НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих**

**комп’ютерних систем**

**РОЗРАХУНКОВО-ГРАФІЧНА РОБОТА**

**«Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL»**

з дисципліни

**«Бази даних та засоби управління»**

Виконав:

студент III курсу

групи КВ-12

Чернишков О.О.

Перевірив:

Павловський В.І.

*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).

*Посилання на репозиторій GitHub:* <https://github.com/SashaBlack01/DBLabs>

*Контакт студента в Телеграм:* @Sasha\_Black01

**Сутності**

Згідно з обраною галуззю для побудови бази даних було виділено наступні сутності:

* Посилка (Parcel) з атрибутами: номер посилки, статус, дата створення, вартість, ідентифікатор кур’єра, ідентифікатор отримувача, ідентифікатор складу. Призначення: збереження даних щодо посилок.
* Отримувач (Recipient) з атрибутами: ідентифікатор, ім’я, адреса. Призначення: збереження даних отримувачів посилок.
* Відправник (Sender) з атрибутами: ідентифікатор, ім’я, адреса. Призначення: збереження даних відправників посилок.
* Кур’єр (Courier) з атрибутами: службовий номер, ім’я, номер телефону.
* Склад (Warehouse) з атрибутами: номер складу, адреса, номер телефону.

**Опис зв’язків**

Посилка має найбільшу кількість зв’язків з іншими сутностями:

* Кожна посилка має зв’язок з кур’єром який доставляє посилку, причому тільки з одним, але кожен кур’єр може доставляти декілька посилок, тому зв’язок 1:N.
* Кожна посилка має зв’язок з отримувачем, до якого вона прямує, при чому тільки з одним, але до отримувача може прямувати багато посилок, тому зв’язок 1:N.
* Кожна посилка має зв’язок з відправником, який створив та надіслав посилку, при чому відправник може бути тільки один для кожної посилки, тому зв’язок 1:N.
* Кожна посилка має зв’язок зі складом, на якому вона тимчасово зберігається, при чому склад для конкретної посилки може бути тільки один, але на одному складі може бути декілька посилок, тому зв’язок 1:N.
* Кожен кур’єр має зв’язок зі складом, до якого від закріплений, щоб доставляти посилки з конкретного складу, при чому кожен кур’єр може бути закріпленим за декількома складами, а за кожним складом може бути закріплено декілька кур’єрів, тому зв’язок M:N.

Графічне подання моделі «сутність зв’язок» побудованої за нотацією Чена представлена на рисунку 1.

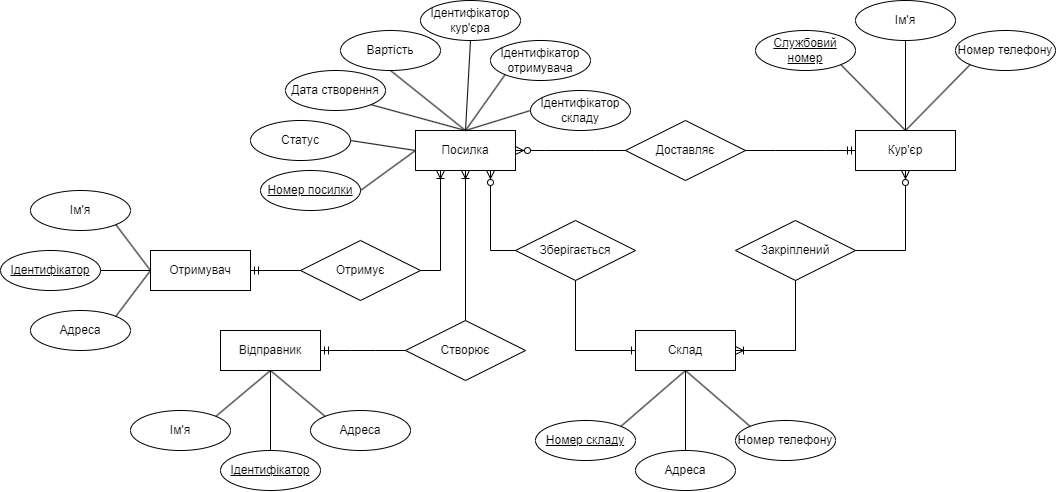


Рисунок 1 – ER-діаграма, побудована за нотацією Чена

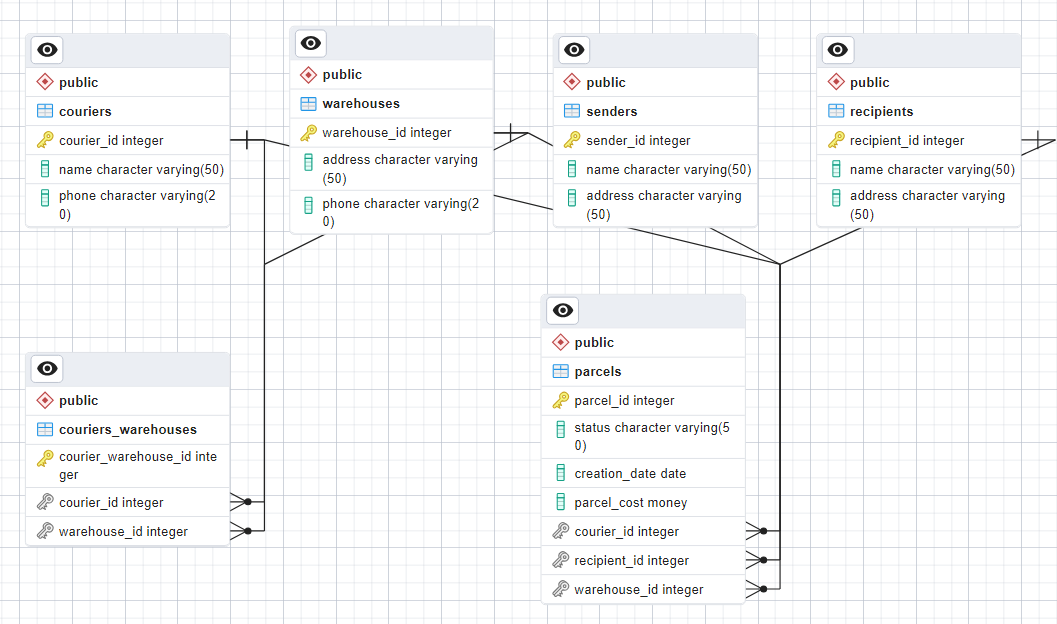
****

Рисунок 2 – Схема бази даних у графічному вигляді

При виконанні розрахунково-графічної роботи використовувалися наступні компоненти:

* 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.

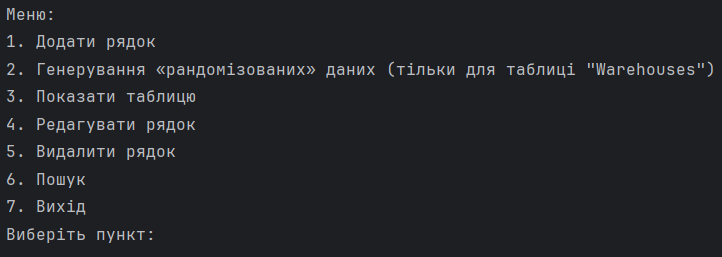


Рисунок 3 – Меню користувача з описом

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 “warehouses”. 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.

