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 *”*

Виконала:

студентка ІІI курсу

групи КВ-21

Оваденко В.М.

Перевірив:

**Київ – 2024**

**Репозиторій GitHub:** <https://github.com/Clemence25/BDZU/tree/main/РГР>

*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(Customer)

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

Призначення: збереження даних щодо клієнтів.

1. Замовлення(Order)

Атрибути: ідентифікатор замовлення, дата.

Призначення: збереження даних щодо замовлень.

1. Продукт(Product)

Атрибути: ідентифікатор товару, назва, ціна

Призначення: збереження даних щодо товарів.

1. Доставка(Delivery)

Атрибути: ідентифікатор доставки, адреса, статус доставки.

Призначення: збереження даних щодо доставок.

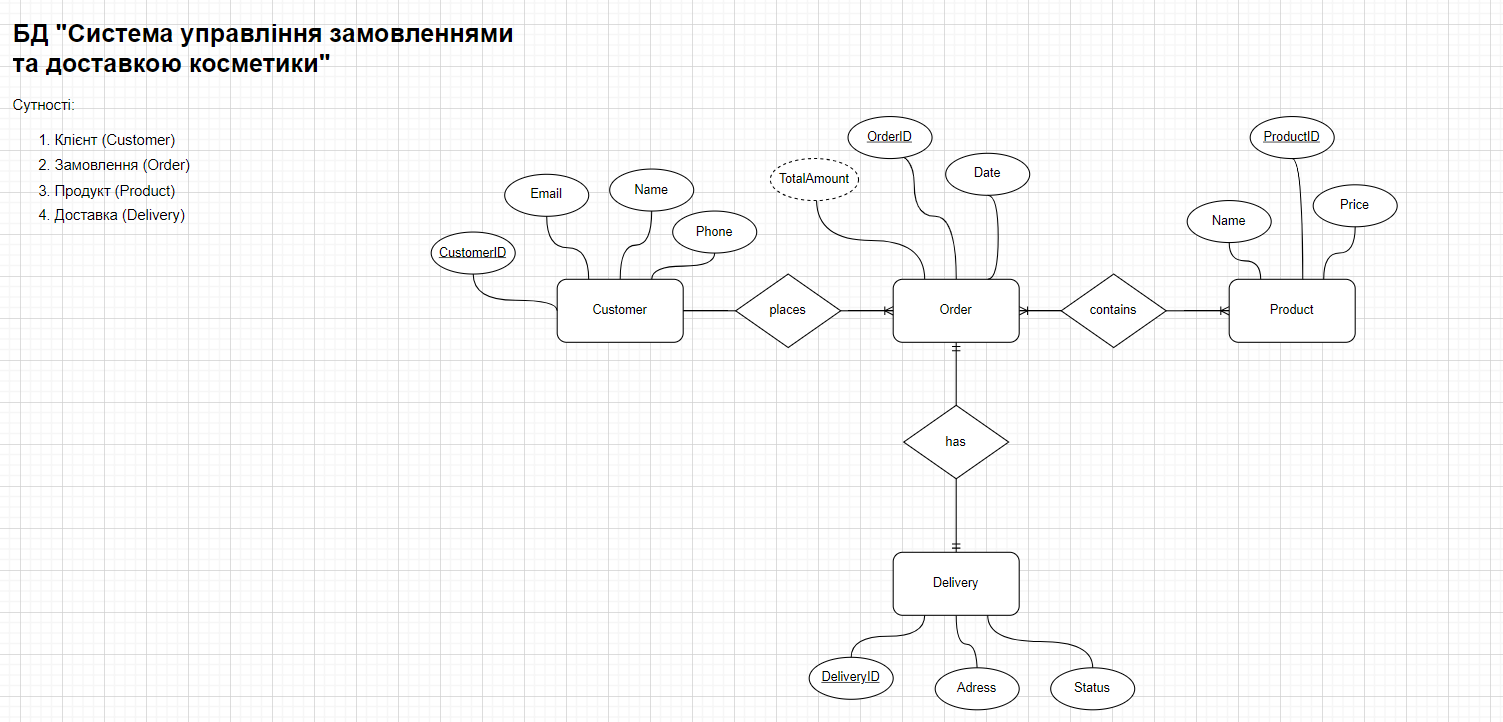
**Опис зв’язків між сутностями**

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

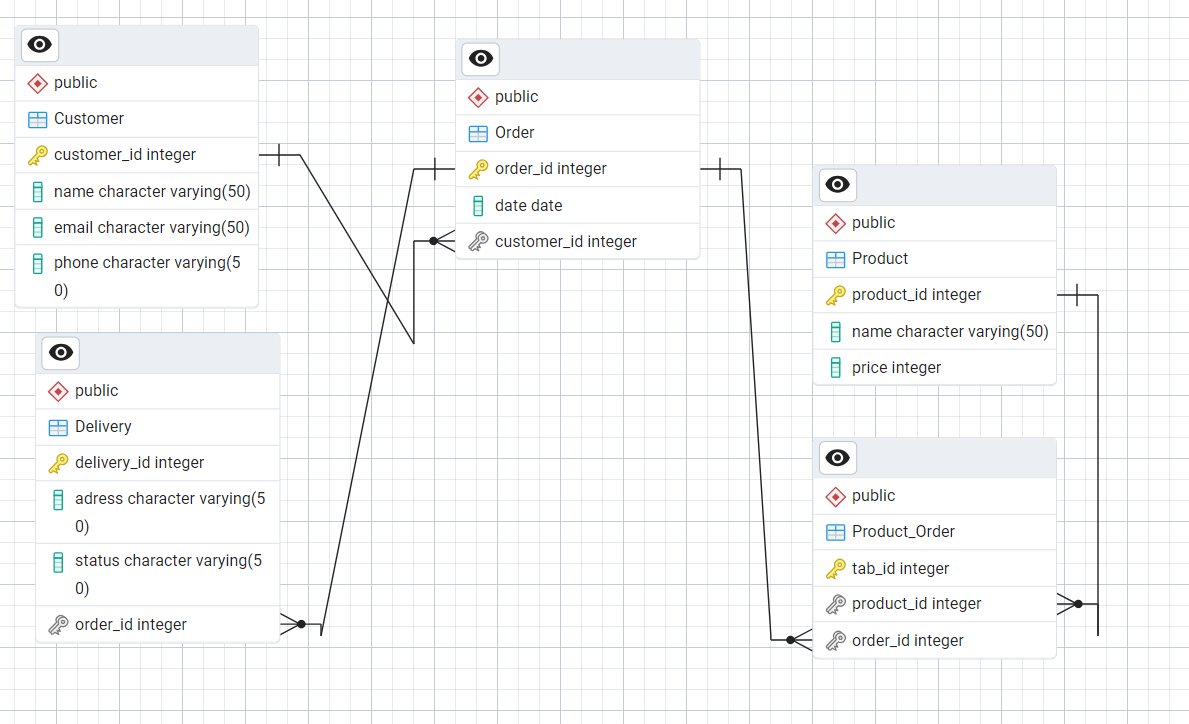
Зв’язок «Замовлення» - «Продукт» є зв’язком N:M. Одне замовлення може містити багато товарів, і один товар може входити до складу багатьох замовлень.

