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»

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

Виконав: студент III курсу групи КВ-33

ПІБ: Калінін М. І.

Перевірив: Павловський В. І.

Київ – 2025

Мета: здобуття вмінь проектування бази даних та практичних навичок створення реляційних баз даних за допомогою PostgreSQL.

Завдання:

1. Розробити модель «сутність-зв’язок» предметної галузі, обраної студентом самостійно, відповідно до пункту «Вимоги до ER-моделі».
2. Перетворити розроблену модель у схему бази даних (таблиці) PostgreSQL.
3. Виконати нормалізацію схеми бази даних до третьої нормальної форми (3НФ).
4. Ознайомитись із інструментарієм PostgreSQL та pgAdmin 4 та внести декілька рядків даних у кожну з таблиць засобами pgAdmin 4.

Опис предметної області

Дана предметна область – система обліку наявності медичних препаратів у аптеках. Вона охоплює процеси збереження та оновлення інформації про аптеки, лікарські препарати та їх кількість у наявності. Система дозволяє ефективно відслідковувати, які препарати доступні в конкретних аптеках, контролювати їх залишки та дату оновлення даних. Це сприяє покращенню обслуговування клієнтів і забезпечує актуальність інформації.

Аптека (Pharmacies)

Атрибути: ідентифікатор аптеки, назва, адреса.

Телефон (Phones)

Атрибути: ідентифікатор телефону, номер телефону, ідентифікатор аптеки, ідентифікатор типу телефону.

Тип телефону (PhoneTypes)

Атрибути: ідентифікатор типу, назва типу.

Виробник (Manufacturers)

Атрибути: ідентифікатор виробника, назва виробника.

Препарат (Drugs)

Атрибути: ідентифікатор препарату, назва, ідентифікатор виробника, ціна.

Наявність (Availabilities)

Атрибути: ідентифікатор аптеки, ідентифікатор препарату, кількість, дата оновлення.

Опис зв’язків між сутностями

Зв’язок «Аптека» – «Наявність» є зв’язком 1:N.

Одна аптека може мати багато записів щодо наявності різних препаратів, але один запис належить лише одній аптеці.

Зв’язок «Препарат» – «Наявність» є зв’язком 1:N.

Один препарат може бути доступний у багатьох аптеках, але один запис про наявність відповідає лише одному препарату.

Зв’язок «Аптека» – «Препарат» через «Наявність» є зв’язком M:N.

Багато аптек можуть мати в наявності багато різних препаратів, і навпаки один препарат може продаватися у багатьох аптеках.

Зв’язок «Аптека» – «Телефон» є зв’язком 1:N.

Одна аптека може мати декілька телефонів для зв’язку, але кожен телефон належить лише одній аптеці.

Зв’язок «Телефон» – «Тип телефону» є зв’язком N:1.

Кожен телефон має лише один тип (наприклад, мобільний чи стаціонарний), але один тип може використовуватись багатьма телефонами.

Зв’язок «Препарат» – «Виробник» є зв’язком N:1.

Кожен препарат має лише одного виробника, але один виробник може випускати багато різних препаратів.

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

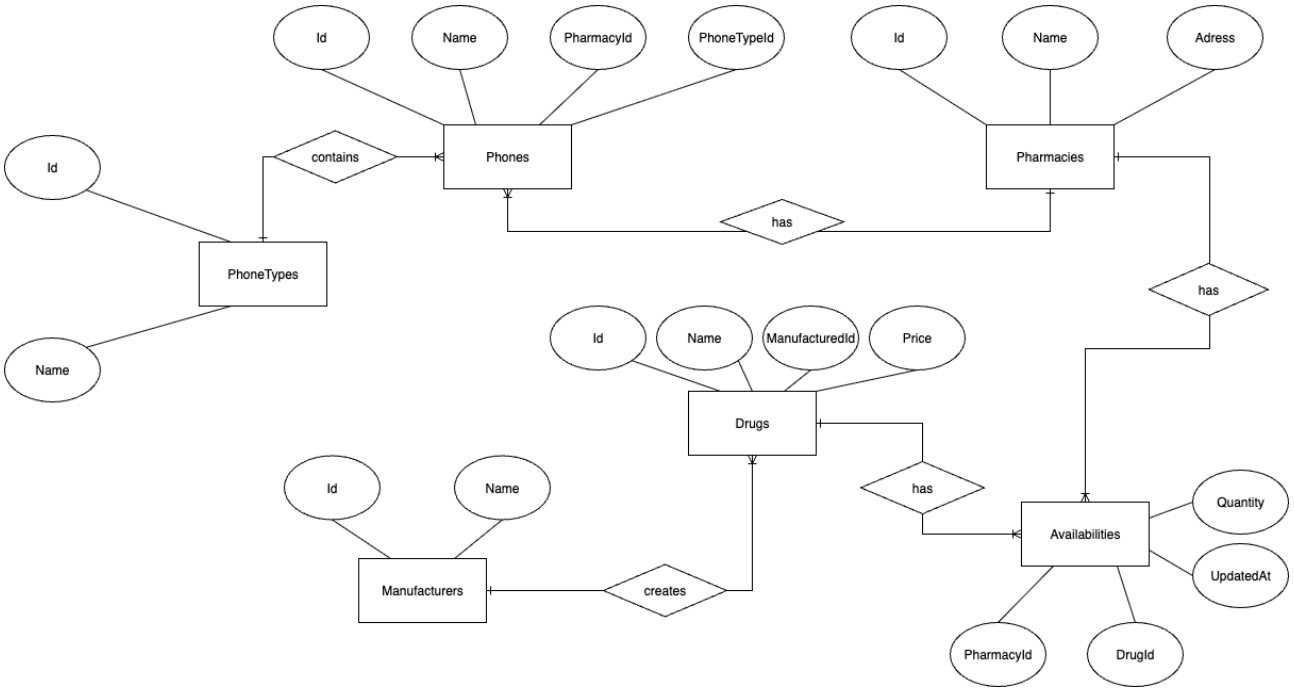

Рисунок 1 – ER-діаграма, побудована за нотацією “Пташиної лапки (Crow’s foot)”

Перетворення концептуальної моделі у логічну модель та схему бази даних

Сутність Pharmacies перетворено в таблицю Pharmacies з первинним ключем Id та атрибутами Name, Adress. Призначення таблиці – зберігання основних даних про аптеки.

Сутність Phones перетворено в таблицю Phones з первинним ключем Id та атрибутами Name (номер телефону), PharmacyId, PhoneTypeId. Атрибут PharmacyId є зовнішнім ключем, що посилається на таблицю Pharmacies, а PhoneTypeId – на таблицю PhoneTypes.

Сутність PhoneTypes перетворено в таблицю PhoneTypes з первинним ключем Id та атрибутом Name. Таблиця використовується для класифікації телефонних номерів (наприклад, мобільний чи стаціонарний).

Сутність Manufacturers перетворено в таблицю Manufacturers з первинним ключем Id та атрибутом Name. Таблиця зберігає дані про виробників лікарських препаратів.

Сутність Drugs перетворено в таблицю Drugs з первинним ключем Id та атрибутами Name, ManufacturerId, Price. Атрибут ManufacturerId є зовнішнім ключем, що посилається на таблицю Manufacturers.

Сутність Availabilities перетворено в таблицю Availabilities з первинним ключем, що складається з PharmacyId та DrugId, та атрибутами Quantity, UpdatedAt. Атрибути PharmacyId і DrugId одночасно є зовнішніми ключами, які посилаються на таблиці Pharmacies та Drugs відповідно. Оскільки в логічній моделі безпосередній зв’язок M:N є неможливим, а в концептуальній моделі він існує між сутностями Pharmacies і Drugs, то для його реалізації було створено додаткову таблицю Availabilities, що розриває цей зв’язок на два зв’язки типу 1:N.