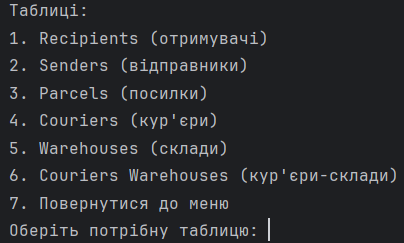


Рисунок 4 – Відображення пункту «Показати тиблицю»

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:

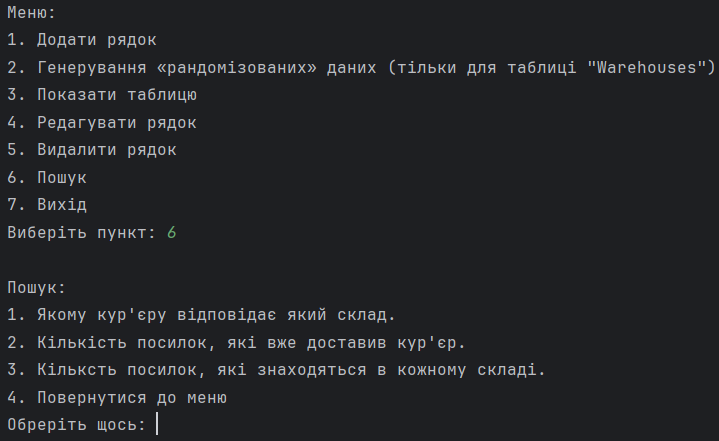


Рисунок 5 – Відображення пункту «Пошук»

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

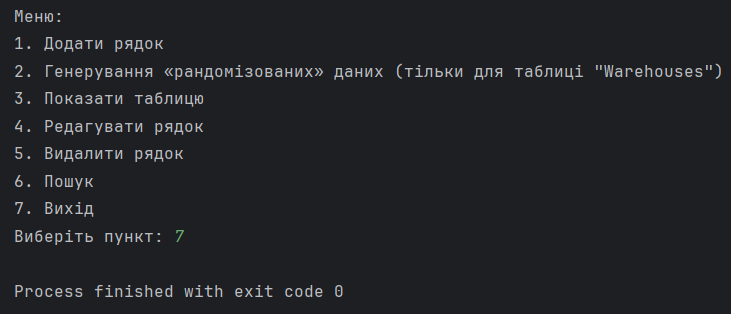
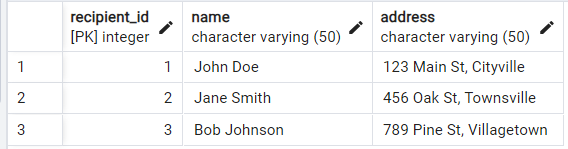


Рисунок 6 – Відображення роботи пункту «Вихід»

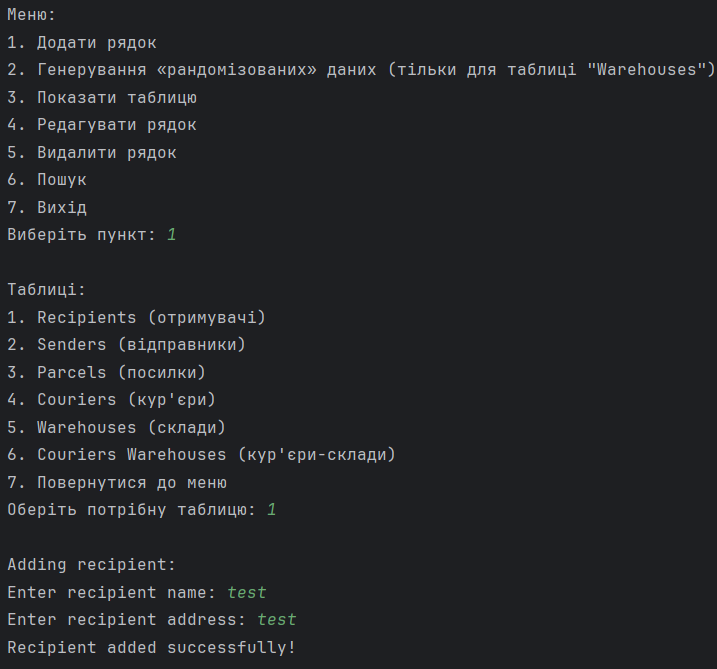
**Завдання 1**

**Додавання рядка**

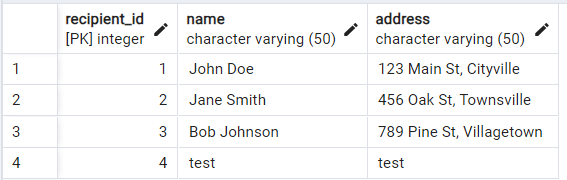
***Таблиця “recipients” до:***

******

***Процес додавання рядка:***

******

***Таблиця “recipients” після:***

******

***Лістинг add:***

**def** add\_recipient(self, name, address):

c = self.conn.cursor()

c.execute('INSERT INTO recipients (name, address) VALUES (%s, %s)', (name, address))

self.conn.commit()

**def** add\_sender(self, name, address):

c = self.conn.cursor()

c.execute('INSERT INTO senders (name, address) VALUES (%s, %s)', (name, address))

self.conn.commit()

**def** add\_parcel(self, status, creation\_date, parcel\_cost, courier\_id, recipient\_id, warehouse\_id):

c = self.conn.cursor()

c.execute('INSERT INTO parcels (status, creation\_date, parcel\_cost, courier\_id, recipient\_id, warehouse\_id) VALUES (%s, %s, %s, %s, %s, %s)', (status, creation\_date, parcel\_cost, courier\_id, recipient\_id, warehouse\_id))

self.conn.commit()

**def** add\_courier(self, name, phone):

c = self.conn.cursor()

c.execute('INSERT INTO couriers (name, phone) VALUES (%s, %s)', (name, phone))

self.conn.commit()

**def** add\_warehouse(self, address, phone):

c = self.conn.cursor()

c.execute('INSERT INTO warehouses (address, phone) VALUES (%s, %s)', (address, phone))

self.conn.commit()

**def** add\_courier\_warehouse(self, courier\_id, warehouse\_id):

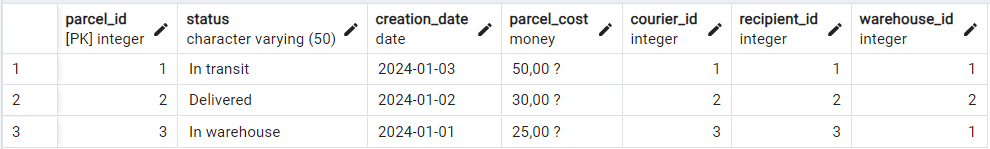
c = self.conn.cursor()

c.execute('INSERT INTO couriers\_warehouses (courier\_id, warehouse\_id) VALUES (%s, %s)', (courier\_id, warehouse\_id))

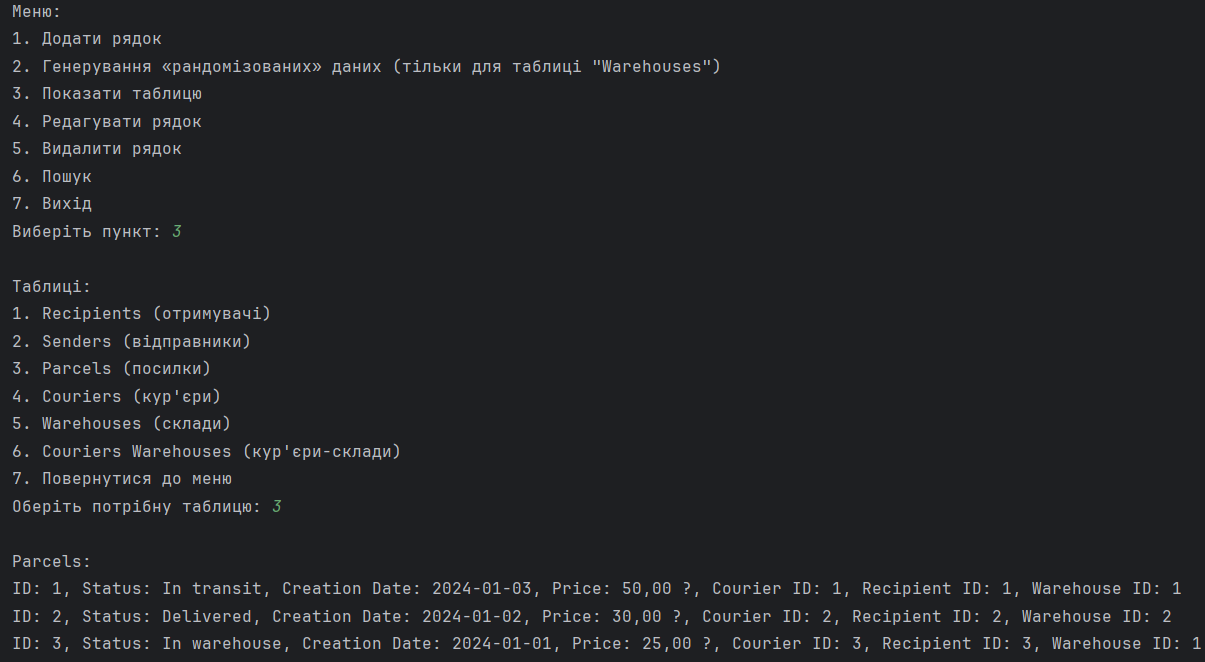
self.conn.commit()

