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
Горбик Д.В.

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).

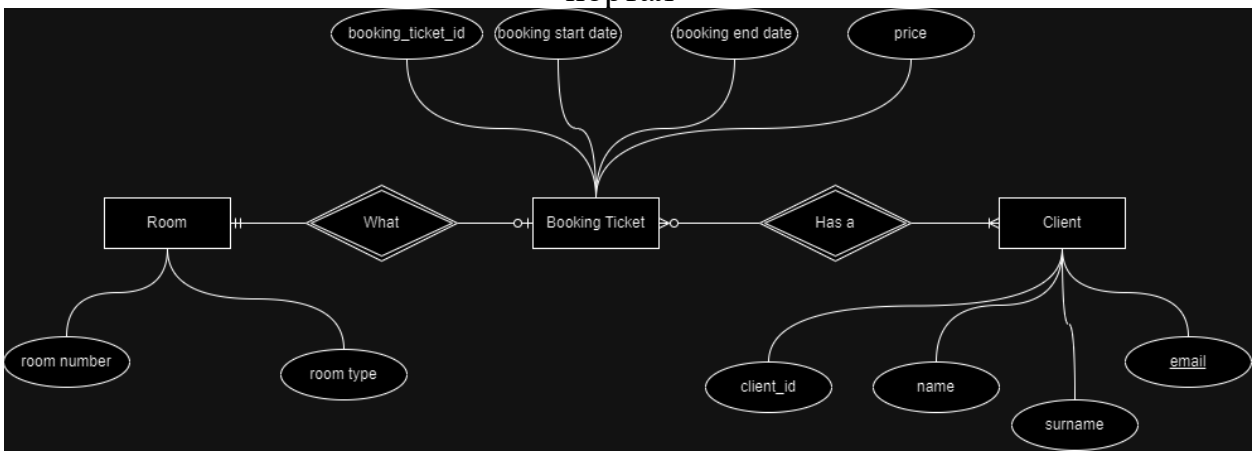
Деталізоване завдання:

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 псевдовипадковими числами. Дані мають бути згенерованими **не мовою програмування, а відповідним SQL-запитом**!

Інформація про модель та структуру бази даних

Рис. 1 - Концептуальна модель предметної області “Готельний бронювальний портал”



Нижче (Рис. 2) наведено логічну модель бази даних:

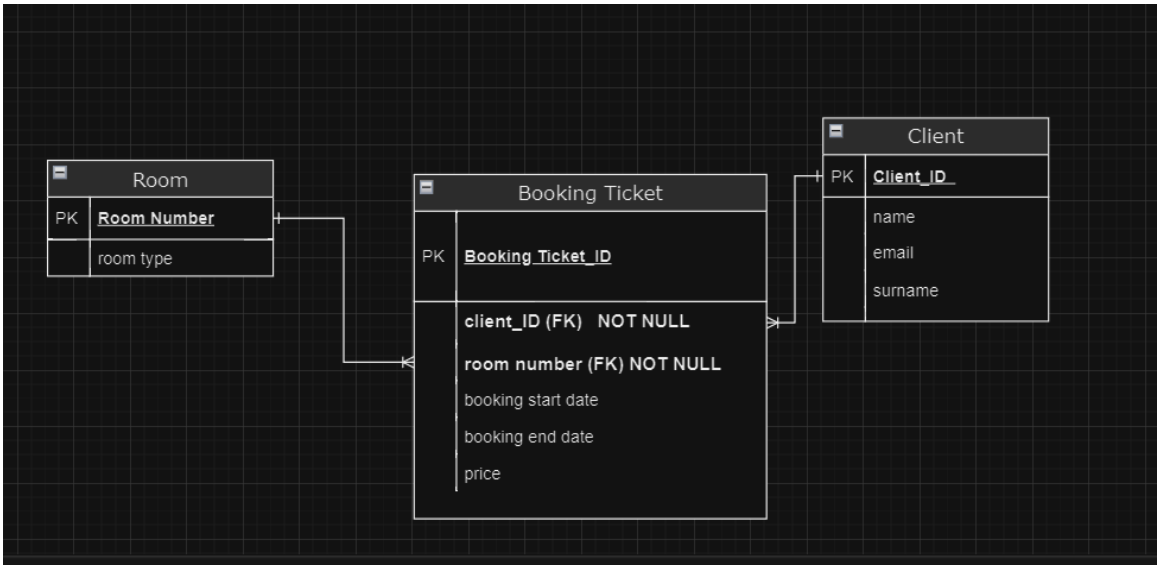


Рис. 2 – Логічна модель бази даних

Зміни у порівнянні з першою лабораторною роботою відсутні.

Середовище розробки та налаштування підключення до бази даних

Для виконання лабораторної роботи використовувалась мова програмування Python .

Для підключення до серверу бази даних PostgreSQL використано модуль «psycopg2».

