по відношенню до сутності "Бронювання" (Booking), оскільки один майданчик може бути забронbований кілька разів на різні дати та час.

Сутність "Бронювання" (Booking) має зв’язок 1:1 по відношенню до сутності "Оплата" (Payment), оскільки кожне бронювання має лише одну відповідну оплату.

Графічне подання логічної моделі «Сутність-зв’язок» зображено на рисунку 1.

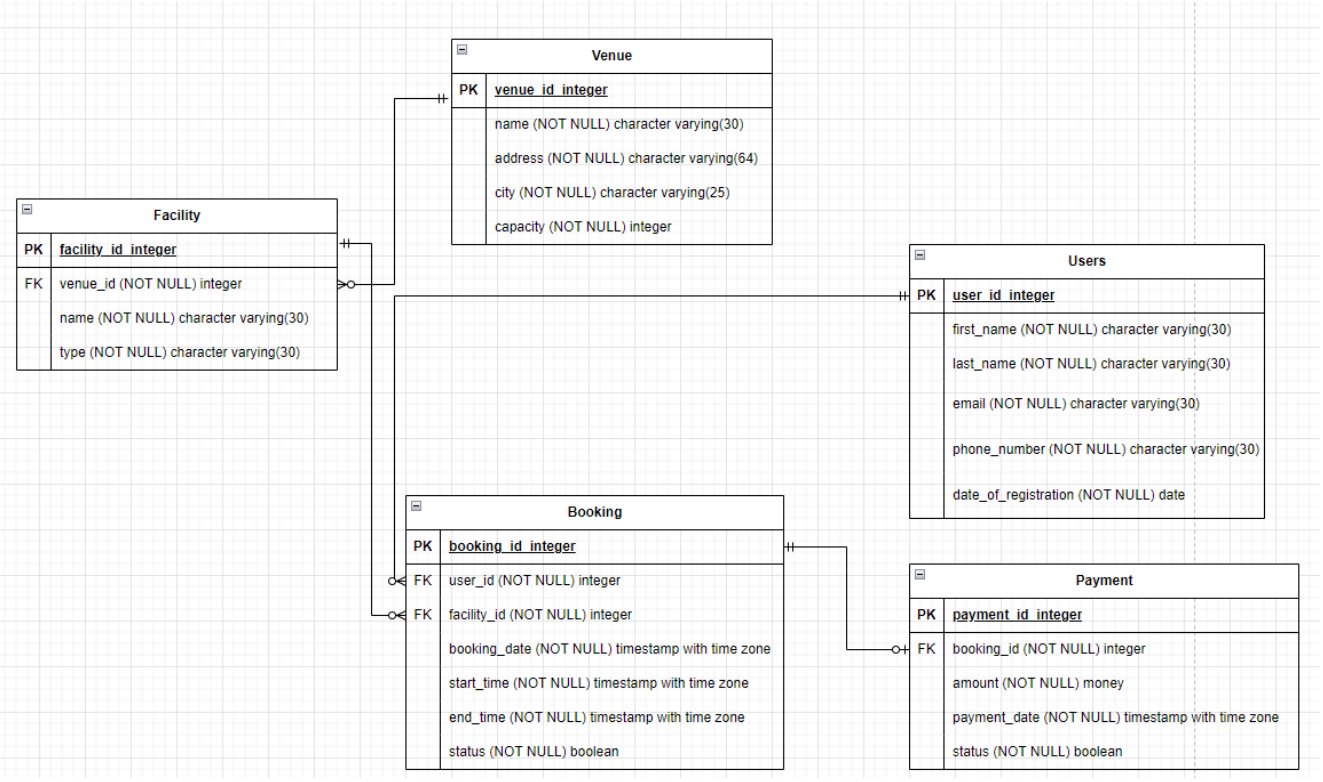


Рисунок 1– Логічна модель

Середовище та компоненти розробки

У процесі розробки була використана мова програмування Python(3.12.3), інтегроване середовище розробки Visual Studio Code, а також була використана бібліотека “Psycopg3”.

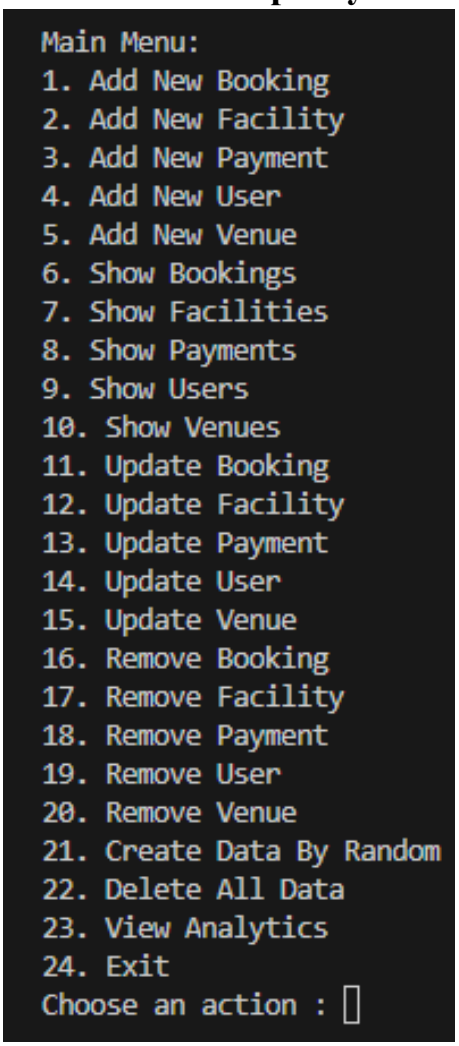
Шаблон проектування

MVC – це шаблон проектування, що використовується у програмі. Кожен компонент відповідає за певну функціональну частину:

1. Model – це клас, що відображає логіку роботи з даними, обробляє всі операції з даними, такі як додавання, оновлення, вилучення.
2. View – це клас, через який користувач взаємодіє з програмою. У даному випадку, консольний інтерфейс, який відображає дані для користувача та зчитує їх з екрану.
3. Controller – це клас, який відповідає за зв'язок між користувачем і системою. Він приймає введені користувачем дані та обробляє їх. В залежності від результатів, викликає відповідні дії з Model або View.