**Перегляд таблиці**

***Таблиця “parcels”:***

******

***Процес перегляду таблиці:***

******

***Лістинг show:***

**def** get\_all\_recipients(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM recipients')

**return** c.fetchall()

**def** get\_all\_senders(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM senders')

**return** c.fetchall()

**def** get\_all\_parcels(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM parcels')

**return** c.fetchall()

**def** get\_all\_couriers(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM couriers')

**return** c.fetchall()

**def** get\_all\_warehouses(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM warehouses')

**return** c.fetchall()

**def** get\_all\_couriers\_warehouses(self):

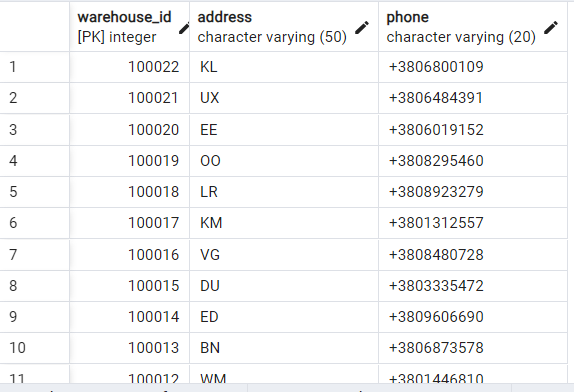
c = self.conn.cursor()

c.execute('SELECT \* FROM couriers\_warehouses')

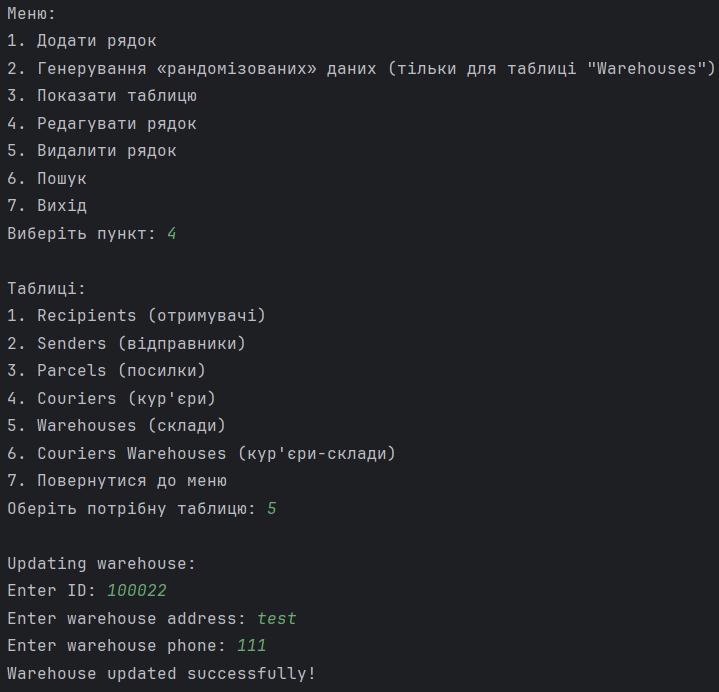
**return** c.fetchall()

**Оновлення рядка**

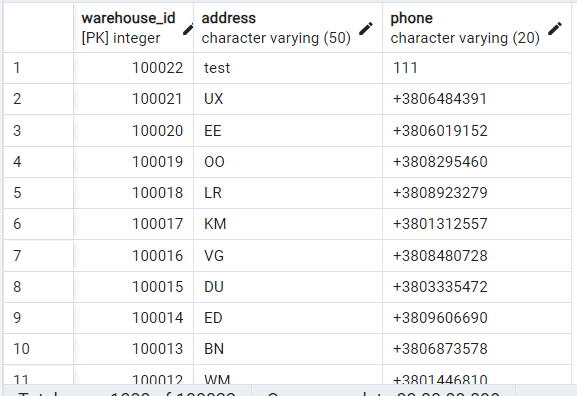
***Таблиця “warehouses” до:***

******

***Процес оновлення рядка:***

******

***Таблиця “warehouses” після:***

******

***Лістинг update:***

**def** update\_recipient(self, name, address, id):

c = self.conn.cursor()

c.execute('UPDATE recipients SET name=%s, address=%s WHERE recipient\_id=%s', (name, address, id))

self.conn.commit()

**def** update\_sender(self, name, address, id):

c = self.conn.cursor()

c.execute('UPDATE senders SET name=%s, address=%s WHERE sender\_id=%s', (name, address, id))

self.conn.commit()

**def** update\_parcel(self, status, creation\_date, parcel\_cost, courier\_id, recipient\_id, warehouse\_id, id):

c = self.conn.cursor()

c.execute('UPDATE parcels SET status=%s, creation\_date=%s, parcel\_cost=%s, courier\_id=%s, recipient\_id=%s, warehouse\_id=%s WHERE parcel\_id=%s', (status, creation\_date, parcel\_cost, courier\_id, recipient\_id, warehouse\_id, id))

self.conn.commit()

**def** update\_courier(self, name, phone, id):

c = self.conn.cursor()

c.execute('UPDATE couriers SET name=%s, phone=%s WHERE courier\_id=%s', (name, phone, id))

self.conn.commit()

**def** update\_warehouse(self, address, phone, id):

c = self.conn.cursor()

c.execute('UPDATE warehouses SET address=%s, phone=%s WHERE warehouse\_id=%s', (address, phone, id))

self.conn.commit()

**def** update\_courier\_warehouse(self, courier\_id, warehouse\_id, id):

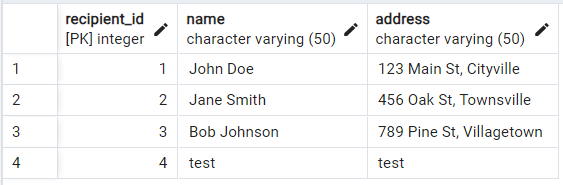
c = self.conn.cursor()

c.execute('UPDATE warehouses SET courier\_id=%s, warehouse\_id=%s WHERE courier\_warehouse\_id=%s', (courier\_id, warehouse\_id, id))

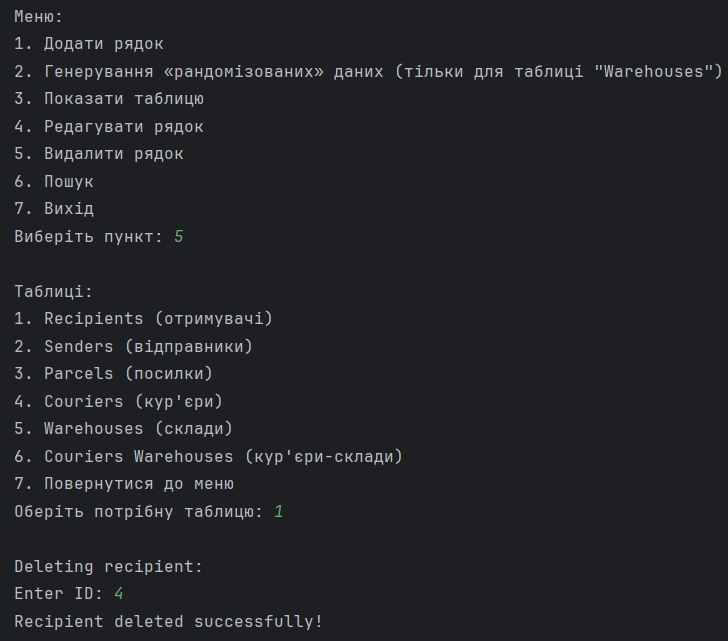
self.conn.commit()

