

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 та засоби управління»**

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

Виконала: студентка 3 курсу

ФПМ групи КВ-12

Бурда Є. А.

Київ – 2024

*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).

Github: [evaburda/rgr (github.com)](https://github.com/evaburda/rgr)

Telegram: https://t.me/night\_fool

**Сутності**

**Patient** з атрибутами : ID пацієнта, ім’я, прізвище, дата народження, адреса.

**Medical Worker** з атрибутами: ID працівника, ім'я, прізвище, спеціальність.

**Medical Record** з атрибутами: ID запису, дата та час, опис симптомів, діагноз.

**Medicine** з атрибутами: ID лікарського засобу, назва, дозування

**Опис зв’язків**

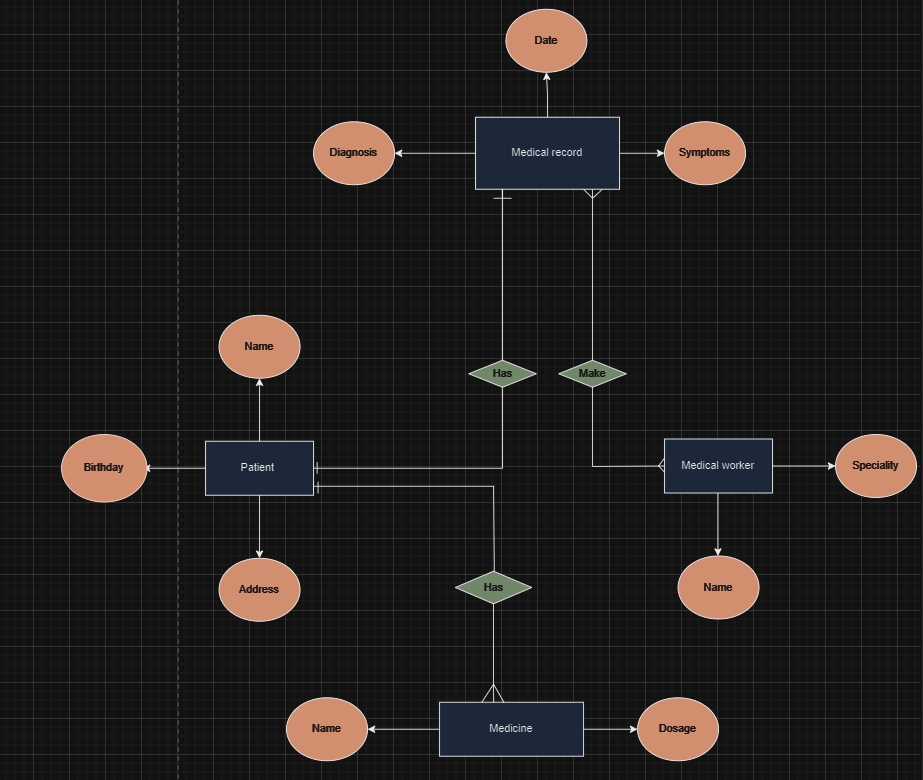
Patient має medical record: вказує на те, що у кожного пацієнта є один медичний запис, та медичний запис відноситься до одного пацієнта.

Medical worker створює medical record: вказує на те, що медичний працівник може створювати багато медичних записів і багато медичних записів створені працівниками

Patient має medicine: вказує на те, що пацієнт може мати декілька видів ліків, але кожні ліки відносяться лише до одного пацієнта.

Medicine – це ліки для кожного пацієнта, вони можуть існувати без пацієнта так само як пацієнт може існувати без ліків.

***ER-діаграма, побудована за нотацією Чена***



***Схема бази даних у графічному вигляді***

**Зображення, що містить текст, знімок екрана, схема, ряд

Автоматично згенерований опис**

*Для роботи було використано:*

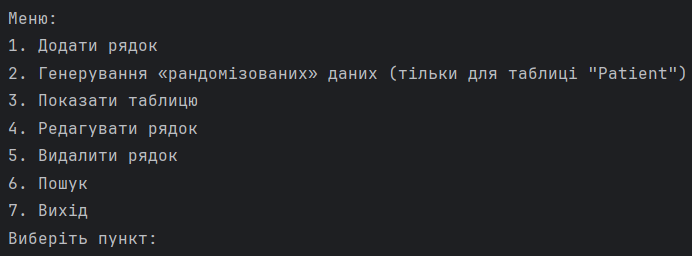
Середовище для відлагодження SQL-запитів до бази даних – PgAdmin4.

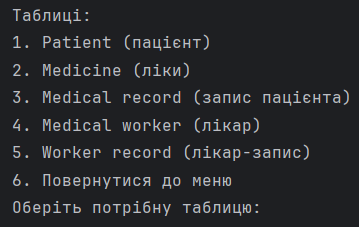
Мова програмування – Python 3.10.

Середовище розробки програмного забезпечення – PyCharm Community Edition.

Бібліотека взаємодії з PostgreSQL - psycopg2.

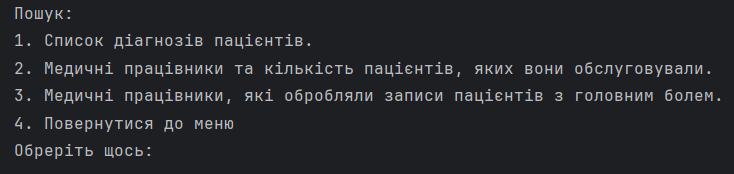
***Меню користувача з описом функціональності кожного пункту***





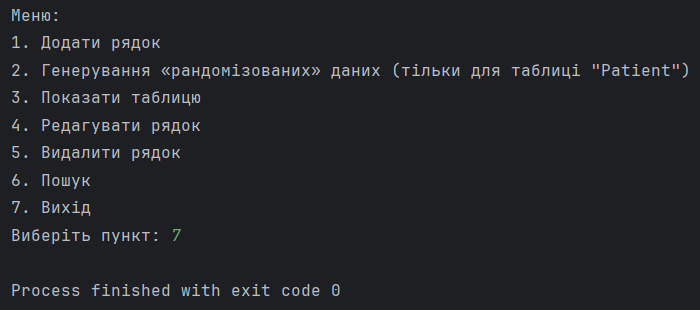
Меню складається з 7 пунктів:

1. **Додати рядок** служить для додавання рядка в таблицю. Після вибору цього пункту, потрібно обрати таблицю, для якої буде виконана ця операція, після чого, треба ввести дані для кожного атрибуту таблиці, щоб додати новий рядок.
2. **Генерування «рандомізованих» даних**. Для цього пункту було обрано таблицю “Patient”. Цей пункт створений для додавання «рандомізованих» даних. Користувачу потрібно ввести число полів, скільки треба додати у таблицю.
3. **Показати таблицю** служить для показу таблиці. Перед виведенням, користувач обирає, яку саме таблицю потрібно вивести. Після цього на екрані виводяться всі поля обраної таблиці БД.
4. **Редагувати рядок** використовується для редагування полів по id у таблицях. Спочатку потрібно обрати, для якої таблиці буде відбуватися, після чого потрібно ввести id поля, яке потрібно змінити. Залишається ввести нові дані для кожного атрибуту таблиці.
5. **Видалити рядок** служить для видалення рядку по id у таблицях. Спочатку потрібно обрати, для якої таблиці буде відбуватися видалення рядка, після чого користувач вводить id рядка, який потрібно видалити.
6. **Пошук** створений для пошуку за атрибутами з декількох таблиць. Пропонується 4 варіанти вибору:



Роботу цих запитів буде розглянуто у завданні №3.

