

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-21

Кузнецов Д. С.

Перевірив:

Павловский В. І.

Всі пункти (1-4) деталізованого завдання виконані

Київ – 2024

**Мета:** здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

## **Виконання роботи**

Нижче наведені сутності предметної області «Онлайн-платформа для здачі та оренди нерухомості» та зв'язки між ними.

### **Опис сутностей**

Для побудови бази даних обраної області, були виділені такі сутності:

#### **1. Користувач (Users)**

**Атрибути:** ідентифікатор користувача, ім'я, електронна пошта, роль (орендодавець, орендар).

**Призначення:** збереження даних користувачів.

#### **2. Оголошення оренди (Rental)**

**Атрибути:** ідентифікатор оголошення, назва, опис, ціна, ідентифікатор користувача.

**Призначення:** збереження даних щодо оголошень оренди.

#### **3. Бронювання (Reservation)**

**Атрибути:** ідентифікатор броні, дата заселення, дата виселення, ідентифікатор користувача, ідентифікатор оголошення.

**Призначення:** збереження даних щодо орендованих квартир.

#### **4. Відгуки (Reviews)**

**Атрибути:** ідентифікатор відгуку, рейтинг, коментар, ідентифікатор користувача, ідентифікатор оголошення.

**Призначення:** збереження даних щодо рейтингу оголошень та відгуків.

## **Опис зв'язків між сутностями**

Зв'язок Користувач - Оголошення оренди є зв'язком 1:N. Один Користувач може публікувати багато оголошень, але одне оголошення може бути створене лише одним користувачем.

Зв'язок Користувач - Бронювання є зв'язком N:M. Один Користувач може здійснити багато бронювань, і нерухомість може бронюватися багатьма Користувачами на різні дати.

Зв'язок Оголошення оренди - Бронювання є зв'язком 1:N. Одне оголошення може мати багато бронювань, але всі бронювання одної нерухомості здійснюються з одного оголошення.

Зв'язок Оголошення оренди - Відгуки є зв'язком 1:M. Одне оголошення може мати багато відгуків, але кожен відгук пов'язаний з одним оголошенням.

Зв'язок Користувач - Відгуки є зв'язком 1:N. Один Користувач може написати багато відгуків, але кожен відгук закріплений за одним користувачем.

Графічне подання концептуальної моделі «Сутність-зв'язок» зображено на рисунку 1.

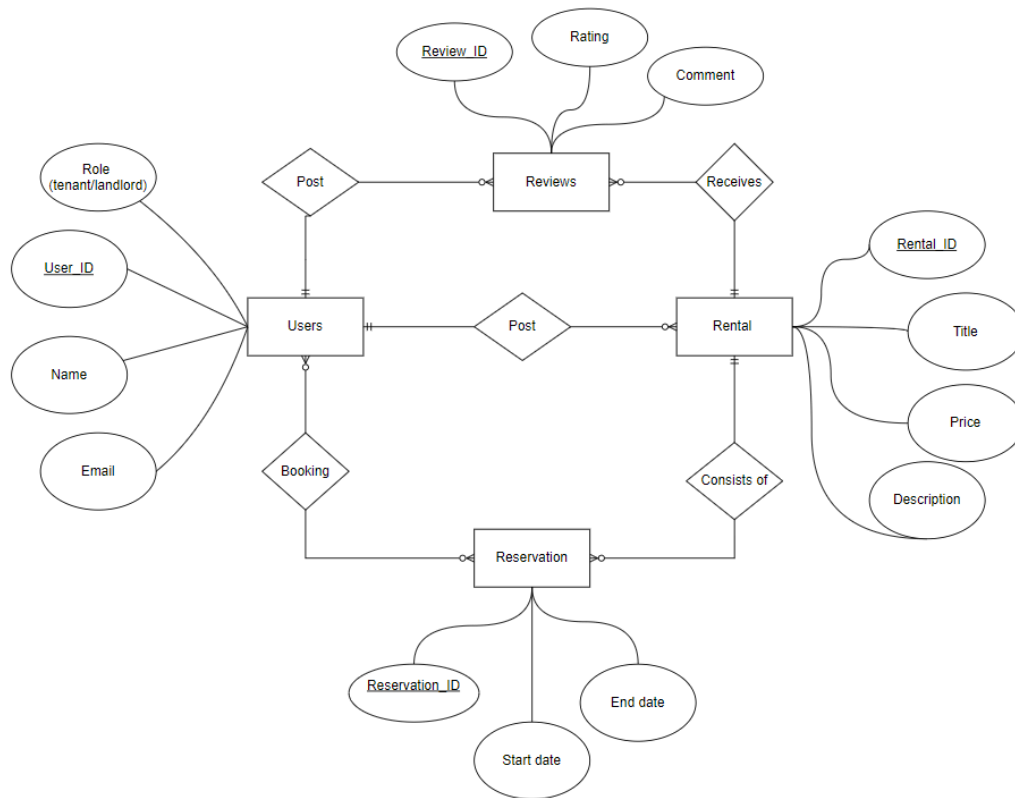


Рисунок 1 – ER-діаграма, побудована за нотацією Crow's Foot

Графічне подання логічної моделі «Сутність-зв'язок» зображено на рисунку 2.

