A black and white drawing of a building

Description automatically generated

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 ІІI курсу

групи КВ-21

Недобойко Д. О.

Перевірив:

**Київ – 2024**

**Репозиторій GitHub:** [**https://github.com/yobananes/DB/tree/main/rgr**](https://github.com/yobananes/DB/tree/main/rgr)

*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).

**Опис сутностей**

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

1. Клієнт(User)

Атрибути: ідентифікатор клієнта, ім’я, електронна скринька, номер телефону.

Призначення: збереження даних щодо клієнтів.

1. Замовлення(Order)

Атрибути: ідентифікатор замовлення, дата замовлення .

Призначення: збереження даних щодо замовлень.

1. Позиція їжі (Food)

Атрибути: ідентифікатор їжі, назва їжі, ціна

Призначення: збереження даних щодо товарів.

1. Доставка(Delivery)

Атрибути: ідентифікатор доставки, адреса доставк и, статус доставки.

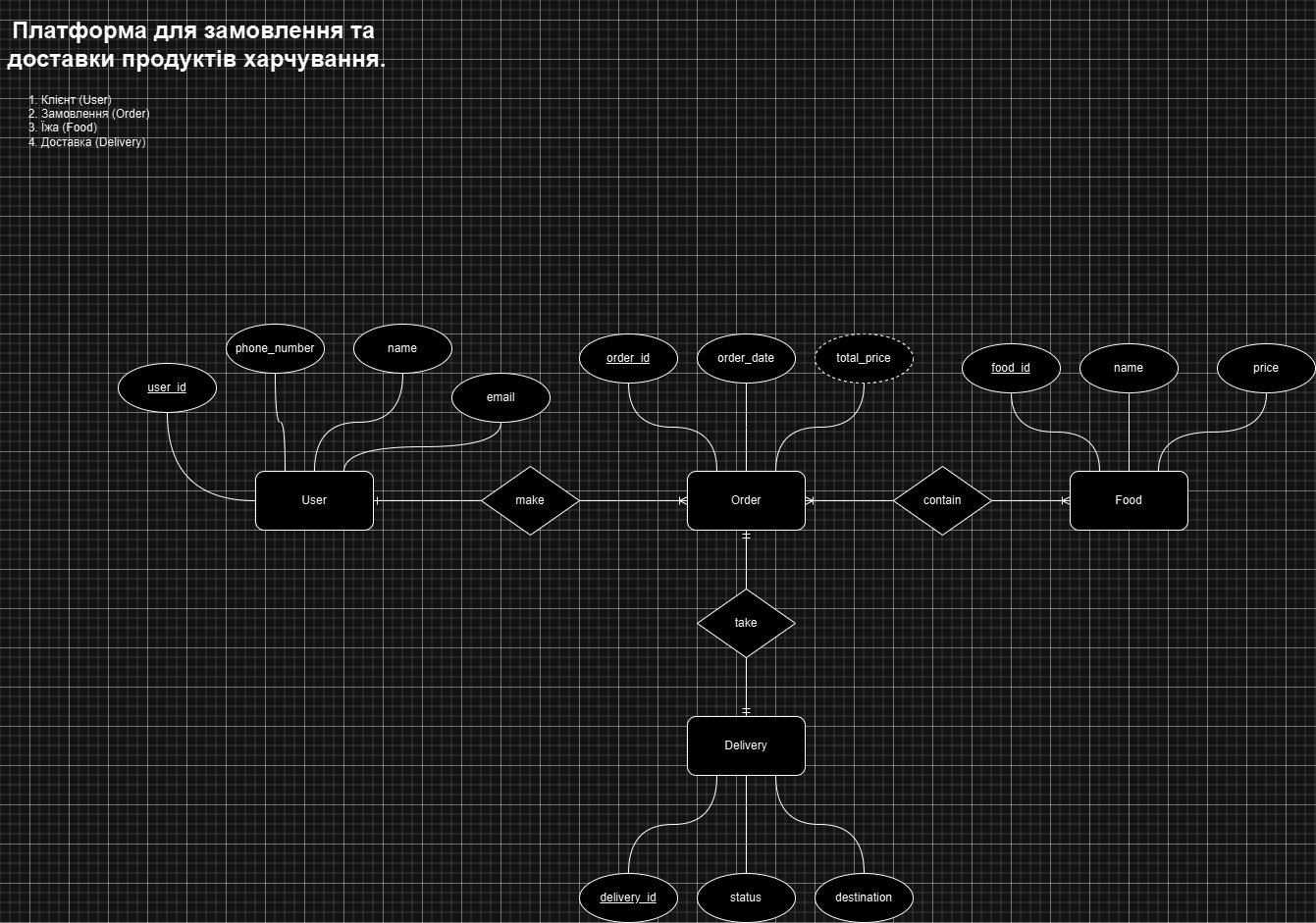
Призначення: збереження даних щодо доставок.

**Опис зв’язків між сутностями**

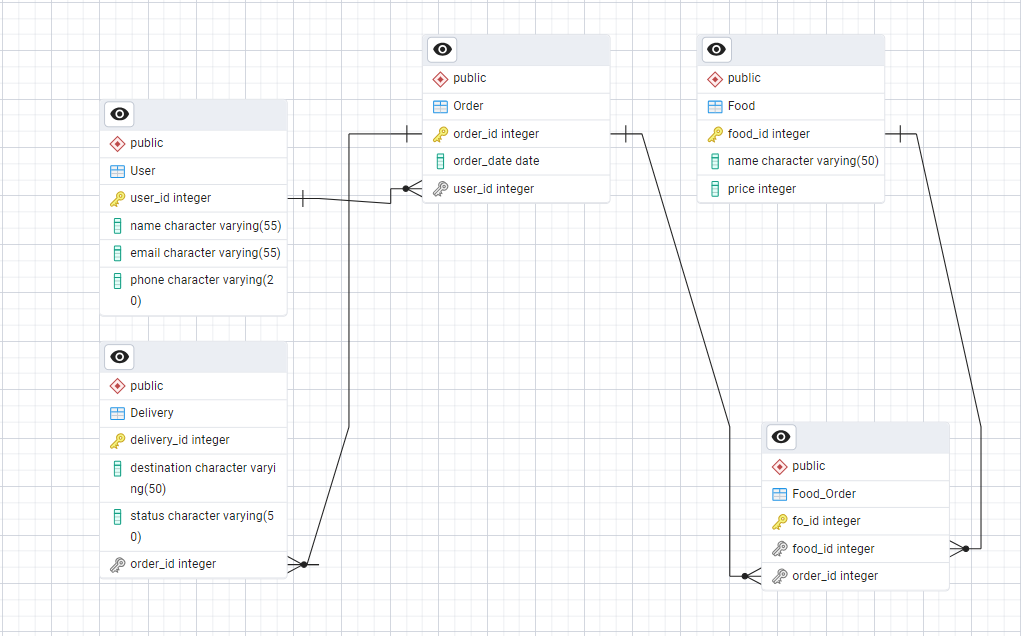
Зв’язок «Клієнт» - «Замовлення» є зв’язком 1:N. Один клієнт може створити багато замовлень, але кожне замовлення прив’язано лише до одного клієнта.

Зв’язок «Замовлення» - «Їжа» є зв’язком N:M. Одне замовлення може містити багато позицій їжі, і одна позиція може входити до складу багатьох замовлень.

Зв’язок «Замовлення» - «Доставка» є зв’язком 1:1. Кожне замовлення може мати тільки одну доставку, і кожна доставка бере лише по одному замовленню.