Даний підхід дозволяє розділити логіку програми на логічні компоненти, що полегшує розробку, тестування і підтримку продукту.

Схема меню користувача



Опис функціональності кожного пункту

1. Add New Booking
2. Add New Facility
3. Add New Payment
4. Add New User
5. Add New Venue

Опція “Add” відповідає за додавання нового запису в 1 з таблиць на вибір.

6. Show Bookings
7. Show Facilities
8. Show Payments
9. Show Users
10. Show Venues

Опція “Show” відповідає за показ всіх даних з 1 таблиці на вибір.

11. Update Booking
12. Update Facility
13. Update Payment
14. Update User
15. Update Venue

Опція “Update” дозволяє оновлювати дані для кожної з таблиць на вибір.

16. Remove Booking
17. Remove Facility
18. Remove Payment
19. Remove User
20. Remove Venue

Опція “Remove” відповідає за видалення даних з таблиці на вибір по id.

21. Create Data By Random

Створення випадкових даних для всіх таблиць.

22. Delete All Data

Видалення всіх даних

23. View Analytics

Перегляд аналітики

24. Exit

Завершення роботи програми

Пункт 1 :

Уведення/редагування/вилучення даних у таблицях бази даних

Додавання нового користувача та спортивного закладу:

```
Choose an action : 4
Input User ID: 1
Input first name: Adam
Input last name: Silver
Input email: a.silver@gmail.com
Input phone number: 0981928721
Input date of registration: 2024-10-11
Successfully Added A User
```

```
Choose an action : 5
Input venue ID: 3
Input name: Sport Arena 3
Input address: st. Shevchenko 1
Input city: Kyiv
Input capacity: 20
Successfully Added A Venue
```

Перевірка присутності нових даних :

```
Choose an action : 9
Users:
ID: 1, First name: Adam, Last name: Silver, Email: a.silver@gmail.com, Phone number: 0981928721, Date of registration: 2024-10-11
```

```
Choose an action : 10
Venues:
ID: 3, Name: Sport Arena 3, Address: st. Shevchenko 1, City: Kyiv, Capacity: 20
```

Редагування :

```
Choose an action : 14
Input User ID: 1
Input first name: Adam
Input last name: Gold
Input email: a.gold@gmail.com
Input phone number: 0981928721
Input date of registration: 2024-10-11
Successfully Updated A User
```

```
Choose an action : 15
Input venue ID: 3
Input name: Sport Arena 3
Input address: st. Shevchenko 2
Input city: Kyiv
Input capacity: 25
Successfully Updated A Venue
```

Перевірка успішності редагування :

```
Choose an action : 9
Users:
ID: 1, First name: Adam, Last name: Gold, Email: a.gold@gmail.com, Phone number: 0981928721, Date of registration: 2024-10-11
```

```
Choose an action : 10
Venues:
ID: 3, Name: Sport Arena 3, Address: st. Shevchenko 2, City: Kyiv, Capacity: 25
```

Видалення даних та перевірка їх відсутності :

```
Choose an action : 19
Input User ID: 1
Successfully Deleted A User
```

```
Choose an action : 20
Input venue ID: 3
Successfully Deleted A Venue
```

```
Choose an action : 9
Users:
```

```
Choose an action : 10
Venues:
```

При введенні неіснуючих даних виводиться відповідна помилка

```
Choose an action : 12
Input Facility ID: 1
Facility With This ID Does Not Exist
```

Пункт 2:

Результат створення випадково згенерованих даних для всіх таблиць

```
Choose an action : 21
Input Number Of Generations: 3
Successfully Created Booking Sequence
3 Bookings Successfully Created
Successfully Created Facility Sequence
3 Facilities Successfully Created
Successfully Created Payment Sequence
3 Payments Successfully Created
Successfully Generated User Sequence
3 Users Successfully Generated
Successfully Generated Venue Sequence
3 Venues Successfully Generated
```

```
Choose an action : 6
Bookings:
Booking ID: 1,User ID: 5, Facility ID: 2 Booking date: 10:50:09.981386, Start time: 17:43:08.530000, End time: 17:55:45.470000, Status: True
Booking ID: 2,User ID: 6, Facility ID: 2 Booking date: 06:55:44.844504, Start time: 19:33:58.990000, End time: 21:57:22.590000, Status: True
Booking ID: 3,User ID: 3, Facility ID: 1 Booking date: 00:40:33.491749, Start time: 23:31:21.630000, End time: 19:53:49.670000, Status: False
Booking ID: 4,User ID: 5, Facility ID: 1 Booking date: 07:44:56.849988, Start time: 22:37:19.900000, End time: 02:22:15.020000, Status: False
Booking ID: 5,User ID: 1, Facility ID: 1 Booking date: 05:41:28.820574, Start time: 21:18:56.300000, End time: 03:00:19.310000, Status: True
Booking ID: 6,User ID: 1, Facility ID: 1 Booking date: 06:27:09.311106, Start time: 00:26:24.150000, End time: 01:09:29.010000, Status: False
```

```
Choose an action : 7
Facilities:
Facility ID: 1,Venue ID: 2, Name: Tennis, Type: Indoor
Facility ID: 2,Venue ID: 2, Name: Football, Type: Indoor
Facility ID: 3,Venue ID: 1, Name: Tennis, Type: Outdoor
```