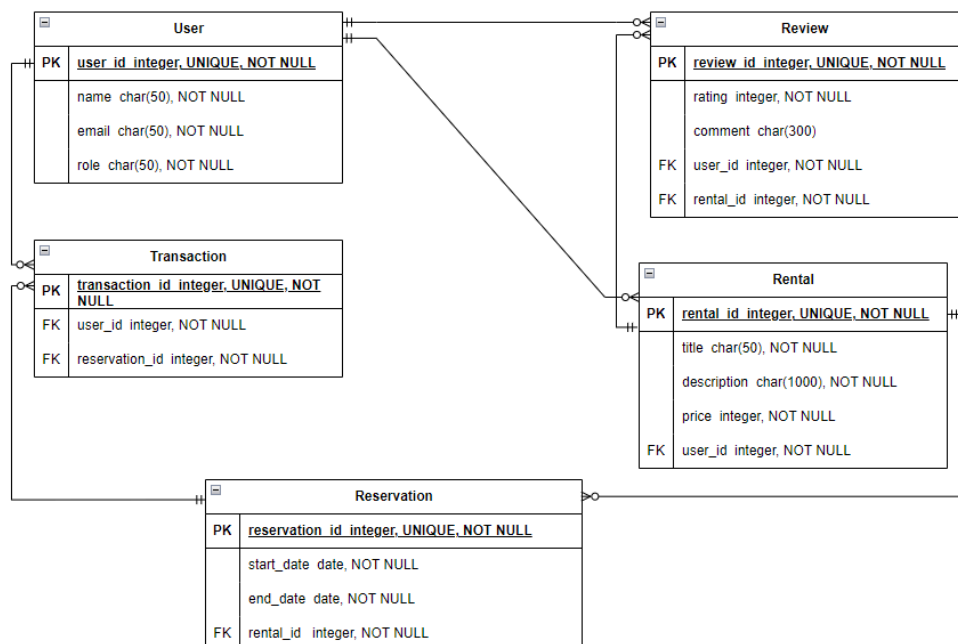


Рисунок 2 – Схема бази даних

## Середовище та компоненти розробки

Середовище для відлагодження SQL-запитів до бази даних – *PgAdmin4*.

Мова програмування – *Python*.

Середовище розробки програмного забезпечення – *PyCharm Community Edition*.

Розроблено консольний інтерфейс користувача.

В ході реалізації було використано такі бібліотеки мови програмування Python:

- *psycopg2* - для взаємодії з базою даних PostgreSQL.
- *time* - для вимірювання часу роботи деяких функцій.
- *sys* - для завершення роботи програми.

В якості шаблону проектування було обрано MVC (Model-View-Controller).

Model – це клас, що відображає логіку роботи з даними та виконує всі операції, такі як додавання, оновлення, видалення, перегляд даних.

View – це клас, через який визначає інтерфейс для взаємодії з користувачем.

Controller – це клас, який відповідає за зв'язок між користувачем і системою. Викликає відповідні дії з Model або View, в залежності від отриманих даних користувача.

## Структура та опис програми

На рисунку 3 показано структуру розробленої програми за шаблоном MVC:

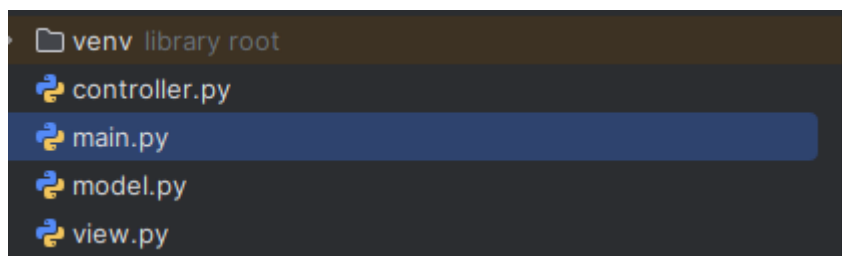


Рисунок 3 – Структура програми

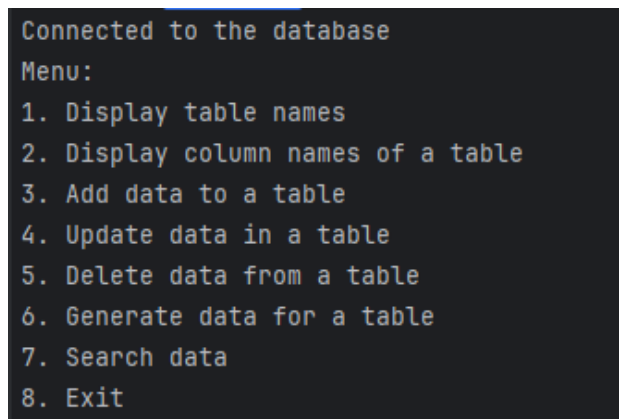
main.py - відбувається виклик контролера.