Опис структури програми

Програма містить 4 основні модулі: **Analytics, Booking, Client, Room,**

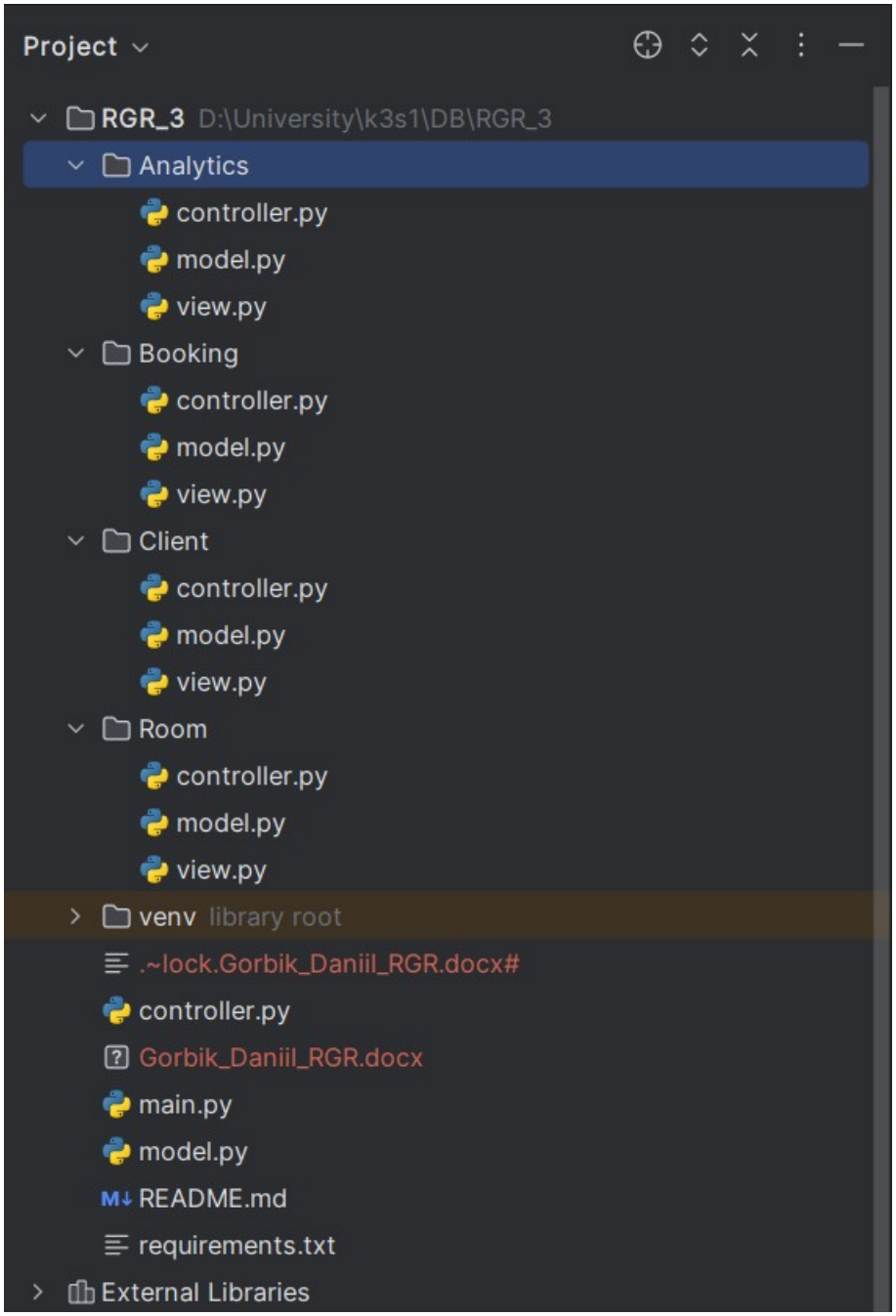


Рис. 3 – Структура програмного коду

Структура меню програми

```
Menu:
1. Add Booking Ticket
2. View Booking Tickets
3. Update Booking Ticket
4. Delete Booking Ticket
5. Add Client
6. View Clients
7. Update Client
8. Delete Client
9. Add Room
10. View Rooms
11. Update Room
12. Delete Room
13. Generate Random Data
14. Truncate All Tables
15. Display Analytics
16. Quit
```

Пункт 1

Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.

Внесення даних

Створення нового Client:

```
Menu:
1. Add Booking Ticket
2. View Booking Tickets
3. Update Booking Ticket
4. Delete Booking Ticket
5. Add Client
6. View Clients
7. Update Client
8. Delete Client
9. Add Room
10. View Rooms
11. Update Room
12. Delete Room
13. Generate Random Data
14. Truncate All Tables
15. Display Analytics
16. Quit
Enter your choice: 5
Enter client id: 6
Enter client name: Daniil
Enter client surname: Gorbik
Enter client email: Daniil@gmail.com
Client added successfully!
```

```
Menu:
1. Add Booking Ticket
2. View Booking Tickets
3. Update Booking Ticket
4. Delete Booking Ticket
5. Add Client
6. View Clients
7. Update Client
8. Delete Client
9. Add Room
10. View Rooms
11. Update Room
12. Delete Room
13. Generate Random Data
14. Truncate All Tables
15. Display Analytics
16. Quit
Enter your choice: 6
Clients:
ID: 1, Name: Emma, Surname: Wilson, Email: 5d639621f1@gmail.com
ID: 2, Name: Emma, Surname: Johnson, Email: bf770dad4f@gmail.com
ID: 3, Name: Michael, Surname: Johnson, Email: 26a9c0dc30@gmail.com
ID: 4, Name: John, Surname: Smith, Email: b59d2c4b39@gmail.com
ID: 5, Name: Emma, Surname: Davis, Email: 2e6d1c1364@gmail.com
ID: 6, Name: Daniil, Surname: Gorbik, Email: Daniil@gmail.com
```