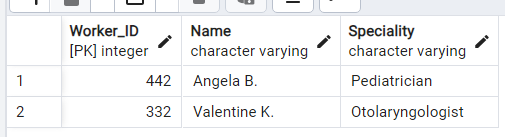
1. **Вихід** служить для виходу з програми.



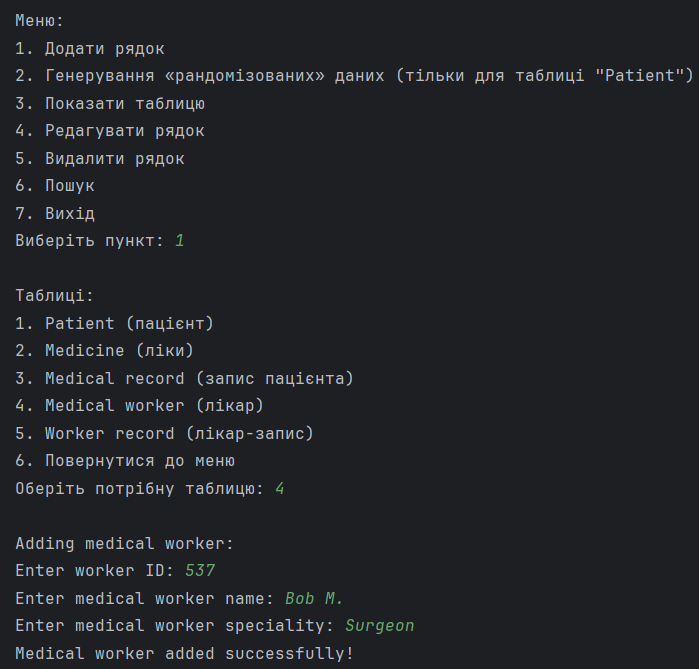
**Завдання 1**

**Додавання рядка**

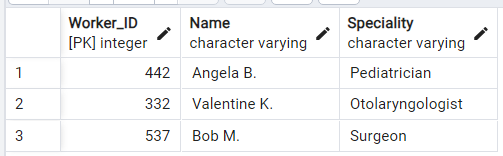
***Таблиця “Medical\_Worker” до:***

******

***Процес додавання рядка:***

******

***Таблиця “Medical\_Worker” після:***

******

***Лістинг add:***

**def** add\_patient(self, patient\_id, name, birthday, address):

c = self.conn.cursor()

c.execute('INSERT INTO "Patients" ("Patient\_ID", "Name", "Birthday", "Address") VALUES (%s, %s, %s, %s)', (patient\_id, name, birthday, address))

self.conn.commit()

**def** add\_medicine(self, medicine\_id, patient\_id, name, dosage):

c = self.conn.cursor()

c.execute('INSERT INTO "Medicine" ("Medicine\_ID", "Patient\_ID", "Name", "Dosage") VALUES (%s, %s, %s, %s)', (medicine\_id, patient\_id, name, dosage))

self.conn.commit()

**def** add\_medical\_record(self, record\_id, patient\_id, date, symptoms, diagnosis):

c = self.conn.cursor()

c.execute('INSERT INTO "Medical\_Record" ("Record\_ID", "Patient\_ID", "Date", "Symptoms", "Diagnosis") VALUES (%s, %s, %s, %s, %s)', (record\_id, patient\_id, date, symptoms, diagnosis))

self.conn.commit()

**def** add\_medical\_worker(self, worker\_id, name, speciality):

c = self.conn.cursor()

c.execute('INSERT INTO "Medical\_Worker" ("Worker\_ID", "Name", "Speciality") VALUES (%s, %s, %s)', (worker\_id, name, speciality))

self.conn.commit()

**def** add\_worker\_record(self, tab\_id, worker\_id, record\_id):

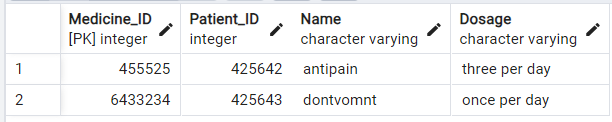
c = self.conn.cursor()

c.execute('INSERT INTO "Worker\_Record" ("Tab\_ID", "Worker\_ID", "Record\_ID") VALUES (%s, %s, %s)', (tab\_id, worker\_id, record\_id))

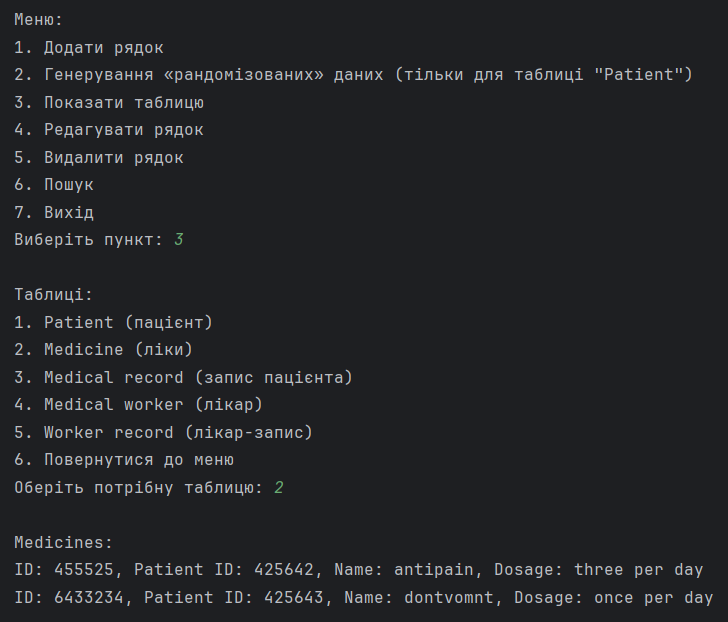
self.conn.commit()

**Перегляд таблиці**

***Таблиця “Medicine”:***

******

***Процес перегляду таблиці:***

******

***Лістинг get:***

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

c = self.conn.cursor()

c.execute('SELECT \* FROM "Patient"')

**return** c.fetchall()

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

c = self.conn.cursor()

c.execute('SELECT \* FROM "Medicine"')