model.py - містить клас моделі, який відповідає за підключення до бази даних і виконанням низькорівневих запитів.

controller.py - містить інтерфейс взаємодії з користувачем, обробку запитів, та викликає відповідні дії з Model або View.

view.py - містить клас, який відповідає за відображення взаємодії користувача з інтерфейсом, його дії, введені дані та обрані функції.

На рисунку 4 зображено меню користувача, яке складається з семи пунктів.



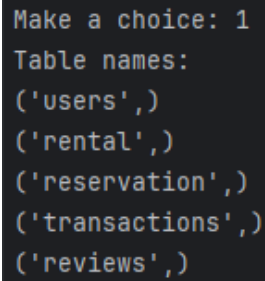
```
Connected to the database
Menu:
1. Display table names
2. Display column names of a table
3. Add data to a table
4. Update data in a table
5. Delete data from a table
6. Generate data for a table
7. Search data
8. Exit
```

Рисунок 4 – Структура меню користувача

Це всі функції для виконання запитів користувача, описані в нашому модулі “Model” - model.py. Функція `__init__` виконує підключення до бази даних за заданими обліковими даними. Детальніше про кожну функцію описано нижче.

Функціональності кожного пункту меню користувача:

1. *Display table names* - виводить на екран назви всіх таблиць бази даних.



```
Make a choice: 1
Table names:
('users',)
('rental',)
('reservation',)
('transactions',)
('reviews',)
```

Рисунок 5 - Вивід на екран назв всіх таблиць

Функція отримання назв всіх таблиць бази даних:

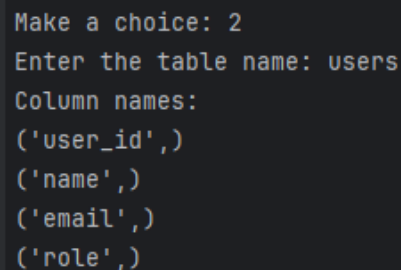
```
def get_all_tables(self):

    c = self.conn.cursor()

    c.execute("SELECT table_name FROM information_schema.tables WHERE
table_schema = 'public'")

    return c.fetchall()
```

2. *Display column names of a table* - виводить на екран всі стовпці обраної користувачем таблиці.



```
Make a choice: 2
Enter the table name: users
Column names:
('user_id',)
('name',)
('email',)
('role',)
```

Рисунок 6 - Вивід на екран назв всіх стовпців

Функція отримання назв всіх стовпців обраної таблиці:

```
def get_all_columns(self, table_name):

    c = self.conn.cursor()

    c.execute("SELECT column_name FROM information_schema.columns WHERE table_name =
%s", (table_name,))

    return c.fetchall()
```

3. *Add data to a table* - дозволяє користувачу додати запис до обраної ним таблиці.

```
Make a choice: 3
Enter the table name: users
Enter column names (space-separated): user_id name email role
Enter corresponding values (space-separated): 2323 lukas luk@gmail.com tenant
Data added successfully!
```

Рисунок 7 - Додавання запису до таблиці

Функція додавання даних до обраної таблиці:

```
def add_data(self, table_name, columns, val):

    c = self.conn.cursor()

    columns_str = ', '.join(columns)

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

    identifier_column = f'{table_name[:-1]}_id' if table_name.endswith('s') else f'{table_name}_id'

    c.execute(f'SELECT "{identifier_column}" FROM "public"."{table_name}"')

    existing_identifiers = [item[0] for item in c.fetchall()]

    if int(val[columns.index(identifier_column)]) in existing_identifiers:

        return 2

    external_keys = [col for col in columns if col.endswith('_id') and col != identifier_column]

    for key_column in external_keys:

        referenced_table = key_column[:-3] + 's'

        c.execute(f'SELECT "{key_column}" FROM "public"."{referenced_table}"')

        if int(val[columns.index(key_column)]) not in [item[0] for item in c.fetchall()]:

            return 3
```



```

        c.execute(f'INSERT INTO "public"."{table_name}" ({columns_str}) VALUES
        ({placeholders})', val)

        self.conn.commit()

        return 1

```

4. *Update data in a table* - дозволяє оновити дані обраного рядка та стовпця обраної користувачем таблиці

```

11 exit()
Make a choice: 4
Enter the table name: users
Enter the name of the column to update: role
Enter the ID of the row to update: 2323
Enter the new value: landlord
Data updated successfully!