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

*Рисунок 1 – ER-діаграма, побудована за нотацією Пташиної лапки (Crow’s foot)*



*Рисунок 2 – Схема бази даних*

*Для роботи було використано:*

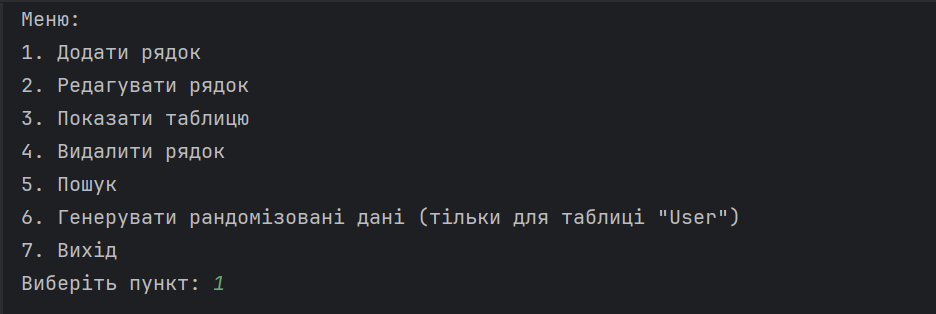
Середовище для відлагодження SQL-запитів до бази даних – PgAdmin4.

Мова програмування – Python 3.10

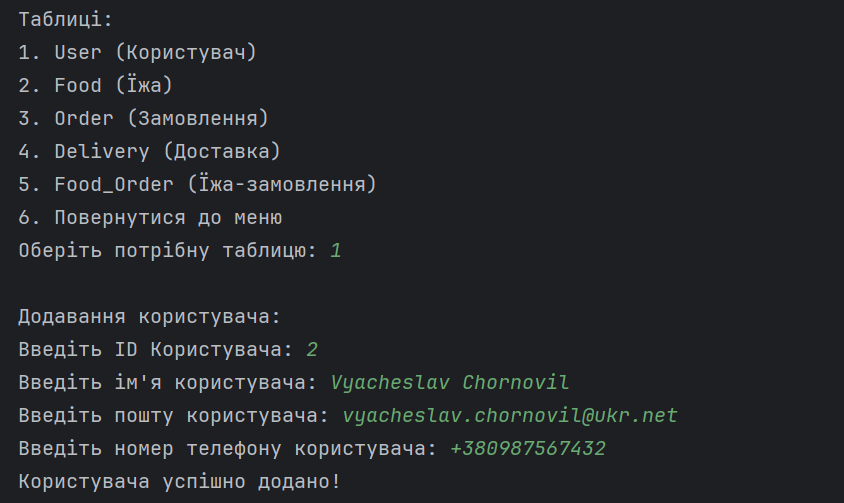
Середовище розробки програмного забезпечення – PyCharm Community Edition.

Бібліотека взаємодії з PostgreSQL - psycopg2.

**Меню користувача з описом функціональності кожного пункту**









**Меню складається з 7 пунктів:**

1) **Додати рядок** служить для додавання рядка в таблицю. Після вибору цього пункту, потрібно обрати таблицю, для якої буде виконана ця операція, після чого, треба ввести дані для кожного атрибуту таблиці, щоб додати новий рядок.

2) **Редагувати рядок** використовується для редагування полів по ID у таблицях. Спочатку потрібно обрати, для якої таблиці буде відбуватися, після чого потрібно ввести id поля, яке потрібно змінити. Залишається ввести нові дані для кожного атрибуту таблиці.

3) **Показати таблицю** служить для показу таблиці. Перед виведенням, користувач обирає, яку саме таблицю потрібно вивести. Після цього на екрані виводяться всі поля обраної таблиці БД.

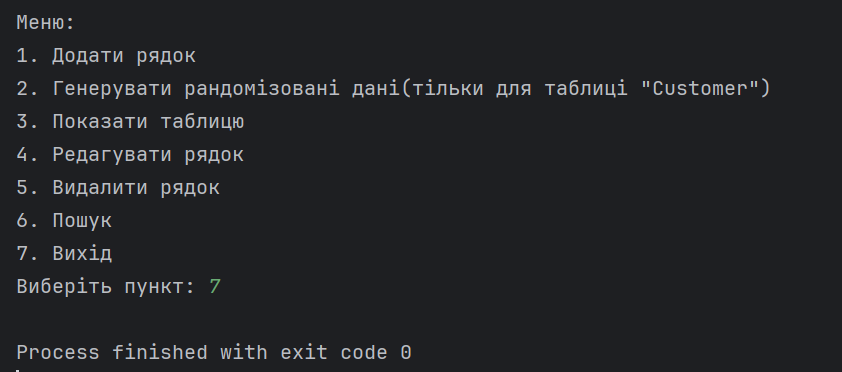
4) **Видалити рядок** служить для видалення рядку по ID у таблицях. Спочатку потрібно обрати, для якої таблиці буде відбуватися видалення рядка, після чого користувач вводить ID рядка, який потрібно видалити.

5) **Пошук** створений для пошуку за атрибутами з декількох таблиць.

6) **Генерування «рандомізованих» даних**. Для цього пункту було обрано таблицю “User”. Цей пункт створений для додавання псевдорандомізованих даних. Користувачу потрібно ввести число полів, скільки треба додати у таблицю.

Роботу цих запитів буде розглянуто у **завданні №3**.

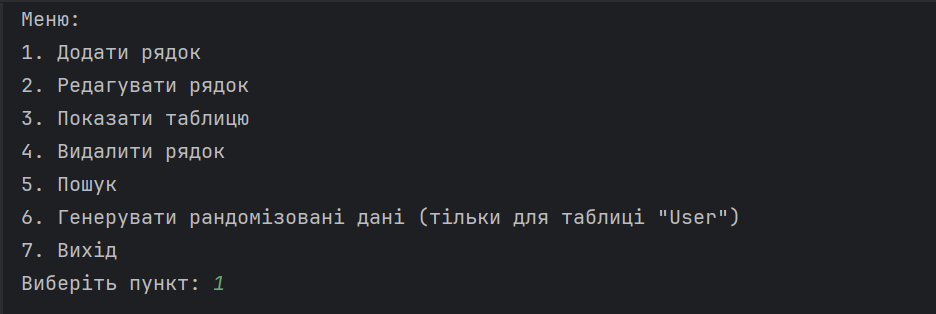
7) **Вихід** служить для виходу з програми.

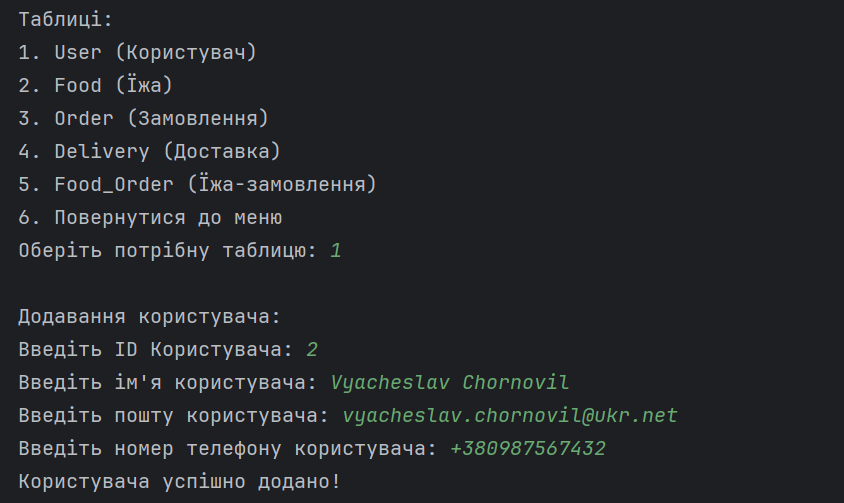
****

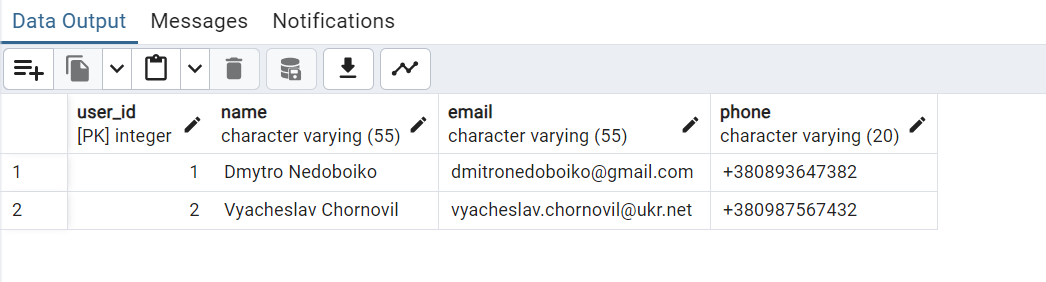
**Завдання №1**

**Додавання рядка**

**Процес додавання рядка:**







**Лістинг add:**

# Таблиця User

def add\_user(self, user\_id, name, email, phone):

c = self.conn.cursor()

c.execute('INSERT INTO "User" ("user\_id", "name", "email", "phone") VALUES (%s, %s, %s, %s)',