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

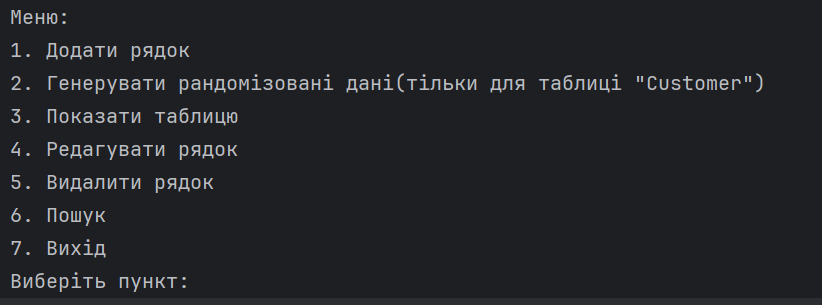
Середовище для відлагодження SQL-запитів до бази даних – PgAdmin4.

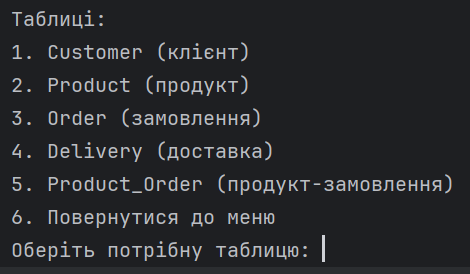
Мова програмування – Python 3.10.

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.

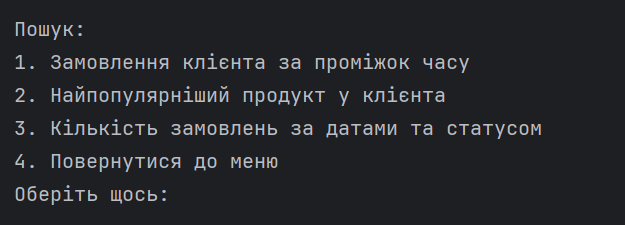
2) **Генерування «рандомізованих» даних**. Для цього пункту було обрано таблицю “Customer”. Цей пункт створений для додавання «рандомізованих» даних. Користувачу потрібно ввести число полів, скільки треба додати у таблицю.

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) **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.

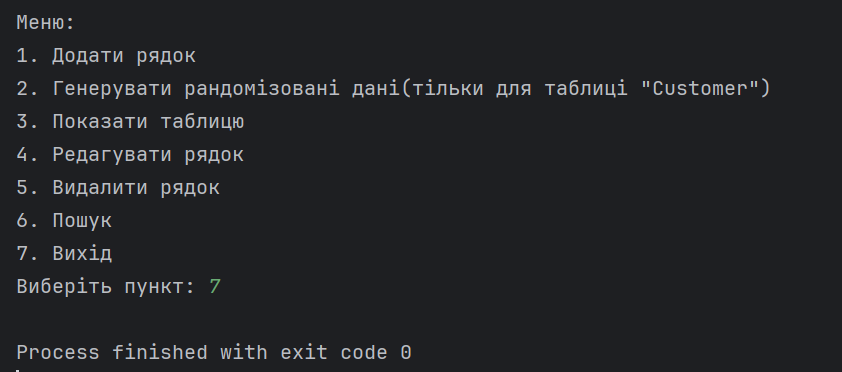
5) **Видалити рядок** служить для видалення рядку по ID у таблицях. Спочатку потрібно обрати, для якої таблиці буде відбуватися видалення рядка, після чого користувач вводить ID рядка, який потрібно видалити.

6) **Пошук** створений для пошуку за атрибутами з декількох таблиць. Пропонується 4 варіанти вибору:



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

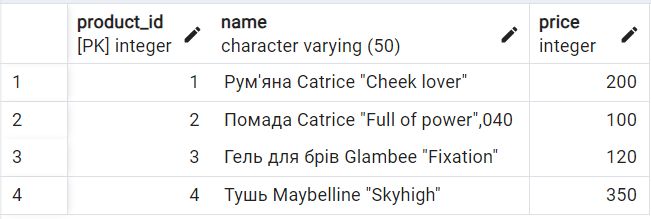
7) **Вихід** служить для виходу з програми.