```

Рисунок 8 - Оновлення даних таблиці

Функція оновлення даних до обраної таблиці:

```

def update_data(self, table_name, column, id, new_value):

    c = self.conn.cursor()

    identifier_column = f'{table_name[:-1]}_id' if table_name.endswith('s') else
    f'{table_name}_id'

    if column.endswith('_id') and column != identifier_column:

        referenced_table = column[:-3] + 's'

        c.execute(f'SELECT "{column}" FROM "public"."{referenced_table}"')

        if int(new_value) not in [item[0] for item in c.fetchall()]:

            return 3

    c.execute(f'UPDATE "public"."{table_name}" SET "{column}" = %s WHERE
    "{identifier_column}" = %s',

            (new_value, id))

    self.conn.commit()

    return 1

```

5. *Delete data from a table* - дозволяє видалити рядок обраної користувачем таблиці

```
Make a choice: 5
Enter the table name: users
Enter the ID of the row to delete: 2323
Row deleted successfully!
```

Рисунок 9 - Видалення даних таблиці

Функція видалення даних до обраної таблиці:

```
def delete_data(self, table_name, id):

    c = self.conn.cursor()

    identifier_column = f'{table_name[:-1]}_id' if table_name.endswith('s') else
f'{table_name}_id'

    c.execute("SELECT table_name FROM information_schema.tables WHERE table_schema =
'public' AND table_type = "

               "'BASE TABLE'")

    tables = [item[0] for item in c.fetchall()]

    for current_table in tables:

        if current_table == table_name:

            continue

        c.execute("SELECT column_name FROM information_schema.columns WHERE
table_name = %s", (current_table,))

        if identifier_column in [col[0] for col in c.fetchall()]:

            c.execute(f'SELECT "{identifier_column}" FROM
"public"."{current_table}"')

            if id in [item[0] for item in c.fetchall()]:
```

```

        return 0

    c.execute(f'DELETE FROM "public"."{table_name}" WHERE "{identifier_column}" = %s', (id,))

    self.conn.commit()

    return 1

```

6. *Generate data for a table* - генерує псевдовипадкові рядки для обраної користувачем таблиці

```

Make a choice: 6
Enter the table name: users
Enter the number of rows to generate: 100000
Data for table users has been generated successfully

```

Рисунок 10 - Генерація даних таблиці

Функція генерації псевдовипадкових даних до обраної таблиці:

```

def generate_data(self, table_name, count):

    c = self.conn.cursor()

    c.execute(f"SELECT column_name, data_type FROM information_schema.columns WHERE table_name = %s", (table_name,))

    columns_info = c.fetchall()

    id_column = f'{table_name[:-1]}_id'

    for i in range(count):

        insert_query = f'INSERT INTO "public"."{table_name}" ('

        select_subquery = ""

        for column_info in columns_info:

            column_name = column_info[0]

            column_type = column_info[1]

            if column_name == id_column:

```

```

        c.execute(f'SELECT max("{id_column}") FROM "public"."{table_name}")

        max_id = c.fetchone()[0] or 0

        select_subquery += f'{max_id + 1},'

    elif column_name == 'role':

        select_subquery += "(CASE WHEN RANDOM() < 0.5 THEN 'tenant' ELSE
'landlord' END),"

    elif column_name == 'email':

        select_subquery += f'"user{i}@example.com",'

    elif column_name.endswith('_id'):

        related_table_name = column_name[:-3].capitalize()

        c.execute(

            f'SELECT {related_table_name.lower()}_id FROM
"public"."{related_table_name}" ORDER BY RANDOM() LIMIT 1')

        related_id = c.fetchone()[0]

        select_subquery += f'{related_id},'

    elif column_type == 'integer':

        select_subquery += f'trunc(random()*100)::INT,'

    elif column_type == 'character varying':

        select_subquery += f'"Text {column_name} {i}',"

    elif column_type == 'date':

        select_subquery += "'2024-01-01',"

    elif column_type == 'timestamp with time zone':

        select_subquery += "'2024-01-01 00:00:00+03',"

    else:

        continue

    insert_query += f'"{column_name}",'

    insert_query = insert_query.rstrip(',') + f') VALUES
({select_subquery[:-1]})'

    c.execute(insert_query)