(user\_id, name, email, phone))

self.conn.commit()

#Таблиця Food

def add\_food(self, food\_id, name, price):

c = self.conn.cursor()

c.execute('INSERT INTO "Food" ("food\_id", "name", "price") VALUES (%s, %s, %s)',

(food\_id, name, price))

self.conn.commit()

#Таблиця Order

def add\_order(self, order\_id, order\_date, user\_id):

c = self.conn.cursor()

c.execute('INSERT INTO "Order" ("order\_id", "order\_date", "user\_id") VALUES (%s, %s, %s)',

(order\_id, order\_date, user\_id))

self.conn.commit()

#Таблиця Delivery

def add\_delivery(self, delivery\_id, destination, status, order\_id):

c = self.conn.cursor()

c.execute('INSERT INTO "Delivery" ("delivery\_id", "destination", "status", "order\_id") VALUES (%s, %s, %s, %s)',

(delivery\_id, destination, status, order\_id))

self.conn.commit()

#Таблиця Food\_Order

def add\_food\_order(self, fo\_id, food\_id, order\_id):

c = self.conn.cursor()

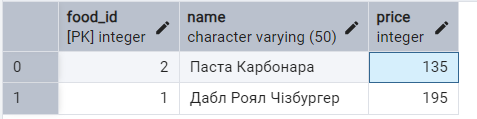
c.execute('INSERT INTO "Food\_Order" ("fo\_id","food\_id", "order\_id") VALUES (%s, %s,%s)',

(fo\_id, food\_id, order\_id))

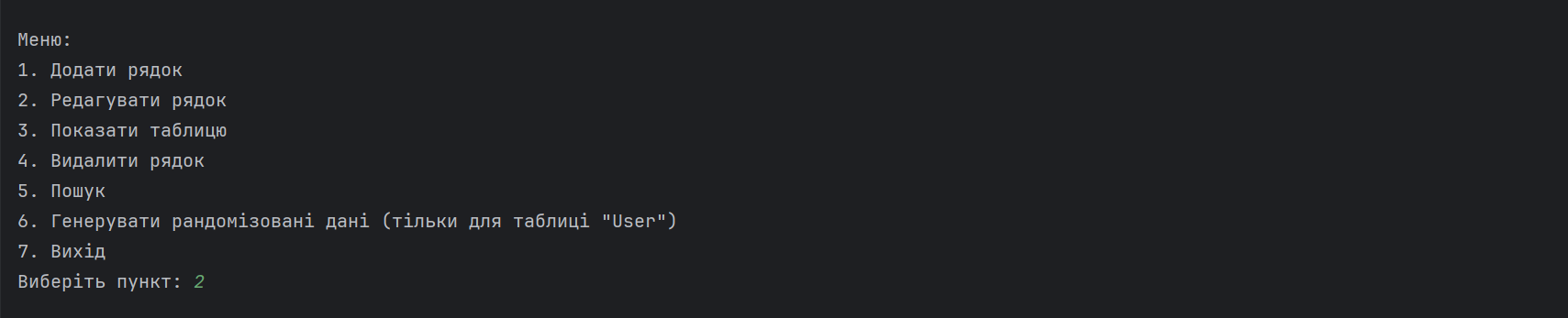
self.conn.commit()

**Редагування рядка**

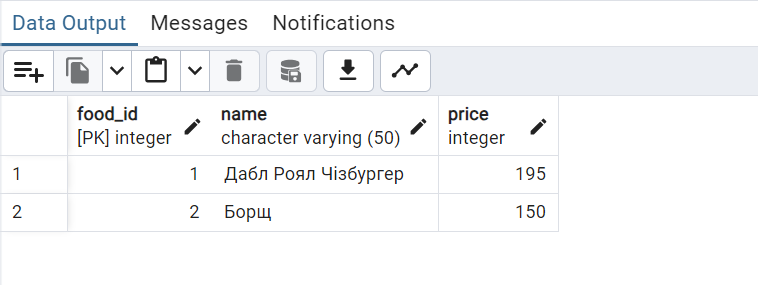
**Таблиця «Food»:**



**Процес редагування рядка:**







**Лістинг update:**

def update\_user(self, name, email, phone, id):

c = self.conn.cursor()

c.execute('UPDATE "User" SET "name"=%s, "email"=%s, "phone"=%s WHERE "user\_id"=%s',

(name, email, phone, id))

self.conn.commit()

def update\_food(self, name, price, id):

c = self.conn.cursor()

c.execute('UPDATE "Food" SET "name"=%s, "price"=%s WHERE "food\_id"=%s',

(name, price, id))

self.conn.commit()

def update\_order(self, order\_date, user\_id, id):

c = self.conn.cursor()

c.execute('UPDATE "Order" SET "order\_date"=%s, "user\_id"=%s WHERE "order\_id"=%s',

(order\_date, user\_id, id))

self.conn.commit()

def update\_delivery(self, destination, status, order\_id, id):

c = self.conn.cursor()

c.execute('UPDATE "Delivery" SET "destination"=%s, "status"=%s,"order\_id"=%s WHERE "delivery\_id"=%s',

(destination, status, order\_id, id))

self.conn.commit()

def update\_food\_order(self, food\_id, order\_id, id):

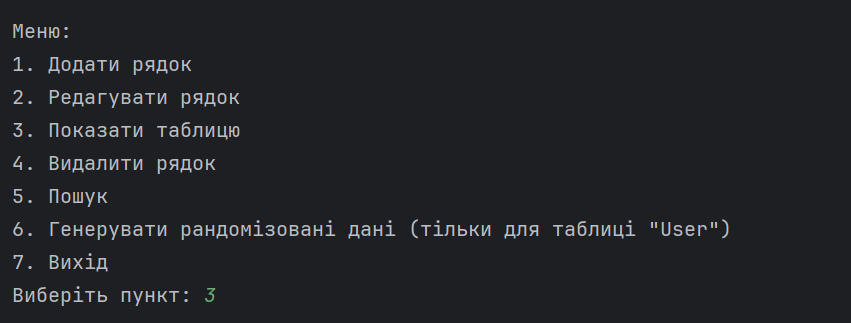
c = self.conn.cursor()

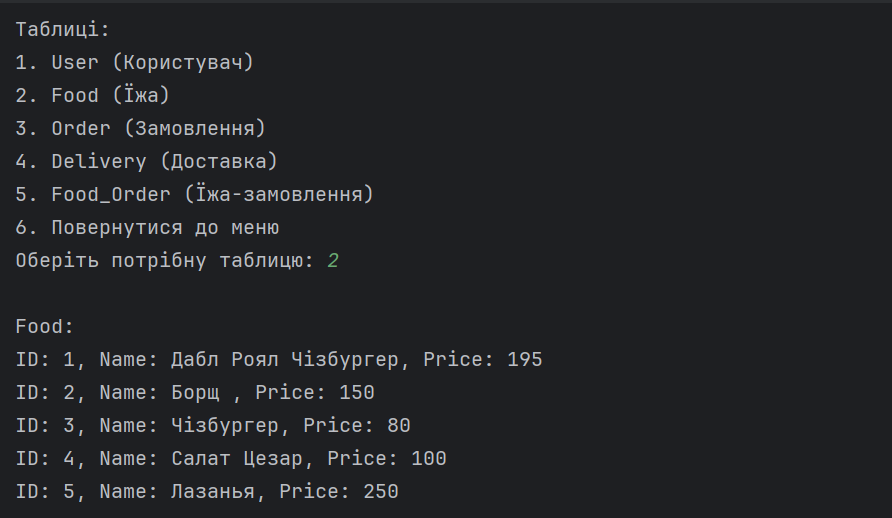
c.execute('UPDATE "Food\_Order" SET "food\_id"=%s,"order\_id"=%s WHERE "fo\_id"=%s',