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

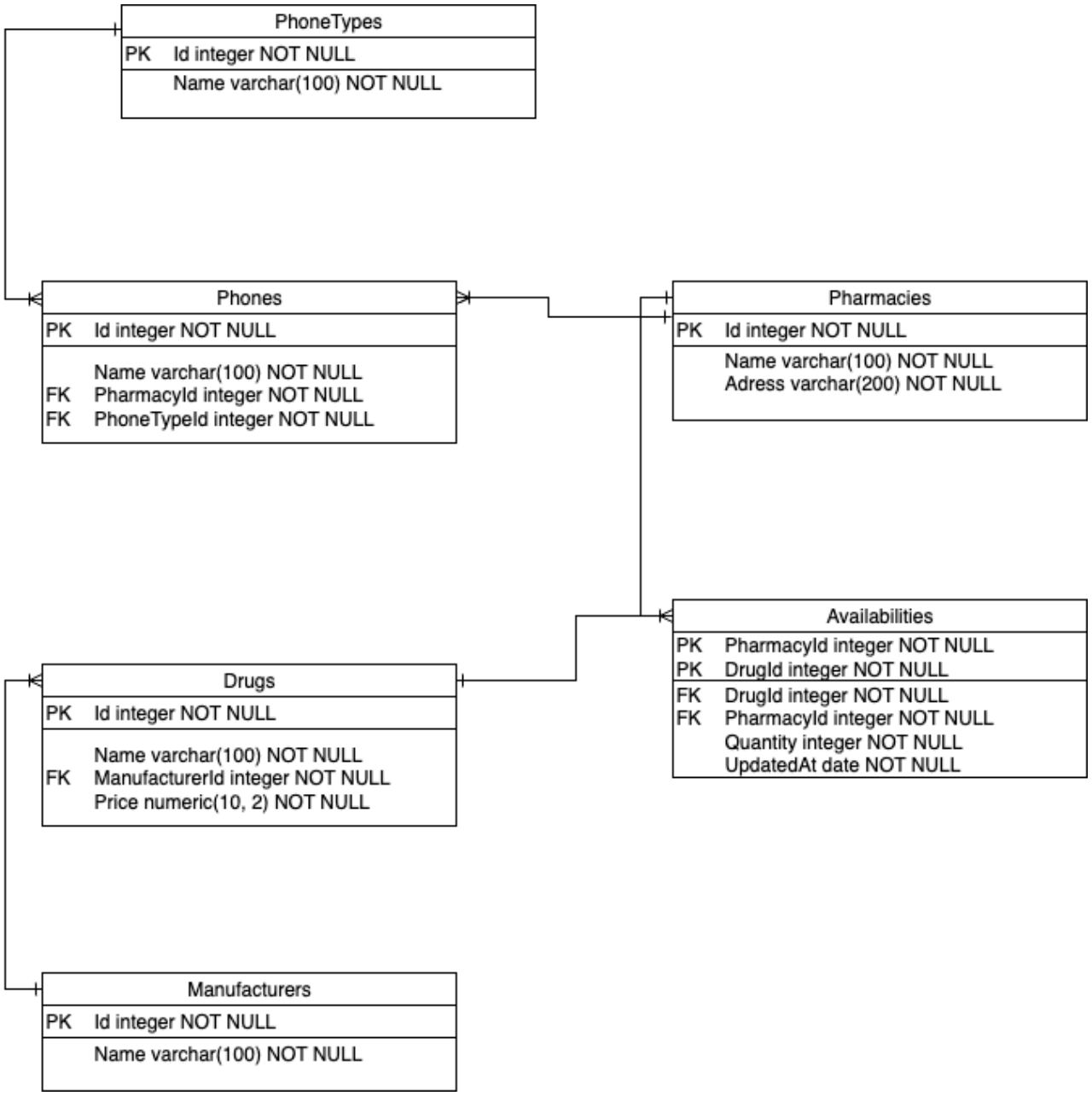

Рисунок 2 – Схема бази даних

Середовище та компоненти розробки

Для розробки використовувалась мова програмування Python, середовище розробки Visual Studio Code, стороння бібліотека для доступу до PostgreSQL – psycopg2, вбудована бібліотека, задля можливості використання таймера спрацювання функцій – time, а також стороння бібліотека, що надає можливість коректного виводу інформації таблиць – tabulate. Встановити бібліотеку tabulate можна шляхом написання наступної команди до консолі користувача: `pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org tabulate`

Шаблон проектування

MVC – Шаблон проектування, який був використаний при проектуванні застосунку.

Model – Представляє собою клас, що описує логіку використання даних, отриманих від користувача, а також тих даних, що початково закладені у саму програму.

View – Представляє собою клас комунікації із користувачем, а також виводу необхідної інформації.

Controller – Представляє собою клас, що описує головну логіку взаємодії усіх інших класів та модулів.

Структура програми та її опис

Подання структури програми зображено на рисунках нижче:

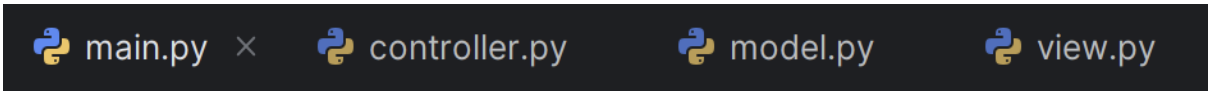

Рисунок 3 – Структура програми

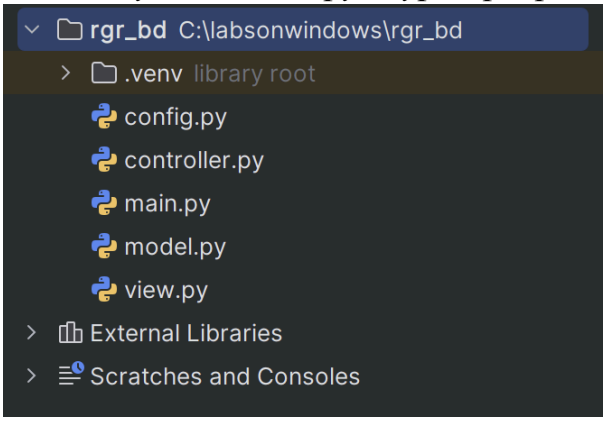

Рисунок 4 – Структура програми

Програма умовно поділена на 4 модулі:

1. Файл main.py;
2. Файл Model.py;
3. Файл View.py;
4. Файл Controller.py;

Файл main.py являє собою модуль початкової точки запуску усієї програми. Саме у цьому файлі відбувається підключення головного командного модуля MVC шаблону – Controller.py

Файл Controller.py являє собою модуль керування, що визначає порядок дій, а також порядок викликів необхідних функцій та методів інших модулів.

Файл View.py являє собою модуль комунікації із користувачем. Даний модуль виводить необхідну інформацію до користувача, отримує від

користувача дані, після чого готує отримані дані до відправки в модуль Model.py.

Файл Model.py являє собою модуль взаємодії із базою даних. Даний модуль отримує дані від модуля View.py, після чого починає спілкування із базою даних, аналізує отримані дані, та надсилає їх до назад до модуля View.py, аби вивести результати до користувача.

Структура меню програми

Меню користувача надає доступ до управління даними аптечної мережі та складається з головного циклу обробки команд, який включає 7 основних пунктів: перегляд, додавання, оновлення, видалення, генерація даних, пошук та аналітика, а також вихід з програми.

Консольне подання структури користувацького меню програми реалізовано через ієрархію вкладених меню для кожної сутності бази даних.

```
>>>
=== MAIN MENU ===

>>> 1. Show data

>>> 2. Add data

>>> 3. Update data

>>> 4. Delete data

>>> 5. Generate data

>>> 6. SEARCH & ANALYTICS (Lab Task)

>>> 7. Quit
Enter choice:
```

Рисунок 5 – Структура меню програми

Нижче розглянемо код кожного модуля, проаналізуємо їх поведінку, функціональне призначення та взаємодію.

Модуль main.py

Даний модуль визначає початкову точку запуску застосунку. Він виконує імпорт класу контролера, створює його екземпляр та ініціює виконання основного методу main_run().

```
from controller import Controller

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

Модуль controller.py

Даний модуль реалізує логіку керування потоком виконання програми. Він виступає посередником між інтерфейсом (View) та даними (Model), визначає реакцію на дії користувача та викликає відповідні методи обробки даних.

- **Ініціалізація роботи**: Створення екземплярів Model та View.

```
def __init__(self):
    self.model = Model()
    self.view = View()
```

- **Головний цикл**: Відображення основного меню та маршрутизація вибору користувача.