```

```
self.conn.commit()
```

7. *Search data* - дозволяє здійснювати пошук по заданим параметрам.

```
Make a choice: 7

Enter search parameters:
Minimum price: 10
Maximum price: 100000
Name (LIKE pattern): D
Email (LIKE pattern): gmail.com
Title (LIKE pattern): квартира
Query executed in: 2.24 ms

Search results:
(9, 'Dmytro', 'dmitr@gmail.com', 'landlord', 11, 9, 'квартира', 'Вул. Київська, 12, Київ', Decimal('50000.00'))
```

Рисунок 11 - Пошук даних

Функція пошуку даних за заданими параметрами:

```
def search_data(self, table1, table2, filter_conditions):

    c = self.conn.cursor()

    query = f"""

        SELECT *

        FROM "public"."{table1}" t1

        JOIN "public"."{table2}" t2 ON t1.user_id = t2.user_id

        WHERE 1=1

    """

    params = []

    if 'price_min' in filter_conditions and 'price_max' in filter_conditions:

        query += " AND t2.price BETWEEN %s AND %s"

        params.extend([filter_conditions['price_min'],
            filter_conditions['price_max']])
```

```

if 'name' in filter_conditions:

    query += " AND t1.name LIKE %s"

    params.append(f"%{filter_conditions['name']}%")

if 'email' in filter_conditions:

    query += " AND t1.email LIKE %s"

    params.append(f"%{filter_conditions['email']}%")

if 'title' in filter_conditions:

    query += " AND t2.title LIKE %s"

    params.append(f"%{filter_conditions['title']}%")

if 'group_by' in filter_conditions:

    query += " GROUP BY " + ", ".join(filter_conditions['group_by'])

start_time = time.time()

c.execute(query, tuple(params))

result = c.fetchall()

end_time = time.time()

execution_time = (end_time - start_time) * 1000

print(f"Query executed in: {execution_time:.2f} ms")

return result

```

## 8. *Exit* - завершения програми

## Результати роботи програми

1. Вивід назв всіх таблиць бази даних:

```
Make a choice: 1
Table names:
('users',)
('rental',)
('reservation',)
('transactions',)
('reviews',)
```

2. Вивід назв всіх стовпців обраної таблиці:

```
Make a choice: 2
Enter the table name: users
Column names:
('user_id',)
('name',)
('email',)
('role',)
```

Для подальших дій оберемо початкову таблицю users:

Початкова таблиця:

1 **SELECT** \* **from** users

Data Output

Messages

Notifications

</

3. Додавання даних до обраної таблиці:

```

Make a choice: 3
Enter the table name: users
Enter column names (space-separated): user_id name email role
Enter corresponding values (space-separated): 2323 lukas luk@gmail.com tenant
Data added successfully!

```

Результат:

1 **SELECT \* from users**

Data Output Messages Notifications

	user_id [PK] integer	name character varying (100)	email character varying (100)	role character varying (20)
1	9	Dmytro	dmitr@gmail.com	landlord
2	10	John	john@mail.com	tenant
3	2323	lukas	luk@gmail.com	tenant

4. Оновлення даних в обраній таблиці:

```

Make a choice: 4
Enter the table name: users
Enter the name of the column to update: role
Enter the ID of the row to update: 2323
Enter the new value: landlord
Data updated successfully!

```

Результат:

1 **SELECT \* from users**

Data Output Messages Notifications

	user_id [PK] integer	name character varying (100)	email character varying (100)	role character varying (20)
1	9	Dmytro	dmitr@gmail.com	landlord
2	10	John	john@mail.com	tenant
3	2323	lukas	luk@gmail.com	landlord



## 5. Видалення даних з обраної таблиці:

```
Make a choice: 5
Enter the table name: users
Enter the ID of the row to delete: 2323
Row deleted successfully!
```

Результат:

Query

Query history











1

SELECT \* from users

Data Output

Messages

Notifications



	user_id [PK] integer	name character varying (100)	email character varying (100)	role character varying (20)
1	9	Dmytro	dmitr@gmail.com	landlord
2	10	John	john@mail.com	tenant

## 6. Генерація 100000 псевдовипадкових рядків:

```
Make a choice: 6
Enter the table name: users
Enter the number of rows to generate: 100000
Data for table users has been generated successfully
```

Результат:

1

SELECT \* from users

Data Output

Messages

Notifications

SQL

	user_id [PK] integer	name character varying (100)	email character varying (100)	role character varying (20)
1	9	Dmytro	dmitr@gmail.com	landlord
2	10	John	john@mail.com	tenant
3	96	Text name 85	user85@example.com	tenant
4	11	Text name 0	user0@example.com	tenant
5	12	Text name 1	user1@example.com	tenant
6	13	Text name 2	user2@example.com	tenant
7	14	Text name 3	user3@example.com	landlord
8	15	Text name 4	user4@example.com	tenant

1

SELECT COUNT(\*) from users

Data Output

Messages

Notifications

≡+

▼

▼

SQL

count

bigint

1

100002

## 7. Пошук по заданим параметрам:

Початкові таблиці:

1SELECT \* from users

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

📦

⬇️

📈

SQL

	user_id [PK] integer	name character varying (100)	email character varying (100)	role character varying (20)
1	9	Dmytro	dmitr@gmail.com	landlord
2	10	John	john@mail.com	tenant

1SELECT \* from rental

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

📦

⬇️

📈

SQL

	rental_id [PK] integer	user_id integer	title character varying (50)	description character varying (1000)	price numeric (12,2)
1	11	9	квартира	Вул. Київська, 12, Київ	50000.00
2	12	9	будинок	Вул. Гагарина, 5, Житомир	100000.00

Результат успішного пошуку:

```

Make a choice: 7

Enter search parameters:
Minimum price: 10
Maximum price: 100000
Name (LIKE pattern): D
Email (LIKE pattern): gmail.com
Title (LIKE pattern): квартира
Query executed in: 2.24 ms

Search results:
(9, 'Dmytro', 'dmitr@gmail.com', 'landlord', 11, 9, 'квартира', 'Вул. Київська, 12, Київ', Decimal('50000.00'))

```

Результат не успішного пошуку (немає орендаторів з таким user\_id):

```
Make a choice: 7

Enter search parameters:
Minimum price: 10
Maximum price: 516546545
Name (LIKE pattern):
Email (LIKE pattern):
Title (LIKE pattern):
Query executed in: 4.87 ms

No data matching the search criteria.
```

## Код програми

### main.py

```
from controller import Controller

if __name__ == "__main__":
    controller = Controller()
    controller.run()
```

### model.py

```
import psycopg2
import time

class Model:
    def __init__(self):
        self.conn = psycopg2.connect(
            dbname='booking_online',
            user='postgres',
            password='38743874',
            host='localhost',
            port=5432
        )

    def get_all_tables(self):
        c = self.conn.cursor()

        c.execute("SELECT table_name FROM information_schema.tables WHERE
table_schema = 'public'")

        return c.fetchall()
```

```

def get_all_columns(self, table_name):

    c = self.conn.cursor()

    c.execute("SELECT column_name FROM information_schema.columns WHERE
table_name = %s", (table_name,))

    return c.fetchall()


def add_data(self, table_name, columns, val):

    c = self.conn.cursor()

    columns_str = ', '.join(columns)

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

    identifier_column = f'{table_name[:-1]}_id' if
table_name.endswith('s') else f'{table_name}_id'

    c.execute(f'SELECT "{identifier_column}" FROM
"public"."{table_name}"')

    existing_identifiers = [item[0] for item in c.fetchall()]

    if int(val[columns.index(identifier_column)]) in
existing_identifiers:

        return 2

    external_keys = [col for col in columns if col.endswith('_id') and
col != identifier_column]

    for key_column in external_keys:

        referenced_table = key_column[:-3] + 's'

        c.execute(f'SELECT "{key_column}" FROM
"public"."{referenced_table}"')

        if int(val[columns.index(key_column)]) not in [item[0] for item
in c.fetchall()]:

            return 3

```

```

        c.execute(f'INSERT INTO "public"."{table_name}" ({columns_str})
VALUES ({placeholders})', val)

        self.conn.commit()

        return 1

def update_data(self, table_name, column, id, new_value):

    c = self.conn.cursor()

    identifier_column = f'{table_name[:-1]}_id' if
table_name.endswith('s') else f'{table_name}_id'

    if column.endswith('_id') and column != identifier_column:

        referenced_table = column[:-3] + 's'

        c.execute(f'SELECT "{column}" FROM
"public"."{referenced_table}"')

        if int(new_value) not in [item[0] for item in c.fetchall()]:

            return 3

        c.execute(f'UPDATE "public"."{table_name}" SET "{column}" = %s WHERE
"{identifier_column}" = %s',

                (new_value, id))

        self.conn.commit()

        return 1

def delete_data(self, table_name, id):

    c = self.conn.cursor()

    identifier_column = f'{table_name[:-1]}_id' if
table_name.endswith('s') else f'{table_name}_id'

    c.execute("SELECT table_name FROM information_schema.tables WHERE
table_schema = 'public' AND table_type = "

            "'BASE TABLE'")