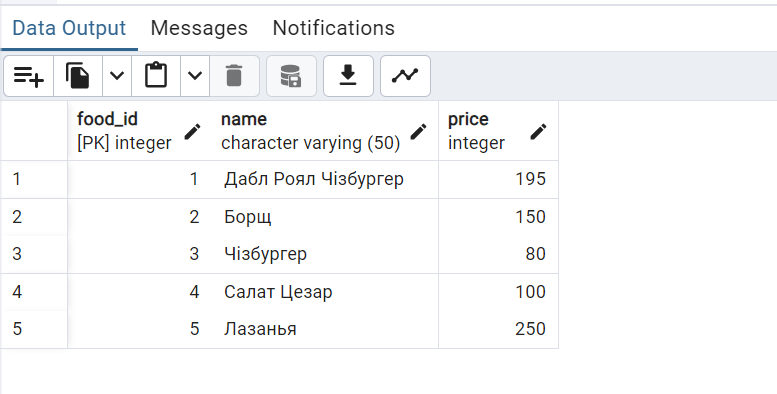
(food\_id, order\_id, id))

self.conn.commit()

**Показати таблицю**







**Лістинг get:**

def get\_all\_users(self):

with self.conn.cursor() as cursor:

cursor.execute('SELECT \* FROM "User" ORDER BY user\_id ASC')

return cursor.fetchall()

def get\_all\_foods(self):

with self.conn.cursor() as cursor:

cursor.execute('SELECT \* FROM "Food" ORDER BY food\_id ASC')

return cursor.fetchall()

def get\_all\_orders(self):

with self.conn.cursor() as cursor:

cursor.execute('SELECT \* FROM "Order" ORDER BY order\_id ASC')

return cursor.fetchall()

def get\_all\_deliveries(self):

with self.conn.cursor() as cursor:

cursor.execute('SELECT \* FROM "Delivery" ORDER BY delivery\_id ASC')

return cursor.fetchall()

def get\_all\_food\_orders(self):

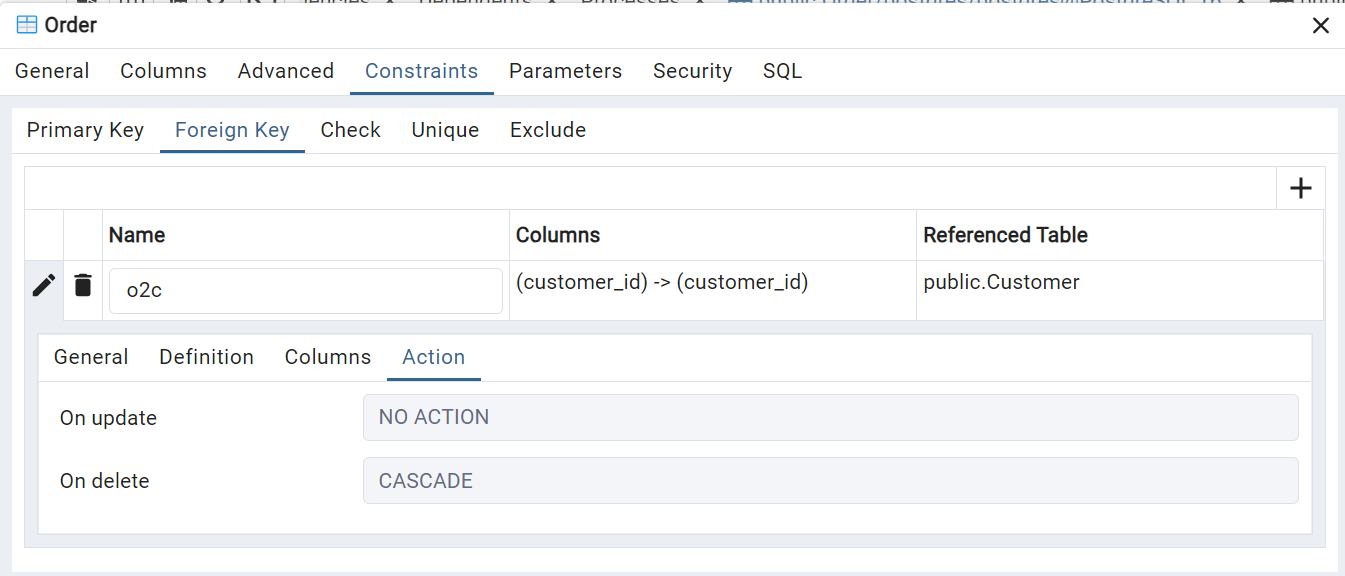
with self.conn.cursor() as cursor:

cursor.execute('SELECT \* FROM "Food\_Order" ORDER BY fo\_id ASC')

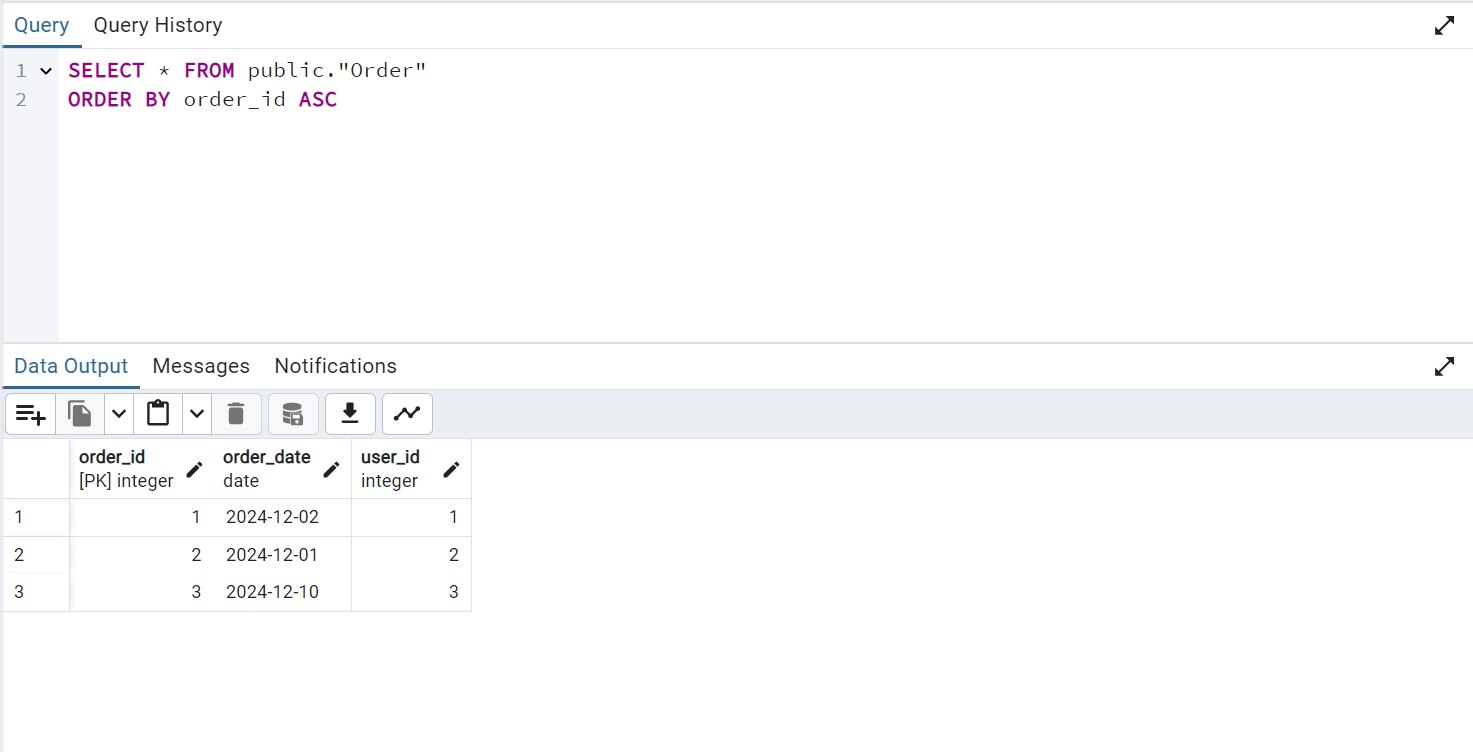
return cursor.fetchall()