```
def main_run(self):
    while True:
        self.view.show_message("\n=== MAIN MENU ===")
        self.view.show_message("1. Show data")
        self.view.show_message("2. Add data")
        self.view.show_message("3. Update data")
        self.view.show_message("4. Delete data")
        self.view.show_message("5. Generate data")
        self.view.show_message("6. SEARCH & ANALYTICS (Lab Task)")  #
Добавлено
        self.view.show_message("7. Quit")

        choice = self.view.get_int("Enter choice: ", min_val=1, max_val=7)
        if choice == 1:
            self.show_menu()
```

```
        elif choice == 2:
            self.add_menu()
        elif choice == 3:
            self.update_menu()
        elif choice == 4:
            self.delete_menu()
        elif choice == 5:
            self.generate_menu()
        elif choice == 6:
            self.search_menu()
        elif choice == 7:
            self.view.show_message("Goodbye!")
            self.model.close()
            break
```

- **Меню перегляду (**show_menu**)**: Організація виводу даних з таблиць з можливістю вибору режиму (всі записи, лімітована вибірка або пошук за ID).

```
def show_menu(self):
    while True:
        self.view.show_message("\n-- SHOW MENU --")
        self.view.show_message("1. Pharmacies")
        self.view.show_message("2. PhoneTypes")
        self.view.show_message("3. Phones")
        self.view.show_message("4. Manufacturers")
        self.view.show_message("5. Drugs")
        self.view.show_message("6. Availabilities")
        self.view.show_message("7. Back")
        choice = self.view.get_int("Enter choice: ", 1, 7)
        if choice == 1:
            print("\n--- VIEW MODES ---")
            print("1. Show limit 50 (default)")
            print("2. Show ALL")
            print("3. Show ONE by ID")
            mode = self.view.get_int("Select mode: ", 1, 3)

            if mode == 1:
```

```
            rows = self.model.get_pharmacies_limited()
            self.view.show_pharmacies(rows)
        elif mode == 2:
            rows = self.model.get_pharmacies()
            self.view.show_pharmacies(rows)
        elif mode == 3:
            id_val = self.view.get_int("Enter Pharmacy ID: ")
            rows = self.model.get_pharmacy_by_id(id_val)
            if rows:
                self.view.show_pharmacies(rows)
            else:
                self.view.show_message("Pharmacy with this ID not found.")
    elif choice == 2:
        rows = self.model.get_phonetypes()
        self.view.show_phonetypes(rows)
    elif choice == 3:
        rows = self.model.get_phones()
        self.view.show_phones(rows)
    elif choice == 4:
        rows = self.model.get_manufacturers()
        self.view.show_manufacturers(rows)
    elif choice == 5:
        rows = self.model.get_drugs()
        self.view.show_drugs(rows)
    elif choice == 6:
        rows = self.model.get_availabilities()
        self.view.show_availabilities(rows)
    elif choice == 7:
        break
```

- **Меню модифікації** (add_menu, update_menu): Збір даних від користувача для створення або оновлення записів.

```
def add_menu(self):
  while True:
    self.view.show_message("\n-- ADD MENU --")
    self.view.show_message("1. Add Pharmacy")
    self.view.show_message("2. Add PhoneType")
```

```
        self.view.show_message("3. Add Phone")
        self.view.show_message("4. Add Manufacturer")
        self.view.show_message("5. Add Drug")
        self.view.show_message("6. Add Availability")
        self.view.show_message("7. Back")
        choice = self.view.get_int("Enter choice: ", 1, 7)

        status = None

        if choice == 1:
            name = self.view.get_input("Pharmacy name: ")
            addr = self.view.get_input("Address: ")
            status = self.model.add_pharmacy(name, addr)
        elif choice == 2:
            name = self.view.get_input("PhoneType name: ")
            status = self.model.add_phonetype(name)
        elif choice == 3:
            name = self.view.get_input("Phone name/label: ")
            pharmacy_id = self.view.get_int("PharmacyId: ")
            phonetype_id = self.view.get_int("PhoneTypeId: ")
            status = self.model.add_phone(name, pharmacy_id, phonetype_id)
        elif choice == 4:
            name = self.view.get_input("Manufacturer name: ")
            status = self.model.add_manufacturer(name)
        elif choice == 5:
            name = self.view.get_input("Drug name: ")
            manuf_id = self.view.get_int("ManufacturerId: ")
            price = self.view.get_float("Price: ")
            status = self.model.add_drug(name, manuf_id, price)
        elif choice == 6:
            pharmacy_id = self.view.get_int("PharmacyId: ")
            drug_id = self.view.get_int("DrugId: ")
            qty = self.view.get_int("Quantity: ")
            status = self.model.add_availability(pharmacy_id, drug_id, qty)
        elif choice == 7:
            break
```

```
        if status:
            if status == "OK":
                self.view.show_message("Added successfully!")
            else:
                self.view.show_error(status)

# ---------------- update ----------------
def update_menu(self):
    while True:
        self.view.show_message("\n-- UPDATE MENU --")
        self.view.show_message("1. Update Pharmacy")
        self.view.show_message("2. Update PhoneType")
        self.view.show_message("3. Update Phone")
        self.view.show_message("4. Update Manufacturer")
        self.view.show_message("5. Update Drug")
        self.view.show_message("6. Update Availability")
        self.view.show_message("7. Back")
        choice = self.view.get_int("Enter choice: ", 1, 7)
        if choice == 1:
            id_ = self.view.get_int("Pharmacy Id to update: ")
            name = self.view.get_input("New name: ")
            addr = self.view.get_input("New address: ")
            self.model.update_pharmacy(id_, name, addr)
        elif choice == 2:
            id_ = self.view.get_int("PhoneType Id to update: ")
            name = self.view.get_input("New name: ")
            self.model.update_phonetype(id_, name)
        elif choice == 3:
            id_ = self.view.get_int("Phone Id to update: ")
            name = self.view.get_input("New name: ")
            pharmacy_id = self.view.get_int("New PharmacyId: ")
            phonetype_id = self.view.get_int("New PhoneTypeId: ")
            self.model.update_phone(id_, name, pharmacy_id, phonetype_id)
        elif choice == 4:
            id_ = self.view.get_int("Manufacturer Id to update: ")
            name = self.view.get_input("New name: ")
            self.model.update_manufacturer(id_, name)
```

```
        elif choice == 5:
            id_ = self.view.get_int("Drug Id to update: ")
            name = self.view.get_input("New name: ")
            manuf_id = self.view.get_int("New ManufacturerId: ")
            price = self.view.get_float("New price: ")
            self.model.update_drug(id_, name, manuf_id, price)
        elif choice == 6:
            pharmacy_id = self.view.get_int("Availability: PharmacyId: ")
            drug_id = self.view.get_int("Availability: DrugId: ")
            qty = self.view.get_int("New quantity: ")
            self.model.update_availability(pharmacy_id, drug_id, qty)
        elif choice == 7:
            break
```

- **Меню видалення** (delete_menu): Обробка запитів на видалення та виклик методів відображення результату або помилок.

```
def delete_menu(self):
    while True:
        self.view.show_message("\n-- DELETE MENU --")
        self.view.show_message("1. Delete Pharmacy (by Id)")
        self.view.show_message("2. Delete PhoneType (by Id)")
        self.view.show_message("3. Delete Phone (by Id)")
        self.view.show_message("4. Delete Manufacturer (by Id)")
        self.view.show_message("5. Delete Drug (by Id)")
        self.view.show_message("6. Delete Availability")
        self.view.show_message("7. Back")

        choice = self.view.get_int("Enter choice: ", 1, 7)
        table_map = {
            1: ("Pharmacies", "Id"),
            2: ("PhoneTypes", "Id"),
            3: ("Phones", "Id"),
            4: ("Manufacturers", "Id"),
            5: ("Drugs", "Id")
        }

        if choice in table_map:
```

```
            table, col = table_map[choice]
            id_ = self.view.get_int(f"{table} ID to delete: ")
            status = self.model.delete_by_id(table, col, id_)
            if status == "OK":
                self.view.show_message("Deleted successfully.")
            else:
                self.view.show_error(status)

        elif choice == 6:
            pharmacy_id = self.view.get_int("PharmacyId: ")
            drug_id = self.view.get_int("DrugId: ")
            self.model.delete_availability(pharmacy_id, drug_id)
            self.view.show_message("Deleted (if existed).")

        elif choice == 7:
            break
```

- **Меню генерації** (generate_menu): Виклик процедур пакетного створення тестових даних.