Видалення даних

```
Menu:
1. Add Booking Ticket
2. View Booking Tickets
3. Update Booking Ticket
4. Delete Booking Ticket
5. Add Client
6. View Clients
7. Update Client
8. Delete Client
9. Add Room
10. View Rooms
11. Update Room
12. Delete Room
13. Generate Random Data
14. Truncate All Tables
15. Display Analytics
16. Quit
Enter your choice: 8
Enter client id: 6
Client deleted successfully!
```

```
Menu:
1. Add Booking Ticket
2. View Booking Tickets
3. Update Booking Ticket
4. Delete Booking Ticket
5. Add Client
6. View Clients
7. Update Client
8. Delete Client
9. Add Room
10. View Rooms
11. Update Room
12. Delete Room
13. Generate Random Data
14. Truncate All Tables
15. Display Analytics
16. Quit
Enter your choice: 6
Clients:
ID: 1, Name: Emma, Surname: Wilson, Email: 5d639621f1@gmail.com
ID: 2, Name: Emma, Surname: Johnson, Email: bf770dad4f@gmail.com
ID: 3, Name: Michael, Surname: Johnson, Email: 26a9c0dc30@gmail.com
ID: 4, Name: John, Surname: Smith, Email: b59d2c4b39@gmail.com
ID: 5, Name: Emma, Surname: Davis, Email: 2e6d1c1364@gmail.com
```

Якщо уведено неіснуючий id :

```
Menu:
1. Add Booking Ticket
2. View Booking Tickets
3. Update Booking Ticket
4. Delete Booking Ticket
5. Add Client
6. View Clients
7. Update Client
8. Delete Client
9. Add Room
10. View Rooms
11. Update Room
12. Delete Room
13. Generate Random Data
14. Truncate All Tables
15. Display Analytics
16. Quit
Enter your choice: 8
Enter client id: 1000
Client with the specified ID does not exist.
```

Порушує обмеження зовнішнього ключа:

```
Menu:
1. Add Booking Ticket
2. View Booking Tickets
3. Update Booking Ticket
4. Delete Booking Ticket
5. Add Client
6. View Clients
7. Update Client
8. Delete Client
9. Add Room
10. View Rooms
11. Update Room
12. Delete Room
13. Generate Random Data
14. Truncate All Tables
15. Display Analytics
16. Quit
Enter your choice: 8
Enter client id: 1
Помилка при видаленні клієнта порушує обмеження зовнішнього ключа
```

Помилка при порушенні обмеження зовнішнього ключа виникає тоді, коли спроба вставити або змінити дані в таблиці, яка містить зовнішній ключ, порушує обмеження цього ключа. Обмеження зовнішнього ключа визначає, що значення в стовпці, який посилається на інший стовпець в іншій таблиці, повинно відповідати існуючим значенням в цій іншій таблиці.

Редагування даних

```
Menu:
1. Add Booking Ticket
2. View Booking Tickets
3. Update Booking Ticket
4. Delete Booking Ticket
5. Add Client
6. View Clients
7. Update Client
8. Delete Client
9. Add Room
10. View Rooms
11. Update Room
12. Delete Room
13. Generate Random Data
14. Truncate All Tables
15. Display Analytics
16. Quit
Enter your choice: 7
Enter client id: 1
Enter client name: Daniil
Enter client surname: Gorbik
Enter client email: Daniil@gmail.com
Client updated successfully!
```

```
Menu:
1. Add Booking Ticket
2. View Booking Tickets
3. Update Booking Ticket
4. Delete Booking Ticket
5. Add Client
6. View Clients
7. Update Client
8. Delete Client
9. Add Room
10. View Rooms
11. Update Room
12. Delete Room
13. Generate Random Data
14. Truncate All Tables
15. Display Analytics
16. Quit
Enter your choice: 6
Clients:
ID: 2, Name: Emma, Surname: Johnson, Email: bf770dad4f@gmail.com
ID: 3, Name: Michael, Surname: Johnson, Email: 26a9c0dc30@gmail.com
ID: 4, Name: John, Surname: Smith, Email: b59d2c4b39@gmail.com
ID: 5, Name: Emma, Surname: Davis, Email: 2e6d1c1364@gmail.com
ID: 1, Name: Daniil, Surname: Gorbik, Email: Daniil@gmail.com
```

```
Menu:
1. Add Booking Ticket
2. View Booking Tickets
3. Update Booking Ticket
4. Delete Booking Ticket
5. Add Client
6. View Clients
7. Update Client
8. Delete Client
9. Add Room
10. View Rooms
11. Update Room
12. Delete Room
13. Generate Random Data
14. Truncate All Tables
15. Display Analytics
16. Quit
Enter your choice: 7
Enter client id: 1000
Client with the specified ID does not exist.
```

Пункт 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**!