Відповідний SQL запит:

```
        c.execute("""
          INSERT INTO "booking" ("booking_id", "booking_date", "start_time", "end_time", "status", "user_id", "facility_id")
        SELECT
            nextval('booking_id_seq'),
            -- Випадкова дата і час в межах 30 днів від сьогодні
            (CURRENT_DATE + (random() * 30)::int * interval '1 day') + (random() * interval '12 hours')::time AS booking_date,
            -- Випадковий час початку (з округленням до секунд)
            (CURRENT_TIME + (random() * interval '10 hours'))::time(2) AS start_time,
            -- Випадковий час завершення (на 2 години пізніше)
            ((CURRENT_TIME + (random() * interval '10 hours')) + interval '2 hours')::time(2) AS end_time,
            -- Випадковий статус
            (random() > 0.5)::boolean AS status,
            -- Випадковий user_id
            floor(random() * (SELECT max("user_id") FROM "Users") + 1)::integer AS user_id,
            -- Випадковий facility_id
            floor(random() * (SELECT max("facility_id") FROM "Facility") + 1)::integer AS facility_id
        FROM generate_series(1, %s);
        """, (number_of_operations,))
```

```
c.execute("""
    INSERT INTO "payment" ("payment_id", "booking_id", "amount", "payment_date", "payment_status")
    SELECT
        nextval('payment_id_seq'),
        floor(random() * (COALESCE((SELECT max("booking_id") FROM "booking"), 1)) + 1)::int AS booking_id,
        round((random() * 100 + 50)::numeric, 2) AS amount,
        clock_timestamp() - (random() * interval '30 days') AS payment_date,
        CASE
            WHEN random() < 0.5 THEN true
            ELSE false
        END AS payment_status
    FROM generate_series(1, %s);
""", (number_of_operations,))
```

```
  c.execute("""
      INSERT INTO "facility" ("facility_id", "facility_name", "facility_type", "venue_id")
      SELECT
          nextval('facility_id_seq'),
          (array['Football', 'Basketball', 'Voleyball', 'Golf', 'Tennis'])[floor(random() * 5) + 1] AS facility_name,
          CASE
              WHEN random() < 0.5 THEN 'Indoor'
              ELSE 'Outdoor'
          END AS facility_type,
          floor(random() * (SELECT max("venue_id") FROM "venue") + 1)::int AS venue_id
      FROM generate_series(1, %s);
  """, (number_of_operations,))
```

```
c.execute("""
INSERT INTO "users" ("user_id", "first_name", "last_name", "email", "phone_number", "date_of_registration")
SELECT
    nextval('user_id_seq'),
    -- Випадкове ім'я
    first_name,
    -- Випадкове прізвище
    last_name,
    -- Генерація email на основі імені та прізвища
    LOWER(first_name || '.' || last_name || '@gmail.com') AS email,
    -- Генерація номера телефону
    '380' || floor(100000000 + random() * 900000000)::bigint,
    -- Дата реєстрації за останній рік
    CURRENT_DATE - floor(random() * 365)::int AS date_of_registration
FROM (
    SELECT
        -- Вибір випадкового імені
        (array['Michael', 'Sofia', 'Tom', 'Alex', 'Stan', 'Anna', 'John', 'Emma', 'Oliver', 'Ava'])[floor(random() * 10) + 1] AS first_name,
        -- Вибір випадкового прізвища
        (array['Wall', 'Johnes', 'Tesla', 'Fire', 'Smith', 'Brown', 'Taylor', 'Wilson', 'Davies', 'Evans'])[floor(random() * 10) + 1] AS last_name
    FROM generate_series(1, %s)
) AS random_data;
""", (number_of_operations,))
```

```
c.execute("""
    INSERT INTO "venue" ("venue_id", "name", "address", "city", "capacity")
    SELECT
        nextval('venue_id_seq'),
        -- Назва спортивного комплексу
        (array['Arena Sports', 'Champion Gym', 'Victory Stadium', 'Golden Field',
           'Elite Fitness', 'Powerhouse Arena', 'Olympic Hall', 'Titanium Dome',
           'Active Life Center', 'Dynamic Gym'])[row_number] AS name,
        -- Унікальна адреса
        'Street ' || row_number || ', Building ' || floor(random() * 100 + 1)::int AS address,
        -- Випадкове місто
        (array['New York', 'Los Angeles', 'Chicago', 'Houston', 'Miami'])[floor(random() * 5) + 1] AS city,
        -- Випадкова місткість
        floor(random() * 100 + 10) AS capacity
    FROM (
        SELECT row_number() OVER () AS row_number
        FROM generate_series(1, %s)
    ) AS numbered_rows;
""", (number_of_operations,))
```

Пункт 3:

Виконано 3 запити:

```
c.execute("""
    SELECT
        v."venue_id",
        v."name" AS venue_name,
        COUNT(b."booking_id") AS total_bookings
    FROM
        "venue" v
    JOIN
        "facility" f ON v."venue_id" = f."venue_id"
    JOIN
        "booking" b ON f."facility_id" = b."facility_id"
    GROUP BY
        v."venue_id",v."name"
    ORDER BY
        total_bookings DESC
    LIMIT 1; -- Найпопулярніше місце
""")
```

```
c.execute("""
    SELECT
        u."user_id",
        u."first_name",
        u."last_name",
        COUNT(b."booking_id") AS total_bookings
    FROM
        "users" u
    JOIN
        "booking" b ON u."user_id" = b."user_id"
    GROUP BY
        u."user_id", u."first_name", u."last_name"
    ORDER BY
        total_bookings DESC
    LIMIT 5; -- П'ять найактивніших користувачів
""")
```

```
c.execute("""
    SELECT
        p."payment_status",
        COUNT(p."payment_id") AS total_payments,
        SUM(p."amount") AS total_revenue
    FROM
        "payment" p
    GROUP BY
        p."payment_status"
    ORDER BY
        total_revenue DESC;
""")
```