```
def generate_menu(self):
    while True:
        self.view.show_message("\n-- GENERATE MENU --")
        self.view.show_message("1. Generate PhoneTypes (defaults)")
        self.view.show_message("2. Generate Pharmacies")
        self.view.show_message("3. Generate Manufacturers")
        self.view.show_message("4. Generate Drugs")
        self.view.show_message("5. Generate Phones")
        self.view.show_message("6. Generate Availabilities")
        self.view.show_message("7. Back")
        choice = self.view.get_int("Enter choice: ", 1, 7)
        if choice == 1:
            self.model.ensure_default_phonetypes()
        elif choice == 2:
            n = self.view.get_int("How many pharmacies to generate: ", 1, 10000)
            self.model.generate_pharmacies(n)
        elif choice == 3:
            n = self.view.get_int("How many manufacturers to generate: ", 1, 10000)
```

```
            self.model.generate_manufacturers(n)
        elif choice == 4:
            n = self.view.get_int("How many drugs to generate: ", 1, 10000)
            self.model.generate_drugs(n)
        elif choice == 5:
            n = self.view.get_int("How many phones to generate: ", 1, 10000)
            self.model.generate_phones(n)
        elif choice == 6:
            n = self.view.get_int("How many availability rows to generate: ", 1,
10000)
            self.model.generate_availabilities(n)
        elif choice == 7:
            break
```

- **Меню пошуку та аналітики** (search_menu): Інтерфейс для виконання складних пошукових запитів та фільтрації даних.

```
def search_menu(self):
    while True:
        self.view.show_message("\n-- SEARCH & ANALYTICS --")
        self.view.show_message("1. Find Drugs (Price Range + Name)")
        self.view.show_message("2. Find Available Drugs in Pharmacies (Stock
check)")
        self.view.show_message("3. Analyze Updates by Date Range")
        self.view.show_message("4. Back")

        choice = self.view.get_int("Enter choice: ", 1, 4)

        if choice == 1:
            min_p = self.view.get_float("Min Price: ")
            max_p = self.view.get_float("Max Price: ")
            name = self.view.get_input("Drug name part (or empty): ")
            rows = self.model.search_drug_stats(min_p, max_p, name)
            self.view.show_custom_results("Drug Search Results", ["Drug",
"Manufacturer", "Price"], rows)

        elif choice == 2:
            name = self.view.get_input("Drug name part: ")
```

```
            min_q = self.view.get_int("Minimum quantity: ")
            rows = self.model.search_pharmacy_stock(name, min_q)
            self.view.show_custom_results("Stock Availability", ["Pharmacy",
"Drug", "Quantity"], rows)

        elif choice == 3:
            d1 = self.view.get_date("Start Date")
            d2 = self.view.get_date("End Date")
            rows = self.model.search_updates_by_date(d1, d2)
            self.view.show_custom_results("Update Activity", ["Pharmacy",
"Updates Count"], rows)

        elif choice == 4:
            break
```

Модуль view.py

Даний модуль відповідає за взаємодію з користувачем (User Interface). Він виконує вивід повідомлень, таблиць даних та помилок у консоль, а також забезпечує зчитування та первинну валідацію введених даних.

- **Вивід повідомлень**: Метод show_message для інформування користувача про стан операцій.

```
def show_message(self, msg):
    print(f"\n>>> {msg}")
```

- **Валідація введення**: Методи get_int, get_float, get_input, get_date для контролю типів даних та діапазонів значень.

```
def get_input(self, prompt):
    return input(prompt)

def get_int(self, prompt, min_val=None, max_val=None):
    while True:
        try:
            v = int(input(prompt))
            if min_val is not None and v < min_val:
                print("Value too small.")
                continue
```

```
            if max_val is not None and v > max_val:
                print("Value too large.")
                continue
            return v
        except Exception:
            print("Invalid integer. Try again.")

def get_float(self, prompt):
    while True:
        try:
            return float(input(prompt))
        except Exception:
            print("Invalid float. Try again.")
def get_date(self, prompt):
    while True:
        val = input(prompt + " (YYYY-MM-DD): ")
        try:
            return val  # PostgreSQL прийме рядок 'YYYY-MM-DD', або можна
парсити в datetime
        except ValueError:
            print("Invalid date format.")
```

- **Відображення таблиць**: Спеціалізовані методи (show_pharmacies, show_drugs тощо) для форматованого виводу списків сутностей.

```
def show_pharmacies(self, rows):
    if not rows:
        print("No pharmacies.")
        return
    for r in rows:
        print(f"Id: {r[0]} | Name: {r[1]} | Adress: {r[2]}")
def show_drugs(self, rows):
    if not rows:
        print("No drugs.")
        return
    for r in rows:
        print(f"Id: {r[0]} | Name: {r[1]} | ManufacturerId: {r[2]} | Price: {r[3]}")
```

- **Відображення результатів пошуку**: Універсальний метод show_custom_results для виводу аналітичних вибірок.

```
def show_custom_results(self, title, headers, rows):
    print(f"\n--- {title} ---")
    if not rows:
        print("No results found.")
        return

    # Виводимо заголовки
    header_str = " | ".join(headers)
    print(header_str)
    print("-" * len(header_str))

    # Виводимо дані
    for r in rows:
        # Перетворюємо всі елементи в стрічки для друку
        row_str = " | ".join(str(x) for x in r)
        print(row_str)
```

- **Обробка помилок**: Метод show_error для інтерпретації кодів помилок (наприклад, FK_ERROR) у зрозумілі текстові повідомлення для користувача.

```
def show_error(self, err_code):
    if err_code == "FK_ERROR":
        print("\n[!] ПОМИЛКА: Неможливо видалити або додати запис.")
        print("    Причина: Порушення цілісності даних (Foreign Key).")
        print("    Якщо видаляєте: Спочатку видаліть залежні дані в інших
таблицях.")
        print("    Якщо додаєте: Перевірте правильність ID батьківського
запису.")

    elif err_code == "NOT_FOUND":
        print("\n[!] ПОМИЛКА: Запис із таким ID не знайдено.")
        print("    Нічого не було видалено.")

    else:
        print(f"\n[!] Error: {err_code}")
```

Модуль model.py

Даний модуль відповідає за безпосередню взаємодію з системою управління базами даних PostgreSQL. Він містить SQL-запити, логіку транзакцій та обробку виключних ситуацій на рівні БД.

- **Ініціалізація та з'єднання**: Встановлення підключення до БД через бібліотеку psycopg2.

```
import psycopg2
from psycopg2 import sql, errors
from functools import wraps
import time
```

- **Декоратор вимірювання часу** (@timeit): Фіксація часу виконання SQL-запитів для аналізу швидкодії.

```
def timeit(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        start = time.time()
        res = func(*args, **kwargs)
        end = time.time()
        print(f"[{func.__name__}] executed in {(end - start) * 1000:.2f} ms")
        return res
    return wrapper
```

- **Допоміжні методи виконання запитів**: Методи fetch_query (читання) та execute_query (зміна) з підтримкою комітів та відкатів (rollback).

```
def fetch_query(self, query, params=None, fetch_all=True):
    cur = self.conn.cursor()
    try:
        cur.execute(query, params or ())
        rows = cur.fetchall() if fetch_all else cur.fetchmany(50)
        cur.close()
        return rows
    except Exception as e:
        cur.close()
```

```
        print("Error fetch_query:", e)
        return []

def execute_query(self, query, params=None):
    cur = self.conn.cursor()
    try:
        cur.execute(query, params or ())
        self.conn.commit()
        cur.close()
        return "OK"
    except errors.ForeignKeyViolation as e:
        self.conn.rollback()
        cur.close()
        return "FK_ERROR" # Повертаємо спеціальний код помилки
    except Exception as e:
        self.conn.rollback()
        cur.close()
        print("Error execute_query:", e)
        return str(e)
```

- **Методи читання даних (CRUD - Read)**: Вибірка даних з таблиць з сортуванням та лімітами.