**Видалення рядка**

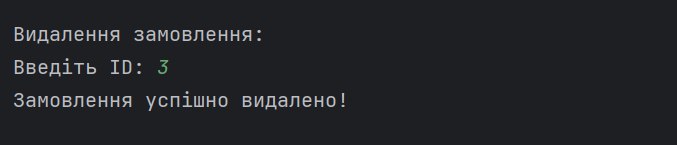
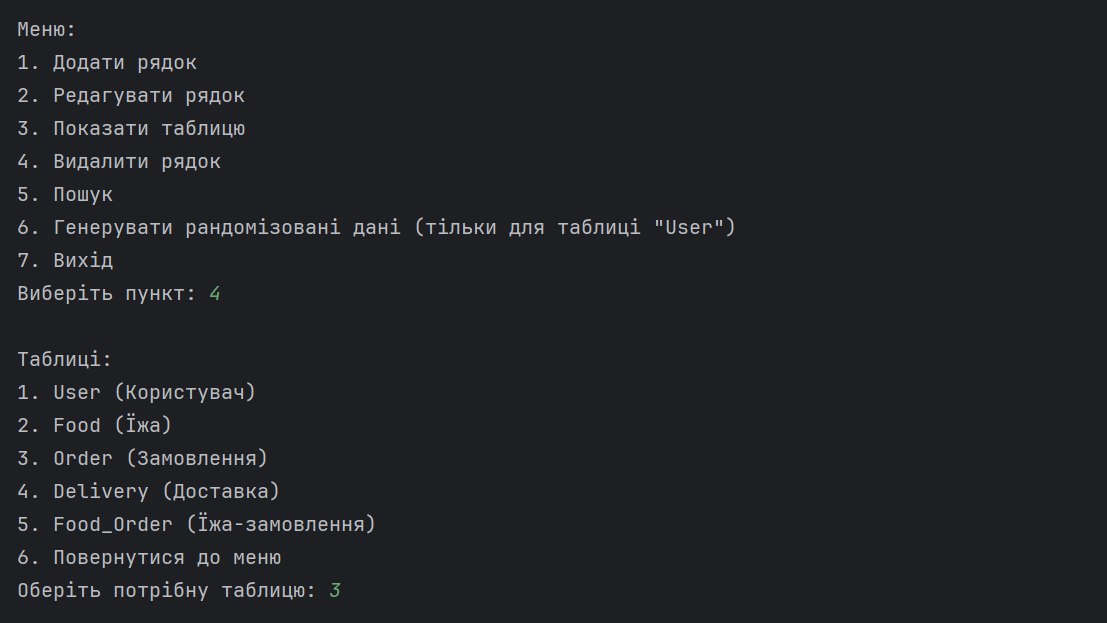
Видалення рядка здійснюється каскадно: при видаленні запису в батьківській таблиці відповідні записи в дочірній таблиці також будуть автоматично видалені. Після завершення операції буде виведено повідомлення із зазначенням кількості рядків, зачеплених цим запитом.



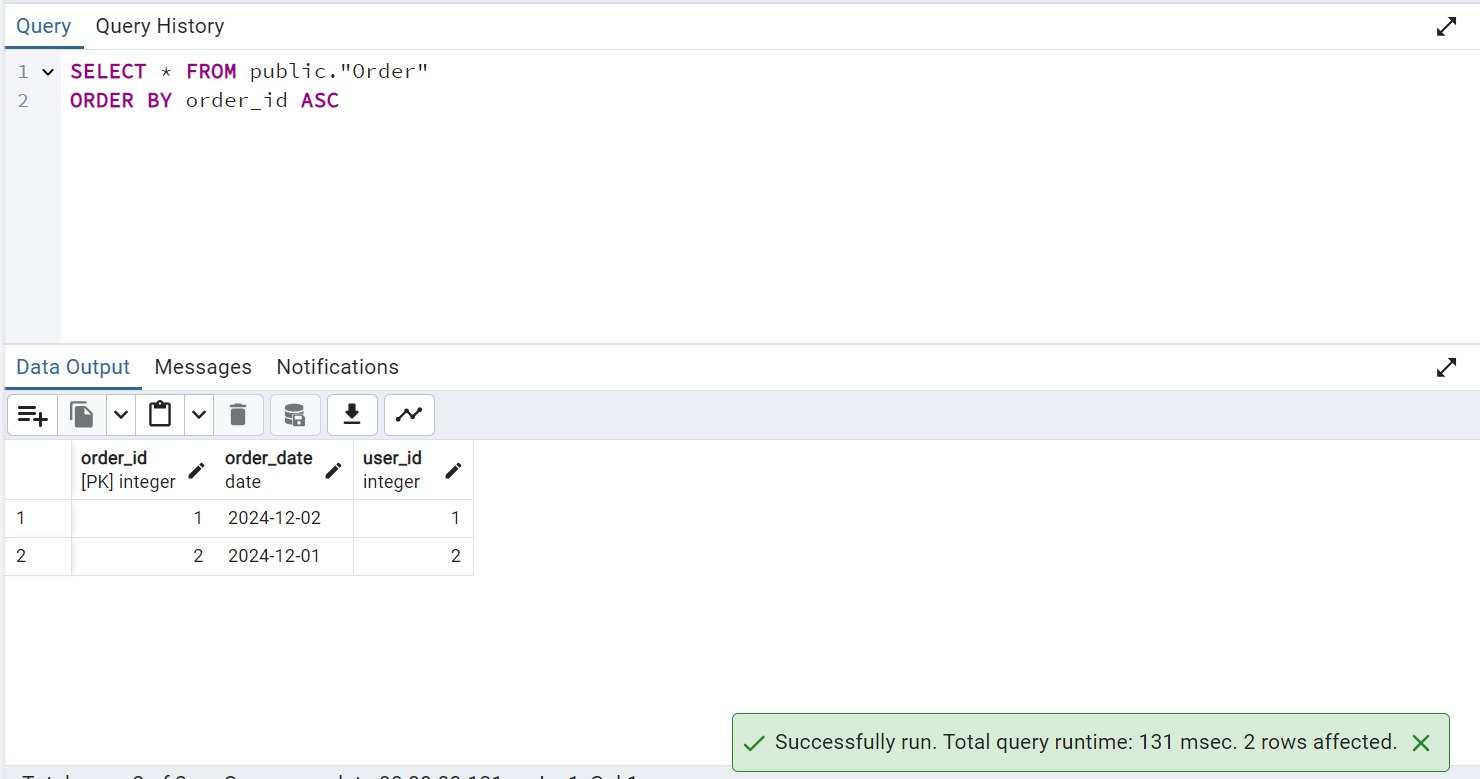
**Таблиця “Order” до:**

****

**Процес видалення рядка:**

****

**Таблиця “Order” після:**

****

**Лістинг delete:**

def delete\_user(self, user\_id):

c = self.conn.cursor()

c.execute('DELETE FROM "User" WHERE "user\_id"=%s', (user\_id,))

self.conn.commit()

def delete\_food(self, food\_id):

c = self.conn.cursor()

c.execute('DELETE FROM "Food" WHERE "food\_id"=%s', (food\_id,))

self.conn.commit()

def delete\_order(self, order\_id):

c = self.conn.cursor()

c.execute('DELETE FROM "Order" WHERE "order\_id"=%s', (order\_id,))

self.conn.commit()

def delete\_delivery(self, delivery\_id):

c = self.conn.cursor()

c.execute('DELETE FROM "Delivery" WHERE "delivery\_id"=%s', (delivery\_id,))

self.conn.commit()

def delete\_food\_order(self, fo\_id):

c = self.conn.cursor()

c.execute('DELETE FROM "Food\_Order" WHERE "fo\_id"=%s', (fo\_id,))

self.conn.commit()

**Завдання №2**

**Генерування «рандомізованих» даних**

Запит, що був використаний для генерування «рандомізованих» даних для таблиці “User”:

def add\_random\_fields(self, number):

c = self.conn.cursor()

first\_names = ['Andriy', 'Olena', 'Bohdan', 'Iryna', 'Oleksiy', 'Mariya', 'Sofiya', 'Viktoriya',