```

```

tables = [item[0] for item in c.fetchall()]

for current_table in tables:

    if current_table == table_name:

        continue

    c.execute("SELECT column_name FROM information_schema.columns
WHERE table_name = %s", (current_table,))

    if identifier_column in [col[0] for col in c.fetchall()]:

        c.execute(f'SELECT "{identifier_column}" FROM
"public"."{current_table}"')

        if id in [item[0] for item in c.fetchall()]:

            return 0

    c.execute(f'DELETE FROM "public"."{table_name}" WHERE
"{identifier_column}" = %s', (id,))

    self.conn.commit()

    return 1

def generate_data(self, table_name, count):

    c = self.conn.cursor()

    c.execute(f"SELECT column_name, data_type FROM
information_schema.columns WHERE table_name = %s", (table_name,))

    columns_info = c.fetchall()

    id_column = f'{table_name[:-1]}_id'

    for i in range(count):

        insert_query = f'INSERT INTO "public"."{table_name}" ('

        select_subquery = ""

```

```

for column_info in columns_info:

    column_name = column_info[0]

    column_type = column_info[1]

    if column_name == id_column:

        c.execute(f'SELECT max("{id_column}") FROM
"public"."{table_name}")

        max_id = c.fetchone()[0] or 0

        select_subquery += f'{max_id + 1},'

    elif column_name == 'role':

        select_subquery += "(CASE WHEN RANDOM() < 0.5 THEN
'tenant' ELSE 'landlord' END),"

    elif column_name == 'email':

        select_subquery += f'"user{i}@example.com",'

    elif column_name.endswith('_id'):

        related_table_name = column_name[:-3].capitalize()

        c.execute(

            f'SELECT {related_table_name.lower()}_id FROM
"public"."{related_table_name}" ORDER BY RANDOM() LIMIT 1')

        related_id = c.fetchone()[0]

        select_subquery += f'{related_id},'

    elif column_type == 'integer':

        select_subquery += f'trunc(random()*100)::INT,'

    elif column_type == 'character varying':

        select_subquery += f'"Text {column_name} {i}',"

    elif column_type == 'date':

        select_subquery += '"2024-01-01",'

    elif column_type == 'timestamp with time zone':

```



```

        select_subquery += "'2024-01-01 00:00:00+03',"

    else:

        continue

    insert_query += f'"{column_name}",'

    insert_query = insert_query.rstrip(',') + f') VALUES
({select_subquery[:-1]})'

    c.execute(insert_query)

self.conn.commit()

def search_data(self, table1, table2, filter_conditions):

    c = self.conn.cursor()

    query = f"""

        SELECT *

        FROM "public"."{table1}" t1

        JOIN "public"."{table2}" t2 ON t1.user_id = t2.user_id

        WHERE 1=1

    """

    params = []

    if 'price_min' in filter_conditions and 'price_max' in
filter_conditions:

        query += " AND t2.price BETWEEN %s AND %s"

        params.extend([filter_conditions['price_min'],
filter_conditions['price_max']])

```

```

if 'name' in filter_conditions:

    query += " AND t1.name LIKE %s"

    params.append(f"%{filter_conditions['name']}%")


if 'email' in filter_conditions:

    query += " AND t1.email LIKE %s"

    params.append(f"%{filter_conditions['email']}%")


if 'title' in filter_conditions:

    query += " AND t2.title LIKE %s"

    params.append(f"%{filter_conditions['title']}%")


if 'group_by' in filter_conditions:

    query += " GROUP BY " + ", ".join(filter_conditions['group_by'])


start_time = time.time()

c.execute(query, tuple(params))

result = c.fetchall()

end_time = time.time()


execution_time = (end_time - start_time) * 1000

print(f"Query executed in: {execution_time:.2f} ms")


return result

```

[view.py](#)

```
import time

class View:

    def show_menu(self):

        while True:

            print("Menu:")

            print("1. Display table names")

            print("2. Display column names of a table")

            print("3. Add data to a table")

            print("4. Update data in a table")

            print("5. Delete data from a table")

            print("6. Generate data for a table")

            print("7. Search data")

            print("8. Exit")

            choice = input("Make a choice: ")

            if choice in ('1', '2', '3', '4', '5', '6', '7', '8'):

                return choice

            else:

                print("Please enter a valid option number (1 to 8)")

                time.sleep(2)

    def show_message(self, message):

        print(message)

        time.sleep(2)

    def ask_continue(self):
```

```

        agree = input("Continue making changes? (y/n) ")

        return agree

def show_tables(self, tables):

    print("Table names:")

    for table in tables:

        print(table)

    time.sleep(2)

def ask_table(self):

    table_name = input("Enter the table name: ")

    return table_name

def show_columns(self, columns):

    print("Column names:")

    for column in columns:

        print(column)

    time.sleep(2)

def insert(self):

    while True:

        try:

            table = input("Enter the table name: ")

            columns = input("Enter column names (space-separated):
").split()

            val = input("Enter corresponding values (space-separated):
").split()

            if len(columns) != len(val):