Витяги деяких рандомізованих рядків з таблиць:

Booking Ticket

```
Booking Tickets:
Booking ID: 1, Client ID: 20, Room Number: 20, Start Date: 2023-03-31, End Date: 2024-09-19, Price: 338
Booking ID: 2, Client ID: 8, Room Number: 12, Start Date: 2023-03-02, End Date: 2024-03-27, Price: 82
Booking ID: 3, Client ID: 6, Room Number: 18, Start Date: 2023-07-14, End Date: 2023-11-23, Price: 201
Booking ID: 4, Client ID: 7, Room Number: 11, Start Date: 2023-09-25, End Date: 2024-01-17, Price: 595
Booking ID: 5, Client ID: 16, Room Number: 6, Start Date: 2023-05-25, End Date: 2024-05-20, Price: 468
Booking ID: 6, Client ID: 2, Room Number: 8, Start Date: 2023-10-04, End Date: 2024-10-15, Price: 212
Booking ID: 7, Client ID: 14, Room Number: 8, Start Date: 2023-10-04, End Date: 2024-03-23, Price: 44
Booking ID: 8, Client ID: 4, Room Number: 19, Start Date: 2023-10-07, End Date: 2024-03-07, Price: 165
Booking ID: 9, Client ID: 7, Room Number: 3, Start Date: 2023-09-11, End Date: 2024-04-01, Price: 556
Booking ID: 10, Client ID: 5, Room Number: 14, Start Date: 2023-05-08, End Date: 2024-02-23, Price: 902
Booking ID: 11, Client ID: 1, Room Number: 11, Start Date: 2023-10-06, End Date: 2023-12-02, Price: 66
Booking ID: 12, Client ID: 10, Room Number: 13, Start Date: 2023-08-30, End Date: 2024-01-05, Price: 166
Booking ID: 13, Client ID: 4, Room Number: 7, Start Date: 2023-05-24, End Date: 2024-05-20, Price: 112
Booking ID: 14, Client ID: 17, Room Number: 2, Start Date: 2023-06-11, End Date: 2024-08-13, Price: 459
Booking ID: 15, Client ID: 11, Room Number: 20, Start Date: 2023-02-15, End Date: 2024-07-02, Price: 966
Booking ID: 16, Client ID: 10, Room Number: 10, Start Date: 2023-03-22, End Date: 2024-06-24, Price: 538
Booking ID: 17, Client ID: 3, Room Number: 12, Start Date: 2023-06-02, End Date: 2024-03-28, Price: 297
Booking ID: 18, Client ID: 10, Room Number: 10, Start Date: 2023-02-13, End Date: 2024-11-16, Price: 614
Booking ID: 19, Client ID: 13, Room Number: 1, Start Date: 2023-04-16, End Date: 2024-08-02, Price: 911
Booking ID: 20, Client ID: 14, Room Number: 9, Start Date: 2023-03-17, End Date: 2024-07-18, Price: 115
```

Client

```
Clients:
ID: 1, Name: John, Surname: Davis, Email: 1ce00bbd5c@gmail.com
ID: 2, Name: Emma, Surname: Brown, Email: 7d2b029369@gmail.com
ID: 3, Name: Emma, Surname: Smith, Email: 64d2a1a68e@gmail.com
ID: 4, Name: John, Surname: Davis, Email: b1438399f2@gmail.com
ID: 5, Name: Alice, Surname: Johnson, Email: cfb859a68f@gmail.com
ID: 6, Name: John, Surname: Johnson, Email: fe144ae338@gmail.com
ID: 7, Name: Emma, Surname: Johnson, Email: 7059320f40@gmail.com
ID: 8, Name: John, Surname: Smith, Email: 52e78f41a2@gmail.com
ID: 9, Name: Michael, Surname: Wilson, Email: b8b2dd9a06@gmail.com
ID: 10, Name: Michael, Surname: Davis, Email: 6edc686ba4@gmail.com
ID: 11, Name: Emma, Surname: Davis, Email: 0924a9914c@gmail.com
ID: 12, Name: John, Surname: Brown, Email: 584f1643e5@gmail.com
ID: 13, Name: Emma, Surname: Brown, Email: 83c7802e87@gmail.com
ID: 14, Name: Alice, Surname: Smith, Email: 3270bd75ef@gmail.com
ID: 15, Name: Bob, Surname: Johnson, Email: 6dec160a22@gmail.com
ID: 16, Name: Alice, Surname: Davis, Email: 00ee00409c@gmail.com
ID: 17, Name: Michael, Surname: Johnson, Email: 59a43cbbe9@gmail.com
ID: 18, Name: John, Surname: Davis, Email: 99e5706ac3@gmail.com
ID: 19, Name: Emma, Surname: Johnson, Email: d1a3a256dc@gmail.com
ID: 20, Name: Alice, Surname: Wilson, Email: d88087f08d@gmail.com
```

Rooms

```
Rooms:
Room Number: 1, Room Type: Suite
Room Number: 2, Room Type: Single
Room Number: 3, Room Type: Suite
Room Number: 4, Room Type: Single
Room Number: 5, Room Type: Suite
Room Number: 6, Room Type: Double
Room Number: 7, Room Type: Suite
Room Number: 8, Room Type: Single
Room Number: 9, Room Type: Suite
Room Number: 10, Room Type: Suite
Room Number: 11, Room Type: Single
Room Number: 12, Room Type: Single
Room Number: 13, Room Type: Single
Room Number: 14, Room Type: Suite
Room Number: 15, Room Type: Suite
Room Number: 16, Room Type: Suite
Room Number: 17, Room Type: Single
Room Number: 18, Room Type: Double
Room Number: 19, Room Type: Suite
Room Number: 20, Room Type: Single
```