'Ivan', 'Nataliya', 'Alina', 'Rostyslav', 'Vadym', 'Vira', 'Maksym', 'Oksana','Liubov', 'Yevhen', 'Liliya',

'Yuriy', 'Kateryna', 'Dmytro', 'Anna', 'Mykhailo', 'Tetyana', 'Oleh', 'Liudmyla', 'Vasyl',

'Serhiy', 'Vladyslav', 'Yevgeniya', 'Oleksandr', 'Daryna', 'Illia', 'Svitlana',

'Pavlo', 'Valeriya', 'Larysa', 'Artem', 'Inna', 'Roman', 'Zoryana', 'Stepan',

'Hanna', 'Taras', 'Veronika', 'Yuliya', 'Kyrylo', 'Yaroslava', 'Hryhoriy'

]

last\_names = ['Shevchenko', 'Kovalenko', 'Bondarenko', 'Melnyk', 'Kravchenko', 'Oliynyk', 'Tkachenko',

'Petrenko', 'Kalinychenko', 'Rudenko', 'Tymoshenko', 'Chernenko', 'Klymenko',

'Savchenko', 'Kuzmenko', 'Sydoenko', 'Lysenko', 'Havrylyuk', 'Demchenko', 'Moroz',

'Zakharchenko', 'Martynyuk', 'Ivanchenko', 'Honcharenko', 'Bondar', 'Rybak',

'Kravchuk', 'Hrytsenko', 'Shapoval', 'Yakovlenko', 'Onyshchenko', 'Pavlenko',

'Solovey', 'Skrypnyk', 'Levchenko', 'Bilan', 'Palamarchuk', 'Korol', 'Kulish'

]

query = '''

WITH max\_id AS (SELECT COALESCE(MAX("user\_id"), 0) FROM public."User")

INSERT INTO public."User" ("user\_id", "name", "email", "phone")

SELECT

(SELECT \* FROM max\_id) + row\_number() OVER () AS "user\_id",

CONCAT\_WS(' ', first\_name, last\_name) AS "name",

LOWER(first\_name || '.' || last\_name) || '@' ||

(CASE (random() \* 10)::integer

WHEN 0 THEN 'gmail'

WHEN 1 THEN 'ukr'

WHEN 2 THEN 'yahoo'

WHEN 3 THEN 'llkpi'

WHEN 4 THEN 'ua'

WHEN 5 THEN 'outlook'

WHEN 6 THEN 'hotmail'

WHEN 7 THEN 'evite'

WHEN 8 THEN 'mailchi'

WHEN 9 THEN 'odido'

WHEN 10 THEN 'gmx'

END) || '.com' AS "email",

CONCAT('+1', (1000000000 + floor(random() \* 9000000000)::bigint)::text) AS "phone"

FROM (SELECT

unnest(ARRAY[%s]) AS first\_name,

unnest(ARRAY[%s]) AS last\_name

LIMIT %s) AS names;

'''

first\_names\_sample = [random.choice(first\_names) for \_ in range(number)]

last\_names\_sample = [random.choice(last\_names) for \_ in range(number)]

c.execute(query, (first\_names\_sample, last\_names\_sample, number))

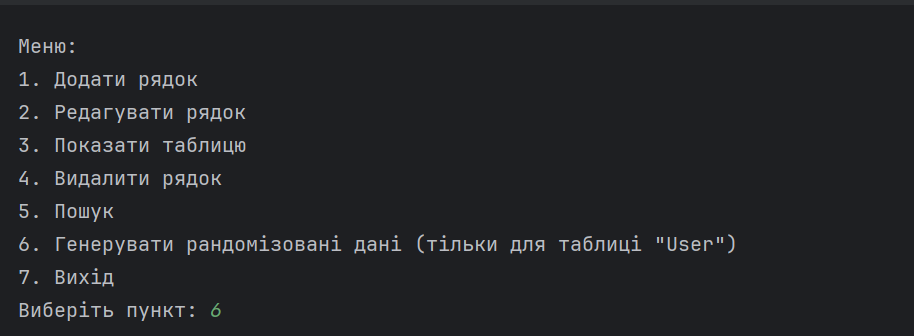
**Примітка:**  
Параметр number визначає кількість записів, які потрібно додати до таблиці "User". Імена та прізвища вибираються випадковим чином із заданих списків. Для кожного запису генерується:

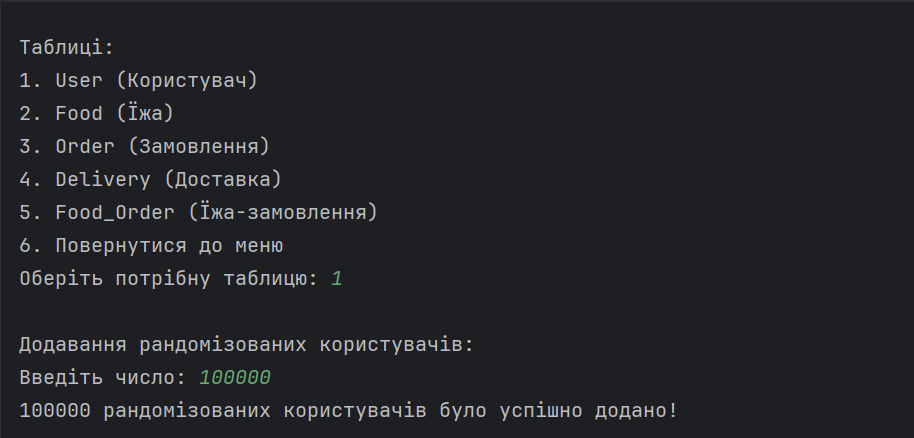
* **Ім'я**: поєднання випадкового імені та прізвища.
* **Email**: автоматично формується за шаблоном ім'я.прізвище@домен.com, де домен вибирається із запропонованого списку поштових сервісів.
* **Телефон**: унікальний номер у форматі +1XXXXXXXXXX, що імітує міжнародний формат.

У коді реалізована можливість створення довільної кількості записів за допомогою параметра number, щоб зберегти масштабованість та адаптивність функції.