****

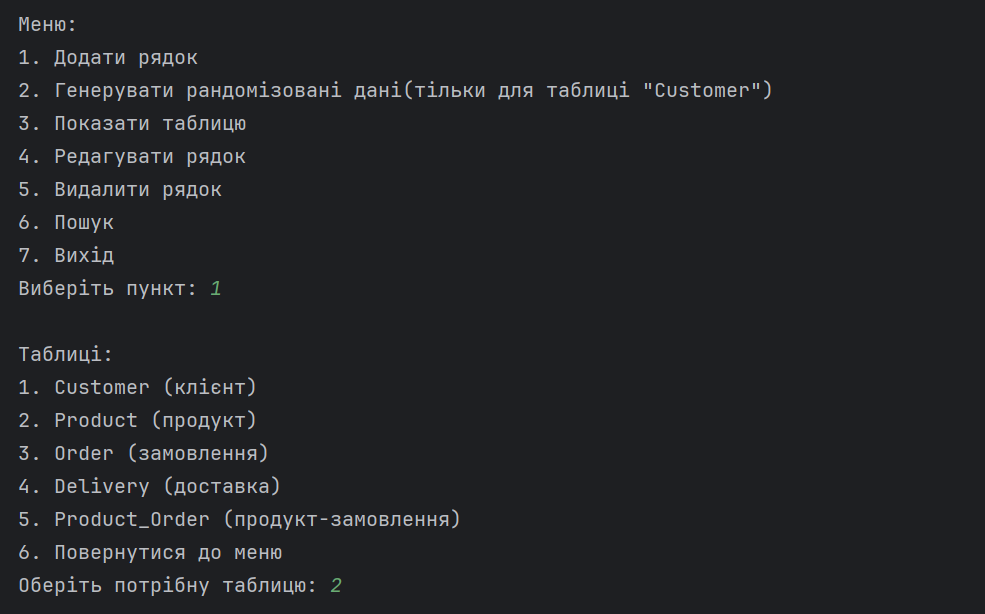
**Завдання №1**

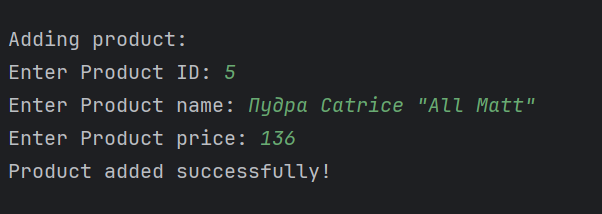
**Додавання рядка**

**Таблиця “Productr” до:**

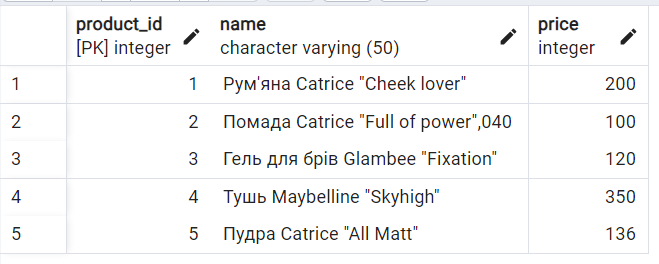


**Процес додавання рядка:**

****

****

**Таблиця “Product” після:**



**Лістинг add:**

def add\_customer(self, customer\_id, name, email, phone):  
 c = self.conn.cursor()  
 c.execute('INSERT INTO "Customer" ("customer\_id", "name", "email", "phone") VALUES (%s, %s, %s, %s)',  
 (customer\_id, name, email, phone))  
 self.conn.commit()

def add\_product(self, product\_id, name, price):  
 c = self.conn.cursor()  
 c.execute('INSERT INTO "Product" ("product\_id", "name", "price") VALUES (%s, %s, %s)',  
 (product\_id, name, price))  
 self.conn.commit()

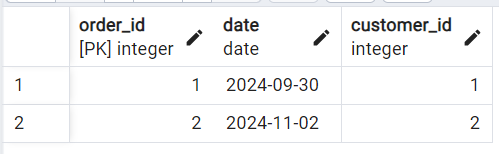
def add\_order(self, order\_id, date, customer\_id):  
 c = self.conn.cursor()  
 c.execute('INSERT INTO "Order" ("order\_id", "date", "customer\_id") VALUES (%s, %s, %s)',  
 (order\_id, date, customer\_id))  
 self.conn.commit()

def add\_delivery(self, delivery\_id, address, status, order\_id):  
 c = self.conn.cursor()  
 c.execute('INSERT INTO "Delivery" ("delivery\_id", "address", "status", "order\_id") VALUES (%s, %s, %s, %s)',  
 (delivery\_id, address, status, order\_id))  
 self.conn.commit()

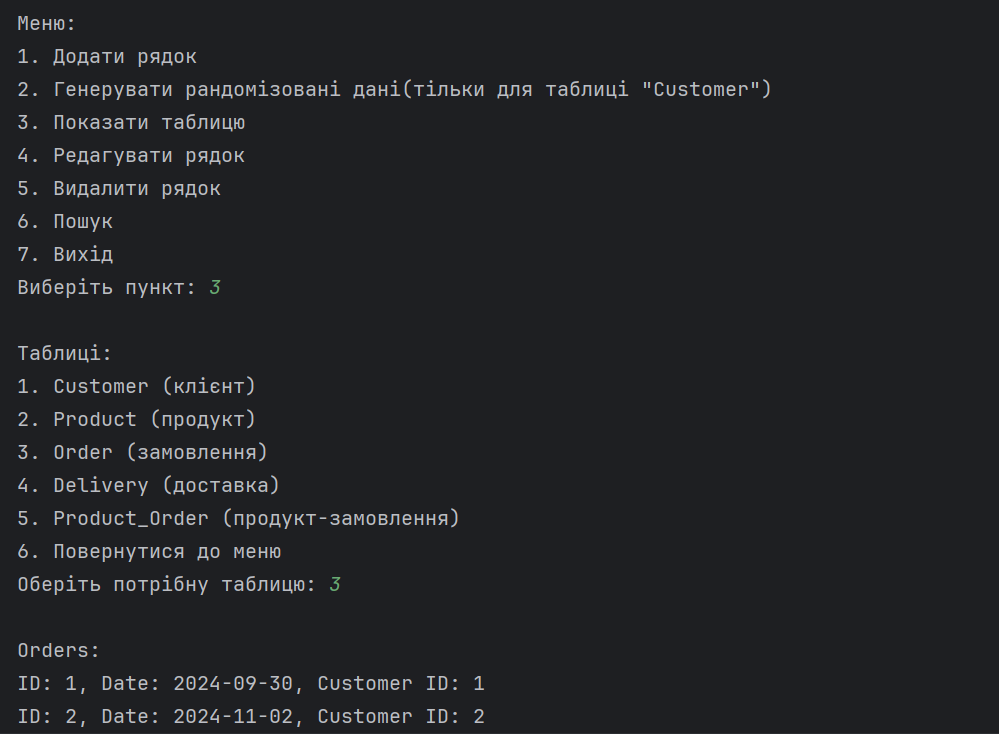
def add\_product\_order(self, tab\_id,product\_id, order\_id):  
 c = self.conn.cursor()  
 c.execute('INSERT INTO "Product\_Order" ("tab\_id","product\_id", "order\_id") VALUES (%s, %s,%s)' ,  
 (tab\_id, product\_id, order\_id))  
 self.conn.commit()

**Перегляд таблиці**

**Таблиця «Order»:**

****

**Процес перегляду таблиці:**



**Лістинг get:**

def get\_all\_customers(self):  
 with self.conn.cursor() as cursor:  
 cursor.execute('SELECT \* FROM "Customer" ORDER BY customer\_id ASC')  
 return cursor.fetchall()

def get\_all\_products(self):  
 with self.conn.cursor() as cursor:  
 cursor.execute('SELECT \* FROM "Product" ORDER BY product\_id ASC')  
 return cursor.fetchall()