**Видалення рядка**

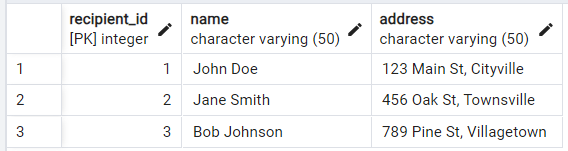
***Таблиця “recipients” до:***

******

***Процес видалення рядка:***

******

***Таблиця “recipients” після:***

******

***Лістинг delete:***

**def** delete\_recipient(self, recipient\_id):

c = self.conn.cursor()

c.execute('DELETE FROM recipients WHERE recipient\_id=%s', (recipient\_id,))

self.conn.commit()

**def** delete\_sender(self, sender\_id):

c = self.conn.cursor()

c.execute('DELETE FROM senders WHERE sender\_id=%s', (sender\_id,))

self.conn.commit()

**def** delete\_parcel(self, parcel\_id):

c = self.conn.cursor()

c.execute('DELETE FROM parcels WHERE parcel\_id=%s', (parcel\_id,))

self.conn.commit()

**def** delete\_courier(self, courier\_id):

c = self.conn.cursor()

c.execute('DELETE FROM couriers WHERE courier\_id=%s', (courier\_id,))

self.conn.commit()

**def** delete\_warehouse(self, warehouse\_id):

c = self.conn.cursor()

c.execute('DELETE FROM warehouses WHERE warehouse\_id=%s', (warehouse\_id,))

self.conn.commit()

**def** delete\_courier\_warehouse(self, courier\_warehouse\_id):

c = self.conn.cursor()

c.execute('DELETE FROM couriers\_warehouses WHERE courier\_warehouse\_id=%s', (courier\_warehouse\_id,))

self.conn.commit()

**Завдання 2**

**Генерування «рандомізованих» даних:**

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

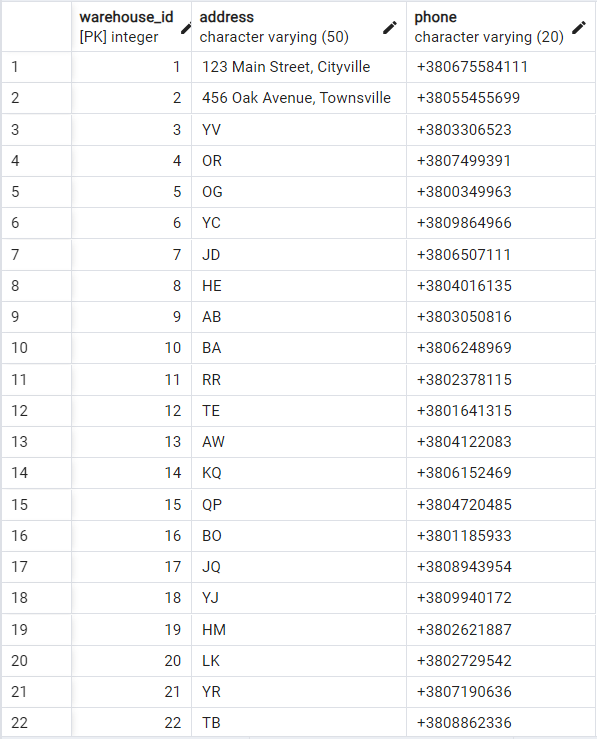
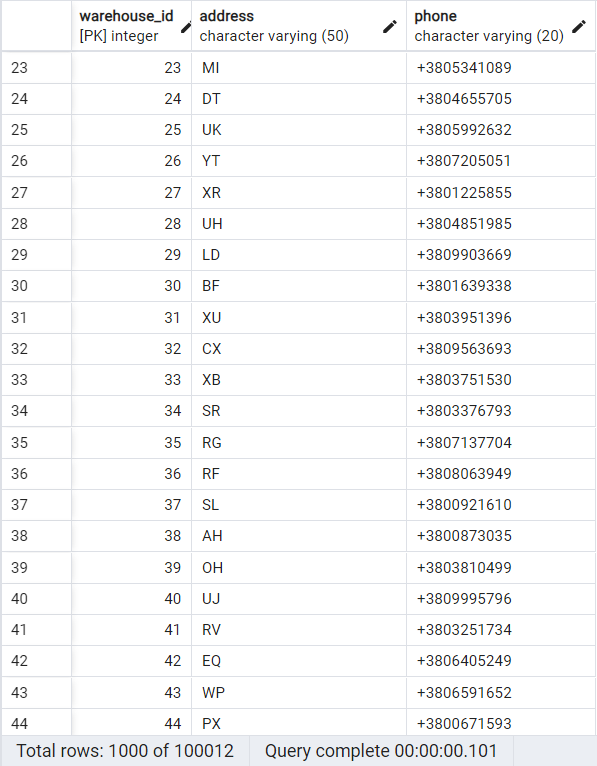
**INSERT** **INTO** warehouses (address, phone)

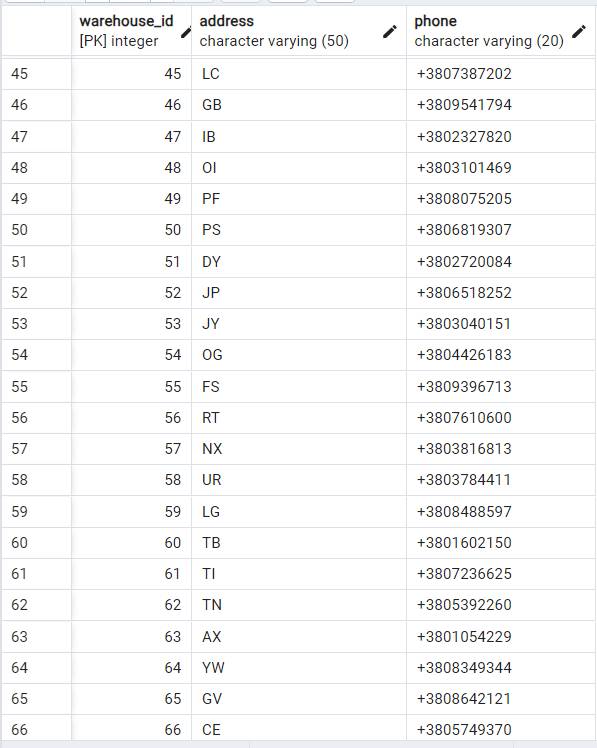
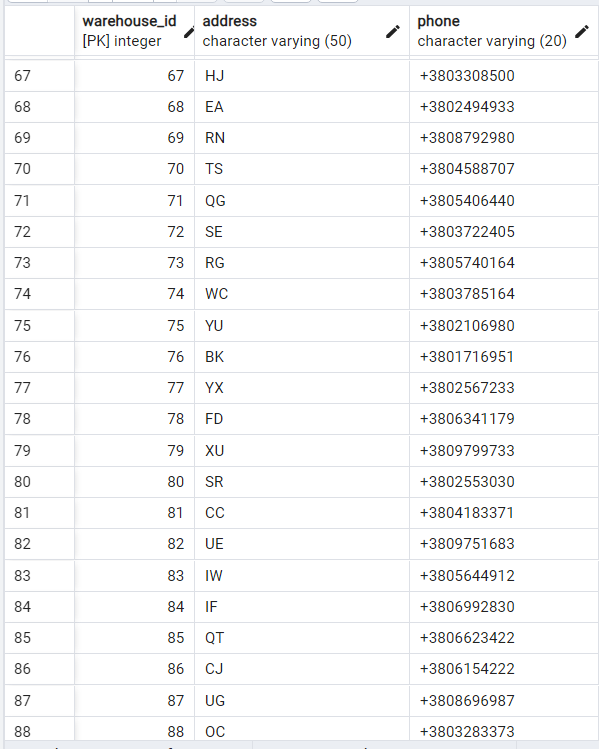
**SELECT** chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int), '+380' || lpad(trunc(random() \* 10000000)::text, 7, '0')