**return** c.fetchall()

**def** get\_all\_medical\_records(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM "Medical\_Record"')

**return** c.fetchall()

**def** get\_all\_medical\_workers(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM "Medical\_Worker"')

**return** c.fetchall()

**def** get\_all\_workers\_records(self):

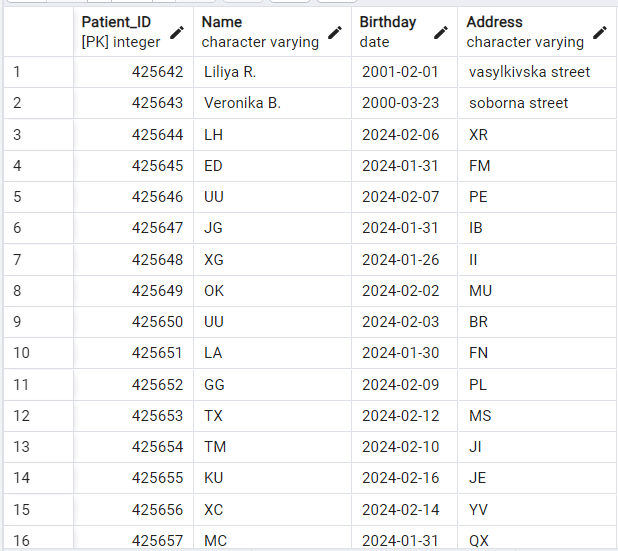
c = self.conn.cursor()

c.execute('SELECT \* FROM "Worker\_Record"')

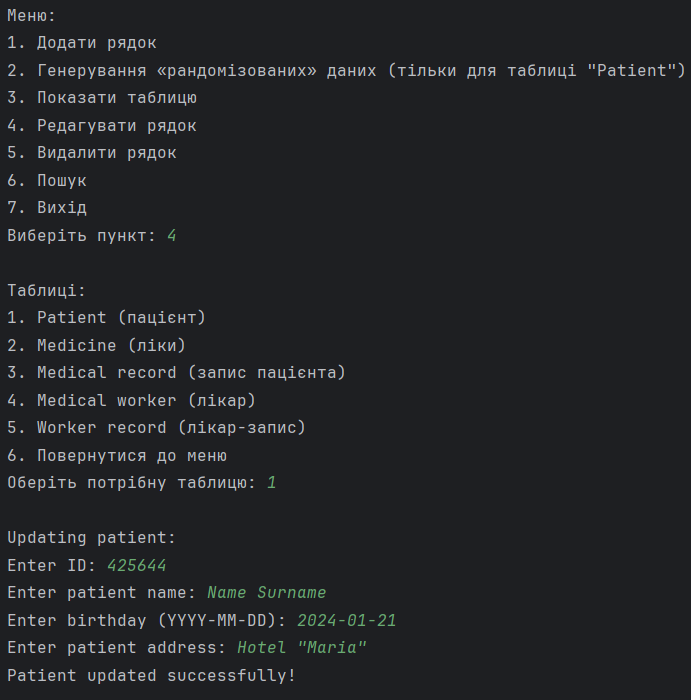
**return** c.fetchall()

**Оновлення рядка**

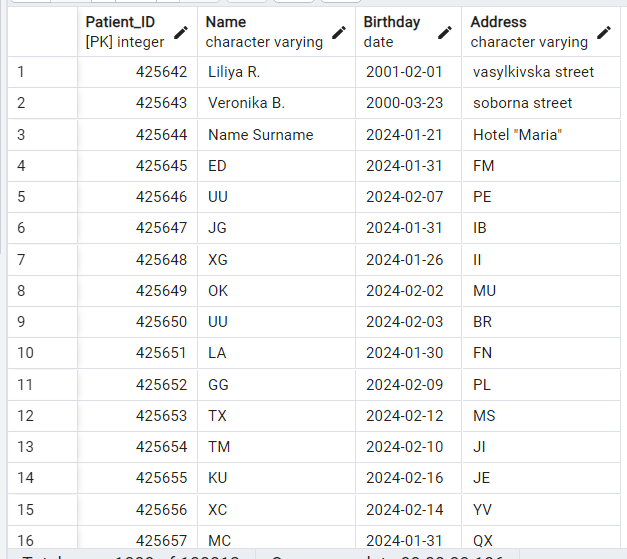
***Таблиця “Patient” до:***

******

***Процес оновлення рядка:***

******

***Таблиця “Patient” після:***

******

***Лістинг update:***

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

c = self.conn.cursor()

c.execute('UPDATE "Patient" SET "Name"=%s, "Birthday"=%s, "Address"=%s WHERE "Patient\_ID"=%s', (name, birthday, address, id))

self.conn.commit()

**def** update\_medicine(self, patient\_id, name, dosage, id):

c = self.conn.cursor()

c.execute('UPDATE "Medicine" SET "Patient\_ID"=%s, "Name"=%s, "Dosage"=%s WHERE "Medicine\_ID"=%s', (patient\_id, name, dosage, id))

self.conn.commit()

**def** update\_medical\_record(self, patient\_id, date, symptoms, diagnosis, id):

c = self.conn.cursor()

c.execute('UPDATE "Medical\_Record" SET "Patient\_ID"=%s, "Date"=%s, "Symptoms"=%s, "Diagnosis"=%s WHERE "Record\_ID"=%s', (patient\_id, date, symptoms, diagnosis, id))

self.conn.commit()

**def** update\_medical\_worker(self, name, speciality, id):

c = self.conn.cursor()

c.execute('UPDATE "Medical\_Worker" SET "Name"=%s, "Speciality"=%s WHERE "Worker\_ID"=%s', (name, speciality, id))

self.conn.commit()

**def** update\_worker\_record(self, worker\_id, record\_id, id):

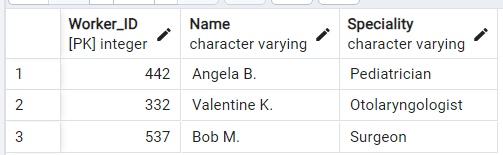
c = self.conn.cursor()

c.execute('UPDATE "Worker\_Record" SET "Worker\_ID"=%s, "Record\_ID"=%s WHERE "Tab\_ID"=%s', (worker\_id, record\_id, id))