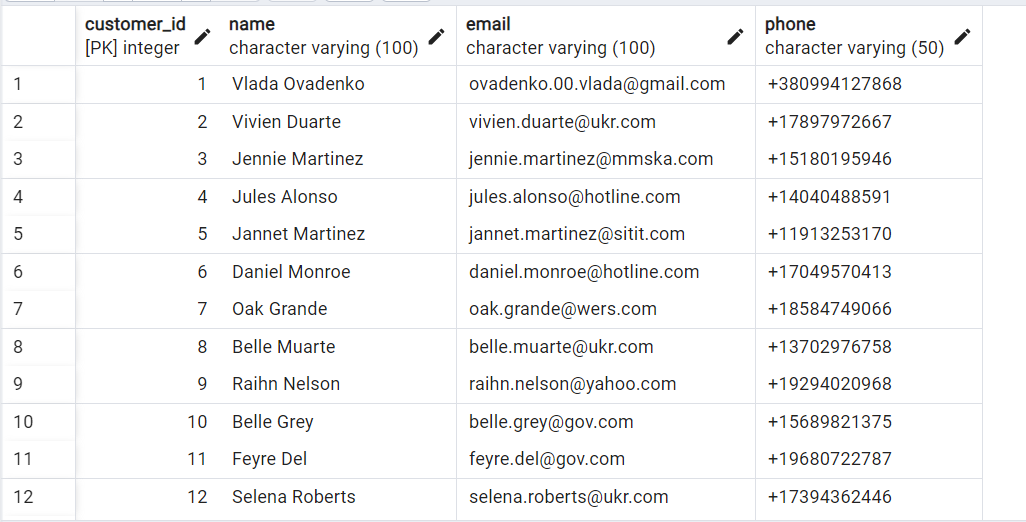
def get\_all\_orders(self):  
 with self.conn.cursor() as cursor:  
 cursor.execute('SELECT \* FROM "Order" ORDER BY order\_id ASC')  
 return cursor.fetchall()

def get\_all\_deliveries(self):  
 with self.conn.cursor() as cursor:  
 cursor.execute('SELECT \* FROM "Delivery" ORDER BY delivery\_id ASC')  
 return cursor.fetchall()

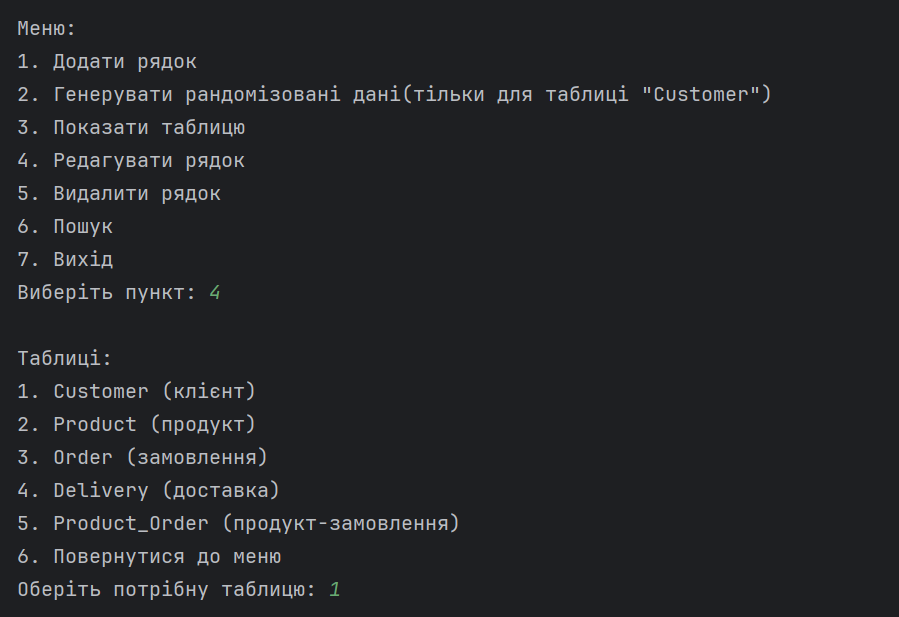
def get\_all\_product\_orders(self):  
 with self.conn.cursor() as cursor:  
 cursor.execute('SELECT \* FROM "Product\_Order" ORDER BY tab\_id ASC')  
 return cursor.fetchall()

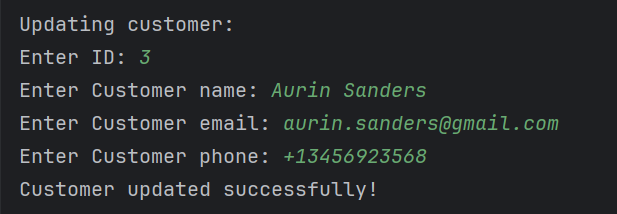
**Оновлення рядка**

**Таблиця “Customer” до:**

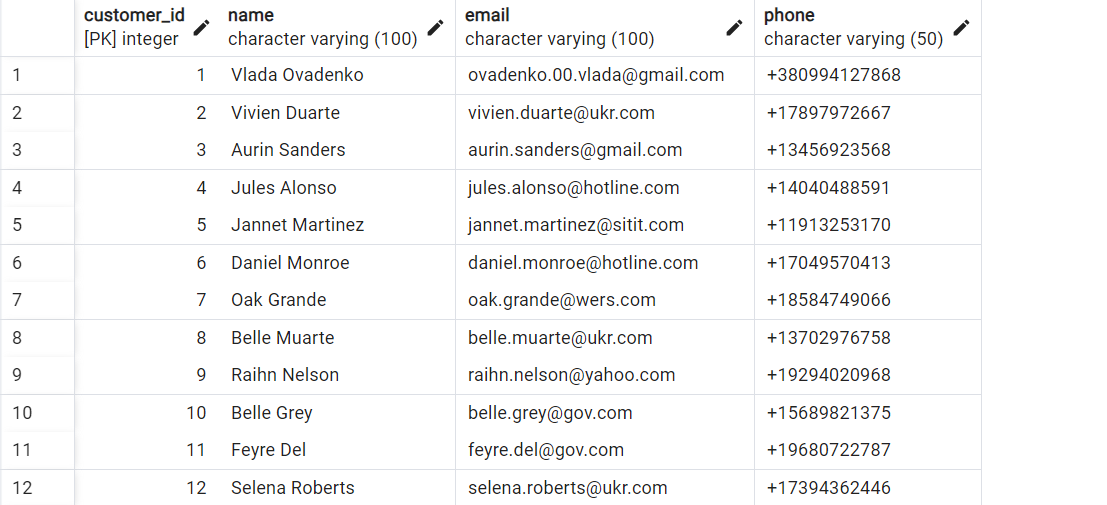


**Процес оновлення рядка:**





**Таблиця “Customer” після:**

****

**Лістинг update:**

def update\_customer(self, name, email, phone, id):  
 c = self.conn.cursor()  
 c.execute('UPDATE "Customer" SET "name"=%s, "email"=%s, "phone"=%s WHERE "customer\_id"=%s',  
 (name, email, phone, id))  
 self.conn.commit()