**FROM** generate\_series(1, %s);

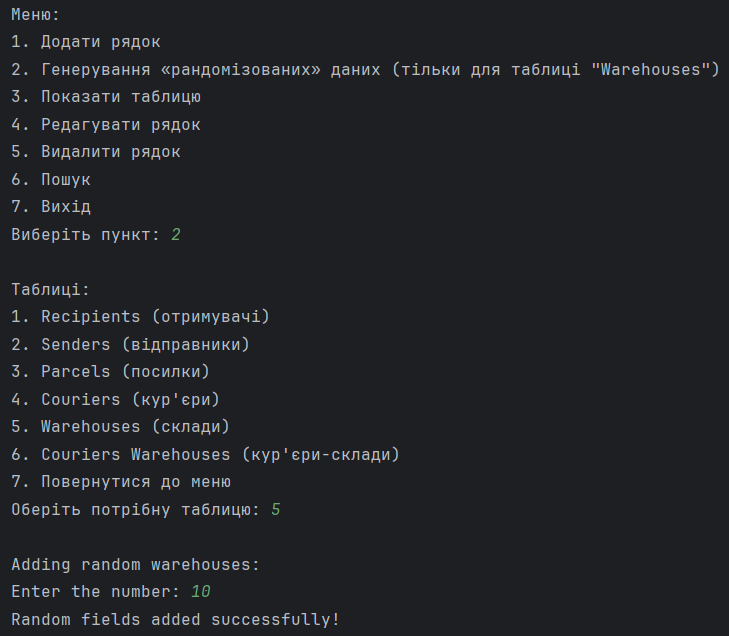
Примітка: %s – це введене користувачем число бажаних рядків для додавання в таблицю “warehouses”. Протестуємо:

***Таблиця “warehouses” до:***

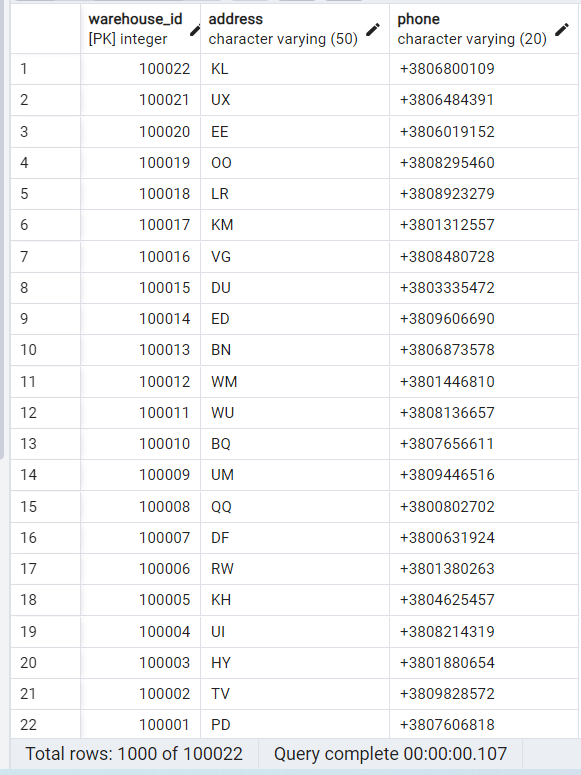
*** ***

****** ******

***Процес додавання «рандомізованих» даних:***

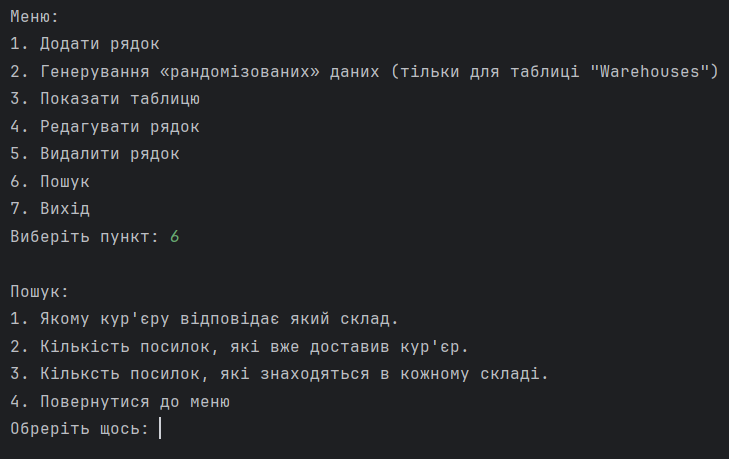
******

***Таблиця “warehouses” після:***

****

**Завдання 3**

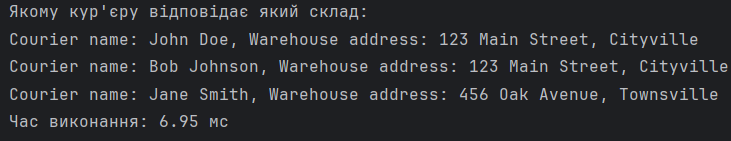
**Пошук даних**

****

***Запит №1***

Якому кур'єру відповідає який склад.

Цей запит створений для отримання списку кур’єрів та складів, які закріплені за ними. На виході отримуємо список кур’єрів та адреси їх складів. Також було засічено час виконання цього запиту. Він становив 6.95 мілісекунд.



SQL запит виглядає наступним чином:

**SELECT** couriers.name **AS** courier\_name, warehouses.address **AS** warehouse\_address

**FROM** couriers

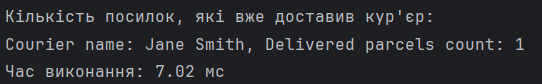
**JOIN** couriers\_warehouses **ON** couriers.courier\_id = couriers\_warehouses.courier\_id

**JOIN** warehouses **ON** couriers\_warehouses.warehouse\_id = warehouses.warehouse\_id;

***Запит №2***

Кількість посилок, які вже доставив кур'єр.

Цей запит створений для отримання кількості посилок, які вже були доставлені кур’єром. На виході отримуємо список кур’єрів з їх іменами та прізвищами та числом доставлених посилок. Також було засічено час виконання цього запиту. Він становив 7.02 мілісекунд.



SQL запит виглядає наступним чином:

**SELECT** couriers.name **AS** courier\_name, COUNT(parcels.parcel\_id) **AS** delivered\_count

**FROM** couriers

LEFT **JOIN** parcels **ON** couriers.courier\_id = parcels.courier\_id

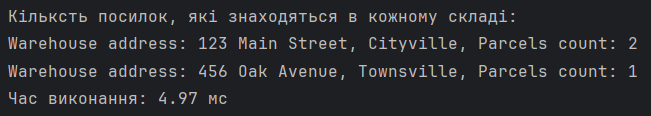
**WHERE** parcels.**status** = 'Delivered'

**GROUP** **BY** couriers.courier\_id, couriers.name;

***Запит №3***

Кільксть посилок, які знаходяться в кожному складі.

Цей запит створений для отримання кількості посилок, що знаходяться у кожному складі. На виході отримуємо список складів з їх адресами та кількість посилок, що зберігаються у них. Також було засічено час виконання цього запиту. Він становив 4.97 мілісекунд.