Очищення всіх таблиць

```
Menu:
1. Add Booking Ticket
2. View Booking Tickets
3. Update Booking Ticket
4. Delete Booking Ticket
5. Add Client
6. View Clients
7. Update Client
8. Delete Client
9. Add Room
10. View Rooms
11. Update Room
12. Delete Room
13. Generate Random Data
14. Truncate All Tables
15. Display Analytics
16. Quit
Enter your choice: 14
Are you sure? Type Yes or No: Yes
All booking tickets truncated successfully!
All client data truncated successfully!
All rooms data truncated successfully!
```

SQL запити рандомізованого заповнення:

```
INSERT INTO booking_ticket (booking_id, client_id, room_number, booking_start_date, booking_end_date, price)
    SELECT
        nextval('booking_id_seq'),
        floor(random() * (SELECT max(client_id) FROM client) + 1),
        floor(random() * (SELECT max(room_number) FROM room) + 1),
        '2023-01-01'::date + floor(random() * (date '2023-11-05' - date '2023-01-01')) * interval '1 day',
        '2023-11-05'::date + floor(random() * (date '2025-01-01' - date '2023-11-05')) * interval '1 day',
        random() * 1000
    FROM generate_series(1, %s)
    """, (number_of_operations,))
```

```
        INSERT INTO client (client_id, name, surname, email)
        SELECT
            nextval('client_id_seq'),
            (array['John', 'Alice', 'Bob', 'Emma', 'Michael'])[floor(random() * 5) + 1],
            (array['Smith', 'Johnson', 'Brown', 'Davis', 'Wilson'])[floor(random() * 5) + 1],
            (substr(md5(random()::text), 1, 10) || '@gmail.com')
        FROM generate_series(1, %s);
        """, (number_of_operations,))
```

```
            INSERT INTO client (client_id, name, surname, email)
            SELECT
                nextval('client_id_seq'),
                (array['John', 'Alice', 'Bob', 'Emma', 'Michael'])[floor(random() * 5) + 1],
                (array['Smith', 'Johnson', 'Brown', 'Davis', 'Wilson'])[floor(random() * 5) + 1],
                (substr(md5(random()::text), 1, 10) || '@gmail.com')
            FROM generate_series(1, %s);
            """, (number_of_operations,))
```

Пункт 3

Для реалізації пошуку необхідно підготувати 3 запити, що включають дані з декількох таблиць і фільтрують (WHERE) та групують (GROUP BY) рядки за 3-4 атрибутами цих таблиць. Забезпечити можливість уведення конкретних значень констант для фільтрації з клавіатури користувачем. Крім того, після виведення даних необхідно вивести час виконання запиту у мілісекундах. Перевірити швидкодію роботи запитів на попередньо згенерованих даних.

Було підготовлено три SQL запити:

- найбільша зайнятість номерів
- кількість замовлень за останні два тижні
- аналітика найбільш активних клієнтів

найбільша зайнятість номерів:

```
c.execute("""
        SELECT room_number, COUNT(*) AS occupancy_count
        FROM booking_ticket
        GROUP BY room_number
        HAVING COUNT(*) >= 3
        ORDER BY occupancy_count DESC;
    """)
```

Результат:

```
Зайнятість номерів:
Номер 24: 4 бронювань
Номер 77: 4 бронювань
Номер 71: 4 бронювань
Номер 93: 3 бронювань
Номер 7: 3 бронювань
Номер 61: 3 бронювань
Номер 58: 3 бронювань
Номер 70: 3 бронювань
Номер 39: 3 бронювань
```

Налаштування фільтрування :

- к-ть бронювань > 3

кількість замовлень за останні два тижні :

```
            SELECT
                room_number,
                COUNT(*) AS orders_count
            FROM
                booking_ticket
            WHERE
                booking_start_date >= current_date - interval '14 days'
            GROUP BY
                room_number
            ORDER BY
            orders_count DESC;

            """)
```

Результат:

```
Кількість замовлень на номери:
Номер 71: 1 замовлень
Номер 60: 1 замовлень
Номер 55: 1 замовлень
```

аналітика найбільш активних клієнтів:

```
c.execute("""SELECT
                client.client_id,
                client.name,
                client.surname,
                client.email,
                COUNT(booking_ticket.booking_id) AS booking_count
            FROM
                client
            JOIN
                booking_ticket ON client.client_id = booking_ticket.client_id
            GROUP BY
                client.client_id, client.name, client.surname, client.email
            HAVING
                COUNT(booking_ticket.booking_id) > 3
            ORDER BY
                booking_count DESC;
            """)
```

Результат:

```
Аналітика клієнтів:
Клієнт ID: 63, Ім'я: Bob, Прізвище: Johnson, Email: 69bbb267db@gmail.com, Кількість бронювань: 4
Клієнт ID: 88, Ім'я: Michael, Прізвище: Smith, Email: 2852c9251f@gmail.com, Кількість бронювань: 4
Клієнт ID: 43, Ім'я: Emma, Прізвище: Smith, Email: 3ddefb3b36@gmail.com, Кількість бронювань: 4
Клієнт ID: 8, Ім'я: Alice, Прізвище: Wilson, Email: 38a20d1100@gmail.com, Кількість бронювань: 4

Мери:
```