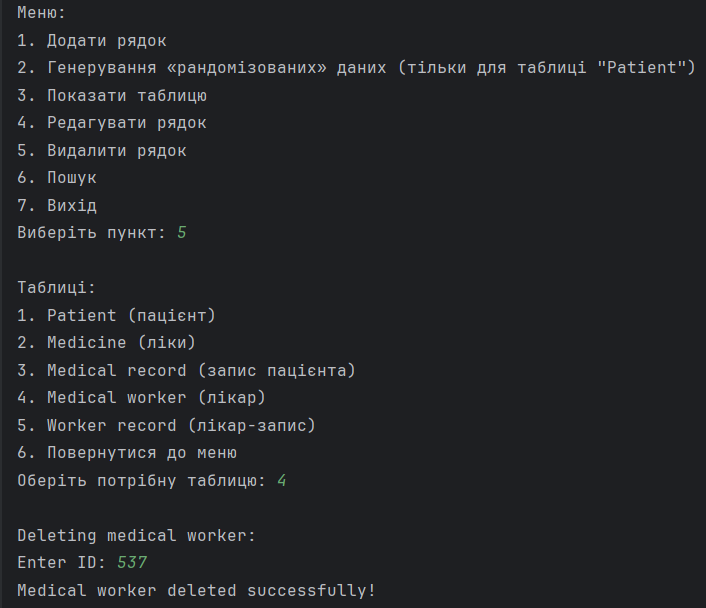
self.conn.commit()

**Видалення рядка**

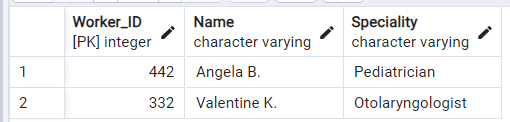
***Таблиця “Medical\_Worker” до:***

******

***Процес видалення рядка:***

******

***Таблиця “Medical\_Worker” після:***

******

***Лістинг delete:***

**def** delete\_patient(self, patient\_id):

c = self.conn.cursor()

c.execute('DELETE FROM "Patient" WHERE "Patient\_ID"=%s', (patient\_id,))

self.conn.commit()

**def** delete\_medicine(self, medicine\_id):

c = self.conn.cursor()

c.execute('DELETE FROM "Medicine" WHERE "Medicine\_ID"=%s', (medicine\_id,))

self.conn.commit()

**def** delete\_medical\_record(self, record\_id):

c = self.conn.cursor()

c.execute('DELETE FROM "Medical\_Record" WHERE "Record\_ID"=%s', (record\_id,))

self.conn.commit()

**def** delete\_medical\_worker(self, worker\_id):

c = self.conn.cursor()

c.execute('DELETE FROM "Medical\_Worker" WHERE "Worker\_ID"=%s', (worker\_id,))

self.conn.commit()

**def** delete\_worker\_record(self, tab\_id):

c = self.conn.cursor()

c.execute('DELETE FROM "Worker\_Record" WHERE "Tab\_ID"=%s', (tab\_id,))

self.conn.commit()

**Завдання 2**

**Генерування «рандомізованих» даних**

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

**WITH** max\_id **AS** (**SELECT** COALESCE(MAX("Patient\_ID"), 0) **FROM** public."Patient")

**INSERT** **INTO** public."Patient" ("Patient\_ID", "Name", "Birthday", "Address")

**SELECT** (**SELECT** \* **FROM** max\_id) + row\_number() OVER () **as** "Patient\_id",

chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int),

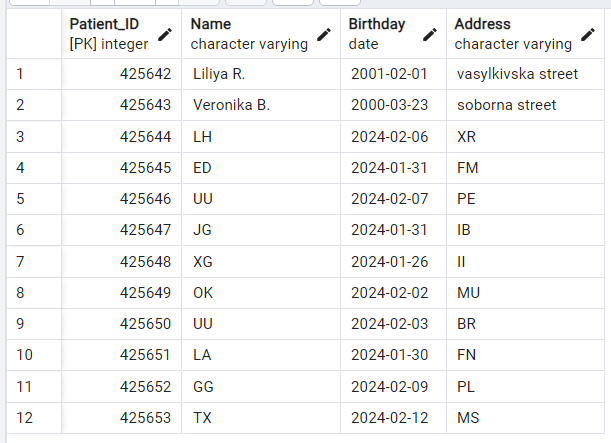
current\_date + (random() \* 30)::integer \* '1 day'::interval,

chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int)

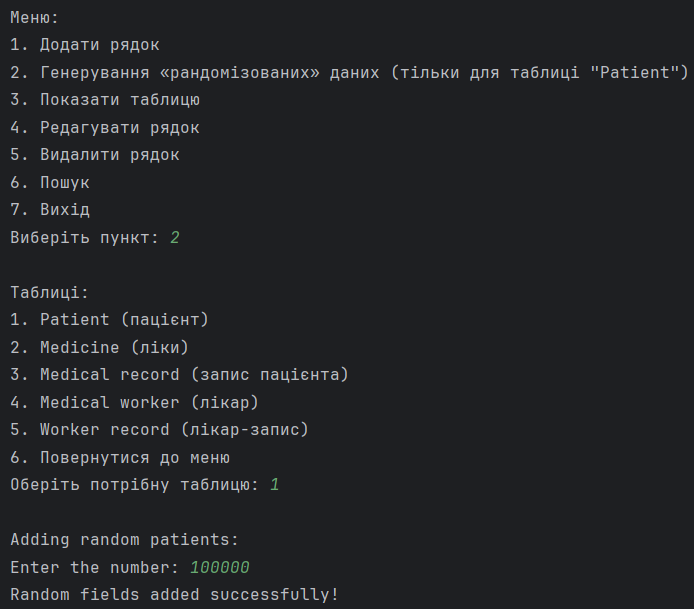
**FROM** generate\_series(1, %s);