SQL запит виглядає наступним чином:

**SELECT** warehouses.address **AS** warehouse\_address, COUNT(parcels.parcel\_id) **AS** parcel\_count

**FROM** parcels

**JOIN** warehouses **ON** parcels.warehouse\_id = warehouses.warehouse\_id

**GROUP** **BY** warehouses.warehouse\_id, warehouses.address;

**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 до View.

**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.

**Код програми:**

**main.py**

**from** controller **import** Controller

**if** \_\_name\_\_ == "\_\_main\_\_":

controller = Controller()

controller.run()

**model.py**

**import** psycopg2

**class** Model:

**def** \_\_init\_\_(self):

self.conn = psycopg2.connect(

dbname='courier\_service',

user='postgres',

password='1111',

host='localhost',

port=3000

)

**def** add\_recipient(self, name, address):

c = self.conn.cursor()

c.execute('INSERT INTO recipients (name, address) VALUES (%s, %s)', (name, address))

self.conn.commit()

**def** add\_sender(self, name, address):

c = self.conn.cursor()

c.execute('INSERT INTO senders (name, address) VALUES (%s, %s)', (name, address))

self.conn.commit()

**def** add\_parcel(self, status, creation\_date, parcel\_cost, courier\_id, recipient\_id, warehouse\_id):

c = self.conn.cursor()

c.execute('INSERT INTO parcels (status, creation\_date, parcel\_cost, courier\_id, recipient\_id, warehouse\_id) VALUES (%s, %s, %s, %s, %s, %s)', (status, creation\_date, parcel\_cost, courier\_id, recipient\_id, warehouse\_id))

self.conn.commit()

**def** add\_courier(self, name, phone):

c = self.conn.cursor()

c.execute('INSERT INTO couriers (name, phone) VALUES (%s, %s)', (name, phone))

self.conn.commit()

**def** add\_warehouse(self, address, phone):

c = self.conn.cursor()

c.execute('INSERT INTO warehouses (address, phone) VALUES (%s, %s)', (address, phone))

self.conn.commit()

**def** add\_courier\_warehouse(self, courier\_id, warehouse\_id):

c = self.conn.cursor()

c.execute('INSERT INTO couriers\_warehouses (courier\_id, warehouse\_id) VALUES (%s, %s)', (courier\_id, warehouse\_id))

self.conn.commit()

**def** get\_all\_recipients(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM recipients')

**return** c.fetchall()

**def** get\_all\_senders(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM senders')

**return** c.fetchall()

**def** get\_all\_parcels(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM parcels')

**return** c.fetchall()

**def** get\_all\_couriers(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM couriers')

**return** c.fetchall()

**def** get\_all\_warehouses(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM warehouses')

**return** c.fetchall()

**def** get\_all\_couriers\_warehouses(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM couriers\_warehouses')

**return** c.fetchall()

**def** get\_all\_couriers\_with\_warehouses(self):

c = self.conn.cursor()

c.execute('SELECT couriers.name AS courier\_name, warehouses.address AS warehouse\_address FROM couriers JOIN couriers\_warehouses ON couriers.courier\_id = couriers\_warehouses.courier\_id JOIN warehouses ON couriers\_warehouses.warehouse\_id = warehouses.warehouse\_id;')

**return** c.fetchall()

**def** get\_delivered\_parcels(self):

c = self.conn.cursor()

c.execute('''

SELECT couriers.name AS courier\_name, COUNT(parcels.parcel\_id) AS delivered\_count

FROM couriers

LEFT JOIN parcels ON couriers.courier\_id = parcels.courier\_id

WHERE parcels.status = 'Delivered'

GROUP BY couriers.courier\_id, couriers.name;

''')

**return** c.fetchall()

**def** get\_parcels\_in\_warehouses(self):

c = self.conn.cursor()

c.execute('''

SELECT warehouses.address AS warehouse\_address, COUNT(parcels.parcel\_id) AS parcel\_count

FROM parcels

JOIN warehouses ON parcels.warehouse\_id = warehouses.warehouse\_id

GROUP BY warehouses.warehouse\_id, warehouses.address;

''')

**return** c.fetchall()

**def** update\_recipient(self, name, address, id):

c = self.conn.cursor()

c.execute('UPDATE recipients SET name=%s, address=%s WHERE recipient\_id=%s', (name, address, id))

self.conn.commit()

**def** update\_sender(self, name, address, id):

c = self.conn.cursor()

c.execute('UPDATE senders SET name=%s, address=%s WHERE sender\_id=%s', (name, address, id))

self.conn.commit()

**def** update\_parcel(self, status, creation\_date, parcel\_cost, courier\_id, recipient\_id, warehouse\_id, id):

c = self.conn.cursor()

c.execute('UPDATE parcels SET status=%s, creation\_date=%s, parcel\_cost=%s, courier\_id=%s, recipient\_id=%s, warehouse\_id=%s WHERE parcel\_id=%s', (status, creation\_date, parcel\_cost, courier\_id, recipient\_id, warehouse\_id, id))

self.conn.commit()

**def** update\_courier(self, name, phone, id):

c = self.conn.cursor()

c.execute('UPDATE couriers SET name=%s, phone=%s WHERE courier\_id=%s', (name, phone, id))

self.conn.commit()

**def** update\_warehouse(self, address, phone, id):

c = self.conn.cursor()

c.execute('UPDATE warehouses SET address=%s, phone=%s WHERE warehouse\_id=%s', (address, phone, id))

self.conn.commit()

**def** update\_courier\_warehouse(self, courier\_id, warehouse\_id, id):

c = self.conn.cursor()

c.execute('UPDATE warehouses SET courier\_id=%s, warehouse\_id=%s WHERE courier\_warehouse\_id=%s', (courier\_id, warehouse\_id, id))

self.conn.commit()

**def** delete\_recipient(self, recipient\_id):

c = self.conn.cursor()

c.execute('DELETE FROM recipients WHERE recipient\_id=%s', (recipient\_id,))

self.conn.commit()

**def** delete\_sender(self, sender\_id):

c = self.conn.cursor()

c.execute('DELETE FROM senders WHERE sender\_id=%s', (sender\_id,))

self.conn.commit()

**def** delete\_parcel(self, parcel\_id):

c = self.conn.cursor()

c.execute('DELETE FROM parcels WHERE parcel\_id=%s', (parcel\_id,))

self.conn.commit()

**def** delete\_courier(self, courier\_id):

c = self.conn.cursor()

c.execute('DELETE FROM couriers WHERE courier\_id=%s', (courier\_id,))

self.conn.commit()

**def** delete\_warehouse(self, warehouse\_id):

c = self.conn.cursor()

c.execute('DELETE FROM warehouses WHERE warehouse\_id=%s', (warehouse\_id,))

self.conn.commit()

**def** delete\_courier\_warehouse(self, courier\_warehouse\_id):

c = self.conn.cursor()

c.execute('DELETE FROM couriers\_warehouses WHERE courier\_warehouse\_id=%s', (courier\_warehouse\_id,))

self.conn.commit()

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

c = self.conn.cursor()

c.execute('''

INSERT INTO warehouses (address, phone)

SELECT chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int), '+380' || lpad(trunc(random() \* 10000000)::text, 7, '0')

FROM generate\_series(1, %s);

''',(number,))

self.conn.commit()

**controller.py**

**import** time

**from** model **import** Model

**from** view **import** View

**class** Controller:

**def** \_\_init\_\_(self):

self.model = Model()

self.view = View()

**def** run(self):

**while** True:

choice = self.view.show\_menu()

**if** choice == '7':

**break**

**if** choice == '6':

self.process\_search\_option()

**elif** choice **in** ['1', '2', '3', '4', '5']:

self.process\_menu\_choice(choice)

**else**:

self.view.show\_message("Wrong choice. Try again.")

**def** process\_menu\_choice(self, choice):

**while** True:

table = self.view.show\_tables()

**if** table == '6':

**break**

**if** choice == '1':

self.process\_add\_option(table)

**elif** choice == '2':

self.process\_add\_random\_option(table)

**elif** choice == '3':

self.process\_view\_option(table)

**elif** choice == '4':

self.process\_update\_option(table)

**elif** choice == '5':

self.process\_delete\_option(table)

**def** process\_add\_option(self, table):

**if** table == '1':

self.view.show\_message("\nAdding recipient:")

self.add\_recipient()

**elif** table == '2':

self.view.show\_message("\nAdding sender:")

self.add\_sender()

**elif** table == '3':

self.view.show\_message("\nAdding parcel:")

self.add\_parcel()

**elif** table == '4':

self.view.show\_message("\nAdding courier:")

self.add\_courier()

**elif** table == '5':

self.view.show\_message("\nAdding warehouse:")

self.add\_warehouse()

**elif** table == '6':

self.view.show\_message("\nAdding courier-warehouse:")

self.add\_courier\_warehouse()

**else**:

self.view.show\_message("Wrong choice. Try again.")

**def** process\_add\_random\_option(self, table):

**if** table == '5':

self.view.show\_message("\nAdding random warehouses:")

self.add\_random\_fields()

**else**:

self.view.show\_message("Wrong choice. Try again.")