```

```
        raise ValueError("The number of columns must match the  
number of values.")
```

```
    return table, columns, val
```

```
except ValueError as e:
```

```
    print(f"Error: {e}")
```

```
def update(self):
```

```
    while True:
```

```
        try:
```

```
            table = input("Enter the table name: ")
```

```
            column = input("Enter the name of the column to update: ")
```

```
            id = int(input("Enter the ID of the row to update: "))
```

```
            new_value = input("Enter the new value: ")
```

```
            return table, column, id, new_value
```

```
        except ValueError as e:
```

```
            print(f"Error: {e}")
```

```
def delete(self):
```

```
    while True:
```

```
        try:
```

```
            table = input("Enter the table name: ")
```

```
            id = int(input("Enter the ID of the row to delete: "))
```

```
            return table, id
```

```
        except ValueError as e:
```

```
            print(f"Error: {e}")
```

```
def generate_data_input(self):
```

```

while True:

    try:

        table_name = input("Enter the table name: ")

        num_rows = int(input("Enter the number of rows to generate:
"))

        return table_name, num_rows

    except ValueError as e:

        print(f"Error: {e}")

def search_data_input(self):

    while True:

        try:

            print("\nEnter search parameters:")

            price_min = input("Minimum price: ")

            price_max = input("Maximum price: ")

            name = input("Name (LIKE pattern): ")

            email = input("Email (LIKE pattern): ")

            title = input("Title (LIKE pattern): ")

            filter_conditions = {

                'price_min': int(price_min) if price_min else None,

                'price_max': int(price_max) if price_max else None,

                'name': name if name else None,

                'email': email if email else None,

                'title': title if title else None,

                'group_by': ['t1.user_id', 't2.rental_id']

            }

```

```
        return filter_conditions

    except ValueError as e:

        print(f"Error: {e}")
```

### controller.py

```
import sys

from model import Model

from view import View


class Controller:

    def __init__(self):

        self.view = View()

        try:

            self.model = Model()

            self.view.show_message("Connected to the database")

        except Exception as e:

            self.view.show_message(f"An error occurred during initialization:
{e}")

            sys.exit(1)

    def run(self):

        while True:

            choice = self.view.show_menu()

            if choice == '1':

                self.view_tables()

            elif choice == '2':
```

```

        self.view_columns()

    elif choice == '3':

        self.add_data()

    elif choice == '4':

        self.update_data()

    elif choice == '5':

        self.delete_data()

    elif choice == '6':

        self.generate_data()

    elif choice == '7':

        self.search_data()

    elif choice == '8':

        break

def view_tables(self):

    tables = self.model.get_all_tables()

    self.view.show_tables(tables)

def view_columns(self):

    table_name = self.view.ask_table()

    columns = self.model.get_all_columns(table_name)

    self.view.show_columns(columns)

def add_data(self):

    while True:

        table, columns, val = self.view.insert()

        error = self.model.add_data(table, columns, val)

        if int(error) == 1:

```



```

        self.view.show_message("Data added successfully!")

        agree = self.view.ask_continue()

        if agree == 'n':

            break

    elif int(error) == 2:

        self.view.show_message("Unique identifier already exists!")

        agree = self.view.ask_continue()

        if agree == 'n':

            break

    else:

        self.view.show_message("Invalid foreign key")

        agree = self.view.ask_continue()

        if agree == 'n':

            break

def update_data(self):

    while True:

        table, column, id, new_value = self.view.update()

        error = self.model.update_data(table, column, id, new_value)

        if int(error) == 1:

            self.view.show_message("Data updated successfully!")

            agree = self.view.ask_continue()

            if agree == 'n':

                break

        elif int(error) == 2:

            self.view.show_message(f"Unique identifier {new_value}
already exists!")

            agree = self.view.ask_continue()

```

```

        if agree == 'n':

            break

    else:

        self.view.show_message(f"Invalid foreign key {new_value} in
column {column}")

        agree = self.view.ask_continue()

        if agree == 'n':

            break

def delete_data(self):

    while True:

        table, id = self.view.delete()

        error = self.model.delete_data(table, id)

        if int(error) == 1:

            self.view.show_message("Row deleted successfully!")

            agree = self.view.ask_continue()

            if agree == 'n':

                break

        else:

            self.view.show_message("Cannot delete row due to related data
existing")

            agree = self.view.ask_continue()

            if agree == 'n':

                break

def generate_data(self):

    table_name, num_rows = self.view.generate_data_input()

    self.model.generate_data(table_name, num_rows)

```

```
        self.view.show_message(f"Data for table {table_name} has been  
generated successfully")
```

```
def search_data(self):
```

```
    result = self.model.search_data('users', 'rental',  
self.view.search_data_input())
```

```
    if result:
```

```
        print("\nSearch results:")
```

```
        for row in result:
```

```
            print(row)
```

```
    else:
```

```
        print("\nNo data matching the search criteria.")
```