Налаштування фільтрування :

- к-ть бронювань > 3

Код програмного модулю model

`Analytics/model.py`

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

    def room_occupancy(self):
        c = self.conn.cursor()
        try:
            c.execute("""
                    SELECT room_number, COUNT(*) AS occupancy_count
                    FROM booking_ticket
                    GROUP BY room_number
                    HAVING COUNT(*) >= 3
                    ORDER BY occupancy_count DESC;
                """)

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

            self.conn.commit()
            return room_occupancy_data
        except Exception as e:
            self.conn.rollback()
            print(f"Error in room occupancy analytics: {str(e)}")
            return None



    def number_of_orders(self):
        c = self.conn.cursor()
        try:
            c.execute("""
                        SELECT
                            room_number,
                            COUNT(*) AS orders_count
                        FROM
                            booking_ticket
                        WHERE
                            booking_start_date >= current_date - interval '14 days'
                        GROUP BY
                            room_number
                        ORDER BY
                        orders_count DESC;

                        """)

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

            self.conn.commit()
            return number_of_orders_data
        except Exception as e:
            self.conn.rollback()
            print(f"Error in analyzing the number of orders: {str(e)}")
            return None
```

```
    def client_analytics(self):
        c = self.conn.cursor()
        try:
            c.execute("""SELECT
                        client.client_id,
                        client.name,
                                client.surname,
                                client.email,
                        COUNT(booking_ticket.booking_id) AS booking_count
                    FROM
                        client
                    JOIN
                        booking_ticket ON client.client_id = booking_ticket.client_id
                    GROUP BY
                        client.client_id, client.name, client.surname, client.email
                    HAVING
                        COUNT(booking_ticket.booking_id) > 3
                    ORDER BY
                        booking_count DESC;
                    """)

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

            self.conn.commit()
            return number_of_orders_data
        except Exception as e:
            self.conn.rollback()
            print(f"Error in customer analytics: {str(e)}")
            return None
```

Booking/model.py

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

    def add_booking_ticket(self, booking_id, client_id, room_number, booking_start_date, booking_end_date, price):
        c = self.conn.cursor()
        try:
            # Check if client_id and room_number match parent tables
            c.execute('SELECT 1 FROM client WHERE client_id = %s', (client_id,))
            client_exists = c.fetchone()

            c.execute('SELECT 1 FROM room WHERE room_number = %s', (room_number,))
            room_exists = c.fetchone()

            if not client_exists or not room_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 booking_ticket (booking_id, client_id, room_number, '
```

```
                'booking_start_date, booking_end_date, price) VALUES (%s, %s, %s, %s, %s, %s)',
                (booking_id, client_id, room_number, booking_start_date, booking_end_date, price))
            self.conn.commit()
            return True
        except Exception as e:
            self.conn.rollback()
            print(f"Error when adding a booking: {str(e)}")
            return False

    def get_all_booking_tickets(self):
        c = self.conn.cursor()
        c.execute('SELECT * FROM booking_ticket')
        return c.fetchall()

    def update_booking_ticket(self, booking_id, client_id, room_number, booking_start_date, booking_end_date,
price):
        c = self.conn.cursor()
        try:
            # Attempting to update a record
            c.execute('UPDATE booking_ticket SET client_id=%s, room_number=%s, booking_start_date=%s, '
                      'booking_end_date=%s, price=%s WHERE booking_id=%s',
                      (client_id, room_number, booking_start_date, booking_end_date, price, booking_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 when updating a reservation: {str(e)}")
            return False   # Returns False if insertion fails

    def delete_booking_ticket(self, booking_id):
        c = self.conn.cursor()
        try:
            # Attempting to update a record
            c.execute('DELETE FROM booking_ticket WHERE booking_id=%s', (booking_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 when deleting a reservation: {str(e)}")
            return False   # Returns False if insertion fails

    def check_booking_existence(self, booking_id):
        c = self.conn.cursor()
        c.execute("SELECT 1 FROM booking_ticket WHERE booking_id = %s", (booking_id,))
        return bool(c.fetchone())

    def create_booking_sequence(self):
        c = self.conn.cursor()
        c.execute("""
            DO $$
            BEGIN
                IF NOT EXISTS (SELECT 1 FROM pg_sequences WHERE schemaname = 'public' AND sequencename
= 'booking_id_seq') THEN
                    CREATE SEQUENCE booking_id_seq;
                ELSE
```

```
            DROP SEQUENCE booking_id_seq;
            CREATE SEQUENCE booking_id_seq;
        END IF;
    END $$;
    """)
    self.conn.commit()

def generate_rand_booking_ticket_data(self, number_of_operations):
    c = self.conn.cursor()
    try:
        c.execute("""
        INSERT INTO booking_ticket (booking_id, client_id, room_number, booking_start_date,
booking_end_date, price)
            SELECT
                nextval('booking_id_seq'),
                floor(random() * (SELECT max(client_id) FROM client) + 1),
                floor(random() * (SELECT max(room_number) FROM room) + 1),
                '2023-01-01'::date + floor(random() * (date '2023-11-05' - date '2023-01-01')) * interval '1 day',
                        '2023-11-05'::date + floor(random() * (date '2025-01-01' - date '2023-11-05')) * interval '1
day',
                        random() * 1000
                        FROM generate_series(1, %s)
            """, (number_of_operations,))

        self.conn.commit()
        return True
    except Exception as e:
        self.conn.rollback()
        print(f"Error while generating booking tickets: {str(e)}")
        return False


def truncate_booking_table(self):
    c = self.conn.cursor()
    try:
        # Insert data
        c.execute("""DELETE FROM booking_ticket""")
        self.conn.commit()
        return True  # Returns True if the insertion was successful
    except Exception as e:
        self.conn.rollback()
        print(f"Error when adding a client: {str(e)}")
        return False  # Returns False if insertion fails
```