**def** process\_view\_option(self, table):

**if** table == '1':

self.view\_recipients()

**elif** table == '2':

self.view\_senders()

**elif** table == '3':

self.view\_parcels()

**elif** table == '4':

self.view\_couriers()

**elif** table == '5':

self.view\_warehouses()

**elif** table == '6':

self.view\_couriers\_warehouses()

**elif** table == '7':

self.view.show\_menu()

**else**:

self.view.show\_message("Wrong choice. Try again.")

**def** process\_update\_option(self, table):

**if** table == '1':

self.view.show\_message("\nUpdating recipient:")

self.update\_recipient()

**elif** table == '2':

self.view.show\_message("\nUpdating sender:")

self.update\_sender()

**elif** table == '3':

self.view.show\_message("\nUpdating parcel:")

self.update\_parcel()

**elif** table == '4':

self.view.show\_message("\nUpdating courier:")

self.update\_courier()

**elif** table == '5':

self.view.show\_message("\nUpdating warehouse:")

self.update\_warehouse()

**elif** table == '6':

self.view.show\_message("\nUpdating courier-warehouse:")

self.update\_courier\_warehouse()

**else**:

self.view.show\_message("Wrong choice. Try again.")

**def** process\_delete\_option(self, table):

**if** table == '1':

self.view.show\_message("\nDeleting recipient:")

self.delete\_recipient()

**elif** table == '2':

self.view.show\_message("\nDeleting sender:")

self.delete\_sender()

**elif** table == '3':

self.view.show\_message("\nDeleting parcel:")

self.delete\_parcel()

**elif** table == '4':

self.view.show\_message("\nDeleting courier:")

self.delete\_courier()

**elif** table == '5':

self.view.show\_message("\nDeleting warehouse:")

self.delete\_warehouse()

**elif** table == '6':

self.view.show\_message("\nDeleting courier-warehouse:")

self.delete\_courier\_warehouse()

**else**:

self.view.show\_message("Wrong choice. Try again.")

**def** process\_search\_option(self):

option = self.view.show\_search()

**if** option == '1':

start\_time = time.time()

self.show\_couriers\_with\_warehouses()

end\_time = time.time()

elapsed\_time = (end\_time - start\_time) \* 1000

**print**(f"Час виконання: {elapsed\_time:.2f} мс")

**elif** option == '2':

start\_time = time.time()

self.show\_delivered\_parcels()

end\_time = time.time()

elapsed\_time = (end\_time - start\_time) \* 1000

**print**(f"Час виконання: {elapsed\_time:.2f} мс")

**elif** option == '3':

start\_time = time.time()

self.show\_parcels\_in\_warehouses()

end\_time = time.time()

elapsed\_time = (end\_time - start\_time) \* 1000

**print**(f"Час виконання: {elapsed\_time:.2f} мс")

**else**:

self.view.show\_menu()

**def** add\_recipient(self):

**try**:

name, address = self.view.get\_recipient\_input()

self.model.add\_recipient(name, address)

self.view.show\_message("Recipient added successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** add\_sender(self):

**try**:

name, address = self.view.get\_sender\_input()

self.model.add\_sender(name, address)

self.view.show\_message("Sender added successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** add\_parcel(self):

**try**:

status, creation\_date, parcel\_cost, courier\_id, recipient\_id, warehouse\_id = self.view.get\_parcel\_input()

self.model.add\_parcel(status, creation\_date, parcel\_cost, courier\_id, recipient\_id, warehouse\_id)

self.view.show\_message("Parcel added successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** add\_courier(self):

**try**:

name, phone = self.view.get\_courier\_input()

self.model.add\_courier(name, phone)

self.view.show\_message("Courier added successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** add\_warehouse(self):

**try**:

address, phone = self.view.get\_warehouse\_input()

self.model.add\_warehouse(address, phone)

self.view.show\_message("Warehouse added successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** add\_courier\_warehouse(self):

**try**:

courier\_id, warehouse\_id = self.view.get\_courier\_warehouse\_input()

self.model.add\_courier\_warehouse(courier\_id, warehouse\_id)

self.view.show\_message("Courier Warehouse added successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** view\_recipients(self):

**try**:

recipients = self.model.get\_all\_recipients()

self.view.show\_recipients(recipients)

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** view\_senders(self):

**try**:

senders = self.model.get\_all\_senders()

self.view.show\_senders(senders)

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** view\_parcels(self):

**try**:

parcels = self.model.get\_all\_parcels()

self.view.show\_parcels(parcels)

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** view\_couriers(self):

**try**:

couriers = self.model.get\_all\_couriers()

self.view.show\_couriers(couriers)

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** view\_warehouses(self):

**try**:

warehouses = self.model.get\_all\_warehouses()

self.view.show\_warehouses(warehouses)

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** view\_couriers\_warehouses(self):

**try**:

couriers\_warehouses = self.model.get\_all\_couriers\_warehouses()

self.view.show\_couriers\_warehouses(couriers\_warehouses)

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** show\_couriers\_with\_warehouses(self):

**try**:

rows = self.model.get\_all\_couriers\_with\_warehouses()

self.view.show\_couriers\_with\_warehouses(rows)

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** show\_delivered\_parcels(self):

**try**:

rows = self.model.get\_delivered\_parcels()

self.view.show\_delivered\_parcels(rows)

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** show\_parcels\_in\_warehouses(self):

**try**:

rows = self.model.get\_parcels\_in\_warehouses()

self.view.show\_parcels\_in\_warehouses(rows)

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** update\_recipient(self):

**try**:

recipient\_id = self.view.get\_id()

name, address = self.view.get\_recipient\_input()

self.model.update\_recipient(name, address, recipient\_id)

self.view.show\_message("Recipient updated successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** update\_sender(self):

**try**:

sender\_id = self.view.get\_id()

name, address = self.view.get\_sender\_input()

self.model.update\_sender(name, address, sender\_id)

self.view.show\_message("Sender updated successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** update\_parcel(self):

**try**:

parcel\_id = self.view.get\_id()

status, creation\_date, parcel\_cost, courier\_id, recipient\_id, warehouse\_id = self.view.get\_parcel\_input()

self.model.update\_parcel(status, creation\_date, parcel\_cost, courier\_id, recipient\_id, warehouse\_id, parcel\_id)

self.view.show\_message("Parcel updated successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** update\_courier(self):

**try**:

courier\_id = self.view.get\_id()

name, phone = self.view.get\_courier\_input()

self.model.update\_courier(name, phone, courier\_id)

self.view.show\_message("Courier updated successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** update\_warehouse(self):

**try**:

warehouse\_id = self.view.get\_id()

address, phone = self.view.get\_warehouse\_input()

self.model.update\_warehouse(address, phone, warehouse\_id)

self.view.show\_message("Warehouse updated successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** update\_courier\_warehouse(self):

**try**:

courier\_warehouse\_id = self.view.get\_id()

courier\_id, warehouse\_id = self.view.get\_courier\_warehouse\_input()

self.model.update\_courier\_warehouse(courier\_id, warehouse\_id, courier\_warehouse\_id)

self.view.show\_message("Courier Warehouse updated successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** delete\_recipient(self):

**try**:

recipient\_id = self.view.get\_id()

self.model.delete\_recipient(recipient\_id)

self.view.show\_message("Recipient deleted successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** delete\_sender(self):

**try**:

sender\_id = self.view.get\_id()

self.model.delete\_sender(sender\_id)

self.view.show\_message("Sender deleted successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** delete\_parcel(self):

**try**:

parcel\_id = self.view.get\_id()

self.model.delete\_parcel(parcel\_id)

self.view.show\_message("Parcel deleted successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** delete\_courier(self):

**try**:

courier\_id = self.view.get\_id()

self.model.delete\_courier(courier\_id)

self.view.show\_message("Courier deleted successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** delete\_warehouse(self):

**try**:

warehouse\_id = self.view.get\_id()

self.model.delete\_warehouse(warehouse\_id)

self.view.show\_message("Warehouse deleted successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** delete\_courier\_warehouse(self):

**try**:

courier\_warehouse\_id = self.view.get\_id()

self.model.delete\_courier\_warehouse(courier\_warehouse\_id)

self.view.show\_message("Courier Warehouse deleted successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** add\_random\_fields(self):