```
# ---------- READ ----------
@timeit
def get_pharmacies_limited(self):
    return self.fetch_query('SELECT "Id", "Name", "Adress" FROM "Pharmacies" ORDER BY "Id" LIMIT 50')

@timeit
def get_pharmacies(self):
    return self.fetch_query('SELECT "Id", "Name", "Adress" FROM "Pharmacies" ORDER BY "Id"')

@timeit
def get_pharmacy_by_id(self, id_val):
    return self.fetch_query('SELECT "Id", "Name", "Adress" FROM "Pharmacies" WHERE "Id"=%s', (id_val,))
```

```
@timeit
def get_phonetypes(self):
    return self.fetch_query('SELECT "Id", "Name" FROM "PhoneTypes"
ORDER BY "Id"')

@timeit
def get_phones(self):
    return self.fetch_query('SELECT "Id", "Name", "PharmacyId",
"PhoneTypeId" FROM "Phones" ORDER BY "Id"')

@timeit
def get_manufacturers(self):
    return self.fetch_query('SELECT "Id", "Name" FROM "Manufacturers"
ORDER BY "Id"')

@timeit
def get_drugs(self):
    return self.fetch_query('SELECT "Id", "Name", "ManufacturerId", "Price"
FROM "Drugs" ORDER BY "Id"')

@timeit
def get_availabilities(self):
    return self.fetch_query('SELECT "PharmacyId", "DrugId", "Quantity",
"UpdatedAt" FROM "Availabilities" ORDER BY "PharmacyId", "DrugId"')
```

- **Методи модифікації даних (CRUD - Create, Update, Delete)**: Параметризовані SQL-запити для вставки, оновлення та видалення записів.

```
@timeit
def add_pharmacy(self, name, adress):
    # Отримати максимальний Id
    max_id_query = 'SELECT COALESCE(MAX("Id"), 0) FROM
"Pharmacies"'
    rows = self.fetch_query(max_id_query)
    next_id = rows[0][0] + 1 if rows else 1
```

```
    return self.execute_query(
        'INSERT INTO "Pharmacies" ("Id", "Name", "Adress") OVERRIDING
SYSTEM VALUE VALUES (%s, %s, %s)',
        (next_id, name, adress)
    )

@timeit
def add_phonetype(self, name):
    max_id_query = 'SELECT COALESCE(MAX("Id"), 0) FROM
"PhoneTypes"'
    rows = self.fetch_query(max_id_query)
    next_id = rows[0][0] + 1 if rows else 1

    return self.execute_query(
        'INSERT INTO "PhoneTypes" ("Id", "Name") OVERRIDING SYSTEM
VALUE VALUES (%s, %s)',
        (next_id, name)
    )

@timeit
def add_phone(self, name, pharmacyid, phonetypeid):
    max_id_query = 'SELECT COALESCE(MAX("Id"), 0) FROM "Phones"'
    rows = self.fetch_query(max_id_query)
    next_id = rows[0][0] + 1 if rows else 1

    return self.execute_query(
        'INSERT INTO "Phones" ("Id", "Name", "PharmacyId", "PhoneTypeId")
OVERRIDING SYSTEM VALUE VALUES (%s, %s, %s, %s)',
        (next_id, name, pharmacyid, phonetypeid)
    )

@timeit
def add_manufacturer(self, name):
    max_id_query = 'SELECT COALESCE(MAX("Id"), 0) FROM
"Manufacturers"'
    rows = self.fetch_query(max_id_query)
    next_id = rows[0][0] + 1 if rows else 1
```

```
    return self.execute_query(
        'INSERT INTO "Manufacturers" ("Id", "Name") OVERRIDING SYSTEM
VALUE VALUES (%s, %s)',
        (next_id, name)
    )

@timeit
def add_drug(self, name, manufactureid, price):
    max_id_query = 'SELECT COALESCE(MAX("Id"), 0) FROM "Drugs"'
    rows = self.fetch_query(max_id_query)
    next_id = rows[0][0] + 1 if rows else 1

    # ВИПРАВЛЕННЯ
    return self.execute_query(
        'INSERT INTO "Drugs" ("Id", "Name", "ManufacturerId", "Price")
OVERRIDING SYSTEM VALUE VALUES (%s, %s, %s, %s)',
        (next_id, name, manufactureid, price)
    )

@timeit
def add_availability(self, pharmacyid, drugid, quantity):
    return self.execute_query(
        'INSERT INTO "Availabilities" ("PharmacyId", "DrugId", "Quantity",
"UpdatedAt") VALUES (%s, %s, %s, now())',
        (pharmacyid, drugid, quantity)
    )

# ---------- SEARCH & ANALYTICS ----------
@timeit
def search_drug_stats(self, min_price, max_price, name_pattern):
    query = """
            SELECT d."Name", m."Name", d."Price"
            FROM "Drugs" d
                    JOIN "Manufacturers" m ON d."ManufacturerId" = m."Id"
            WHERE d."Price" BETWEEN %s AND %s
              AND d."Name" LIKE %s
```

```
            ORDER BY d."Price" DESC \
            """
    return self.fetch_query(query, (min_price, max_price,
f"%{name_pattern}%"))

@timeit
def search_pharmacy_stock(self, drug_name_part, min_qty):
    query = """
            SELECT p."Name", d."Name", a."Quantity"
            FROM "Availabilities" a
                    JOIN "Pharmacies" p ON a."PharmacyId" = p."Id"
                    JOIN "Drugs" d ON a."DrugId" = d."Id"
            WHERE d."Name" LIKE %s \
              AND a."Quantity" >= %s
            ORDER BY a."Quantity" DESC \
            """
    return self.fetch_query(query, (f"%{drug_name_part}%", min_qty))

@timeit
def search_updates_by_date(self, start_date, end_date):
    query = """
            SELECT p."Name", count(a."DrugId") as UpdatedCount
            FROM "Availabilities" a
                    JOIN "Pharmacies" p ON a."PharmacyId" = p."Id"
            WHERE a."UpdatedAt" BETWEEN %s AND %s
            GROUP BY p."Name"
            ORDER BY UpdatedCount DESC \
            """
    return self.fetch_query(query, (start_date, end_date))

# ---------- UPDATE ----------
@timeit
def update_pharmacy(self, id_, name, adress):
    return self.execute_query('UPDATE "Pharmacies" SET "Name"=%s,
"Adress"=%s WHERE "Id"=%s', (name, adress, id_))

@timeit
```

```
def update_phonetype(self, id_, name):
    return self.execute_query('UPDATE "PhoneTypes" SET "Name"=%s
WHERE "Id"=%s', (name, id_))

@timeit
def update_phone(self, id_, name, pharmacyid, phonetypeid):
    return self.execute_query('UPDATE "Phones" SET "Name"=%s,
"PharmacyId"=%s, "PhoneTypeId"=%s WHERE "Id"=%s', (name, pharmacyid,
phonetypeid, id_))

@timeit
def update_manufacturer(self, id_, name):
    return self.execute_query('UPDATE "Manufacturers" SET "Name"=%s
WHERE "Id"=%s', (name, id_))

@timeit
def update_drug(self, id_, name, manufactureid, price):
    return self.execute_query('UPDATE "Drugs" SET "Name"=%s,
"ManufactureId"=%s, "Price"=%s WHERE "Id"=%s', (name, manufactureid,
price, id_))

@timeit
def update_availability(self, pharmacyid, drugid, quantity):
    return self.execute_query('UPDATE "Availabilities" SET "Quantity"=%s,
"UpdatedAt"=now() WHERE "PharmacyId"=%s AND "DrugId"=%s',
(quantity, pharmacyid, drugid))

# ---------- DELETE ----------
@timeit
def delete_by_id(self, table, id_field, id_value):
    q = sql.SQL('DELETE FROM {} WHERE {} =
%s').format(sql.Identifier(table), sql.Identifier(id_field))
    cur = self.conn.cursor()
    try:
        cur.execute(q, (id_value,))

        rows_deleted = cur.rowcount
```

```
            self.conn.commit()
            cur.close()

            if rows_deleted == 0:
                return "NOT_FOUND"
            return "OK"

        except errors.ForeignKeyViolation:
            self.conn.rollback()
            cur.close()
            return "FK_ERROR"
        except Exception as e:
            self.conn.rollback()
            cur.close()
            return str(e)
```

- **Генерація даних**: Виконання складних SQL-запитів із використанням generate_series, random() та CROSS JOIN для наповнення бази великим обсягом даних.