def update\_product(self, name, price, id):  
 c = self.conn.cursor()  
 c.execute('UPDATE "Product" SET "name"=%s, "price"=%s WHERE "product\_id"=%s',  
 (name, price, id))  
 self.conn.commit()

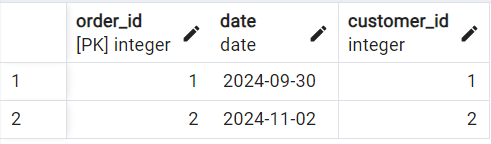
def update\_order(self, date, customer\_id, id):  
 c = self.conn.cursor()  
 c.execute('UPDATE "Order" SET "date"=%s, "customer\_id"=%s WHERE "order\_id"=%s',  
 (date,customer\_id, id))  
 self.conn.commit()

def update\_delivery(self, address, status,order\_id, id):  
 c = self.conn.cursor()  
 c.execute('UPDATE "Delivery" SET "address"=%s, "status"=%s,"order\_id"=%s WHERE "delivery\_id"=%s',  
 (address,status,order\_id, id))  
 self.conn.commit()

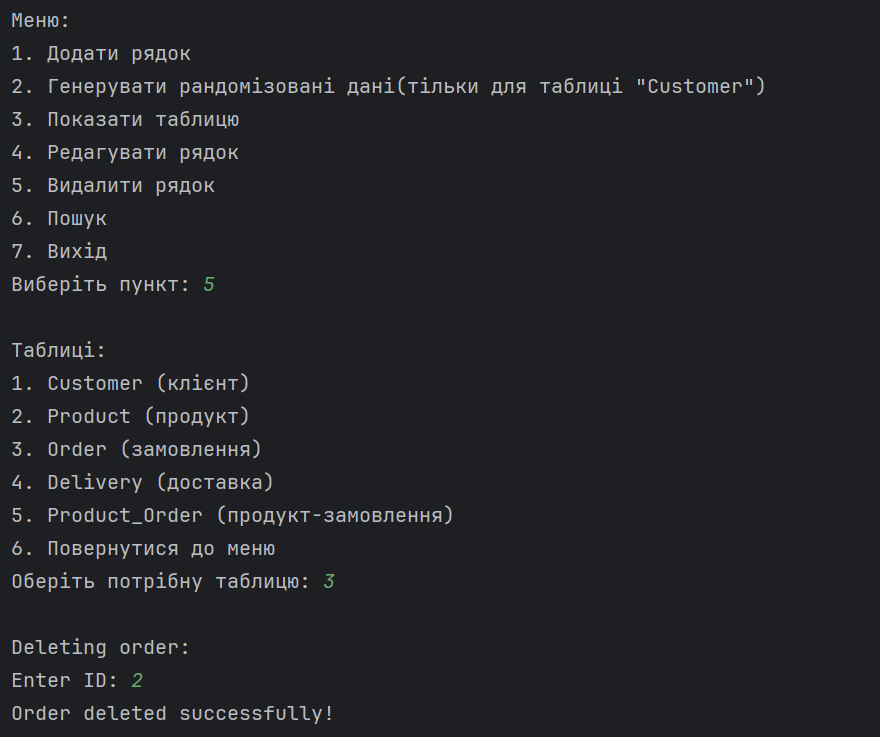
def update\_product\_order(self, product\_id, order\_id, id):  
 c = self.conn.cursor()  
 c.execute('UPDATE "Product\_Order" SET "product\_id"=%s,"order\_id"=%s WHERE "tab\_id"=%s',  
 (product\_id,order\_id, id))  
 self.conn.commit()

**Видалення рядка**

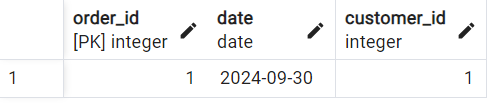
**Таблиця “Order” до:**



**Процес видалення рядка:**



**Таблиця “Order” після:**



**Лістинг delete:**

def delete\_customer(self, customer\_id):  
 c = self.conn.cursor()  
 c.execute('DELETE FROM "Customer" WHERE "customer\_id"=%s', (customer\_id,))  
 self.conn.commit()

def delete\_product(self, product\_id):  
 c = self.conn.cursor()  
 c.execute('DELETE FROM "Product" WHERE "product\_id"=%s', (product\_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\_product\_order(self, tab\_id):  
 c = self.conn.cursor()  
 c.execute('DELETE FROM "Product\_Order" WHERE "tab\_id"=%s', (tab\_id,))  
 self.conn.commit()