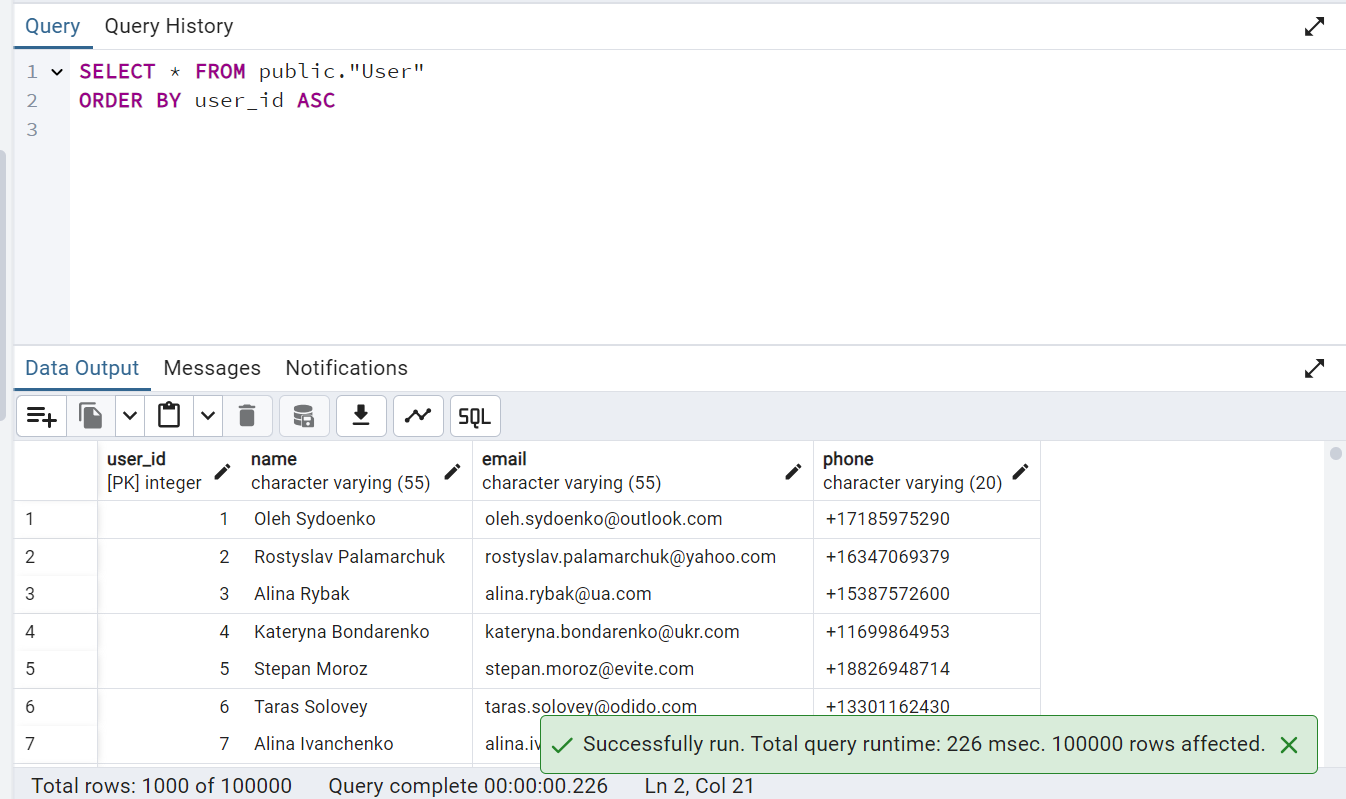
**Виконання:**





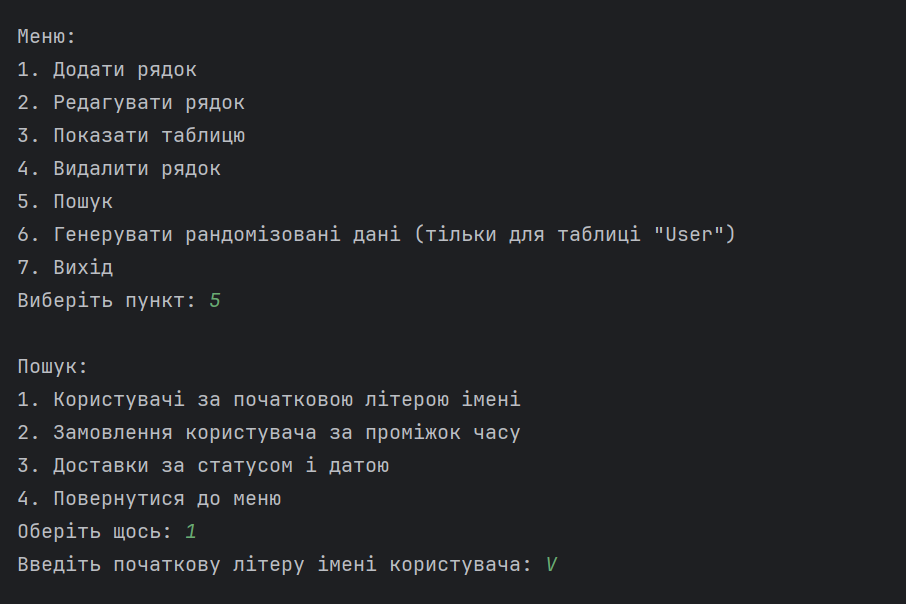






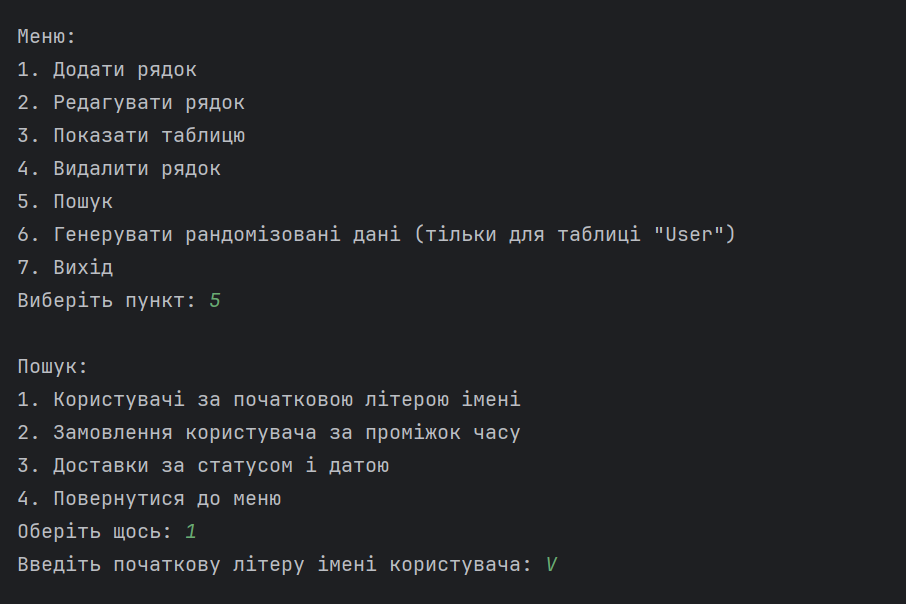
**Завдання №3**

**Пошук даних**

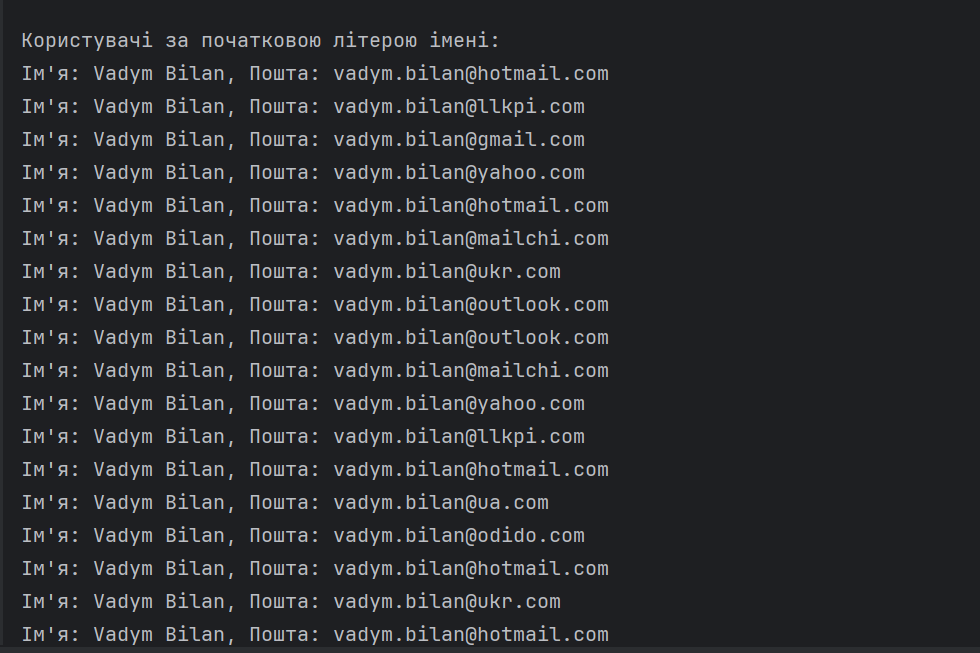


**Запит №1**

Користувачі за початковою літерою імені:



Цей запит створений для отримання списку певних клієнтів за першою літерою їхнього імені. На виході ми отримуємо список користувачів, з ім’ям клієнта та поштою. Під час виконання цього запиту було виміряно час, який становив 180.15 мс.