```
@timeit
def ensure_default_phonetypes(self):
    # insert basic phone types if table empty
    rows = self.fetch_query('SELECT count(*) FROM "PhoneTypes"')
    if rows and rows[0][0] == 0:
        types = ['mobile', 'landline', 'fax']
        for t in types:
            self.execute_query('INSERT INTO "PhoneTypes" ("Name") VALUES
(%s)', (t,))
        print("Default PhoneTypes inserted.")
    else:
        print("PhoneTypes not empty, skipped.")

@timeit
def generate_pharmacies(self, n):
    max_id_query = 'SELECT COALESCE(MAX("Id"), 0) FROM
"Pharmacies"'
```

```
    rows = self.fetch_query(max_id_query)
    start_id = rows[0][0] + 1 if rows else 1

    q = f"""
    INSERT INTO "Pharmacies" ("Id", "Name", "Adress")
    OVERRIDING SYSTEM VALUE
    SELECT
        {start_id} + gs - 1,
        'Pharmacy_' || gs,
        'Address ' || gs
    FROM generate_series(1, {int(n)}) AS gs;
    """
    return self.execute_query(q)

@timeit
def generate_manufacturers(self, n):
    max_id_query = 'SELECT COALESCE(MAX("Id"), 0) FROM
"Manufacturers"'
    rows = self.fetch_query(max_id_query)
    start_id = rows[0][0] + 1 if rows else 1

    q = f"""
    INSERT INTO "Manufacturers" ("Id", "Name")
    SELECT
        {start_id} + gs - 1,
        'Manufacturer_' || gs
    FROM generate_series(1, {int(n)}) AS gs;
    """
    return self.execute_query(q)

def generate_drugs(self, n):
    query = f"""
        WITH max_id AS (
            SELECT COALESCE(MAX("Id"), 0) AS start_id FROM "Drugs"
        ),
        generated AS (
            SELECT
```

```
            (ROW_NUMBER() OVER () + (SELECT start_id FROM max_id))
AS "Id",
            'Drug_' || (ROW_NUMBER() OVER ()) AS "Name",
            (SELECT "Id" FROM "Manufacturers" ORDER BY random() LIMIT
1) AS "ManufacturerId",
            round((random() * 200)::numeric, 2) AS "Price"
        FROM generate_series(1, {n})
    )
    INSERT INTO "Drugs" ("Id", "Name", "ManufacturerId", "Price")
    SELECT "Id", "Name", "ManufacturerId", "Price" FROM generated;
    """
    self.execute_query(query)

@timeit
def generate_phones(self, n):
    max_id_query = 'SELECT COALESCE(MAX("Id"), 0) FROM "Phones"'
    rows = self.fetch_query(max_id_query)
    start_id = rows[0][0] + 1 if rows else 1

    q = f"""
    INSERT INTO "Phones" ("Id", "Name", "PharmacyId", "PhoneTypeId")
    SELECT
        {start_id} + gs - 1,
        'Phone_' || gs,
        ph.ids[1 + floor(random() * array_length(ph.ids, 1))::int],
        pt.ids[1 + floor(random() * array_length(pt.ids, 1))::int]
    FROM generate_series(1, {int(n)}) AS gs
    CROSS JOIN (SELECT array_agg("Id") as ids FROM "Pharmacies") ph
    CROSS JOIN (SELECT array_agg("Id") as ids FROM "PhoneTypes") pt;
    """
    return self.execute_query(q)

@timeit
def generate_availabilities(self, n):
    """Генерує або оновлює записи про наявність"""
    q = f"""
    INSERT INTO "Availabilities" ("PharmacyId", "DrugId", "Quantity",
```

```
"UpdatedAt")
    SELECT
        p."Id",
        d."Id",
        (floor(random()*100)::int + 1),
        now()
    FROM (SELECT "Id" FROM "Pharmacies" ORDER BY random() LIMIT
{int(n)}) p,
         (SELECT "Id" FROM "Drugs" ORDER BY random() LIMIT {int(n)}) d
    LIMIT {int(n)}
    ON CONFLICT ("PharmacyId", "DrugId")
    DO UPDATE SET
        "Quantity" = EXCLUDED."Quantity",
        "UpdatedAt" = EXCLUDED."UpdatedAt";
    """
    return self.execute_query(q)
```

- **Пошук та аналітика**: Реалізація складних запитів із використанням JOIN, GROUP BY та фільтрації за діапазонами (цін, дат) та шаблонами рядків (LIKE).

```
@timeit
def search_drug_stats(self, min_price, max_price, name_pattern):
    query = """
            SELECT d."Name", m."Name", d."Price"
            FROM "Drugs" d
                    JOIN "Manufacturers" m ON d."ManufacturerId" = m."Id"
            WHERE d."Price" BETWEEN %s AND %s
              AND d."Name" LIKE %s
            ORDER BY d."Price" DESC \
            """
    return self.fetch_query(query, (min_price, max_price,
f"%{name_pattern}%"))

@timeit
def search_pharmacy_stock(self, drug_name_part, min_qty):
    query = """
```

```
        SELECT p."Name", d."Name", a."Quantity"
        FROM "Availabilities" a
            JOIN "Pharmacies" p ON a."PharmacyId" = p."Id"
            JOIN "Drugs" d ON a."DrugId" = d."Id"
        WHERE d."Name" LIKE %s \
          AND a."Quantity" >= %s
        ORDER BY a."Quantity" DESC \
        """
    return self.fetch_query(query, (f"%{drug_name_part}%", min_qty))

@timeit
def search_updates_by_date(self, start_date, end_date):
    query = """
        SELECT p."Name", count(a."DrugId") as UpdatedCount
        FROM "Availabilities" a
            JOIN "Pharmacies" p ON a."PharmacyId" = p."Id"
        WHERE a."UpdatedAt" BETWEEN %s AND %s
        GROUP BY p."Name"
        ORDER BY UpdatedCount DESC \
        """
    return self.fetch_query(query, (start_date, end_date))
```

Тестування

Задля підтвердження коректності виконання програми, необхідно провести тестування кожного з пунктів меню користувача.

Проведемо тестування 1-го пункту меню. Занесемо необхідні параметри задля отримання конкретних даних таблиць.

Внесення тестових даних 1-го пункту користувацького меню зображено на рисунку нижче:

```
>>>
-- SHOW MENU --

>>> 1. Pharmacies

>>> 2. PhoneTypes

>>> 3. Phones

>>> 4. Manufacturers

>>> 5. Drugs

>>> 6. Availabilities

>>> 7. Back
Enter choice: 1

--- VIEW MODES ---
1. Show limit 50 (default)
2. Show ALL
3. Show ONE by ID
Select mode:
```

Рисунок 6 – Внесення тестових даних 1-го пункту користувацького меню

Результати роботи програми після внесення даних до 2-го пункту користувацького меню зображено на рисунку нижче:

```
--- VIEW MODES ---
1. Show limit 50 (default)
2. Show ALL
3. Show ONE by ID
Select mode: 2
[get_pharmacies] executed in 49.60 ms
Id: 1 | Name:  | Adress: appa
Id: 2 | Name: Pharmacy_1 | Adress: Address 1
Id: 3 | Name: Pharmacy_2 | Adress: Address 2
Id: 4 | Name: Pharmacy_3 | Adress: Address 3
Id: 5 | Name: Pharmacy_1 | Adress: Address 1
Id: 6 | Name: Pharmacy_2 | Adress: Address 2
Id: 7 | Name: Pharmacy_3 | Adress: Address 3
Id: 8 | Name: Pharmacy_1 | Adress: Address 1
Id: 9 | Name: Pharmacy_2 | Adress: Address 2
Id: 10 | Name: Pharmacy_3 | Adress: Address 3
Id: 11 | Name: Pharmacy_4 | Adress: Address 4
Id: 12 | Name: Pharmacy_1 | Adress: Address 1
Id: 13 | Name: Pharmacy_2 | Adress: Address 2
Id: 14 | Name: Pharmacy_1 | Adress: Address 1
```

Рисунок 7 – Результати роботи програми після внесення даних до 1-го пункту користувацького меню

Порівняння результатів роботи програми та загального вигляду таблиці users у PostgreSQL зображено на рисунку нижче:

	Id [PK] integer	Name character varying (100)	Adress character varying (200)
1	1		appa
2	2	Pharmacy_1	Address 1
3	3	Pharmacy_2	Address 2
4	4	Pharmacy_3	Address 3
5	5	Pharmacy_1	Address 1
6	6	Pharmacy_2	Address 2
7	7	Pharmacy_3	Address 3
8	8	Pharmacy_1	Address 1
9	9	Pharmacy_2	Address 2
10	10	Pharmacy_3	Address 3
11	11	Pharmacy_4	Address 4
12	12	Pharmacy_1	Address 1
13	13	Pharmacy_2	Address 2
14	14	Pharmacy_1	Address 1

Рисунок 8 – Порівняння результатів роботи програми та загального вигляду таблиці Users у PostgreSQL

Внесення тестових даних 1-го пункту користувацького меню зображено на рисунку нижче:

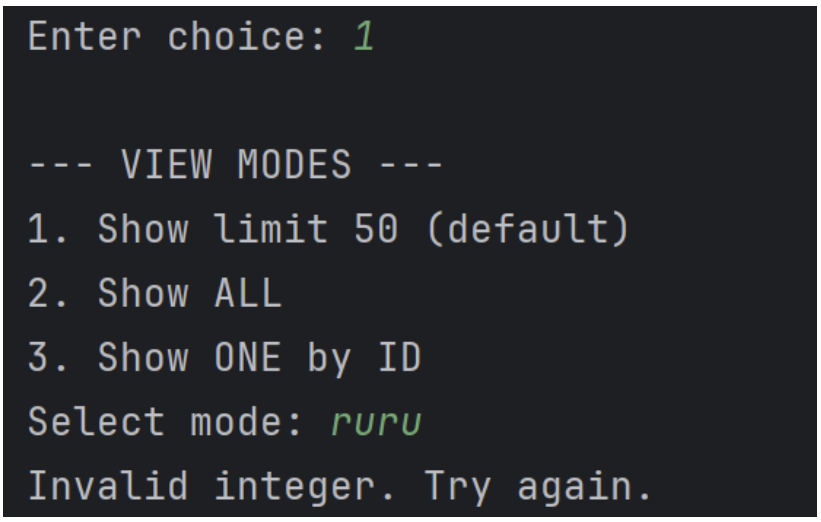

Рисунок 9 – Результат роботи програми внаслідок введення некоректних даних

Проведемо тестування 2-го пункту меню за подібним принципом

Внесення тестових даних 2-го пункту користувацького меню зображено на рисунку нижче:

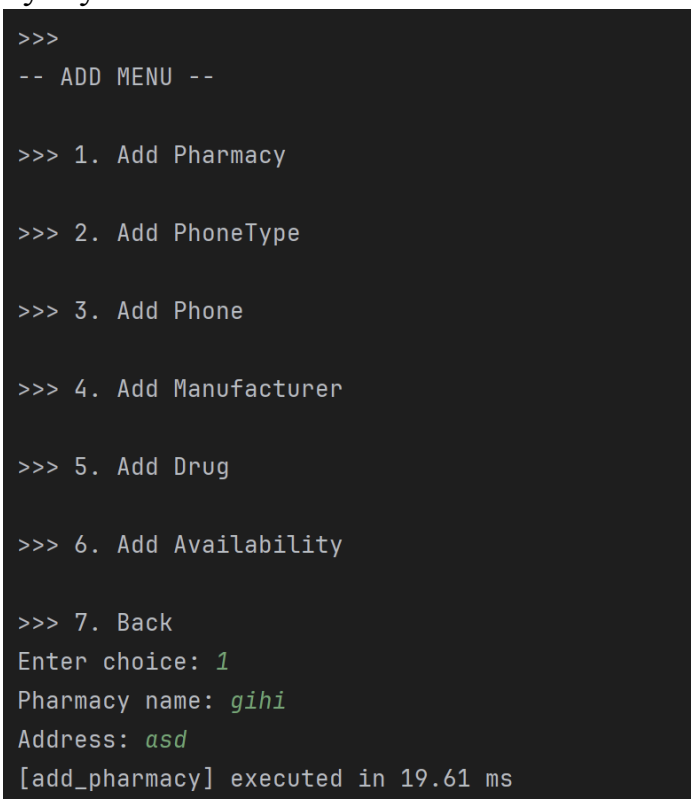

Рисунок 10 – Внесення тестових даних 2-го пункту користувацького меню

Результат роботи програми на прикладі отриманих даних PostgreSQL зображено на рисунку нижче:

15	15	gihi	asd

Рисунок 11 – Результат роботи програми на прикладі отриманих даних PostgreSQL

Внесення некоректних даних до 2-го пункту користувацького меню зображено на рисунку нижче:

```
>>>
-- ADD MENU --

>>> 1. Add Pharmacy

>>> 2. Add PhoneType

>>> 3. Add Phone

>>> 4. Add Manufacturer

>>> 5. Add Drug

>>> 6. Add Availability

>>> 7. Back
Enter choice: 5
Drug name: tyt
ManufacturerId: 99999
Price: 12
[add_drug] executed in 44.23 ms

[!] ПОМИЛКА: Неможливо видалити або додати запис.
    Причина: Порушення цілісності даних (Foreign Key).
    Якщо видаляєте: Спочатку видаліть залежні дані в інших таблицях.
    Якщо додаєте: Перевірте правильність ID батьківського запису.
```

Рисунок 12 – Внесення некоректних даних до 2-го пункту користувацького меню

Проведемо тестування 3-го пункту меню «Update data» (Оновлення даних). Змінимо атрибути обраного запису (наприклад, змінимо назву та адресу аптеки).

Процес введення нових даних для оновлення запису та результат успішного виконання операції зображено на рисунку нижче:

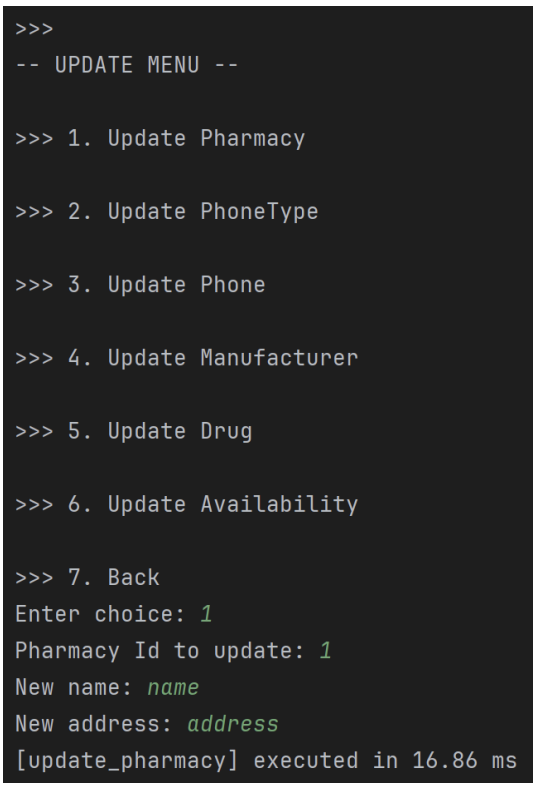

Рисунок 13 – Внесення змін до існуючого запису та повідомлення про успішне оновлення запису

1	1	name	address

Рисунок 14 – Відображення оновлених даних у середовищі PostgreSQL

Проведемо тестування 4-го пункту меню «Delete data» (Видалення даних). Згідно з технічним завданням, необхідно перевірити два сценарії: успішне видалення та спроба видалення запису, що має залежні дані (Foreign Key Constraint).

Сценарій 1. Спроба видалення запису (наприклад, Аптеки), що має пов'язані записи у таблиці наявності ліків.

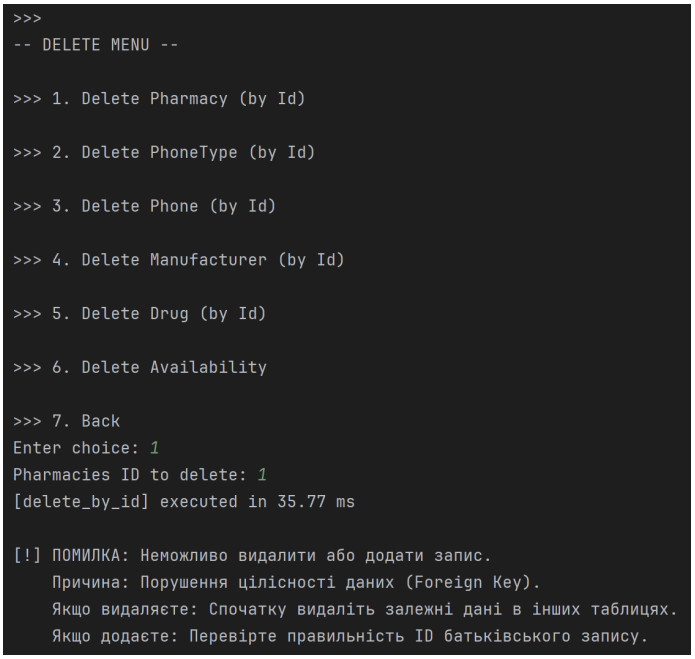

Рисунок 15 – Результат перехоплення помилки порушення цілісності (Foreign Key) при видаленні

Сценарій 2. Успішне видалення запису, що не має залежностей.

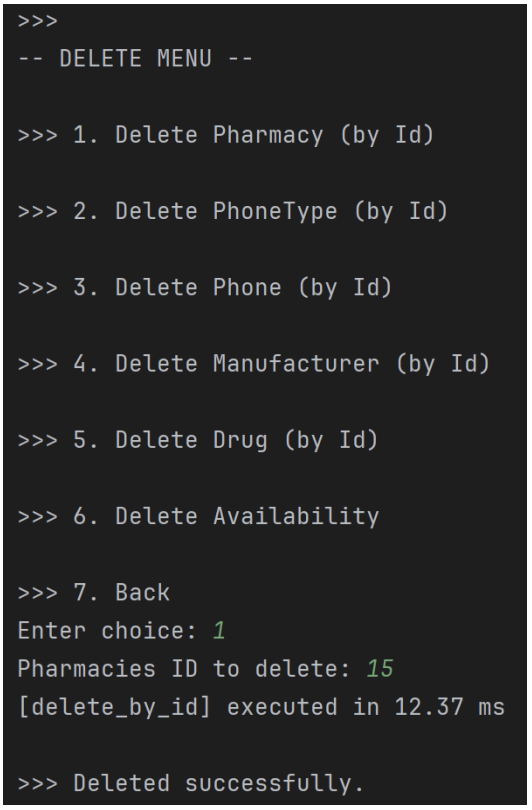

Рисунок 16 – Результат успішного видалення незалежного запису

Проведемо тестування 5-го пункту меню «Generate data» (Генерація даних). Перевіримо швидкість виконання пакетної вставки даних засобами SQL.

Процес генерації масиву даних та час виконання запиту зображено на рисунку нижче:

```
>>>
-- GENERATE MENU --

>>> 1. Generate PhoneTypes (defaults)

>>> 2. Generate Pharmacies

>>> 3. Generate Manufacturers

>>> 4. Generate Drugs

>>> 5. Generate Phones

>>> 6. Generate Availabilities

>>> 7. Back
Enter choice: 6
How many availability rows to generate: 10000
[generate_availabilities] executed in 213.82 ms
```

Рисунок 17 – Генерація великого обсягу даних та час виконання операції

Результат наповнення таблиці у базі даних зображено на рисунку нижче:

	PharmacyId [PK] integer	DrugId [PK] integer	Quantity integer	UpdatedAt date
1	1	1	22	2025-11-06
2	1	4	93	2025-11-07
3	1	5	1	2025-12-03
4	1	6	67	2025-11-07
5	1	7	50	2025-12-03
6	1	10	82	2025-11-07
7	1	13	99	2025-12-03
8	1	23	29	2025-12-03
9	1	28	87	2025-12-03
10	1	41	83	2025-12-03

Total rows: 10011 | Query complete 00:00:01.217

Рисунок 18 – Перевірка кількості згенерованих записів у PostgreSQL

Проведемо тестування 6-го пункту меню «Search & Analytics», який реалізує складні пошукові запити з використанням об'єднання таблиць та фільтрації.

Введення параметрів пошуку (діапазон цін та частковий збіг назви) та результат виконання пошукового запиту та вивід відфільтрованої таблиці зображено на рисунку нижче:

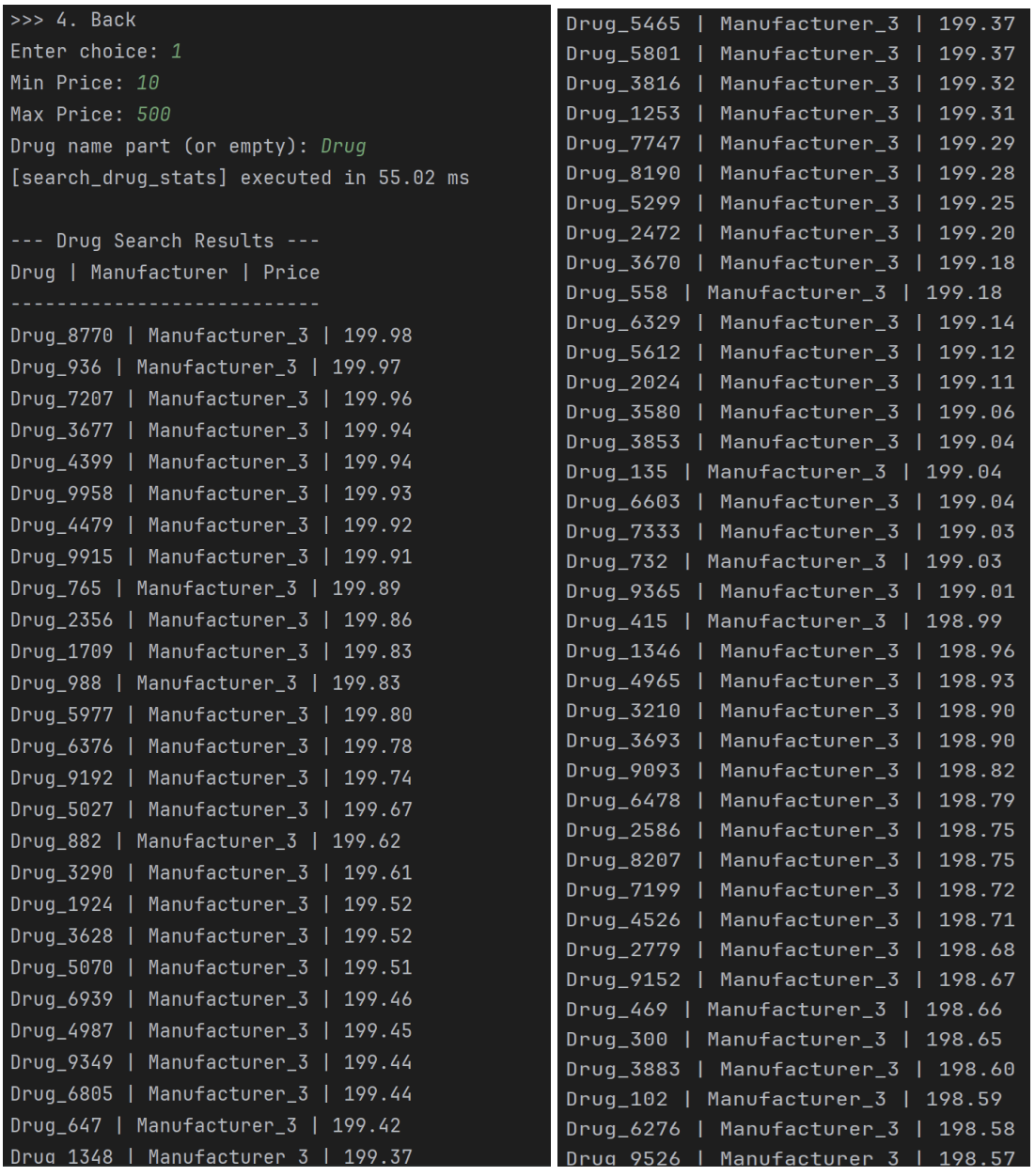

Рисунок 20, 21 – Введення параметрів для розширеного пошуку та Результати виконання пошукового запиту за декількома критеріями

Контакти в телеграм:
Username: @Kalinin_MikhaiI

Електронна пошта:
misha06408@gmail.com