Результат запитів:

```
Choose an action : 23
--------------------------------------------------------------------------------
Найпопулярніші місця:
ID: 1, Назва: Arena Sports, Кількість бронювань: 7
--------------------------------------------------------------------------------
Найактивніші користувачі:
ID: 1, Ім'я: John Smith, Кількість бронювань: 4
ID: 6, Ім'я: Tom Smith, Кількість бронювань: 3
ID: 2, Ім'я: Emma Johnes, Кількість бронювань: 2
ID: 3, Ім'я: John Taylor, Кількість бронювань: 1
ID: 5, Ім'я: Ava Wilson, Кількість бронювань: 1
--------------------------------------------------------------------------------
Аналіз платежів:
Статус: Успішні, Загальна сума: 8, Кількість платежів: 866,75 ?
Статус: Неуспішні, Загальна сума: 3, Кількість платежів: 323,05 ?
--------------------------------------------------------------------------------
```

Код модулів

Опис роботи модулів :
Booking– Робота з даними із таблиці booking
Facility– Робота з даними із таблиці facility
Payment– Робота з даними із таблиці payment
Users– Робота з даними із таблиці users
Venue– Робота з даними із таблиці venue
Analytics – Виклик запитів для пункту №3

model.py

```
import psycopg

class Model:
    def __init__(self):
        self.conn = psycopg.connect(
            dbname='postgres',
            user='postgres',
            password='Ddd.12350987',
            host='localhost',
            port=5432
        )
        self.create_tables()

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

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

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

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

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

        if not booking_table_exists:
            c.execute('''
                        CREATE TABLE "booking" (
                            "booking_id" SERIAL PRIMARY KEY,
                            "user_id" INTEGER NOT NULL,
                            "facility_id" INTEGER NOT NULL,
                            "booking_date" TIME NOT NULL,
                            "start_time" TIME NOT NULL,
                            "end_time" TIME NOT NULL,
                            "status" BOOLEAN NOT NULL
                        )
                    ''')
        if not facility_table_exists:
            c.execute('''
                        CREATE TABLE "facility" (
                            "facility_id" SERIAL PRIMARY KEY,
                            "venue_id" INTEGER NOT NULL,
                            "facility_name" TEXT NOT NULL,
                            "facility_type" TEXT NOT NULL

                        )
                    ''')
        if not payment_table_exists:
            c.execute('''
                        CREATE TABLE "payment" (
                            "payment_id" SERIAL PRIMARY KEY,
                            "booking_id" INTEGER NOT NULL,
                            "amount" MONEY NOT NULL,
                            "payment_date" TIME NOT NULL,
                            "payment_status" BOOLEAN NOT NULL
                        )
                    ''')
        if not users_table_exists:
            c.execute('''
                        CREATE TABLE "users" (
```

```
                            "user_id" SERIAL PRIMARY KEY,
                            "first_name" TEXT NOT NULL,
                            "last_name" TEXT NOT NULL,
                            "email" TEXT NOT NULL,
                            "phone_number" TEXT NOT NULL,
                            "date_of_registration" DATE NOT NULL
                        )
                    ''')
        if not venue_table_exists:
            c.execute('''
                        CREATE TABLE "venue" (
                            "venue_id" SERIAL PRIMARY KEY,
                            "name" TEXT NOT NULL,
                            "address" TEXT NOT NULL,
                            "city" TEXT NOT NULL,
                            "capacity" INTEGER NOT NULL
                        )
                    ''')

        self.conn.commit()
```