**Завдання №2**

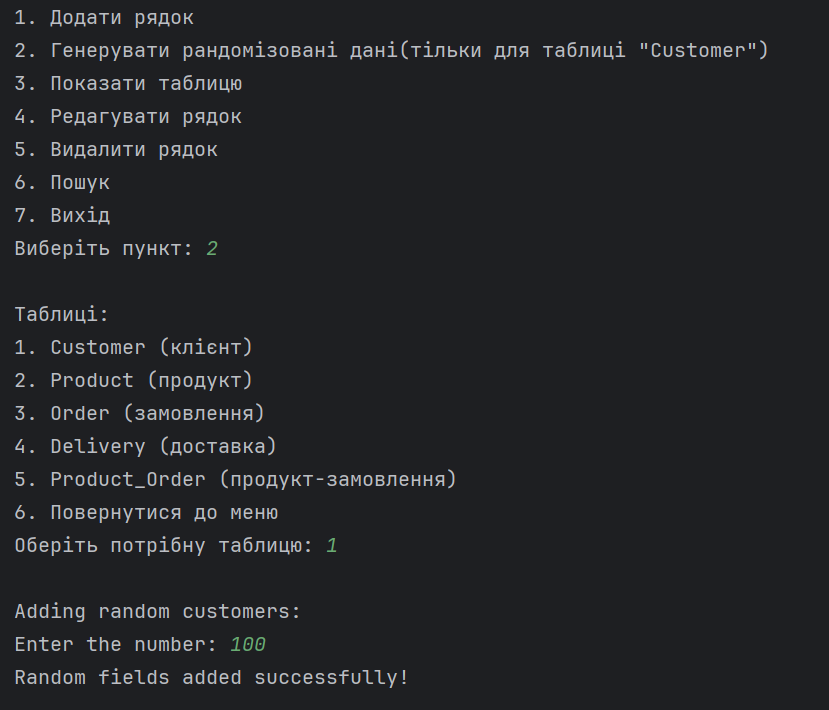
**Генерування «рандомізованих» даних**

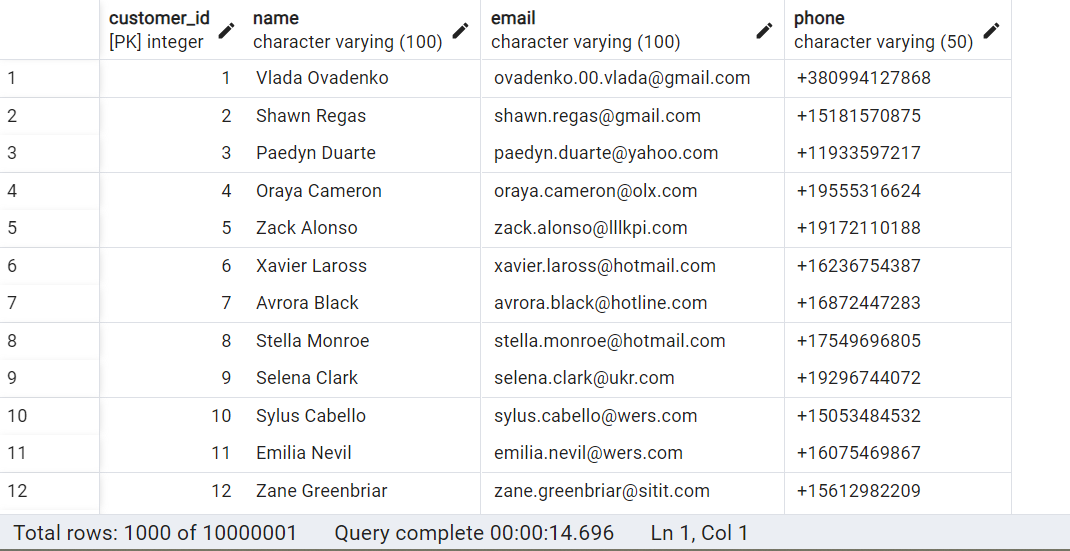
Запит, що був використаний для генерування «рандомізованих» даних для таблиці “Customer”:

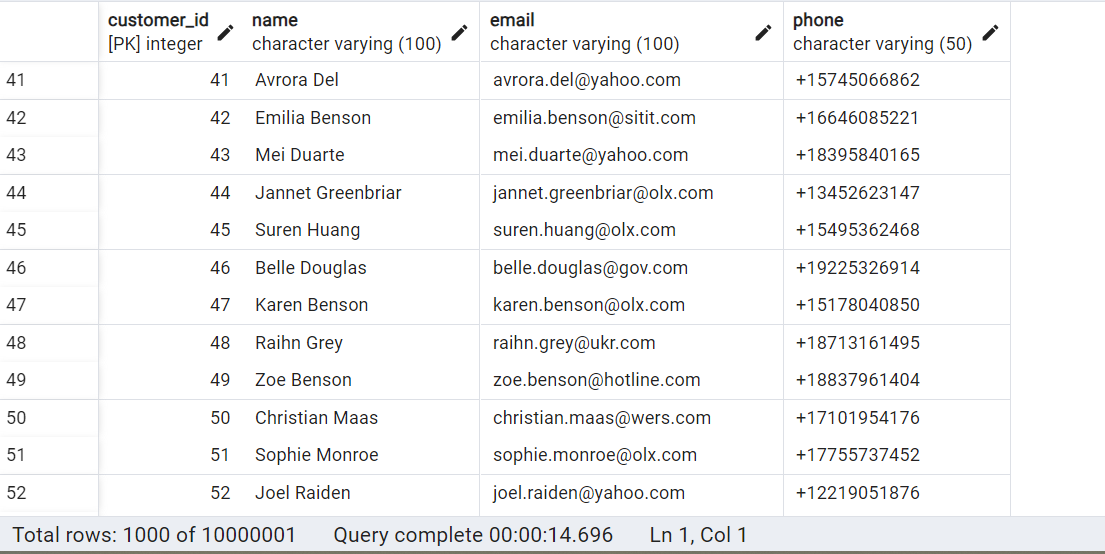
first\_names = ['John', 'Ann', 'Bob', 'Joel', 'Jannet', 'Ria','Mia','Sophie','Jennie','Stella','Jules',  
 'Ava','Bridget','Juliet','Rhys','Alex','Cardan','Jude','Vivien','Seth','Christian',  
 'Sylus','Tarin','Rhysand','Feyre','Emilia','Carissa','Oraya','Raihn','Suren','Oak',  
 'Lauren','Kai','Paedyn','Stephanie','Evangeline','Karen','Kerri','Daniel','Ashley',  
 'Aaryan','Isabel','Selena','Shawn','Avrora','James','Zoe','Belle','Zane','Xavier',  
 'Rafayel','Zack','Mei'  
  