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

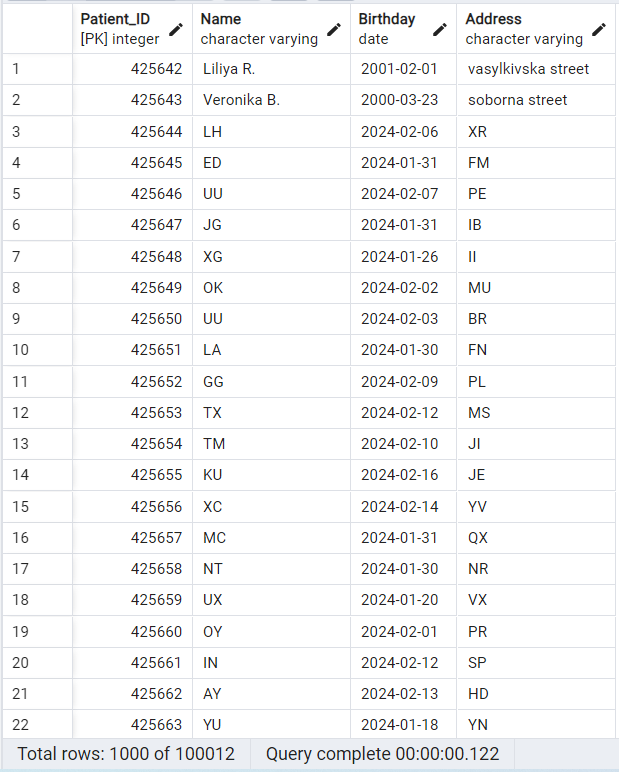
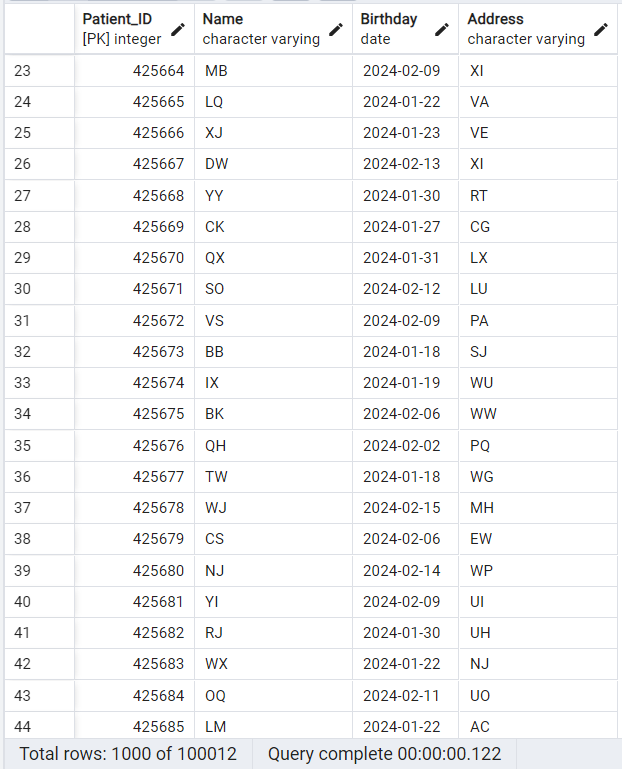
***Таблиця “Patients” до:***

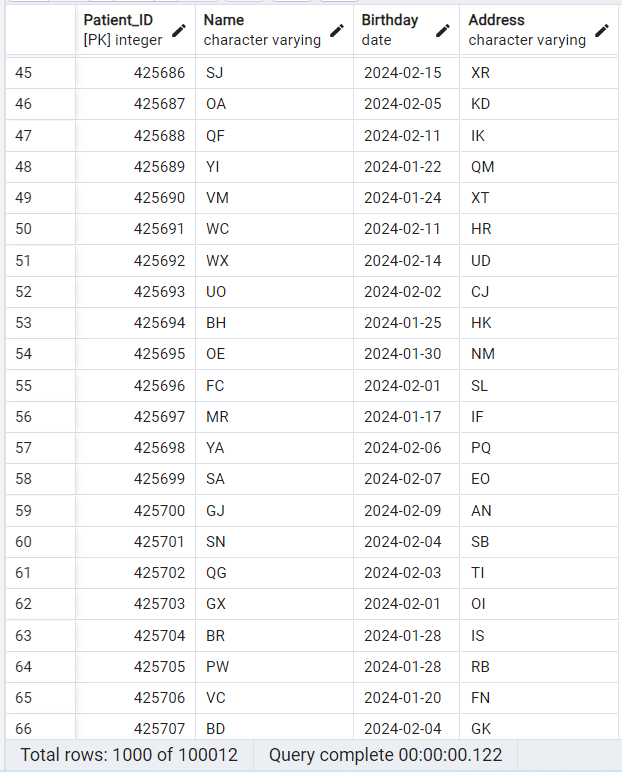
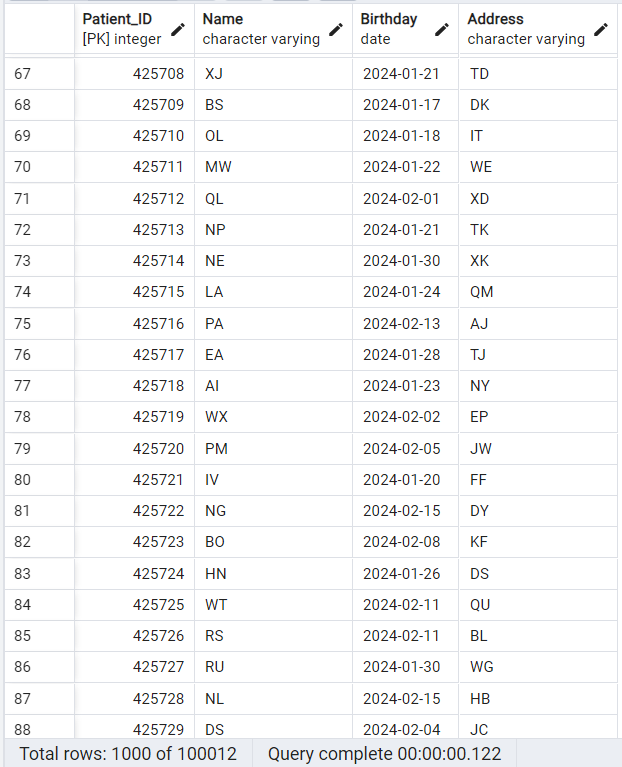
******

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

******

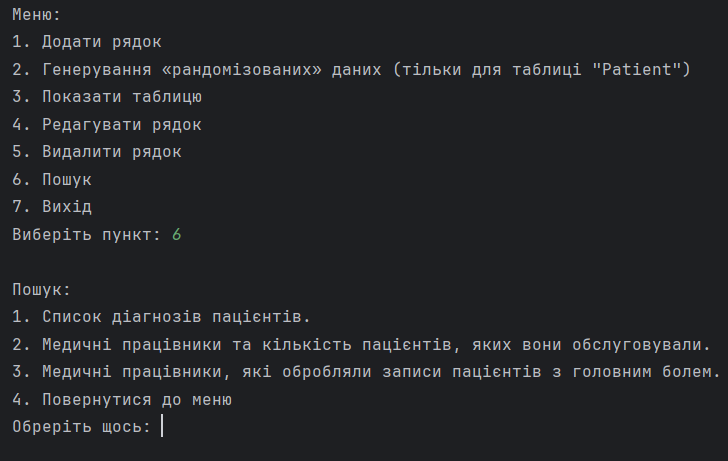
***Таблиця “Patients” після:***

**Завдання 3**

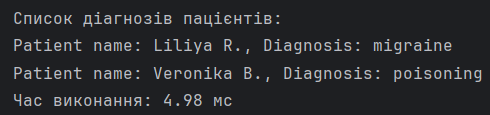
**Пошук даних**

****

***Запит №1***

Список діагнозів пацієнтів.

Цей запит створений для отримання списку із діагнозами пацієнтів. На виході отримуємо список пацієнтів, з їхніми іменами та власне діагнозом. Також було засічено час виконання цього запиту. Він становив 4.98 мс.