Booking: model.py

```
class ModelBooking:
    def __init__(self, db_model):
        self.conn = db_model.conn

    def add_booking(self, booking_id, user_id, facility_id,
booking_date, start_time, end_time, status):
        c = self.conn.cursor()
        try:
            # Перевірка, чи існує user_id у таблиці user
            c.execute('SELECT 1 FROM "user" WHERE "user_id" = %s',
(user_id,))
            user_exists = c.fetchone()

            if not user_exists:
                print("Error: User ID does not exist.")
                return False

            # Перевірка, чи існує facility_id у таблиці facility
```

```
            c.execute('SELECT 1 FROM "facility" WHERE "facility_id" =
%s', (facility_id,))
            facility_exists = c.fetchone()

            if not facility_exists:
                print("Error: Facility ID does not exist.")
                return False

            # Додавання нового запису до таблиці booking
            c.execute(
                'INSERT INTO "booking" ("booking_id", "user_id",
"facility_id", "booking_date", "start_time", "end_time", "status") '
                'VALUES (%s, %s, %s, %s, %s, %s, %s)',
                (booking_id, user_id, facility_id, booking_date,
start_time, end_time, status)
            )
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Adding A Booking: {str(e)}")
            return False

    def get_all_bookings(self):
        c = self.conn.cursor()
        try:
            # Отримання всіх записів з таблиці booking
            c.execute('SELECT * FROM "booking"')
            return c.fetchall()
        except Exception as e:
            print(f"Error With Retrieving Bookings: {str(e)}")
            return None

    def update_booking(self, booking_id, user_id, facility_id,
booking_date, start_time, end_time, status):
        c = self.conn.cursor()
        try:
            # Перевірка, чи існує user_id у таблиці user
            c.execute('SELECT 1 FROM "user" WHERE "user_id" = %s',
(user_id,))
            user_exists = c.fetchone()

            if not user_exists:
```

```
                print("Error: User ID does not exist.")
                return False

            # Перевірка, чи існує facility_id у таблиці facility
            c.execute('SELECT 1 FROM "facility" WHERE "facility_id" =
%s', (facility_id,))
            facility_exists = c.fetchone()

            if not facility_exists:
                print("Error: Facility ID does not exist.")
                return False

            # Оновлення запису в таблиці booking
            c.execute(
                'UPDATE "booking" SET "user_id" = %s, "facility_id" =
%s, "booking_date" = %s, "start_time" = %s, '
                '"end_time" = %s, "status" = %s WHERE "booking_id" =
%s',
                (user_id, facility_id, booking_date, start_time,
end_time, status, booking_id)
            )
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Updating A Booking: {str(e)}")
            return False

    def delete_booking(self, booking_id):
        c = self.conn.cursor()
        try:
            # Видалення запису з таблиці booking
            c.execute('DELETE FROM "booking" WHERE "booking_id" = %s',
(booking_id,))
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Deleting A Booking: {str(e)}")
            return False

    def check_booking_existence(self, booking_id):
        c = self.conn.cursor()
```

```
        try:
            # Перевірка існування запису
            c.execute('SELECT 1 FROM "booking" WHERE "booking_id" =
%s', (booking_id,))
            return bool(c.fetchone())
        except Exception as e:
            print(f"Error With Checking Booking Existence: {str(e)}")
            return False

    def create_booking_sequence(self):
        c = self.conn.cursor()
        try:
            # Створення або оновлення послідовності для booking_id
            c.execute("""
                DO $$
                DECLARE
                    max_id INT;
                BEGIN
                    -- Знаходимо максимальний booking_id
                    SELECT COALESCE(MAX(booking_id), 0) INTO max_id
FROM "booking";

                    -- Перевіряємо, чи існує послідовність
                    IF NOT EXISTS (
                        SELECT 1
                        FROM pg_sequences
                        WHERE schemaname = 'public' AND sequencename =
'booking_id_seq'
                    ) THEN
                        -- Створення нової послідовності
                        EXECUTE 'CREATE SEQUENCE booking_id_seq START
WITH ' || (max_id + 1);
                    ELSE
                        -- Оновлення існуючої послідовності
                        EXECUTE 'ALTER SEQUENCE booking_id_seq RESTART
WITH ' || (max_id + 1);
                    END IF;
                END $$;
            """)
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
```

```
            print(f"Error With Creating Booking Sequence: {str(e)}")
            return False

    def generate_rand_booking_data(self, number_of_operations):
        c = self.conn.cursor()
        try:
            c.execute("""
              INSERT INTO "booking" ("booking_id", "booking_date",
"start_time", "end_time", "status", "user_id", "facility_id")
            SELECT
                nextval('booking_id_seq'),
                -- Випадкова дата і час в межах 30 днів від сьогодні
                (CURRENT_DATE + (random() * 30)::int * interval '1
day') + (random() * interval '12 hours')::time AS booking_date,
                -- Випадковий час початку (з округленням до секунд)
                (CURRENT_TIME + (random() * interval '10
hours'))::time(2) AS start_time,
                -- Випадковий час завершення (на 2 години пізніше)
                ((CURRENT_TIME + (random() * interval '10 hours')) +
interval '2 hours')::time(2) AS end_time,
                -- Випадковий статус
                (random() > 0.5)::boolean AS status,
                -- Випадковий user_id
                floor(random() * (SELECT max("user_id") FROM "Users")
+ 1)::integer AS user_id,
                -- Випадковий facility_id
                floor(random() * (SELECT max("facility_id") FROM
"Facility") + 1)::integer AS facility_id
            FROM generate_series(1, %s);
            """, (number_of_operations,))
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Generating Booking Data: {str(e)}")
            return False

    def truncate_booking_table(self):
        c = self.conn.cursor()
        try:
            # Очищення таблиці booking
            c.execute('DELETE FROM "booking"')
            self.conn.commit()
```

```
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Truncating Booking Table: {str(e)}")
            return False
```

Facility: model.py

```
class ModelFacility:
    def __init__(self, db_model):
        self.conn = db_model.conn

    def add_facility(self, facility_id, facility_name, facility_type,
venue_id):
        c = self.conn.cursor()
        try:
            # Перевірка, чи існує venue_id у таблиці venue
            c.execute('SELECT 1 FROM "venue" WHERE "venue_id" = %s',
(venue_id,))
            venue_exists = c.fetchone()

            if not venue_exists:
                print("Error: Venue ID does not exist.")
                return False

            # Додавання нового запису до таблиці facility
            c.execute(
                'INSERT INTO "facility" ("facility_id",
"facility_name", "facility_type", "venue_id") VALUES (%s, %s, %s,
%s)',
                (facility_id, facility_name, facility_type, venue_id,)
            )
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Adding A Facility: {str(e)}")
            return False

    def get_all_facilities(self):
        c = self.conn.cursor()
```

```
        try:
            # Отримання всіх записів з таблиці facility
            c.execute('SELECT * FROM "facility"')
            return c.fetchall()
        except Exception as e:
            print(f"Error With Retrieving Facilities: {str(e)}")
            return None

    def update_facility(self, facility_id, facility_name,
facility_type, venue_id):
        c = self.conn.cursor()
        try:
            # Перевірка, чи існує venue_id у таблиці venue
            c.execute('SELECT 1 FROM "venue" WHERE "venue_id" = %s',
(venue_id,))
            venue_exists = c.fetchone()

            if not venue_exists:
                print("Error: Venue ID does not exist.")
                return False

            # Оновлення запису в таблиці facility
            c.execute(
                'UPDATE "facility" SET "facility_name" = %s,
"facility_type" = %s, "venue_id" = %s WHERE "facility_id" = %s',
                (facility_name, facility_type, venue_id, facility_id)
            )
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Updating A Facility: {str(e)}")
            return False

    def delete_facility(self, facility_id):
        c = self.conn.cursor()
        try:
            # Видалення запису з таблиці facility
            c.execute('DELETE FROM "facility" WHERE "facility_id" =
%s', (facility_id,))
            self.conn.commit()
            return True
        except Exception as e:
```

```
            self.conn.rollback()
            print(f"Error With Deleting A Facility: {str(e)}")
            return False

    def check_facility_existence(self, facility_id):
        c = self.conn.cursor()
        try:
            # Перевірка існування запису
            c.execute('SELECT 1 FROM "facility" WHERE "facility_id" =
%s', (facility_id,))
            return bool(c.fetchone())
        except Exception as e:
            print(f"Error With Checking Facility Existence: {str(e)}")
            return False

    def create_facility_sequence(self):
        c = self.conn.cursor()
        try:
            # Створення або оновлення послідовності для facility_id
            c.execute("""
                DO $$
                DECLARE
                    max_id INT;
                BEGIN
                    -- Знаходимо максимальний facility_id
                    SELECT COALESCE(MAX(facility_id), 0) INTO max_id
FROM "facility";

                    -- Перевіряємо, чи існує послідовність
                    IF NOT EXISTS (
                        SELECT 1
                        FROM pg_sequences
                        WHERE schemaname = 'public' AND sequencename =
'facility_id_seq'
                    ) THEN
                        -- Створення нової послідовності
                        EXECUTE 'CREATE SEQUENCE facility_id_seq START
WITH ' || (max_id + 1);
                    ELSE
                        -- Оновлення існуючої послідовності
                        EXECUTE 'ALTER SEQUENCE facility_id_seq
RESTART WITH ' || (max_id + 1);
                    END IF;
```

```
                END $$;
            """)
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Creating Facility Sequence: {str(e)}")
            return False

    def generate_rand_facility_data(self, number_of_operations):
        c = self.conn.cursor()
        try:
            c.execute("""
                INSERT INTO "facility" ("facility_id",
"facility_name", "facility_type", "venue_id")
                SELECT
                    nextval('facility_id_seq'),
                    (array['Football', 'Basketball', 'Voleyball',
'Golf', 'Tennis'])[floor(random() * 5) + 1] AS facility_name,
                    CASE
                        WHEN random() < 0.5 THEN 'Indoor'
                        ELSE 'Outdoor'
                    END AS facility_type,
                    floor(random() * (SELECT max("venue_id") FROM
"venue") + 1)::int AS venue_id
                FROM generate_series(1, %s);
            """, (number_of_operations,))
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Generating Facility Data: {str(e)}")
            return False

    def truncate_facility_table(self):
        c = self.conn.cursor()
        try:
            # Очищення таблиці facility
            c.execute('DELETE FROM "facility"')
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
```

```
            print(f"Error With Truncating Facility Table: {str(e)}")
            return False
```