SQL запит виглядає наступним чином:

query = """

SELECT u.name, u.email

FROM "User" u

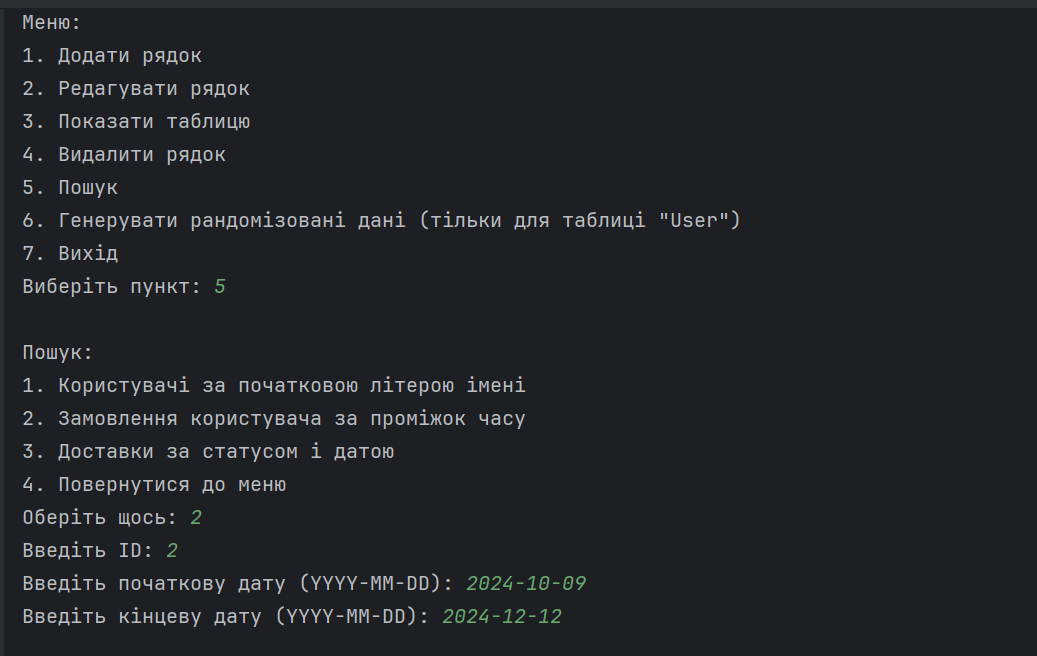
WHERE u.name LIKE %s

ORDER BY u.name;

"""

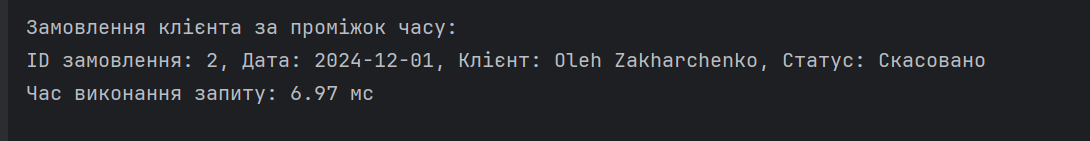
**Запит №2**

Замовлення користувача за певний проміжок часу:



Цей запит створений для визначення замовлень за певний проміжок часу. Цей запит знаходить усі замовлення та користувачів, які його зробили у визначені часові межі.

Під час виконання цього запиту було виміряно час, який становив 6.97 мс.



SQL запит виглядає наступним чином:

query = '''

SELECT o.order\_id, o.order\_date, u.name AS user\_name, d.status

FROM "Order" o

JOIN "User" u ON o.user\_id = u.user\_id

LEFT JOIN "Delivery" d ON o.order\_id = d.order\_id

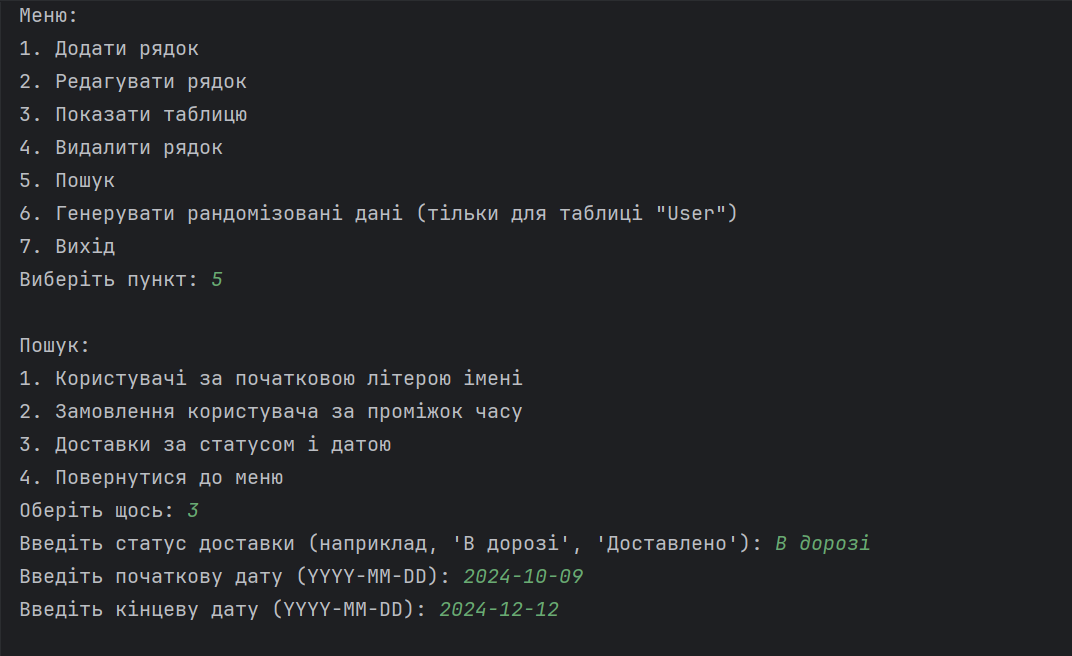
WHERE o.user\_id = %s AND o.order\_date BETWEEN %s AND %s

ORDER BY o.order\_date;

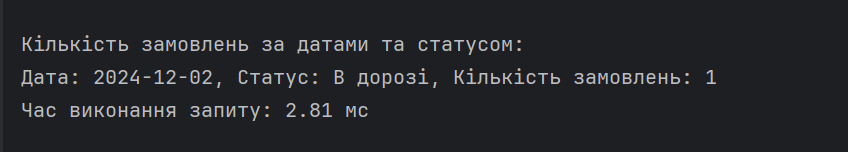
'''

**Запит №3**

Доставки за статусом і датою



Цей запит призначений для підрахунку кількості замовлень за статусом доставки та датою протягом визначеного періоду. Він забезпечує статистику щодо кількості замовлень із конкретним статусом доставки для кожної дати в заданому інтервалі. У результаті отримуємо дату, статус та кількість замовлень, здійснених у цей день із зазначеним статусом. Час виконання запиту склав 2.81 мс.



SQL запит виглядає наступним чином:

query = """

SELECT o.order\_date, d.status, COUNT(\*) AS order\_count

FROM "Order" o

JOIN "Delivery" d ON o.order\_id = d.order\_id

WHERE d.status = %s AND o.order\_date BETWEEN %s AND %s

GROUP BY o.order\_date, d.status

ORDER BY o.order\_date;

"""

**Завдання №4**

**Шаблон MVC**

**MVC** визначає архітектурний шаблон програмування, який включає три основні компоненти: Модель (Model), Вид (View) та Контролер (Controller). Цей шаблон дозволяє розділити логічні частини програми, щоб полегшити розробку, управління та розуміння коду.

Основні компоненти шаблону MVC:

**Model** – представляє клас, що описує логіку використовуваних даних. Клас реалізований у файлі model.py, у ньому відбуваються найважчі процеси (вставка, видалення, оновлення, пошук, рандомізація даних, звернення до бази даних) і після виконаної події відправляє результат доView.

**View** – це консольний інтерфейс, з яким взаємодіє користувач. Відповідає за введення/виведення даних. У програмі це реалізовано за допомогою файлу view.py (клас View та клас Menu).

**Controller** – забезпечує зв'язок між користувачем і системою, поданням і сховищем даних. Він отримує введені користувачем дані і обробляє їх. У програмі це реалізовано у файлі controller.py.

Користуючись шаблоном MVC, розробники можуть розділити програмний код на логічно зв'язані компоненти, що полегшує розуміння, тестування та зміну програми. Це особливо корисно для великих проектів, де структурованість і підтримка коду грають важливу роль.