Client/model.py

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

    def add_client(self, client_id, name, surname, email):
```

```
        c = self.conn.cursor()
        try:
            c.execute('INSERT INTO client (client_id, name, surname, email) VALUES (%s, %s, %s, %s)',
                      (client_id, name, surname, email))
            self.conn.commit()
            return True  # Returns True if the update was successful
        except Exception as e:
            self.conn.rollback()
            print(f"Error when adding a client: {str(e)}")
            return False  # Returns False if insertion fails

    def get_all_clients(self):
        c = self.conn.cursor()
        c.execute('SELECT * FROM client')
        return c.fetchall()

    def update_client(self, client_id, name, surname, email):
        c = self.conn.cursor()
        try:
            c.execute('UPDATE client SET name=%s, surname=%s, email=%s WHERE client_id=%s',
                      (name, surname, email, client_id))
            self.conn.commit()
            return True  # Returns True if the update was successful
        except Exception as e:
            self.conn.rollback()
            print(f"Error when updating the client: {str(e)}")
            return False   # Returns False if insertion fails

    def delete_client(self, client_id):
        c = self.conn.cursor()
        try:
            c.execute('DELETE FROM client WHERE client_id=%s', (client_id,))
            self.conn.commit()
            return True  # Returns True if the update was successful
        except Exception as e:
            self.conn.rollback()
            print(f"An error when deleting a client breaks the foreign key restriction: {str(e)}")
            return False   # Returns False if insertion fails

    def check_client_existence(self, client_id):
        c = self.conn.cursor()
        c.execute("SELECT 1 FROM client WHERE client_id = %s", (client_id,))
        return bool(c.fetchone())

    def create_client_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 =
'client_id_seq') THEN
                -- Якщо послідовності не існує, створюємо її
                CREATE SEQUENCE client_id_seq;
            ELSE
                -- Якщо послідовність існує, видаляємо і створюємо нову
                DROP SEQUENCE client_id_seq;
```

```
            CREATE SEQUENCE client_id_seq;
        END IF;
    END $$;
    """)
    self.conn.commit()

  def generate_rand_client_data(self, number_of_operations):
    c = self.conn.cursor()
    try:
        # Insert data
        c.execute("""
        INSERT INTO client (client_id, name, surname, email)
        SELECT
            nextval('client_id_seq'),
            (array['John', 'Alice', 'Bob', 'Emma', 'Michael'])[floor(random() * 5) + 1],
            (array['Smith', 'Johnson', 'Brown', 'Davis', 'Wilson'])[floor(random() * 5) + 1],
            (substr(md5(random()::text), 1, 10) || '@gmail.com')
        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 when adding a client: {str(e)}")
        return False   # Returns False if insertion fails



  def truncate_client_table(self):
    c = self.conn.cursor()
    try:
        # Insert data
        c.execute("""DELETE FROM client""")
        self.conn.commit()
        return True  # Returns True if the update was successful
    except Exception as e:
        self.conn.rollback()
        print(f"Error when adding a client: {str(e)}")
        return False   # Returns False if insertion fails
```

Room/model.py

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

  def add_room(self, room_number, room_type):
    c = self.conn.cursor()
    try:
        c.execute('INSERT INTO room (room_number ,room_type) VALUES (%s, %s)', (room_number,
room_type,))
        self.conn.commit()
        return True  # Returns True if the update was successful
    except Exception as e:
```

```
            self.conn.rollback()
            print(f"Error when adding a room: {str(e)}")
            return False   # Returns False if insertion fails

    def get_all_rooms(self):
        c = self.conn.cursor()
        c.execute('SELECT * FROM room')
        return c.fetchall()

    def update_room(self, room_number, room_type):
        c = self.conn.cursor()
        try:
            c.execute('UPDATE room SET room_type=%s WHERE room_number=%s', (room_type, room_number))
            self.conn.commit()
            return True  # Returns True if the update was successful
        except Exception as e:
            self.conn.rollback()
            print(f"Error when updating a room: {str(e)}")
            return False   # Returns False if insertion fails

    def delete_room(self, room_number):
        c = self.conn.cursor()
        try:
            c.execute('DELETE FROM room WHERE room_number=%s', (room_number,))
            self.conn.commit()
            return True  # Returns True if the update was successful
        except Exception as e:
            self.conn.rollback()
            print(f"Error when deleting a room: {str(e)}")
            return False  # Returns False if insertion fails

    def check_room_existence(self, room_number):
        c = self.conn.cursor()
        c.execute('SELECT 1 FROM room WHERE room_number = %s', (room_number,))
        return c.fetchone() is not None

    def create_room_sequence(self):
        # Check for the existence of a sequence
        c = self.conn.cursor()
        c.execute("""
        DO $$
        BEGIN
            IF NOT EXISTS (SELECT 1 FROM pg_sequences WHERE schemaname = 'public' AND sequencename =
'room_number_seq') THEN
                -- Якщо послідовності не існує, створюємо її
                CREATE SEQUENCE room_number_seq;
            ELSE
                -- Якщо послідовність існує, видаляємо і створюємо нову
                DROP SEQUENCE room_number_seq;
                CREATE SEQUENCE room_number_seq;
            END IF;
        END $$;
        """)
        self.conn.commit()
    def generate_rand_room_data(self, number_of_operations):
        c = self.conn.cursor()
        try:
```

```
            # Insert data
            c.execute("""
            INSERT INTO room (room_number, room_type)
            SELECT
                nextval('room_number_seq'),
                (array['Single', 'Double', 'Suite'])[floor(random() * 3) + 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 when adding a room: {str(e)}")
            return False   # Returns False if insertion fails

    def truncate_room_table(self):
        c = self.conn.cursor()
        try:
            # Insert data
            c.execute("""DELETE FROM room""")
            self.conn.commit()
            return True  # Returns True if the update was successful
        except Exception as e:
            self.conn.rollback()
            print(f"Error when adding a client: {str(e)}")
            return False   # Returns False if insertion fails
```