Payment: model.py

```
class ModelPayment:
    def __init__(self, db_model):
        self.conn = db_model.conn

    def add_payment(self, payment_id, booking_id, amount,
payment_date, payment_status):
        c = self.conn.cursor()
        try:
            # Перевірка, чи існує booking_id у таблиці booking
            c.execute('SELECT 1 FROM "booking" WHERE "booking_id" =
%s', (booking_id,))
            booking_exists = c.fetchone()

            if not booking_exists:
                print("Error: Booking ID does not exist.")
                return False

            # Додавання нового запису до таблиці payment
            c.execute(
                'INSERT INTO "payment" ("payment_id", "booking_id",
"amount", "payment_date", "payment_status") VALUES (%s, %s, %s, %s,
%s)',
                (payment_id, booking_id, amount, payment_date,
payment_status)
            )
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Adding A Payment: {str(e)}")
            return False

    def get_all_payments(self):
        c = self.conn.cursor()
```

```
        try:
            # Отримання всіх записів з таблиці payment
            c.execute('SELECT * FROM "payment"')
            return c.fetchall()
        except Exception as e:
            print(f"Error With Retrieving Payments: {str(e)}")
            return None

    def update_payment(self, payment_id, booking_id, amount,
payment_date, payment_status):
        c = self.conn.cursor()
        try:
            # Перевірка, чи існує booking_id у таблиці booking
            c.execute('SELECT 1 FROM "booking" WHERE "booking_id" =
%s', (booking_id,))
            booking_exists = c.fetchone()

            if not booking_exists:
                print("Error: Booking ID does not exist.")
                return False

            # Оновлення запису в таблиці payment
            c.execute(
                'UPDATE "payment" SET "booking_id" = %s, "amount" =
%s, "payment_date" = %s, "payment_status" = %s WHERE "payment_id" =
%s',
                (booking_id, amount, payment_date, payment_status,
payment_id)
            )
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Updating A Payment: {str(e)}")
            return False

    def delete_payment(self, payment_id):
        c = self.conn.cursor()
        try:
            # Видалення запису з таблиці payment
            c.execute('DELETE FROM "payment" WHERE "payment_id" = %s',
(payment_id,))
            self.conn.commit()
```

```
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Deleting A Payment: {str(e)}")
            return False

    def check_payment_existence(self, payment_id):
        c = self.conn.cursor()
        try:
            # Перевірка існування запису
            c.execute('SELECT 1 FROM "payment" WHERE "payment_id" =
%s', (payment_id,))
            return bool(c.fetchone())
        except Exception as e:
            print(f"Error With Checking Payment Existence: {str(e)}")
            return False

    def create_payment_sequence(self):
        c = self.conn.cursor()
        try:
            # Створення або оновлення послідовності для payment_id
            c.execute("""
                DO $$
                DECLARE
                    max_id INT;
                BEGIN
                    -- Знаходимо максимальний payment_id
                    SELECT COALESCE(MAX(payment_id), 0) INTO max_id
FROM "payment";

                    -- Перевіряємо, чи існує послідовність
                    IF NOT EXISTS (
                        SELECT 1
                        FROM pg_sequences
                        WHERE schemaname = 'public' AND sequencename =
'payment_id_seq'
                    ) THEN
                        -- Створення нової послідовності
                        EXECUTE 'CREATE SEQUENCE payment_id_seq START
WITH ' || (max_id + 1);
                    ELSE
                        -- Оновлення існуючої послідовності
```

```
                        EXECUTE 'ALTER SEQUENCE payment_id_seq RESTART
WITH ' || (max_id + 1);
                    END IF;
                END $$;
            """)
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Creating Payment Sequence: {str(e)}")
            return False

    def generate_rand_payment_data(self, number_of_operations):
        c = self.conn.cursor()
        try:
            c.execute("""
                INSERT INTO "payment" ("payment_id", "booking_id",
"amount", "payment_date", "payment_status")
                SELECT
                    nextval('payment_id_seq'),
                    floor(random() * (COALESCE((SELECT
max("booking_id") FROM "booking"), 1)) + 1)::int AS booking_id,
                    round((random() * 100 + 50)::numeric, 2) AS
amount,
                    clock_timestamp() - (random() * interval '30
days') AS payment_date,
                    CASE
                        WHEN random() < 0.5 THEN true
                        ELSE false
                    END AS payment_status
                FROM generate_series(1, %s);
            """, (number_of_operations,))
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Generating Payment Data: {str(e)}")
            return False

    def truncate_payment_table(self):
        c = self.conn.cursor()
        try:
            # Очищення таблиці payment
```

```
            c.execute('DELETE FROM "payment"')
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Truncating Payment Table: {str(e)}")
            return False
```