**try**:

number = self.view.get\_number()

self.model.add\_random\_fields(number)

self.view.show\_message("Random fields added successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**view.py**

**from** datetime **import** datetime

**class** View:

**def** show\_menu(self):

self.show\_message("\nМеню:")

self.show\_message("1. �"одати рядок"***)***

self.show\_message('2. �"енерування «рандомізованих» даних (тільки для таблиці "Warehouses")'***)***

self.show\_message("3. Показати таблицю"***)***

self.show\_message("4. Редагувати рядок"***)***

self.show\_message("5. �'идалити рядок"***)***

self.show\_message("6. Пошук"***)***

self.show\_message("7. �'ихід"***)***

choice = input("�'иберіть пункт: "***)***

return choic***e***

def show\_tables(self)***:***

self.show\_message("\nТаблиці:"***)***

self.show\_message("1. Recipients (отримувачі)"***)***

self.show\_message("2. Senders (відправники)"***)***

self.show\_message("3. Parcels (посилки)"***)***

self.show\_message("4. Couriers (кур'�"ри)"***)***

self.show\_message("5. Warehouses (склади)"***)***

self.show\_message("6. Couriers Warehouses (кур'�"ри-склади)")

self.show\_message("7. Повернутися до меню")

table = input("Оберіть потрібну таблицю: ")

**return** table

**def** show\_search(self):

self.show\_message("\nПошук:")

self.show\_message("1. Якому кур'�"ру відповіда�" який склад.")

self.show\_message("2. Кількість посилок, які вже доставив кур'�"р."***)***

self.show\_message("3. Кільксть посилок, які знаходяться в кожному складі."***)***

self.show\_message("4. Повернутися до меню"***)***

choice = input("Обреріть щось: "***)***

return choic***e***

def show\_recipients(self, recipients)***:***

print("\nRecipients:"***)***

for recipient in recipients***:***

print(f"ID: {recipient[0]}, Name: {recipient[1]}, Address: {recipient[2]}"***)***

def show\_senders(self, senders)***:***

print("\nSenders:"***)***

for sender in senders***:***

print(f"ID: {sender[0]}, Name: {sender[1]}, Address: {sender[2]}"***)***

def show\_parcels(self, parcels)***:***

print("\nParcels:"***)***

for parcel in parcels***:***

print(f"ID: {parcel[0]}, Status: {parcel[1]}, Creation Date: {parcel[2]}, Price: {parcel[3]}, Courier ID: {parcel[4]}, Recipient ID: {parcel[5]}, Warehouse ID: {parcel[6]}"***)***

def show\_couriers(self, couriers)***:***

print("\nCouriers:"***)***

for courier in couriers***:***

print(f"ID: {courier[0]}, Name: {courier[1]}, Phone: {courier[2]}"***)***

def show\_warehouses(self, warehouses)***:***

print("\nWarehouses:"***)***

for warehouse in warehouses***:***

print(f"ID: {warehouse[0]}, Address: {warehouse[1]}, Phone: {warehouse[2]}"***)***

def show\_couriers\_warehouses(self, couriers\_warehouses)***:***

print("\nCouriers Warehouses:"***)***

for courier\_warehouse in couriers\_warehouses***:***

print(f"ID: {courier\_warehouse[0]}, Courier ID: {courier\_warehouse[1]}, Warehouse ID: {courier\_warehouse[2]}"***)***

def show\_couriers\_with\_warehouses(self, rows)***:***

print("\nЯкому кур'�"ру відповіда�" який склад:"***)***

for row in rows***:***

print(f"Courier name: {row[0]}, Warehouse address: {row[1]}"***)***

def show\_delivered\_parcels(self, rows)***:***

print("\nКількість посилок, які вже доставив кур'�"р:")

**for** row **in** rows:

**print**(f"Courier name: {row[0]}, Delivered parcels count: {row[1]}")

**def** show\_parcels\_in\_warehouses(self, rows):

**print**("\nКільксть посилок, які знаходяться в кожному складі:")

**for** row **in** rows:

**print**(f"Warehouse address: {row[0]}, Parcels count: {row[1]}")

**def** get\_recipient\_input(self):

**while** True:

**try**:

name = input("Enter recipient name: ")

**if** name.strip():

**break**

**else**:

**print**("Name cannot be empty.")

**except** ValueError:

**print**("It must be a string.")

**while** True:

**try**:

address = input("Enter recipient address: ")

**if** address.strip():

**break**

**else**:

**print**("Address cannot be empty.")

**except** ValueError:

**print**("It must be a string.")

**return** name, address

**def** get\_sender\_input(self):

**while** True:

**try**:

name = input("Enter sender name: ")

**if** name.strip():

**break**

**else**:

**print**("Name cannot be empty.")

**except** ValueError:

**print**("It must be a string.")

**while** True:

**try**:

address = input("Enter sender address: ")

**if** address.strip():

**break**

**else**:

**print**("Address cannot be empty.")

**except** ValueError:

**print**("It must be a string.")

**return** name, address

**def** get\_parcel\_input(self):

**while** True:

**try**:

status = input("Enter parcel status: ")

**if** status.strip():

**break**

**else**:

**print**("Status cannot be empty.")

**except** ValueError:

**print**("It must be a string.")

**while** True:

**try**:

date = input("Enter creation date (YYYY-MM-DD): ")

creation\_date = datetime.strptime(date, "%Y-%m-%d").date()

**break**

**except** ValueError:

**print**("Invalid date format. Please enter the date in YYYY-MM-DD format.")

**while** True:

**try**:

parcel\_cost\_input = input("Enter parcel price: ")

**if** parcel\_cost\_input.strip():

parcel\_cost = float(parcel\_cost\_input)

**break**

**else**:

**print**("Price cannot be empty.")

**except** ValueError:

**print**("Price must be a valid number.")

**while** True:

**try**:

courier\_id = int(input("Enter parcel courier ID: "))

**break**

**except** ValueError:

**print**("Courier ID must be a number.")

**while** True:

**try**:

recipient\_id = int(input("Enter parcel recipient ID: "))

**break**

**except** ValueError:

**print**("Recipient ID must be a number.")

**while** True:

**try**:

warehouse\_id = int(input("Enter parcel warehouse ID: "))

**break**

**except** ValueError:

**print**("Warehouse ID must be a number.")

**return** status, creation\_date, parcel\_cost, courier\_id, recipient\_id, warehouse\_id

**def** get\_courier\_input(self):

**while** True:

**try**:

name = input("Enter courier name: ")

**if** name.strip():

**break**

**else**:

**print**("Name cannot be empty.")

**except** ValueError:

**print**("It must be a string.")

**while** True:

**try**:

phone = input("Enter courier phone: ")

**if** phone.strip():

**break**

**else**:

**print**("Phone cannot be empty.")

**except** ValueError:

**print**("It must be a string.")

**return** name, phone

**def** get\_warehouse\_input(self):

**while** True:

**try**:

address = input("Enter warehouse address: ")

**if** address.strip():

**break**

**else**:

**print**("Address cannot be empty.")

**except** ValueError:

**print**("It must be a string.")

**while** True:

**try**:

phone = input("Enter warehouse phone: ")

**if** phone.strip():

**break**

**else**:

**print**("Phone cannot be empty.")

**except** ValueError:

**print**("It must be a string.")

**return** address, phone

**def** get\_courier\_warehouse\_input(self):

**while** True:

**try**:

courier\_id = int(input("Enter courier ID: "))

**break**

**except** ValueError:

**print**("Courier ID must be a number.")

**while** True:

**try**:

warehouse\_id = int(input("Enter warehouse ID: "))

**break**

**except** ValueError:

**print**("Warehouse ID must be a number.")

**return** courier\_id, warehouse\_id

**def** get\_id(self):

**while** True:

**try**:

id = int(input("Enter ID: "))

**break**

**except** ValueError:

**print**("It must be a number.")

**return** id

**def** show\_message(self, message):

**print**(message)

**def** get\_number(self):

**while** True:

**try**:

number = int(input("Enter the number: "))

**break**

**except** ValueError:

**print**("It must be a number.")

**return** number