model.py

```
import psycopg2


class Model:
    def __init__(self):
        self.conn = psycopg2.connect(
            dbname='postgres',
            user='postgres',
            password='1',
            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_ticket')")
        booking_ticket_table_exists = c.fetchone()[0]

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

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

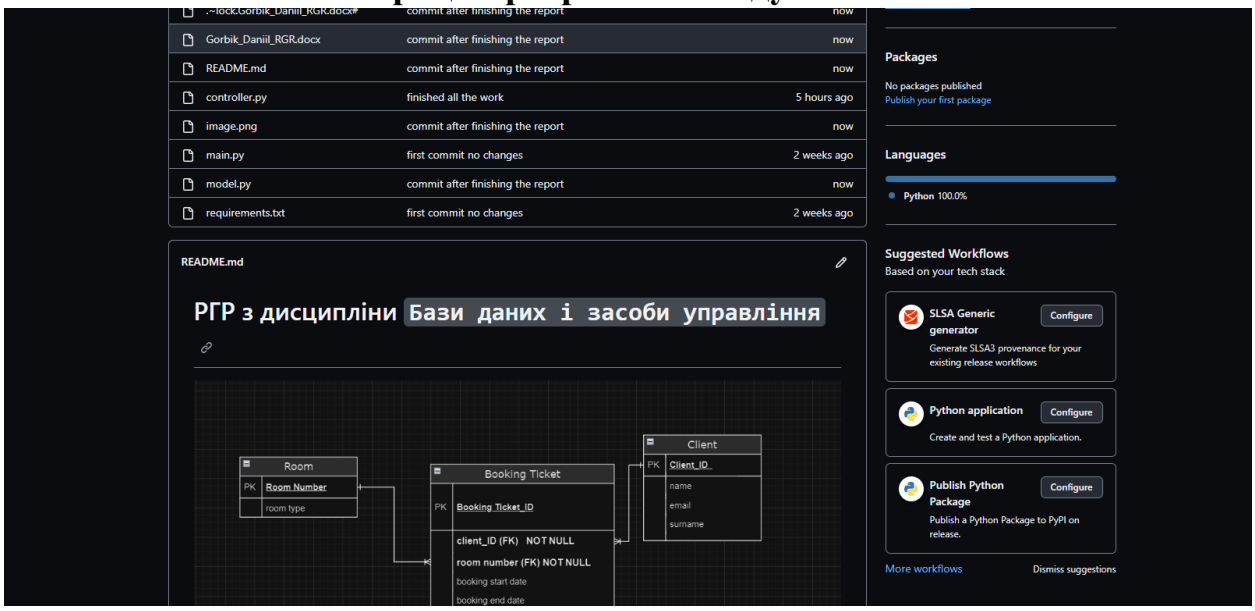
```
    if not booking_ticket_table_exists:
        c.execute('''
                CREATE TABLE booking_ticket (
                    booking_id SERIAL PRIMARY KEY,
                    client_id INTEGER NOT NULL,
                    room_number INTEGER NOT NULL,
                    booking_start_date DATE NOT NULL,
                    booking_end_date DATE NOT NULL,
                    price DECIMAL(10, 2) NOT NULL
                )
            ''')
    if not client_table_exists:
        c.execute('''
                CREATE TABLE client (
                    client_id SERIAL PRIMARY KEY,
                    name TEXT NOT NULL,
                    surname TEXT NOT NULL,
                    email TEXT
                )
            ''')
    if not room_table_exists:
        c.execute('''
                CREATE TABLE room (
                    room_number SERIAL PRIMARY KEY,
                    room_type TEXT NOT NULL
                )
            ''')

    self.conn.commit()
```

Короткий опис функцій

Програма ділиться на 4 папки і головні файли model, controller, main.
Файли:

1. Analytics – відповідає за 3 пункт РГР
2. Booking – відповідає за таблицю booking
3. Client – відповідає за таблицю client
4. Room – відповідає за таблицю room

Ілюстрації програмного коду на Github



Посилання на репозиторій:

https://github.com/Dan-live/c3s1_RGR_DB/tree/main