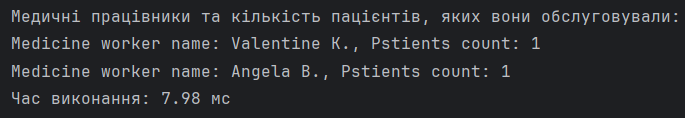
SQL запит виглядає наступним чином:

**SELECT** "Patient"."Name", "Medical\_Record"."Diagnosis" **FROM** "Patient" **JOIN** "Medical\_Record" **ON** "Patient"."Patient\_ID" = "Medical\_Record"."Patient\_ID";

***Запит №2***

Медичні працівники та кількість пацієнтів, яких вони обслуговували.

Цей запит створений для отримання кількості пацієнтів, яких обслуговували медичні працівники. На виході отримуємо список мед. працівників з їх іменами та числом прийнятих пацієнтів. Також було засічено час виконання цього запиту. Він становив 7.98 мс.



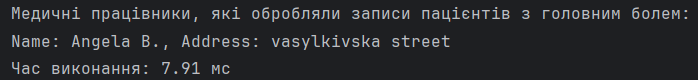
SQL запит виглядає наступним чином:

**SELECT** "Medical\_Worker"."Name", COUNT("Patient"."Patient\_ID") **AS** "Patient\_Count" **FROM** "Medical\_Worker" LEFT **JOIN** "Worker\_Record" **ON** "Medical\_Worker"."Worker\_ID" = "Worker\_Record"."Worker\_ID" LEFT **JOIN** "Medical\_Record" **ON** "Worker\_Record"."Record\_ID" = "Medical\_Record"."Record\_ID" LEFT **JOIN** "Patient" **ON** "Medical\_Record"."Patient\_ID" = "Patient"."Patient\_ID"**GROUP** **BY** "Medical\_Worker"."Name";

***Запит №3***

Медичні працівники, які обробляли записи пацієнтів з головним болем.

Цей запит створений для отримання медичних працівників, які приймали пацієнтів із головним болем. На виході отримуємо список мед. працівників з їх іменами адресами. Також було засічено час виконання цього запиту. Він становив 7.91 мс.



SQL запит виглядає наступним чином:

**SELECT** "Medical\_Worker"."Name", "Patient"."Address" **FROM** "Medical\_Worker" **JOIN** "Worker\_Record" **ON** "Medical\_Worker"."Worker\_ID" = "Worker\_Record"."Worker\_ID" **JOIN** "Medical\_Record" **ON** "Worker\_Record"."Record\_ID" = "Medical\_Record"."Record\_ID" **JOIN** "Patient" **ON** "Medical\_Record"."Patient\_ID" = "Patient"."Patient\_ID" **WHERE** "Medical\_Record"."Symptoms" = 'headache';

**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='patients',

user='postgres',

password='1111',

host='localhost',

port=3000

)

**def** add\_patient(self, patient\_id, name, birthday, address):

c = self.conn.cursor()

c.execute('INSERT INTO "Patients" ("Patient\_ID", "Name", "Birthday", "Address") VALUES (%s, %s, %s, %s)', (patient\_id, name, birthday, address))

self.conn.commit()

**def** add\_medicine(self, medicine\_id, patient\_id, name, dosage):

c = self.conn.cursor()

c.execute('INSERT INTO "Medicine" ("Medicine\_ID", "Patient\_ID", "Name", "Dosage") VALUES (%s, %s, %s, %s)', (medicine\_id, patient\_id, name, dosage))

self.conn.commit()

**def** add\_medical\_record(self, record\_id, patient\_id, date, symptoms, diagnosis):

c = self.conn.cursor()

c.execute('INSERT INTO "Medical\_Record" ("Record\_ID", "Patient\_ID", "Date", "Symptoms", "Diagnosis") VALUES (%s, %s, %s, %s, %s)', (record\_id, patient\_id, date, symptoms, diagnosis))

self.conn.commit()

**def** add\_medical\_worker(self, worker\_id, name, speciality):

c = self.conn.cursor()

c.execute('INSERT INTO "Medical\_Worker" ("Worker\_ID", "Name", "Speciality") VALUES (%s, %s, %s)', (worker\_id, name, speciality))

self.conn.commit()

**def** add\_worker\_record(self, tab\_id, worker\_id, record\_id):

c = self.conn.cursor()

c.execute('INSERT INTO "Worker\_Record" ("Tab\_ID", "Worker\_ID", "Record\_ID") VALUES (%s, %s, %s)', (tab\_id, worker\_id, record\_id))

self.conn.commit()

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

c = self.conn.cursor()

c.execute('SELECT \* FROM "Patient"')

**return** c.fetchall()

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

c = self.conn.cursor()

c.execute('SELECT \* FROM "Medicine"')

**return** c.fetchall()

**def** get\_all\_medical\_records(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM "Medical\_Record"')

**return** c.fetchall()

**def** get\_all\_medical\_workers(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM "Medical\_Worker"')

**return** c.fetchall()

**def** get\_all\_workers\_records(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM "Worker\_Record"')

**return** c.fetchall()

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

c = self.conn.cursor()

c.execute('UPDATE "Patient" SET "Name"=%s, "Birthday"=%s, "Address"=%s WHERE "Patient\_ID"=%s', (name, birthday, address, id))

self.conn.commit()

**def** update\_medicine(self, patient\_id, name, dosage, id):

c = self.conn.cursor()

c.execute('UPDATE "Medicine" SET "Patient\_ID"=%s, "Name"=%s, "Dosage"=%s WHERE "Medicine\_ID"=%s', (patient\_id, name, dosage, id))

self.conn.commit()

**def** update\_medical\_record(self, patient\_id, date, symptoms, diagnosis, id):

c = self.conn.cursor()

c.execute('UPDATE "Medical\_Record" SET "Patient\_ID"=%s, "Date"=%s, "Symptoms"=%s, "Diagnosis"=%s WHERE "Record\_ID"=%s', (patient\_id, date, symptoms, diagnosis, id))

self.conn.commit()

**def** update\_medical\_worker(self, name, speciality, id):

c = self.conn.cursor()

c.execute('UPDATE "Medical\_Worker" SET "Name"=%s, "Speciality"=%s WHERE "Worker\_ID"=%s', (name, speciality, id))

self.conn.commit()

**def** update\_worker\_record(self, worker\_id, record\_id, id):

c = self.conn.cursor()

c.execute('UPDATE "Worker\_Record" SET "Worker\_ID"=%s, "Record\_ID"=%s WHERE "Tab\_ID"=%s', (worker\_id, record\_id, id))

self.conn.commit()

**def** delete\_patient(self, patient\_id):

c = self.conn.cursor()

c.execute('DELETE FROM "Patient" WHERE "Patient\_ID"=%s', (patient\_id,))

self.conn.commit()

**def** delete\_medicine(self, medicine\_id):

c = self.conn.cursor()

c.execute('DELETE FROM "Medicine" WHERE "Medicine\_ID"=%s', (medicine\_id,))

self.conn.commit()