 ]  
 last\_names = ['Smith', 'Nelson', 'Wilson', 'Parker', 'Garber', 'Roberts','Black','Chen','Harper',  
 'Ambrose','Alonso','Greenbriar','Larsen','Angelo','Grey','Laross','Martinez','Gomez',  
 'Skye','Atlantic','Cameron','Regas','Grande','Huang','Muarte','Floranse','Nevil','Cabello',  
 'Mendes','Olives','Monroe','Monroe','Jordan','Del','Rey','Clark','Spilberg','Han','Duarte',  
 'Douglas','Maas','Bridgerton','Feather','Benson','Raiden'  
 ]  
  
 query = '''  
 WITH max\_id AS (SELECT COALESCE(MAX("customer\_id"), 0) FROM public."Customer")  
INSERT INTO public."Customer" ("customer\_id", "name", "email", "phone")  
SELECT   
 (SELECT \* FROM max\_id) + row\_number() OVER () AS "customer\_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 'hotmail'  
 WHEN 2 THEN 'yahoo'  
 WHEN 3 THEN 'gov'  
 WHEN 4 THEN 'ukr'  
 WHEN 5 THEN 'wers'  
 WHEN 6 THEN 'sitit'  
 WHEN 7 THEN 'mmska'  
 WHEN 8 THEN 'hotline'  
 WHEN 9 THEN 'olx'  
 WHEN 10 THEN 'lllkpi'  
 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  
 FROM generate\_series(1, %s \* 1000)) AS names;  
 '''

*Примітка:* %s – це введене користувачем число бажаних рядків для додавання в таблицю “Customer”. Також, щоб забезпечити гнучкість коду і можливість створювати різну кількість записів, було реалізовано параметр number, що приймає введене користувачем число. Однак для автоматизації створення великої кількості записів без ручного введення 100,000, скористаємося масштабуванням. Тобто, множимо введене значення number на 1000, щоб швидко досягти великої кількості записів. Таким чином, при введенні number = 100 отримуємо 100 \* 1000 = 100,000 записів.

**Протестуємо:**





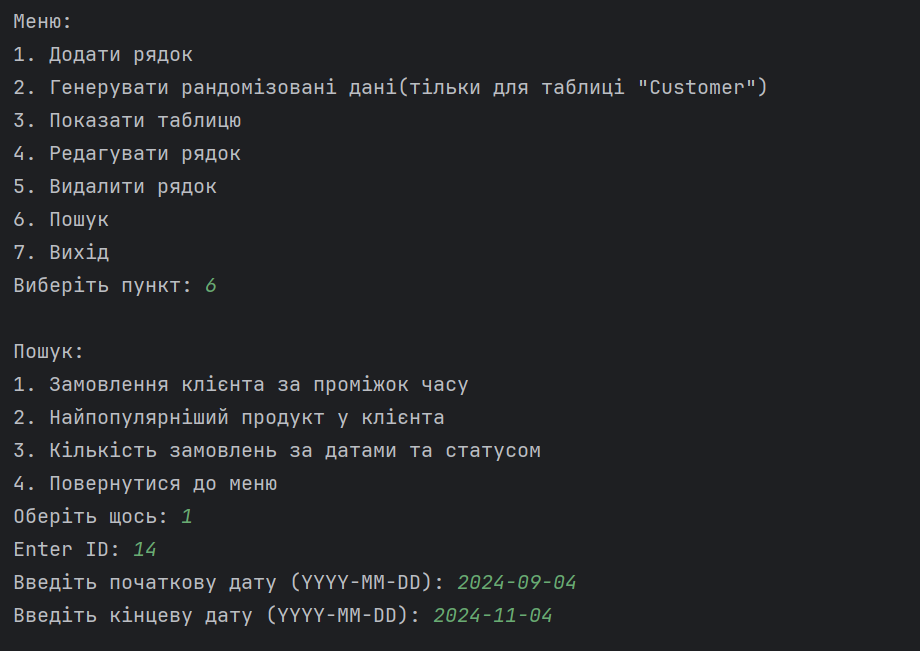


**Завдання №3**

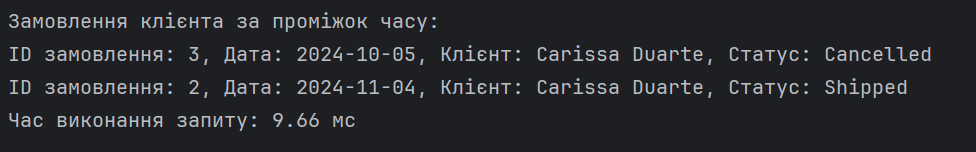
**Пошук даних**

**Запит №1**

Замовлення клієнта за проміжок часу



Цей запит створений для отримання списку замовлень певного клієнта за вибраний часовий інтервал. На виході ми отримуємо список замовлень із ID замовлення, датою замовлення, ім’ям клієнта та статусом доставки. Під час виконання цього запиту було виміряно час, який становив 9.66 мс.

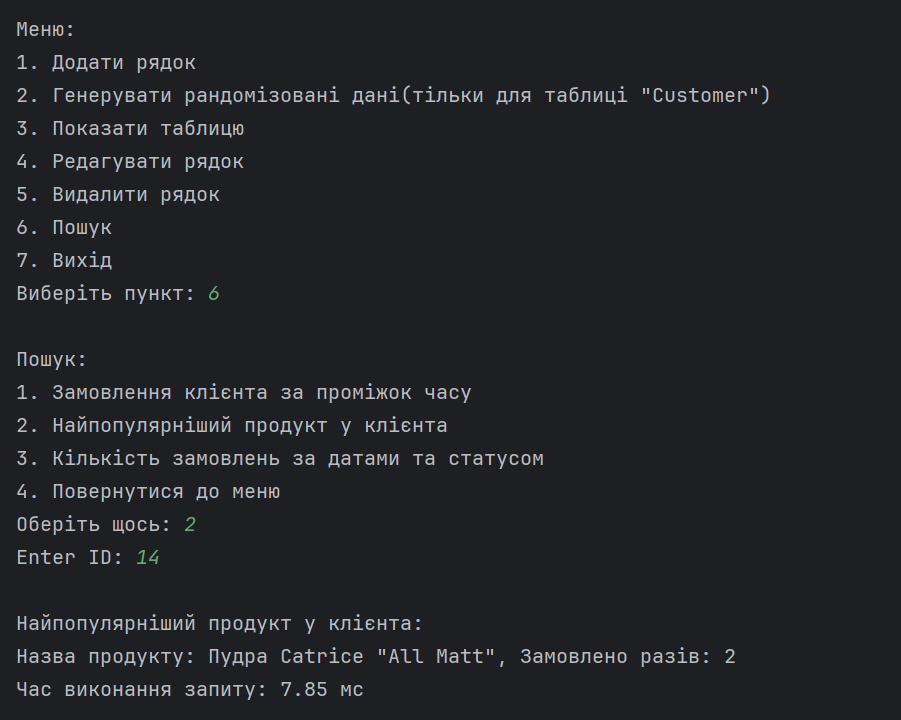


SQL запит виглядає наступним чином:

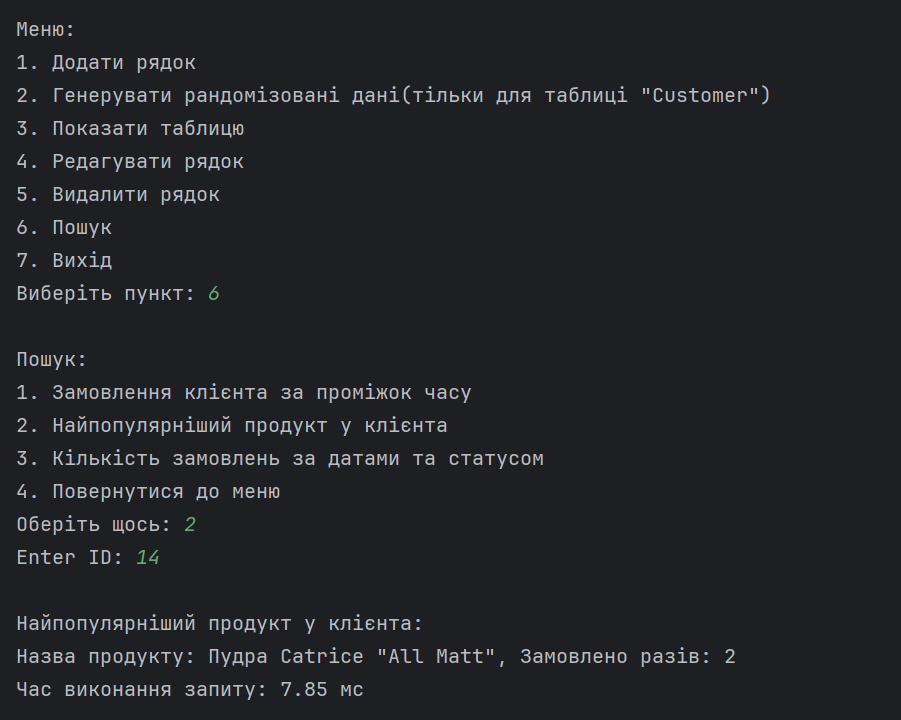
SELECT o.order\_id, o.date, c.name AS customer\_name, d.status  
FROM "Order" o  
JOIN "Customer" c ON o.customer\_id = c.customer\_id  
LEFT JOIN "Delivery" d ON o.order\_id = d.order\_id  
WHERE o.customer\_id = %s AND o.date BETWEEN %s AND %s  
ORDER BY o.date;

**Запит №2**

Найпопулярніший продукт у клієнта



Цей запит створений для визначення найпопулярнішого товару для певного клієнта. Він знаходить товар, який цей клієнт замовляв найчастіше. На виході ми отримуємо назву продукту й скільки разів його замовляли. Під час виконання цього запиту було виміряно час, який становив 7.85 мс.

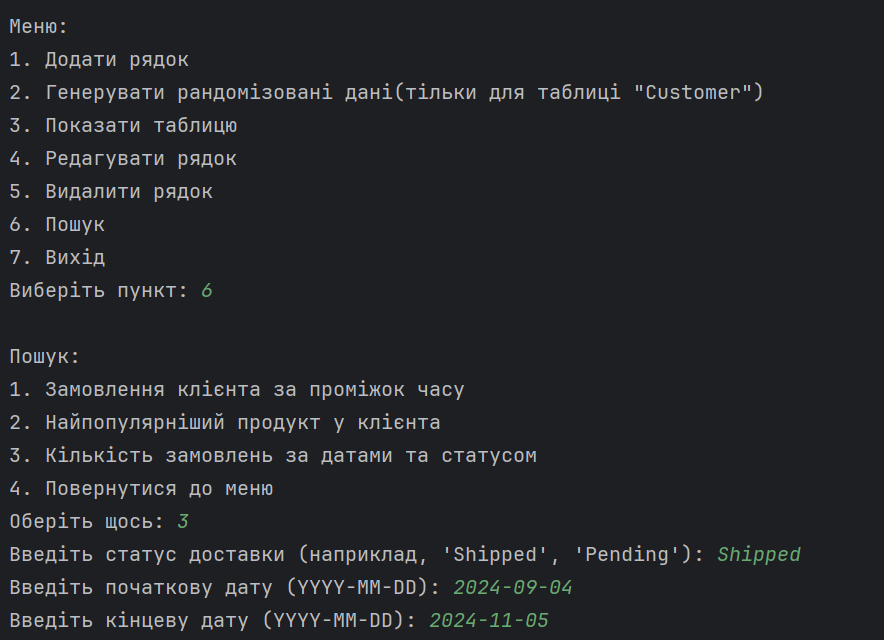


SQL запит виглядає наступним чином:

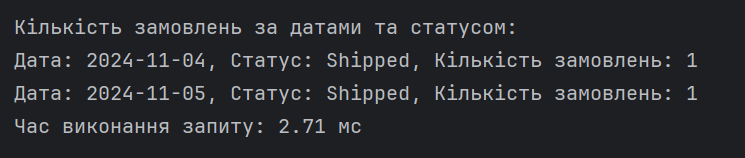
SELECT p.name AS product\_name, COUNT(po.product\_id) AS times\_ordered  
FROM "Customer" c  
JOIN "Order" o ON c.customer\_id = o.customer\_id  
JOIN "Product\_Order" po ON o.order\_id = po.order\_id  
JOIN "Product" p ON po.product\_id = p.product\_id  
WHERE c.customer\_id = %s  
GROUP BY p.name  
ORDER BY times\_ordered DESC  
LIMIT 1;

**Запит №3**

Кількість замовлень за датами та статусом

****

Цей запит створений для отримання кількості замовлень за датою і статусом доставки протягом заданого проміжку часу. Він надає статистику про кількість замовлень з певним статусом доставки для кожної дати в межах заданого інтервалу. На виході отримуємо дату, статус та кількість замовлень зроблених в цей день й з цим статусом. Під час виконання цього запиту було виміряно час, який становив 2.71 мс.

****

SQL запит виглядає наступним чином:

SELECT o.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.date BETWEEN %s AND %s  
GROUP BY o.date, d.status  
ORDER BY o.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.