Users: model.py

```
class ModelUser:
    def __init__(self, db_model):
        self.conn = db_model.conn

    def add_user(self, user_id, first_name, last_name, email,
phone_number, date_of_registration):
        c = self.conn.cursor()
        try:
            c.execute(
                'INSERT INTO "users" ("user_id", "first_name",
"last_name", "email", "phone_number", "date_of_registration") VALUES
(%s, %s, %s, %s, %s, %s)',
                (user_id, first_name, last_name, email, phone_number,
date_of_registration)
            )
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error adding user: {str(e)}")
            return False

    def get_all_users(self):
        c = self.conn.cursor()
        try:
            c.execute('SELECT * FROM "users"')
            return c.fetchall()
        except Exception as e:
            print(f"Error retrieving users: {str(e)}")
            return None

    def update_user(self, user_id, first_name, last_name, email,
phone_number, date_of_registration):
```

```
        c = self.conn.cursor()
        try:
            c.execute(
                'UPDATE "users" SET "first_name" = %s, "last_name" =
%s, "email" = %s, "phone_number" = %s, "date_of_registration" = %s
WHERE "user_id" = %s',
                (first_name, last_name, email, phone_number,
date_of_registration, user_id)
            )
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error updating user: {str(e)}")
            return False

    def delete_user(self, user_id):
        c = self.conn.cursor()
        try:
            c.execute('DELETE FROM "users" WHERE "user_id" = %s',
(user_id,))
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error deleting user: {str(e)}")
            return False

    def check_user_existence(self, user_id):
        c = self.conn.cursor()
        try:
            c.execute('SELECT 1 FROM "users" WHERE "user_id" = %s',
(user_id,))
            return bool(c.fetchone())
        except Exception as e:
            print(f"Error checking user existence: {str(e)}")
            return False

    def create_user_sequence(self):
        c = self.conn.cursor()
        try:
            c.execute('''
                DO $$
```

```
                DECLARE
                    max_id INT;
                BEGIN
                    SELECT COALESCE(MAX(user_id), 0) INTO max_id FROM
"users";

                    IF NOT EXISTS (
                        SELECT 1
                        FROM pg_sequences
                        WHERE schemaname = 'public' AND sequencename =
'user_id_seq'
                    ) THEN
                        EXECUTE 'CREATE SEQUENCE user_id_seq START
WITH ' || (max_id + 1);
                    ELSE
                        EXECUTE 'ALTER SEQUENCE user_id_seq RESTART
WITH ' || (max_id + 1);
                    END IF;
                END $$;
            ''')
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error creating user sequence: {str(e)}")
            return False

    def generate_rand_user_data(self, number_of_operations):
        c = self.conn.cursor()
        try:
            c.execute("""
            INSERT INTO "users" ("user_id", "first_name", "last_name",
"email", "phone_number", "date_of_registration")
            SELECT
                nextval('user_id_seq'),
                -- Випадкове ім'я
                first_name,
                -- Випадкове прізвище
                last_name,
                -- Генерація email на основі імені та прізвища
                LOWER(first_name || '.' || last_name || '@gmail.com')
AS email,
                -- Генерація номера телефону
```

```
                '380' || floor(100000000 + random() *
900000000)::bigint,
                -- Дата реєстрації за останній рік
                CURRENT_DATE - floor(random() * 365)::int AS
date_of_registration
            FROM (
                SELECT
                    -- Вибір випадкового імені
                    (array['Michael', 'Sofia', 'Tom', 'Alex', 'Stan',
'Anna', 'John', 'Emma', 'Oliver', 'Ava'])[floor(random() * 10) + 1] AS
first_name,
                    -- Вибір випадкового прізвища
                    (array['Wall', 'Johnes', 'Tesla', 'Fire', 'Smith',
'Brown', 'Taylor', 'Wilson', 'Davies', 'Evans'])[floor(random() * 10)
+ 1] AS last_name
                FROM generate_series(1, %s)
            ) AS random_data;
            """, (number_of_operations,))
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Generating User Data: {str(e)}")
            return False

    def truncate_users_table(self):
        c = self.conn.cursor()
        try:
            c.execute('DELETE FROM "users"')
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error truncating users table: {str(e)}")
            return False
```