**def** delete\_medical\_record(self, record\_id):

c = self.conn.cursor()

c.execute('DELETE FROM "Medical\_Record" WHERE "Record\_ID"=%s', (record\_id,))

self.conn.commit()

**def** delete\_medical\_worker(self, worker\_id):

c = self.conn.cursor()

c.execute('DELETE FROM "Medical\_Worker" WHERE "Worker\_ID"=%s', (worker\_id,))

self.conn.commit()

**def** delete\_worker\_record(self, tab\_id):

c = self.conn.cursor()

c.execute('DELETE FROM "Worker\_Record" WHERE "Tab\_ID"=%s', (tab\_id,))

self.conn.commit()

**def** get\_patients\_diagnosis(self):

c = self.conn.cursor()

c.execute('SELECT "Patient"."Name", "Medical\_Record"."Diagnosis" FROM "Patient" JOIN "Medical\_Record" ON "Patient"."Patient\_ID" = "Medical\_Record"."Patient\_ID";')

**return** c.fetchall()

**def** get\_workers\_patients(self):

c = self.conn.cursor()

c.execute(

'SELECT "Medical\_Worker"."Name", COUNT("Patient"."Patient\_ID") AS "Patient\_Count" FROM "Medical\_Worker" LEFT JOIN "Worker\_Record" ON "Medical\_Worker"."Worker\_ID" = "Worker\_Record"."Worker\_ID" LEFT JOIN "Medical\_Record" ON "Worker\_Record"."Record\_ID" = "Medical\_Record"."Record\_ID" LEFT JOIN "Patient" ON "Medical\_Record"."Patient\_ID" = "Patient"."Patient\_ID"GROUP BY "Medical\_Worker"."Name";')

**return** c.fetchall()

**def** get\_patients\_with\_headache(self):

c = self.conn.cursor()

c.execute(

'''SELECT "Medical\_Worker"."Name", "Patient"."Address" FROM "Medical\_Worker" JOIN "Worker\_Record" ON "Medical\_Worker"."Worker\_ID" = "Worker\_Record"."Worker\_ID" JOIN "Medical\_Record" ON "Worker\_Record"."Record\_ID" = "Medical\_Record"."Record\_ID" JOIN "Patient" ON "Medical\_Record"."Patient\_ID" = "Patient"."Patient\_ID" WHERE "Medical\_Record"."Symptoms" = 'headache';''')

**return** c.fetchall()

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

c = self.conn.cursor()

c.execute('''

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

INSERT INTO public."Patient" ("Patient\_ID", "Name", "Birthday", "Address")

SELECT (SELECT \* FROM max\_id) + row\_number() OVER () as "Patient\_id",

chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int),

current\_date + (random() \* 30)::integer \* '1 day'::interval,

chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int)

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 patient:")

self.add\_patient()

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

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

self.add\_medicine()

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

self.view.show\_message("\nAdding medical record:")

self.add\_medical\_record()

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

self.view.show\_message("\nAdding medical worker:")

self.add\_medical\_worker()

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

self.view.show\_message("\nAdding worker record:")

self.add\_worker\_record()

**else**:

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

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

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

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

self.add\_random\_fields()

**else**:

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

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

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

self.view\_patients()

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

self.view\_medicines()

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

self.view\_medical\_records()

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

self.view\_medical\_workers()

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

self.view\_workers\_records()

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

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 patient:")

self.update\_patient()

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

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

self.update\_medicine()

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

self.view.show\_message("\nUpdating medical record:")

self.update\_medical\_record()

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

self.view.show\_message("\nUpdating medical worker:")

self.update\_medical\_worker()

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

self.view.show\_message("\nUpdating worker record:")

self.update\_worker\_record()

**else**:

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

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

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

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

self.delete\_patient()

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

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

self.delete\_medicine()

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

self.view.show\_message("\nDeleting medical record:")

self.delete\_medical\_record()

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

self.view.show\_message("\nDeleting medical worker:")

self.delete\_medical\_worker()

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

self.view.show\_message("\nDeleting worker record:")

self.delete\_worker\_record()

**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\_patients\_diagnosis()

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\_workers\_patients()

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\_patients\_with\_headache()

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\_patient(self):

**try**:

patient\_id, name, birthday, address = self.view.get\_patient\_input()

self.model.add\_patient(patient\_id, name, birthday, address)

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

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

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

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

**try**:

medicine\_id, patient\_id, name, dosage = self.view.get\_medicine\_input()

self.model.add\_medicine(medicine\_id, patient\_id, name, dosage)

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

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

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

**def** add\_medical\_record(self):

**try**:

record\_id, patient\_id, date, symptoms, diagnosis = self.view.get\_medical\_record\_input()

self.model.add\_medical\_record(record\_id, patient\_id, date, symptoms, diagnosis)

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

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

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

**def** add\_medical\_worker(self):

**try**:

worker\_id, name, speciality = self.view.get\_medical\_worker\_input()

self.model.add\_medical\_worker(worker\_id, name, speciality)

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

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

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

**def** add\_worker\_record(self):

**try**:

tab\_id, worker\_id, record\_id = self.view.get\_worker\_record\_input()

self.model.add\_worker\_record(tab\_id, worker\_id, record\_id)

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

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

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

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

**try**:

patients = self.model.get\_all\_patients()

self.view.show\_patients(patients)

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

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

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

**try**:

medicines = self.model.get\_all\_medicines()

self.view.show\_medicines(medicines)

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

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

**def** view\_medical\_records(self):

**try**:

medical\_records = self.model.get\_all\_medical\_records()

self.view.show\_medical\_records(medical\_records)

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

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

**def** view\_medical\_workers(self):

**try**:

medical\_workers = self.model.get\_all\_medical\_workers()

self.view.show\_medical\_workers(medical\_workers)

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

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

**def** view\_workers\_records(self):

**try**:

workers\_records = self.model.get\_all\_workers\_records()

self.view.show\_workers\_records(workers\_records)

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

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

**def** show\_workers\_patients(self):

**try**:

rows = self.model.get\_workers\_patients()

self.view.show\_workers\_patients(rows)

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

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

**def** show\_patients\_diagnosis(self):

**try**:

rows = self.model.get\_patients\_diagnosis()

self.view.show\_patients\_diagnosis(rows)

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

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

**def** show\_patients\_with\_headache(self):

**try**:

rows = self.model.get\_patients\_with\_headache()

self.view.show\_patients\_with\_headache(rows)

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

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

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

**try**:

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

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

self.model.update\_patient(name, birthday, address, patient\_id)

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

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

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

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

**try**:

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

patient\_id, name, dosage = self.view.get\_medicine\_input()

self.model.update\_medicine( patient\_id, name, dosage, medicine\_id)

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

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

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

**def** update\_medical\_record(self):

**try**:

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

patient\_id, date, symptoms, diagnosis = self.view.get\_medical\_record\_input()

self.model.update\_medical\_record(patient\_id, date, symptoms, diagnosis, record\_id)

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

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

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

**def** update\_medical\_worker(self):

**try**:

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

name, speciality = self.view.get\_medical\_worker\_input()

self.model.update\_medical\_worker(name, speciality, worker\_id)

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

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

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

**def** update\_worker\_record(self):

**try**:

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

worker\_id, record\_id = self.view.get\_worker\_record\_input()

self.model.update\_worker\_record(worker\_id, record\_id, tab\_id)

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

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

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

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

**try**:

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

self.model.delete\_patient(patient\_id)

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

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

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

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

**try**:

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

self.model.delete\_medicine(medicine\_id)

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

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

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

**def** delete\_medical\_record(self):

**try**:

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

self.model.delete\_medical\_record(record\_id)

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

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

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

**def** delete\_medical\_worker(self):

**try**:

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

self.model.delete\_medical\_worker(worker\_id)

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

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

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

**def** delete\_worker\_record(self):

**try**:

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

self.model.delete\_worker\_record(tab\_id)

self.view.show\_message("Worker record 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 "Patient")'***)***

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. Patient (паці�"нт)")

self.show\_message("2. Medicine (ліки)")

self.show\_message("3. Medical record (запис паці�"нта)"***)***

self.show\_message("4. Medical worker (лікар)"***)***

self.show\_message("5. Worker record (лікар-запис)"***)***

self.show\_message("6. Повернутися до меню"***)***

table = input("Оберіть потрібну таблицю: "***)***

return tabl***e***

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

self.show\_message("\nПошук:"***)***

self.show\_message("1. Список діагнозів паці�"нтів.")

self.show\_message("2. Медичні працівники та кількість паці�"нтів, яких вони обслуговували."***)***

self.show\_message("3. Медичні працівники, які обробляли записи паці�"нтів з головним болем.")

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

choice = input("Обреріть щось: ")

**return** choice

**def** show\_patients(self, patients):

**print**("\nPatients:")

**for** patient **in** patients:

**print**(f"ID: {patient[0]}, Name: {patient[1]}, Birthday: {patient[2]}, Address: {patient[3]}")

**def** show\_medicines(self, medicines):

**print**("\nMedicines:")

**for** medicine **in** medicines:

**print**(f"ID: {medicine[0]}, Patient ID: {medicine[1]}, Name: {medicine[2]}, Dosage: {medicine[3]}")

**def** show\_medical\_records(self, medical\_records):

**print**("\nMedical records:")

**for** medical\_record **in** medical\_records:

**print**(f"ID: {medical\_record[0]}, Patient ID: {medical\_record[1]}, Date: {medical\_record[2]}, Symptoms: {medical\_record[3]}, Diagnosis: {medical\_record[4]}")

**def** show\_medical\_workers(self, medical\_workers):

**print**("\nMedical workers:")

**for** medical\_worker **in** medical\_workers:

**print**(f"ID: {medical\_worker[0]}, Name: {medical\_worker[1]}, Speciality: {medical\_worker[2]}")

**def** show\_workers\_records(self, workers\_records):

**print**("\nWorkers records:")

**for** worker\_record **in** workers\_records:

**print**(f"ID: {worker\_record[0]}, Worker ID: {worker\_record[1]}, Record ID: {worker\_record[2]}")

**def** show\_patients\_diagnosis(self, rows):

**print**("\nСписок діагнозів паці�"нтів:"***)***

for row in rows***:***

print(f"Patient name: {row[0]}, Diagnosis: {row[1]}"***)***

def show\_workers\_patients(self, rows)***:***

print("\nМедичні працівники та кількість паці�"нтів, яких вони обслуговували:")

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

**print**(f"Medicine worker name: {row[0]}, Pstients count: {row[1]}")

**def** show\_patients\_with\_headache(self, rows):

**print**("\nМедичні працівники, які обробляли записи паці�"нтів з головним болем:"***)***

for row in rows***:***

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

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

while True***:***

try***:***

name = input("Enter patient name: "***)***

if name.strip()***:***

brea***k***

else***:***

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

except ValueError***:***

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

while True***:***

try***:***

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

birthday = datetime.strptime(date, "%Y-%m-%d").date(***)***

brea***k***

except ValueError***:***

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

while True***:***

try***:***

address = input("Enter patient address: "***)***

if address.strip()***:***

brea***k***

else***:***

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

except ValueError***:***

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

return name, birthday, addres***s***

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

while True***:***

try***:***

patient\_id = int(input("Enter patient ID: ")***)***

brea***k***

except ValueError***:***

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

while True***:***

try***:***

name = input("Enter medicine name: "***)***

if name.strip()***:***

brea***k***

else***:***

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

except ValueError***:***

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

while True***:***

try***:***

dosage = input("Enter medicine dosage: "***)***

if dosage.strip()***:***

brea***k***

else***:***

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

except ValueError***:***

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

return patient\_id, name, dosag***e***

def get\_medical\_record\_input(self)***:***

while True***:***

try***:***

patient\_id = int(input("Enter patient ID: ")***)***

brea***k***

except ValueError***:***

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

while True***:***

try***:***

date\_input = input("Enter date (YYYY-MM-DD): "***)***

date = datetime.strptime(date\_input, "%Y-%m-%d").date(***)***

brea***k***

except ValueError***:***

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

while True***:***

try***:***

symptoms = input("Enter medical record symptoms: "***)***

if symptoms.strip()***:***

brea***k***

else***:***

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

except ValueError***:***

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

while True***:***

try***:***

diagnosis = input("Enter medical record diagnosis: "***)***

if diagnosis.strip()***:***

brea***k***

else***:***

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

except ValueError***:***

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

return patient\_id, date, symptoms, diagnosi***s***

def get\_medical\_worker\_input(self)***:***

while True***:***

try***:***

name = input("Enter medical worker name: "***)***

if name.strip()***:***

brea***k***

else***:***

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

except ValueError***:***

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

while True***:***

try***:***

speciality = input("Enter medical worker speciality: "***)***

if speciality.strip()***:***

brea***k***

else***:***

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

except ValueError***:***

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

return name, specialit***y***

def get\_worker\_record\_input(self)***:***

while True***:***

try***:***

worker\_id = int(input("Enter worker ID: ")***)***

brea***k***

except ValueError***:***

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

while True***:***

try***:***

record\_id = int(input("Enter record ID: ")***)***

brea***k***

except ValueError***:***

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

return worker\_id, record\_i***d***

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

while True***:***

try***:***

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

brea***k***

except ValueError***:***

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

return i***d***

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

print(message***)***

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

while True***:***

try***:***

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

brea***k***

except ValueError***:***

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

return numbe***r***