Venue: model.py

```
class ModelVenue:
    def __init__(self, db_model):
        self.conn = db_model.conn
```

```
    def add_venue(self, venue_id, name, address, city, capacity):
        c = self.conn.cursor()
        try:
            c.execute('INSERT INTO "venue" ("venue_id", "name",
"address", "city", "capacity") VALUES (%s, %s, %s, %s, %s)',
                      (venue_id, name, address, city, capacity))
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error Adding Venue: {str(e)}")
            return False

    def get_all_venues(self):
        c = self.conn.cursor()
        try:
            c.execute('SELECT * FROM "venue"')
            return c.fetchall()
        except Exception as e:
            print(f"Error Retrieving Venues: {str(e)}")
            return None

    def update_venue(self, venue_id, name, address, city, capacity):
        c = self.conn.cursor()
        try:
            c.execute('UPDATE "venue" SET "name" = %s, "address" = %s,
"city" = %s, "capacity" = %s WHERE "venue_id" = %s',
                      (name, address, city, capacity, venue_id))
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error Updating Venue: {str(e)}")
            return False

    def delete_venue(self, venue_id):
        c = self.conn.cursor()
        try:
            c.execute('DELETE FROM "venue" WHERE "venue_id" = %s',
(venue_id,))
            self.conn.commit()
            return True
```

```
        except Exception as e:
            self.conn.rollback()
            print(f"Error Deleting Venue: {str(e)}")
            return False

    def check_venue_existence(self, venue_id):
        c = self.conn.cursor()
        try:
            c.execute('SELECT 1 FROM "venue" WHERE "venue_id" = %s',
(venue_id,))
            return bool(c.fetchone())
        except Exception as e:
            print(f"Error Checking Venue Existence: {str(e)}")
            return False

    def create_venue_sequence(self):
        c = self.conn.cursor()
        try:
            c.execute("""
                DO $$
                DECLARE
                    max_id INT;
                BEGIN
                    SELECT COALESCE(MAX(venue_id), 0) INTO max_id FROM
"venue";

                    IF NOT EXISTS (
                        SELECT 1
                        FROM pg_sequences
                        WHERE schemaname = 'public' AND sequencename =
'venue_id_seq'
                    ) THEN
                        EXECUTE 'CREATE SEQUENCE venue_id_seq START
WITH ' || (max_id + 1);
                    ELSE
                        EXECUTE 'ALTER SEQUENCE venue_id_seq RESTART
WITH ' || (max_id + 1);
                    END IF;
                END $$;
            """)
            self.conn.commit()
            return True
        except Exception as e:
```

```
            self.conn.rollback()
            print(f"Error Creating Venue Sequence: {str(e)}")
            return False

    def generate_rand_venue_data(self, number_of_operations):
        c = self.conn.cursor()
        try:
            c.execute("""
                INSERT INTO "venue" ("venue_id", "name", "address",
"city", "capacity")
                SELECT
                    nextval('venue_id_seq'),
                    -- Назва спортивного комплексу
                    (array['Arena Sports', 'Champion Gym', 'Victory
Stadium', 'Golden Field',
                        'Elite Fitness', 'Powerhouse Arena', 'Olympic
Hall', 'Titanium Dome',
                        'Active Life Center', 'Dynamic
Gym'])[row_number] AS name,
                    -- Унікальна адреса
                    'Street ' || row_number || ', Building ' ||
floor(random() * 100 + 1)::int AS address,
                    -- Випадкове місто
                    (array['New York', 'Los Angeles', 'Chicago',
'Houston', 'Miami'])[floor(random() * 5) + 1] AS city,
                    -- Випадкова місткість
                    floor(random() * 100 + 10) AS capacity
                FROM (
                    SELECT row_number() OVER () AS row_number
                    FROM generate_series(1, %s)
                ) AS numbered_rows;
            """, (number_of_operations,))
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error Generating Venue Data: {str(e)}")
            return False

    def truncate_venue_table(self):
        c = self.conn.cursor()
        try:
            c.execute('DELETE FROM "venue"')
```

```
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error Truncating Venue Table: {str(e)}")
            return False
```

Analytics: model.py

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

    def most_booked_venue(self):
        """
        Запит для визначення найпопулярнішого місця (Venue),
        на основі кількості бронювань (Booking).
        """
        c = self.conn.cursor()
        try:
            c.execute("""
                SELECT
                    v."venue_id",
                    v."name" AS venue_name,
                    COUNT(b."booking_id") AS total_bookings
                FROM
                    "venue" v
                JOIN
                    "facility" f ON v."venue_id" = f."venue_id"
                JOIN
                    "booking" b ON f."facility_id" = b."facility_id"
                GROUP BY
                    v."venue_id",v."name"
                ORDER BY
                    total_bookings DESC
                LIMIT 1; -- Найпопулярніше місце
            """)

            data = c.fetchall()
            self.conn.commit()
            return data
        except Exception as e:
            self.conn.rollback()
```

```
            print(f"Error With Analytics Of Most Booked Venue:
{str(e)}")
            return None

    def user_activity(self):
        """
        Запит для визначення найактивніших користувачів (Users),
        на основі кількості їхніх бронювань (Booking).
        """
        c = self.conn.cursor()
        try:
            c.execute("""
                SELECT
                    u."user_id",
                    u."first_name",
                    u."last_name",
                    COUNT(b."booking_id") AS total_bookings
                FROM
                    "users" u
                JOIN
                    "booking" b ON u."user_id" = b."user_id"
                GROUP BY
                    u."user_id", u."first_name", u."last_name"
                ORDER BY
                    total_bookings DESC
                LIMIT 5; -- П'ять найактивніших користувачів
            """)

            data = c.fetchall()
            self.conn.commit()
            return data
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Analytics Of User Activity: {str(e)}")
            return None

    def payment_analysis(self):
        """
        Запит для аналізу платежів (Payment) за статусом,
        загальною сумою та кількістю.
        """
        c = self.conn.cursor()
        try:
```

```
            c.execute("""
                SELECT
                    p."payment_status",
                    COUNT(p."payment_id") AS total_payments,
                    SUM(p."amount") AS total_revenue
                FROM
                    "payment" p
                GROUP BY
                    p."payment_status"
                ORDER BY
                    total_revenue DESC;
            """)

            data = c.fetchall()
            self.conn.commit()
            return data
        except Exception as e:
            self.conn.rollback()
            print(f"Error With Analytics Of Payments: {str(e)}")
            return None
```